Modality in Dialogue: Planning, Pragmatics and Computation

Matthew Stone

September 1998

Modality in Dialogue: Planning, Pragmatics and Computation

Matthew Stone

September 1998

Abstract

Natural language generation (NLG) is first and foremost a reasoning task. In this reasoning, a system plans a communicative act that will signal key facts about the domain to the hearer. In generating action descriptions, this reasoning draws on characterizations both of the causal properties of the domain and the states of knowledge of the participants in the conversation. This dissertation shows how such characterizations can be specified declaratively and accessed efficiently in NLG.

The heart of this dissertation is a study of logical statements about knowledge and action in modal logic. By investigating the proof-theory of modal logic from a logic programming point of view, I show how many kinds of modal statements can be seen as straightforward instructions for computationally manageable search, just as Prolog clauses can. These modal statements provide sufficient expressive resources for an NLG system to represent the effects of actions in the world or to model an addressee whose knowledge in some respects exceeds and in other respects falls short of its own. To illustrate the use of such statements, I describe how the SPUD sentence planner exploits a modal knowledge base to assess the interpretation of a sentence as it is constructed incrementally.

Contents

Acknowledgments

As befits its protracted and formative creation, this work deeply reflects not only its author but also the environment which allowed it at last to flourish. From its defining moments, this research was stimulated and refined by Mark Steedman and his students—Beryl Hoffman, Nobo Komagata, Michael Niv, Charlie Ortiz, Scott Prevost and Mike White—whose shared interest in knowledge representation and natural language generation made for an intellectual environment at once supportive and challenging. As it developed, the research enjoyed the infectious enthusiasm of close collaborators, particularly on projects related to SPUD, TAG and generation—first Christy Doran, and later including Daniel Hardt, Aravind Joshi, Martha Palmer, Vijay Shankar, Bonnie Webber, Tonia Bleam, Julie Bourne and Gann Bierner. I have sought, and received, the sage attention of a range of experts on specialized questions—Justine Cassell, Robin Clark, Dale Miller, Leora Morgenstern, Ellen Prince and Tandy Warnow. And now, even the final assembly of this document has been honed by the close readings and comments provided by Mark Steedman, Aravind Joshi, Rich Thomason, Bonnie Webber and Scott Weinstein. Add the broader camaraderie and intellectual exchange among students in CIS and across IRCS and even Penn—those involved know who they are—and you may, like me, also suspect that nothing like this dissertation would have been possible anywhere else in the world.

So I'm thankful to have gotten to Penn: for that I owe Pauline Jacobson, whose cultivation of my semantic interests proved a definitive pointer in this direction—and, of course, my first and foremost academic influences, Mom and Dad, who taught me how to take ideas seriously and precisely, and how to write. And I'm thankful to have been able to stay: for the financial support, thanks to NSF and IRCS graduate fellowships, and a number of other supporting grants, including NSF grant IRI95-04372, ARPA grant N66001-94-C6043, and ARO grant DAAH04-94-G0426; for the emotional support (no less necessary), thanks to Doug DeCarlo.

1 Introduction

Natural language generation (NLG) promises to provide an exciting technology for improving the way computer systems communicate their results to users. Using NLG, systems can customize the information they present to user and context. The Migraine system [Carenini *et al.*, 1994], for example, takes the user's history and diagnosis into account to provide medical information that more precisely matches the user's needs. The ILEX system [Mellish *et al.*, 1998], meanwhile, exploits the opportunities raised by previous dialogue to pack important and interesting information into an interactive, on-line museum tour. Using NLG, computer expertise can also be deployed in new settings, particularly using speech. For instance, the TraumAid system [Gertner and Webber, 1998] issues concise natural language critiques of treatment plans in the emergency room during the initial management of chest trauma. The Computer fix-it shop system [Biermann *et al.*, 1993] uses voice dialogue over phone lines to give a computer technician in the field convenient remote access to an expert fault-diagnosis system.

When NLG succeeds in systems such as these, two factors are at work. On the one hand, the system must have substantive and correct knowledge about its domain, so that users could benefit from the information provided by the system. On the other, the system must communicate that knowledge in a concise and natural form, so that users can understand the information easily, without being distracted from their other tasks and concerns. These two requirements apparently conflict. The more detailed and rich are the system's representations of the domain, the more interesting and valuable NLG would be—but the further removed those representations are in content and organization from natural linguistic messages.

Because of this gap, NLG is first and foremost a reasoning task. In this reasoning, a system plans a communicative act that will signal key facts about the domain to the hearer. This reasoning thus draws on characterizations both of the causal properties of the domain and the states of knowledge of the participants in the conversation. This dissertation shows how such characterizations can be specified declaratively and accessed efficiently in generating NL action descriptions.

The heart of this dissertation is a study of logical statements about knowledge and action in modal logic. Modal logic has been a standard tool for describing knowledge and action for decades (see [Fagin *et al.*, 1995] and references there), but reasoning with these descriptions has suffered from the unique difficulties of proving modal theorems (see

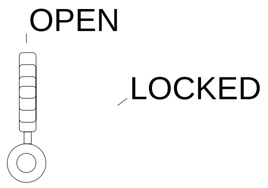
[Wallen, 1990; Ohlbach, 1991]). By investigating the proof-theory of modal logic from a logic programming point of view, I show how to eliminate or avoid these problems for the kinds of modal statements we typically need in writing useful specifications of action and knowledge in computer systems. Thanks to these results, many kinds of modal statements can be seen as simple and straightforward instructions for search, just as Prolog clauses can. Moreover, these results show that, while problems of reasoning about causal interactions in the domain may be genuinely difficult, the overlay of reasoning about knowledge needed in NLG can be accommodated with relatively little extra effort.

In the remainder of this introduction, I first motivate in more detail the problem of reasoning in NLG for action descriptions, in section 1.1. Then I outline more fully the results and contributions of the dissertation in section 1.2.

1.1 Reasoning in NLG

The domain action representation which an NLG module is given as input must support a wide range of reasoning tasks elsewhere in a larger computer system. Because of these tasks, the representations will likely exhibit a complexity and structure which is, by and large, independent of any linguistic motivations. As I explain in this section, the discrepancy accounts for the reasoning processes required in NLG.[1]

The NLG module will not be provided with the inputs one might first expect, taking only linguistic theory into account. Inspired by linguistic semantics, for example, one might view the domain representation underlying an instruction as a relatively simple, high-level action description, built hierarchically from an inventory of primitive concepts. Then NLG would just require a map from underlying concepts and structure into overt words and surface structure, as studied formally in theoretical work on semantic head-driven generation [Shieber *et al.*, 1990], and as implemented (albeit in restricted form) as a module in most practical NLG systems [Reiter, 1994]. As a case study, consider (1).

(1) a 

b Turn handle to locked position.

c TURN(HANDLE, LOCKED)

d (((IMPERATIVE (PRESENT TURN))
  (THE HANDLE))
 (TO (THE (LOCKED POSITION))))

On the simplest realization of this view, the action in (1b) would be represented underlyingly as schematized in (1c). (For proposals postulating such underlying simplicity, see [Schank

[1]Nigel Ward motivates rich input and detailed reasoning in NLG by a complementary argument, based on the subtle functional and grammatical decisions that the generator must make [Ward, 1994].

and Abelson, 1977; Jackendoff, 1990].) But even the more subtle articulation of the semantics of (1b) as in (1d)—interpreted in a sophisticated ontology like Dynamic Montague Grammar [Groenendijk and Stokhof, 1990a]—reflects the same linguistic characterization of inputs to generation.

Unfortunately, the kind of action representation illustrated in (1c) or (1d) is unlikely to be available in an interesting, large system. The domain reasoner is likely to represent what happens in action with characterizations that are much more detailed than these formulas suggest. At the same time, the domain reasoner is likely to omit features of these formulas that specify how actions depend on the shared state of the environment and the shared goals of the conversationalists. I describe these two differences in turn.

1.1.1 Reasoning to omit detail

If the system is to be a reliable and interesting giver of advice, it will need a precise characterization of the actions it instructs. How else will it be able to precisely assess the consequences of its instructions, or clarify the instructions to the user after followup questions? To take just one example, one might expect the system to be able to display a computer animation of a human carrying out the intended action, to help indicate what action was expected (or perhaps to confirm that the action was possible). This problem, which has been studied for many years at Penn [Webber *et al.*, 1995; Webber, 1998], draws attention to the many details that the representations in (1) suppress.

In executing this instruction, the figure first approaches the handle, and positions a hand to grasp the handle and then draw it through a smooth, continuous motion. Most likely, the figure grabs with the right hand, palm out and thumb down, so as to curl the fingers around the back of the handle. Then, moving from the shoulder with elbow high, the figure will rotate the handle smoothly around its pivot, until the handle settles or stops at its locked position—as judged by varying exertion against the rotary friction and the visible orientation of the handle with respect to its position marker.

Obviously, when it comes to the precise characterization of actions such as these, an animated agent will require an action specification with many more parameters than are found in a term like (1c). The specification must settle features like the kind and placement of grasps and the future intentions that constrain the position and motion of the agent, as well as the beginning and endpoints of the action, any object affected, and so forth. The identification of such features is a long and ongoing research effort [Webber *et al.*, 1995; Douville *et al.*, 1996; Levison, 1996], which involves establishing not only what parameters are needed but what sources of information—general knowledge about agents, actions and objects; sensed properties of objects before the action; and feedback obtained during action execution—are required to determine appropriate values for those parameters.

These many parameters will not show up in a natural and concise instruction, any more than they will show up in high-level concepts like (1c). Low-level details will frequently be omitted. But it would be a mistake to look for a single cut that determined once-and-for-all which features of actions are too low-level ever to appear in NL instructions and which features of actions are sufficiently high-level to belong in any NL instruction. People's instructions exhibit great variability in what they make explicit.

For example, the maintenance manuals for the fuel system on the F-16 represent a highly-edited corpus of very stylized instructions. The manuals are rigidly structured, separately identifying needed equipment and personnel, presenting actions to carry out, describing favorable results and warning about unfavorable ones. Each operation is described extremely precisely; in the military, these instructions count as orders and cannot be disobeyed. Instructions in this corpus may explicitly specify at least five parameters for motion actions; the instructions in (2) illustrate how different features of an action are described explicitly in different cases.

(2) a Lift assembly at hinge.
b Disconnect cable from receptacle.
c Rotate assembly downward.
d Slide sleeve onto tube.

In all we find the object moved; (2a) identifies the point of contact with the object (where force may be applied); (2b) identifies the beginning configuration of the object; (2c) describes the intermediate path along which the object is moved; and (2d) describes the final configuration of the object. Despite the precision and editing of the instructions, most instructions omit most of these features. The examples in (2) are representative in signaling just the object affected and one other parameter of the action. Other variants occur as well—note in particular that even the object moved need not be explicitly provided as a parameter, as in (3).

(3) Push in on poppet.

The variability in these natural instructions suggests that any of the parameters that the hearer must derive to perform the action successfully—any part of the low-level record of an action—might potentially surface as an explicit component of an NL instruction. If many parameters are usually omitted, it is because the right value is so often obvious in context. In such cases, by starting from the explicit description of objects, places and actions provided in the instruction, and using shared knowledge about the possibilities for action in the domain and about the effects that must be achieved, the hearer can (without much effort) infer a full suite of appropriate parameters for action.

This analysis makes clear the heavy demands of inference that NLG requires. In order to both provide an adequate level of detail and leave out the obvious, the generator must assess what information the hearer would naturally recover from that description. This assessment determines whether a given description suffices, or whether more detail is required. These assessments combine reasoning about the causal relations between actions and effects in the domain and about the sources of information that the hearer can use to identify actions.

1.1.2 Reasoning to include detail

We have just presented one way the detail in the input to an NLG module exceeds the detail that is desired in the output, and argued that this introduces a burden of reasoning on the NLG module. In other respects, however, the action specification in the input may have to be enriched during the course of NLG. One important source of elaboration of the input action description comes in choosing a manner of identifying the needed action to the hearer that clearly links the action to the shared features of the environment. This places a different burden of reasoning on the NLG module.

The F-16 corpus again provides a good example of this process. In the vent system, pipes are sealed together using a sleeve, which fits snugly over the ends of adjacent pipes, and a coupling, which snaps shut around the sleeve and holds the sleeve in place. At the start of maintenance, one *removes* the coupling and *slides* the sleeve away from the junction between the pipes. After maintenance, one *(re-)positions* the sleeve at the junction and *(re-)installs* the coupling in place around it. In the F-16 corpus, these actions are always described using these verbs.

These choice of verb reflect not only the structure of the particular motion performed but also general features of the design and function of the equipment. To see this, consider the movement of the sleeve from the point of view of animation. The motion involved in sliding the sleeve away is just the reverse of the motion involved in positioning the sleeve back. Since the verb *slide* indicates smooth motion along a surface but not direction, *slide* seems to describe both actions equally well. The verb *position*, meanwhile, is used to describe a motion that leaves its object in some definite location, where the object can perform its usual intended function. In the case of the sleeve, it is *in position* when straddling the pipes whose junction it seals. Thus, when the manual has *slide* for one motion but *position* for the other, the difference depends not on the properties of the motion as it might be animated, but on what functions objects serve in which positions, and what paths keep objects in contact with one another. Thus, selecting one description over another in NLG may require combining details from the explicit record of the action to be performed with background knowledge (most likely as shared by speaker and hearer). Of course, this is true of descriptions that identify objects and places, as well as those that identify actions. Accessing this background is another task for reasoning about knowledge and action in NLG.

1.1.3 Action descriptions in NLG and conversational agents

As outlined in this section, an NLG module faces the following problem in constructing a description of an action in an instruction. The NLG module is given a detailed description of the action to perform (as computed elsewhere in the system), and is also provided with a description of the background knowledge about the domain that the hearer shares with the speaker (or has in private), and a characterization of the hearer's ability to fill in details about a described action by inference from such knowledge. Using the grammar, the NLG module constructs a sufficient description of the action to allow the hearer to determine what to do. This process of construction reorganizes the content input to the NLG module, by suppressing some features of the action description (as obvious) and by factoring in some features of the shared knowledge (to link the description to the context), so as to obtain a customized, concise and precise indication of what action the system expects of the user.

The ability to carry out this reorganization depends on reasoning about action and knowledge. This inferential dependence is what this dissertation is all about. Examining naturally-occurring instructions, such as those we have just seen for the F-16, puts the need for such reasoning into relief. So does examining the instructions that can be constructed automatically in a system that does not attempt it.

It was work on this latter kind of system that provided the original motivation for this dissertation research. The system consisted of a pair of conversational agents that interacted in a simulated environment by exchanging symbolic messages [Cassell *et al.*, 1994a; Cassell *et al.*, 1994b]. The messages surfaced as natural language instructions, combined with appropriate facial expressions, intonation and gestures. The realization of the resulting messages offered a simple setting to evaluate a model of how people select words and nonverbal displays in conversation and how they synchronize them together.

The agents we developed were able to cooperate to achieve tasks that neither could perform independently, by using messages for asking or answering questions, for proposing goals or actions, or for reporting on the execution of agreed actions. However, in the absence of an NLG module based on reasoning about action and knowledge, these messages amounted to direct readouts of the agents' internal states. As a result, the agents' actual dialogues were so filled with obviously inferrable statements that they bordered on the absurd. It was clear that to attempt further exploration and testing of models of conversation in this framework would only be possible under a richer and more faithful model of action description in NLG.

1.2 The Results of the Dissertation

This dissertation describes results about reasoning in NLG in three parts. Part I presents new techniques for proof-search in modal logic. Part II introduces new ways to bring these techniques to bear in the course of constructing modal specifications of knowledge and action. Part III discusses how the SPUD system incorporates and exploits these results in generating natural language action descriptions.

1.2.1 Modal deduction

Part I begins with an introduction to modal logic, in Chapter 2. The aim of this introduction is to show why modal logic offers a promising framework not only for expressing the knowledge about time, agents and problem-solving we would need for good NLG, but also for drawing the right conclusions from this knowledge in practical applications. In this introduction to modal reasoning, a particular focus is the use of specialized sequent calculi to describe strategies for proof-search. Informally, a sequent is a formal, syntactic record of a particular state in the construction of a proof. Sequent calculi describe how these states are rewritten by a proof-search engine in the process of deriving a proof. Viewing sequent calculi as descriptions of search algorithms, it is natural to associate sequents with explicit representations of the data structures used in implementations to match logical terms or guide search through logical formulas. Such calculi provide a formal framework for analyzing and optimizing proof-search algorithms.

Chapter 3 applies this framework to an important special case: proof-search strategies that mirror the logic programming search implemented in languages like Prolog. With this view of logic programming [Miller *et al.*, 1991; Andreoli, 1992], each sequent contains a distinguished formula that indicates the goal that the theorem-prover is currently working on. Rules of a logic-programming sequent calculus derive new theorem-proving states by decomposing this goal whenever possible. This allows logical connectives in the goal formula to be viewed as explicit instructions for search. When the goal is atomic and cannot be broken down further, sequent calculus rules describe a backward-chaining process that first selects a given formula to match against the goal and then breaks that formula down so as to reduce the atomic goal to an appropriate series of new theorem-proving tasks. This process continues (nondeterministically) until all tasks have been completed (or until no further alternatives remain).

The contribution of Chapter 3 is to derive and analyze a sequent calculus of this kind that describes inference in modal logic. The advantage of modal logic is that it allows the independence of search problems to be explicitly specified. A modal specification can create a modular task, which allows local assumptions to made just for the duration of the task and which permits only limited information to be taken into account. This notion of modularity is presaged by previous work on logic programming in expressive languages [Miller, 1989; Giordano and Martelli, 1994; Baldoni *et al.*, 1993], as well as computational proposals for building separable specifications using contexts [Guha, 1991; McCarthy and Buvač, 1994]. Chapter 3 extends these lines of work by allowing for indefinite information to be specified using existential quantifiers and disjunctions in programs in a modular way. For example, an ordinary disjunction creates a global ambiguity in a proof, because analysis of alternative cases could potentially impact every inference in the proof. A modular disjunction, in contrast, creates a local ambiguity where case analysis can only impact inferences in the same module, and which may therefore be dispatched more tractably. This notion of modularity gives a powerful technique for understanding the search problems associated with particular modal specifications, or for designing specifications (in modal languages) with the complexity of search in mind. As subsequent chapters underscore, it is particularly suited to the ubiquitous task faced by NLG systems in reasoning about possible (mis-) interpretations of an utterance using a characterization of the hearer based on indefinite information.

Specialized sequent-calculi can be used not only to characterize an overall strategy for proof search but also to analyze the complexity of individual steps during proof search. This technique is illustrated by Chapter 4. At each backchaining step, the logic programming interpreter described in Chapter 3 must determine in what ways the program clause can be applied toward the modular task (or in the possible world) where the goal must be proved. In prior research, this determination has required an expensive operation of equational unification [Wallen, 1990; Auffray and Enjalbert, 1992; Otten and Kreitz, 1996]. Chapter 4 uses the framework of a specialized sequent calculus to show that a constraint algorithm can be used to resolve these matches between program statements and goals, incrementally and in polynomial time. This helps establish the practical benefit of the logic programming language described in Chapter 3. Modal logic programs can impose a modular structure on search without simultaneously introducing combinatorial problems which might offset the advantages of modularity.

Chapter 5 completes the logical study with an empirical evaluation of the techniques introduced earlier. It looks at deciding the validity of a sentence of the minimal propositional calculus (MPC), a particular problem in modal deduction which has been widely studied using a variety of theorem-proving techniques. The results of Chapter 5 highlight the variability in performance that different search strategies exhibit under variability in the structure of formulas analyzed or the structure of needed proofs. MPC suggests several search strategies and a plethora of specialized representations (not always compatible) for avoiding poor performance for particular kinds of search problems. As collected and analyzed in Chapter 5, run-times for deductive problems on different search strategies reveal that many, but not all, MPC search problems demand the constraint treatment of possible worlds developed in Chapter 4. The exceptions can be characterized in an interesting way based on semantic properties of MPC and on the compatibilities between various representational refinements and search strategies.

1.2.2 Modal representations for action

Part II considers the applications of the techniques developed in Part I to the problems of reasoning about knowledge and action that are particularly important in interpreting natural language descriptions. I begin in Chapter 6 by considering how modal logic can be used to reason about the temporal effects of actions. Reasoning about change is plagued by the default character of inertia: the effects of an action persist only as long as no further actions interfere with them. Chapter 6 proposes to use modal deduction, as investigated in Part I, to characterize putative consequences of actions. The question of whether putative consequences actually obtain is settled separately, by a process of argumentation that arbitrates among these competing hypotheses. The overall system contributes a synthesis of theoretical and practical proposals for reasoning about action. The mathematical approach to deduction allows the system to be related to other formal approaches to action based on logic programming [Gelfond and Lifschitz, 1993; Gelfond, 1994; Baral and Gelfond, 1993]. At the same time, the proof-theoretic perspective shows how consequences can be deduced in this framework by an intuitive algorithm. Its data structures—including temporal constraint algorithm based on the results of Chapter 4—mirror those used in special-purpose planners [McAllister and Rosenblitt, 1991; Penberthy and Weld, 1992]; but its algorithms—including methods to perform case analysis for explanatory reasoning based on the results of Chapter 3—extend them.

Reasoning in NLG requires characterizing the choices an agent can make as well as the actions that could occur in the domain: the action described in an instruction must be one the agent could choose to do. Chapter 7 examines how the arguments introduced in Chapter 6 can be extended to capture this notion of choice. The key is to explicitly represent the state of an agent's knowledge as well as the state of the world. With this representation, whenever an agent can make an appropriate choice, there is some concrete action which the agent knows will achieve the desired effects (cf. [Hintikka, 1971]). More generally, an agent can carry out any plan as long as such a feasible choice can be made, and then the selected action can be performed, at every step in the plan. Reasoning with specifications of these choices, naturally formulated by combining modal logic and existential quantification, represents another application of the study of indefinite information in Chapter 3. Thus, here as in Chapter 6, our analysis uses modal proof-theory to reconcile theoretical approaches to reasoning about actions and plans [Moore, 1985a; Morgenstern, 1987; Davis, 1994] with practical algorithms for building plans and reasoning about actions when an agent has changing access to information [Etzioni *et al.*, 1992; Golden and Weld, 1996; Goldman and Boddy, 1996].

This definition of choice is complemented in Chapter 8 by an analysis of how a specification can describe the information on which an agent will base such choices. Chapter 8 proposes constructing such specifications by dividing up agents' knowledge into sources or kinds of information. This strategy echoes Clark and Marshall's psycholinguistic proposal for organizing information by sources or kinds [Clark and Marshall, 1981]. By exploiting the deductive results of Part I—particularly its approach to logical modularity—Chapter 8 shows that modal logic, though a simple formalism with appealing search properties, can nevertheless capture the range of characterizations of sources of information that Clark and Marshall consider.

1.2.3 Modal inference and modal specifications in NLG

Part III consists of Chapter 9, a practical demonstration of how these results lead to a feasible approach to reasoning in NLG. Chapter 9 briefly introduces the SPUD system for constructing sentences [Stone and Doran, 1996; Stone and Doran, 1997; Stone and Webber, 1998]. SPUD uses a grammar in which alternative choices for generation are linked directly to lexical items (using the LTAG formalism [Schabes, 1990]). It builds a sentence by repeatedly making choices to add particular lexical items to the ongoing sentence. SPUD's choices of words must be guided by inferences about the domain and about the knowledge of the hearer, as outlined in section 1.1. Chapter 9 shows how the answers SPUD needs can be characterized as modal logic queries using the techniques of Part II, and how those answers can be found with logic programming using the techniques presented in Part I. Chapter 9 goes on to illustrate sample modal specifications given to SPUD in sample domains, shows the queries SPUD makes in describing particular actions in those domains, and shows how the results are used to guide SPUD in the generation of concise, natural descriptions of actions.

The results of the dissertation are summarized in Chapter 10; the dissertation concludes with a discussion of issues and problems for future research.

1.3 Advice to Readers

To present a convincing study of a formalism—whether for logical deduction, for representing and reasoning about causality, or for the interpretation of natural language sentences in discourse—inevitably requires a substantial amount of technical detail. In collecting together the material for this dissertation, I have aimed to include this kind of detail in each part. This means that readers whose interest centers on particular problems will find more here than they need. In general, those readers can proceed directly to the chapters that interest them. Readers with specialized interests may nevertheless want a high-level view of how the different results fit together. At this point, I briefly suggest some abbreviated paths through the document for such readers, according to their main research interest.

For NLG researchers, one path would be to simply skip ahead to Chapter 9 and read it on its own. A deeper reading would trace the informal motivation and development of the logic programming language DIALUP through sections 2.1, 3.1, 3.2, and 5.2. Then it would briefly touch on reasoning about action as a problem for logic and language through sections 6.1, 6.5 and 7.1. This background for Chapter 9 gives a flavor for the logical and computational results that NLG can exploit, but still omits the more technical material. Afterwards in reading Chapter 9, it will be clear what earlier results the work relies on and why.

Researchers in modal logic, after going through Part I, may be content to merely sample the results of Parts II and III to see how modal logic is used there. Sections 6.3, 7.1, 7.2, 8.4 and 9.2 describe how modal models and modal proof figure in these results.

Finally, researchers in cognitive robotics might want not only to look at Part II, but also to extract from Part I a high-level view of modular search in languages for describing planning and action. This suggests tracing through sections 2.1, some of 2.3 (e.g., 2.3.3 and 2.3.4), then 3.1, 3.2, and 3.3.

Part I

Modal Deduction

2 Modal Logic as a Modular Language

Modal logic and its proof theory provide basic resources for this study. This chapter provides a brief introduction to these subjects for reference in later chapters. Standard texts on modal logic include [Chellas, 1980; Hughes and Cresswell, 1968]; [Fagin *et al.*, 1995] provides a thorough introduction to modal representations in artificial intelligence. This chapter does not duplicate such surveys. Instead, it reviews the computational treatment of modal languages and modal proofs. Of central concern is the MODULARITY of modal logic—by which modal formulas describe not only what needs to be derived in a proof, but also how derivations should be broken down into parts and what information should be taken into account in each.

Modularity has both a denotational and an operational force. Denotationally, modularity underlies the different interpretations that modal operators can take on. Section 2.1 describes informally how the ability to distinguish statements into separate groups in modal logic enables descriptions of the changing states of the world in modal temporal logic, descriptions of the various possibilities compatible with the knowledge of agents in modal epistemic logic, and even the description of the subprograms of a logic program or the different contexts to which a knowledge base applies. Section 2.2 provides a more technical perspective on the same material, including the syntax and semantics of modal logic [Kripke, 1963].

Operationally, modularity can translate into constraints on search—under favorable circumstances. There are a variety of syntactic characterizations of proofs in modal logic; in principle, any can be used to derive modal theorems automatically. In practice, however, the choice of proof system can have a profound impact not only on the overall difficulty of automatic proof construction, but on the ability to exploit modularity as a constraint in proof search. Section 2.3 describes several proof systems for modal logic, exploring this variation at a theoretical level. The last decade has seen key developments in such systems, with the result that modal deduction can now be shown to satisfy the advantageous metatheory of classical logic in many respects, so that research and experience with classical deduction now transfers to modal logic [Fitting, 1972; Smullyan, 1973; Wallen, 1990; Ohlbach, 1991]. These results are important and deserve to be more widely known, but claims for their significance—“modal logic is now as a result just as tractable from a deductive point of view as is ordinary first-order logic” [Bibel, 1993], p.167—have sometimes been misleading.

Experience with classical deduction is crucial to designing proof procedures with good performance for modal logic, but this does not mean that henceforth modal theorem-proving will rely only on classical results. As this and subsequent chapters argue, modal deduction benefits from keeping the syntax and modularity of modal logic explicit and available during proof search. Accordingly, the principal objective of section 2.3 is to lay the groundwork for a Gentzen-style sequent calculus proof system that supports the syntax and modularity of modal logic as well as the algorithms and data structures of classical theorem-proving. The reader may consult [Mints, 1992] for a more thorough introduction to modern modal proof theory and [Gallier, 1986] for an introduction to the connections between proof theory and automated deduction. [Stone, to appear] presents a more explicit presentation of this material (including detailed proofs) for the special case of intuitionistic logic, a fragment of S4 modal logic with particular computational interest.

2.1 An Informal Survey of Modularity in Modal Logic

Modal logic extends the syntax of first-order logic by introducing propositional operators of NECESSITY and POSSIBILITY. Examples (4)–(7) illustrate the range of concepts which these operators fruitfully describe.

Chapter 3 investigates the use of necessity operators to describe the modular structure of logic programs; this use interprets necessity operators as in (4).

(4) [M]p: p is (to be) derived in the M module of the active logic program

In Chapter 6, meanwhile, we use the necessity operators described in (5) to represent and reason about change over time.

(5) a [N]p: p is true after the next action is taken
b [H]p: p holds now and will continue to hold until further notice

In Chapters 7 and 8, we use necessity operators schematized in (6) to describe the overall knowledge of an agent or components of that knowledge.

(6) a [K]p: the agent knows p
b [TOPIC]p: p follows from the part of the agent's knowledge that provides a theory of TOPIC.

Chapter 9 describes reasoning in discourse in modal terms, using the interpretations of operators in (7).

(7) a [CP]p: p is a consequence of the shared record of the conversation (as perhaps derived by copresence heuristics).
b [COMM]p: p can be taken as shared knowledge among members of community COMM (in virtue of members' shared access to particular kinds of information).

In these examples, each of the operators [NAME] represents a necessity operator; necessity is also written $\Box$ or $\Box_i$. Modal logic can also include dual operators of possibility, written $\Diamond$,

$\Diamond_i$ or $\langle$NAME$\rangle$; $\Diamond p$ is true if and only if $\Box\neg p$ is false. (Possibility will not figure much in this dissertation.) We will work in a multimodal logic, in which any finite number of distinct kinds of necessity may be admitted; we take the names of the modal operators as arbitrary and rely on explicit statements when necessary to relate them. (The alternative, exploiting a more structured space of modal operators as in [Thomason, 1998], is also fruitful.)

Modal operators can be interpreted with the breadth found in (4)–(7) because modal logic extends classical logic in a very general way. Formulas in first-order classical logic characterize ordinary entities once-and-for-all. In contrast, the necessary formulas of modal logic provide characterizations of entities that take only restricted information into account. Consider how this applies to (5)–(7). [N]p introduces a characterization p that takes into account only what is true after the next action. Analogously, for [K]p, p takes into account only what the agent knows; and for [CP]p, p takes into account only the information shared in a conversation. These interpretations underlie the original applications of modal logic and its relatives as representation languages [Hintikka, 1962; Prior, 1967; Halpern and Moses, 1985a; Moore, 1985a].

At the same time as a modal formula indicates what kind of information it represents, the formula places restrictions on how it can be used in reasoning. I refer to this restriction using the notion of SCOPE or MODULARITY of deductions: two formulas must lie in the same scope to be combined in reasoning. This restriction follows naturally from interpretations, such as those suggested in (5)–(7), which we want modal statements to have. For example, a specification that describes agents' propositional attitudes must ensure that statements can combine in inference only when they describe the content of a single attitude of a single agent. (Nothing follows if I believe p and YOU believe that p implies q.) The agent and attitude a formula describes determines the scope for that formula in reasoning. Similarly, a specification that describes multiple moments in time must guarantee that only facts true at the same time can be combined in inference. (Nothing follows if p is true TODAY but TOMORROW p implies q.) Here, the time at which a formula holds determines its scope.

The modal discipline of information and modularity applies recursively. A specification that characterizes what I know you know offers such a recursive example. It describes and allows us to reason with a body of limited information—what you know—that is itself NESTED within a scope where only what I know can be taken into account. McCarthy's metaphor of "entering" and "leaving" offers a good handle on this nesting. One is always free to create and enter a new scope, where only limited information derived from the broader scope can apply.

An important theme of this chapter is that proof systems and proof-search algorithms for modal logic can be constructed not just to derive semantically correct judgments about restricted kinds of information, but to derive them syntactically in a way that respects and exploits an explicit discipline of scope during search. To prove $\Box A$, we prove A in a new, nested scope; this new scope is constrained so that only information of the form $\Box A$ applies there.

The advantage of this discipline for search is potentially very strong. Once a body of knowledge is organized into a scoped module subject to this discipline, its logical interactions will be limited to facts in the same or compatible modules. For the purposes of proving *p* in module M, any separate module N will be effectively ignored during search. Neither the possible results nor the alternatives for search derived in module M will be affected by the addition of other modules. Thus, by creating such modules, programmers can ensure that facts are logically unaffected by the broader knowledge base where they are found (thus making specifications more reusable), and can limit the alternative combinations of facts that must be considered during search (thus making specifications more efficient).

What's more, a specification need not have been designed as modular for these insights to apply. An explicit discipline of modularity and scope allows any modal operator to be seen both as specifying its distinct kind of information and as creating its own scopes in proofs. This provides an intuition for understanding how search proceeds in existing modal specifications for temporal or epistemic domains, or for redesigning modal specifications in such domains to improve search.

An explicit discipline of modularity and scope would be intrinsically valuable for knowledge representation. Because of the promise of this idea, research on CONTEXT and on the structure of knowledge bases and logic programs is aimed at exploiting modal languages (and related languages) in just this way [McCarthy, 1987; Miller, 1989; Guha, 1991; McCarthy and Buvač, 1994; Giordano and Martelli, 1994]. The remainder of this section describes how these two perspectives on modal logic have led to the design of different, practical modular representations. The perspective of contextual reasoning, reviewed in section 2.1.1, embraces all connectives of first-order logic, and is able to enforce modularity only because of restrictions on the use of modal operators. The perspective of logic programming, reviewed in section 2.1.2, restricts the use of classical connectives but allows modal operators to be used more generally. These views are orthogonal; indeed, the desideratum of a strong, modular disjunction—introduced in section 2.1.3—suggests the possible advantage of reconciling these perspectives.

Enforcing a discipline of modularity during modal proof search is not automatic, however. The reason is that most domains involve systematic relationships between the different kinds of information represented by different classes of formulas. Such relationships can compromise the semantic basis for modularity, by greatly reducing the extent to which different operators specify independent information; obviously this must seriously weaken or even disable the use of modularity to narrow the search space. Even when modularity remains in the logic, however, it may be impossible to enforce modularity under particular strategies for proof search. On the basis of the available information about the goals and structure of proof, those strategies may no longer be able to detect cases where, given modularity, there is nothing to be gained from combining facts.

Modularity of proof search thus depends both on how different modal operators are related to one another and on the precise strategy by which an algorithm carries out search. The representation and treatment of these factors is the main concern of the rest of this chapter, and recurs during the development of new, more powerful modular langauges in

Chapters 3 and 4.

2.1.1 Context, Modularity and Modality

A concrete example, adapted from [McCarthy and Buvač, 1994], is representative of the operators and the range of inferences about them found in a logical approach to generality and context in AI. The example concerns the representation of product information from different points of view. In particular, General Electric and the Navy have different ideas about what the price of a component is. GE establishes base list prices for each component separately. Navy specifications refer to prices that include not only the cost of the individual component but also the cost of other equipment, such as spare parts, that the Navy will purchase along with it.

The first step in representing this scenario is to make room for the two distinct views of price. We do this using modal operators $[\textsc{list}]$ and $[\textsc{spec}]$. The formula $[\textsc{list}]A$ represents that A reflects the perspective taken in GE's catalogue; likewise, $[\textsc{spec}]A$ represents that A reflects the perspective taken in the Navy's specifications. These modal operators allow us to be precise in making claims about GE's list prices, as in (8a), or about the Navy's specification prices, as in (8b).

(8) a $[\textsc{list}]\ price(x, p)$
    b $[\textsc{spec}]\ price(x, q)$

Following [Guha, 1991], [McCarthy and Buvač, 1994] write formulas like $[\textsc{list}]A$ in the notation $ist(list, A)$, short for A is true in *list*. In this notation, terms like *list* name CONTEXTS [McCarthy, 1987]. The expected tight relationship between these contexts and ordinary modal operators is substantiated in [Buvač *et al.*, 1995]. The main innovation for contexts is a technical device that allows the available vocabulary to depend on context; the extension is designed to allow knowledge representation languages to accommodate ambiguous or partially-defined terms and relations. This kind of context-dependence is peripheral to many uses of the formalism (including that in (8)).

The term "context" provides a useful mnemonic that describes, in a more generally accessible way than "necessity", the use of propositional operators to modulate the force of logical statements. For example, as observed in [McCarthy, 1993], suppose we make the assertion of a formula A in a context C, instead of asserting A directly. Then we can regard C as a source of information embodying the many assumptions and qualifications on which the truth of A inevitably depends. When these assumptions (whatever they may be) are really satisfied, then A is true in C just in case A is true. The use of the context C gives us a new handle on these qualifications, without forcing us to itemize them all explicitly. With such examples, McCarthy argues that contexts provide a key tool for representing generalizations about the world in logic—irrespective of any role of contextual reasoning in search.

However, Guha describes a direct way to improve search using the formulas such as those in (8) in which operators are nested to depth one only [Guha, 1991]. This strategy is implemented in CYC, the large common-sense knowledge base [Guha and Lenat, 1990]. Any query Q is posed in a distinct problem-solving context, as $[\text{PSC}]Q$. The rules that might be applied are organized as MICROTHEORIES—collections of facts asserted in a single context. Facts are included for reasoning in the problem-solving context by selecting them from the (most) relevant and appropriate microtheories for the current problem. Facts from other microtheories are ignored (though perhaps only until a later stage of deduction). This regime establishes the overall query Q as a modular or local goal for proof-search, and introduces a simple discipline of scope to restrict the inferences that are entertained while searching for a derivation of Q. This is the only role of modularity in the system, though. The facts expressed and the inference performed follow classical first-order logic in other respects, and indeed Guha describes the logical apparatus as an ordinary first-order problem-solver operating "in" the problem-solving context.

Guha's strategy is founded on the ability to express general relations of inclusion between sources of information. For example, in reasoning from (8), suppose the perspective of GE's list prices is relevant and appropriate to the current query $[\text{PSC}]Q$. Then from conclusions of the form $[\text{LIST}]A$, results of the form $[\text{PSC}]A$ will follow generally. This is expressed in the axiom scheme in (9).

(9) $[\text{LIST}]A \supset [\text{PSC}]A$

Once the applicable contexts or sources of information are declared, a second step of formalization may be needed. This step introduces axioms that allow the assumptions made in one microtheory to be reconciled with the assumptions in force in another microtheory or in the problem-solving context. For example, for (8), an inference might have to combine information about list prices with information about the Navy's specifications. Recall that to determine the price of a component in the Navy's specification, we start with its list price, and add the list price for its specified spare components. We now formalize this, using the representation in (10):

(10) $$\forall xpyq\ ([\text{LIST}]\ price(x,p) \wedge [\text{SPEC}]\ spares(x,y) \wedge [\text{LIST}]\ price(y,q) \supset [\text{SPEC}]price(x,p+q))$$

Such statements are known as LIFTING axioms; their benefits include allowing different logical modules to express conclusions in local and natural terms, while nevertheless supporting compatible formalisms [Guha, 1991; McCarthy and Buvač, 1994]. Lifting axioms obviously have the potential to complicate the identification of relevant facts during proof-search—for example, if it is necessary to apply them recursively. Guha's implementation permits only a restricted class of lifting rules; lifting is then computed directly by an "access module", which transforms contextualized statements into the ordinary, first-order formulas used for inference in the current problem-solving context.

2.1.2 Modular Logic Programming

Where researchers on context establish a design strategy for generality in knowledge representation on the use of modal logic, researchers in logic programming [Miller, 1989; Giordano and Martelli, 1994; Baldoni *et al.*, 1993] have invoked modal logic—not dissimilarly—to achieve the classic benefits of modular software design. In particular, a modal, modular logic program can streamline the statement of commonalities among problems and algorithms, and can be more easily maintained and reused.

Here is an illustration in the current domain. Since both the list and the specification represent kinds of accounting information, many parallel inferences may be required both in the scope of [LIST] and in the scope of [SPEC]. These common inferences motivate an operator [ACCT] for specifying facts about price that list and specification share. One simple example might be the fact that prices are measured in dollars:

(11) $\forall xp\ [\textsc{acct}]\ (\mathit{price}(x,p) \supset \mathit{dollar\text{-}value}(p))$

For such statements to play their intended role in reasoning, we again need a way to apply information of one kind (or one context) towards results of another. To draw inferences about the units in which list and specification record prices using (11), we need to be able to infer [LIST]A and [SPEC]A when we have [ACCT]A.

With operators that interact this way, we can present needed generalizations once, with an intuitive annotation that succinctly describes the range of contexts to which the facts apply. They also give useful separate roles to the operators and to the first-order components of formulas like [LIST]$\mathit{price}(x,p)$ and [SPEC]$\mathit{price}(x,p)$. This illustrates how a modal or contextual representation can be more natural and convenient than a first-order representation with corresponding compound symbols, such as $\mathit{list\text{-}price}(x,p)$ and $\mathit{spec\text{-}price}(x,p)$, for which common generalizations like the measurement of price in dollars must be spelled out explicitly and separately.

Similarly, we might want to represent that GE and the Navy may have different, partial information about what is in the list and the specification, by introducing operators [GE] and [NAVY]: [GE]A indicates that A is to be proved taking only GE's information into account. Then an operator [BOTH] that records information that GE and the Navy share will also be needed to capture commonalities in [GE] and [NAVY]. It might be used, for example, to record that the two organizations are aware of the method of calculating prices described by (10):

(12) $$\forall xpyq\ [\textsc{both}]\ ([\textsc{list}]\ \mathit{price}(x,p) \wedge [\textsc{spec}]\ \mathit{spares}(x,y) \wedge \\ [\textsc{list}]\ \mathit{price}(y,q) \supset [\textsc{spec}]\mathit{price}(x,p+q))$$

Again, to use (12) as intended, we need a way to infer [GE]A and [NAVY]A from [BOTH]A.

The use of nested operators, as in (12), is characteristic of the modal approach to structuring logic programs. In fact, Miller's original proposal [Miller, 1989] (as reconstructed in [Giordano and Martelli, 1994]) involves only a single modal operator and uses nesting exclusively to assemble modular specifications. Here's the basic idea. Suppose we have a query Q which we interpret in a modular way: we want to take into account only clauses C, D and E when proving Q. We formalize this using a hypothetical implication, as in (13).

(13) $\Box\ (C \wedge D \wedge E \supset Q)$

The modal operator restricts the information to take into account and ensures that the implication exhibits a strong form of modularity—it instructs the interpreter to make local assumptions of C, D and E which are to be used only in proving Q. (Shoham offers a similar proposal in the context literature [Shoham, 1991].) Because of this strong modularity, the goal can be processed correctly and efficiently—no matter how it is nested in other modal goals—on a logic programming search strategy, where goals are broken down directly according to their syntax and processed independently. With strong modularity, a variety of regimes for assembling modular logic programs can be implemented using modal operators and nested meaningfully, usefully and efficiently [Giordano and Martelli, 1994]. The identification of this strong interpretation of modularity is thus a key contribution of research on modal logic programming.

Further benefits come in the logic programming approach from RELATING the information that can be used in the different scopes created by nested uses of modal operators. For knowledge representation, this may make possible a range of natural inferences that would be difficult or impossible to describe otherwise. The need for this strategy in one direction is already present with (12). To use (12) as we would use (10), we need a way to infer A from $[\text{BOTH}]A$. Here we apply nested information in a wider setting.

In the other direction, transferring results from outside into nested scopes, this strategy is required in modeling the HYPOTHETICAL reasoning of agents. An agent needs such modeling to explain other agents' motivations and intentions. Moreover, as argued in Chapter 7, an agent must also use it to derive its own plans for the future.

To illustrate hypothetical reasoning and nested scopes, suppose what the Navy knows is specified as in (14).

(14) $$[\text{NAVY}]\ [\text{LIST}]\ \mathit{price}(\mathit{fx22\text{-}engine}, 3600) \wedge \\ [\text{NAVY}]\ [\text{SPEC}]\ \mathit{spares}(\mathit{fx22\text{-}engine}, \mathit{fx22\text{-}fan\text{-}blades})$$

That is, the Navy knows it needs FX22 fan blades with its FX22 engine, and the Navy has GE's list price as \$3.6 million. Suppose we know that the Navy needs to know the spec price for the FX22 engine. Should we then expect the Navy to ask what the price of FX22 fan blades is?

The answer must depend in part on our characterization of what the Navy knows. Underlying any question from the Navy, there must be a plan whereby the Navy recognizes (taking just what it knows into account) that it will be able to determine its specification price once it has the list price for FX22 fan blades. It would seem that (14) and (12) ascribe to the Navy enough information to accomplish this recognition. Formally, the fact that must be derived is (15).

(15) $$\forall p\ [\text{NAVY}]\ ([\text{NAVY}]\ [\text{LIST}]\ \mathit{price}(\mathit{fx22\text{-}fan\text{-}blades}, p) \supset \\ [\text{NAVY}]\ [\text{SPEC}]\ \mathit{price}(\mathit{fx22\text{-}engine}, 3600 + p))$$

We would like to derive (15) as a consequence of (12), but our goal has the form $[\text{NAVY}]\ [\text{NAVY}]\ \mathit{price}(x, p)$, with a double embedding. Our strategy is appeal to an inference from $[\text{NAVY}]A$ to $[\text{NAVY}][\text{NAVY}]A$—what the Navy knows, it knows that it knows.

This inference expresses a different, natural relationship between scopes. We continue by establishing

(16) $[\text{NAVY}]\ [\text{NAVY}]\ [\text{LIST}]\ \textit{price}(\textit{fx22-engine}, 3600)\ \wedge$
$[\text{NAVY}]\ [\text{NAVY}]\ [\text{NAVY}]\textit{spares}(\textit{fx22-engine}, \textit{fx22-fan-blades})\ \wedge$
$[\text{NAVY}]\ [\text{NAVY}]\ [\text{GE}]\textit{price}(\textit{fx22-fan-blades}, p)$

The first two must be obtained indirectly from (14), by the same inference about nested scopes. We get the third because it is assumed in proving the implication. This establishes the result.

2.1.3 Modularity: Limits and Prospects

The rich logical statements needed to represent common-sense knowledge about the world—as explored in research on context like [Guha, 1991] or as described in Chapters 6 and 7 of this dissertation—fall outside the expressive power of the logic programming languages proposed by [Miller, 1989; Giordano and Martelli, 1994; Baldoni *et al.*, 1993; Baldoni *et al.*, 1996]. Nevertheless, the nested modal operators and strong modularity of the logic programming approach ought to be just as valuable for those richer statements.

Modal disjunctions, the topic of Chapter 3, offer a provocative such example. Modal disjunctions can not only support reusable and precise descriptions of rich domains; under the strong, logic programming interpretation of modularity, they can also reduce the size of proofs and the number of ambiguities in proofs that need to be considered simultaneously. The following argument suggests why. From the point of view of search, a disjunction such as $C \vee D$ can be seen as an instruction to perform proof by cases, in which the proof proceeds separately for the one case where C is true and for the other case where D is true and (perhaps) C is false. In general, the two cases introduced by a disjunction may span the entire proof and may have no inferences in common. A modal disjunction, in contrast, is much more restricted. First, the disjunction $\Box(C \vee D)$ applies towards some goal $\Box G$ which depends on the same kind of information. In addition, if this information is modular and local, the disjunction permits case analysis to be considered for the proof of this goal, and this goal only. In other words, the two cases introduced by $\Box(C \vee D)$ span just the subproof in which $\Box G$ is proved; the overall derivation to which $\Box G$ contributes has no global ambiguity.

Not every plausible principle for relating nested operators is compatible with the strong interpretation of modularity needed for modular disjunction or the logic programming interpretation of $\supset$ illustrated by (13), however. For example, in formalizing the logic of contexts, [Buvač *et al.*, 1995] postulates an axiom scheme (Δ) which can be written in modal logic as in (17).

(17) $(\Delta) : \Box(\Box A \vee B) \supset (\Box\Box A \vee \Box B)$

In the presence of (Δ) axiom, it is no longer correct to treat assumptions made during proof search as local or modular, and to attack modular goals by separate inferencing. With separate inferencing, whenever a goal $\Box(\Box p \supset q)$ arises, it suffices to apply the assumption $\Box p$ to the proof of q. But consider the complex goal in (18).

(18) $\Box\Box p \vee \Box(\Box p \supset q)$

This goal is provable given (Δ). From a formal point of view, to see why, start with the formula $\Box p \vee \neg\Box p \vee q$ which, because it is a tautology, must be necessarily true. From this statement of necessity, (Δ) can apply to get $\Box\Box p \vee \Box(\neg\Box p \vee q)$, a logical equivalent of (18). Informally, to prove (18) requires the assumption of $\Box\Box p$, ostensibly made as part of proving q in the second disjunct, instead to be applied towards proving the first disjunct.

Also incompatible with strong modularity is the axiom scheme of negative introspection presented in (19).

(19) $\neg\Box A \supset \Box(\neg\Box A)$

This axiom is often used to characterize the knowledge of agents in finite domains [Fagin *et al.*, 1995]. (19) entails the statement in (20) which is again incompatible with strong modularity; (20) follows from (19) in much the same way (18) follows from (17).

(20) $\Box p \vee \Box(\Box p \supset q)$

This study eschews axioms like (Δ) and negative introspection that are incompatible with strong modularity. Strong modularity is too useful for structuring and searching logical specifications. Ultimately, this choice must be assessed based on the logical results that follow from it, in Chapters 3, 4 and 5, and the illustrative specifications constructed in this framework, in Part II. In the meantime, note anyway that axioms like (Δ) and negative introspection are most often motivated not by key inferences of this pattern that domains require, but by the ease of analysis of their simple mathematical semantics—permitting the use of simple normal forms for formulas or models. As far as applications to AI or NLG go, such considerations are not particularly compelling.

Note that representations with the expressive power to describe entities based on limited information are not always couched as modal languages. [Morgenstern, 1987; Ballim *et al.*, 1991; McCarthy and Buvač, 1994] offer alternative proposals to describe concepts like those in (4)–(7) in a computational setting. Although the formalizations differ, the key feature of each is an operator that defines scopes in deductions in which the use of formulas is restricted—variously, necessity, quotation, boxes, and contexts—along with rules that govern the transfer of formulas from one scope to another. The effects that these features achieve in the different formalisms are strikingly similar. Accordingly, we can stick to modal logic here without regret.

2.2 Modal Logic in a Nutshell

2.2.1 Syntax

We consider a family of first-order modal languages, $\mathcal{L}^{\Box}_{m,T}$. $\mathcal{L}^{\Box}_{m,T}$ is parameterized by a set of m paired operators of necessity, $\Box_i$, and possibility $\Diamond_i$, for finite integer m; and by a theory T specifying relations between operators in terms of the three modular axiom schemes introduced in the previous section:

(INC)	(inclusion)	$\Box_i A \supset \Box_j A$
(VER)	(veridicality)	$\Box_i A \supset A$
(PI)	(positive introspection)	$\Box_i A \supset \Box_i \Box_i A$

(In general, we may invoke a variety of axioms to augment the basic modal logic K to better match modal operators and the common-sense notions they are meant to model. For example, the combination of (VER) and (PI) known as S4, provides a model of knowledge, because whatever an agent knows is true (in keeping with (VER)) [Hintikka, 1962]. Since an agent's beliefs may not be true, (PI) alone may be a better model of belief. This logic is known as K4. Finally, modalities are called T when governed just by (VER).)

As usual, we presume a signature describing available constant symbols and applicable relation symbols (with arities), and thus a set of atomic formulas of the form $p(t_1, \ldots, t_n)$. (We will stick to a basic language without function symbols and equality to keep closer to what we can ultimately implement in a logic programming language.) Schematizing such formulas as P, the formulas of $\mathcal{L}^{\Box}_{m,T}$ are described as A by the following grammar:

$$A ::= P \mid A \wedge A \mid A \supset A \mid A \vee A \mid \neg A \mid \Box_i A \mid \Diamond_i A \mid \forall x.A \mid \exists x.A$$

Although some of these connectives may be defined in terms of others, we will refrain from doing so as we will be interested not only in this language as a whole, but in FRAGMENTS of the language which can not express those definitions. Two fragments of particular importance are the PROPOSITIONAL fragment, which omits the quantifiers, and the $\Box$-ONLY fragment, which omits negation and all of the $\Diamond_i$.

The usual definitions of free and bound variables carry over to modal logic. $A[t/x]$ denotes the result of substituting t for x in A, with bound variables in A renamed when the same variable appears free in t, to avoid capture. (We will treat formulas differing only in the names of bound variables as identical.) In allowing terms to be substituted freely inside $\Box_i A$ and $\Diamond_i A$, we implicitly adopt the INCREASING or CUMULATIVE domain constraint for modal logics, which allows formulas in nested scopes to refer freely to objects introduced outside. Objects introduced in nested scopes need not be available outside. We will, of course, make a corresponding assumption in the semantics.

2.2.2 Semantics

Modal semantics is based on a model of possible worlds [Kripke, 1963]. Possible worlds record different ways things could turn out; for example, different entities exist at different possible worlds and relation symbols characterize different tuples of entities at different possible worlds. A formula is true or false, then, only at a particular possible world. Effectively, in the modal semantics, each relation in an atomic formula, and each compound formula, has a new parameter on which it implicitly depends, its WORLD OF EVALUATION.

In possible worlds models, the connectives of first-order logic are interpreted classically at the current world of evaluation. Modal operators are interpreted as restricted quantifiers over worlds. The restriction is accomplished by using an ACCESSIBILITY RELATION R_i to interpret $\Box_i$ and $\Diamond_i$. At each world w, we identify a set of possibilities accessible by i using R_i: $\{w' \mid R_i ww'\}$. A formula is necessary at w when it is true at all these accessible possibilities; a formula is possible at w when it is true at one of these accessible possibilities.

Various properties of modal operators can be captured by constraints on the worlds and accessibility relations used to interpret them. In particular, given some intended interpretation of the modal operators in a domain and appropriate combinations of the three axiom schemes (VER), (PI), and (INC), we can obtain a suitable representative class of models this way. For example, by admitting only reflexive relations for R_i, we capture exactly the consequences of the (VER) axiom scheme for $\Box_i$. Now (VER) may also hold in models which are not reflexive; in fact, no modal formula picks out exactly the reflexive (or the irreflexive) accessibility relations [van Benthem, 1984]. Associating (VER) with reflexive models suffices, however, because no modal formula can be true (or false) at a world in an irreflexive model which satisfies all instances of (VER) unless that formula is also true (or false) at some world in some reflexive model.

Similarly, we can impose a constraint of transitivity on R_i to obtain a representative class of models to capture the (PI) scheme for $\Box_i$, and we can use a constraint of inclusion $R_j \subseteq R_i$ to implement the (INC) scheme $\Box_i A \supset \Box_j A$. Moreover, we can capture combinations of these schemes straightforwardly by imposing combinations of these accessibility constraints.

Formally, these ideas are fleshed out using the following definitions. A Kripke model for $\mathcal{L}^{\Box}_{m,T}$ is a tuple $M = \langle W, R, D, F \rangle$ such that:

- W is a nonempty set of worlds.
- R is a family of m binary relations on W. If $\Box_i$ is governed by (VER), then we impose the additional restriction that R_i is REFLEXIVE: for all w in W, $R_i ww$. If $\Box_i$ is governed by (PI), then we impose the additional restriction that R_i is TRANSITIVE: for all u, v and w in W, if $R_i uv$ and $R_i vw$ then $R_i uw$. Finally, if $\Box_i$ and $\Box_j$ are governed by (INC) with $\Box_i A \supset \Box_j A$, then we impose the additional restriction that $R_j \subseteq R_i$.
- D is a function assigning sets to elements of W such that all $D(w)$ are nonempty and for all u and v in W and all R_i, if $R_i uv$ then $D(u) \subseteq D(v)$
- F is an interpretation of constant symbols and relation symbols. For any constant symbol c, $F(c) \in \cap\{D(w) \mid w \in W\}$. For any n-ary relation symbol p, $F(p)(w) \subseteq D(w)^n$.

A term t is interpreted as an element of the model $[\![c]\!]^{w,g}$, where g is an assignment taking variables to values in $D(w)$. If t is a constant c, then $[\![c]\!]^{w,g} = F(c)$; if t is a variable x, then $[\![x]\!]^{w,g} = g(x)$.

Then the truth of a formula p in M respect to a world of evaluation w and an assignment g is defined by an inductive condition $[\![p]\!]^{w,g}$ as follows:

- $[\![r(t_1,\ldots,t_n)]\!]^{w,g}$ exactly when $\langle [\![t_1]\!]^{w,g},\ldots,[\![t_n]\!]^{w,g}\rangle \in F(r)(w)$.
- $[\![p \wedge q]\!]^{w,g}$ exactly when $[\![p]\!]^{w,g}$ and $[\![q]\!]^{w,g}$.
- $[\![p \vee q]\!]^{w,g}$ exactly when either $[\![p]\!]^{w,g}$ or $[\![q]\!]^{w,g}$.
- $[\![p \supset q]\!]^{w,g}$ exactly when either $[\![p]\!]^{w,g}$ is false or $[\![q]\!]^{w,g}$ is true.
- $[\![\neg p]\!]^{w,g}$ exactly when not $[\![p]\!]^{w,g}$.
- $[\![\forall xp]\!]^{w,g}$ exactly when for all e in $D(w)$, $[\![p]\!]^{w,g'}$ for the function g' exactly like g except that $g'(x) = e$.
- $[\![\exists xp]\!]^{w,g}$ exactly when there is some e in $D(w)$ such that $[\![p]\!]^{w,g'}$ for the function g' exactly like g except that $g'(x) = e$.
- $[\![\Box_i p]\!]^{w,g}$ exactly when for all v such that $R_i wv$, $[\![p]\!]^{v,g}$.
- $[\![\Diamond_i p]\!]^{w,g}$ exactly when for some v such that $R_i wv$, $[\![p]\!]^{v,g}$.

A formula p is valid in $\mathcal{L}^{\Box}_{m,T}$ exactly when, for every Kripke model M, $[\![p]\!]^{w,g}$ is true for all g and all w in W.

2.3 Proofs in Modal Logic and Modular Search

This section describes syntactic methods that allow the validity of formulas in $\mathcal{L}^{\Box}_{m,T}$ to be determined. For expository purposes, we focus on proof systems for propositional modal logic with a single modal operator only. Quantified systems with multiple modalities can be constructed along the same lines, although they tend to involve considerable technical complexity; their properties are similar to the corresponding simple propositional systems.

The analysis of these methods reveals the surprising difficulty of exploiting modularity in inference while avoiding redundancy in search. As outlined informally in section 2.1, imposing modularity allows the structure of modal formulas to be used to constrain candidate inferences. Several proof systems establish this modularity directly, by dividing the construction of a proof into modular stages and restricting what inferences are applicable at each stage. These constraints on inference make the modularity of modal logic concrete in an intuitive way. However, the division of proof-search into modular stages entails substantial redundancy. For example, the same inferences must be considered repeatedly at many stages of the proof. In addition, statements whose use involves inferences at several stages of proof are difficult to control. These redundancies prohibit the practical use of these systems except on restricted classes of problems. This section reviews two such systems: the Hilbert-style axiomatic method for modal inference, as typically considered in proofs of soundness and completeness for modal deduction, which is explored in section 2.3.1; and structurally-scoped proof systems in the style of Gentzen, which are explored in section 2.3.2.

The semantics of modal logic suggests how these explicit stages of proof can be avoided. A proof can associate each formula with the possible world to which it belongs and reason explicitly about relations among worlds. This treats modal formulas exactly like formulas of classical logic, and therefore allows deduction methods for classical logic like resolution to be used directly. Such techniques, described in section 2.3.3, have the advantage of avoiding redundancy. However, they have the disadvantage that there is no provision for modularity in the structure of the proof; the classical techniques they are based allow global interactions among ambiguities in the proof—as is required by the general absence of modularity in classical logic.

To achieve modularity without redundancy, we need to exploit both modal syntax and classical techniques. Sections 2.3.4 and 2.3.5 describe systems in which such exploitation can be more easily described. Section 2.3.4 introduces a sequent calculus that operates on modal formulas labeled with terms for possible worlds; section 2.3.5 shows how this system is LIFTED to use unification. In this system, the labels on formulas and unification eliminates the redundancy and unnecessary commitment inherent in structurally-scoped sequent calculi. However, the modal syntax of formulas is available in deductions so that constraints of modularity can be explicitly imposed.

This lifted, explicitly-scoped proof system will be our focus throughout the remainder of the dissertation. In particular, Chapter 3 shows how to refine this proof system to impose a modular, logic-programming search strategy, while Chapter 4 shows how to refine it further to constrain the interaction of modular statements more efficiently.

2.3.1 Hilbert Systems

Inference in modal logic is most succinctly and intuitively characterized by Hilbert Systems. In these systems, a proof is a linear sequence of formulas where each formula is either an instance of one of numerous axioms, or derivable from earlier formulas by the action of simple inference rules. (By contrast, the proofs considered in sections 2.3.2–2.3.5 have tree structure and only a single axiom.) For the simplest propositional modal logic, *K*, there are three axiom schemes:

$$\begin{array}{ll} A1. & \textit{Any tautology of classical propositional logic} \\ A2. & \Box A \supset (\Box(A \supset B) \supset \Box B) \\ A3. & \Diamond A \equiv \neg\Box\neg A \end{array}$$

These are combined by two rules of inference:

$$\begin{array}{ll} R1.\textit{(modus ponens)} & \text{From } A \text{ and } A \supset B \text{ infer } B. \\ R2.\textit{(necessitation)} & \text{From } A \text{ infer } \Box A. \end{array}$$

Principles relating scopes are accommodated by simply by adding the appropriate additional axiom schemas.

This proof system is sound and complete: all and only valid statements have derivations in this system. A recent and thorough treatment of this system and proofs for this result can be found in [Fagin *et al.*, 1995, pp. 48–62] (among other places).

The axioms and inference rules of Hilbert systems square straightforwardly with the informal conception of modularity and scope motivated in section 2.1. In particular, the axiom $A2$ represents a direct statement that modular information can be combined in inference with other modular information of the same type. This modularity is elevated to a global characteristic of the proof system simply by the absence of any other rules that could combine information across modules. Even the rule $R2$ of necessitation does not create a new combination of information across scopes. It only allows a single, timelessly valid statement to percolate into a nested scope.

Because of this straightforwardness, and because they directly import the results of classical propositional logic, Hilbert systems can sometimes facilitate mathematical study of modal systems, for example in proofs of soundness and completeness. However, there are a number of computational drawbacks to Hilbert systems as a method for proof search. One problem is the size of Hilbert proofs. Proofs in Hilbert systems can be rather involved, because Hilbert systems involve rather cumbersome manipulation of assumptions. Consider the S4 theorem in (21).

(21) $\Box(a \supset \Box b) \supset \Box\Box(a \supset b)$

A proof is provided in Figure 2.1; it requires fifteen steps and some brutally explicit propositional reasoning. There is a compensating factor—the fact that Hilbert systems allow results to be reused repeatedly after they have been established. This is required for some formulas to have compact proofs and is not present in all proof systems [D'Agostino, 1992]. However the same advantage can be obtained without the drawbacks of Hilbert systems with methods such as resolution. In deference to the complexity of Figure 2.1, I will forgo the presentation of alternative proofs in this system (although theoretically-minded readers might be interested in working out for themselves proofs in this system which showed different ways of propagating modular information, or which illustrated the modularity of modal case analysis).

A more substantial obstacle to automated proof in Hilbert systems is control of the search space. This lack of control is reflected in the difficulty of summarizing intuitively why some non-theorem cannot be proved in a Hilbert system. Take the non-theorem $\Box(p \supset q) \vee \Box p$. In a Hilbert system, we cannot say that no rules could have applied to yield this conclusion; in fact, many propositional tautologies might have been used to infer this statement by modus ponens. We can only observe vaguely that because the Hilbert system maintains a discipline of scope and modularity, none of these steps could have been completed backward into a full proof.

A formal statement of this difficulty is that Hilbert Systems lack the SUBFORMULA property common to proof systems used in efficient classical theorem-proving methods. The subformula property guarantees that if a result Γ is provable in a system, then there is a proof of Γ in the system in which each formula used is a substitution instance of a

1. $\Box b \supset b$ [(VER)]
2. $\Box(\Box b \supset b)$ [necessitation, 1]
3. $(\Box b \supset b) \supset (a \supset \Box b) \supset (a \supset b)$ [instance of tautology $(p \supset q) \supset (r \supset p) \supset (r \supset q)$]
4. $\Box((\Box b \supset b) \supset (a \supset \Box b) \supset (a \supset b))$ [necessitation, 3]
5. $\Box(\Box b \supset b) \supset \Box((\Box b \supset b) \supset (a \supset \Box b) \supset (a \supset b)) \supset$ $\Box((a \supset \Box b) \supset (a \supset b))$ [A2]
6. $\Box((\Box b \supset b) \supset (a \supset \Box b) \supset (a \supset b)) \supset \Box((a \supset \Box b) \supset (a \supset b))$ [modus ponens, 2, 5]
7. $\Box((a \supset \Box b) \supset (a \supset b))$ [modus ponens, 4, 6]
8. $\Box(a \supset \Box b) \supset \Box((a \supset \Box b) \supset (a \supset b)) \supset \Box(a \supset b)$ [A2]
9. $(\Box(a \supset \Box b) \supset \Box((a \supset \Box b) \supset (a \supset b)) \supset \Box(a \supset b)) \supset$ $(\Box((a \supset \Box b) \supset (a \supset b)) \supset \Box(a \supset \Box b) \supset \Box(a \supset b))$ [instance of tautology $(p \supset q \supset r) \supset (q \supset p \supset r)$]
10. $\Box((a \supset \Box b) \supset (a \supset b)) \supset \Box(a \supset \Box b) \supset \Box(a \supset b)$ [modus ponens, 8, 9]
11. $\Box(a \supset \Box b) \supset \Box(a \supset b)$ [modus ponens, 10, 7]
12. $\Box(a \supset b) \supset \Box\Box(a \supset b)$ [(PI)]
13. $(\Box(a \supset \Box b) \supset \Box(a \supset b)) \supset (\Box(a \supset b) \supset \Box\Box(a \supset b)) \supset$ $(\Box(a \supset \Box b) \supset \Box\Box(a \supset b))$ [instance of tautology $(p \supset q) \supset (q \supset r) \supset (p \supset r)$]
14. $(\Box(a \supset b) \supset \Box\Box(a \supset b)) \supset (\Box(a \supset \Box b) \supset \Box\Box(a \supset b))$ [modus ponens, 11, 13]
15. $\Box(a \supset \Box b) \supset \Box\Box(a \supset b)$ [modus ponens, 12, 14]

Figure 2.1: Proving $\Box(a \supset \Box b) \supset \Box\Box(a \supset b)$ in a Hilbert system.

subformula of Γ (that is, the formula may be obtained from by this subformula by freely substituting terms for free variables). In general, the use of complex axioms and modus ponens runs counter to the subformula property, because it forces the deduction of a formula B from a formula A to appeal to an explicit derivation of a more complicated formula, $A \supset B$. In modal logic, in virtue of the nested application of modus ponens (using A2), the more complicated formula that must be derived to carry the inference forward—$\Box(A \supset B)$—is even more indirectly related to premise ($\Box A$) and conclusion ($\Box B$).

In theorem-proving, the subformula property is crucial for controlling search, because it allows a search engine to rule out options for extending a proof as soon as those options would introduce non-subformulas. Such methods of ruling out options are vital in allowing a theorem-prover to detect failure in one branch of proof search and move on to another. The subformula property also streamlines theorem-proving by enabling a variety of methods for improving space usage by structure-sharing [Boyer and Moore, 1972].

2.3.2 Structurally Scoped Sequent Calculi

One way to construct a modal proof system that does satisfy the subformula property is to extend the cut-free sequent calculus of classical logic of [Gentzen, 1935] with rules for modal operators. The sequent calculus works with statements of the form $\Gamma \longrightarrow \Delta$, where Γ and Δ are both multisets of formulas. Such statements, called SEQUENTS, indicate that one of the Δ formulas always holds, assuming all the formulas in Γ do. Rules for logical connectives can be written so as to derive any sequent containing a complex formula from a related sequent or sequents involving its constituents. [Gentzen, 1935] proved a fundamental result showing that such rules, which guarantee the subformula property, suffice to describe classical and intuitionistic deduction. This result can be extended to many modal logics—notably including S4, where Gentzen-style sequent systems go back to [Onishi and Matsumoto, 1957; Onishi and Matsumoto, 1959; Kanger, 1957]. (See also [Fitting, 1983].) The new rules for modal operators are set up so as to directly follow the modularity of modal logic. For example, to implement the requirement that a necessary statement must be proved by taking only other necessary information into account, the sequent rule for proving necessary statements discards formulas from Γ and Δ that do not represent necessary information.

A sample modal sequent calculus is shown in Figure 2.2. It covers the propositional logic of a single S4 modal operator □ (governed by (VER) and (PI)). This sequent calculus represents a sound and complete inference system for the same semantics as the Hilbert system characterizes; it is an equivalent system. (For example, [Smullyan, 1973] leverages his trademark metatheoretic concision to prove this.) However, this system respects the subformula property, because reasoning can be performed directly inside the scope of modal rules, without the mediation of rules like necessitation (R2) or axioms like consequential closure (A2).

Proofs in this system are trees built in accordance with the inference rules in the figure. The label of a node in a proof-tree is a sequent; the label of the root of a proof is called its

$$\frac{}{\Gamma, A \longrightarrow A, \Delta}\,\text{initial}$$

$$\frac{\Gamma, A \wedge B, A, B \longrightarrow \Delta}{\Gamma, A \wedge B \longrightarrow \Delta}\,\wedge \rightarrow$$

$$\frac{\Gamma \longrightarrow A \wedge B, A, \Delta \qquad \Gamma \longrightarrow A \wedge B, B, \Delta}{\Gamma \longrightarrow A \wedge B, \Delta}\,\rightarrow \wedge$$

$$\frac{\Gamma, A \vee B, A \longrightarrow \Delta \qquad \Gamma, A \vee B, B \longrightarrow \Delta}{\Gamma, A \vee B \longrightarrow \Delta}\,\vee \rightarrow$$

$$\frac{\Gamma \longrightarrow A \vee B, A, B, \Delta}{\Gamma \longrightarrow A \vee B, \Delta}\,\rightarrow \vee$$

$$\frac{\Gamma, A \supset B \longrightarrow A, \Delta \qquad \Gamma, A \supset B, B \longrightarrow \Delta}{\Gamma, A \supset B \longrightarrow \Delta}\,\supset\rightarrow$$

$$\frac{\Gamma, A \longrightarrow A \supset B, B, \Delta}{\Gamma \longrightarrow A \supset B, \Delta}\,\rightarrow\supset$$

$$\frac{\Gamma, \neg A \longrightarrow A, \Delta}{\Gamma, \neg A \longrightarrow \Delta}\,\neg \rightarrow$$

$$\frac{\Gamma, A \longrightarrow \neg A, \Delta}{\Gamma \longrightarrow \neg A, \Delta}\,\rightarrow \neg$$

$$\frac{\Gamma, \Box A, A \longrightarrow \Delta}{\Gamma, \Box A \longrightarrow \Delta}\,\Box \rightarrow$$

$$\frac{\{\Box B \mid \Box B \in \Gamma\} \longrightarrow A, \{\Diamond B \mid \Diamond B \in \Delta\}}{\Gamma \longrightarrow \Box A, \Delta}\,\rightarrow \Box$$

$$\frac{\Gamma \longrightarrow A, \Diamond A, \Delta}{\Gamma \longrightarrow \Diamond_i A, \Delta}\,\rightarrow \Diamond$$

$$\frac{\{\Box B \mid \Box B \in \Gamma\}, A \longrightarrow \{\Box B \mid \Box B \in \Delta\}}{\Gamma, \Diamond A \longrightarrow \Delta}\,\Diamond \rightarrow$$

Figure 2.2: Structurally-scoped, cut-free sequent calculus for the propositional modal logic of a single S4 modal operator $\Box$.

end-sequent.

Any instantiation of the initial rule $\Gamma, A \longrightarrow A, \Delta$ is a proof (so a result of A and any other facts can be derived from an assumption of A and any other facts). Given proofs $\mathcal{D}_1$ and $\mathcal{D}_2$ with end-sequents $\Gamma_1 \longrightarrow \Delta_1$ and $\Gamma_2 \longrightarrow \Delta_2$, for any

$$\frac{\Gamma_1 \longrightarrow \Delta_1}{\Gamma \longrightarrow \Delta}$$

that instantiates a (unary) inference rule of Figure 2.2, the tree

$$\frac{\mathcal{D}_1}{\Gamma \longrightarrow \Delta}$$

is a proof; for any

$$\frac{\Gamma_1 \longrightarrow \Delta_1 \qquad \Gamma_2 \longrightarrow \Delta_2}{\Gamma \longrightarrow \Delta}$$

that instantiates a (binary) inference rule, the tree

$$\frac{\mathcal{D}_1 \qquad \mathcal{D}_2}{\Gamma \longrightarrow \Delta}$$

is a proof. All proofs are constructed in accordance with these schemes.

Although it is convenient to define proofs by this top-down characterization, it is typically more natural to read proofs from bottom up, as a record of proof-search for an end-sequent. Read thus, each rule DECOMPOSES the outer connective in a distinguished formula in the end-sequent, called the PRINCIPAL FORMULA of the rule. This yields new, typically smaller search problems: the immediate subformulas of the principal formula, the SIDE FORMULAS of the rule application, occur in the end-sequents of $\mathcal{D}_1$ (and $\mathcal{D}_2$) in place of the principal formula. As written in Figure 2.2, the inference rules also carry over the principal formula from the end-sequent to higher sequents. This convention allows formulas to be used repeatedly in proofs (without it, a structural rule of contraction is required), but since the duplicated formulas clutter proofs I will occasionally suppress them. (For more on alternative representations of structure in sequent proofs, see e.g. [Gallier, 1993].)

Informally, proofs in this system consist of contiguous regions where reasoning is performed in a single scope or modular context. In the proof, applications of $(\rightarrow)$ and $(\Diamond \rightarrow)$ mark the boundaries between scopes. The entire subproof ABOVE each application is more deeply NESTED in scope, by the application of one $\Box$ operator. The $(\Box \rightarrow)$ and $(\rightarrow \Diamond)$ rules represent applying necessary information in the current scope.

The restriction that only necessary information can be used in nested scopes—or that necessary information can only be used in nested scopes—is achieved by filtering the formulas in the sequent at scope transitions. This filtering can be designed to reflect the laws of a variety of modal logics. Only those formulas that describe the nested scope will survive the transition from below the application of $(\rightarrow \Box)$ to above it; the rest will be discarded. Above the transition, surviving formulas are modified to reflect their strength in the new, nested scope. Thus, in the S4 rules shown in Figure 2.2, necessary formulas are kept while others are discarded; and the necessary formulas remain necessary above the transition, in keeping with (PI). For some logics, the $(\Box \rightarrow)$ and $(\rightarrow \Diamond)$ rules also have to introduce boundaries, because in these logics necessary information can be applied ONLY in nested scopes. In Figure 2.2 the rules do not introduce boundaries; this achieves the effect of the (VER) axiom of S4.

The informal characterization of modularity in the structurally scoped sequent calculus of Figure 2.2 can be substantiated with concrete illustrations of the proof system. For example, with reference to a few simple cases, we can explain why there is no proof for a non-theorem like $\Box(p \supset q) \vee \Box p$ (where a proof would have to circumvent the modularity of the assumption of p). The proof must end with the $(\rightarrow \vee)$ rule, above which $(\rightarrow \Box)$ must apply to one of the new formulas. Two alternative proof fragments might be derived:

(22) a
$$\cfrac{\cfrac{\xrightarrow{!} p \supset q}{\longrightarrow \Box(p \supset q), \Box q}\;\rightarrow \Box}{\longrightarrow \Box(p \supset q) \vee \Box q}\;\rightarrow \vee$$

b
$$\cfrac{\cfrac{\xrightarrow{!} q}{\longrightarrow \Box(p \supset q), \Box q}\;\rightarrow \Box}{\longrightarrow \Box(p \supset q) \vee \Box q}\;\rightarrow \vee$$

Note how in either case, when the $(\rightarrow \Box)$ rule applies, alternative formulas from Δ are discarded, accomplishing a transition to a new modular subproof of the overall derivation. The goals that remain in these subproofs are classical sequents that are obviously unprovable (as the sequent calculus rules are promptly, if not immediately, exhausted).

Where (22) shows how structure of proofs enforces modularity, Figures 2.3, 2.4 and 2.5 show how proofs can record different strategies for applying modal information. The figures provide proofs in this system of three sequents involving a single S4 modality:

$$\Box(a \supset \Box b) \longrightarrow \Box\Box(a \supset b)$$
$$\Box(a \supset \Box b) \longrightarrow \Box(a \supset \Box b)$$
$$\Box(a \supset \Box b) \longrightarrow a \supset \Box\Box b$$

The theorems involve necessary assumptions that may be used in three different scopes: twice nested, once nested, or not nested at all.

The key difference between the different proofs is the scope (and thus the order) in which the lower $(\Box \rightarrow)$ rule applies. This rule is highlighted by a box in the proofs. In the first proof, this rule lies inside two nested scopes—above both applications of $(\rightarrow \Box)$. In the second, it lies inside one—above one application of $(\rightarrow \Box)$. In the third, it is used at root scope.

The scoped location of this application of $(\Box \rightarrow)$ is crucial in each case to allowing

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a \longrightarrow a \qquad \dfrac{b \longrightarrow b}{\Box b \longrightarrow b}\,\Box\rightarrow}{a, a \supset \Box b \longrightarrow b}\,\supset\rightarrow}{a, \Box(a \supset \Box b) \longrightarrow b}\,\boxed{\Box\rightarrow}}{\Box(a \supset \Box b) \longrightarrow a \supset b}\,\rightarrow\supset}{\Box(a \supset \Box b) \longrightarrow \Box(a \supset b)}\,\rightarrow\Box}{\Box(a \supset \Box b) \longrightarrow \Box\Box(a \supset b)}\,\rightarrow\Box$$

Figure 2.3: Example theorem 1 in a structural system.

$$\dfrac{\dfrac{\dfrac{\dfrac{a \longrightarrow a \qquad \dfrac{\dfrac{b \longrightarrow b}{\Box b \longrightarrow b}\,\Box\rightarrow}{\Box b \longrightarrow \Box b}\,\rightarrow\Box}{a, a \supset \Box b \longrightarrow \Box b}\,\supset\rightarrow}{a, \Box(a \supset \Box b) \longrightarrow \Box b}\,\boxed{\Box\rightarrow}}{\Box(a \supset \Box b) \longrightarrow a \supset \Box b}\,\rightarrow\supset}{\Box(a \supset \Box b) \longrightarrow \Box(a \supset \Box b)}\,\rightarrow\Box$$

Figure 2.4: Example theorem 2 in a structural system.

the proof to be completed. All three proofs rely on an application of $(\supset\rightarrow)$ whose left branch consists of the axiom link $a \longrightarrow a$. This $(\supset\rightarrow)$ application must be performed in the scope in which a is introduced. On the one hand, the rule cannot be used before a is assumed—and thus before the nested scope is introduced from the formula to be proved. On the other, this assumption, once made, is contingent: it can be used as an assumption only in the scope in which it is introduced, and will not pass the filtering of higher $(\rightarrow \Box)$ rules.

As a final example of the sequent calculus of Figure 2.2, we look at its modular treatment of modal disjunction. We consider the theory Γ specified in (23).

$$\dfrac{\dfrac{\dfrac{a \longrightarrow a \qquad \dfrac{\dfrac{\dfrac{b \longrightarrow b}{\Box b \longrightarrow b}\,\Box\rightarrow}{\Box b \longrightarrow \Box b}\,\rightarrow\Box}{\Box b \longrightarrow \Box\Box b}\,\rightarrow\Box}{a, a \supset \Box b \longrightarrow \Box\Box b}\,\supset\rightarrow}{a, \Box(a \supset \Box b) \longrightarrow \Box\Box b}\,\boxed{\Box\rightarrow}}{\Box(a \supset \Box b) \longrightarrow a \supset \Box\Box b}\,\rightarrow\supset$$

Figure 2.5: Example theorem 3 in a structural system.

$$\boxed{1} = \boxed{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, a \longrightarrow a \qquad \Gamma, c \longrightarrow c}{\Gamma, a, a \supset c \longrightarrow c}\supset\rightarrow}{\Gamma, a \longrightarrow c}\Box\rightarrow \qquad \dfrac{\dfrac{\Gamma, b \longrightarrow b \qquad \Gamma, c \longrightarrow c}{\Gamma, b, b \supset c \longrightarrow c}\supset\rightarrow}{\Gamma, b \longrightarrow c}\Box\rightarrow}{\Gamma, a \vee b \longrightarrow c}\vee\rightarrow}{\Gamma \longrightarrow c}\Box\rightarrow}{\Gamma \longrightarrow \Box c}\rightarrow\Box}$$

$$\boxed{2} = \boxed{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, d \longrightarrow d \qquad \Gamma, f \longrightarrow f}{\Gamma, d, d \supset f \longrightarrow f}\supset\rightarrow}{\Gamma, d \longrightarrow f}\Box\rightarrow \qquad \dfrac{\dfrac{\Gamma, e \longrightarrow e \qquad \Gamma, f \longrightarrow f}{\Gamma, e, e \supset f \longrightarrow f}\supset\rightarrow}{\Gamma, e \longrightarrow f}\Box\rightarrow}{\Gamma, d \vee e \longrightarrow f}\vee\rightarrow}{\Gamma \longrightarrow f}\Box\rightarrow}{\Gamma \longrightarrow \Box f}\rightarrow\Box}$$

$$\dfrac{\boxed{1} \qquad \boxed{2}}{\Gamma \longrightarrow \Box c \wedge \Box f}\rightarrow\wedge$$

Figure 2.6: Modular disjunction in the structurally scoped sequent calculus

(23) $\Gamma \equiv \Box(a \vee b), \Box(a \supset c), \Box(b \supset c), \Box(d \vee e), \Box(d \supset f), \Box(e \supset f)$

Γ is a modular variant of a disjunctive database presented in [Loveland, 1991]; it offers the simplest nontrivial illustration of the possible independence of case analysis. The theory establishes two modular disjunctions, $\Box(a \vee b)$ and $\Box(d \vee e)$. From these modular disjunctions, modular results $\Box c$ and $\Box f$ follow.

The proof of $\Box c \wedge \Box f$ is presented in Figure 2.6 and opens the door to an explanation of how the sequent calculus enforces the modularity of disjunction by syntactic manipulations on sets of formulas. The explanation goes as follows. Consider the placement of the $(\Box \rightarrow)$ rule that applies to the assumption of $\Box(a \vee b)$. This inference must occur higher in the proof than the $(\rightarrow \Box)$ inference that applies to $\Box c$. Otherwise, any assumptions of $a \vee b$, a and b would eventually have to be discarded, and no progress toward the proof of c would have been made. For similar reasons, the $(\Box \rightarrow)$ rule that applies to the assumption of $\Box(d \vee e)$ must occur higher than the $(\rightarrow \Box)$ inference that applies to $\Box f$. But the two $(\rightarrow \Box)$ inferences must apply in different branches of the proof—this is a consequence of the $(\rightarrow \wedge)$ inference figure. This means that the two disjunctions must also occur in different branches of the proof tree. The arguments-by-cases in this proof are thus forced to be independent. Thus, in exhibiting this independence, the proof in Figure 2.6 is not biased but in fact illustrates the only strategy by which a proof of this theorem could be constructed.

The informal discussion of these examples has used the structural discipline of scope realized in the sequent calculus of Figure 2.2 to reason concretely about the possibilities and limits of modular inference. And indeed this calculus can be useful for logic programming, where it offers a starting point for specialized inference algorithms for restricted fragments of the logic; it underlies [Miller, 1989; Giordano and Martelli, 1994]. However, this calculus turns out to be inappropriate for computational use in a general setting. For more general inference problems, the significance of the relative positions of rules in a proof introduces a problematic redundancy into the search space that is not found in classical logic.

In classical sequent calculi, rules can be freely interchanged, as long as the structure of formulas is respected and quantifier rules continue to introduce new variables as necessary [Kleene, 1951]. Exploiting this property in search is a key feature of classical theorem provers. For example, resolution [Robinson, 1965b] and matrix [Andrews, 1981; Bibel, 1982] theorem-proving methods can be seen as optimizations of sequent calculi which bypass the redundancy of considering different orderings of rules separately. In contrast, the sequent calculus of Figure 2.2 does not allow this optimization, because of the qualitative difference of different orderings of rules.

In practice (especially for first-order problems), the redundancy of rule-ordering introduces more than combinatorial difficulties—as if combinatorial difficulties aren't bad enough! Automated deduction engines must build sequent proofs from the root up, but can only determine whether a move is helpful by matching atomic formulas at leaves. Since rules must be introduced in the right order—at the right time in construction of the proof—automated methods must be prepared to apply a rule before they know whether the application will even be needed. Thus, the regime for imposing scope on proofs means that proofs can no longer be constructed in a goal-directed manner.

2.3.3 Semantic Translation and Resolution

If departures from classical logic are so problematic, perhaps we must completely assimilate modal theorem-proving to classical theorem-proving. This is the idea behind semantic translation. Kripke semantics puts any modal formula in correspondence with a classical logic formula which describes a modal model explicitly. Translating the modal formula into this corresponding classical one permits the use of ordinary classical reasoning methods, particularly resolution [Robinson, 1965b].

The basic strategy is known as the reified method for modal deduction. This method translates modal formulas using primitive terms for possible worlds and ordinary formulas that explicitly state links of accessibility between worlds. After this translation, constraints of accessibility are treated at the same time and by the same mechanisms as the ordinary first-order features of a modal specification. [Moore, 1985a; Jackson and Reichgelt, 1987] represent early computational work on the reified approach to modal deduction. A caveat about this technique is in order. Although all the axioms dealt with in this dissertation can be modeled by first-order conditions on accessibility—a fact which is assumed henceforth—

not all modal frame axioms do correspond to first-order constraints on models; see [van Benthem, 1983; van Benthem, 1984]. The basic reified approach will not work in such cases, although refinements might (particularly those based on equational reasoning) [Ohlbach, 1993].

More refined strategies perform modal reasoning by combining an inference method for classical logic with special-purpose techniques for handling accessibility. One example of such a hybrid system is developed by [Frisch and Scherl, 1991] in the context of the substitutional framework for constrained resolution [Frisch, 1991]. In this system, ordinary first-order clauses are paired with constraints that indicate accessibility relations among worlds. As resolution combines clauses, a special module ensures that the associated set of constraints are consistent with an appropriate logical theory of accessibility.

A similar alternative is to use compound terms for worlds instead of first-order statements to encode accessibility. Instead of naming a world atomically, each term specifies the sequence of transitions that is needed to reach that world from a designated starting point (the real world). This FUNCTIONAL TRANSLATION goes back to modal proof-theory of [Fitting, 1972; Smullyan, 1973] and is explored in [Wallen, 1990; Ohlbach, 1991; Auffray and Enjalbert, 1992]. Like the constraint approach, functional translation allows a specialized module to deal with relations between worlds in parallel with resolution: now it is equational unification, rather than constraint propagation.

All of these methods are designed to allow modal theorem proving to work using resolution, or related techniques like the connection or matrix method [Andrews, 1981; Bibel, 1982]. By nature, these methods are designed to allow information to flow freely around a proof. They can match any positive occurrence of an atomic formula against a negative occurrence of the same atomic formula no matter where the two literals derive from in the original statement of the theorem-proving problem. Because of this, it is very difficult to get resolution to enforce modularity—restrictions on how formulas can be used—in a predictable and powerful way during proof search. Improvements on the basic form of resolution, including constraint or equational methods, are significant and successful largely because they allow systems to use modularity more directly to prune search. Nevertheless, not even the improvements provide as strong a realization of modularity as the structurally-scoped sequent calculi.

To appreciate these points requires a careful understanding of resolution proof, as it is applied in reified deduction. Generally, resolution proof is a technique for showing that a set of statements is inconsistent; to show that a goal follows from a set of assumptions, resolution can derive a contradiction from the assumptions together with the negation of the goal.

The process of reified resolution theorem-proving begins by transforming the input statements into a canonical form as a set or conjunction of CLAUSES, where a clause is a disjunction of LITERALS and a literal is an atomic formula or its negation. Any classical statement is equivalent to a conjunction of clauses—thus using the Kripke semantics (and first-order frame properties) any modal statement is too. Resolution then repeatedly derives new clauses by combining input clauses together according to a simple operation. If this process derives the empty clause (the disjunction of no literals), the statements are inconsistent.

A modal statement is transformed into a corresponding set of clauses in several steps. The statement is first replaced by its semantics, which is then converted to prenex form, so that all quantifiers are initial. The existential quantifiers are replaced by Skolem functions and the universal quantifiers are replaced by free variables. This gives a propositional statement with free variables. This propositional statement is then converted to clausal form using de Morgan's laws and the distribution of disjunction over conjunction.

For example, take $\Box(a \supset \Box b)$ in S4. Using a Kripke semantics where s_0 names the initial world of evaluation, this formula is true in a model exactly when (24) is.

(24) $\forall w_1(R(s_0, w_1) \supset a(w_1) \supset \forall w_2(R(w_1, w_2) \supset b(w_2)))$

After quantifier elimination, (24) simplifies to (25).

(25) $R(s_0, w_1) \supset a(w_1) \supset R(w_1, w_2) \supset b(w_2)$

Its clausal form is given in (26).

(26) $\neg R(s_0, w_1) \vee \neg a(w_1) \vee \neg R(w_1, w_2) \vee b(w_2)$

A goal for proof is negated and then added as an additional set of clausal assumptions. Thus, to test whether $\Box(a \supset \Box b)$ entails $\Box\Box(a \supset b)$ from $\Box(a \supset \Box b)$, we must add to (26) the negation of $\Box\Box(a \supset b)$—translated to its four clauses in (27) using Skolem constants s_1 and s_2—and attempt to derive a contradiction.

(27) $R(s_0, s_1), R(s_1, s_2), a(s_2), \neg b(s_2)$

For a particular modal logic, we must also provide clausal premises that specify any constraints on the accessibility relation. For example, for S4, we add clauses for reflexivity and transitivity of the relation, as in (28).

(28) a $R(x, x)$
b $\neg R(x, y) \vee \neg R(y, z) \vee R(x, z)$

The inference operation of resolution takes two clauses and yields a new clause that is a logical consequence of its inputs. Formally, the rule goes as follows. The inputs are a clause $C_1 \vee l$ which is the disjunction of a positive literal l and a set of other literals C_1; and a clause $C_2 \vee \neg r$ which is the disjunction of a negative literal $\neg r$ and a set of other literals C_2. The two clauses are assumed to have no variables in common (variables can be renamed if necessary to meet this assumption). The literals l and r are UNIFIED if possible, to produce a substitution σ which assigns values to the variables in l and r such that $l\sigma = r\sigma$. Given σ, resolution derives the new clause $N = C_1\sigma \vee C_2\sigma$. To see why this follows, suppose $l\sigma$ is true; then $\neg l\sigma$ (which is identical to $\neg r\sigma$) is false and hence some other disjunct in

1. $\neg R(s_0, w_1) \vee \neg a(w_1) \vee \neg R(w_1, w_2) \vee b(w_2)$ [(26)]
2. $a(s_2)$ [(27)]
3. $\neg R(s_0, s_2) \vee \neg R(s_2, w_2) \vee b(w_2)$ [1,2]
4. $\neg b(s_2)$ [(27)]
5. $\neg R(s_0, s_2) \vee \neg R(s_2, s_2)$ [3,4]
6. $R(w, w)$ [(28)]
7. $\neg R(s_0, s_2)$ [5,6]
8. $\neg R(x, y) \vee \neg R(y, z) \vee R(x, z)$ [(28)]
9. $\neg R(x, y) \vee \neg R(y, s_2)$ [7,8]
10. $R(s_1, s_2)$ [(27)]
11. $\neg R(x, s_1)$ [9,10]
12. $R(s_0, s_1)$ [(27)]
13. contradiction [11,12]

Figure 2.7: A resolution proof by semantic translation of the S4 theorem $\Box(a \supset \Box b) \rightarrow \Box\Box(a \supset b)$

$C_2\sigma \vee \neg r\sigma$ must be true for this case. Otherwise, $l\sigma$ is false—and hence some other disjunct in $C_1\sigma \vee l\sigma$ must be true for this case. This establishes that some disjunct in N is always true.

Resolution theorem-proving consists in repeatedly deriving new clauses by resolution from assumptions and those clauses already derived until the empty clause is derived and a contradiction is thereby established. A sample resolution proof based on the clauses presented in (26), (27) and (28) is given in Figure 2.7.

In a proof like Figure 2.7, the scope at which a rule is used is represented by the world-term at which the rule is instantiated and the facts of accessibility linking that world-term to others. For example, here as in Figure 2.3, the rule $a \supset \Box b$ is applied in a twice-nested scope. Here that is expressed not by the position of inferences in the derivation, but by the term s_2 at which the rule is instantiated, and by the double link of accessibility that connects s_0 to s_2.

In some cases, modularity is enforced in resolution just by the names assigned to possible worlds. Our non-theorem $\Box(p \supset q) \vee \Box p$ is a good example. It translates into the semantic clauses given in (29).

(29) $R(s_0, s_1), p(s_1), \neg q(s_1), R(s_1, s_2), \neg p(s_2)$

The universal quantifiers associated with the two modal operators are translated by different Skolem constants, s_1 and s_2. Thanks to the difference, there is no substitution that unifies $p(s_1)$ and $\neg p(s_2)$. Therefore, no step of resolution will succeed in combining these facts. This respects the modularity of the modal statements.

When it comes to the use of modular assumptions, however, pure reified translation and resolution is less satisfactory. Evidence of this is available already in Figure 2.7. Up to line 5, the proof takes only the first-order part of the specification into account; the proof resolves away the occurrences of $\neg a(w_1)$ and $b(w_2)$ in (26) without even considering any relations of accessibility. Now, $\neg a(w_1)$ and $b(w_2)$ will continue be translated in terms of free variables w_1 and w_2 as long as necessity operators occur at the same positions in initial modal formula. Which necessity operator is chosen affects only which relations of accessibility link s_0, w_1 and w_2. This means that the beginning of the proof in Figure 2.7 doesn't depend on the choice of necessity operators either. So modularity might not be taken into account in restricting search at all!

Indeed, using pure resolution, no necessary statement is guaranteed to be used only in a modular way. The application of any statement to the current problem is contingent on showing the relationship between sets of worlds. This is one goal among many in resolution. The resolution search strategy doesn't say anything about when to attack such goals. Thus goals of relatedness might always be delayed. But as long as they are delayed, modal operators don't constrain the search in any way over the corresponding classical case.

Constraint and equational approaches to modal deduction are ways to get some of the modularity back. The constraint approach, developed in [Frisch and Scherl, 1991] as an extension of [Frisch, 1991], is perhaps the easier to understand. In this approach, each statement takes the form of a constrained clause C/R. C is an ordinary clause describing first-order information, while R is a constraint that specifies a conjunction of accessibility restrictions on the free world variables in C. Such constraints are governed by the usual theory of accessibility in the modal language, and, if necessary, a further specification of what accessibility constraints hold of Skolem constants that name worlds.

Statements are combined by a hybrid reasoning system. The first-order clauses are resolved using ordinary resolution, and an updated constraint is computed simultaneously. This computation must consistently reduce relatedness constraints among constants or Skolem-functions to relatedness constraints that follow from the basic theory; if the constraints become unsatisfiable the new statement is rejected. The remaining constraints of relatedness on free world variables are associated with the new statement. This processing strategy means that constraints on the accessibility relation are taken into account as soon as possible. This helps to guarantee that information is used in a modular way.

For example, the statement corresponding to $\Box(a \supset \Box b)$ on the constraint approach is (30).

(30) $\neg a(w_1) \vee b(w_2)/R(s_0, w_1), R(w_1, w_2)$

When we combine (30) with $a(s_2)$ by resolution, we get the first-order clause $b(w_2)$. At the same time, we are obliged to meet the constraint $R(s_0, s_2)$ which now relates ground terms. In this case, we can meet this constraint by using the transitivity of accessibility to link s_0 to s_2 through s_1. In alternative circumstances, if the relation $R(s_0, s_2)$ could not be immediately established, this step of constrained resolution would be impossible. Thus, the discipline of checking constraints is a tool that permits inferential combinations of facts that violate modularity to be detected and rejected as early as possible.

The resulting constrained clause is given in (31).

(31) $b(w_2)/R(s_3, w_2)$

Just a single further modular step of constrained resolution is required to complete the hybrid proof.

Where the approach of [Frisch and Scherl, 1991] encodes accessibility by distinguished constraints on terms, equational approaches encode accessibility by giving a special structure to terms themselves. The technique goes back to Fitting's use of prefixes [Fitting, 1972], and has since been considerably refined [Smullyan, 1973; Fitting, 1983; Wallen, 1990; Ohlbach, 1991; Auffray and Enjalbert, 1992].

From a semantic point of view, the heart of the equational technique is a refinement of the usual definition of Kripke models. This alternative takes paths between possible worlds as primitive [Ohlbach, 1991; Ohlbach, 1993]. Instead of using relations R_i directly, models are constructed in terms of a set of partial functions AF_i such that $R_i(u, v)$ if and only if $\exists f \in AF_i . v = f(u)$. This change works its way into the truth-conditions for modal formulas: $\Box_i A$ is true at w just in case A is true at $f(w)$ for all $f \in AF_i$ with $f(w)$ defined.

The new model theory induces a change in the translation of modal formulas into classical logic. In the translation, worlds are named by complex terms that give the sequence of transitions required to reach them from the real world s_0. For example, the term $\alpha\beta\gamma$ names the world reached by starting at s_0, following the transition α, then following the transition β and finally following the transition γ.

The usual axioms of modal logic now correspond to equational generalizations about sets of accessibility functions. For example, to obtain sound and complete models for (VER), we introduce the constraint that AF_i include an identity transition ϵ such that $\mu\epsilon = \mu$. (The presence of this axiom also means that for these logics we can restrict attention to cases where AF_i includes only total functions.) Meanwhile, to obtain sound and complete models for (PI), we introduce the constraint that AF_i include a transition $\alpha \circ \beta$ for any pair of transitions α and β, with $\mu(\alpha \circ \beta) = \mu\alpha\beta$. (The notation $\circ$ serves as a reminder that $\alpha \circ \beta \in AF_i$.) We capture the (INC) scheme $\Box_i A \supset \Box_j A$ with the constraint that $AF_j \subseteq AF_i$.

To implement the equational approach, appropriate equations must be factored into the unification method used to resolve literals. The equational method guarantees that there is a consistent way to link the worlds mentioned in a clause along paths of accessibility where modularity of inference is eagerly enforced. Thus, the effect of these steps of equational unification is similar to the effect of constraints in constrained resolution. (In fact, [Frisch and Scherl, 1991] show that an equational unifier would count as an instance of their substitutional framework.)

We can illustrate with the example of Figure 2.7 again. $\Box(a \supset \Box b)$ is now translated and put in the clausal form of (32).

(32) $\neg a(s_0 x_1) \vee b(s_0 x_1 x_2)$

The goal $\Box\Box(a \supset b)$ is negated and put in clausal form as in (33).

(33) $a(s_0 \alpha_1 \alpha_2), \neg b(s_0 \alpha_1 \alpha_2)$

We unify the literals $a(s_0 x_1)$ and $a(s_0 \alpha_1 \alpha_2)$ by assigning $x_1 = \alpha_1 \circ \alpha_2$. The assignment exploits the equation implementing the (PI) scheme of S4; it indicates that x_1 should span the two steps of accessibility corresponding to the two modal operators in the original goal. The result of resolution is $b(s_0(\alpha_1 \circ \alpha_2)x_2)$. This resolves against $\neg b(s_0 \alpha_1 \alpha_2)$ in one further step of equational resolution, in which the variable x_2 gets the value ϵ in keeping with the need to instantiate the necessary statement $\Box b$ at the present world, and the equational interpretation of the (VER) scheme of S4.

While constraint and equational approaches to modal resolution can help to ensure that two facts are not combined together unless they specify compatible kinds of information—so that resolution proofs always maintain modular information-flow—these approaches are not sufficient to ensure that the structure of proofs is also modular. This means that the search space will include proofs that are unnecessarily large. One consequence of this is that these refinements of resolution do not factor into search the modularity of disjunction motivated in section 2.1.3 and illustrated for structurally scoped proofs in section 2.3.2.

For example, consider again the modular theory Γ introduced in (23) and repeated as (34) below.

(34) $\Box(a \vee b), \Box(a \supset c), \Box(b \supset c), \Box(d \vee e), \Box(d \supset f), \Box(e \supset f)$

$\Box c \wedge \Box f$ is a consequence of this theory. The clausal version of (34), and this query negated, are presented in Figure 2.8 in the notation of constrained resolution.

As we saw in section 2.3.2, this consequence ought to be established in a modular way. Any proof by cases that depends on a modular disjunction must be resolved at the world where the disjunction is introduced. The labeling of a resolution proof is not sufficient to ensure this, however. It's true that you can perform resolution steps in a modular way, as in Figure 2.9. We can see the modular structure of this proof at lines 5 and 12. Take line 5. Here we have a single literal in the clause $c(w_1)$, where (strictly speaking) you might expect the disjunction $c(w_1) \vee c(w_1)$. The two identical disjuncts are collapsed by an operation called FACTORING which is implemented in most resolution provers. The use of factoring here indicates that the case analysis introduced at $a(w_1) \vee b(w_1)$ is now effectively over. We are left with a single common case: $c(w_1)$. Factoring of $f(w_1)$ also occurs at line 12. This shows that the two case analyses in the proof are being resolved independently.

$$
\begin{gathered}
a(w_1) \lor b(w_1)/R(s_0, w_1) \\
\neg a(w_1) \lor c(w_1)/R(s_0, w_1) \\
\neg b(w_1) \lor c(w_1)/R(s_0, w_1) \\
d(w_1) \lor e(w_1)/R(s_0, w_1) \\
\neg d(w_1) \lor f(w_1)/R(s_0, w_1) \\
\neg e(w_1) \lor f(w_1)/R(s_0, w_1) \\
\neg c(u_1) \lor \neg f(v_1)
\end{gathered}
$$

Figure 2.8: Modal disjunctive specification for constrained resolution.

1. $a(w_1) \lor b(w_1)/R(s_0, w_1)$ [premise]
2. $\neg a(w_1) \lor c(w_1)/R(s_0, w_1)$ [premise]
3. $b(w_1) \lor c(w_1)/R(s_0, w_1)$ [1,2]
4. $\neg b(w_1) \lor c(w_1)/R(s_0, w_1)$ [premise]
5. $c(w_1)/R(s_0, w_1)$ [3,4]
6. $\neg c(u_1) \lor \neg f(v_1)$ [premise]
7. $\neg f(v_1)$ [5,6]
8. $d(w_1) \lor e(w_1)/R(s_0, w_1)$ [premise]
9. $\neg d(w_1) \lor f(w_1)/R(s_0, w_1)$ [premise]
10. $e(w_1) \lor f(w_1)/R(s_0, w_1)$ [8,9]
11. $\neg e(w_1) \lor f(w_1)/R(s_0, w_1)$ [premise]
12. $f(w_1)/R(s_0, w_1)$ [10,11]
13. contradiction [7,12]

Figure 2.9: Resolution proof that uses modal disjunction in a modular way

This separation would always be allowed in a resolution proof—whether in modal logic or classical logic. Here the separation naturally fits the meanings of the modal statements. The two disjunctions are modular, so the meanings of these statements indicate that the two ambiguities should not interact in the proof.

Although this modular force is naturally associated with the modal statements, nothing about resolution proof search mandates the independence. In fact, in resolution, the values of terms for possible worlds has no impact on the independence of disjunctions or the explicitly modular structure of a proof. For example, derivations like the one in Figure 2.10 are perfectly possible. This proof abandons modularity at line 5 because the case analysis based on whether a or b is true in world u_1 is not resolved at world u_1 itself. There is no factoring. Instead line 5 permits global case analysis—one case where b is true at world u_1 and another case where f is false at world v_1. This global case analysis causes several extra steps to be included in the proof. Overall, lines 1 through 9 of the proof handle only the global case where a is true at world u_1 and d is true at world v_1. Then lines 10 through 13 are required to handle the case where b is true at world u_1 and d is true at world v_1. At this point—when the modular proof is already completed—this proof has only dispensed with the case where d is true, leaving the cases where e is true yet to be handled. Lines 14 through 17 handle the cases where e and a are true; that leaves lines 18 through 20 to finish the proof on the cases where e and b are true.

Of course, this proof has the same meaning as the previous one. The modular use of information that is part of modal logic continues to be respected. This could not be otherwise, given the correctness of the proof method. However, modularity is not being used to constrain search in as strong a way as it could be. To impose such constraints we need to look more closely at the structure of modal proofs—and stick more closely to the syntax of modal logic itself.

2.3.4 Explicitly-scoped Sequent Calculi

The starting point for this analysis is an EXPLICITLY-SCOPED sequent calculus, such as that presented for propositional S4 in Figure 2.11. This sequent calculus is a notational variant of a tableau system presented in [Smullyan, 1973]. It can be motivated in several ways.

From one point of view, this calculus presents rules for modal proof that interleave and combine functional translation for modal formulas with the inference figures of the classical sequent calculus.

Each formula in the proof is labeled with with a STRING from a distinguished alphabet of scope variables. (A string is just a term built using an associative binary operation of concatenation with left- and right- identity ϵ. I will write annotation variables α, β, etc.; I will use μ, ν etc. to represent strings.) As in the functional translation, this string encodes the path to the world where the formula is evaluated semantically.

The inference figures reduce labeled modal formulas into components by accessing the classical translation of the main connective of the modal formula and realizing the corresponding classical inference. For example, for the connectives of ordinary first-order

1. $a(w_1) \lor b(w_1)/R(s_0, w_1)$ [premise]
2. $\neg a(w_1) \lor c(w_1)/R(s_0, w_1)$ [premise]
3. $b(w_1) \lor c(w_1)/R(s_0, w_1)$ [1,2]
4. $\neg c(u_1) \lor \neg f(v_1)$ [premise]
5. $b(u_1) \lor \neg f(v_1)$ [3,4]
6. $d(w_1) \lor e(w_1)/R(s_0, w_1)$ [premise]
7. $\neg d(w_1) \lor f(w_1)/R(s_0, w_1)$ [premise]
8. $e(w_1) \lor f(w_1)/R(s_0, w_1)$ [6,7]
9. $b(u_1) \lor e(v_1)$ [5,8]
10. $\neg b(w_1) \lor c(w_1)/R(s_0, w_1)$ [premise]
11. $c(u_1) \lor e(v_1)$ [9,10]
12. $\neg c(u_1) \lor e(v_1)$ [4,8]
13. $e(v_1)$ [11,12]
14. $\neg e(w_1) \lor f(w_1)/R(s_0, w_1)$ [premise]
15. $f(v_1)$ [13,14]
16. $\neg c(u_1)$ [4,15]
17. $\neg a(u_1)$ [2,16]
18. $b(u_1)$ [1,17]
19. $c(u_1)$ [10,18]
20. contradiction [16,19]

Figure 2.10: Resolution proof with modal disjunction but without modular structure.

$$\frac{\mu = \nu}{\Gamma, A^{\mu} \longrightarrow A^{\nu}, \Delta}\,\text{initial}$$

$$\frac{\Gamma, A \wedge B^{\mu}, A^{\mu}, B^{\mu} \longrightarrow \Delta}{\Gamma, A \wedge B^{\mu} \longrightarrow \Delta}\wedge\rightarrow$$

$$\frac{\Gamma \longrightarrow A \wedge B^{\mu}, A^{\mu}, \Delta \qquad \Gamma \longrightarrow A \wedge B^{\mu}, B^{\mu}, \Delta}{\Gamma \longrightarrow A \wedge B^{\mu}, \Delta}\rightarrow\wedge$$

$$\frac{\Gamma, A \vee B^{\mu}, A^{\mu} \longrightarrow \Delta \qquad \Gamma, A \vee B^{\mu}, B^{\mu} \longrightarrow \Delta}{\Gamma, A \vee B^{\mu} \longrightarrow \Delta}\vee\rightarrow$$

$$\frac{\Gamma \longrightarrow A \vee B^{\mu}, A^{\mu}, B^{\mu}, \Delta}{\Gamma \longrightarrow A \vee B^{\mu}, \Delta}\rightarrow\vee$$

$$\frac{\Gamma, A \supset B^{\mu} \longrightarrow A^{\mu}, \Delta \qquad \Gamma, A \supset B^{\mu}, B^{\mu} \longrightarrow \Delta}{\Gamma, A \supset B^{\mu} \longrightarrow \Delta}\supset\rightarrow$$

$$\frac{\Gamma, A^{\mu} \longrightarrow A \supset B^{\mu}, B^{\mu}, \Delta}{\Gamma \longrightarrow A \supset B^{\mu}, \Delta}\rightarrow\supset$$

$$\frac{\Gamma, \neg A^{\mu} \longrightarrow A^{\mu}, \Delta}{\Gamma, \neg A^{\mu} \longrightarrow \Delta}\neg\rightarrow$$

$$\frac{\Gamma, A^{\mu} \longrightarrow \neg A^{\mu}, \Delta}{\Gamma \longrightarrow \neg A^{\mu}, \Delta}\rightarrow\neg$$

$$\frac{\Gamma, \Box A^{\mu}, A^{\mu\sigma} \longrightarrow \Delta}{\Gamma, \Box A^{\mu} \longrightarrow \Delta}\Box\rightarrow$$

$$\frac{\Gamma \longrightarrow \Box A^{\mu}, A^{\mu\alpha}, \Delta}{\Gamma \longrightarrow \Box A^{\mu}, \Delta}\rightarrow\Box^{\dagger}$$

$$\frac{\Gamma \longrightarrow \Diamond A^{\mu}, A^{\mu\sigma}, \Delta}{\Gamma \longrightarrow \Diamond A^{\mu}, \Delta}\rightarrow\Diamond$$

$$\frac{\Gamma, \Diamond A^{\mu}, A^{\mu\alpha} \longrightarrow \Delta}{\Gamma, \Diamond A^{\mu} \longrightarrow \Delta}\Diamond\rightarrow^{\dagger}$$

Figure 2.11: Path-based, explicitly-scoped, cut-free sequent calculus for propositional S4 modal logic. The initial rule specifies a derivation as long as μ and ν are equal strings. † For $(\rightarrow \Box)$, $(\Diamond \rightarrow)$, α must not appear in the conclusion sequent.

logic, the inference figures preserve the labels of principal formulas just as the modal semantics preserves the world of evaluation. (Note that the inference figures of the calculus are presented with the same conventions of composition and structure as the structurally-scoped calculus of Figure 2.2. In particular, the inference figures here also carry over the principal formula from the end-sequent to higher sequents to avoid a contraction rule—and again I will generally suppress this for clarity.)

Meanwhile, the $(\rightarrow \Box)$ and $(\Diamond \rightarrow)$ rules create a NEW nested scope by appending a new variable (representing an arbitrary transition from one possible world to another) to the label of the side formula. Conversely, the $(\Box \rightarrow)$ and $(\rightarrow \Diamond)$ rules allow the path to the world of evaluation to be extended by one transition by instantiation. Thus, these figures implement the inference for the quantifier in the modal truth conditions. Finally, the initial rule in this system requires the labels of formulas to match, as well as the formulas themselves. Thus, an atomic result can be inferred at a given world only in virtue of an assumption introduced at that world by some lower modal rule. I write this using an auxiliary judgment $\mu = \nu$ because in a computational implementation of the calculus, this check requires work; μ and ν may have different representations. Accordingly, the leaves of a derivation are statements $\mu = \nu$ where μ and ν are equal as strings. From such statements, derivations are constructed according to the rules of the calculus as usual.

Because of its explicit scoping, this new system is somewhat more expressive than the structurally-scoped modal proof system. The correspondence between them is stated as follows: there is a derivation with end-sequent $\Gamma \longrightarrow \Delta$ in the structurally-scoped sequent calculus of Figure 2.2 if and only if there is a derivation with end-sequent $\Gamma \longrightarrow \Delta$ in the explicitly-scoped sequent calculus (i.e., every formula in antecedent and succedent of the sequent is labeled with ϵ). This result is most directly established by showing that the explicitly-scoped calculus is also sound and complete for the usual semantics of modal logic [Smullyan, 1973].

From another point of view, this calculus represents merely a syntactic elaboration of the structurally-scoped sequent calculus presented in section 2.3.2. The new calculus simply rewrites the rules of the old calculus in a notation that gives inferences the same interpretation no matter where those rules appear in the proof. We achieve this by labeling each formula *A* in a proof with a distinguished term μ that represents the SCOPE of the formula. The term LISTS the sequence of scope-changing inferences—$(\rightarrow \Box)$ and $(\Diamond \rightarrow)$—that should apply in a structurally-scoped proof from the root to the inference that introduces this occurrence of *A*. The scope of each rule application follows from the labels of its principal and side formulas.

Thus, the use of an explicitly-scoped calculus need not be regarded as a semantic method, despite the apparent similarity. [Stone, to appear] considers intuitionistic logic, where the proofs of a structurally-scoped sequent calculus derive independent interest because of their interpretation as programs [Howard, 1980], and shows that an explicitly-scoped sequent calculus describes exactly the same proofs as the structurally-scoped system. By this result (which is stronger than mere equivalence of provability or semantics), that explicitly-

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha\beta = \alpha\beta}{a^{\alpha\beta} \longrightarrow a^{\alpha\beta}}\,\text{initial} \qquad \dfrac{\dfrac{\alpha\beta = \alpha\beta}{a^{\alpha\beta}, b^{\alpha\beta} \longrightarrow b^{\alpha\beta}}\,\text{initial}}{a^{\alpha\beta}, \Box b^{\alpha\beta} \longrightarrow b^{\alpha\beta}}\,\to\Box}{a^{\alpha\beta}, a \supset \Box b^{\alpha\beta} \longrightarrow b^{\alpha\beta}}\,\supset\!\to}{a^{\alpha\beta}, \Box(a \supset \Box b) \longrightarrow b^{\alpha\beta}}\,\boxed{\Box\to}}{\Box(a \supset \Box b) \longrightarrow a \supset b^{\alpha\beta}}\,\to\!\supset}{\Box(a \supset \Box b) \longrightarrow \Box(a \supset b)^{\alpha}}\,\to\Box}{\Box(a \supset \Box b) \longrightarrow \Box\Box(a \supset b)}\,\to\Box$$

Figure 2.12: Example theorem 1 in an explicitly-scoped system.

scoped calculus can be considered a purely proof-theoretic optimization. Further, as in this dissertation, the explicitly-scoped calculus can be studied fruitfully as a proof-theoretic object in its own right (see [Schmidt, 1996] for another example).

Examples of this proof system show that it falls between the structurally-scoped calculus of section 2.3.2 and the resolution approaches discussed in section 2.3.3. The imposition of modularity in failing to prove $\Box(p \supset q) \lor \Box p$ is similar to the resolution calculus:

(35)
$$\dfrac{\dfrac{\dfrac{\dfrac{p^{\alpha} \xrightarrow{!} q^{\alpha}, p^{\beta}}{\longrightarrow p \supset q^{\alpha}, p^{\beta}}\,\to\!\supset}{\longrightarrow p \supset q^{\alpha}, \Box p}\,\to\Box}{\longrightarrow \Box(p \supset q), \Box p}\,\to\Box}{\longrightarrow \Box(p \supset q) \lor \Box p}\,\to\lor$$

Again, the problem is a mismatch due to the unequal path labelings of atomic formulas.

Like the structurally-scoped proof system, however, the explicitly-scoped system captures modularity in the structure of its proofs. Consider again the three theorems proved in Figures 2.3, 2.4 and 2.5. Proofs identical in structure to those presented earlier can be worked out in the explicitly-scoped calculus, by adding appropriate labels to formulas throughout the proofs. Such proofs are presented in Figures 2.12, 2.13 and 2.14. Note how the labels encode the scopes of the different $(\Box \to)$ applications. In Figure 2.12, the side formula of the lower $(\Box \to)$ gets $\alpha\beta$, indicating the double nesting; likewise, in Figure 2.13, it gets α; and in Figure 2.14, the empty string.

In the explicitly-scoped system, the proof can still be constructed if the $(\Box \to)$ rules are permuted HIGHER. The assumption of *a*, in whatever scope, remains available on the left of the sequent until the leaves of the proof tree. This contrasts with the structurally-scoped system, where the assumption of *a* must be discarded above any inference that effects a change in scope.

However, as with the structurally-scoped system, these $(\Box \to)$ rules cannot be permuted down across the remaining $(\to \Box)$ rules. Otherwise, they would violate the eigenvariable condition that says that when a scope variable is introduced by a $(\to \Box)$ rule, it cannot appear anywhere (else) in the sequent.

$$\dfrac{\dfrac{\dfrac{\alpha\beta}{\alpha = \alpha}}{a^{\alpha} \longrightarrow a^{\alpha}}\;\text{initial} \qquad \dfrac{\dfrac{\dfrac{\alpha\beta = \alpha\beta}{a^{\alpha}, b^{\alpha\beta} \longrightarrow b^{\alpha\beta}}\;\text{initial}}{a^{\alpha}, \Box b^{\alpha} \longrightarrow b^{\alpha\beta}}\;\Box\rightarrow}{a^{\alpha}, \Box b^{\alpha} \longrightarrow \Box b^{\alpha}}\;\rightarrow\Box}{\dfrac{a^{\alpha}, a \supset \Box b^{\alpha} \longrightarrow \Box b^{\alpha}}{\dfrac{a^{\alpha}, \Box(a \supset \Box b) \longrightarrow \Box b^{\alpha}}{\dfrac{\Box(a \supset \Box b) \longrightarrow a \supset \Box b^{\alpha}}{\Box(a \supset \Box b) \longrightarrow \Box(a \supset \Box b)}\;\rightarrow\Box}\;\rightarrow\supset}\;\boxed{\Box\rightarrow}}\;\supset\rightarrow$$

Figure 2.13: Example theorem 2 in an explicitly-scoped system.

$$\dfrac{\dfrac{\epsilon = \epsilon}{a \longrightarrow a}\;\text{initial} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\alpha\beta = \alpha\beta}{b^{\alpha\beta} \longrightarrow b^{\alpha\beta}}\;\text{initial}}{a, \Box b \longrightarrow b^{\alpha\beta}}\;\Box\rightarrow}{a, \Box b \longrightarrow \Box b^{\alpha}}\;\rightarrow\Box}{a, \Box b \longrightarrow \Box\Box b}\;\rightarrow\Box}{\dfrac{a, a \supset \Box b \longrightarrow \Box\Box b}{\dfrac{a, \Box(a \supset \Box b) \longrightarrow \Box\Box b}{\Box(a \supset \Box b) \longrightarrow a \supset \Box\Box b}\;\rightarrow\supset}\;\boxed{\Box\rightarrow}}\;\supset\rightarrow$$

Figure 2.14: Example theorem 3 in an explicitly-scoped system.

Downward impermutability also means that modal disjunctions induce a modular structure in proofs of this explicitly-scoped calculus. The modular proof of Figure 2.6 can be carried over into the explicitly-scoped systems by labeling the scopes of inferences, just as in the proofs of Figures 2.12, 2.13 and 2.14. For reference, such a proof is shown in Figure 2.15. Observe that the modal disjunction $\Box(a \vee b)$ is instantiated along the path α introduced by the query $\Box c$. The eigenvariable condition therefore ensures that no inference can usefully apply to this disjunction below the $(\rightarrow \Box)$ inference where α is introduced. Analogously, no inference can usefully apply to the disjunction $\Box(d \vee e)$ below the $(\rightarrow \Box)$ inference where β is introduced. Thus, here as before, the two disjunctions are irrevocably confined to different components of the proof, and cannot interact.

In allowing instantiations of necessary information to be permuted upward but not downward in a proof, the explicitly-scoped sequent calculus of Figure 2.11 goes only halfway toward eliminating the redundancies of search that encumber the structurally-scoped sequent calculus. The explicitly-scoped calculus is an important intermediary, however, because it still retains all the properties of modularity that we might want in a modal proof system—unlike the resolution-based translation approaches described in section 2.3.3 or even the unification-based variant of the explicitly-scoped sequent calculus we introduce next in section 2.3.5. As a result, the basic explicitly-scoped calculus serves as

$$\boxed{1} = \boxed{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, a^{\alpha} \longrightarrow a^{\alpha} \qquad \Gamma, c^{\alpha} \longrightarrow c^{\alpha}}{\Gamma, a^{\alpha}, a \supset c^{\alpha} \longrightarrow c^{\alpha}}\supset\rightarrow}{\Gamma, a^{\alpha} \longrightarrow c^{\alpha}}\Box\rightarrow \qquad \dfrac{\dfrac{\Gamma, b^{\alpha} \longrightarrow b^{\alpha} \qquad \Gamma, c^{\alpha} \longrightarrow c^{\alpha}}{\Gamma, b^{\alpha}, b \supset c^{\alpha} \longrightarrow c^{\alpha}}\supset\rightarrow}{\Gamma, b^{\alpha} \longrightarrow c^{\alpha}}\Box\rightarrow}{\Gamma, a \vee b^{\alpha} \longrightarrow c^{\alpha}}\vee\rightarrow}{\Gamma \longrightarrow c^{\alpha}}\Box\rightarrow}{\Gamma \longrightarrow \Box c}\rightarrow\Box}$$

$$\boxed{2} = \boxed{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, d^{\beta} \longrightarrow d^{\beta} \qquad \Gamma, f^{\beta} \longrightarrow f^{\beta}}{\Gamma, d^{\beta}, d \supset f^{\beta} \longrightarrow f^{\beta}}\supset\rightarrow}{\Gamma, d^{\beta} \longrightarrow f^{\beta}}\Box\rightarrow \qquad \dfrac{\dfrac{\Gamma, e^{\beta} \longrightarrow e^{\beta} \qquad \Gamma, f^{\beta} \longrightarrow f^{\beta}}{\Gamma, e^{\beta}, e \supset f^{\beta} \longrightarrow f^{\beta}}\supset\rightarrow}{\Gamma, e^{\beta} \longrightarrow f^{\beta}}\Box\rightarrow}{\Gamma, d \vee e^{\beta} \longrightarrow f^{\beta}}\vee\rightarrow}{\Gamma \longrightarrow f^{\beta}}\Box\rightarrow}{\Gamma \longrightarrow \Box f}\rightarrow\Box}$$

$$\dfrac{\boxed{1} \qquad \boxed{2}}{\Gamma \longrightarrow \Box c \wedge \Box f}\rightarrow\wedge$$

Figure 2.15: Modular disjunction in the explicitly scoped sequent calculus

a benchmark for assessing the properties of proofs and proof-systems that allow modularity of proof structure to be exploited during proof search, in Chapter 3 and, to some extent, in Chapter 4. Indeed, the unification-based calculus that we consider next provides a useful alternative to resolution-based inference methods principally because the flexible proofs it allows can be compared and transformed more easily into simpler explicitly-scoped proofs with guaranteed modular structure.

2.3.5 A Lifted System

Using general proof-theoretic techniques (as in e.g. [Lincoln and Shankar, 1994]), the explicitly-scoped sequent calculus can be lifted to use unification. The use of unification streamlines search in two ways. First, the choice of instantiated terms is delayed until formulas containing them appear as axioms. This is of course when information becomes available about which values might be useful. Second, requirements for variables to be new are replaced by the use of Skolem terms. From a proof-theoretic point of view, Skolem terms are purely syntactic devices. Any value that would have to appear on the sequent where a variable was introduced—taking into account possible permutations—is a subterm of its corresponding Skolem term. By ruling out circular terms by an occur-check in unification, we ensure that a variable can be chosen in place of the Skolem term and the proof reordered so that the variable is new. This eliminates the remaining impermutabilities of the calculus—and, unfortunately, at the same time frees proofs from any structural constraints of modularity.

$$\frac{}{C/C, A = B, \mu = \nu \triangleright \Gamma, A^{\mu}_{X} \longrightarrow B^{\nu}_{X}, \Delta}\ \text{initial}$$

$$\frac{C/C' \triangleright \Gamma, (A \wedge B)^{\mu}_{X}, A^{\mu}_{X}, B^{\mu}_{X} \longrightarrow \Delta}{C/C' \triangleright \Gamma, (A \wedge B)^{\mu}_{X} \longrightarrow \Delta}\ \wedge \rightarrow$$

$$\frac{C'/C'' \triangleright \Gamma \longrightarrow (A \wedge B)^{\mu}_{X}, A^{\mu}_{X}, \Delta \qquad C/C' \triangleright \Gamma \longrightarrow (A \wedge B)^{\mu}_{X}, B^{\mu}_{X}, \Delta}{C/C'' \triangleright \Gamma \longrightarrow (A \wedge B)^{\mu}_{X}, \Delta}\ \rightarrow \wedge$$

$$\frac{C'/C'' \triangleright \Gamma, (A \vee B)^{\mu}_{X}, A^{\mu}_{X} \longrightarrow \Delta \qquad C/C' \triangleright \Gamma, (A \vee B)^{\mu}_{X}, B^{\mu}_{X} \longrightarrow \Delta}{C/C'' \triangleright \Gamma, (A \vee B)^{\mu}_{X} \longrightarrow \Delta}\ \vee \rightarrow$$

$$\frac{C/C' \triangleright \Gamma \longrightarrow (A \vee B)^{\mu}_{X}, A^{\mu}_{X}, B^{\mu}_{X}, \Delta}{C/C' \triangleright \Gamma \longrightarrow (A \vee B)^{\mu}_{X}, \Delta}\ \rightarrow \vee$$

$$\frac{C'/C'' \triangleright \Gamma, (A \supset B)^{\mu}_{X} \longrightarrow A^{\mu}_{X}, \Delta \qquad C/C' \triangleright \Gamma, (A \supset B)^{\mu}_{X}, B^{\mu}_{X} \longrightarrow \Delta}{C/C'' \triangleright \Gamma, (A \supset B)^{\mu}_{X} \longrightarrow \Delta}\ \supset\rightarrow$$

$$\frac{C/C' \triangleright \Gamma, A^{\mu}_{X} \longrightarrow (A \supset B)^{\mu}_{X}, B^{\mu}_{X}, \Delta}{C/C' \triangleright \Gamma \longrightarrow (A \supset B)^{\mu}_{X}, \Delta}\ \rightarrow\supset$$

$$\frac{C'/C'' \triangleright \Gamma, (\neg A)^{\mu}_{X} \longrightarrow A^{\mu}_{X}, \Delta}{C'/C'' \triangleright \Gamma, (\neg A)^{\mu}_{X} \longrightarrow \Delta}\ \neg \rightarrow$$

$$\frac{C'/C'' \triangleright \Gamma, A^{\mu}_{X} \longrightarrow (\neg A)^{\mu}_{X}, \Delta}{C'/C'' \triangleright \Gamma \longrightarrow (\neg A)^{\mu}_{X}, \Delta}\ \rightarrow \neg$$

$$\frac{C/C' \triangleright \Gamma, (\Box A)^{\mu}_{X}, A^{\mu x}_{X,x} \longrightarrow \Delta}{C/C' \triangleright \Gamma, (\Box A)^{\mu}_{X} \longrightarrow \Delta}\ \Box \rightarrow^{\dagger}$$

$$\frac{C/C' \triangleright \Gamma \longrightarrow (\Box_{i\alpha} A)^{\mu}_{X}, A^{\mu\alpha(X)}_{X}, \Delta}{C/C' \triangleright \Gamma \longrightarrow (\Box_{i\alpha} A)^{\mu}_{X}, \Delta}\ \rightarrow \Box$$

$$\frac{C/C' \triangleright \Gamma \longrightarrow (\Diamond_{i} A)^{\mu}_{X}, A^{\mu x}_{X,x}, \Delta}{C/C' \triangleright \Gamma \longrightarrow (\Diamond_{i} A)^{\mu}_{X}, \Delta}\ \rightarrow \Diamond^{\dagger}$$

$$\frac{C/C' \triangleright \Gamma, (\Diamond_{i\alpha} A)^{\mu}_{X}, A^{\mu\alpha(X)}_{X} \longrightarrow \Delta}{C/C' \triangleright \Gamma, (\Diamond_{i\alpha} A)^{\mu}_{X} \longrightarrow \Delta}\ \Diamond \rightarrow$$

Figure 2.16: Lifted path-based, explicitly-scoped, cut-free sequent calculus for propositional S4 modal logic † The variables u and x may not appear in Σ.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{xy=\alpha\beta/xy=\alpha\beta,\alpha\beta=x\triangleright a^{\alpha\beta}\longrightarrow a^{x}\qquad \dfrac{/xy=\alpha\beta\triangleright b^{xy}\longrightarrow b^{\alpha\beta}}{/xy=\alpha\beta\triangleright(\Box b)^{x}\longrightarrow b^{\alpha\beta}}\;{\to\Box}}{/xy=\alpha\beta,\alpha\beta=x\triangleright a^{\alpha\beta},(a\supset\Box b)^{x}\longrightarrow b^{\alpha\beta}}\;{\supset\to}}{/xy=\alpha\beta,\alpha\beta=x\triangleright a^{\alpha\beta},\Box(a\supset\Box b)\longrightarrow b^{\alpha\beta}}\;{\Box\to}}{/xy=\alpha\beta,\alpha\beta=x\triangleright\Box(a\supset\Box b)\longrightarrow(a\supset b)^{\alpha\beta}}\;{\to\supset}}{/xy=\alpha\beta,\alpha\beta=x\triangleright\Box(a\supset\Box b)\longrightarrow(\Box_{\beta}(a\supset b))^{\alpha}}\;{\to\Box}}{/xy=\alpha\beta,\alpha\beta=x\triangleright\Box(a\supset\Box b)\longrightarrow\Box_{\alpha}\Box_{\beta}(a\supset b)}\;{\to\Box}$$

Figure 2.17: Example theorem 1 in the lifted system.

Figure 2.16 shows a lifted system corresponding to the system of Figure 2.11. In this system, the inference rules describe not proofs but simply DERIVATIONS or PROOF-ATTEMPTS. Each derivation is associated with a set of equations which must be solved to obtain a proof.

More precisely, each sequent is of the form:

$$C/C' \triangleright \Gamma \longrightarrow \Delta$$

As always, formulas in Γ and Δ are labeled by terms explicitly indicating scope. During the proof, we accumulate a list of equations indicating constraints on the values of variables: C is the input list of equations and C' is the output list of equations.

Each formula in a sequent is associated with a list of free variables schematized by a subscript X in the inference rules of Figure 2.16; quantifier and modal rules which introduce a variable add the variable to this list. Skolem terms involve function symbols associated uniquely with quantifiers and modal operators (as indicated by subscripting); we build a Skolem term as a placeholder for a fresh eigenvariable by applying this function symbol to the list of free variables on the formula. The resulting system is necessarily rather dense in notation, but operates straightforwardly.

A proof of $\Gamma \longrightarrow \Delta$ is pair consisting of a derivation with end-sequent

$$/C \triangleright \Gamma \longrightarrow \Delta$$

where every formula in Γ and Δ is labeled with ϵ, together with a substitution θ—a finite map from scope variables to scope terms—such that $l\theta = r\theta$ for each equation $l = r$ in C.

The correctness theorem for this system states that $\Gamma \longrightarrow \Delta$ is provable in the lifted system if and only if it is provable in the ground system. When presented in the style of Herbrand's theorem for classical logic [Herbrand, 1971], as in [Lincoln and Shankar, 1994], the proof gives explicit transformations between the derivations of the two systems.

Proofs in the lifted system of our three S4 theorems are illustrated in Figures 2.17, 2.18 and 2.19. The figures present UNIFORM PROOFS [Miller *et al.*, 1991], as an illustration of how the lifted system facilitates systematic, goal-directed proof search. (Uniform proofs such as these are discussed much more carefully in Chapter 3.) In all three proofs, we

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{xy=\alpha\beta / xy=\alpha\beta, \alpha=x \rhd a^{\alpha} \longrightarrow a^{x} \qquad \dfrac{/xy=\alpha\beta \rhd b^{xy} \longrightarrow b^{\alpha\beta}}{/xy=\alpha\beta \rhd (\Box b)^{x} \longrightarrow b^{\alpha\beta}}\ \to\Box}{/xy=\alpha\beta, \alpha=x \rhd a^{\alpha}, (a \supset \Box b)^{x} \longrightarrow b^{\alpha\beta}}\ \supset\!\to}{/xy=\alpha\beta, \alpha=x \rhd a^{\alpha}, \Box(a \supset \Box b) \longrightarrow b^{\alpha\beta}}\ \Box\to}{/xy=\alpha\beta, \alpha=x \rhd a^{\alpha}, \Box(a \supset \Box b) \longrightarrow (\Box_{\beta} b)^{\alpha}}\ \to\Box}{/xy=\alpha\beta, \alpha=x \rhd \Box(a \supset \Box b) \longrightarrow (a \supset \Box_{\beta} b)^{\alpha}}\ \to\supset}{/xy=\alpha\beta, \alpha=x \rhd \Box(a \supset \Box b) \longrightarrow \Box_{\alpha}(a \supset \Box_{\beta} b)}\ \to\Box
$$

Figure 2.18: Example theorem 2 in the lifted system.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{xy=\alpha\beta / xy=\alpha\beta, \epsilon=x \rhd a \longrightarrow a^{x} \qquad \dfrac{/xy=\alpha\beta \rhd b^{xy} \longrightarrow b^{\alpha\beta}}{/xy=\alpha\beta \rhd (\Box b)^{x} \longrightarrow b^{\alpha\beta}}\ \to\Box}{/xy=\alpha\beta, \alpha=x \rhd a, (a \supset \Box b)^{x} \longrightarrow b^{\alpha\beta}}\ \supset\!\to}{/xy=\alpha\beta, \epsilon=x \rhd a, \Box(a \supset \Box b) \longrightarrow b^{\alpha\beta}}\ \Box\to}{/xy=\alpha\beta, \epsilon=x \rhd a, \Box(a \supset \Box b) \longrightarrow (\Box_{\beta} b)^{\alpha}}\ \to\Box}{/xy=\alpha\beta, \epsilon=x \rhd a, \Box(a \supset \Box b) \longrightarrow \Box_{\alpha}\Box_{\beta} b}\ \to\Box}{/xy=\alpha\beta, \epsilon=x \rhd \Box(a \supset \Box b) \longrightarrow (a \supset \Box_{\alpha}\Box_{\beta} b)}\ \to\supset
$$

Figure 2.19: Example theorem 3 in the lifted system.

proceed by performing all possible left rules, so as to decompose the formula to be proved into the atomic goal $b^{\alpha\beta}$. We then apply right rules strategically to the assumption $\Box(a \supset \Box b)$ so as to match the literal b in the assumption with the goal. This generates an equation $xy = \alpha\beta$ and a new goal a^{x}. This goal is established by matching it against the assumption of a in the right subtree of each proof. In the lifted system, the different theorems can be proved using rules in the same order—because of the permutabilities, only this order need be considered in proof search. The different scopes of rules are represented by the values of variables and are determined by unification. Here, the lower application of $(\Box \to)$ is scoped by the value of x. As always, the scope is identical to the scope of the assumption of a: either ϵ, α, or $\alpha\beta$.

2.4 Summary

A modal goal $[\mathrm{M}]A$ has the potential to be modular in two important respects. First, it can set up an independent (modular) subproblem for proof search, in which all assumptions and ambiguities introduced are local. Second, it can set up a goal for proof in which only restricted (modular) information can be taken into account. As described in section 2.1, modal operators seen this way provide a way to talk not just about the attitudes of agents or states in time, but more abstractly about the structure and context-dependence of logical specifications.

Exploiting this modularity to the fullest is both subtle and difficult. In previous work, modularity has been limited—because of the OVERLY STRONG semantic principles added to the logic, as in section 2.1.3; because of INFLEXIBLE proof procedures, which introduce prohibitive redundancy into the search space for general inference, as in section 2.3.1, 2.3.2 and 2.3.4; or, paradoxically, because of overly FLEXIBLE proof procedures, which are blind to the constraints of modularity in the logic, as in sections 2.3.3 and 2.3.5.

The next chapter refines the perspectives on the sequent calculus and functional translation developed in this chapter, and extends our ability to realize both kinds of modularity in efficient, general and predictable logic programming inference.

3 Logic Programming and Modal Logic

In Chapter 2, we investigated the potential of modal logic to describe both which assumptions can be applied within a proof, and how inferences involving those assumptions can be linked together. In this chapter, we show how to operationalize this view, so that modal formulas can be viewed as instructions for modular proof-search. Our result is a modal logic programming language DIALUP in which programs can establish disjunctions and existentially quantified sentences—and can embed these constructs under modal operators that describe the structure of a specification and at the same time resolve ambiguities in how programs will be executed. Such programs are said to encode MODULAR INDEFINITE INFORMATION.

Admitting indefinite information represents a significant advance over previous modal extensions of logic programming [Fariñas del Cerro, 1986; Debart *et al.*, 1992; Giordano and Martelli, 1994; Baldoni *et al.*, 1993; Baldoni *et al.*, 1996]. We can appreciate the conceptual and technical contribution of the extension in light of the results of Chapter 2. Conceptually, the work on context reviewed in section 2.1 reflects the increased applicability that knowledge representation schemes derive from the ability to describe indefinite information. (We will also see later, in Part II, that this ability makes this language expressive enough to describe the partial information needed in planning and in natural language generation.) Technically, DIALUP combines general-purpose inference mechanisms with a modular regime for structuring modal proofs and proof-search. We explained the difficulty of achieving such modularity in reviewing modal proof methods in section 2.3.

This chapter begins in section 3.1 by describing the scheme that DIALUP uses to interpret modal statements as instructions for proof-search. This description, which suggests both an informal characterization of the action DIALUP takes in search and a formal way to view that action in terms of the construction of sequent calculus proofs with a special form, grounds the discussion of the rest of the chapter. Armed with the concrete statement of how DIALUP works, we go on describe the modularity that DIALUP achieves, in section 3.2, and the uses of that modularity in structuring and transforming specifications, in section 3.3. Then, following up on the formal connection between DIALUP and the sequent calculus, we justify the soundness and completeness of DIALUP in section 3.4.

3.1 Modal Logic Programming in the Abstract

Building a proof is always a search problem. The goal is to find a way to use the available assumptions to establish a desired conclusion. The available operators transform and combine assumptions according to the rules of the logic. Proof-construction by a logic programming language is no exception. What makes logic programming different from a pure theorem-proving method like resolution—and from the refinements of resolution, such as ordered hyperresolution [Robinson, 1965a], which must actually be used in fast, general automatic provers [McCune, 1994]—is that search is made up of simple, predictable steps. At each step, the alternatives for search are determined from the available assumptions and the needed conclusion by a straightforward, intuitive and easily analyzed algorithm.

More precisely, the logical structure of a goal directly determines the search options available when that goal arises. Thus, logical symbols in goals can be seen as instructions for decomposing and transforming the search problem that the interpreter faces. Similarly, the atomic formulas that an assumption can be used to derive—the HEAD or HEAD of that assumption—serve as indexes that regulate whether an assumption can be applied. And the logical structure of the assumption provides an instruction for creating a set of new search problems whenever the assumption is used. This general perspective on logic programming has been formalized and analyzed under the name ABSTRACT LOGIC PROGRAMMING LANGUAGE [Miller *et al.*, 1991].

In this section, we describe the operation of a new modal logic programming language, DIALUP, in these terms. We begin in section 3.1.1 by introducing Prolog and describing how its operation can be understood in terms of specialized sequent calculi. This allows us to relate Prolog and the calculi for modal proof presented in sections 2.3.4 and 2.3.5. Then, in section 3.1.2, we introduce the search strategy that DIALUP uses; the strategy is presented both as a specialized sequent calculus and as an algorithm governing the steps that the interpreter can take in building a proof.

3.1.1 Prolog, its extensions, and logic

Prolog, as the generic logic programming language, provides the simplest illustration of an abstract logic programming language [Clocksin and Mellish, 1994]. Prolog specifications are written in statements of the following simple form:

(36) $$\forall \overline{X}(g_1(\overline{X}) \wedge \ldots \wedge g_n(\overline{X}) \supset h(\overline{X}))$$

Here $h(\overline{X})$ and all the $g_i(\overline{X})$ are just atomic formulas. Such statements are called Horn clauses; by eliminating the universal quantifiers and translating the implication to disjunction, they correspond to a subclass of the clauses used in resolution deduction described in section 2.3.3. The formula $h(\overline{X})$ is the HEAD of the clause; the conjunction $g_1(\overline{X}) \wedge \ldots \wedge g_n(\overline{X})$ is the BODY of the clause. (The effect of other connectives can in some cases be obtained using logical equivalences—existential quantification can be simulated using function terms; some disjunctions can be simulated by duplicating clauses—but ultimately specifications of the form (36) are all Prolog offers.)

The Prolog interpreter embodies a simple search strategy for identifying whether a query Q is a consequence of a Prolog specification P. At each point, the interpreter keeps track of the work remaining to establish the query as a stack of atomic formulas that the interpreter must prove from P to establish the query. The top of this stack is the current goal G. When the query Q is first posed, this list contains just Q itself and Q is the current goal.

The interpreter determines how the search state might be extended by an operation of BACKWARD CHAINING that applies the clauses in the specification toward the current goal G. A clause matches when its universal quantifiers can be instantiated to fresh variables that allow the head of the clause to unify with the current goal G. A successful match allows the state to be transformed by eliminating G from the goal stack while pushing the body of the matched clause onto it. The Prolog interpreter explores these alternatives following the order that matched clauses appear in the original specification and using depth-first search.

The view of Prolog as an abstract logic programming language is based on explicating this behavior in logical terms. At each stage in search, there is a multiset P of clauses in force and a particular goal formula G that the interpreter is addressing. The task of the interpreter is to derive G from P. We can write this using a sequent $P \xrightarrow{?} G$, to underscore the parallel between the deductive task faced by the interpreter and the formal judgments derived in sequent systems for logical proof. In Prolog, the clauses P are always exactly the clauses in the original specification; the current goal is always the first atomic formula on the goal list. The transformations involved in backward chaining implement the sequent calculus rules $(\forall \rightarrow)$, $(\supset\rightarrow)$ and $(\rightarrow \wedge)$ that govern clauses: $(\forall \rightarrow)$ introduces fresh logic variables to instantiate the clause, $(\supset\rightarrow)$ allows the head to be deduced but introduces a new conjunctive goal which $(\rightarrow \wedge)$ reduces to atoms. The step of unification implements the link of an initial sequent that connects the head of the clause with the current goal.

The extensions of Prolog described in [Miller *et al.*, 1991; Hodas and Miller, 1994; Miller, 1994] emphasize the close connection of Prolog to logic by describing this search strategy—where goals are viewed as instructions for search—directly in terms of rules of the ordinary sequent calculus (for intuitionistic or linear logic). Viewing a complex goal G as an instruction for search means that the options to prove $P \xrightarrow{?} G$ depend only on the logical structure of G. In particular, if G is a compound formula, the next rule applied in building the proof must be the sequent rule for G. Proofs where the ordinary sequent rules are used in this order are called UNIFORM PROOFS. Using the idea of uniform proofs, [Miller *et al.*, 1991; Hodas and Miller, 1994; Miller, 1994] show more generally how goals and clauses may be more complex formulas in intuitionistic logic or linear logic and how, for example, the clauses in effect will evolve as search for these formulas proceeds.

DIALUP represents a way of adapting this framework to modal logic. Informally, DIALUP starts from an explicitly-scoped sequent calculus for modal logic, like those introduced in sections 2.3.4 and 2.3.5. Recall that these systems interleave semantics-based translation of modal formulas and classical reasoning about the translation; to keep track of the translation, they annotate formulas with string terms representing paths to possible worlds. (In this chapter, paths will be given by a superscript term labeling formulas; α, β, etc. represent arbitrary elements in paths; *x*, *y*, etc. represent logic variables in paths, and μ, ν, etc. are metavariables over paths as a whole.)

By analogy to the abstract logic programming view of Prolog, then, the search problem faced by the DIALUP interpreter at any stage has a form schematized by $P \xrightarrow{?} G^{\mu}$. The goal is a formula G to be shown true at world μ; the proof must make use of the formulas in P, each of which is also labeled by the world at which it is supposed true.

More so than for Prolog, for DIALUP, we must be careful to specify what this search schema really represents. The perspective of abstract logic programming languages encodes the search of the interpreter not as a concrete procedure but only as a subclass of proofs; and the perspective ignores the use of unification to implement the interpreter. It is possible to be more precise on both counts.

The explicit structure of logic programming proofs can be formalized by writing alternative sequent rules. These rules are modified so that they can only be chained together as inferences in uniform proof order. For example, the structure of sequent rules can distinguish the current goal, as well as the program clause currently being matched against that goal, when applicable. This gives a FOCUSING PROOF SYSTEM, whose deductions are called FOCUSING PROOFS [Andreoli, 1992]. Then the correctness of the logic programming language lies in showing that the restricted proof system is sound and complete—that every proof in the ordinary system can be transformed into a proof with the restricted form, and vice versa.

Similarly, the role of unification in building the proof can also be formalized by providing alternative sequent rules. Recall that section 2.3.5 illustrated how this could be done by annotating sequents with equations that describe restrictions on the values of variables that have been derived by the interpreter at various stages in proof search.

This precision is called for in DIALUP for two reasons. The first is our goal of modularity for indefinite information: as we saw in sections 2.3.4 and 2.3.5, we will not succeed if we exploit simple ground (or simple lifted) proof. Only analyzing search order and unification explicitly will allow us to construct the intermediate proof system we need, with the modularity of the ground proof system of section 2.3.4 and the flexibility of the lifted proof system of section 2.3.5. The second is the goal of handling path representations of possible worlds efficiently, which is addressed in Chapter 4. The algorithms proposed there exploit the structure of equations in proofs; this structure must therefore be made explicit. At the same time, these new algorithms introduce constraint-based alternatives to unification; a careful, equational presentation of the logic programming language is therefore required to correctly describe the alternatives that are actually considered during search.

Thus, we shall see that a description of a DIALUP task like $P \xrightarrow{?} G^{\mu}$ implicitly includes two additional kinds of information. One kind records aspects of the state and history of the derivation that can be used to determine what instructions for search will be performed next. The other kind records information about the values of variables as the problem is begun and, when appropriate, after the problem is solved. The formal description of DIALUP must make this explicit.

3.1.2 Defining DIALUP

We will begin by describing DIALUP schematically. The natural starting point is to describe the instructions for search that break down complex goals into atomic ones. The interpretation of some connectives is easy; for example, in decomposing $P \xrightarrow{?} G^{\mu}$ for a goal G with prefix μ, proof search can observe the directives in (37).

(37) a If G is of the form $B \wedge C$, the proof must be constructed by solving $P \xrightarrow{?} B^{\mu}$ and $P \xrightarrow{?} C^{\mu}$.

b If G is of the form $B \vee C$, the proof must be constructed either by solving $P \xrightarrow{?} B^{\mu}$ or by solving $P \xrightarrow{?} C^{\mu}$.

c If G is of the form $\exists xA$, the proof must be constructed by solving $P \xrightarrow{?} A[X/x]^{\mu}$, using a new logic variable X to leave open some term t defined at world μ.

(37a) says that a conjunctive goal encodes two subproblems, one for the left conjunct and one for the right conjunct; both must solved. (37b) says that a disjunctive goal also defines two subproblems, but that either may be solved to complete the specified search. Finally, (37c) indicates that existential quantifiers describe parametric search problems; part of the search is finding a value for a variable X that allows the goal to be solved.

A modal operator, meanwhile, transforms the search problem by considering a transition to a new possible world. More formally, its action is given in (38).

(38) In general, if G is of the form $[\textsc{o}]A$, the proof must be constructed by solving $P \xrightarrow{?} B^{\mu\alpha}$, where α is a new constant representing an arbitrary transition of accessibility in $AF_{[\mathrm{O}]}$.

In some sense, (38) also describes a parametric search problem, but in this case the value to be substituted is picked at random before the search begins. Proof-theoretically, this process ensures that no information that we discover about the parameter will be accidental. Only necessary conclusions about this parameter will be drawn. Of course, this suits the meaning of the necessity operator $[\textsc{o}]$. Using the same idea, we can handle the additional connectives $\forall$ and $\supset$ in goals in certain special cases. These rules are given in (39).

(39) a If G is of the form $[\textsc{o}]\forall xA$, the proof must be constructed by solving $P \xrightarrow{?} A[c/x]^{\mu\alpha}$, for a new constant α representing a transition in $AF_{[\mathrm{O}]}$ and a new constant c defined (only) at world $\mu\alpha$.

b If G is of the form $[\textsc{o}](B \supset C)$, the proof must be constructed by solving $P, B^{\mu\alpha} \xrightarrow{?} C^{\mu\alpha}$ where α is a new constant representing a transition in $AF_{[\mathrm{O}]}$.

Notice that these rules implement explicitly-scoped search by combining features of ground and lifted proof. On the one hand, we use logic variables and unification to allow values to be delayed; DIALUP uses lifted proof. On the other hand, when we use fresh parameters in (38) to implement $(\rightarrow \Box)$ rule, we are adopting a feature of the ground system. This combination is what gives DIALUP efficient structural modularity; in particular, analysis of DIALUP proofs will show that the use of fresh variables restricts the use of disjunctions in a way that the interpreter can detect and exploit.

This combination also represents the key obstacle to overcome in showing that DIALUP is correct. In a lifted system, we would expect to Skolemize the representation of the transition introduced for G by rules such as (38) and (39). We would associate the goal G with some function, and construct an appropriate parameter for search by applying this function to the logic variables that had been introduced in deriving G. Such Skolemizing is LESS CONSTRAINING than the use of a fresh constant. With Skolem functions, but not with fresh constants, a single inference elsewhere in the proof can contribute both towards this new goal and towards other occurrences where G is proved. All that is required for this, if Skolem functions are used, is that we equate the logic variables revealed in processing the two occurrences of G. The relaxation involved in Skolemizing must be accommodated to retain completeness in general. Why is it not needed here?

The answer exploits the technical role unification plays in proof search, cf. [Gallier, 1986; Lincoln and Shankar, 1994]. Unification essentially allows proofs to be built out of order; it allows some inferences to be added to the proof later than they could be otherwise. This is good because with unification those inferences can be added only when they are known to be needed. However, there is the complication that in delaying an inference, the interpreter may decompose a conjunctive search problem. In this case, what must really be a single, earlier inference appears as two or more, later inferences. The point of Skolemization is to handle these cases where proof search discovers that two proofs of the same goal or two uses of the same premise should be collapsed. Skolemization is therefore necessary only to the extent that the inferences can be delayed in proof search. By processing goals as soon as they arise, using rules such as (37) and (38), modal inferences in goals are never delayed and there is no need for Skolemization. (Clauses in the program are delayed, however, so they must be treated differently.)

The rules in (37), (38) and (39) describe the processing DIALUP will do in breaking down any complex goal into a combination of atomic goals. Once this process is completed, the program itself is consulted; the interpreter performs an appropriate version of backward chaining. The interpreter chooses a clause that might match the goal nondeterministically from the program and dissects it—by rules dual to the ones above that dissect goals—to obtain an atomic fact and a sequence of new subgoals. The atomic fact is constrained to equal the goal to discharge that goal; then the new subgoals are processed in turn.

As befits backward chaining, the action of the interpreter is specified in terms of how a particular clause formula should be matched against a particular atomic goal. Again, we can flesh out this process by describing some easy cases first.

(40) a To match an atomic formula P^{ν} against the atomic goal G^{μ}, unify P and G and set ν equal to μ.

b To match a formula $A \wedge B^{\nu}$ against the atomic goal G^{μ}, either match A^{ν} against G^{μ} or match B^{ν} against G^{μ}.

c To match a formula $\forall x A^{\nu}$ against the atomic goal G^{μ}, introduce a new logic variable X which is to take a value at world ν, and match $A[X/x]^{\nu}$ against G^{μ}.

(40a) just specifies precisely how deriving a goal discharges it. The two paths are constrained to be equal, rather than unified, to capture the effect of axioms on modalities. Because of the composition and identity transitions that interpret these axioms, path equations can have many solutions. Indeed, in general—and for many cases of practical interest—exponentially many distinct matches arise each time a goal is matched against a clause. Explicit UNIFICATION requires each alternative to be considered separately. This is unacceptable. Nevertheless, previous modal logic programming languages enumerated all alternative modal unifiers at each stage and backtracked separately among them [Debart *et al.*, 1992; Baldoni *et al.*, 1993]. In contrast, DIALUP maintains modal matches using a constraint algorithm that builds a tree of worlds to match of all the clauses and goals in the proof incrementally in polynomial time—avoiding all backtracking among matches. This algorithm is fully described and justified in Chapter 4.

(40b) arises when the search tasks to which the interpreter is already committed will result in two facts being true; (40b) states only that at such times either fact may be used to discharge the goal. Similarly, (40c) arises when the interpreter's commitments entail a proposition that can be arbitrarily instantiated; (40c) leaves open the instantiation by introducing a fresh logic variable and continuing the match.

Three cases are somewhat more involved, but will still be relatively familiar to Prolog programmers (given the preceding discussion).

(41) a To match $[\mathrm{O}]A^{\nu}$ against the atomic goal G^{μ}, introduce a fresh logic variable x over transitions in $AF_{[\mathrm{O}]}$ and match $A^{\nu x}$ against G^{μ}.

b To match $A \supset B^{\nu}$ against the atomic goal G^{μ}, post the goal $P \xrightarrow{?} A^{\nu}$ to derive A at ν given the clauses P currently in effect, and continue to match B^{ν} against G^{μ}.

c To match $\exists x A^{\nu}$ against the atomic goal G^{μ}, Skolemize. That is, we assume that the occurrence of the existential quantifier in the program is associated with a function f, and the sequence of logic variables that have been introduced during matching is given by the list V. This allows us to use the entity $f(V)$ existing at world ν as a witness for the existential quantifier. Thus, the statement $\exists x A^{\nu}$ is transformed into the statement $A[f(V)/x]^{\nu}$ and this is matched against G^{μ}.

(41a) indicates that modal operators, like quantifiers, allow arbitrary instantiations; a logic variable allows an appropriate transition of accessibility to be instantiated by unification with the goal. (41b) indicates that once an implication is derived, we can discharge the goal by matching it against the consequent of the implication, provided we undertake the new goal of establishing the antecedent. Finally, (41c) indicates how existential quantifiers are Skolemized as matches take place. Skolem functions are necessary with existential quantifiers because during matching they will frequently be unified with logic variables that appear on many other goals. When we tackle those later goals, we might want to apply this same program clause again to prove them, so we must take care to have a compatible representation of the generic witnesses of the existential on both occasions when the clause is used.

The most difficult matching action is that taken for disjunctive specifications. Disjunctive specifications offer a way to prove a goal by cases. To use $A \vee B$, first assume A and prove the goal, then assume B and prove the goal. The wrinkle for matching lies in the question, what is the goal to prove by cases? The logic programming strategy identifies the need for a proof by cases somewhere in the middle of the first case. Generally, some inferences that depend on A will already have been found when we match against the disjunction, recognize that A applies, and tackle the A case. When we consider the case where B is used, we will want to ignore these goals and inferences that depend on A and instead reprove some earlier goal that can be proved both using A and using B. In general, the safe thing to do after finishing the A case is just to restart the original query assuming B. This is the basis of the restart rule of the Near-Horn Prolog family of disjunctive logic programming languages [Loveland, 1991; Reed *et al.*, 1992; Nadathur and Loveland, 1995].

We will adopt such a restart rule, with an important modification that specializes it to modal logic. In modal logic, when we derive $A \vee B$ at world ν, we don't have to restart the whole search for the query. We only need to restart some RESTART query—one which has introduced a new transition and shifted the consideration of the interpreter to a new possible world ν'—where ν' is reachable from ν. As proved in Section 3.2, the logic programming search regime of DIALUP allows the modularity of modal logic to be read off of programs and goals, so a case analysis that is introduced in world ν must be confined to that world. This property is encoded in the rule (42).

(42) To match $A \vee B^{\nu}$ against the atomic goal G^{μ}, pick A^{ν} (or, under certain circumstances B^{ν}) and match it against G^{μ}; post the goal $P, C^{\nu} \xrightarrow{?} O^{\nu'}$, where C is the unmatched case of the disjunction, P is the program clauses currently in effect, and O is any restart goal whose annotation ν' extends ν (O may be selected when the goal is reached).

The scope of restarts is only one of the difficulties for search control that disjunctions introduce. Another is the problem of redundancy, which may explode the search space when goals follow one way using case analysis and another way without. One regime for avoiding redundancy involves keeping track of CANCELLATIONS, which mark the contribution of a disjunct to a proof in case analysis [Loveland, 1991]. The possibility of matching B instead of A in (42) is governed by cancellations. In particular, it is triggered only by the need to find a cancellation for a previous disjunct before restarting with this disjunction—and it is restricted to circumstances where this cancellation can be obtained only by using B immediately in this way. Section 3.4 describes in more detail how DIALUP uses cancellations not only to eliminate redundant search paths, but also to help justify modular search in general.

The definitions in (37), (38), (39), (40), (41) and (42) together describe a logic programming interpreter that executes specifications with the following syntax:

(43) $G ::= [\text{O}]G \mid G \wedge G \mid G \vee G \mid [\text{O}](\forall x G) \mid \exists x G \mid [\text{O}](D \supset G)$
$D ::= [\text{O}]D \mid D \wedge D \mid D \vee D \mid \forall x D \mid \exists x G \mid G \supset D$

The syntactic class D (for definitions) describes possible specification clauses; the class G describes possible goals. This syntax gives the logic programming language DIALUP which we will take as a reference point for executing modal specifications for the rest of this study. DIALUP is so named because it handles Disjunction and Intensional Abstractions (i.e. possible worlds) in a Logic with Uniform Proofs.

3.2 Modality and Modularity in Design

The outline of DIALUP in section 3.1.2 offered very concrete instructions for navigating formal search problems, and motivated those instructions in section 3.1.1 with quite a technical approach to logic. This is not an easy perspective to take in writing a DIALUP program. Fortunately, as this section shows, it is rarely necessary.

The payback from any use of logic in programming is that programming often becomes an exercise of writing formulas with the right meanings. DIALUP allows programmers to refine those meanings by describing computations in modal terms. These descriptions can draw on the intuitive meanings of modal operators, as well as the general connection between modal logic and modularity. For example, a modal goal $[\text{A}]p$ can be thought of as a query to a specialized agent A about whether it knows p. Under DIALUP's search strategy, such intuitions can offer guides—coarse ones or precise ones, as necessary—to help a programmer understand and control the operational definitions DIALUP uses to execute a specification.

The simplest illustration of modality and modularity is with specifications and search problems that use only modal connectives. Here is a representative problem:

(44) $p, [\text{A}]p, [\text{B}]p \xrightarrow{?} [\text{B}]p$

(44) involves a program specified with three modules: what is really true, what is described by module or agent [A] and what is described by module or agent [B]. Each module has its own content, which can be set up by a programmer in a modular way, and must in turn be accessed by DIALUP in a modular way. (44) represents the goal of showing that p is part of the content specified by [B]. Accordingly, a programmer can predict at a high level that the clause $[\text{B}]p$ will be taken into account in proving the goal, whereas the other clauses will not be.

This intuition can be confirmed by analyzing the concrete actions of the interpreter. In processing the goal, the first low-level action of the interpreter is to introduce some new [B] transition—β say—and to attempt to prove p after β. Then the interpreter tries backward chaining. All of the statements in the specification describe the formula p that we need to prove, but only the statement $[\text{B}]p$ allows the transition β to be matched. By applying this statement, of course, the interpreter reaches a goal state that answers the query.

Formally, here is why the other two clauses do not apply. To apply the fact p, we would have to match the real world against the world β. This is impossible because the worlds are represented by distinct ground terms. To apply the fact $[\text{A}]p$, meanwhile, we would have to unify a logic variable X representing a transition of $[\text{A}]$ accessibility with the ground transition β. This would cause a type mismatch; $[\text{A}]$ and $[\text{B}]$ do not interact and therefore β does not represent a transition contained in $AF_{[\text{A}]}$.

Thus, we can see the new term β as allowing the interpreter to record the fact that the goal p is modular, while continuing to process the goal. Each clause is checked against terms like β before it can contribute to the proof of p. The structure and type of β prevents outside information from affecting how the goal p is established. This generalization allows a programmer to take a higher-level view. Details of modal matching are abstracted, and the modal operator of the goal $[\text{B}]p$ reduces to an annotation that directly restricts what clauses should apply to prove p.

In modal logic, modularity not only blocks outside information from flowing into a modular goal, it also prevents inside information from flowing out. A good illustration of this is the query represented in (45).

(45) $\xrightarrow{?} [\text{A}](p \supset q) \vee [\text{A}]p$

The specification for this query is empty. (45) is not a theorem of modal logic, so the interpreter does not find a proof of it. The interpreter first tries the left disjunct, $[\text{A}](p \supset q)$; it introduces a transition to a new possible world α where it assumes p and tries to prove q. The interpreter is stuck there, so it backtracks to the right disjunct $[\text{A}]p$; it introduces a transition to a new possible world β and tries to prove p there. Again it fails; now it must abandon the goal. This result matches our intuitions about what the query would mean: we don't expect a module automatically either to specify $p \supset q$ or to specify p. Note in contrast that the corresponding first-order formula, $(p \supset q) \vee p$, is valid, and that several modal axioms presented in section 2.1.3 even make (45) valid.

Again for (45), the low-level operation of DIALUP in manipulating sequents and possible worlds is easily abstracted into a simple form that can be read off the program. Namely, q in (45) is an independent goal, whose search automatically takes into account just the $[\text{A}]$ information and the special added formula p.

Modularity can provide a link between DIALUP's low-level operation and a programmer's abstract intent not just for modal Horn clauses, but for quantified and disjunctive programs as well. Quantifiers and disjunctions can both introduce ambiguities in modal proofs, but these ambiguities must be resolved where they are created—by the module or agent that introduces them.

Let's start with quantifiers. The search problem in (46) is a representative example where the modularity of quantification comes into play.

(46) $[\text{A}]\exists X.r(X) \xrightarrow{?} \exists X[\text{A}]r(X)$

Here is the informal difference between the specified formula and the query. According to the formula in the specification, it is part of the content of $[\text{A}]$ that there should be some individual X in relation r. The query asks whether there is any entity X about which $[\text{A}]$ says X is in r. It is instructive to read this query as asking whether agent $[\text{A}]$ knows what is r [Hintikka, 1971]—asking $[\text{A}]$ to provide a specific, concrete global value that it knows is r. When we contrast the local knowledge $[\text{A}]$ has here (just that there is an r) with the global information the query demands, it is easy to see that the query is stronger than the specification. For example, if agent $[\text{A}]$ really knew only that either a or b was r, then it would meet the specification but would not satisfy the query.

DIALUP's rejection of (46) of course depends once again on a relatively subtle account of how the interpreter manipulates possible worlds during search. In particular, in proving the query, DIALUP first introduces a variable X over entities in the real world, then considers whether it can show $r(X)$ at some new $[\text{A}]$-world α. DIALUP consults the specification: for any $[\text{A}]$-world x, the specification describes an individual $f(x)$ that exists at x and is in r. We can unify α with x, but we cannot then unify X with $f(\alpha)$. The world where $f(\alpha)$ is introduced does not match the world where the value of X is needed.

Note by contrast that if we reverse the role of query and specification, the interpreter succeeds. For then the interpreter only needs to find an individual X at world α that has property r. The specification defines a real individual $f(\epsilon)$ that has property r in α (or any other $[\text{A}]$-accessible world). X can be unified with $f(\epsilon)$ because, by the increasing domain constraint if X exists in the real world then X also exists at α (cf. section 2.2).

Again, we can invoke the idea of modularity to link these two descriptions of (46), as a bridge between operations on possible worlds and high-level intuitions about what DIALUP should do. $[\text{A}]\exists X.r(X)$ introduces X as a modular individual. Its very existence is limited to the module $[\text{A}]$, to agent $[\text{A}]$'s momentary thought. Moreover, different $[\text{A}]$ goals must appeal to different values of X; we cannot assume that the value of X is constant as we consider different ways the knowledge or content of $[\text{A}]$ could be fleshed out. As with propositions, the maintenance of possible worlds in the interpreter is what keeps X a modular individual. The limitation on X's existence is encoded by associating X with a possible world. Since different goals appeal to different possible worlds, the same association ensures different goals do not succeed by using the same X as the r.

Finally, we have disjunction. Disjunction really just parallels existential quantification, although the modal restart rule for disjunction gives disjunction a different look. (47) is an example to bring this out.

(47) $$[\text{A}](q \vee r) \xrightarrow{?} [\text{A}]q \vee [\text{A}]r$$

(47) invites us to consider whether, once we have specified an agent A that knows that either q or r is true, whether it follows that either A knows q or that A knows r.

Consider how proof search for (47) proceeds. The interpreter follows the left disjunct in the goal, introduces world α, and looks to the program to establish q at α. The specification reveals that proof by cases is appropriate: in one case outlined by the disjunctive specification, *q* holds at any [A]-world and α in particular. The other case requires the use of *r* at α. It is to this case that the interpreter now turns.

The modal restart rule dictates that the goal to reprove for this case should be some restart goal after α was introduced: this must be the goal of showing *q* at α. The interpreter attempts this goal, and after some search identifies that no progress can be made in analyzing this case: there is no way to get to *q* by using *r*. At this point the interpreter identifies failure.

The choice of restarting to the goal of showing *q* at α encodes and exploits the fact that disjunction is modular. To see this, suppose the interpreter was to attempt to reprove the entire query assuming *r* at α, as part of the case analysis. As it processed each disjunct, the interpreter would introduce transitions to new possible worlds, β and γ. Because these terms are distinct from α, the interpreter would never be able to use the fact that *r* was true at α to prove something about what was true at β or γ. So the interpreter would ultimately determine that there was no progress to be made by its case analysis. It would abandon its search.

Again, underlying this discussion is a general fact that allows a programmer to construct and think about a DIALUP program at a high level. There is no way to exploit case analysis at one world if the proof is restarted to a closer world. Disjunction is modular. The modular restart rule, with its bookkeeping of worlds and goals, is DIALUP's low-level strategy to exploit the modularity of disjunction and shortcuts these doomed searches.

3.3 Exploiting Modularity for Search Control

Modular execution is an important resource for describing inference problems in tractable and reusable specifications. Modal operators can be used to limit the number of alternatives that need to be considered at any point, to constrain the size of proofs that need to be constructed, and to allow specifications to be flexibly reused. Let's look at some examples of this.

3.3.1 Problem Decomposition

A record of the possible interactions that may arise in problem-solving can be an important part of a specification of how to reason in a domain. Here is a simple example.

In planning a trip, it is important to determine before you begin that you will be able to complete the trip successfully. To be stranded midway would be a real disaster. Often, however, many details about the trip cannot be resolved in advance. For such situations, showing that the trip will be successful means showing that you will be able to negotiate these details when the time comes, no matter how they turn out. What makes it possible to quickly derive confidence in a planned journey is the knowledge that such details cannot conspire together to require global revisions of the plan.

Thus, suppose one leg of a journey involves taking an early train. At the station, you have to get a ticket for the train and (if you're like me) get a cup of coffee. Tickets can be purchased from a teller at a window or from automatic machines; the windows

$$\boxed{\begin{array}{ll} \multicolumn{2}{c}{t \wedge c \supset j.} \\ a \supset t. & d \supset c. \\ w \supset t. & m \supset c. \\ a \vee w. & d \vee m. \end{array}}$$

Figure 3.1: Unstructured logic program

$$\boxed{\begin{array}{ll} \multicolumn{2}{c}{[\textsc{t}]t \wedge [\textsc{c}]c \supset j.} \\ [\textsc{t}](a \supset t). & [\textsc{c}](d \supset c). \\ [\textsc{t}](w \supset t). & [\textsc{c}](m \supset c). \\ [\textsc{t}](a \vee w). & [\textsc{c}](d \vee m). \end{array}}$$

Figure 3.2: Modular logic program

Symbol	Content
j	I can have a successful train-trip
t	I can obtain a ticket
c	I can obtain coffee
a	I can use an automatic ticketing machine
w	I can use a teller's window
d	I can get coffee from Dunkin Donuts
m	I can get coffee from McDonalds

Figure 3.3: Interpretations of symbols for the example

can have prohibitive lines and the automatic machines can be out of order, but the station management always makes sure that one quick and easy method is available. Similarly, there are a couple of places to get coffee at the station; you can be sure at least one will be open at any time trains are leaving, but since their hours vary, you may not know which. Using the abbreviations in Figure 3.3, we might formalize this situation by the logic program of Figure 3.1.[1]

In general, without knowing more about a specification, we can expect a number of cases to be considered in proof search that is exponential in the number of ambiguities in it. Here, for example, searching automatically for proofs of j is likely to require showing that j holds independently in the four cases that the program specifies (cases in which we assume

[1]The use of proposition letters is a harmless abbreviation that allows the MODULARITY of the example—our main focus—to shine through. DIALUP could also represent the problem using complex formulas in a logic of knowledge and action like that developed in chapter 7; this expressiveness provides familiar motivation for modal logic programming [Debart *et al.*, 1992]. For example, j could be fleshed out in terms of the knowledge $[\textsc{k}]$ of the planning agent about the hypothetical consequences of an event—$\exists e\, [\textsc{k}]\, \forall t\, [\textsc{k}]([\textsc{k}]\; do(e, i, now, t) \supset [\textsc{k}]\; \exists e'\, jrny(e', now, t) \wedge succ(e'))$.

one of *a* and *w* and assume one of *d* and *m*). A representative example of this is the naive restart rule that starts to process the original query in each case; consider proving *j* under this strategy. The first subproof of *j* takes care of the *d* and *a* case by using the first disjuncts. This subproof requires two restarts of the goal *j*, once assuming *a* and once assuming *w*—leading to a proof of *j* from *m* and *a* and a proof of *j* from *d* and *w*. The last case, proving *j* from *m* and *w*, arises as a restart goal in both of these subproofs. This represents good behavior for the naive restart rule; in problems with multiple case analyses, subproofs can reintroduce cases in such a way that search never terminates [Loveland, 1991]. Expanding the search space can delay the point where this happens, but cannot avoid difficulties if the full space must be explored—whether because all solutions are needed or because a dead end must fail before an alternative is tried.

The specification of Figure 3.1 omits the important fact that the ambiguities are independent. Where you get your ticket doesn't affect where you get coffee, and vice versa. If we provide this better description of the domain, a simple search strategy will be able to break up the proof in advance using the knowledge that these alternatives are independent: It will restrict the ambiguity *d* or *m* to proving *c*, and restrict *a* or *w* to proving *t*. In general, when alternatives are specified not to interact, worst-case proof size increases only linearly as new independent ambiguities are added. This makes for fast failure as well as fast success.

Modal specifications can indicate that alternatives do not interact. They do this, metaphorically, by describing how problem-solving tasks in a domain can be assigned to separate and independent problem-solving agents with specialized information. The agents work individually, and only combine their results after they have derived their solutions independently. Such a specification is given for our train problem in Figure 3.2. The specification invokes two necessity operators, $[\text{T}]$ and $[\text{C}]$ to distinguish the goals and program clauses describing getting a ticket from those describing getting coffee. Metaphorically, Figure 3.2 describes how problem-solving tasks in catching a train can be assigned to separate problem-solving agents with specialized information—one $[\text{T}]$ that knows just about tickets and another $[\text{C}]$ that knows just about coffee.

This metaphor leads directly to intuitions about search that can be applied correctly to DIALUP without delving into the intepreter's internals. The problem of getting coffee is assigned to agent *C* by the goal $[\text{C}]c$. *C* has certain alternatives to consider in getting coffee (*d* or *m*?); *C* considers these alternatives and no others in solving its task. Likewise, the problem of getting a ticket is assigned to agent *T* by the goal $[\text{T}]t$. *T* considers just its alternatives—*a* or *w*? Since the two agents are reasoning separately about different goals and ambiguities, the record of their problem solving is just a combined record of their independent steps—not, as before, an interacting record with combined resolutions of ambiguities.

The detailed record of DIALUP's search for *j* illustrates how DIALUP captures this high-level search process with a more detailed manipulation of formulas and possible worlds. We begin by backchaining from *j* to new goals $[\text{T}]t$ and $[\text{C}]c$. To interpret the modal operator

```
fib(0,1).
fib(1,1).
*I *J *K *L *M *N (succ(I,J), succ(J,K),
                   fib(I,L), fib(J,M),
                   sum(L,M,N) -> fib(K,N)).
succ(0,1).      succ(1,2).      succ(2,3).
succ(3,4).      succ(4,5).      succ(5,6).
succ(6,7).      succ(7,8).      succ(8,9).
succ(9,10).     succ(10,11).    succ(11,12).
*X *Y ?Z sum(X,Y,Z).
```

Figure 3.4: Naive Fibonacci program

[T], we introduce a fresh T TRANSITION α. This corresponds to moving to the *T* module or considering the inference of a *T* problem-solving agent. We must now prove the goal *t* at the world reached by the transition α. Backchaining, we obtain the goal of showing *a* at α. Matching the disjunction, we discharge this goal but are left with the case where *w* is true (still at α). We must restart an earlier goal taking *w* into account. Now, all transitions lead further from the initial world and goals always introduce fresh transitions; this is the model-theoretic counterpart to the independence of agents and modules described earlier. So, *w* must ultimately contribute to our goal of *t* at α; we restart that goal. Immediately, we backchain to *w* at α, use the assumption and finish the case. In an analogous process, we handle the goal [C]*c* by introducing a new C transition and consider proving *c* at the new world β. Backchaining and matching the disjunction discharges the goal, but leaves us with a case analysis in which we assume *m* at β. As before, the world at which *m* is assumed requires us to restart *c* at β. We backchain to the assumption *m* at β. The labels of formulas—model-theoretic representations of the modular structure of clauses—dynamically bound the restarts of the interpreter and force case analyses to proceed independently.

3.3.2 Memoization

Modality and implication provide a way to eliminate redundancy in proof search by recording and reusing intermediate results. This process is known generally as memoization. Encodings of memoization in expressive logic programming languages go back to [Miller *et al.*, 1991].

Figure 3.4 shows a logic program that cries out for memoization. The program gives a naive specification of the Fibonacci function, expressed as a relation `fib(K,N)` true when `N` is the `K`th Fibonacci number. The first two clauses establish the value of 1 for the first two Fibonacci numbers; the next clause establishes that any subsequent Fibonacci number is the sum of the two previous ones. The remainder of the specification lays out the mathematical facts to which we will appeal: the linear order of the first (baker's) dozen numbers and the

N	Time to find A with fib(N,A), in Linc CPU seconds
0	.01
1	.01
2	.03
3	.06
4	.1
5	.2
6	.4
7	.8
8	2
9	4
10	10
11	25
12	60

Figure 3.5: Executing the naive program

existence of a total function computing addition.

The clauses in Figure 3.4 are written in a convenient concrete keyboard syntax for first-order formulas. In this convention, `*X` represents the universal quantifier $\forall x$; `?X` represents the existential quantifier $\exists x$; `->` represents implication; `,` represents conjunction; and `;` represents disjunction. Lower-case strings name atomic propositions and primitive relations; upper case strings indicate variables in terms, and indicate modal operators elsewhere.

The program of Figure 3.4 is not exactly an executable specification. The proofs are too big. Using logic programming proof search, any proof we find that the `K`th Fibonacci number has value `N` has a size that exceeds `N`. The proof has to have at least `N` steps because logic programming search attacks each subproblem of proving `fib` separately, and these subproblems only ground out at clauses that make unit contributions to the overall sum. But the Fibonacci numbers grow exponentially in `K`. Famously, the ratio of successive Fibonacci numbers approaches the constant $\frac{1+\sqrt{5}}{2}$. This explosive growth is illustrated in Figure 3.5, which gives the running times for solving `fib` queries using this specification.

The complexity of executing the program of Figure 3.4 is a property of the program, not a property of the Fibonacci function. The `K`th Fibonacci number can in fact be computed immediately given the values of the previous Fibonacci numbers. Thus, if the computations of these values are saved and reused, proofs of size `K` can the specify the value of the `K`th Fibonacci number.

This characterization has an unsettlingly procedural flavor. Let's turn this characterization into a declarative statement about the dynamic state of the interpreter. In computing some Fibonacci number, the interpreter is to compute the Fibonacci numbers in order, working up from the first one to the needed one. At each stage, the interpreter maintains two kinds of information: both the particular results it has accumulated and its general rules for computation. These general rules describe the inferences that the interpreter may apply in each stage of computation to the particular conclusions it has at that point. That is, these rules identify the additional particular conclusions that the interpreter will have at its disposal in the next stage of computation. In particular, of course, these rules dictate that the interpreter is to derive the next Fibonacci number as the sum of the two previous values it most recently derived.

Modal logic gives us the tools to articulate formally this kind of statement about the state of an interpreter. We can use a modal operator `Memo` to describe the particular conclusions that the interpreter has derived at a certain point, and a modal operator `Rule` to describe the content of the general rules that the interpreter observes in going from one state of deduction to another. Both `Rule` and `Memo` should be S4 modalities, expressing the fact that both kinds of information could only grow as the computation proceeded; and it is convenient to postulate that `Rule` elaborates on `Memo`. With these assumptions, the statement in (48) represents the condition that if the interpreter registered the particular conclusion *p*, then the conclusion *q* would be available at the next step.

(48) $\texttt{Rule}(\texttt{Memo}p \supset q)$

More generally, (49) formalizes the directive to perform an overall computation *all* by immediately deriving *step*, then registering the result *step*, and finally undertaking *rest*.

(49) $\texttt{Rule}(step \wedge \texttt{Rule}(\texttt{Memo}step \supset rest) \supset all)$

If we combine these intuitions to specify the intended search behavior for deriving Fibonacci numbers, we end up with something like the specification in Figure 3.6. The first three clauses describe the derivation of particular Fibonacci numbers. The interpreter begins with the initial pair of Fibonacci numbers explicitly represented; it can derive that the `K`th Fibonacci number is `N` as long as it has explicitly represented the values `L` and `M` of the two previous Fibonacci numbers, and `N` is `L` + `M`.

The next two clauses describe the evolving state of the interpreter during the derivation of a sequence of Fibonacci numbers. According to the first one, the interpreter finishes when the sequence ends. According to the second one, the interpreter process a nonempty sequence `X` beginning with `I` and continuing with `R` by deriving the value `N` of the `I`th Fibonacci number, and, following the pattern illustrated in (48) and (49), registering the value and continuing the computation of `R`.

The remainder of the program outlines the mathematical facts needed to carry things along, and defines a predicate `nlist(X,Z)` true when `Z` is the list of the successive integers from 2 to `X`; this list `Z` defines the sequence for the computation of the first `X` Fibonacci numbers. In other words, the query `nlist(K,L), fiblist(L,A)` succeeds when `A` is a list of the first `K` Fibonacci numbers (save the two initial 1s).

The program in Figure 3.6 is an executable specification (measured in the value of the number `K` of Fibonacci numbers it computes), because each Fibonacci number is calculated

```
Memo fib(0,1).
Memo fib(1,1).
*I *J *K *L *M *N Rule
      (Memo (succ(J,K), succ(I,J),
             fib(I,L), fib(J,M), sum(L,M,N))
      -> fib(K,N)).

Rule fiblist(nil, nil).
*X *Y *I *R *N *A Rule
      (list(I,R,X), list(N,A,Y),
       fib(I,N),
       Rule (Memo fib(I,N) -> fiblist(R,A))
      -> fiblist(X, Y)).

Memo succ(0,1).      Memo succ(1,2).
Memo succ(2,3).      Memo succ(3,4).
Memo succ(4,5).      Memo succ(5,6).
Memo succ(6,7).      Memo succ(7,8).
Memo succ(8,9).      Memo succ(9,10).
Memo succ(10,11).    Memo succ(11,12).

*X *Y ?Z Memo sum(X,Y,Z).
*X *Y ?Z Memo list(X,Y,Z).

*Y Memo rnlist(1,Y,nil).
*I *J *M *N *R *L Memo
      (succ(I,J), succ(M,N),
       rnlist(I,N,R), list(M,R,L)
      -> rnlist(J,M,L)).
*X *Z Memo (rnlist(X,1,Z) -> nlist(X,Z)).
```

Figure 3.6: Memoizing fibonacci program

only once in any proof of `fiblist`. We already have a high-level explanation why, namely that the specification corresponds to a description of the intended behavior of an engine for efficient inference of Fibonacci numbers. A more mechanical explanation may also be illuminating. First, note that the `fiblist` relation is invoked about `K` times in any proof; it is invoked recursively one time for each element in the list `X` (corresponding to each remaining stage of the computation). Each time, it also makes one call to `fib`.

This is where there might be a problem. Recall that the difficulty that arises in executing the specification of Figure 3.4 is that a logic programming interpreter repeatedly backchains against the complex clause for calculating Fibonacci numbers. This backchaining is explosive. In the new specification, such backchaining is impossible. In calculating `fib(K,N)` at world μ, the interpreter turns to a new world $\mu\alpha$ where α is a new transition of the kind described by `Memo`, and looks to prove further `fib` facts at $\mu\alpha$. At this world,

N	Time to find A with fib(N,A), in Linc CPU seconds
0	.01
1	.01
2	.05
3	.13
4	.25
5	.4
6	.6
7	.8
8	1.1
9	1.4
10	1.8
11	2.3
12	2.8

Figure 3.7: Executing the memoized program

`Rule` content is unavailable—the interpreter is to rely only on facts explicitly recognized as part of the computation thus far—and so the complex clause for `fib` cannot be applied recursively. Only the memoized atomic `fib` facts are available here. Using these atomic facts is of course an easy computation.

The relative ease of executing the specification in Figure 3.6 is substantiated by the calculation times reported in Figure 3.7. The times don't increase exponentially, but they grow faster than the linear rate you might at first expect. The reason is that each step involves matches against possible-world terms representing the state of the interpreter. These terms grow linearly in the number of Fibonacci numbers considered; as they grow, the cost of manipulating them correctly in the interpreter also grows.

We can contrast the specification of Figure 3.6 with the more conventional fast specification of the Fibonacci numbers shown in Figure 3.8. I claim that the program of Figure 3.8 encodes essentially the same insight that the program of Figure 3.6 does, but relies on further optimization and encoding to present the insight in a low-level and first-order representation.

In Figure 3.8, the predicate `fstate` records the state of interpreter as a relation among key terms: the previous two Fibonacci numbers computed (the `U`ltimate and `P`enultimate ones), the number of steps of computation remaining `I`, and the final result `R`. The state of the computation is represented by the `fstate` goal currently under consideration. The final pair of clauses describes explicitly how this state evolves. First, when there is nothing left to do, the overall value to be computed is identical to the most recent computed value. At any other stage of the computation, the next Fibonacci number is computed as the sum of the previous two and the number of steps of computation remaining is reduced by one.

```
fib(0,1).
fib(1,1).
*I *K *L *M *N (succ(I,K), fstate(1,1,I,N)
                -> fib(K,N)).
*X *Y fstate(X,Y,0,Y).
*K *L *R *P *U *N (sum(P,U,N), succ(K,L),
                   fstate(U,N,K,R) -> fstate(P,U,L,R)).
```

Figure 3.8: A hacked memoizing Fibonacci program

The resulting new state of computation is encoded as a further `fstate` goal to be derived. The only role of the `fib` predicate itself in this specification is to set up the initial state of this computation.

Thus, like the program of Figure 3.6, this program offers a way of finding Fibonacci numbers efficiently by describing a machine that finds them efficiently. The program of Figure 3.6 describes the state of the machine as an object with content in terms of the operator `Memo`; the entire history of the computation is available at each stage; and update of the state is accomplished by logical operations. In the program of Figure 3.8, only the relevant history of the machine is represented, and that history takes a special, concrete form: the two previous calculated values are stored in a distinguished position. The history is no longer an object with content; it is just an object. This transformation speeds up execution by dispensing with the logic of states (giving in fact an order of magnitude improvement); but this comes at the price of encoding the manipulation of state as an arbitrary concrete transformation on the history between goals.

3.4 Formalities

In the remainder of this chapter, we describe the design of DIALUP formally. Its correctness rests on a variety of proof-theoretic ideas about modal logic and logic programming, which we combine in a novel way. To describe logic programming, we use the idea of uniform proof search described in [Miller *et al.*, 1991] and extended to different kinds of disjunctions in [Miller, 1994; Nadathur and Loveland, 1995]. To facilitate the development of a uniform proof system, we use a lifted, path-based sequent calculus for modal logic that assimilates modal proof to classical proof. This calculus refines and extends the one presented in section 2.3.5, by explicitly accounting for multiple modalities and first-order quantification following particularly [Wallen, 1990; Auffray and Enjalbert, 1992]. We describe this calculus completely in section 3.4.1.

This calculus has the advantage that inferences can be freely interchanged, allowing arbitrary proofs to be transformed easily into goal-directed proofs—we show in Theorem 1, presented in section 3.4.2, how to obtain goal-directed proofs in this calculus. However, as noted already in section 2.3.5, this calculus neither respects nor represents the potential structural modularity of modal inference.

To guarantee the modular behavior of the uniform system, we rely on proof-theoretic analyses of path-based sequent calculi. These analyses establish that path representations enforce modularity and locality in the uses of formulas in proofs, even with otherwise classical reasoning. The foundational step in this analysis is to establish an important general result about lifted modal proof systems, Theorem 2, which is proved in section 3.4.3. The theorem shows how to streamline proofs so that all inferences directly contribute to the proof, and observes some important constraints on labels for possible worlds that such streamlined proofs exhibit.

The operational rules of DIALUP are obtained by transforming the uniform proof system to take advantage of these results; as a consequence, the interpreter can dynamically exploit locality in the use of modular assumptions. The transformation starts in section 3.4.4 by dividing uniform proofs into separate BLOCKS to analyze separate cases. As a preliminary to modularity, we organize these blocks so that each one contains a CANCELLATION whereby the most-recently introduced case contributes to the goal being proved [Loveland, 1991]. Finally, in section 3.4.5, we combine the presence of cancellations and the inherent ability of the modal language to modularly restrict the contributions premises can make (together with the uniformity of proof search and the independence of cases) to derive a final sequent calculus (in Figures 3.11 and 3.12) which specifies the interpreter of a logic programming language.

3.4.1 Modal sequent calculus

We begin by describing how the lifted, explicitly-scoped sequent calculus, as introduced in section 2.3.5, is extended to a multi-modal, first-order modal logic. This extension straightforwardly reflects the model-theoretic changes required to interpret accessibility transitions of multiple types and first-order statements. Our presentation roughly follows [Debart *et al.*, 1992]; we diverge in using a sequent calculus system with interleaved translation in place of their sequential scheme of translation and resolution. (Alas, from a technical point of view, such divergences are somewhat unsatisfying; they leave the sequent calculus less directly grounded in established practice than it perhaps ought to be.)

With the multi-modal logic, atomic modal transitions can have different types depending on the kind of transitions they represent. This requires two elaborations to the maintenance of path terms as characterized in a ground calculus 2.3.5. First, in order to handle non-serial modalities correctly, we need to explicitly enforce a UNIQUE PREFIX PROPERTY on occurrences of variables in path terms (cf. [Wallen, 1990; Auffray and Enjalbert, 1992]). For each eigenvariable μ, we require that there be a term π_μ such that each occurrence of μ in the proof is in a term of the form $\pi_\mu \mu \nu$; π_μ is the unique prefix associated with μ. The unique prefix property means that the equations that arise in proof search describe a TREE in which variables occur uniquely; equating terms means identifying the nodes the terms designate. In fact, the unique prefix property may be imposed on lifted proofs as well—describing both logic variables and Skolem terms—without loss of generality. In

$$\Sigma, \mu : i \triangleright \mu : i \;\; (\textsc{ax}_t)$$

$$\frac{\Sigma \triangleright \mu : i \qquad \Box_j A \supset \Box_i A \;\; (\textsc{inc})}{\Sigma \triangleright \mu : j} \; (\textsc{inc}_t)$$

$$\frac{\Box_i A \supset A \;\; (\textsc{ver})}{\Sigma \triangleright \epsilon : i} \; (\textsc{ver}_t)$$

$$\frac{\Sigma \triangleright \mu : i \quad \Sigma \triangleright \nu : i \quad \Box_i A \supset \Box_i \Box_i A \;\; (\textsc{pi})}{\Sigma \triangleright \mu\nu : i} \; (\textsc{pi}_t)$$

Figure 3.9: Deriving the judgment $\Sigma \triangleright \nu : i$.

some frameworks, the unique prefix property is observed as a meta-theorem; but for our purposes it is convenient to insist on it from the start.

The second elaboration involves assigning types to transitions and sequences of transitions. Sequences are assigned types based on the elements of which they are composed. We will indicate these type restrictions by associating each sequent with a set of auxiliary premises of the form $\alpha : i$ where α represents an atomic modal transition and i names a modal operator. These premises can be used to derive JUDGMENTS that more complex scope representations have particular types. For scope terms, the judgment $\Sigma \triangleright \nu : i$ indicates that ν describes a transition of i-accessibility, according to the auxiliary premises Σ. Such judgments are derived according to the rules shown in Figure 3.9, which realize axioms (INC), (VER) and (PI) as rules of inference amalgamating and reclassifying terms. We can obtain a ground explicitly-scoped sequent calculus for propositional multi-modal logic by adding these specifications to sequents. Initially, the specification is empty.[2] $(\rightarrow \Box_i)$ and $(\Diamond_i \rightarrow)$ will introduce a fresh variable α and add the specification $\alpha : i$; then, $(\Box_i \rightarrow)$ and $(\rightarrow \Diamond_i)$ require a judgment $\mu : i$ to be derived from the attached specification.

To endow the system with first-order reasoning, we also need to keep track of the worlds that give the domains for first-order terms. For this, we associate sequents with a set of premises of the form $a : \mu$, where a is a first-order eigenvariable and μ gives the world where we first assume a has a value. Similarly, judgments of the form $\Sigma \triangleright t : \mu$ indicate that the first-order term t is available in scope μ, and are determined by the following definition:

Definition 1 $\Sigma \triangleright t : \mu$ *if and only if for every free variable x that occurs in t with an*

[2]Because this specification is empty and because we insist on the unique prefix property, the treatment of non-serial modalities is correct. This departs from [Debart *et al.*, 1992] but effectively follows [Wallen, 1990]. With non-serial modalities, we may have $\Box p$ at μ without having $\Diamond p$ there, because no worlds are accessible. But the rules of Figure 3.9 correctly fail to allow a proof of $\Box ip \longrightarrow \Diamond ip$ except if (VER) holds. In the absence of any typing premises, no modal term can be shown to have type i. Therefore, neither $(\Box \rightarrow)$ or $(\rightarrow \Diamond)$ rules apply. In fact, the treatment of serial but not veridical modalities requires the inference rules of to be extended so that we can introduce arbitrary fresh transitions $\alpha_i : i$, to permit the application of necessary facts at any point in the proof.

assignment $x : \nu \in \Sigma$, ν *is a prefix of* μ.

The condition that ν be a prefix of μ implements the increasing domain constraint. The ground $(\rightarrow \forall)$ and $(\exists \rightarrow)$ rules introduce a new variable x and a declaration $x : \mu$ where μ is the label of the principal formula of the rule. The corresponding $(\rightarrow \exists)$ and $(\forall \rightarrow)$ rules require deriving a judgment $t : \mu$ from the available declarations before a term t can be substituted into a principal formula at world μ.

Again, the demands of efficient, general inference require us to work not in this ground system, which implements ground inference on the classical semantics, but in a lifted system which uses logic variables, Skolemization and unification. Here too, the lifted system can be constructed correctly and automatically from the ground system by applying a proof-theoretic version of Herbrand's theorem, as in [Lincoln and Shankar, 1994]. As in the basic lifted system of section 2.3.5, the inference rules describe derivations or proof attempts; derivations are associated with constraints that must be solved to obtain a proof. Now derivations are also associated with declarations of types that must be respected in solving the constraints. In particular, logic variables are declared with the types that their values must have, while Skolem terms are declared with the types belonging to the corresponding eigenvariables. A substitution is possible only if the value assigned to each logic variable can be assigned to the type associated with the logic variable using the declarations of Skolem terms as premises Σ and the rules in Figure 3.9.

Thus, now sequents take the form

$$\Sigma; C/\Sigma'; C' \triangleright \Gamma \longrightarrow \Delta$$

The sequent shows not only the formulas Γ and Δ and the initial constraints C which increase over the course of the subderivation to C', but also the initial declarations Σ of types for modal terms and domains for first-order terms, which increase to Σ' over the course of the subderivation. (The sequent suppresses the association between formulas and free variables used in the ongoing process of Skolemization and translation.)

For example, the full rule $(\rightarrow \Box_i)$ is annotated this way:

$$\frac{\Sigma, \alpha(X) : i; C/\Sigma'; C' \triangleright \Gamma \longrightarrow A_X^{\mu;\alpha(X)}, \Box_{i\alpha} A_X^{\mu}, \Delta}{\Sigma; C/\Sigma'; C' \triangleright \Gamma \longrightarrow \Box_{i\alpha} A_X^{\mu}, \Delta} \rightarrow \Box_i$$

This rule encodes in two places the bookkeeping required to prove $\Box_i A$ by introducing a Skolem term for a transition to a fresh possible world and proving A there.

The dense notation involved in this strategy is a drawback, however. In fact, the underlying state of proof search is easily recovered from logical operations and the correct update of state is easily dealt with in proof transformations. Accordingly, for the remainder of this chapter, we will use abbreviations that leave these details implicit, such as these:

$$\Gamma, A^{\mu} \longrightarrow B^{\nu}, \Delta \ \ (A^{\mu} = B^{\nu})$$

$$\frac{\Gamma \longrightarrow A^{\mu;\alpha(X)}, \Box_i A^{\mu}, \Delta}{\Gamma \longrightarrow \Box_i A^{\mu}, \Delta} \rightarrow \Box_i \ (\text{SK } \alpha(X) : i)$$

(SK indicates Skolemization; LV, the introduction of a logic variable.) Using these abbreviations, we give in Figure 3.10 the sequent calculus SCL from which we start constructing DIALUP by proof-theoretic analysis. Note that, because we base proof search on a cut-free sequent calculus, we can dispense with inference rules for connectives that lie outside the DIALUP fragment (cf. (43) in section 3.1.2).

The proofs of this section require a number of easy consequences of the simple structure of SCL. These consequences continue to hold, suitably adapted, for the intermediate proof systems that we will construct from SCL in later sections (including SCLA and SCLB). Let us adopt some terminology from [Kleene, 1951] and then present these results.

A (PROPER) DERIVATION is a tree of sequents derived from initial (or axiom) sequents according to the rules of Figure 3.10 (or whatever sequent calculus is under discussion). A tree similarly constructed except for containing some arbitrary sequent *S* as a leaf is a DERIVATION FROM *S*.

Each formula occurrence in the immediately higher sequent is traceable to a particular occurrence in the end-sequent, and so we can identify for each formula occurrence in any higher sequent (or the end-sequent itself) its ANCESTOR in the end-sequent.

Lemma 1 (weakening) *For any SCL derivation $\mathcal{D}$ with end-sequent*

$$\Gamma \longrightarrow \Delta$$

we can obtain another SCL derivation $\mathcal{D}'$ with end-sequent

$$\Gamma, \Gamma' \longrightarrow \Delta$$

that differs from the end-sequent of $\mathcal{D}$ only in adding the formulas Γ' to the left-hand side of each sequent in $\mathcal{D}$; we can likewise add additional conclusions to Δ.

Lemma 2 (contraction) *For any SCL derivation $\mathcal{D}$ with end-sequent*

$$\Gamma, A^{\mu}, A^{\mu} \longrightarrow \Delta$$

we can obtain another SCL derivation $\mathcal{D}'$ with end-sequent

$$\Gamma, A^{\mu} \longrightarrow \Delta$$

that differs from the end-sequent of $\mathcal{D}$ only in eliminating one occurrence of A^{μ} on the left throughout $\mathcal{D}$; we can likewise eliminate duplicate formulas from Δ.

$$\Gamma, A^{\mu} \longrightarrow B^{\nu}, \Delta \quad (A^{\mu} = B^{\nu})$$

$$\frac{\Gamma, A \wedge B^{\mu}, A^{\mu}, B^{\mu} \longrightarrow \Delta}{\Gamma, A \wedge B^{\mu} \longrightarrow \Delta} \wedge \rightarrow$$

$$\frac{\Gamma \longrightarrow A^{\mu}, A \wedge B^{\mu}, \Delta \qquad \Gamma \longrightarrow B^{\mu}, A \wedge B^{\mu}, \Delta}{\Gamma \longrightarrow A \wedge B^{\mu}, \Delta} \rightarrow \wedge$$

$$\frac{\Gamma \longrightarrow A^{\mu}, B^{\mu}, A \vee B^{\mu}, \Delta}{\Gamma \longrightarrow A \vee B^{\mu}, \Delta} \rightarrow \vee$$

$$\frac{\Gamma, A \vee B^{\mu}, A^{\mu} \longrightarrow \Delta \qquad \Gamma, A \vee B^{\mu}, B^{\mu} \longrightarrow \Delta}{\Gamma, A \vee B^{\mu} \longrightarrow \Delta} \vee \rightarrow$$

$$\frac{\Gamma, A^{\mu} \longrightarrow B^{\mu}, A \supset B^{\mu}, \Delta}{\Gamma \longrightarrow A \supset B^{\mu}, \Delta} \rightarrow\supset$$

$$\frac{\Gamma, A \supset B^{\mu} \longrightarrow A^{\mu}, \Delta \qquad \Gamma, A \supset B^{\mu}, B^{\mu} \longrightarrow \Delta}{\Gamma, A \supset B^{\mu} \longrightarrow \Delta} \supset\rightarrow$$

$$\frac{\Gamma, \Box_i A^{\mu}, A^{\mu x} \longrightarrow \Delta}{\Gamma, \Box_i A^{\mu} \longrightarrow \Delta} \Box_i \rightarrow (\text{LV } x : i)$$

$$\frac{\Gamma \longrightarrow A^{\mu\alpha(X)}, \Box_i A^{\mu}, \Delta}{\Gamma \longrightarrow \Box_i A^{\mu}, \Delta} \rightarrow \Box_i \ (\text{SK } \alpha(X) : i)$$

$$\frac{\Gamma, A[u/x]^{\mu}, \forall x.A^{\mu} \longrightarrow \Delta}{\Gamma, \forall x.A^{\mu} \longrightarrow \Delta} \forall \rightarrow (\text{LV } u : \mu)$$

$$\frac{\Gamma \longrightarrow A[f(X)/x]^{\mu}, \forall x.A^{\mu}, \Delta}{\Gamma \longrightarrow \forall x.A^{\mu}, \Delta} \rightarrow \forall \ (\text{SK } f(X) : \mu)$$

$$\frac{\Gamma, \exists x.A^{\mu}, A[f(X)/x]^{\mu} \longrightarrow \Delta}{\Gamma, \exists x.A^{\mu} \longrightarrow \Delta} \exists \rightarrow (\text{SK } f(X) : \mu)$$

$$\frac{\Gamma \longrightarrow A[u/x]^{\mu}, \exists x.A^{\mu}, \Delta}{\Gamma \longrightarrow \exists x.A^{\mu}, \Delta} \rightarrow \exists \ (\text{LV } u : \mu)$$

Figure 3.10: SCL. Modal sequent calculus using unification and dynamic Skolemization (abbreviated)

(In later systems, the weakening transformation need only add formulas up to rules which discard formulas; in later cases, the contraction transformation can only collapse formulas with the same status in the sequent.)

Lemma 3 (monotonicity) *For any SCL derivation $\mathcal{D}$ with end-sequent written in full as*

$$\Sigma/\Sigma'; C/C' \triangleright \Gamma \longrightarrow \Delta$$

for any set Σ_1 containing only elements of Σ and list C_1 containing only elements of C, we can obtain a derivation (like $\mathcal{D}$) with end-sequent:

$$\Sigma_1/\Sigma_1'; C_1/C_1' \triangleright \Gamma \longrightarrow \Delta$$

where: Σ_1' contains all the elements of Σ_1 and only elements of Σ'; and C_1' contains all the elements of C_1 and only elements of C'.

Proof. Straightforward induction on the structure of derivations. ■

Definition 2 *A derivation in which no two rules introduce the same logic variable is a* pure-variable *derivation.*

Lemma 4 *Every SCLB derivation can be transformed into a pure-variable derivation; any unifier solving the equations associated with the first can be transformed to a unifier solving the equations associated with the second.*

The argument that shows this is summarized informally as follows. We can rewrite any derivation so that all occurrences of a logic variable y that are due to the action of a particular inference are replaced by a fresh variable z (that does not occur elsewhere in the proof). By this process, any rule involving a duplicate variable can be rewritten so as to create a derivation in which the number of rules involving duplicate variables is decreased by one. Induction allows all the duplicate variables to be eliminated.

This rewriting involves substituting z for occurrences of y at positions corresponding to the location of the bound variable in the principal formula of the inference introducing y in all formulas with the side formula of that inference as an ancestor. Substitutions are also required appropriately in the record of variables maintained for dynamic Skolemization, in the record of equations derived during the proof, and in the specification of types for variables. (Because the latter two are sets, and the original proof may duplicate its constraints on y, this step could involve adding constraints on z while also preserving those formerly placed on y.) The unifier for the resulting derivation is obtained for the unifier for the original derivation by assigning z the same value given to y. ■

We can also observe that this regime permits the abbreviation of goal occurrences of $\Box(A \supset B)$ by a single formula $(A > B)$ and the consolidation of corresponding inferences

$(\rightarrow \Box)$ and $(\rightarrow\supset)$ into a single figure $(\rightarrow>)$:

$$\cfrac{\Gamma, A^{\mu\alpha(X)} \longrightarrow B^{\mu\alpha(X)}, A > B^{\mu}, \Delta}{\Gamma \longrightarrow A > B^{\mu}, \Delta} \rightarrow> (\text{SK } \alpha(X) : i)$$

This corresponds to the pair of inferences:

$$\cfrac{\cfrac{\Gamma, A^{\mu\alpha(X)} \longrightarrow B^{\mu\alpha(X)}, A \supset B^{\mu\alpha(X)}, A > B^{\mu}, \Delta}{\Gamma \longrightarrow A \supset B^{\mu\alpha(X)}, A > B^{\mu}, \Delta} \rightarrow\supset}{\Gamma \longrightarrow \Box(A \supset B)^{\mu}, \Delta} \rightarrow \Box\, (\text{SK } \alpha(X) : i)$$

Derivations using the new rule can be transformed to derivations using the old rule by substituting this pattern for the new rule and weakening the higher derivation by $A \supset B^{\mu\alpha}$. Meanwhile, derivations using the old rule can be transformed to use the new rule by eliminating appropriate instances of $(\rightarrow \Box)$ (and hence also eliminating its side formula in higher derivations). This elimination proceeds by replacing higher instances of $(\rightarrow\supset)$ that apply to the side formula of the $(\rightarrow \Box)$ rule by applications of $(\rightarrow>)$ to the principal formula of $(\rightarrow \Box)$. (More formally, the replaced rules apply to formulas identical to the side formula of the lower $(\rightarrow \Box)$ which also have this formula as an ancestor.) This is possible because the preservation of formulas in sequents ensures that a copy of the principal formula is available, and because side formulas of the two rules are the same, and is complete because axioms apply only to atomic formulas.

3.4.2 Uniform proofs and eager proofs

[Miller, 1994] uses Definition 3 to characterize ABSTRACT LOGIC PROGRAMMING LANGUAGES.

Definition 3 *A cut-free sequent proof $\mathcal{D}$ is* uniform *if for for every subproof $\mathcal{D}'$ of $\mathcal{D}$ and for every non-atomic formula occurrence B in the right-hand side of the end-sequent of $\mathcal{D}'$ there is a proof $\mathcal{D}''$ that is equal to $\mathcal{D}'$ up to a permutation of inference rules and is such that the last inference rule in $\mathcal{D}''$ introduces the top-level logical connective of B.*

Definition 4 *A logic with a sequent calculus proof system is an* abstract logic programming language *if restricting to uniform proofs does not lose completeness.*

It is easy to show that the sequent calculus SCL is an abstract logic programming language in this sense. In fact, as we shall see, by this definition EVERY SCL derivation is uniform.

More precisely, each SCL derivation can be transformed into an EAGER derivation, which we can define in terms of the transformations of sequent derivations described in [Kleene, 1951]. Because of the automatic copying of formulas in this calculus, adjacent logical inferences are applied in succession. A higher inference H is applied at the root of the immediate subderivation of a lower inference L. (No structural rules intervene.) If a side formula of L is not the principal formula of H, we may attempt to replace the derivation of the end-sequent of L by a new derivation of the same end-sequent with H at the root, followed immediately by L, capped by subderivations copied from the original derivation (but possibly weakened to reflect the availability of additional logical premises, and the earlier introduction of terms and the earlier impositions of equalities). Performing such a replacement constitutes an interchange of rules L and H and demonstrates the permutability of L and H; see [Kleene, 1951]. The calculus of Figure 3.10 is formulated so that no rules impose structural conditions on the sequents where they apply (as substantiated by Lemmas 1, 2 and 3) and therefore any such pair of inferences may be exchanged in this way.

Now consider a derivation $\mathcal{D}$ containing a right rule R.

Definition 5 *R is* delayed *exactly when there is a subderivation $\mathcal{D}'$ of $\mathcal{D}$ that contains R and ends in a left rule L, such that the ancestor of the principal formula of R in the end-sequent of $\mathcal{D}'$ is an occurrence of the principal formula of R.*

Intuitively, R is delayed in that we have waited in $\mathcal{D}$ to apply R until after consulting the program by applying L, when we might have applied R earlier.

Definition 6 *D is* eager *exactly when it contains no delayed applications of right rules.*

By transforming any derivation $\mathcal{D}$ into an eager derivation $\mathcal{D}'$ by permutations of inferences, we effectively witness that $\mathcal{D}$ is uniform and provide a starting point for further analysis.

Theorem 1 *Any SCL derivation $\mathcal{D}$ is equal to an eager derivation $\mathcal{D}'$ up to permutations of inferences.*

The **proof** follows [Kleene, 1951, theorem 2]. A double induction transforms each derivation into an eager one; the inner induction rectifies the final rule of a derivation whose subderivations are eager by an interchange of inferences and induction [Kleene, 1951, Lemma 10]; the outer one rectifies a derivation by rectifying the furthest violation from the root and induction.

We can present this proof in full using a lemma.

Lemma 5 *Consider a derivation $\mathcal{D}$ which has eager immediate subderivations and which ends in rule application A. From $\mathcal{D}$ we can construct a derivation with the same end-sequent that is eager.*

Proof. If the rule A that ends $\mathcal{D}$ is a right rule, we are done. A cannot be delayed in $\mathcal{D}$, nor can any rule not delayed in the immediate subderivations of $\mathcal{D}$ be delayed in $\mathcal{D}$. Otherwise, we have A a left rule. We proceed after [Kleene, 1951], lemma 10. Define the GRADE of $\mathcal{D}$ as the number of delayed right rule applications in $\mathcal{D}$. We show by induction on the grade that any derivation can be transformed to an eager one.

The base case is derivations of grade 0. This case has $\mathcal{D}$ itself eager. Thus, suppose the lemma holds for derivations of grade g, and consider $\mathcal{D}$ of grade $g + 1$. Since the grade is nonzero, and the immediate subderivations are eager, at least one of them—call it $\mathcal{D}'$—must end with a right rule R whose principal formula is preserved from the end-sequent of $\mathcal{D}'$ to the end-sequent of $\mathcal{D}$.

(If an eager derivation ends in a left rule, then the end-sequent contains no identical ancestor to the principal formula of a right rule. All rules add structure to the ancestors of formulas, so we cannot change this property by applying further rules to the derivation.)

We interchange rules R and A. In the result, the subderivation(s) ending in A satisfy the condition of the lemma with grade g or less. By applying the induction hypothesis, we can replace these subderivations with eager ones. Since in this last transformed derivation, the immediate subderivation(s) are eager and the last rule is a right rule, we have an overall eager derivation. ■

Now, continuing the proof of Theorem 5, define the RELUCTANCE of $\mathcal{D}$ to be the number of rule applications R such that $\mathcal{D}_R$ is not eager. We proceed by induction on reluctance. If reluctance is zero, $\mathcal{D}$ is itself eager.

Now suppose the theorem holds for derivations of reluctance d, and consider $\mathcal{D}$ of reluctance $d+1$. Since $\mathcal{D}$ is finite, there must be some subderivation $\mathcal{D}_R$ that is not eager and such that every subderivation $(\mathcal{D}_R)_{R'}$ is eager. This $\mathcal{D}_R$ satisfies the condition of lemma 5. Therefore this $\mathcal{D}_R$ can be replaced with a corresponding eager derivation, giving a new derivation of smaller reluctance. Induction then shows that the resulting derivation can be made eager. ■

3.4.3 Linking and Variable Introduction

Eager derivations do not make a satisfactory specification for a logic programming interpreter because they do not embody a very focused search strategy. This shows up in two ways. First, eager derivations are free to work in parallel on different disjuncts of a goal; in logic programming we want SEGMENTS in which a single goal is in force. Moreover, eager derivations can reuse work across separate case analyses; in logic programming we want BLOCKS where particular cases are investigated separately. Second, because of their classical formulation, eager derivations do not enforce or exploit the modularity of their underlying logic.

In subsequent sections, we will remedy these faults of eager derivations. To do so, we need to characterize more precisely derivations both in SCL and in the related formalisms that we will derive from it in subsequent sections. This section provides this characterization. We show that SCL derivations need only apply inferences when those inferences contribute to the proof, in this technical sense:

Definition 7 *An inference R* contributes *to a derivation $\mathcal{D}$ iff some side formula A of R is a the ancestor of a principal formula of an axiom in $\mathcal{D}$.*

At the same time, we show that in an SCL derivation all of whose inferences contribute to the proof, the proof respects a VARIABLE INTRODUCTION PROPERTY that describes the restricted way in which variables are propagated in the proof. We define this as follows. Let C be the list of annotation equations resulting from a SCL proof attempt $\mathcal{D}$, numbered in increasing order. (Recall that equations in C generated by axioms in right branches of a proof PRECEDE those on left branches.) Denote the term in C_i coming from the right formula of the ith axiom as r_i and that coming from the left formula as l_i.

Definition 8 *$\mathcal{D}$ has the* variable introduction property *if and only if every variable x that is introduced by a $(\Box \rightarrow)$ rule in $\mathcal{D}$ and occurs in some term r_i also occurs in some l_j for some $j < i$.*

First we note the following crude sense in which every rule should bring something new to the proof.

Lemma 6 (simplicity) *For any SCL derivation $\mathcal{D}$, there is another $\mathcal{D}'$ with the same end-sequent such that every rule-application in $\mathcal{D}'$ has a different principal formula or different side-formulas from every lower rule-application.*

Proof. We can eliminate any higher application identical to some lower one using the contraction lemma: observe that the side formulas of the lower application are available by preservation, and thus the side formulas of the higher application are duplicates. ■

Now we state and prove the main result. Because of its importance, we provide a sample illustration of some technical details we will generally pass over later: We refer explicitly to constraints and specifications accumulated during the proof. We also consider the rules of SCL directly, to make it clear that the result applies to this fragment of modal logic more generally, irrespective of logic programming considerations.

Theorem 2 (variable introduction) *Given any SCL derivation $\mathcal{D}$ with end-sequent*

$$\Sigma/\Sigma'; C/C' \triangleright \Gamma \longrightarrow \Delta$$

we construct a derivation $\mathcal{D}'$ with end-sequent

$$\Sigma/\Sigma''; C/C'' \triangleright \Gamma' \longrightarrow \Delta$$

with the following properties: Σ'' contains only elements from Σ'; C'' contains only elements from C'; Γ' contains only elements from Γ; and $\mathcal{D}'$ contains only inferences from $\mathcal{D}$ (and does not interchange them); every formula in Γ' is the ancestor of the principal formula of some axiom in $\mathcal{D}'$; every inference in $\mathcal{D}'$ contributes to it; and $\mathcal{D}'$ enjoys the variable introduction property.

Proof. Consider an SCL derivation $\mathcal{D}$ with end-sequent:

$$\Sigma/\Sigma'; C/C' \triangleright \Gamma \longrightarrow \Delta.$$

We say a formula A^{μ} is GROUNDED in $\mathcal{D}$ if A^{μ} occurs in Γ and there is some formula B^{ν} in Δ such that μ is a prefix of ν. If A^{μ} occurs in Γ but is not grounded, we say A^{μ} is

UNGROUNDED in $\mathcal{D}$. If an occurrence of an equation $l = r$ appears in C' but not in C, we say $\mathcal{D}$ GIVES RISE to C.

Induction on the structure of $LU\Box$ proof attempts shows that there is a proof attempt $\mathcal{D}'$ corresponding to $\mathcal{D}$ satisfying the conditions of the statement of the theorem, and where one of two further properties holds of each ungrounded A^μ in $\mathcal{D}$. Either (1) $\mathcal{D}'$ gives rise to no equation $l_i = r_i$ in which μ is a prefix of l_i and A^μ does not occur in the end-sequent of $\mathcal{D}'$; or (2), the end-sequent of $\mathcal{D}'$ includes A^μ, and $\mathcal{D}'$ contains a FIRST LEFT USE OF μ—in other words $\mathcal{D}'$ gives rise to some equation $l_j = r_j$ where μ is a prefix of l_j and μ is not a prefix of any right equation term r_i with $i \leq j$. The intuition behind these conditions is that any problematic formula that starts out ungrounded in $\mathcal{D}$ should disappear in $\mathcal{D}'$ unless it makes a contribution.

For the base case, we start from an instance of the axiom rule:

$$\Sigma/\Sigma; C/C, A = B, \mu = \nu \rhd \Gamma, A^\mu \longrightarrow B^\nu, \Delta$$

We construct the axiom:

$$\Sigma/\Sigma; C/C, A = B, \mu = \nu \rhd A^\mu \longrightarrow B^\nu, \Delta$$

It is immediate that the new axiom satisfies the condition of the theorem. The auxiliary conditions also hold—since either A^μ is grounded or μ is used here first on the left.

Now suppose the claim is true for derivations of height h or less, and consider derivations of height h+1. The transformation depends on the inference rule at the root of the derivation. We consider the case of $(\supset\rightarrow)$ in detail as a key illustration. Consider a derivation $\mathcal{D}$ ending in $(\supset\rightarrow)$, as below:

$$\frac{\Sigma'/\Sigma''; C'/C'' \rhd \Gamma, (A \supset B)^\mu \longrightarrow A^\mu, \Delta \qquad \Sigma/\Sigma'; C/C' \rhd \Gamma, (A \supset B)^\mu, B^\mu \longrightarrow \Delta}{\Sigma/\Sigma''; C/C'' \rhd \Gamma, (A \supset B)^\mu \longrightarrow \Delta} \supset\rightarrow$$

The same multiset of formulas Δ appears above the rule-application in the right subderivation and below the rule-application. Hence the ungrounded formulas in the whole derivation are all ungrounded in right subderivation. (The same property holds for right rules that do not change annotation, even though they add a formula to Δ.) We apply the induction hypothesis to the RIGHT subderivation; we thereby eliminate or find first left uses for all these ungrounded formulas.

In particular, if our principal formula $(A \supset B)^\mu$ is ungrounded, either we will eliminate it and its side formula in the subderivation, or we will find a first left use for all formulas labeled by a prefix of μ. If we eliminate it, we obtain a new subderivation, in which the end-sequent is of a form

$$\Sigma/\Sigma_1; C/C_1 \rhd \Gamma_1 \longrightarrow \Delta$$

in which neither $(A \supset B)^\mu$ nor its subformula appears. The monotonicity lemma ensures that Σ_1 contains only elements of Σ'' and that C_1 contains only elements of C'', so the new subderivation satisfies the needed conditions.

Otherwise, the principal formula is preserved in the right subderivation. In this case, we apply the induction hypothesis to the left subderivation—observing that it applies only to those ungrounded formulas in the overall derivation that are NOT labeled by prefixes of μ. We apply the weakening lemma to each new derivation and the formulas that appear in the other new derivation but not in it. The two subderivations then agree on a multiset Γ_2 of formulas that survive. These two derivations can be combined into the needed overall derivation using $(\supset\rightarrow)$, as below:

$$\dfrac{\Sigma_1/\Sigma_2; C_1/C_2 \triangleright \Gamma_2, (A \supset B)^{\mu}_X \longrightarrow A^{\mu}_X, \Delta \qquad \Sigma/\Sigma_1; C/C_1 \triangleright \Gamma_2, (A \supset B)^{\mu}_X, B^{\mu}_X \longrightarrow \Delta}{\Sigma/\Sigma_2; C/C_2 \triangleright \Gamma_2, (A \supset B)^{\mu}_X \longrightarrow \Delta} \supset\rightarrow$$

This derivation satisfies the conditions of the theorem. The weakening step does not introduce rules, constraints, or specifications, nor formulas foreign to Γ. Annotations of formulas weakened onto either subderivation contribute to this new overall deduction and are first used in equations on the left here, by the induction hypothesis (these annotations have a first left use in one subderivation and no use in the other). Meanwhile, for any ungrounded formula C^{ν}_Y where ν occurs in equations from both new subderivations, ν has a first left use—even with ν a prefix of μ, if applicable. For, in the new B subderivation, there is a first equation-term involving ν on the left; this term will precede all equation-terms involving ν from the A subderivation as well. The monotonicity lemma again ensures that Σ_2 and C_2 contain only elements of Σ'' and C''.

Similar reasoning takes care of the five right rules that do not alter annotations—$(\rightarrow \wedge)$, $(\rightarrow \vee)$, $(\rightarrow\supset)$, $(\rightarrow \forall)$, and $(\rightarrow \exists)$—as well as the five left cases that do not alter annotations—$(\wedge \rightarrow)$, $(\vee \rightarrow)$, $(\supset\rightarrow)$, $(\forall \rightarrow)$, and $(\exists \rightarrow)$.

Two cases remain. First, suppose the proof attempt ends in $(\rightarrow \Box_i)$:

$$\dfrac{\Sigma, \alpha(X) : i/\Sigma'; C/C' \triangleright \Gamma \longrightarrow \Box_i A^{\mu}, A^{\mu\alpha(X)}, \Delta}{\Sigma/\Sigma'; C/C' \triangleright \Gamma \longrightarrow \Box_i A^{\mu}, \Delta} \rightarrow \Box_i$$

Observe that the ungrounded formulas in the immediate subderivation are exactly those that are ungrounded in the overall derivation. For, consider any ungrounded formula B^{ν} in the overall derivation. By definition, ν is not a prefix of μ. Thus, the only way we could could have ν a prefix of $\mu\alpha(X)$ is if $\nu = \mu\alpha(X)$. Now, $\alpha(X)$ is a unique Herbrand function application associated with this occurrence of the formula $\Box_i A$. Since labels are preserved or extended by all sequent rules, by monotonicity, if $\nu = \mu\alpha(X)$ then B must be a descendant of a lower occurrence of $A^{\mu\alpha(X)}$. By Lemma 6, we may assume this is not so without loss of generality.

So the induction hypothesis applies to the subderivation with the same ungrounded formulas. Assuming the side formula of this inference is preserved there, weakening by $\Box_i A^{\mu}$ if necessary and applying $(\rightarrow \Box)$ to the result (as below) gives a derivation with the needed properties:

$$\frac{\Sigma, \alpha(X) : i/\Sigma_1; C/C_1 \triangleright \Gamma_1 \longrightarrow \Box_i A^\mu, A^{\mu\alpha(X)}, \Delta}{\Sigma/\Sigma_1; C/C_1 \triangleright \Gamma_1 \longrightarrow \Box_i A)^\mu, \Delta} \rightarrow \Box_i$$

Finally, consider $(\Box_i \rightarrow)$:

$$\frac{\Sigma, x : i/\Sigma'; C/C' \triangleright \Gamma, \Box_i A^\mu, A^{\mu x} \longrightarrow \Delta}{\Sigma/\Sigma'; C/C' \triangleright \Gamma, \Box_i A^\mu \longrightarrow \Delta} \Box_i \rightarrow$$

Here x is a logic variable which by Lemma 4 we may assume is different from other variables and Herbrand terms. In the immediate subderivation therefore, $A^{\mu x}$ is ungrounded, because μx cannot be the prefix of the annotation of any Δ formula. Apply the induction hypothesis. If $A^{\mu x}$ disappears, the subderivation suffices. Otherwise, use the new subderivation to construct a derivation ending:

$$\frac{\Sigma, x : i/\Sigma_1; C/C_1 \triangleright \Gamma_1, \Box_i A^\mu, A^{\mu x} \longrightarrow \Delta}{\Sigma/\Sigma'; C/C' \triangleright \Gamma_1, \Box_i A^\mu \longrightarrow \Delta} \Box \rightarrow$$

Here, μx—along with all surviving ungrounded formulas—appears last on a left equation, by the induction hypothesis. Thus the overall deduction has the variable introduction property and witnesses the needed properties of the ungrounded formulas. ■

3.4.4 Block structure

We can now reformulate the rules of the sequent calculus so as to preserve provability but introduce the block and segment structure characteristic of logic programming. After this transformation, the proof system treats the work done in decomposing a goal or backward chaining against a clause as temporary. The work contributes to the current problem in a fixed way, and then is discarded.

We begin with a trick that for now is purely formal—introducing an ARTICULATED SCL. We represent assumptions as a pair $\Pi; \Gamma$ with Π encoding the global program and Γ encoding local clauses; eventually local clauses will be processed only in the current segment and then discarded. Similarly, we represent goals as a pair $\Delta; \Theta$, with Θ encoding the restart goals and Δ encoding the local goals (eventually Δ will be discarded between segments). With this representation, principal formulas of logical rules are local formulas (in Γ and Δ)—and with these exceptions so are the side formulas: the $(\rightarrow \Box)$ and $(\rightarrow >)$ rules augment Π instead of Γ (when they add a new program clause) and Θ in addition to Δ (when they add new restart goals, but continue to be processed).

New (decide) and (restart) rules keep this change general; they allow a global formula—

a program clause or restart goal—to be selected and added to the local state.

$$\frac{\Pi, P^\mu; \Gamma, P^\mu \longrightarrow \Delta; \Theta}{\Pi, P^\mu; \Gamma \longrightarrow \Delta; \Theta}\text{(decide)} \qquad \frac{\Pi; \Gamma \longrightarrow \Delta, G^\mu; \Theta, G^\mu}{\Pi; \Gamma \longrightarrow \Delta; \Theta, G^\mu}\text{(restart)}$$

Lemma 7 (articulation) *Every ordinary SCL deduction can be converted into an articulated SCL deduction with an end-sequent of the form* $\Pi; \longrightarrow; \Theta$ *(i.e, in which the local parts* Γ *and* Δ *are empty) in such a way that if the initial derivation is eager then so is the resulting derivation (and vice versa).*

The **proof** forward argues by straightforward structural induction that the derivation can be transformed assuming each formula in the end-sequent is allocated somehow either to Π or Γ; we preserve and extend this allocation in immediate subderivations, introducing instances of (decide) and (restart) as necessary when the principal formula occurs in Π only; then argue by induction. Backward, another straightforward structural induction shows we return to SCL by forgetting the distinction between Π and Γ, forgetting the (decide) and (restart) rules, and contracting copied formulas. ∎

To give content to the splitting of sequents, we first introduce new rules for $(\vee \rightarrow)$. In these new rules, local work (and perhaps some global work) is discarded in the subderivation written on the right; this subderivation is said to begin a new BLOCK of the derivation separate from the block in which the rule is applied. (Blocks of a derivation are thus derivations from the end-sequents of right subderivations of the revised $(\vee \rightarrow)$ inferences.) This discarding normally means that while each block of a derivation is eager, the derivation as a whole is not eager. A derivation that meets this weaker condition is called BLOCKWISE EAGER. As observed in [Nadathur and Loveland, 1995], derivations with blocks can nevertheless be seen as eager throughout by reconstructing the (restart) rule as backchaining against the negation of a subgoal.

$$\frac{\Pi; \Gamma, A \vee B^\mu, A^\mu \longrightarrow \Delta; \Theta \qquad \Pi', B^\mu; \longrightarrow; \Theta'}{\Pi; \Gamma, A \vee B^\mu \longrightarrow \Delta; \Theta} \vee \rightarrow_L$$

$$\frac{\Pi; \Gamma, A \vee B^\mu, B^\mu \longrightarrow \Delta; \Theta \qquad \Pi', A^\mu; \longrightarrow; \Theta'}{\Pi; \Gamma, A \vee B^\mu \longrightarrow \Delta; \Theta} \vee \rightarrow_R$$

(Formally, Π' is a sub-multiset of Π; Θ' is a sub-multiset of Θ.) Having both rules introduces the apparent ambiguity of which to apply in proof search. The ambiguity is resolved by a policy of enforcing CANCELLATIONS—immediate uses in the derivation of any disjunct introduced (cf. [Loveland, 1991], but note restriction of immediacy).

Definition 9 *A* cancellation *of a side formula P of a (revised)* $(\vee \rightarrow)$ *inference is a principal formula of an axiom in the block in which P is introduced which has that occurrence of P as an ancestor.*

Definition 10 *A derivation is* strict *if for every (revised)* $(\vee \rightarrow)$ *inference in the derivation, both side formulas have cancellations.*

As we see in the proof of Lemma 8, $(\vee \to_R)$ is used only when necessary to ensure disjuncts have cancellations—i.e, only when the lower formula that needs cancellation contributes to the B derivation but wouldn't contribute to the A subderivation. Call the calculus so revised SCLA. Obviously, we can use weakening to transform an SCLA derivation into an articulated SCL derivation, so the rule is sound. The completeness of the rule is the content of the following lemma.

Lemma 8 *Any eager, articulated SCL derivation can be transformed into an SCLA derivation that is strict and blockwise eager.*

The **proof** takes advantage of an initial, subsidiary lemma, which is a direct corollary of Theorem 2.

Definition 11 *The side formula of a rule R is linked in $\mathcal{D}$ if it is the ancestor of a principal formula of an axiom in the same block of $\mathcal{D}$ as R. The inference R is linked iff it has a linked side formula. A derivation is linked iff all of its inferences are.*

Lemma 9 *Any strict, articulated SCL derivation $\mathcal{D}$ (possibly containing revised $(\vee \to)$ inferences) can be transformed into another in which the order of (retained) rules is preserved and every rule application in the block that includes the root is linked.*

Given this lemma, the proof of Lemma 8 assumes a strict, linked, blockwise eager SCL derivation $\mathcal{D}$, with the property that no revised $(\vee \to)$ inferences appear above any original $(\vee \to)$ inference, and (perhaps) with a distinguished formula G in the end-sequent that is an ancestor of the principal formula of an axiom in the lowest block. It shows that $\mathcal{D}$ can be transformed into a strict, linked, blockwise eager SCLA derivation by induction on the number of ordinary $(\vee \to)$ applications in D, where G remains an ancestor of the principal formula of an axiom in the lowest block.

In the base case there are no ordinary $(\vee \to)$ rules, so $\mathcal{D}$ is in fact an SCLA derivation.

In the inductive case, we are given $\mathcal{D}$ with n ordinary disjunctions, and assume the hypothesis true for derivations with fewer. We find an application R of SCL $(\vee \to)$ to replace with no ordinary $(\vee \to)$ closer to the root of $\mathcal{D}$.

To describe the transformation, we restrict attention to the subderivation $\mathcal{D}'$ of $\mathcal{D}$ containing R and rooted at the highest right subderivation of a revised $(\vee \to)$ rule S if any (and the root of $\mathcal{D}$ otherwise). Since $\mathcal{D}$ is strict, any side formula F of S is linked in the lowest block of $\mathcal{D}'$ (at the root of $\mathcal{D}$ use the key formula G as F). We decide whether to use $(\vee \to_L)$ or $(\vee \to_R)$ in place of R based on how F fits into the derivation.

Let $A \vee B$ be the principal formula of R; call the subderivation of $\mathcal{D}'$ below R (from R) $\mathcal{D}^R$ and call the right subderivation above R (in which B is assumed) $\mathcal{D}^B$. Weaken $\mathcal{D}^R$ by the formula B (and by equalities and introductions of terms as needed) to obtain $W(\mathcal{D}^R)$. We also weaken $\mathcal{D}^B$ by equalities and introductions of terms if necessary, so that we can identify the open leaf of $W(\mathcal{D}^R)$ with the end-sequent of $\mathcal{D}^B$. This new derivation contains $n \Leftrightarrow 1$ occurrences of $(\vee \to)$, since it contains only applications from $\mathcal{D}$ and does not contain

R. For similar reasons, it remains strict and blockwise eager. We can thus obtain a strict, linked, blockwise eager version $\mathcal{B}$ by applying Lemma 9. A parallel construction will give a strict, linked, blockwise eager version $\mathcal{A}$ obtained from the left subderivation above R.

We now observe two facts about the derivations $\mathcal{A}$ and $\mathcal{B}$, which follow because the construction of the linking lemma preserves the inferences in linked derivations. First, a derivation containing the inferences of $\mathcal{D}^R$ occurs as a subderivation of $\mathcal{B}$ and a derivation containing the inferences of its alternative $\mathcal{D}^L$ occurs as a subderivation of $\mathcal{A}$. Second, every axiom from $\mathcal{D}'$ is represented either in $\mathcal{A}$ or in $\mathcal{B}$.

As a result of these facts, we can select either of these derivations, say $\mathcal{B}$, and apply the induction hypothesis to obtain a new strict, blockwise eager SCLA derivation in which B is linked in the initial block. Replacing this derivation as the right subderivation of R turns R into a strict inference of $(\vee \rightarrow_L)$. Lemma 9 can now be applied; afterwards, because the $\mathcal{B}$ derivation is in a different block, the inferences in the lowest block will now match $\mathcal{A}$. Thus, if F is linked in $\mathcal{A}$, this is the construction to use. Otherwise, the mirror construction must be used, to generate a strict inference of $(\vee \rightarrow_R)$. Overall, we now have a strict, blockwise eager SCL derivation with fewer original $(\vee \rightarrow)$ inferences, so applying the induction hypothesis to the result gives the final needed derivation. ■

The next step is to develop a local $(\supset\rightarrow_L)$ rule that imposes a SEGMENT structure on derivations:

$$\frac{\Pi; \longrightarrow A^\mu, \Delta; \Theta \qquad \Pi; \Gamma, A \supset B^\mu, B^\mu \longrightarrow \Delta; \Theta}{\Pi; \Gamma, A \supset B^\mu \longrightarrow \Delta; \Theta}\ (\supset\rightarrow_L)$$

The use of this rule in blockwise eager derivations ensures that the processing of each new goal refers directly to global program clauses. This fact is formalized as Lemma 10.

Lemma 10 *Let $\mathcal{D}$ be a blockwise eager SCLA derivation, perhaps including $(\supset\rightarrow_L)$ inferences, with a end-sequent of the form $\ldots \triangleright \Pi; \rightarrow \Delta; \Theta$. Let R be any* fresh *right rule in $\mathcal{D}$—one such that the path from R to the root never follows the left branch of $(\supset\rightarrow)$. Then the end-sequent of $\mathcal{D}_R$ has the form $\ldots \triangleright \Pi'; \rightarrow \Delta'; \Theta'$.*

Proof. Suppose otherwise, and consider the lowest fresh right rule R in $\mathcal{D}$ with some local clauses Γ. R cannot be the first inference of $\mathcal{D}$, so there must be an inference S in $\mathcal{D}$ immediately below R. If S is a left rule, then the fact that $\mathcal{D}$ is blockwise eager leads to a contradiction: S must be $(\supset\rightarrow')$ or $(\vee \rightarrow')$ and R must appear along the branch without local clauses. Meanwhile, if S is a right rule, it follows from the formulation of the rules that if the end-sequent of $\mathcal{D}_R$ has nonempty local clauses then the end-sequent of $\mathcal{D}_L$ must also. This contradicts the assumption that R is first. ■

The use of $(\supset\rightarrow_L)$ exclusively provides a modified calculus SCLB. The correctness of SCLB over SCLA mirrors the correctness of SCLA over articulated SCL. The soundness of the $(\supset\rightarrow_L)$ rule can be established by weakening SCLB derivations to match the SCLA $(\supset\rightarrow)$ figure. Completeness is demonstrated by a hierarchical copying strategy, as outlined in Lemma 11.

Lemma 11 *Any SCLA derivation that is blockwise eager and strict can be transformed to an SCLB derivation with the same properties.*

The **proof** is by induction on the number of occurrences of $(\supset\rightarrow)$ rules in a given derivation $\mathcal{D}$ (possibly also including $(\supset\rightarrow_L)$ inferences). The induction hypothesis returns a blockwise eager, strict SCLB derivation $\mathcal{D}'$, where also $\mathcal{D}'$ applies no inference to any right formula in its end-sequent in its lowest block unless the same inference is applied to the same principal formula in $\mathcal{D}$ in its lowest block.

In the base case $\mathcal{D}$ is just $\mathcal{D}'$, and all these conditions are immediately satisfied.

Suppose $\mathcal{D}$ is a derivation in which the $(\supset\rightarrow)$ is used n+1 times. Let L be some occurrence of $(\supset\rightarrow)$ with no other application of $(\supset\rightarrow)$ closer to the root of $\mathcal{D}$. Follow the path below L to the highest sequent that lies immediately above a right rule or a $(\supset\rightarrow_L)$ rule on the left branch or the root of $\mathcal{D}$; and let R be the rule immediately above this. Suppose L applies to a formula $A \supset B$ and H is the rule immediately above L on the left branch. R marks the beginning of the current segment of left rules, and H marks the beginning of the next segment of right rules. The subderivation ending with the inference R, from the end-sequent of L—$\mathcal{D}^L_R$ abstracts the left rules performed in this segment (and any resulting search). By Lemma 10, $\mathcal{D}^L_R$ ends with a sequent of the form $\ldots \triangleright \Pi; \longrightarrow \Delta; \Theta$. By the identification of R, we have the same succedent $\Delta; \Theta$ at R and at L.

We use $\mathcal{D}^L_R$ to construct a derivation $\mathcal{D}'$ corresponding to the subderivation of $\mathcal{D}$ rooted at H, $\mathcal{D}_H$, by induction on the structure of derivations. The induction uses cases to keep the proof blockwise eager. If the derivation ends in a right rule, the corresponding derivation is constructed by obtaining corresponding subderivations and recombining them by the same right rule.

Otherwise, our construction first weakens $\mathcal{D}^L_R$ by appropriate formulas, constraints and introductions of terms, and then identifies the open leaf in $\mathcal{D}^L$ with the input derivation. By gluing two eager derivations together at places where left rules could apply in both, this gives an eager derivation; it has at most n uses of $(\supset\rightarrow)$. If necessary, Lemma 9 can be applied to reduce this derivation to a strict, linked component. Then the induction hypothesis applies to give a derivation in which only $(\supset\rightarrow_L)$ is used. This is the needed result.

Now we substitute the proof $\mathcal{D}'$ for $\mathcal{D}_H$ in $\mathcal{D}$, obtaining a new proof $\mathcal{D}''$ in which the occurrence of L now represents an application of $(\supset\rightarrow_L)$ and which therefore contains at most n uses of $(\supset\rightarrow)$. $\mathcal{D}''$ remains locally eager since no inference appearing in $\mathcal{D}'$ can be delayed (in $\mathcal{D}''$) unless a corresponding inference appearing in $\mathcal{D}_H$ is delayed (in $\mathcal{D}$). The replacement is limited to $\mathcal{D}''_R$, and no right rule applies higher in the same block to a formula in the end-sequent of $\mathcal{D}''_R$. This ensures that right rules apply in $\mathcal{D}''$ and in $\mathcal{D}$ to the same formulas from the end-sequent in the same block as the lowest inference. Now the induction hypothesis applies to give the needed overall derivation. ∎

3.4.5 Modularity

The final step in the justification of the calculus is to enforce modularity. This again is accomplished by adapting the rules of the sequent calculus in several stages. First, we refashion $(\rightarrow \Box)$ to use path eigenvariables instead of Skolem terms. These specialized rules are:

(50) a
$$\frac{\Pi; \Gamma \longrightarrow G^{\mu\alpha}, \Delta; G^{\mu\alpha}, \Theta}{\Pi; \Gamma \longrightarrow \Box_i G^{\mu}, \Delta; \Theta} \rightarrow \Box_i \ (\alpha : i \text{ new})$$

b
$$\frac{\Pi, F^{\mu\alpha}; \Gamma \longrightarrow G^{\mu\alpha}, \Delta; G^{\mu\alpha}, \Theta}{\Pi; \Gamma \longrightarrow F >_i G^{\mu}, \Delta; \Theta} \rightarrow >_i \ (\alpha : i \text{ new})$$

The element α is new iff it is distinct from any other element in the input declaration Σ. This suffices because, in proving Herbrand's theorem, Skolem terms are needed because a single rule which belongs early in the ground proof may be delayed. This won't happen with $(\rightarrow \Box)$ rules in (blockwise) eager proofs.

To work this out formally, we first describe a transformation on derivations that describes conditions when eigenvariables can be introduced into derivations.

Lemma 12 *Given a pure-variable SCLB proof $\mathcal{D}$, θ with empty input constraints and specification and output constraints C_f and specification Σ_f. Suppose $\mathcal{D}$ contains a subderivation $\mathcal{E}$ ending:*

$$\Sigma, \alpha : i/\Sigma'; C/C' \triangleright \Pi, (B^{\mu\alpha}); \Gamma \rightarrow A^{\mu\alpha}, \Delta; A^{\mu\alpha}, \Theta$$

Here α is some term representing a modal transition, such that the only occurrences of $\alpha\theta$ when θ applies to the end-sequent are those explicitly indicated above and others in Σ' and C'. Then for any other term β a derivation $\mathcal{D}'$ can be constructed with the following properties. The inferences of $\mathcal{D}'$ correspond to $\mathcal{D}$. The equalities generated by $\mathcal{D}'$ are a list of occurrences from $C'' \cup (C_f \backslash C')$ (suppressing the initial list structure). Similarly, the declarations generated by $\mathcal{D}'$ occur in $\Sigma'' \cup (\Sigma_f \backslash \Sigma')$. $\mathcal{D}'$ is solved by a unifier θ', and, finally, the subderivation of $\mathcal{D}'$ corresponding to $\mathcal{E}$ ends

$$\Sigma, \beta : i/\Sigma''; C/C'' \triangleright \Pi, (B^{\mu\beta}); \Gamma \rightarrow A^{\mu\beta}, \Delta; A^{\mu\alpha}, \Theta$$

Again, we can describe this transformation by an informal argument about where substitutions must be performed on the derivation. In this case, we can simply replace α by β throughout $\mathcal{E}$ (and propagate the change to the records of equations and declarations as necessary in $\mathcal{D}$). The new substitution is obtained by replacing α by β in the values of all and only the logic variables introduced in $\mathcal{E}$. Since we have an overall pure-variable derivation, these variables will not occur outside equations C''. ■

The new rules are introduced by exploiting this result.

Lemma 13 *For any strict, blockwise eager SCLB derivation $\mathcal{D}$ and any unifier θ solving its constraints, there is a corresponding strict, blockwise eager derivation $\mathcal{D}'$ using SCLB with the revised rules of (50) and a unifier θ' solving its constraints.*

We begin by observing a consequence of Theorem 2, as it applies to $\mathcal{D}$.

Definition 12 *Say a multiset of formulas* Π *is* spanned *by a multiset* Θ under θ *iff every prefix of a label of a formula in* Π *is equal under* θ *to the label of some* Θ *formula.*

Consider a proof $\mathcal{D}, \theta$ whose end-sequent

$$\Pi; \Gamma \longrightarrow \Delta; \Theta$$

is spanned by Θ under θ. Then every sequent

$$\Pi'; \Gamma' \longrightarrow \Delta'; \Theta'$$

in $\mathcal{D}$ is spanned by Θ' under θ. Because rules introduce new prefixes into the global workspace, and because the proof can adopt a regime where the global workspace is only extended by subderivations, the only step at which this could fail is that where some logic variable x is introduced:

$$\frac{\ldots A^{\mu x} \longrightarrow \Delta; \Theta}{\ldots \Box A^{\mu} \longrightarrow \Delta; \Theta} \Box \rightarrow$$

By Theorem 2, this inference contributes to an axiom; this must lie in the current segment, since x represents local work. Because the derivation is blockwise eager, no higher rules apply to Δ or Θ formulas within the segment. Therefore the right term of whatever equation x first appears in is the label of a Δ formula. This shows that θ equates μx (and every prefix of it) with some label of a Θ formula.

Using this observation, we transform $\mathcal{D}$ to use the new rules by induction. For most cases we just apply the transformation to subderivations and recombine; the only exceptional cases are $(\rightarrow \Box)$ and $(\rightarrow >)$. Let $\alpha(X)$ be the variable introduced at the rule in question. Consider whether the conditions for replacing $\alpha(X)$ by a fresh variable β apply for the unifier θ. If they do, perform the replacement and continue. Otherwise, θ finds an occurrence of $\alpha(X)$ outside of the principal formula of the rule. Therefore, by the previous argument, a formula whose annotation ends in $\alpha(X)$ must already be an element of Θ. Because Skolem functions are uniquely associated with connectives, and because of their global preservation, this means that the side formulas of the inference are already present in the end-sequent. The inference can therefore be eliminated (by applying the contraction transformation or introducing a restart rule, as appropriate). ■

We now turn to the second step in establishing modularity. We rewrite inference figures so that every sequent in the proof has at most one formula in the left and right local areas, and further if a right rule applies the left local area is empty. This rewriting is likewise achieved by straightforward induction on proofs. Multiple formulas in sequents are needed only for passing ambiguities and work done across branches in the search; but this is precisely what the use of $(\vee \rightarrow_L)$, $(\vee \rightarrow_R)$ and $(\supset\rightarrow_L)$ prevents.

Lemma 14 *Given a strict, blockwise eager derivation $\mathcal{D}$ using SCLB with the revised rules of (50), we can construct a corresponding derivation using the inferences schematized in Figure 3.11 and 3.12 (ignoring restrictions of modularity).*

Proof. We construct by induction on the structure of $\mathcal{D}$ a derivation $\mathcal{D}'$ with the property of the lemma plus three additional invariants. $\mathcal{D}'$ will contain in each segment all and only the axioms of the corresponding segment of $\mathcal{D}$ (and likewise for blocks); whenever $\mathcal{D}'$ contains a sequent of the form $\ldots \rightarrow F; \Theta$ then F will be the only right formula which is the ancestor of the right principal formula of an axiom in that block; and whenever $\mathcal{D}'$ contains a sequent of the form $\ldots \triangleright \Pi; F \rightarrow \ldots$, then F is the only left formula which is the ancestor of the left principal formula of an axiom in that segment.

In the base case, $\mathcal{D}$ is

$$\Pi; \Gamma, A^{\mu}_{X} \longrightarrow B^{\nu}_{X}, \Delta; \Theta$$

and $\mathcal{D}'$ is

$$\Pi; A^{\mu}_{X} \longrightarrow B^{\nu}; \Theta$$

Supposing the claim true for proofs of height h, consider a proof $\mathcal{D}$ with height h+1. We consider cases for the different rules with which $\mathcal{D}$ could end.

The treatment of $(\rightarrow \wedge)$ is representative of the case analysis for most right rules. $\mathcal{D}$ ends

$$\frac{\ldots \longrightarrow A, \ldots \qquad \ldots \longrightarrow B, \ldots}{\ldots \longrightarrow A \wedge B, \ldots} \rightarrow \wedge$$

We observe from Lemma 10 that in the initial derivation there is an empty local area. We simply apply the induction hypotheses to the immediate subderivations. If the resulting derivations end with (restart), consider the immediate subderivation of the results, otherwise consider the results themselves. They end

$$\ldots \longrightarrow C; \ldots$$
$$\ldots \longrightarrow D; \ldots$$

We must have $C = A$; we know from the structure of $\mathcal{D}$ that A is linked, and A could not be linked in $\mathcal{D}$ unless $C = A$ since $\mathcal{D}'$ shows that all of the axioms in $\mathcal{D}$ derive from C. For the same reason $D = B$. So we can combine the resulting proofs by $(\rightarrow \wedge)$ rule.

For the case of $(\rightarrow >)$, we need to make an additional observation. When induction gives us a derivation of

$$\triangleright \Pi, A^{\mu\alpha}; \longrightarrow C; B^{\mu\alpha}, \Theta$$

we know from the structure of $\mathcal{D}$ only that one of A and B must be linked. However, since α must be new, if A is linked then so is B.[3] But if C is different from B, B cannot be linked. So $C = B$.

[3]This is a consequence of the introduction of an eigenvariable here cf. [Stone, to appear, lemma 2]. Consider a derivation $\mathcal{D}$ using the new rules which has a unifier θ for which some left formula A in the end-sequent has

Now suppose $\mathcal{D}$ ends in a left rule other than $(\supset\rightarrow)$ or $(\vee\ \rightarrow)$. Apply the induction hypothesis to the immediate subderivation(s), and strip off the lowest (decide) rule, if there is one. That gives new derivation(s) with an end-sequent of the form:

$$\Pi; T \longrightarrow A; \Theta$$

In all cases, T must be a side formula of the left rule; otherwise the original left rule could not have been linked in $\mathcal{D}$. The corresponding rule from the system of Figure 3.11 applies to extend the induction hypothesis.

For $(\supset\rightarrow)$, applied to $A \supset B$, we first apply the induction hypothesis to the left subderivation. After stripping off any (restart), we get a proof of some new right formula C; by linking, this must in fact be A. We then apply the induction hypothesis also to the right subderivation. Again, after stripping off any (decide), we get a proof from some new left formula D; by linking this must in fact be B. Otherwise, we can combine the two derivations by the $(\supset\rightarrow)$ rule of Figure 3.11.

Finally, for $(\vee\ \rightarrow)$, applied to $A \vee B$, we first apply the induction hypothesis to the subderivation in the current block. By linking, there is a local program consisting of the side formula of this inference in the resulting subderivation. We apply the induction hypothesis to the other subderivation. Since both local areas are empty in the input subderivation, they remain empty in the result subderivation. The two subderivations can be recombined by the appropriate rule of Figure 3.11. ■

Finally, we enforce modularity, using the following observation. Using these rules, whenever $\mathcal{D}'$ contains a sequent of the form $\Pi; \rightarrow\ G^{\nu}; \Theta$ then G^{ν} will be the only right formula which is the ancestor of the right principal formula of an axiom in that block. Induction on path logic proofs shows that if a left formula P^{μ} from Π is the ancestor of a principal formula of an axiom in such a block, μ must equal a prefix of ν. Now we have ensured that each time a $(\vee\ \rightarrow)$ rule applies, any disjunct P^{μ} has such a cancellation. Thus, in search, we can restrict the subsequent (restart) rule to goals G^{ν} with μ a prefix of ν.

The calculus SCLP that we obtain is shown in two parts: Figure 3.11 shows the rules for decomposing program statements; Figure 3.12 shows the rules for decomposing goals.

The discussion of the previous subsections represents an outline of the proof of the following theorem.

Theorem 3 *There is a proof of* $\Gamma \longrightarrow \Delta$ *in SCL exactly when there is a proof of* $\Gamma; \longrightarrow; \Delta$ *in SCLP.*

3.5 Summary

The action the interpreter specified by SCLP can be summarized as follows. A distinguished formula on the right in sequents represents the current goal at any state in proof search;

an annotation μ that is not a prefix of any annotation of any (locally) linked right formula in the end sequent. Induction shows that no descendant of A can ever be labeled in a higher (local) sequent with term equal by θ to a prefix of the label of any (locally) linked right formula. So this occurrence of A is not (locally) linked in $\mathcal{D}$.

$$\Gamma; P^{\mu} \longrightarrow A^{\nu}; \Delta(P^{\mu} = A^{\nu})$$

$$\frac{\Gamma, P^{\mu}; P^{\mu} \longrightarrow A^{\nu}; \Delta}{\Gamma, P^{\mu}; \longrightarrow A^{\nu}; \Delta}\text{decide}$$

$$\frac{\Gamma; P^{\mu} \longrightarrow A^{\nu}; \Delta}{\Gamma; P \wedge Q^{\mu} \longrightarrow A^{\nu}; \Delta} \wedge \rightarrow_{L}$$

$$\frac{\Gamma; Q^{\mu} \longrightarrow A^{\nu}; \Delta}{\Gamma; P \wedge Q^{\mu} \longrightarrow A^{\nu}; \Delta} \wedge \rightarrow_{R}$$

$$\frac{\Gamma; P^{\mu} \longrightarrow A^{\nu}; \Delta \qquad \Gamma, Q^{\mu}; \longrightarrow ; \Delta}{\Gamma; P \vee Q^{\mu} \longrightarrow A^{\nu}; \Delta} \vee \rightarrow_{L}$$

$$\frac{\Gamma; Q^{\mu} \longrightarrow A^{\nu}; \Delta \qquad \Gamma, P^{\mu}; \longrightarrow ; \Delta}{\Gamma; P \vee Q^{\mu} \longrightarrow A^{\nu}; \Delta} \vee \rightarrow_{R}$$

$$\frac{\Gamma; \longrightarrow Q^{\mu}; \Delta \qquad \Gamma; P^{\mu} \longrightarrow A^{\nu}; \Delta}{\Gamma; Q \supset P^{\mu} \longrightarrow A^{\nu}; \Delta} \supset\rightarrow_{L}$$

$$\frac{\Gamma; P^{\mu x} \longrightarrow A^{\nu}; \Delta}{\Gamma, \Box_i P^{\mu} \longrightarrow A^{\nu}; \Delta} \Box_i \rightarrow (\text{LV } x : i)$$

$$\frac{\Gamma; P[u/x]^{\mu} \longrightarrow A^{\nu}; \Delta}{\Gamma; \forall x.P^{\mu} \longrightarrow A^{\nu}; \Delta} \forall \rightarrow (\text{LV } u : \mu)$$

$$\frac{\Gamma; P[h(X)/x])^{\mu} \longrightarrow A^{\nu}; \Delta}{\Gamma; \exists x.P^{\mu} \longrightarrow A^{\nu}; \Delta} \exists \rightarrow (\text{SK } h(X) : \mu)$$

Figure 3.11: Logic programming sequent calculus—programs (abbreviated). Further restrictions maintain and exploit cancellations: For (decide), in current block, P^{μ} must not be forbidden and if P^{μ} needs cancellation, record its contribution.

if possible, the interpreter first breaks this goal down into its components. In particular, as claimed in section 3.2, modular goals like $[\text{C}]c$ and $[\text{T}]t$ are processed by considering transitions to fresh possible worlds where only the information from that module is available.

Once an atomic goal is derived, the program is consulted by applying (decide); the chosen clause is decomposed and matched against the current goal by applicable logical rules. In particular, at $(\vee \rightarrow)$, the second case analysis allows the current goal to be chosen flexibly by the (restart) rule. The (restart) rule is modular in that it limits the work that is reanalyzed to the scope of the ambiguity just introduced; this conforms to the description in section 3.2.

The SCLP presentation may be delicate to construct, but it works according to a simple intuition. A modular goal can be thought of as an assignment of a problem to a new independent agent that has access to precisely the information in the corresponding modu-

$$\frac{\Gamma; \longrightarrow G^{\nu}; G^{\nu}, \Delta}{\Gamma; \longrightarrow ; G^{\nu}, \Delta}\ \text{restart}$$

$$\frac{\Gamma; \longrightarrow F^{\mu}; \Delta \qquad \Gamma; \longrightarrow G^{\mu}; \Delta}{\Gamma; \longrightarrow F \wedge G^{\mu}; \Delta} \rightarrow \wedge$$

$$\frac{\Gamma; \longrightarrow F^{\mu}; \Delta}{\Gamma; \longrightarrow F \vee G^{\mu}; \Delta} \rightarrow \vee_L$$

$$\frac{\Gamma; \longrightarrow G^{\mu}; \Delta}{\Gamma; \longrightarrow F \vee G^{\mu}; \Delta} \rightarrow \vee_R$$

$$\frac{\Gamma, F^{\mu\alpha}; \longrightarrow G^{\mu\alpha}; G^{\mu\alpha}, \Delta}{\Gamma; \longrightarrow \Box_i(F \supset G)^{\mu}; \Delta} \rightarrow \Box_i \supset \text{(New } \alpha : i)$$

$$\frac{\Gamma; \longrightarrow G^{\mu\alpha}; G^{\mu\alpha}, \Delta}{\Gamma; \longrightarrow \Box_i G^{\mu}; \Delta} \rightarrow \Box_i \text{ (New } \alpha : i)$$

$$\frac{\Gamma; \longrightarrow G[h(X)/x]^{\mu}; \Delta}{\Gamma; \longrightarrow \forall x.G^{\mu}; \Delta} \rightarrow \forall \text{ (SK } h(X) : \mu)$$

$$\frac{\Gamma; \longrightarrow G[u/x]^{\mu}; \Delta}{\Gamma; \longrightarrow \exists x.G^{\mu}; \Delta} \rightarrow \exists \text{ (LV } u : \mu)$$

Figure 3.12: Logic programming sequent calculus—goals (abbreviated). Further restrictions maintain and exploit cancellations: For (restart) find Q^{μ} needing cancellation in current block with $\mu \leq \nu$; if $R^{\mu'}$ forbidden in current block, check that $R^{\mu'}$ contributes only to G^{ν} in the previous block. For $(\vee \rightarrow_L)$, it's Q^{μ} that needs cancellation in the new block. For $(\vee \rightarrow_R)$, $R^{\mu'}$ needs cancellation in current block; it's P^{μ} that needs cancellation and $R^{\mu'}$ that's forbidden in the new block.

lar statements. This intuition provides a powerful handle on the close connection between modularity and ambiguity, proof size and search control in deductions. As suggested in section 3.2, modularity can mean the difference between efficiently executable specifications and prohibitive search in reasoning tasks involving partial information.

4
Constraints for Possible Worlds

Thus far, this dissertation has emphasized the modularity of modal logic. Chapter 2 described the modularity of modal logic, particularly as it is operationalized in restrictions on the possible inferences and the possible structures in modal proofs. Chapter 3 showed how these kinds of modularity could be supported in a logic programming language based on modal logic. In both chapters, we saw illustrations in logic programming and artificial intelligence of how modularity informs the control and analysis of search for declarative representations. This suggests that modal logic offers an interesting and promising representation for practical problems.

In attempting to exploit modal logic for such purposes, however, we are immediately confronted by a further computational problem: how to compute with path-based representations of possible worlds. In the explicitly-scoped modal proof systems which we reviewed in Chapter 2 and refined in Chapter 3, each formula is labeled by a string recording the sequence of embedded operators along the path to the possible worlds where the formula holds. EQUATIONAL UNIFICATION is one technique that allows the label of a formula to be partially instantiated as the formula is used during proof search.

This process is complex, and too expensive for practical use. Even for a small putative modal proof—containing one step for each symbol in the theorem to be derived—it can be intractable to find a unifier that correctly instantiates inferences at appropriate possible worlds. Moreover, the ambiguities involved arise as a result of the equational theory governing modal terms, and are therefore difficult to avoid by judicious reformulation of logical statements (of the sort familiar to Prolog programmers). This kind of ambiguity has no analogue in ordinary first-order deduction without equality. The intractability it brings provides a profound obstacle to the general use of modal logic for efficient, declarative search control.

Not every feature of the complexity of modal logic represents such an obstacle to its use in specification languages. For example, it is well-known that many propositional modal logics involve (putatively) higher complexity than propositional classical logic. PSPACE-completeness results for propositional modal logic [Ladner, 1977; Halpern and Moses, 1985b] show that proofs must in some cases be unreasonably large. Global problems with the possible size of proofs are familiar from ordinary first-order deduction without equality. Indeed, they are an inevitable concomitant of any attempt to specify algorithms in a logical way, if no fixed limit is to be placed on the number of states in the computation. Moreover, the proofs found under a logic programming search strategy already involve an explosion in size compared to other proofs, and the addition of modality will not increase the size of logic programming proofs in particular.

This chapter extends the proof-theoretic investigation begun in Chapter 3 to describe reasoning about possible worlds more precisely. This chapter shows that the final logic programming calculus SCLP exhibits two properties that allow efficient constraint-based reasoning about possible worlds. The first is the constrained use of variables described in Theorem 2, made possible by the logical fragment in which logic programming takes place; the second is the use of atoms in place of Skolem terms to represent modal transitions, made possible by the search strategy which logic programming adopts. By analyzing the unification problems that respect these constraints, I show that solutions respect the order in which modal transitions are introduced into the proof.

With a single modal operator, this constraint resolves all essential ambiguities in equational unification. Unifiability can therefore be determined in polynomial time; moreover, the constraints encountered at any point in proof search can be represented by a partial-order mechanism that avoids the need to backtrack among alternative unifiers. The same strategy generally applies in logics with multiple modalities, although the correctness of this strategy requires constraints on the interactions between modal operators. The ability to invoke this algorithm means that the logic programming language described in Chapter 3 does not in fact suffer from general intractability in its basic operations—and so using the language in practice is not a ridiculous idea. Later chapters reinforce the utility of these algorithms for a range of practical problems, including temporal reasoning and automatic planning, and reasoning about the knowledge of communicating agents in dialogue.

The organization of the rest of the chapter is as follows. I begin in section 4.1 by describing the problem of equational unification for modal proof in more detail. In section 4.2, I review the proof-theoretic properties which distinguish logic programming search from more general kinds of modal search. A constraint algorithm exploiting these properties is presented in section 4.3.

4.1 The Problem

Using explicitly-scoped modal proof systems, modal inference is as tractable as classical logic in the following sense: just as in classical logic, proof search can be carried out modulo permutations of rules, using unification. In particular, unification rather than explicit choice can be used to determine the scoped locations at which modal operators must be introduced.

However, these results do not make modal logic practical, because the unification involved is not ordinary unification, but STRING UNIFICATION. General algorithms exist for such problems (see [Schulz, 1993] and references therein). These procedures typically extend transformation-based algorithms for ordinary unification [Martelli and Montanari, 1982] by guessing inclusion relations between initial free variables in equal strings and possibly backtracking. Existing modal inference systems use nondeterministic equational unification algorithms of this sort [Debart *et al.*, 1992; Otten and Kreitz, 1996]. These methods are extremely expensive.

To illustrate, we return to the proofs shown in figures 2.17, 2.18 and 2.19 of the formulas

$$\Box(a \supset \Box b) \longrightarrow \Box\Box(a \supset b)$$
$$\Box(a \supset \Box b) \longrightarrow \Box(a \supset \Box b)$$
$$\Box(a \supset \Box b) \longrightarrow a \supset \Box\Box b$$

In light of Chapter 3, we can observe that these proofs illustrate logic programming search; in particular, a logic programming interpreter will begin by constructing the right branch which addresses the b goal. Completing this branch requires solving an equation $xy = \alpha\beta$ in all cases. For this problem, string unification algorithms will return unifiers corresponding to the three different solutions exhibited in the three proofs ($x = \epsilon; x = \alpha; x = \alpha\beta$). Which possibility is needed is resolved only when the next axiom is reached and the final equation processed. Branching among the possible unifiers is prohibitive (it is easy to see the number may be exponential in the length of the strings being unified). Yet there is also no effective way to exploit unifiability as a constraint. Because of the backtracking internals of equational unification algorithms, they frequently fail to solve systems of equations more efficiently than would a backtracking program that called the algorithm in sequence on each equation in the system.

As we shall see in this section, determining appropriate values for modal variables by unification is in fact an intractable problem. Before presenting this result, I observe that this problem is quite different in nature and origin from the well-known space complexities of modal logic. Although classical propositional logic is co-NP complete, Ladner [Ladner, 1977] and Halpern and Moses [Halpern and Moses, 1985b] have shown that a number of propositional modal logics, including all those considered here, are PSPACE-complete. The proof that these logics are PSPACE-hard relies on describing large objects concisely using modal theories. Such descriptions apply the same formula across a number of possible worlds in a modal proof; when statements of possibility introduce several worlds, a proof may have to proceed by applying necessary information in each. First-order quantifiers provide a good point of reference in interpreting these results about quantifiers over worlds. In first-order logic, the number of instantiations of a universal statement needed to complete a proof cannot be bounded at all. This makes first-order logic undecidable.

Because what matters for the proof is the sheer number of instantiations, modal provability can be PSPACE-complete even when instantiating modal variables by unification is easy. For example, since K variables can only be instantiated to single terms, modal equations for K can be solved using ordinary (linear-time) unification. But K provability is PSPACE-complete. Moreover, as in first-order logic, the number of instantiations and size of proof depends greatly on the logical theory, and often much better bounds can be easily derived—arguably in most cases of interest. Prolog programmers can analyze theories to ensure efficient proof-search; [Kanovich, 1990] reports an application of a PSPACE-complete deduction system for intuitionistic logic in which proof size corresponds to the number of interacting subtasks and is rarely problematic. When bounds on proof-size are known for a given theory, general PSPACE-completeness results have nothing to add. However, complexity results for unifying modal terms continue to apply. In fact, the complexity of this unification is likely to pose the most significant obstacle to the use of modal logic in practical applications, because alternatives in unifying modal terms arise because of the very axioms for relating kinds of information that make modal logic attractive as a representation in the first place.

It is also noteworthy that the complexity of unification for modal terms cannot be established by the usual encodings of hard problems using string unification, such as those presented in [Kapur and Narendran, 1986; Kapur and Narendran, 1992]. These encodings repeat variables in different contexts to enforce constraints. Such repetitions are unavailable because of the UNIQUE PREFIX PROPERTY on occurrences of variables in modal equations (cf. [Wallen, 1990; Auffray and Enjalbert, 1992]).

The unique prefix property makes reasoning about equations between modal paths much easier than reasoning about string equations in general. A polynomial amount of information specifies the tree corresponding to any unifier; therefore, annotation equations have only a finite number of most general solutions, which is not guaranteed in general for string unification problems. Moreover, since an efficient algorithm can determine whether a set of strings are equal under a polynomial size substitution, the problem of solving the modal unification problem associated with a modal logic proof attempt is in NP.

Nevertheless, the problem is hard.

Theorem 4 *For propositional modal logic of a single modality governed by* (VER)*, in the lifted, explicitly-scoped sequent calculus, the problem of determining whether there is a proof containing a given derivation as its first component is NP-hard (as a function of the size of the derivation).*

Proof. We proceed by reduction from three-partition, a standard NP-complete problem defined as follows (cf. [Garey and Johnson, 1979] p. 96). We are given a finite set A containing $3m$ elements, a positive integer B and a size function s such that $B/4 < s(a) < B/2$ for each $a \in A$, and such that $\sum_{a \in A} s(a) = mB$. We are to determine whether A can be partitioned into m disjoint sets such that the sizes of the elements of each set sum to B.

We proceed in two steps. The first is to construct a unification problem that corresponds to the instance of three-partition; the second is to describe a modal sequent $\Gamma \longrightarrow \Delta$ such that a proof attempt for $\Gamma \longrightarrow \Delta$ gives rise to this unification problem.

First, the unification problem is this. For each element $a \in A$, we construct a string Q_a of the form $X_a C_a Y_a$. X_a and Y_a are strings containing $m(B+1)$ variables; C_a is a string containing $s(a)$ constants. We also construct a string G containing m successive sequences of B variables Z_i followed by a constant K_i. All of the variables in X_a, Y_a and Z_i are distinct, as are all of the constants in C_a and K_i. All constants and variables are governed by an equation $\mu; 1 = \mu$; this corresponds to assigning each to a T modality 1 (governed by the (VER) axiom $\Box_i A \supset A$). The unification problem is the set of equations $Q_a = G$ for each

$a \in A$.

Three-partition is NP-complete in the strong sense, which means there is a polynomial p in the LENGTH of the problem specification such that the problem remains NP-complete when the VALUES of the bound and the size function are bounded by p. Our encoding depends on this, because we represent the size $s(a)$ of each element as a string of length $s(a)$. Since we can bound $s(a)$ by a polynomial in the length of the three-partition instance, the length of the unification problem is also a polynomial in the length of the instance.

The unification problem has a solution if and only if the original three-partition problem has a solution. Suppose there is a solution θ. Note that each variable can be bound to a string containing either zero or one constant, and that all the constants of the Q_a must appear in $G\theta$. Since there are mB constants in the Q_a and mB variables in G, each variable in G must be bound to exactly one constant, and each constant appears exactly once in $G\theta$. Now look at Z_i. If $Z_i\theta$ contains any of the constants from Q_a, it must contain all of them, because the constants in Q_a are adjacent, and Z_i is bordered by the beginning of the string, or by K_j constants. Thus, the needed partition is given by taking for each i the set of elements of A whose constants appear in $Z_i\theta$. Meanwhile, suppose the three-partition problem has a solution. Naming the elements of each S_i S_{i1}, S_{i2} and S_{i3}, we can construct a unifier θ such that $G\theta = Q_{S_{11}}Q_{S_{12}}Q_{S_{13}}K_1 \ldots Q_{S_{m1}}Q_{S_{m2}}Q_{S_{m3}}K_m$. Solutionhood ensures that we can let $Z_i\theta = Q_{S_{i1}}Q_{S_{i2}}Q_{S_{i3}}$: we assign the jth variable in Z_i to the jth constant in $Q_{S_{i1}}Q_{S_{i2}}Q_{S_{i3}}$. Now let $l(a)$ be the prefix of Q_a in this string, and let $r(a)$ be the suffix of Q_a, and let $p(a)$ be the length of $l(a)$. To complete θ, we assign the first $p(a)$ variables in X_a to $l(a)$ and the remainder the empty string; we assign the last $p(a) + s(a)$ variables in Y_a to the empty string, and the remainder to $r(a)$.

Now, the second step: designing a proof attempt which gives rise to this problem. We assign a distinct proposition letter p_a for each element a of A. We prove the formula

$$\Delta = (\Diamond_1^B \Box_1)^m \bigwedge_{a \in A} p_a$$

(The notation $\Diamond_i^k \varphi$ represents a formula in which φ is preceded by k nested $\Diamond_i$ operators; and similarly for $\Box_i$ and sequences of operators.)

As we shall see, it is important that this proof attempt use the proof rules for the possibility operator $\Diamond_1$. We omitted these proof rules from the presentation of SCL for the $\Box$-only fragment in Figure 3.10. Of course, the additional rules are:

$$\frac{\Gamma, \Diamond_i A^\mu, A^{\mu\alpha(X)} \longrightarrow \Delta}{\Gamma, \Diamond_i A^\mu \longrightarrow \Delta} \Diamond_i \rightarrow (\text{SK } \alpha(X) : i)$$

$$\frac{\Gamma \longrightarrow A^{\mu x}, \Diamond_i A^\mu, \Delta}{\Gamma \longrightarrow \Diamond_i A^\mu, \Delta} \rightarrow \Diamond_i \text{ (LV } x : i)$$

Thus, each $\Diamond_1$ in this goal formula introduces a fresh variable, while each $\Box_1$ introduces a Skolem term with a unique head function constant. Thus, proving the goal ensures that each p_a is established at a world denoted by the string G. For each a, we include available the following assumption in Γ:

$$\Box_1^{m(B+1)} \Diamond_1^{s(a)} \Box_1^{m(B+1)} p_a$$

Each axiom makes available an assumption of p_a at a world that can be represented by Q_a, as the $\Box_1$ and $\Diamond_1$ operators will introduce the correct sequence of variables and distinct constants represented as Skolem terms. Now proving $\Gamma \longrightarrow \Delta$ generates a proof attempt in which each goal p_a at world G is matched with the assumption of p_a at world Q_a. This is precisely the unification problem considered above. ■

4.2 □-only Logic and Variable Introduction

The proof systems reviewed in Chapter 2 make possible streamlined deduction procedures, but their efficiency is limited by the inherent ability of modal theories to express hard problems. Building a proof requires choosing the right intercalation of modal operators among an exponential number of possibilities; in some cases, such choices make for intractable search problems. To support efficient, sound and complete inference, modal specifications must avoid the expressive features that give rise to these problems.

This section identifies possibility and classical negation as the problematic features of modal logic. In the absence of possibility and negation—in □-only logics—a simple rule suffices to determine the order of modal Skolem terms in unifiers: When the interpreter finds a string in which either α must follow β or β must follow α, the one that comes later in the string is the one that is introduced later in the proof. This theorem follows from the Theorem 2, the variable introduction theorem presented in section 3.4.3. The restriction on negation is not as dire as it may seem, as shown in section 4.2.2: in K, K4, T and S4, negation can be encoded using a scoped constant $\perp$. The effect of this encoding is to transform certain alternatives for unifying modal terms into alternative axiomatic links in proof search, so that the remaining modal alternatives can be managed efficiently. Anyway, the □-only restriction naturally describes the logic programming fragment of Chapter 3.

Why does the invariant on the introduction of terms hold? Informally, the invariant is combined effect of two properties that proofs in SCL enjoy in virtue of omitting possibility and negation. First, the terms representing the world where a formula holds can only grow through the application of modal rules. Accordingly, all the terms labeling a formula will appear in the label of any formula derivable from it. This is a property of the equational theory and typing rules governing paths of accessibility—a consequence of logical modularity—and can fail in accounts of additional axiomatic relationships between modalities. For example, adding the (NI) axiom $\Diamond A \supset \Box\Diamond A$ to S4 gives the system S5 in which a necessary formula (irrespective of its own label) can be applied at any world whatsoever.

Second, when □ alone appears in the proof, variables are introduced on annotations precisely when annotations change in LEFT rules, while Skolem terms are introduced on annotations only when annotations change in RIGHT rules. This fails if possibility is added to the language. Moreover, only the left $\supset$ rule allows new variable positions to be transferred to the right of a sequent from the left. But the left $\supset$ rule leaves these positions on the left of the sequent also. In contrast, the sequent rule for classical negation simply moves a formula from right to left.

Together, these two conditions propagate variables so that the first occurrence of a variable in an equation appears in a left term. From this, we can conclude using induction that each left term must be assigned a ground string of Skolem terms introduced earlier in the proof in order to be unified with the corresponding right term. Skolem terms, meanwhile, are introduced only on right terms, so there is no way for a newer Skolem term to represent a world closer to the real world than any older one. This constraint rules out or resolves search ambiguities such as those investigated in section 4.1.

4.2.1 Substitution Ordering

Recall that Theorem 2 shows that any $\Box$-only derivation can be transformed into one that satisfies the VARIABLE INTRODUCTION PROPERTY.

Definition 13 *$\mathcal{D}$ has the* variable introduction property *if and only if every modal logic variable x that occurs in some equation term r_i also occurs in some equation term l_j with $j < i$.*

The variable introduction property represents a strong constraint on equations, as the following result shows.

Lemma 15 (substitution ordering) *Suppose $\mathcal{D}$ is a proof attempt in SCL (or its variants SCLA, SCLB, SCLP) that enjoys the variable introduction property, and suppose the end-sequent of $\mathcal{D}$ is*

$$/\Sigma; /C \triangleright \Gamma \longrightarrow \Delta$$

Let θ be a substitution that unifies the strings in each equation of C (whether or not θ respects the typings in Σ). Then for any variable x appearing in C, first used in l_j, $x\theta$ is a string of Skolem terms, and if $x\theta$ contains Skolem term c there is a Skolem term f in some term r_i such that $i \leq j$ and $f\theta = c$.

Proof. By induction on the number of equations in C.

In the base case, there are no equations and nothing to show.

Suppose the proposition is true for the first $i \Leftrightarrow 1$ equations of C and consider a solution θ for the first i equations of C. Naturally, θ is a solution to the first $i \Leftrightarrow 1$ equations, so by the induction hypothesis, variables introduced in the last $i \Leftrightarrow 1$ equations are bound to earlier Skolem terms. But the proposition on variable introduction asserts that any variables of r_i all occur earlier. Therefore $r_i\theta$ is a string of Skolem terms; θ must associate any new variable in l_i with some of them; and $l_i\theta$ can contain no new Skolem terms. ■

With these two results, we can establish the main result:

Theorem 5 (constant ordering) *For any proof $\mathcal{D}, \theta$ in SCL (and its variants) there is a proof $\mathcal{D}', \theta$ where $\mathcal{D}'$ enjoys the variable introduction property and satisfies the substitution ordering property.*

Proof. By the variable introduction theorem, we can construct a $\mathcal{D}'$ satisfying the variable introduction property from $\mathcal{D}$. The only difficulty is to show that in obtaining the smaller $\mathcal{D}'$ we have not eliminated any premises needed to show that θ respects types. Since θ unifies the modal equations imposed in $\mathcal{D}$, and a subset of these equations are imposed in $\mathcal{D}'$, θ also unifies the modal equations imposed in $\mathcal{D}'$. By the substitution ordering lemma, θ assigns strings of Skolem terms to each modal logic variable (that appears in the equations of $\mathcal{D}'$). Every modal Skolem term and logic variable θ mentions is introduced in $\mathcal{D}'$ and therefore assigned identical types in $\mathcal{D}$ and $\mathcal{D}'$. And, as for first-order variables, eliminating typing premises only eliminates typing requirements. It follows from this that $\mathcal{D}', \theta$ is a proof. ∎

In $\Box$-only logics, a number of proof strategies allow transitions in modal terms to be represented as constants that are distinguished before unification (instead of as Skolem function applications with free variables). The obvious example is the logic programming inference of SCLP, developed in Chapter 3. But for example the mating theorem-proving method also does this, via the method of multiplicities [Andrews, 1981; Bibel, 1982]. We take SCLP as a reference.

In SCLP, we can define the following ordering on transitions in advance of solving unification equations:

Definition 14 ($\sqsubset$) *Let the equations corresponding to an SCLP proof attempt be ordered as before, and let c and d be ground elements appearing in these equations. $c \sqsubset d$ if and only if*

1. *c's first occurrence is in term C_{i1} and d's is in term C_{j1} with $i < j$, or*
2. *Both c and d's first occurrences are in term E_{i1}, in which c precedes d.*

$\sqsubset$ is a TOTAL ORDER on constants. Moreover, the substitution ordering property entails that for any solution θ for C, if $(\pi_c c)\theta$ is a proper substring of $(\pi_d d)\theta$, then $c \sqsubset d$. For this can occur only if d follows c in the same term in some equation, or d appears in a term after some variable x such that $x\theta$ includes c.

4.2.2 Encoding Negation

In general, we can describe $\neg A$ as $A \supset \bot$ using a propositional constant $\bot$ governed by the inference below:

$$\Sigma \triangleright \Gamma, \bot^{\mu} \longrightarrow \Delta$$

If A is the original formula, we denote by A^t the result of recursively replacing its subformulas $\neg B$ by $B \supset \bot$ and its subformulas $\Diamond B$ by $\Box(B \supset \bot) \supset \bot$. (We return to ground, explicitly-scoped modal sequent calculi. Lifting these rules is straightforward; presenting the lifted versions is distracting.) This encoding also describes a correspondence between proofs. The original rule-instances:

$$\frac{\Sigma \triangleright \Gamma, \neg A^{\mu} \longrightarrow A^{\mu}, \Delta}{\Sigma \triangleright \Gamma, \neg A^{\mu} \longrightarrow \Delta}\,\neg\rightarrow$$

match encoded rule-instances:

$$\frac{\Sigma \triangleright \Gamma, A \supset \bot^{\mu} \longrightarrow A^{\mu}, \Delta \qquad \Sigma \triangleright \Gamma, A \supset \bot^{\mu}, \bot^{\mu} \longrightarrow \Delta}{\Sigma \triangleright \Gamma, A \supset \bot^{\mu} \longrightarrow \Delta}\,\supset\rightarrow$$

The right subproof is an instance of the new $\bot$ rule. Meanwhile, the encoding puts rule-instances:

$$\frac{\Sigma \triangleright \Gamma, A^{\mu} \longrightarrow \neg A^{\mu}, \Delta}{\Sigma \triangleright \Gamma \longrightarrow \neg A^{\mu}, \Delta}\,\rightarrow\neg$$

in correspondence with patterns:

$$\frac{\Sigma \triangleright \Gamma, A^{\mu} \longrightarrow A \supset \bot^{\mu}, \bot^{\mu}, \Delta}{\Sigma \triangleright \Gamma \longrightarrow A \supset \bot^{\mu}, \Delta}\,\rightarrow\supset$$

Since the $\bot$ rule is in fact the only rule that can establish $\bot$ on the right and it can establish anything, the addition of $\bot$ on the right does not change the provability of the end-sequent of the immediate subderivation. The new constant $\bot$ is unscoped. Because it can establish any Δ, it breaks the invariant used in the variable introduction theorem.

However, for K, K4, T and S4, it in fact suffices to introduce a SCOPED constant $\bot$ governed by the rule:

$$\Sigma \triangleright \Gamma, \bot^{\mu} \longrightarrow A^{\mu}, \Delta$$

The use of this rule is clearly sound, because it is a specialization of the more general unscoped $\bot$ rule. The completeness of the rule is a consequence of the fact that any provable sequent in K, K4, T and S4 has a proof where all modal rules extend modal terms by strings of eigenvariables introduced lower. Under this circumstance, whenever we establish $\bot^{\mu}$ on the left, there will in fact be some formula A^{μ} on the right. Therefore no generality is lost by the scoped $\bot$ rule. However, with the scoped $\bot$ rule, the variable introduction theorem goes through, and the hence the algorithms of section 4.3 may be correctly applied.

I present presently a formal proof of correctness of the encoding of K, K4, T and S4 proofs using a scoped constant $\bot$. But first I want to show that there is no magic involved in the translation. In the original proof, there are ambiguities in which modal constants are nested under which. The translation does not eliminate these ambiguities. Instead, it recodes them at the level of proof search as ambiguities in which rule inferring $\bot$ is used to deduce which conclusion of $\bot$. In accordance with the constant ordering theorem, the modal constants introduced by whichever rule is used first appear first.

The proof search in S4 for the sequent below illustrates this point:

$$\Box\Diamond\Box A,\ \Box\Diamond\Box B \longrightarrow \Diamond(A \wedge B)$$

There are two kinds of proof. In the first, we first apply sequent rules to establish $\Box A^{\alpha}$, then establish $\Box B^{\alpha\beta}$. From this follows $A^{\alpha\beta}$ and $B^{\alpha\beta}$ and hence $\Diamond(A \wedge B)$. The other proof is similar, but we instantiate $\Box B^{\beta}$, and then $A^{\beta\alpha}$.

The translation of this sequent is:

$$\Box(\Box(\Box A \supset \bot) \supset \bot), \Box(\Box(\Box B \supset \bot) \supset \bot) \longrightarrow \Box(A \wedge B \supset \bot) \supset \bot$$

Consider proof search now. We reduce the end-sequent by $(\rightarrow\supset)$ to

$$\Box(\Box(\Box A \supset \bot) \supset \bot), \Box(\Box(\Box B \supset \bot) \supset \bot), \Box(A \wedge B \supset \bot) \longrightarrow \bot$$

At this point, we may use either the A-formula or the B-formula to establish $\bot$. If we use the A-formula, we can simplify to:

$$\Box(\Box(\Box B \supset \bot) \supset \bot), \Box(A \wedge B \supset \bot), \Box A^{\alpha} \longrightarrow \bot^{\alpha}$$

Now we use the B-formula:

$$\Box(A \wedge B \supset \bot), \Box A^{\alpha}, \Box B^{\alpha\beta} \longrightarrow \bot^{\alpha\beta}$$

Finally, we use the negation of possibility at $\alpha\beta$, and the remainder of the proof becomes clear:

$$\Box A^{\alpha}, \Box B^{\alpha\beta} \longrightarrow A \wedge B^{\alpha\beta}$$

It is clear how this proof corresponds to the original proof with A first. We can likewise find a translated proof with B first. Note that by translating the deduction problem we have introduced a number of new dead-ends for proof search, corresponding to early instantiation of the negation of possibility. Here these can be quickly dispensed with, since early on there are no possibilities for establishing A or B. As translation of possibility and negation proliferates, we will obviously start to need faster mechanisms for identifying and ruling out these new alternatives.

Translation of negation using the scoped $\bot$ rule is not without its pitfalls, but is is correct. Informally, what underlies its correctness is the following observation. K, K4, T and S4 proofs need never instantiate necessary formulas at arbitrary accessible annotations. In T and S4, this is because the current annotation can always serve as a witness possibility at which to apply a necessary formula. In K and K4, this is because there need not be such a possibility, and hence such instantiation would actually be incorrect. This is a property of these particular logics. Note that in KD and KD4 logics, which support consistency but not veridicality, one sometimes must instantiate necessary formulas at new, arbitrary accessible worlds.

This informal observation falls out formally as follows. Suppose $\mathcal{D}$ is deduction in an explicitly-scoped ground sequent system for $\mathcal{L}^{\Box}_{m,T}$, with end-sequent $\triangleright\Gamma \longrightarrow \Delta$. Consider any rule-application of $(\Box \rightarrow)$ or $(\rightarrow \Diamond)$ in $\mathcal{D}$. The rule extends the modal annotation of the principal formula μ by a string σ, where σ is a string of modal variables introduced at lower $(\rightarrow \Box)$ and $(\Diamond \rightarrow)$ rules in $\mathcal{D}$. This observation is a consequence of the sequent rules, which extend the typing contexts only by declarations of variables and only at $(\rightarrow \Box)$ and $(\Diamond \rightarrow)$ rules; and of the rules of figure 3.9, which only combine the terms declared in Σ into longer strings. Because of the unique prefix property, we can assume that $\mu\sigma$ in fact annotates some lower formula. Then, because lower formulas are preserved as logical rules are applied, $\mu\sigma$ must be the label of some formula on the sequent itself. So whenever we apply a necessary formula, we apply it in a world that is already under consideration.

We exploit this observation in the following lemma.

Lemma 16 (encoding negation) *To any proof $\mathcal{D}$ with end-sequent $\Gamma \longrightarrow \Delta$ in the ground, explicitly scoped system of figure 2.11, there corresponds a proof of $\Gamma^t \longrightarrow \Delta^t$ using the scoped $\bot$ rule.*

Proof. We define a translation $T(\mathcal{D}, E, F)$ recursively on the structure of $\mathcal{D}$; the end-sequent of $\mathcal{D}$ must be $\Sigma \triangleright \Gamma \longrightarrow \Delta$, where every Γ annotation and every E annotation is a prefix of some Δ or F annotation—E specifies additional formulas to add to Γ, F additional translated formulas to add to Δ.

We consider $\Diamond$ and $\neg$ rules explicitly. (The other cases are straightforward in light of these cases.) If $\mathcal{D}$ ends by applying $(\rightarrow \neg)$ with principal formula $\neg A^{\mu}$ and subderivation $\mathcal{D}'$, $T(\mathcal{D}, E, F)$ is:

$$\frac{T(\mathcal{D}', E, (F, \bot^{\mu}))}{\Sigma \triangleright \Gamma^t, E \longrightarrow \Delta, A \supset \bot^{\mu}, F} \rightarrow\supset$$

Observe that, even in a system with a contraction rule instead of preservation, the presence of $\bot^{\mu}$ ensures that the annotation of the Γ side formula A^{μ} is OK.

If $\mathcal{D}$ ends by applying $(\neg \rightarrow)$ to principal formula $\neg A^{\mu}$ and subderivation $\mathcal{D}'$, $T(\mathcal{D}, E, F)$ is:

$$\frac{T(\mathcal{D}', E, F) \qquad \Sigma \triangleright \Gamma^t, E, A \supset \bot^{\mu}, \bot^{\mu} \longrightarrow \Delta^t, F}{\Sigma \triangleright \Gamma^t, E, A \supset \bot^{\mu} \longrightarrow \Delta^t, F} \supset\rightarrow$$

By assumption, μ appears on some Δ or F formula, so the right subderivation is an instance of the scoped $\bot$ rule. (Since no new left formulas appear in the subderivation, the translation applies there.)

If $\mathcal{D}$ ends by applying $(\Diamond_i \rightarrow)$ to $\Diamond_i A^{\mu}$ with subderivation $\mathcal{D}'$, $T(\mathcal{D}, E, F)$ is:

$$\frac{\dfrac{\dfrac{T(\mathcal{D}', E, (F, \Box_i(A \supset \bot)^{\mu}, A \supset \bot^{\mu\alpha}, \bot^{\mu\alpha}))}{\Sigma, \alpha : i \triangleright \Gamma^t, E, \Box_i(A \supset \bot) \supset \bot^{\mu} \longrightarrow A \supset \bot^{\mu\alpha}, \Delta^t, F} \rightarrow\supset}{\Sigma \triangleright \Gamma^t, E, \Box_i(A \supset \bot) \supset \bot^{\mu} \longrightarrow \Box_i(A \supset \bot)^{\mu}, \Delta^t, F} \rightarrow\Box \qquad \Sigma \triangleright \bot^{\mu} \longrightarrow \Delta^t, F}{\Sigma \triangleright \Gamma^t, E, \Box_i(A \supset \bot) \supset \bot^{\mu}, \longrightarrow \Delta^t, F} \supset\rightarrow$$

The final step follows from the assumption that μ appears on some Δ or F formula; as with $(\rightarrow \neg)$, even with contraction replacing preservation, the root invocation of T satisfies the needed invariant.

Finally, if $\mathcal{D}$ ends in an application of $(\rightarrow \Diamond)$ to $\Diamond_i A^\mu$ with subderivation $\mathcal{D}'$, $T(\mathcal{D}, E, F)$ is:

$$\dfrac{\dfrac{\dfrac{T(\mathcal{D}', (E, \Box_i(A \supset \bot)^\mu, A \supset \bot^{\mu\sigma}), (F, \bot^\mu)) \qquad \Sigma \rhd \bot^{\mu\sigma} \longrightarrow \Delta^t, F, \bot^\mu}{\Sigma \rhd \Gamma^t, E, \Box_i(A \supset \bot)^\mu, A \supset \bot^{\mu\sigma} \longrightarrow \Delta^t, F, \bot^\mu}\,{\supset\rightarrow}}{\Sigma \rhd \Gamma^t, E, \Box_i(A \supset \bot)^\mu \longrightarrow \Delta^t, F, \bot^\mu}\,{\Box\rightarrow}}{\Sigma \rhd \Gamma^t, E \longrightarrow \Delta^t, \Box_i(A \supset \bot)^\mu, F}\,{\rightarrow\supset}$$

The key step here is to establish that $\mu\sigma$ occurs on some Δ formula. But σ is a string of eigenvariables introduced by lower modal rules, by our observation. Thus, by the unique prefix property of annotations, $\mu\sigma$ must be the annotation of some formula used lower—which remains in Δ because of preservation. ■

4.3 Constructing Trees from Constraints

With the constant ordering theorem, we have established an invariant that eliminates one source of nondeterminism in the unification of modal equations. Given Skolem terms α and β which appear on modal terms that must be equal, the one that is introduced first into the proof must appear first in the unified term. However, even in $\Box$-only proofs, modal equations may still have an exponential number of unifiers; the constant ordering theorem leaves open how strings of Skolem terms should be partitioned among matching LOGIC VARIABLES. We have already seen a concrete example of this, in the unification problem common to the proofs of figures 2.17, 2.18 and 2.19: $xy = \alpha\beta$. In order to complete those proofs, we need to be able to assign any of the possible prefixes of $\alpha\beta$ to x. So there are still too many possibilities for brute force search.

This section develops a constraint algorithm that finds representative unifiers for a set of equations efficiently, and which allows additional equations to be added incrementally. This algorithm relies on viewing the set of equations as describing a tree in terms of simple relationships between nodes. These constraints are operationalized as simple, local rules. The rules enforce constraints by making the smallest possible changes to the structure of the tree and to the representation of variables and constants within it. We develop the algorithm in three steps, deferring some technical complications so that the essentials of the algorithm can be presented as accessibly as possible. In 4.3.1, we present and analyze a basic version of the algorithm which solves constraints over a single modal operator; we illustrate the action of this algorithm on the proofs of figures 2.17, 2.18 and 2.19 in 4.3.2. The complications come in 4.3.3, where it is observed that inclusion axioms may introduce hard problems even into variable ambiguities. Accordingly, in 4.3.4 we consider restrictions on interaction axioms to rule out the problematic cases observed in 4.3.3, and provide a constraint solver for multi-modal logics under these restrictions which uses the algorithm from 4.3.1 as a subroutine.

4.3.1 Mono-modal Languages

Our strategy will be to recast the unification problems for K, T, K4 and S4 in terms of constructing a tree to satisfy three types of constraints:

1. The relation $u \le v$, meaning that u is an ancestor of v in the tree representation (corresponding to the constraint that $\pi_u u$ be a prefix of $\pi_v v$ as a string equation).

2. The relation $u \not\le v$, meaning that u is not an ancestor of v.

3. The relation $u \Rightarrow v$, meaning that the parent of u is an ancestor of v.

The encoding depends on the assumption introduced in the last section that equality of Skolem terms can be determined in advance of unification, as in SCLP or the matrix proof method. The encoding consists of a way to describe annotations and substitutions, a way to impose equalities between annotations, and a way to manage the domain constraints on the values of first-order variables. The encoding is described and justified as follows.

Substitutions. The set of images of prefixes of equations under θ describes a tree by the unique prefix property. We associate each modal Skolem term or logic variable u is mapped to a node in the tree $\hat{u}$.

To derive a substitution from a tree, we identify each node n in the tree with some canonical symbol c such that $n = \hat{c}$. By reading the canonical symbols along the path in the tree from the root to $\hat{u}$, we obtain the value of $\pi_u u$ under θ; the path from the node $\hat{v}$ representing π_u to $\hat{u}$ (not including $\hat{v}$ itself) therefore encodes $u\theta$.

A first set of constraints ensures that Skolem terms are mapped to themselves under this induced substitution. We impose on t constraints of the form $d \not\le c$ whenever $c \sqsubset d$ (as earlier $\sqsubset$ refers to the order of introduction of Skolem terms in the proof). Since $\sqsubset$ is a total order, these constraints ensure that any pair of Skolem terms are associated with distinct nodes in the tree. The constant ordering theorem allows us to impose this constraint on trees and substitutions without loss of generality. For, the constant ordering theorem says that it is indeed impossible in any solution θ for a path $(\pi_d d)\theta$ to be a prefix of $(\pi_c c)\theta$ when $c \sqsubset d$.

We may now assume that the symbol identifying each node n in t is the unique Skolem term c for which $\hat{c} = n$ (if one exists). This ensures that $c\theta$ takes the form xc. To ensure that x is the empty string, we add further constraints. To describe the node for constant c with prefix π_c, we find the node u representing π_c and add the constraint $u <_I c$, meaning that c is a child of u. $u < v$—meaning u is a proper ancestor of v—can be defined as the conjunction of $u \le v$ and $v \not\le u$. Then $u <_I v$ can be defined as the conjunction of $u < v$ and $v \Rightarrow u$. With this constraint, $\hat{c}$ must be the unique node on the path from the node representing π_c to $\hat{c}$.

A similar constraint manages the values of variables. To introduce a node for variable v with prefix π_v, we find the node u representing π_v and we add a constraint appropriate to the logic: $u <_I v$ for *K*, $u < v$ for *K4*, $u \le v$ for *S4* and $u \le_{\le 1} v$ for *T*. $u \le_{\le 1} v$—meaning v is u or a child of u—can be represented as the conjunction of $u \le v$ and $v \Rightarrow u$.

As a final step, we should stipulate arbitrary symbols to correspond to each node in the tree to which no Skolem term is assigned. In fact, however, the constant ordering theorem ensures that any substitution that solves the equations includes no such arbitrary symbols. This step is therefore superfluous; the constraints identified thus far describe only trees t that encodes possible solution substitutions of values to variables.

Equations. The equations themselves that θ must solve are likewise realized as simple constraints on t. To equate $\pi_u u$ and $\pi_v v$, we add the constraint $u = v$—$u = v$ is equivalent to the conjunction of $u \leq v$ and $v \leq u$.

Domain constraints. Modal constraints on first-order unification are represented by associating a node u_t with each first-order variable or term t. Each first-order Skolem term f is introduced at some world μ, as recorded in an typing pair $f : \mu$. Because the arguments of f are introduced from the same formula as f at wider scope, the arguments will be associated with prefixes of μ. Thus, u_f is just the node corresponding to μ. Meanwhile, each first-order variable x is associated with a new node u_x which represents the first world in which the value of x is defined. The typing pair $x : \mu$ is represented by the constraint $u_x \leq v$, if v is the node corresponding to μ. This constraint may also be represented as an equation, given access to S4 variables: for $u_x \leq v$ we introduce a new variable l_x and add the equation $u_x l_x = v$. The variable l_x has a unique occurrence in the resulting set of equations. For proofs analyzing unification problems in terms of equations, it will be convenient to adopt this representation and give domain constraints and modal equations the same treatment.

Now, to impose the correct domain constraints, we simply extend any ordinary first-order unification algorithm so that when first-order terms t and s are unified, the corresponding nodes u_t and u_s are constrained to be equal. If modal variables appear as arguments of first-order Skolem terms, they can also be unified by imposing equality constraints. When the overall unifier is computed, the domains of definition of all unified terms will refer to the same, correct nodes in the tree; and necessary constraints on the values of variables will be respected. ■

The problem of unifying annotations is therefore equivalent to the problem of solving a set of simple tree constraints. I now present an efficient algorithm to solve this problem. The algorithm extends the tree construction algorithm of [Aho *et al.*, 1981], which handles $\leq$ and $\not\leq$.[1] In the algorithm, the node corresponding to a variable or constant u is represented as the least common ancestor of a distinct pair of leaves u_1 and u_2, denoted (u_1, u_2). The tree is constructed by grouping leaves into sets according to the constraints. A set of disjoint sets is constructed for each depth in the tree; nodes in the same set at depth n indicate leaves that must be descendants of the same node at depth n in any tree that solves the constraints. In this process, we need only consider N levels of partitions, where N is the number of leaves of the tree. If a tree satisfying the constraints exists, a tree satisfying the constraints exists that has only branching nodes—because all constraints refer to least common ancestors,

[1]Be warned: [Aho *et al.*, 1981] use $u \leq v$ with the opposite sense I do; their notation conflicts with the present intuition that the tree represents a collection of paths from the root to leaves, ordered by the prefix relation.

which must be branching nodes. So the tree has depth at most N. Correspondingly, should we discover the need to merge two cells at depth N, we will know that the constraints have no solution.

Given a set of constraints C, algorithm $\mathcal{A}$ computes a tree by applying the following rules for merging partitions:

1. Initial. All leaves are in the same cell at depth 0.
2. $(i,j) \leq (k,l)$. If i and j are in the same cell at depth n, then i, j, k and l are in the same cell at depth n.
3. $(i,j) \nleq (k,l)$. If i, j, k and l are in the same cell at depth n, then i and j are in the same cell at depth n+1.
4. $(i,j) \Rightarrow (k,l)$. If i and j are in the same cell at depth n+1, then i, j, k and l are in the same cell at depth n.

These rules respect the following natural property:

Lemma 17 (sanity) *If i and j are in the same cell at depth n+1 (because of a proof of length h), then i and j are in the same cell at depth n (because of a proof of length at most h).*

Proof. By induction on the length of the proof that i and j are in the same n+1 cell. ■

Accordingly, $\mathcal{A}$ ends by building an internal node at depth n+1 for each non-unit cell there, and making it a child of the internal node at depth n which it is a subset of. (The sanity lemma ensures that there will be at least one such node; the disjointness of partitions ensures that there will be at most one.) Leaves attach to the greatest depth non-unit cell to which they belong.

Theorem 6 (correctness) *Any tree t so constructed satisfies the constraint set C.*

Proof. As in [Aho *et al.*, 1981], by consideration of constraints. For example, for a constraint $(i,j) \Rightarrow (k,l)$, let S be the partition associated with (i,j) in t. Rule 4 must have fired, putting i, j, k and l in the partition of the parent of (i,j). Since (k,l) must be a descendant of this node, the constraint is satisfied. ■

We prove that the algorithm is complete by means of a lemma:

Lemma 18 (descendants) *Let t be any tree satisfying constraint set C, and let S be a cell at depth n containing more than one leaf. Then there is a node b in t of depth n such that every leaf in S is a descendant of b.*

Proof. As in [Aho *et al.*, 1981], by induction on the number of steps of rule-application in constructing partitions. For example, consider a step for $(i,j) \Rightarrow (k,l)$ causing i, j, k and l to be in the same cell at depth n. By induction hypothesis, there is a node b_1 at depth n+1 in

t which dominates all leaves in i and j's cell in S at depth n+1. Moreover, nodes in t at depth n must dominate the unmerged cells of i, j, k, and l in S. Now we can show that it must be a single node that dominates all of them: the parent b_0 in t of node b_1. Since t satisfies the constraints, (k,l) is a descendant of the parent of (i,j) in t; since $b_1 \leq (i,j)$, $b_0 \leq (k,l)$ in t. ■

In fact, the proof of the descendants lemma is straightforwardly extended to the following least commitment property. Let T be any set of trees satisfying constraint set C. Initialize algorithm $\mathcal{A}$ with nodes i and j in the same cell at depth d according to any relation $r(i,j,d)$ which holds only if every tree in T assigns (i,j) to a node at depth d or greater, and run algorithm $\mathcal{A}$ to completion. Then for any cell in S containing more than one leaf at any depth n, there is in every tree in T some node b of depth n such that every leaf in S is a descendant of b.

Theorem 7 (completeness) *If algorithm $\mathcal{A}$ returns no tree, no tree satisfies the constraints.*

Proof. The procedure succeeds unless two nodes are in the same partition at depth N—in which case we terminate the algorithm and report failure. By the descendants lemma, this means that any solution has two nodes together at depth N—so any solution has depth greater than N. But we've already observed that if there is a solution, there is a solution with depth at most N, so in this case there must in fact be no solution. ■

Algorithm $\mathcal{A}$ can be performed in time $O(MN \log N)$, where M is the number of constraints and N is the number of leaves in the tree. Cells are represented using a union-find algorithm [Hopcroft and Ullman, 1973]; each cell stores not only a set of nodes but also a set of productions that may be triggered when this set is merged with another set. Considering only the shorter list when two cells are merged ensures that only $O(M \log N)$ productions are considered in merges of cells at any given depth in the tree.

If a proof attempt $\mathcal{D}$ contains K rule-applications, this means that algorithm $\mathcal{A}$ contributes time $O(K^3 \log K)$ toward constructing a unifier under which $\mathcal{D}$ is a proof. There can be no more modal constants and variables than rule applications, since each has its origin in some rule application, so N is $O(K)$. Likewise, there are $O(K)$ first-order variables, which can be unified by imposing a linear number of equalities between terms by standard algorithms [Martelli and Montanari, 1982]. There are $O(K)$ equalities imposed by axioms. However, the simple presentation of algorithm $\mathcal{A}$ above requires adding $O(K^2)$ constraints to enforce the distinctness of constants.

This running time can be brought down to $O(K^2 \log K)$ by a specialized representation of the distinctness constraints. Only the distinctness constraints corresponding to the $\sqsubset$-least constant pair in a cell need be triggered at each step. The other distinctness constraints will only duplicate their effects. If we can identify the relevant constraints, we can ensure that only $O(K)$ production-firings are needed to keep constants distinct. But the $\sqsubset$-least constant pair in each cell is easily maintained, since $\sqsubset$ is given and inspection of the rules shows that a pair (c_1, c_2) are together in any cell dominating c_1 and any other leaf.

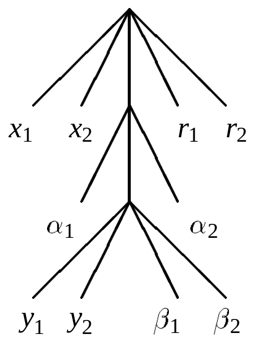


Figure 4.1: Tree for $xy = \alpha\beta$.

In algorithm $\mathcal{A}$, constraints corresponding to additional equations can be added dynamically, because the trees this algorithm produces make the least possible commitment. This is a consequence of the (generalized) descendants lemma. The only commitments the algorithm makes are that strings $\pi_u u$ and $\pi_v v$ share a prefix of a given length. That is, if u_1, u_2, v_1, and v_2 are members of a common cell at depth k in the tree, we know the value of $\pi_u u$ and $\pi_v v$ share a prefix of length at least k. Other features of the tree, for example the ancestor or command relation of prefixes $\pi_u u$ and $\pi_v v$, may be changed if possible by merging the appropriate cells later. Now, because of the descendants lemma, we know that any nodes in the same cell at depth k in algorithm $\mathcal{A}$ are children of some node of depth k in every tree that satisfies the constraints. That means that if algorithm $\mathcal{A}$ constructs a unifier that assigns $\pi_u u$ and $\pi_v v$ a common prefix of length k, EVERY unifier assigns $\pi_u u$ and $\pi_v v$ a common prefix of length at least k.

4.3.2 An Example

Let us return to the simple example of figures 2.17, 2.18 and 2.19. We start with the equation $xy = \alpha\beta$. In $S4$, this corresponds to the following constraints, if (r_1, r_2) names the root (or real world):

$$(r_1, r_2) <_I (\alpha_1, \alpha_2) \quad (\alpha_1, \alpha_2) <_I (\beta_1, \beta_2)$$
$$(r_1, r_2) \leq (x_1, x_1) \quad (x_1, x_2) \leq (y_1, y_2) \quad (y_1, y_2) = (\beta_1, \beta_2)$$

The algorithm computes the tree shown in figure 4.1. The first $<_I$ constraint causes α_1 and α_2 to merge at depth 1; then the second $<_I$ constraint causes β_1 and β_2 to merge under α_1 and α_2 at depth 2; finally, the $=$ constraint merges y_1 and y_2 with this cell at depth 2. At this point, the tree satisfies all the constraints, and solves the needed equation. Note that x is provisionally identified with the root, in keeping with the algorithm's policy of leaving the endpoints of path variables as close to the root as possible.

Recall that we had to impose one of three equations on x to finish the proof: $x = 1$, $x = \alpha$, or $x = \alpha\beta$. Each of these can be imposed by adding additional constraints to the problem in progress. The first causes no further merges; the second merges x_1, x_2 and α_1 at depth 1; the third merges x_1, x_2 and β_1 at depths 1 and 2.

4.3.3 Problematic Interactions in Multi-modal Languages

Efficient multi-modal deduction requires some limitations on introspection axioms, because some combinations introduce ambiguities that allow hard problems to be encoded into unifications of modal indices. These ambiguities are not associated with the problem of determining what types a string has. As the following result shows, the type interactions in the multi-modal language remain quite simple.

Lemma 19 (subset lemma) *If $\sigma : j$ is derivable, and σ' is a nonempty string containing only symbols that appear in σ, then $\sigma' : j$ is also derivable.*

Proof. By induction on the height of derivations of typing judgments. For (AX_t) and (VER_t), there is nothing to prove, since only atomic strings are involved. For (INC_t), we derive $\sigma : j$ from $\sigma : i$. Apply the induction hypothesis to the derivation of $\sigma : i$ to show $\sigma' : i$. Then reapply (INC_t) to show $\sigma' : j$. For (PI_t), σ has the form $\mu\nu$ and we derive $\sigma : j$ from $\mu : j$ and $\nu : j$. Each symbol α in σ' appears either in μ or ν, so by applying the induction hypothesis to the appropriate subderivation, we may derive $\alpha : j$. The new proofs, one for each symbol in σ', may be combined in the appropriate order by successive applications of (PI_t). ■

Instead, the problematic ambiguities of multi-modal deduction arise in logical theories which force a modal path to have SEVERAL DIFFERENT TYPES because of the different formulas which must apply along the path. As a characteristic example, consider the following interaction axioms:

$$\begin{array}{cc} \Box_{\text{Y}} A \supset \Box_{\text{YT}} A & \Box_{\text{Y}} A \supset \Box_{\text{YF}} A \\ \Box_{\text{Y}*} A \supset \Box_{\text{Y}*} \Box_{\text{Y}*} A & \Box_{\text{Y}*} A \supset A \\ \multicolumn{2}{c}{\Box_{\text{Y}*} A \supset \Box_{\text{Y}} A} \end{array}$$

These axioms relate K modalities YT and YF to a more specific K modality Y and to a still more specific S4 modality Y*. The following theory provides an illustration of an associated ambiguity:

$$\Box_{\text{Y}} \Box_{\text{Y}*} C \ \wedge \ \Box_{\text{Y}*} (\Box_{\text{YF}} C \supset \Box_{\text{Y}*} B) \ \wedge \ \Box_{\text{YT}} B \supset A$$

To prove A, we backchain against $\Box_{\text{YT}} B \supset A$, introducing constant α. Applying the second clause introduces a two variable string uv that must match α and a new goal C to be proved at world $u\beta$. The first clause introduces variables yz that reach this world. Thus, the proof attempt for A gives rise to equations:

$$uv = \alpha, yz = u\beta$$

These equations are governed by the typing context:

$$u : \text{Y*}, v : \text{Y*}, y : \text{Y}, z : \text{Y*}, \alpha : \text{YT}, \beta : \text{YF}$$

There are two solutions:

$$\{u = \alpha, v = \epsilon, y = \alpha, z = \beta\}$$
$$\{u = \epsilon, v = \alpha, y = \beta, z = \epsilon\}$$

In these solutions, the variable y must be bound to exactly one of α and β. In a mono-modal language, there is no problem with such ambiguities in values; as long as different values of a variable have the same length, the only way to force a particular resolution of the ambiguity is to impose an equation that specifies exactly which value the variable should have. These explicit equations can be included straightforwardly into a set of constraints. In the multi-modal language, we have a more general method of forcing such ambiguities to be resolved. Since α has type YT and β has type YF, we can think of y as encoding a boolean variable whose value is determined by the type of the string unified with y. If we add additional conditions for establishing C, the multi-modal language could allow us to impose new equations that TEST which kind of value y has. Interaction axioms allow these tests to impose disjunctive constraints on the values of several variables at once. We obtain the following result by following this strategy of describing possible assignments of values to a variable using types, and by using types to impose disjunctive constraints on those values:

Theorem 8 *It is NP-hard to determine whether there is a solution to the equations resulting from a proof attempt in a $\Box$-only modal where S4 and K modalities interact by unrestricted* (INC) *axioms.*

Proof. By reduction from 3-SAT. In 3-SAT, we are given a set of disjunctions, each containing three propositional literals (letters or negations of letters). The problem is to determine whether there is an assignment of true or false to the letters under which each disjunction is true. As with the proof of Theorem 1, we describe the proof in two steps, giving first a set of equations corresponding to the 3-SAT instance, then a proof search problem that gives rise to these equations.

The string equations first construct a string σ constrained so that its possible values encode assignments of truth-values to proposition letters. Meanwhile, each disjunction is associated with a string δ_i which unifies with σ if and only if the disjunction is true on the assignment σ represents. Thus, the unification problem is completed by equating σ with each δ_i.

In particular, the encoding uses the following definitions of modalities. For each proposition letter y, we have modalities YT, YF, Y and Y* related by the axiom schemata described in the previous example. An additional S4 modality * has inclusions to all of these modalities. For each disjunct i, we introduce add a K modality I with inclusion axioms to the modal types corresponding to the cases when i is true. For instance, if the ith disjunct is $(u \vee \overline{v} \vee y)$, we get:

$$(\Box_{\mathrm{I}}\, A \supset \Box_{\mathrm{UT}}\, A) \wedge (\Box_{\mathrm{I}}\, A \supset \Box_{\mathrm{VF}}\, A) \wedge (\Box_{\mathrm{I}}\, A \supset \Box_{\mathrm{YT}}\, A)$$

Given these inclusions, we can represent δ_i by a string $u_i t_i v_i$, where u_i, t_i and v_i are fresh variables governed by the typings:

$$u_i : *, \ \ t_i : \mathrm{I}, \ \ v_i : *$$

For any string σ that contains no constants of type I, δ_i is unifiable with σ if and only if σ is a string containing a constant whose type characterizes an assignment, specifying truth or falsehood for a literal, under which the *i*th disjunction is true.

To construct a single, overall assignment string, we repeatedly invoke the equations of the preceding example. If σ_{k-1} is an assignment to the first *k*-1 proposition letters, we can extend the assignment to the *k*th letter by adding the equations:

$$m_{k1} u_k v_k = \sigma_{k-1} \alpha_k, \ \ m_{k2} y_k z_k = m_{k1} u_k \beta_k$$

Under the typing context:

$$m_{k1} : *, \ \ m_{k2} : *, \ \ u_k : \mathrm{Y}*_k, \ \ v_k : \mathrm{Y}*_k, \ \ y_k : \mathrm{Y}_k, \ \ z_k : \mathrm{Y}*_k, \ \ \alpha_k : \mathrm{YT}_k, \ \ \beta_k : \mathrm{YF}_k,$$

For the reasons described above, $m_{k2} y_k$ includes the string σ_{k-1} as the value of m_{k2}, followed by either α_k or β_k as the value of y_k. This establishes an assignment σ and completes the construction of a unification problem that corresponds to the 3-SAT instance.

We now present a logical theory Γ where proof search for $\Gamma \longrightarrow G$ gives rise to this equational unification problem. The theory refers to propositions A_y, B_y and C_y for each variable y and a proposition D_i for each conjunct i. For the first variable, we add $A_y \supset G$. For consecutive variables y and z, add statements of the following form:

$$\Box_* \Box_{\mathrm{Y}} (A_z \supset \Box_{\mathrm{Y}*}\, C_y) \wedge \Box_* \Box_{\mathrm{Y}*} (\Box_{\mathrm{YF}}\, C_y \supset \Box_{\mathrm{Y}*}\, B_y) \wedge \Box_* (\Box_{\mathrm{YT}}\, B_y \supset A_y)$$

For the last variable, for which there is no successive z, replace A_z with the conjunction of the D_i. Finally, for each disjunct i, add:

$$\Box_* \Box_{\mathrm{I}} \Box_* D_i$$

In this theory, the proof attempt for G gives rise to the unification problem observed above. The successive proofs for A_y, B_y and C_y chain together to construct an assignment as outlined in the example; the need to prove each D_i in the ultimate, nested world introduces equations $\delta_i = \sigma$. ∎

The practical relevance of this result is unclear, as the axioms involved in this construction are rather pathological. The example of section 2.1.1, involving the Navy and GE, is a typical example of a useful modal representation that exercises a variety of introspection axioms on related modalities without introducing the kinds of pathologies that Theorem 6 exploits—a point which we justify more carefully presently. Because of the rarity of problematic examples, the response of [Debart *et al.*, 1992; Baldoni *et al.*, 1993] would seem natural: they provide a complete solution and to trust that programmers will never unwittingly hack hard problems into $\Box$ manipulations. However, since algorithm $\mathcal{A}$ exploits an invariant that no longer holds in this domain, it is problematic to make it complete for these cases without spoiling its performance for easy problems. The next section adopts a different approach: it gives broad syntactic conditions on the interaction axioms for modalities and axioms about those modalities that ensure that unification of modal paths remains easy.

4.3.4 Restrictions for Multi-modal Languages

Problems arise in multi-modal languages when the same variable may be unified with constants of different types under different unifiers. Under these conditions, variables can take on binary values, and a constraint algorithm that computes a simplest unifier becomes impossible.

I have found that the theories needed for many practical applications have syntactic characteristics that eliminate such ambiguities. This section explores two such characteristics. In the first, typing conditions in fact reduce to length conditions; this is representative of typing in planning representations. In the second, conditions can be satisfied by relaxation; this is representative of typing in logic programming representations.

Typing by Length Constraints

We first observe that a simple but useful syntactic restriction of HOMOGENEITY on the multi-modal language allows algorithm $\mathcal{A}$ to be used directly. The restriction starts from a distinguished negative modality $\Box_{\mathrm{N}}$. The theory T specifying relations between modal operators is homogeneous just in case it contains an inclusion $\Box_i\ A \supset \Box_{\mathrm{N}}\ A$ for every modality $\Box_i$. Further, the sequent $\Gamma \longrightarrow \Delta$ to be proved is homogeneous just in case the Γ formulas have P-syntax and the Δ formulas have N-syntax according to the definitions below:

$$\begin{aligned} P &::= \ A \mid P \vee P \mid P \wedge P \mid N \supset P \mid \Box_i P \mid \forall x.P \mid \exists x.P \\ N &::= \ A \mid N \vee N \mid N \wedge N \mid P \supset N \mid \Box_{\mathrm{N}}\ N \mid \forall x.N \mid \exists x.N \end{aligned}$$

(A schematizes over atomic formulas.) Thus, any modality may appear in positive positions, but only $\Box_{\mathrm{N}}$ may appear in negative positions.

If $\Gamma \longrightarrow \Delta$ is homogeneous, then all modal Skolem terms in the proof must represent arbitrary $\Box_{\mathrm{N}}$ transitions. If also T is homogeneous, then each of these Skolem terms match variables of any modal type whatsoever. The different types of modal operators may therefore be specified completely in terms of the length of sequences of $\Box_{\mathrm{N}}$ operators they match. There are only four possibilities for introducing a node for variable v with prefix π_v, depending on whether the variable matches sequences of $\Box_{\mathrm{N}}$ variables of length 0 and length 2. (In any demonstration that the variable matches a sequence of length 2, we must have applied (PI$_t$); we can repeat the step to match any longer sequence.) To represent the variable, therefore, we find the node u representing π_v and add: (1) $u <_I v$ (only length 1); (2) $u < v$ (only length 1 or greater); (3) $u \leq v$ (any length); and (4) $u \leq_{\leq 1} v$ (only length 1 or less). Thus, we again obtain a sound and complete specification of the unification problem by incorporating first-order and distinctness constraints as in section 4.3.1. Algorithm $\mathcal{A}$ computes a solution or reports that none exists in time $O(K^2 \log K)$.

Typing for Least Commitment

A different kind of restriction is to enforce a strict uniformity in solutions that allows a unifier to be constructed by local modifications of a prospective solution. The central notions involved in this restriction are those of FORCED modalities and SEPARATE modalities. Given a theory T specifying relations between modal operators, we say a modality i is FORCED if i is not governed by (PI), or equivalently that $\epsilon : i$ cannot be derived. By extension, we say constants and variables are forced when they can be assigned a forced modality as a type. Forced variables are the ones that may insist on binary values. Modality i is SEPARATE from j (under T) if for every typing context Σ, there are no terms c and d for which we can derive $c : i$, $d : j$, and $cd : i$. The significance of separate modalities is this. Given variables u of type i and v of type j with i separate from j, there is at most one solution of any equation $uv = \sigma$. For suppose we had $u\theta$ a proper prefix of $u\theta'$ for two unifiers θ and θ'. Then $v\theta$ has a nonempty overlap d with $u\theta'$; by the subset lemma $d : j$. And now $u\theta : i$, $d : j$ and $u\theta d = u\theta' : i$. Separate modalities are to be distinguished from DISJOINT modalities: i and j are disjoint if there is no context Σ and term c for which $c : i$ and $c : j$.

We apply forcedness and separateness in three auxiliary notions. First, a modality i is SIMPLE if i is separate from i. A simple modality looks like a K modality even taking possible inclusions into account. Second, a modality i is CLEAN if whenever $\Box_i A \supset \Box_j A$ for forced j, then j is separate from i. A clean modality can never be responsible for ambiguities in the number and identity of forced constants on a string: it either always matches none or always matches exactly one. Finally, given the goal of proving $\Gamma \longrightarrow G$, modality i is UNAMBIGUOUS if G is a G-formula and Γ is a multiset of D-formulas, according to the following grammar:

$$\begin{array}{rcl} G & ::= & P \mid G \vee G \mid G \wedge G \mid D \supset G \mid \Box_k G \mid \forall x.G \mid \exists x.G \\ D & ::= & P \mid D \vee D \mid D \wedge D \mid G \supset D \mid \Box_i D_{i,\{\}} \mid \Box_k D\ [k \neq i] \mid \forall x.G \mid \exists x.G \\ D_{i,S} & ::= & P \mid D_{i,S} \vee D_{i,S} \mid D_{i,S} \wedge D_{i,S} \mid G \supset D_{i,S} \mid \forall x.G \mid \exists x.G \\ & & \Box_k D\ [i \text{ separate from } k\text{, each } j \text{ in } S \text{ disjoint from } k] \mid \\ & & \Box_k D_{i,S\cup\{k\}}\ [\text{K simple}] \end{array}$$

(P schematizes over atomic formulas.) An unambiguous modality is one whose interactions might be problematic in general, but happen not to be, given the manipulations of modalities in the particular logical theory in question.

Given interactions T and desired end-sequent $\Gamma \longrightarrow G$, we will require every modality to be either clean or unambiguous. Intuitively, because of the role of separateness in the definitions, by imposing the restriction, we ensure that the forced SKOLEM TERMS do not vary across unifiers. In turn, this ensures that forced VARIABLES have the same values across all unifiers; ambiguous forced variables are impossible. It is a consequence of this that equations have simplest solutions—not just in terms of lengths of values of variables but also in terms of the constants that appear on the values of variables. This result is in fact stronger than the least commitment result for mono-modal languages. Formally, we have:

Theorem 9 (agreement) *Let θ and θ' be two substitutions that solve the equations arising from $\Box$-only proof search for $\Gamma \longrightarrow G$ with interactions T, where every modality is either clean or unambiguous. Then for every forced Skolem term c, $\pi_u u\theta$ contains c if and only if $(\pi_u u)\theta'$ does.*

Proof. By induction on the number n of equations. For the base case, there are no equations, no Skolem terms, and nothing to prove.

Suppose the claim is true for the first $n \Leftrightarrow 1$ equations and consider solutions θ and θ' for the first n equations. Apply the induction hypothesis to show θ and θ' agree on the forced Skolem terms in the the first $n \Leftrightarrow 1$ equation, and consider the nth equation, E. By the variable introduction theorem, E has the form $l\vec{x} = r\vec{c}$, where l and r contain only terms which appear earlier, $\vec{x}$ is a sequence of new variables and $\vec{c}$ is a sequence of new Skolem terms. From the induction hypothesis, we know that the same forced Skolem terms appear on $l\theta$ and $l\theta'$, and likewise for $r\theta$ and $r\theta'$: so $E\theta$ has the same such terms as $E\theta'$. If E is an equation representing a domain constraint we are done: there is a unique new variable x and hence no new prefixes.

For other equations, we show by contradiction that there cannot be a Skolem term c, a forced modality j such that $c : j$, and a variable x of type i such that c appears in $(\pi_x x)\theta$ but not in $(\pi_x x)\theta'$. Suppose otherwise, and consider the first counterexample; we have two cases according to whether i is clean or unambiguous.

Suppose i is clean. Then j is separate from i, and since $c : i$ and $c : j$ we cannot have $\epsilon : i$. So i is forced. This means $x\theta'$ includes a forced Skolem term c', which precedes c since c does not appear in $(\pi_x x)\theta'$. By the constant ordering theorem, c' must precede c in $(\pi_x x)\theta$ also. But $x\theta$ cannot include both c' and c, since j is separate from i. And $\pi_x\theta$ cannot contain c': if so, some earlier variable would contain c' on one substitution but not the other, and we know c is first. Thus, if i is clean, our assumptions about c are incoherent.

Suppose i is unambiguous; this ensures that x is followed by a string $\vec{v}z$ where z is a variable of type h with i separate from h, and $\vec{v}$ is a sequence of n variables v_k each of simple type M_k disjoint from h. Why is this? The sequence of variables following x is constrained by the sequences of modalities permitted in $D_{\{i\}}$ formulas; the only alternative $\vec{v}z$ is for the equation to end before any z. But since c does not appear in $(\pi_x x)\theta'$ and appears in $r\theta$, x cannot be final. Nor can any v_k be final: since each matches exactly one Skolem term, any $(\pi_x x\vec{v})\theta$ must contain forced Skolem terms that $(\pi_x x\vec{v})\theta'$ does not.

So, given this string of variables $x\vec{v}z$, we compare $(x\vec{v}z)\theta$ with $(x\vec{v}z)\theta'$. Observe that

$v_1\theta' = c$ since $v_1\theta'$ must be forced and c is the first forced Skolem term from $x\theta$ not to appear in $x\theta'$. Continuing, we find $\vec{v}\theta' = \vec{c}$, for some string of n Skolem terms $\vec{c}$, and $z\theta'$ begins with some constant d. By separateness, d cannot appear in $x\theta$, so it must appear afterward. By disjointness, d cannot appear in $\vec{v}\theta$, so it must appear later still. But $\vec{v}\theta$ must include n constants following c; d must be one of them. This is absurd: we conclude that no counterexample can exist. ■

Theorem 10 (least commitment) *Given a set of equations U arising from a $\Box$-only proof attempt for $\Gamma \longrightarrow G$ without possibility or negation, and with every modality clean or unambiguous. Then if E has a solution, it has a solution θ such that if c appears on $(\pi_u u)\theta$ then c appears on $(\pi_u u)\theta'$ for any other solution θ'.*

Proof. The proof to a relation $\leq$ between unifiers; $\theta \leq \theta'$ holds if and only if any c that appears in $(\pi_u u)\theta$ also appears in $(\pi_u u)\theta'$. This relation is well-founded, since each unifier assigns values to only a finite number of variables, and those values are finite strings. Thus, it suffices to show that for any two unifiers θ' and θ'' there is a unifier θ with $\theta \leq \theta'$ and $\theta \leq \theta''$.

We show this by induction on the number of equations in U; we show simultaneously that for any u, $(\pi_U u)\theta = (\pi_U u)\theta' \cap (\pi_U u)\theta''$. That is, under θ, $(\pi_U u)$ contains exactly the constants it has both under θ' and under θ'', in the order dictated by the constant ordering theorem.

For the base case, zero equations, there is nothing to show.

Now, suppose we have constructed such a θ for the first $n \Leftrightarrow 1$ equations, and consider equation n, E, which involves k additional variables. As before, for each $x_i : i$, construct the value $x_i\theta$ inductively as follows:

$$x_i\theta = ((lx_1 \ldots x_i)\theta' \cap (lx_1 \ldots x_i)\theta'') \text{ minus } (lx_1 \ldots x_{i-1})\theta$$

$x_i\theta$ either only contains Skolem terms from $x_i\theta'$ or only contains Skolem terms from $x_i\theta''$. Otherwise there would be a Skolem term a that appears in $(lx_1 \ldots x_{i-1})\theta$ but not $(lx_1 \ldots x_{i-1})\theta'$ and a Skolem term b that appears on $(lx_1 \ldots x_{i-1})\theta'$ but not $(lx_1 \ldots x_{i-1})\theta$, and where moreover a and b appear in both $(lx_1 \ldots x_i)\theta$ and $(lx_1 \ldots x_i)\theta'$. This means that a precedes b in the θ' solution but b precedes a in the θ' solution—in conflict with the constant ordering theorem. So as long as $x_i\theta$ is nonempty, the subset lemma shows that $x_i\theta$ has type I. Meanwhile, if $x_i\theta$ is empty then I cannot be forced. If I is forced, the value of $x\theta'$ shares the forced Skolem terms with $x_i\theta''$; this follows by the previous result. These will appear on $x_i\theta$. Again, $E\theta$ is the intersection of $E\theta'$ and $E\theta''$, and θ solves E. ■

Relaxation for Least-Commitment Multi-modal Languages

This section outlines a relaxation algorithm for computing modal matches that repeatedly performs algorithm $\mathcal{A}$ and modifies the result to make progress toward type-checking. This progress is achieved using a straightforward procedure that computes the next-larger well-typed modal match for a particular equation. The arguments of section 4.3.4 show why an overall simplest global solution exists and are easily adapted to show why local improvements toward it are always possible.

We begin by presenting an algorithm for computing small well-typed modal matches. In principle, we will need to match the left term l and right term r of an equation. The value of r and the match for the variables and constants from l that appear in earlier equations will already be determined; this fixes a final string of Skolem terms from r that are unaccounted for. We need only match the final string of new variables in l against these constants, subject to any constraints on the values of those variables that we have already identified. Thus, we have the following task: we are given a string $\vec{v}$ of variables and a string $\vec{c}$ of Skolem terms, and a base substitution θ_0 with $\vec{v}\theta_0 = \vec{c}$. The problem is to find a unifier θ where for each variable v_i, $v_i\theta$ is well-typed and $(\pi_{v_i}v_i)\theta$ is as short as possible while still including $(\pi_{v_i}v_i)\theta_0$ as a prefix. As in the proof of the least commitment theorem, an argument from intersecting substitutions shows that if any match exists, one match assigns fewer Skolem terms to each prefix than any other; so we describe θ as the least match above θ_0 of $\vec{v}$ against $\vec{c}$—$lm(\theta_0, \vec{v}, \vec{c})$. Observe that θ restricted to the first i variables must be the least match above θ_0 of $\pi_{v_i}v_i$ against $(\pi_{v_i}v_i)\theta$. Otherwise we could use θ and the smaller prefix match to construct an smaller match on the whole string.

Thus, we characterize $lm(\theta_0, \pi_{v_i}v_i, \vec{c})$ as follows. No match exists unless $(\pi_{v_i}v_i)\theta_0$ is a prefix of $\vec{c}$. If this prefix condition is met, let $\vec{d}$ be the longest string of constants matching the type of v_i such that there is a μ where $\vec{c} = \mu\vec{d}$ and a $\theta_{i-1} = lm(\theta_0, \pi_{v_i}, \mu)$. If no such $\vec{d}$ exists, there must be no match. Otherwise, $lm(\theta_0, \pi_{v_i}v_i, \vec{c})$ is the substitution that sends v_i to $\vec{d}$ and otherwise agrees with θ_{i-1}. Given θ_0, this characterization can be operationalized directly as a dynamic programming algorithm that maintains a table of $lm(\theta_0, \pi, \mu)$ values for prefixes π of $\vec{v}$ and prefixes μ of $\vec{c}$.

This procedure can be combined with algorithm $\mathcal{A}$ to construct unifying trees for multi-modal languages. The combination, algorithm $\mathcal{B}$, goes as follows:

Construct constraints for the input equations U and the domain restrictions on first-order variables as in algorithm $\mathcal{A}$ and propagate the consequences of those constraints. Then, while changes occur: consider the equations in order until some sequence of variables lies lower than their next least match against the constants to which they are bound; perform merges of cells in $\mathcal{A}$ so as to bump those variables up into the least match configuration; and recompute $\mathcal{A}$. Whenever some sequence of variables has no next least match, fail.

Theorem 11 (correctness and completeness) *If algorithm $\mathcal{B}$ produces a tree, it is the least solution to its input equations U and the associated domain constraints; if there is a solution, $\mathcal{B}$ produces it.*

Proof. Any tree that algorithm $\mathcal{B}$ produces corresponds to a correct unifier θ of U. The fact the tree is a fixed point of algorithm $\mathcal{A}$ means that a substitution that solves U and satisfies the domain constraints can be extracted from the tree. Since the algorithm terminates only when every sequence of variables matches a path of the appropriate type, this substitution respects the types of variables and constants.

Moreover, any other correct unifier θ' assigns no prefix π a string $\pi\theta'$ shorter than $\pi\theta$; and if $\mathcal{B}$ returns failure, there is no unifier. We establish this using a somewhat stronger claim and induction on the number of equations k so far solved.

Call a relation r CONSERVATIVE FOR U when $r(\pi, \pi', n)$ entails that $\pi\theta$ and $\pi'\theta$ share a common prefix of length n in any solution θ of U. As remarked in section 4.3.1, the proof of correctness of algorithm $\mathcal{A}$ can be adapted to show that if $\mathcal{A}$ is initialized with leaves of nodes put in common cells according to a conservative relation and run to completion, then the output relation includes the input one but remains conservative. Given any conservative relation that solves the first k equations, we will show that the new relation induced by bumping up a sequence of variables $\vec{x}$ from equation k+1 to match $\vec{c}$ (by match θ) also includes the old one and remains conservative.

We use the claim to show by induction that input a conservative relation for U, algorithm $\mathcal{B}$ returns a conservative relation that includes the input and represents a solution to the first k equations. For 0 equations, there is nothing to show. Suppose the claim is true when running $\mathcal{B}$ on $k \Leftrightarrow 1$ equations and consider solving k equations. Following algorithm $\mathcal{B}$, we first use this induction hypothesis to solve the first $k \Leftrightarrow 1$ equations and extend the input relation conservatively. Then, we bump up the variables in the kth equation as dictated by the least match; by the claim, also extends the relation conservatively. We continue this process as needed until a fixed point is reached or until we discover the need to place a variable impossibly deep in the tree. Since the relations remain conservative, we lose no solutions. As no variable can be bumped past depth N in the tree, we must reach a fixed point which gives a least solution extending the input relation for the $k+1$ equations, if a solution exists.

We are left with the claim that including the match of $\vec{x}$ against $\vec{c}$ by θ keeps the relation conservative. Consider an arbitrary solution θ' in which $\vec{x}$ is matched against some different string $\vec{d}$. Because the d terms must appear in the first k equations, which have been solved conservatively, $\vec{d}$ must include at least the constants of $\vec{c}$ in order. Further, $\vec{d}$ must contain exactly the same forced constants that appear in $\vec{c}$, by the agreement theorem. We now know enough about the string $\vec{c}$ and the match θ' against $\vec{d}$ to construct a match of $\vec{x}$ against $\vec{c}$ at least as small as θ', by intersection (as in the least commitment theorem). Thus, if no match against $\vec{c}$ exists, there can be no other solution to the earlier equations which allows a match in this equation; so local progress is complete. Meanwhile, since we compute θ as the least match against $\vec{c}$, we can conclude θ is smaller than θ' as needed, so local progress is conservative. ■

Algorithm $\mathcal{B}$ runs in time worst case $O(N^4)$, where N is the number of variables and constants in U. The analysis given earlier is general enough to show that the multiple invocations of $\mathcal{A}$ require total time only $O(N^2 \log N)$. Meanwhile, algorithm $\mathcal{B}$ requires no more than $O(N^2)$ iterations to converge (otherwise some node must be bumped to depth N+1), and in each iteration there are at most $O(N)$ nodes to check. There are two kinds of checks; we shall see that these have different complexities. The first case, the easy case, is when the input substitution θ_0 is identical to the output substitution θ. All but the last of the

$O(N)$ checks that we perform fall into this class, since they introduce no changes. In the other case, we compute a new θ different from θ_0. Thus, if the time of an easy check is $f(N)$ and the time of a hard check is $g(N)$, algorithm $\mathcal{B}$ takes $O(N^2 \log N + N^3 f(N) + N^2 g(N))$. To compute the time each check takes, observe that successive variables must originate in the same formula occurrence. Thus the maximum number k of successive variables needed to be matched in each equation is bounded statically by the complexity of axioms. For current purposes, k can be considered constant. Likewise, since the possible interactions between modalities must be specified in advance (and hence can be computed in advance), we may assume that the relationship between the type of a constant and the type of a variable can be computed in constant time. Meanwhile, the string of constants matched has worst-case length $O(N)$. In principle, given these bounds on the input, computing θ requires filling a table of size $O(kN)$, where each entry may require checking $O(N)$ earlier entries. So $g(N)$ is $O(N^2)$. On the other hand, if the table is filled in by demand and the input match θ_0 is well-typed, we access (and compute) only $O(k)$ entries of the table. The easy checks thus require time $O(N)$. Thus, we conclude that algorithm $\mathcal{B}$ has worst-case complexity $O(N^4)$.

4.4 Summary

To achieve modularity in practice, we need more than a unification-based proof method that allows modularity to be enforced, like the SCLP calculus of Chapter 3. We also need a good way to reason about possible worlds. Section 4.1 puts the need for such a method into relief, by providing a new, formal demonstration of the complexity inherent in equational approaches to reasoning about possible worlds.

The positive results of this chapter identify a new invariant for deduction problems in modal logic, which naturally applies to SCLP in particular. This invariant leads to fast algorithms for correctly applying axioms in modal proofs, described in section 4.3. Presentations of modal logic in terms of modal equations thus provide both a theoretical tool for analyzing proofs and proof search in new ways and a practical tool for implementing fast deduction. The next chapter looks at the empirical impact of these techniques for implemented deductive procedures.

5 Evaluating Modal Deduction Methods

Chapters 3 and 4 have introduced two new techniques for modal deduction. We can now combine unification-based search with structural modularity of proof; and we can enforce modularity using a constraint algorithm over possible worlds. The DIALUP search engine employs both to interpret modal logic programs.

This chapter looks concretely at what advantages these techniques afford, and when. This evaluation partly involves comparing DIALUP as implemented against alternative versions without modularity or without constraints. These alternatives are easy to obtain, at least in simple cases. Structural modularity is disabled by eliminating the locality condition on the (restart) proof rule. Constraint reasoning is disabled by incorporating a modal equational unification module instead [Otten and Kreitz, 1996].

The evaluation also involves comparing DIALUP against theorem provers from different paradigms. One kind of competitor involves general first-order equational reasoners based on resolution. Another involves specialized proof methods based on structurally-scoped sequent calculus. As we shall see, to put the comparison fairly, one must augment DIALUP's distinctive constraints and modularity—where possible—with the other strategies applied in these systems for pruning search.

The bulk of the chapter, section 5.1, addresses performance on a benchmark problem of deciding the validity of formulas in propositional minimal logic. We close in section 5.2 by discussing DIALUP in light of the specific fit we intend for DIALUP in the context of Natural Language Generation (NLG). Speed isn't everything; NLG seems a task where the new approach provided by DIALUP is valuable just from a conceptual point of view.

5.1 A Case Study in Deduction Efficiency

To look concretely at questions of efficiency, we investigate the problem of deciding the validity of formulas of a particular logical language, minimal propositional logic. This problem is studied directly in some detail in [Tennant, 1992], but—because minimal logic is the negation-free fragment of intuitionistic logic which is in turn a fragment of S4 modal logic—research on intuitionistic and modal deduction also applies; here we will particularly refer to [Sahlin *et al.*, 1992; Dyckhoff, 1992].

The organization of this section is as follows. We briefly introduce propositional minimal logic in section 5.1.1, and motivate why its decision problem requires special-purpose search techniques. Then, in section 5.1.2, we describe these search techniques, and show how they improve performance on simple problems. Finally, in section 5.1.3 we consider broader evaluations—corpora of sample problems, random problems, and exponential series. Each has its bias, and its technical difficulties—and unfortunately, none is particularly representative of the queries a system is likely to face in practice (for example in NLG).

5.1.1 Propositional Minimal Logic

From a proof-theoretic point of view, propositional minimal logic (MPC) can be described as a restriction of propositional classical logic; see e.g. [Troelstra and van Dalen, 1988]. MPC modifies classical natural deduction by eliminating two rules:

$$\dfrac{\begin{array}{c}\vdots\\ \bot\end{array}}{p} \qquad \dfrac{\begin{array}{c}[\neg p]\\ \vdots\\ \bot\end{array}}{p}$$

The first allows anything to follow from a contradiction. The second is the strong classical version of *reductio ad absurdum*, according to which, if it is impossible for p to be false, p must be true. Formally, the rule infers p from a contradiction while simultaneously discharging an assumption of $\neg p$. This regime gives MPC proofs an attractive structure that recommends them for many applications. For example, Kanovich uses minimal logic deduction to synthesize programs [Kanovich, 1990]. Tennant relies on minimal logic deduction as a module in an envisaged system for commonsense and scientific reasoning [Tennant, 1992]. Because of these applications, MPC deduction (and the closely related system of deduction in propositional intuitionistic logic) is an important and well-studied problem.

Semantically, MPC can be viewed as a modal logic. Without the second rule, the MPC statement $p \supset q$ has the same meaning as a scoped statement $\Box(p' \supset q')$ in S4 modal logic. Without the first rule, $\neg p$ becomes equivalent to $p' \supset \Box f$, where f is a distinguished (but ordinary) proposition letter representing falsity in place of $\bot$. (This latter change embeds minimal logic in intuitionistic logic.) Overall, we get a translation T from MPC to S4:

$$\begin{aligned}
&T(A) = \Box A\\
&T(A \vee B) = T(A) \vee T(B)\\
&T(A \wedge B) = T(A) \wedge T(B)\\
&T(A \supset B) = \Box(T(A) \supset T(B))\\
&T(\neg A) = \Box(T(A) \supset \Box f)
\end{aligned}$$

Because of this modal interpretation, MPC provides a good test for the different approaches to modal deduction described in section 2.3, as well as the new strategies developed in Chapters 3 and 4.

The first lesson of MPC is that wholly general methods, which ignore key features of the MPC search space, are doomed to failure. The remainder this subsection illustrates this, by comparing how the general systems OTTER [McCune, 1994] and SPASS [Wiedenbach *et al.*, 1996] compare to FT [Sahlin *et al.*, 1992] on Pelletier's propositional problems. The problems are seventeen theorems of classical propositional logic reported in [Pelletier, 1986] as having proved problematic for past deduction systems, particularly those that used natural deduction directly for proof. Only four of the seventeen (numbers 9, 10, 11 and 13) are valid in MPC. Even when Pelletier's paper appeared, none represented a difficult problem for sophisticated deduction systems. This is confirmed by the performance of FT.

Clever special-purpose techniques

FT [Sahlin *et al.*, 1992] is an intuitionistic theorem prover founded on a structurally-scoped proof system much like that described in section 2.3.2. Search in FT is streamlined in several ways. FT implements a number of optimizations naturally available to any framework for MPC deduction (see section 5.1.2); and it also adopts some distinctive strategies to mitigate some of the redundancies in search associated with structurally-scoped proof.

For current purposes, the most important of these distinctive strategies is a device called IMPLICATION LOCKING. Implication locking reduces alternatives for search by enforcing a canonical order for reasoning involving implications; this order in fact commits FT to perform a cascade of forward-chaining inferences at each scope (or possible world) in the proof, and to delay transitions to nested scopes (or possible worlds) as long as this cascade proceeds. Note then that implication locking embodies a strategy different from logic programming. Because there are few distinct propositional symbols and identity is easily tested, the forward chaining set up by implication locking is clearly a good idea on propositional problems. However, a similar strategy would be difficult to mirror on an explicitly-scoped system, because of the increase in distinct symbols and matching complexity that would come with explicit management of worlds.

FT includes a decision procedure for MPC. On LINC, FT returns the correct result for each of the propositional Pelletier problems without measurable CPU time; in fact it solves them all within 30 milliseconds.

General equational techniques

The obvious alternative is is to translate the Pelletier problems into first-order equational logic and solve them using a general system. (This is exactly the methodology proposed in research on semantics-based translation; recall section 2.3.3.) Good general systems are OTTER [McCune, 1994] and SPASS [Wiedenbach *et al.*, 1996]. These systems are freely available; they were the top performing general first-order theorem provers at the CASC-13 competition in 1996. These programs are the product of years of effort; their search strategies have been carefully pruned and their representations involve highly-optimized C implementations.

OTTER and SPASS both accept input in clausal form and reason primarily using resolution; SPASS also includes a splitting rule for case analysis. To deal with equations, OTTER and SPASS combine REWRITING TECHNIQUES, to transform ground terms to canonical representations, and inference by PARAMODULATION, in which any subterm of a derived fact can be unified with one term in an equation and then replaced by the other term.

In comparison to FT, the cost of solving the provable problems can be significantly increased under the general methods; (51) reports LINC CPU times in milliseconds.

(51)

program	9	10	11	13
OTTER	290	820	60	220
SPASS	30	150	10	50

But these are the good results. On a large fraction of the problems that are not provable—11 of 13 for OTTER and 9 of 13 for SPASS—neither program is able to return a result given several seconds of computation time.

The contrast in performance between FT and SPASS in particular offers a striking contrast with [Hustadt and Schmidt, 1997]. They report results where SPASS vastly outperforms a tableau prover (with similar inference to FT). The difference is that [Hustadt and Schmidt, 1997] are working with *K* modal logic, which has easy and deterministic unification. The equational theory of S4—at least as treated by OTTER and SPASS—is causing trouble.

First, because of the equational theory, OTTER and SPASS are missing important information about the finiteness of the search space. The equational theory permits paths of unbounded length, so it is not surprising that OTTER and SPASS might fail to terminate in some cases. In fact, however, something even more basic is going wrong with OTTER and SPASS. On many examples the programs get caught in loops in which the general equational methods they use wind up enumerating terms for possible worlds in a very simple-minded way. Under paramodulation, a path xy can be unified with the associativity equation and replaced by a longer path $u(v;y)$ with $x = (u;v)$ for two new variables u and v. The logical apparatus counts the resulting formula as different from the original one, but the rule of paramodulation can apply to it again in the same non-productive way.

Such absurd dead ends can be avoided by taking the constraints of modal languages into account in unification. [Otten and Kreitz, 1996] provide one way to do that; their procedure, T-string unification, is a specialized algorithm for equational unification that takes the unique prefix property path terms into account (section 4.1). ([Schmidt, 1998] offers another alternative, which is less efficient but allows certain optimizations in the representations of path terms.) The T-string unification procedure takes two modal paths as input and returns a substitution that equates them as strings by nondeterministically reducing the strings according to the transformations in Figure 5.1. (Each step transforms an equation into a possible further equation to solve and a possible substitution to apply.) A key advantage of this formulation is that a prover need never derive alternative representations of paths except as part of a resolution step.

T-string unification has a number of further advantages. It uses the separator $|$ as a bookkeeping device to ensure that variables are never split—as $x = u; v$—unless splitting is necessary. Meanwhile, the different cases of transformation ensure that the set of unifiers is MINIMAL; no substitution is an instance of two unifiers, so the search space

$R1$	$\{\epsilon = \epsilon \mid \epsilon\}$	$\rightarrow \{\}, \{\}$
$R2$	$\{\epsilon = \epsilon \mid t^+\}$	$\rightarrow \{t^+ = \epsilon \mid \epsilon\}, \{\}$
$R3$	$\{Xs = \epsilon \mid Xt\}$	$\rightarrow \{s = \epsilon \mid t\}, \{\}$
$R4$	$\{Cs = \epsilon \mid Vt\}$	$\rightarrow \{Vt = \epsilon \mid Cs\}, \{\}$
$R5$	$\{Vs = z \mid \epsilon\}$	$\rightarrow \{s = \epsilon \mid \epsilon\}, \{V \leftarrow z\}$
$R6$	$\{Vs = \epsilon \mid C_1 t\}$	$\rightarrow \{s = \epsilon \mid C_1 t\}, \{V \leftarrow \epsilon\}$
$R7$	$\{Vs = z \mid C_1 C_2 t\}$	$\rightarrow \{s = \epsilon \mid C_2 t\}, \{V \leftarrow z C_1\}$
$R8$	$\{Vs^+ = \epsilon \mid V_1 t\}$	$\rightarrow \{V_1 t = V \mid s^+\}, \{\}$
$R9$	$\{Vs = z^+ V_1 t\}$	$\rightarrow \{V_1 t = V' \mid s^+\}, \{V \leftarrow z^+ V'\}$
$R10^*$	$\{Vs = z \mid Xt\}$	$\rightarrow \{Vs = zX \mid t\}, \{\}$

X,V,V_1,C,C_1,C_2 denote single characters with X unconstrained, V and V_1 are variables, C,C_1,C_2 are constants, and V' represents a new variable which does not occur in the substitution computed so far. s, t and z denote (arbitrary) strings and s^+, t^+, z^+ non-empty strings. *For $R10$, we must have $V \neq X$ as well as one of $s = \epsilon$, $t \neq \epsilon$ or X a constant.

Figure 5.1: T-string unification algorithm from [Otten and Kreitz, 1996, p. 251].

is explored systematically. (T-string unification has drawbacks as well; the steps differ greatly depending on the kinds of values that a modal operator can take, and no algorithm is yet available for multiple modal operators—much less for operators with interactions as covered in Chapter 4.)

The use of refined unification techniques like those of [Otten and Kreitz, 1996] already demands that we depart from general methods when performing translation deduction. Section 5.1.2 shows that this is only one illustration of a pervasive phenomenon.

5.1.2 Invariants of Search in MPC

We have already noted several features of modal proof that general equational resolution methods like those in OTTER and SPASS do not exploit, including structural modularity (section 2.3.3) and the unique prefix property of path terms. But there are a number of optimizations that have been independently identified that turn out to be extremely important in narrowing the MPC search space. Collectively, they have the effect of rewarding approaches like tableau or sequent calculus provers, which (by contrast to resolution, at least) explore the MPC search space in a highly-structured way. Tennant offers a systematic exploration of many of these regularities (and others) [Tennant, 1992].

First, MPC provers can enforce a prohibition on identical nested assumptions; this guarantees decidability. Second, MPC allows provers to place strong constraints on backtracking, because failures on separate branches are typically independent (and are always independent in some proof systems). Third, the structure of MPC can be exploited in translation methods to radically cut down the number of sequence variables involved in translation. Finally, there are some simple but effective ways to reuse saved results.

Prohibitions on Identical Nested Assumptions

Assumptions in MPC search can often be avoided. Here is why. In an MPC model, if a formula A is true at a world μ, A must also be true at any world $\mu\nu$ accessible from μ. So suppose that in some branch of the search space, we have assumed A at μ, and we are considering proving a formula $A \supset B$ at a world $\mu\nu$. Semantically, $A \supset B$ requires that B be true at any accessible world where A is true. But because of our assumption, we know that A is in fact true at all accessible worlds. So it suffices show B true at all accessible worlds. Thus, in this context the goal $A \supset B$ reduces to B.

This observation shows that MPC is decidable; exploiting this observation in proof search guarantees termination. The reason is that, according to this observation, paths in a proof will only grow as long as new assumptions can be made. But only special formulas can introduce new assumptions along a path: formulas containing a negative occurrence of an implication (like an assumption $(A \supset B) \supset C$). Once each such formula has been used along a path, the path can no longer be extended. Thereafter, propositional reasoning will settle what formulas can be proved true along that path—resulting in a proof or failure.

There are many ways to exploit this observation in search. Dyckhoff realizes it in a sequent calculus with revised rules for implication [Dyckhoff, 1992]. This calculus builds in the constraint into the structure of sequent calculus rules, a way that dispenses with testing for redundant assumptions. Alternatively, Korn and Kreitz propose a translation of MPC to classical propositional logic based on using this observation to constrain enumeration of relevant worlds [Korn and Kreitz, 1997]. Unfortunately, both Dyckhoff and Korn and Kreitz's proposals turn out to interfere with goal-directed search—Dyckhoff's calculus requires inferences to appear in a special configuration while Korn and Kreitz must enumerate possible worlds before it is known whether those worlds will contribute to any proof. FT implements the test on nested assumptions directly, and our implementations will do the same.

Strong Constraints on Backtracking

The second optimization, called SIFTING in [Sahlin *et al.*, 1992], refines the use of backtracking. In general, backtracking is a bookkeeping method for exhaustively exploring the search space for proofs. For binary rules, which decompose an overall problem into two subproblems, backtracking enumerates all the ways that a solution for the first problem can be combined with a solution for the second. In the case of a propositional problem, such exhaustiveness is superfluous. The problems will be independent. Sifting is an optimization of backtracking that reflects this.

This optimization is quite perspicuous in the sequent calculus, particularly under a logic-programming discipline. For example, suppose a (structurally scoped) MPC prover encounters the goal $A \wedge B$, and breaks it down into subproblems of proving A and proving B. Suppose a proof is found for A, the prover moves on to prove B, and there fails. After this failure, ordinary backtracking will look for alternative proofs of A, expecting to search again afterwards for a new proof of B. (It must exhaust all ways to combine proofs of A

with proofs of B.) This is hopeless. No matter what other way A is proved, the proof of B afterwards will still fail. Sifting allows the goal of $A \wedge B$ to fail immediately, once the subgoal B fails.

With semantics-based methods, sifting remains possible, but must be argued for and implemented more carefully. Conjunctive goals are no longer so obviously independent, because it might be possible to bind modal logic variables by unification as part of finding a proof. Actually, with logic programming search and incremental equational unification, we can show that this worry is unfounded. As we observed in Lemma 13, whenever right rules apply under logic programming search, all of the path variables in the sequent must equal labels of some current, global goal. Thus, the variables will be set to ground values by whatever unifier is being maintained along with the proof. There can be no communication between the proof of A and the proof of B using such variables. So if the proof of B fails, the whole proof still must fail.

The argument is still more subtle in the case of a constraint unification method. Indeed, it will be necessary to admit some backtracking; the values of variables are now significant. However, we can still see sifting as a strong optimization. Conceivably a change in the value of any variable might be enough to move to an unexplored part of the search space, making the sifting test very expensive. However, in fact, only the label of the goal formula needs to be examined. We know that if any formula contributes to the proof of A, that formula must be labeled with a prefix of the label of A. Under logic programming search, every assumption ends in a constant, so an assumption cannot be used in the first proof of A unless the constant it ends with appears on the value of the label of A.

When the proof of B fails, we have in fact derived that any proof of B is incompatible with the presence of these constants on the label of A and B. However, a proof of B might succeed if the prefix associated with A contained fewer or different constants. So the thing to do in the constraint-based method is to post this distinctness constraint and then to backtrack into the proof of A.

Streamlined Translations

In the case of translation systems, it is possible to effect a further simplification by refining the translation itself. The idea is to insert transitions to new possible worlds only where they are strictly needed. For example, $(a \supset b)$ is currently translated $\Box(\Box a \supset \Box b)$. This translation makes for expensive path reasoning. To use this as an assumption first involves unifying a path term xy—sure to introduce ambiguities into the solution space. Then the new subgoal involves considering yet another new possible world. Now, in the minimal logic translation, a is provable exactly when $\Box a$ is. That means that the translation of $(a \supset b)$ could be simplified just to $\Box(a \supset b)$. This simpler formula introduces no new possible worlds and involves a path term x which does not introduce equational ambiguities. This formula therefore puts a much lower demand on path reasoning.

Let $Q(A)$ be A if A occurs negatively (as a goal) and $\Box A$ if A occurs positively (as an assumption). Then the we can use the optimized translation G given below to convert from

MPC to S4.

$$\begin{array}{ll} G(A) = Q(A) & L(A) = A \\ G(A \wedge B) = G(A) \wedge G(B) & L(A \wedge B) = L(A) \wedge L(B) \\ G(A \vee B) = G(A) \vee G(B) & L(A \vee B) = G(A) \vee G(B) \\ G(A \supset B) = \Box L(A \supset B) & L(A \supset B) = G(A) \supset L(B) \end{array}$$

G stands for GLOBAL—it checks that the formula is true at any accessible world. L stands for LOCAL and performs a more restricted check against the current world.

An interesting consequence of the use of G is that Hereditary Harrop formulas—in this fragment, formulas without disjunction [Miller *et al.*, 1991]—have very attractive translations. Path logic variables can occur at most once on any positive path through the formula, which means that no ambiguities will result in solving path equations, even with equational unification. Under logic programming discipline, search in the G-translation will mirror λProlog search on the untranslated formula. Unfortunately, this result cannot be taken to confirm that equational unification is a good idea in these cases; rather it shows that for Hereditary Harrop formulas we ought to dispense with worlds altogether, and use a purely propositional search strategy (indeed, most likely a forward-chaining one, as in FT).

Recording Results

Two optimizations for saving results can also be used. Tennant reports that they are very important to his provers. One involves saving successful proofs for possible reuse. The other involves recording failed attempts at proof so that they can be immediately dispatched if they recur. These optimizations could be applied more or less the same way in any sequent proof system. What differs is the complexity of bookkeeping involved in recording a search result and accessing it later. (This again depends on the order in which search proceeds and the abstraction at which search problems are represented).

In fact, FT does not apply either of these optimizations. This is the main reason why Korn and Kreitz are able to show dramatic speedup over FT on pigeonhole problems [Korn and Kreitz, 1997]. With saved results, proofs for pigeonhole problems can be exponentially shorter than any proof available without saving results [D'Agostino, 1992]. For widest coverage, however, avoiding these optimizations seems reasonable. For example, anticipating (52), I have found that the bookkeeping involved in these methods on the constraint method costs more in execution time than it saves in pruned search. (What's more, with expressive representations, the bookkeeping becomes rather difficult to code correctly.)

Tracking the Improvement of Proof Search

For the special case of MPC deduction, I implemented the logic programming search strategy described in Chapter 3, the constraint algorithm described in Chapter 4, and the T-string unification algorithm described in [Otten and Kreitz, 1996]. The implementations are written in SML, and are parameterized by the optimizations introduced in this subsection (obtaining a decision method by restricting assumptions, sifting search problems, optimized translations, and saving results). (52) shows total run time in LINC CPU milliseconds for the propositional Pelletier problems, as these optimizations are factored in.

(52)

technique	decision	sifting	translation	saving
unification	330	80	60	70
constraint	560	160	110	310

Saving results is clearly not worth the trouble on such simple problems. Otherwise, we arrive at methods that are well within the ballpark of FT and don't suffer nearly the difficulties of general translation methods.

The reader may perhaps be disappointed that the T-string unification implementation runs in half the time that the constraint method requires. The performance difference is inevitable. The overhead of the T-string method is much lower than the constraint method, and these problems are easy. (Recall the discussion of Hereditary Harrop formulas earlier.) In fact, in solving the entire suite of problems, there isn't one occasion on which the T-string unification method finds a first unifier for an equation, rejects it, and actually has to backtrack to consider an alternative unifier for the equation!

5.1.3 Broader Comparison and Evaluation

We now have three provers for MPC—the benchmark FT, a new unification-based prover UNI and a new constraint-based prover CON—that cope well against a simple baseline. This section compares these provers more systematically. Three paradigms are common in the literature: banks of test examples, as in [Pelletier, 1986; Tennant, 1992; Sahlin *et al.*, 1992]; construction of difficult random problems, as in [Mitchell *et al.*, 1992; Giunchiglia *et al.*, 1997; Hustadt and Schmidt, 1997]; and the construction of families of problems of increasing complexity [Sahlin *et al.*, 1992]. We consider each in turn.

Test examples

For MPC, a suite of provable examples is documented in [Tennant, 1992]; 32, the ASSET problems, are derived from [Thistlethwaite *et al.*, 1987] and 33, the MINLOG problems, are derived from [Slaney, 1994]. Both sets have a similar form: each problem is an implication $p \star (q \star r) \supset (p \star q) \star r$, where $\star$ is a binary operation on propositions. The problems run 100–150 symbols.

Performance on these test suites is greatly variable, and on the whole rather disappointing. The reason is the structure of the formulas involved. (53) examines performance on the first five ASSET problems.

(53)

	1	2	3	4	5
FT	0	0	0	0	0
UNI	750	50	–	–	–
CON	6290	260	–	–	–

FT solves all without measurable CPU time. Meanwhile, for both UNI and CON, the first two take disturbingly long, the latter three unacceptably long. It turns out, however, that these problems are all Hereditary Harrop formulas; there is no ambiguity in the representation of worlds. The difference between UNI and CON derives entirely from the overhead of the (unnecessary) constraint solver, and much of the difference between UNI and FT derives from the unnecessary overhead of representing worlds in the first place. (The remainder must be due to the difference in search strategy.)

With the MINLOG problems, the results are rather more encouraging. Performance for the first ten is summarized in (54).

(54)

	1	2	3*	4*	5	6	7*	8*	9	10
FT	0	0	60	50	10	0	20	0	10	0
UNI	20	190	10	20	110	20	30	80	–	50
CON	70	790	50	50	580	100	90	290	–	190

At least now there are four examples that genuinely go beyond the Hereditary Harrop fragment (3, 4, 7 and 8). These examples show weakness in FT for the first time, while both UNI and CON do relatively well. In the MINLOG set, too, there is still the occasional disaster (e.g., number 9) when a Hereditary Harrop formula gets long and backward chaining search gets misled.

The test problems in [Tennant, 1992] are not uninteresting; they derive from certain mathematical investigations of minimal logic. But they are not the kinds of problems that we should expect the research of Chapters 3 and 4 to apply well to. They have a complex structure but with very few distinct propositions (three or four, counting falsehood) and very little ambiguity about worlds.

Random problems

Several recent research results report performance results for deduction on random formulas of (multimodal, propositional) *K* modal logic [Giunchiglia and Sebastiani, 1996; Giunchiglia *et al.*, 1997; Hustadt and Schmidt, 1997; Giunchiglia *et al.*, 1998]. This paradigm of evaluation follows on the tradition of using random formulas to test algorithms for propositional satisfiability begun in [Mitchell *et al.*, 1992].

I have not run random tests, for three reasons. First, random tests are computationally expensive. It makes little sense to run a small number—they are random—and the convention in the literature is to allow examples to run for 1000 seconds each. Second, random tests are also difficult to get right. It is important to work in a class of formulas which contains diverse and interesting problems—and few problems that can be solved trivially. At the same time, the random construction of formulas most easily follows the syntax of formulas. This is a tough balance. Even in the case of *K*, proposals for generating hard problems have been the subject of dramatic refinement from [Giunchiglia *et al.*, 1997] through [Hustadt and Schmidt, 1997] and [Giunchiglia *et al.*, 1998] in an attempt to find more uniformly difficult problems. It is an open problem to construct difficult random problems in S4, and seems likely to require a change in methods. The syntax is much less of a guide to how constrained an S4 deduction problem is, because S4 (unlike in *K*) allows necessary formulas to apply both at the present world and at distant worlds. With MPC, the restricted syntax of clauses compounds the problem; without negation clauses must take the form $C \supset D$ with C a conjunction of negative literals and D a NONEMPTY disjunction of positive literals.

Third, the literature on evaluation of modal decision procedures is equivocal. Or, more precisely, the literature emphasizes that large disparities in performance—in both directions—can result from seemingly small steps of preprocessing and simplification, and shows that researchers are typically surprised by (and in the case of alternatives to their favorite deduction method, quite unsympathetic to) these differences. Think of the improvements reported in section 5.1.2, which cannot exhaust the cascade. With this in mind, there seems little point on running the big evaluations until proof procedures are relatively refined.

Exponential series

We are left with the possibility of showcasing particular problems or classes of problems where the representations of UNI or CON are particularly valuable.

Here is one such class of problems. We write a collection Γ of k axioms of the form $A_i \wedge G_i \supset G_{f(i)} \vee T_{f(i)}$, parameterized by the function $f : k \to k$. We will interpret these axioms operationally in the following way. In the antecedent, A_i names ASSUMPTION number i, and it will link with an atom we'll assume later. G_i names GOAL number i, and will link either with an initial axiom G_1 which we assume, or with the consequent of another of these axioms. Finally, in the consequent, T_j names the test number j, which we will look for as the assumptions are introduced. Thus, to flesh out this interpretation, we also design a query formula Q of the form $A_1 \supset T_1 \vee (A_2 \supset T_2 \vee \ldots E \supset T_k \vee G_k \ldots)$. This successively adds an assumption, introduces the possibility of making a test, or alternatively continuing on, until finally the last test and goal are reached. Again, for k and f, the inference problem here is

$$G_1, \Gamma \xrightarrow{?} Q$$

What is search like on this class of problems? Ultimately, we must try to chase back the final G_k through to the assumed G_1 using the axioms from Γ. It becomes possible to make one step in the chase if we are currently looking at $G_{f(i)}$ and we can link to an earlier G_i. (The disjunct $T_{f(i)}$, which we must link with the corresponding disjunct in Q, will prevent us from linking to a later G_i, by modularity.) So this query is provable exactly when there is an increasing sequence $1, f(1), f(f(1)), \ldots, k$.

This is query is far from the Hereditary Harrop fragment. In fact, every axiom in Γ involves a matching a pair of transitions, while every step of the goal Q introduces another modal transition. So equational unification methods, and the implicit ordering search of FT, will encounter many possibilities on this method—most of them redundant. However, the only ambiguity the constraint method sees is in applying the (restart) rule.

The informal analysis of this query is confirmed by timing. I generated a few random problems of this form, with k increasing, and tried them out. (Drawing on our abstract characterization of these examples, it is not hard to balance provable and unprovable ones.)

The average results are given in (55).

(55)

k:	12	14	15	16	18
(#)	(8)	(4)	(8)	(16)	(8)
FT	710	1010	1530	11370	–
UNI	120	90	370	660	29500
CON	50	90	80	90	110

FT and UNI are clearly growing exponentially, as expected. CON, as expected, is not.

Modularity is also a key factor in allowing CON to fail on unprovable examples as quickly as it does. Disabling modularity yields uniformly unacceptable results.

The form of this query seems arbitrary at first, but it can be motivated from planning domains. Suppose we have k actions where any might fail, but where each will establish the precondition of another if it succeeds. The query supposes that we try these actions in a particular order, and demands that two things follow. Firstly, if a sufficient number of actions succeeds, then the overall goal of the plan should be accomplished; secondly, if any of those actions fails, we should know that it fails right away. To solve this kind of problem, formalized in this kind of way, the constraint algorithm of Chapter 4 is a clear win.

5.2 Logical Specifications and Reports of Success and Failure

While section 5.1 emphasized the use of constraint modular reasoning on independent hard problems, the language DIALUP for which I developed the techniques first and foremost offers a tool that applications can draw on to execute specifications of knowledge and modularity. For example, in NLG, as motivated in Chapter 1 and described in detail in Chapter 9, a generator can use DIALUP to construct and assess its options for extending a contribution to conversation. In such applications, two features of the reasoner are particularly important. The reasoner must be able to obtain for any query a result that accurately reflects the input specification, and it must be able to present its results in a simple form useful to the application. These design goals are furthered by DIALUP's modularity and constraints.

Consider how these two goals apply to NLG, for example. First, to guide decision-making, the reasoner must be prepared to respond automatically to whatever questions of grammar and interpretation are put to it. These questions will have a range of different answers in different contingencies; otherwise there is no decision to make. Providing a result in accord with the input specification is also crucial to NLG. NLG is an engineering enterprise, and assuring the quality of its results is important and difficult [Reiter *et al.*, 1995]. With logic programming, we get a theoretical starting point for such validation of NLG systems: a guarantee that output content reflects input content exactly.

Second, in NLG, logical queries often characterize the interpretation that the hearer is to derive from an utterance. Such queries demand answers that make only linguistically relevant distinctions. If the reasoner itself reports results more finely, it will have to be layered beneath an additional module that collapses and transforms answers to a form appropriate to NLG, perhaps at a significant computational penalty.

Many theorem provers result from a different perspective on the goals of reasoning. “We should be seeking cybernetic extension of our deductively extracted beliefs” [Tennant, 1992, p. 13]. By these criteria, the system counts as a success not if it can settle contingent questions for another application, but if it does a good job finding proofs of valid mathematical statements. Theorem-proving test sets are illustrative of this; there are compilations of difficult or interesting TRUE statements, but rarely of difficult or interesting FALSE statements. Reflecting the same perspective, many methods for modal deduction—particularly, resolution with equational unification—transform statements and results into the internal representation dictated by the theorem proving method rather than the application.

In contrast to a typical theorem prover, DIALUP’s logic programming strategy is particularly suited to these two requirements. This section reviews some reasons why. Section 5.2.1 describes why DIALUP can be made to deliver reliable answers to a range of questions when you view DIALUP as a programming language. Section 5.2.2 suggests that DIALUP’s constraint representations allow the work of delivering answers with natural structure to be done at programming or compile time.

5.2.1 Modularity: Exhausting the search space

An NLG system needs to construct messages that reflect the knowledge and goals of the hearer. One role of logic, which we return to in Chapter 9, is to facilitate these assessments of the hearer. For example, as part of evaluating a description intended to refer to a given object, an NLG system might need to assess what other objects the hearer thinks will match the description. For such queries, the best thing would be to exhaust the search space and find all proofs. Any proof we miss now is a possible source of misunderstanding later.

This kind of assessment depends on a representation of partial information, both to characterize gaps in the hearer’s knowledge (so the system knows what new to say) and gaps in the system’s knowledge of the hearer. Partial information is of course difficult to handle by restricted inference, particularly when partial information is described by disjunctions and existential quantifiers. Prolog is out; so are simple strategies like the global restart rule of Near-Horn Prolog. (Recall from section 3.3 that when the global restart rule of Near-Horn Prolog is admissible, then proof search never finishes as long as there are two independent relevant disjunctions. The interpreter can continue to construct new proofs indefinitely, by alternating between blocks.)

Without any way to describe constraints on search in partial information, the only choice is to use an open-ended prover and end search not by failure but simply by timing out. This is not a terrible strategy—especially in that it seems to automatically account for the fact that the inferential capacities of people, as presupposed in conversation, are limited [Walker, 1993]—but it is clearly a compromise.

DIALUP gives a careful programmer an opportunity to avoid such compromise. Structural modularity is an important technique for constraining and controlling proof search for knowledge bases with independent partial information. By keeping case analysis separate and modular, you can more easily use logic programming search to exhaust the search space by keeping case analysis separate. Once cases are separated, a simple cancellation mechanism provides all the bookkeeping needed to run through all the possibilities for search. It becomes possible to try to get exact answers in a simple way to questions such as NLG systems will ask.

5.2.2 Constraints: Reporting results concisely

Another aspect of hearer assessment required in NLG is uniqueness. NLG often requires objects or inferences to be identified or agreed on; this is a key step for a speaker and hearer to arrive at shared understanding. This means that NLG not only involves finding all proofs of certain forms, but also counting those proofs. The most attractive way to count is to use the number of results returned from proof search. If there is only one, you know you have satisfied things uniquely. Otherwise, there is ambiguity.

This strategy fails when there are multiple derivations with the same content. Then an NLG system may have achieved its goals of uniqueness even when many derivations are available. Under these circumstances, the system must apply a filter whenever a logical query is posted. Before analyzing the answers, the filter must match them pairwise and keep only a single representative of each type. This indirection and filtering can become a burden.

The equational approach to possible worlds is one case where indirection is required. Proof search will return many unifiers that differ, not in what rules were applied or what objects the rules were applied to, but only in how variables over possible worlds are interpreted. These separate solutions will not be interestingly different from the point of view of NLG. What's more, collapsing the results is particularly complicated, because to determine which objects are identical, it may be necessary to consult that expensive equational theory of paths.

On the other hand, the constraint representation of possible worlds alleviates the need for indirection. Any time the same set of inferences apply across a range of cases, constraints provide a useful tool for deriving meaningfully distinct proofs. This is especially true when it comes to possible worlds; the constraint algorithms of Chapter 4 eliminate all ambiguity in worlds.

5.3 Summary

When it comes to quantifying performance of modal decision procedures, particularly for the case MPC studied in section 5.1, there is no easy answer and plenty of room for further research. But from the test examples and families of problems considered in 5.1.3, we can tentatively identify a range of problems where modularity and constraints for deduction are important ideas for maximizing performance. These are problems that make meaningful use of disjunction. (In fact, we anticipate that this extends to problems that require partial information more generally, based on informal experiments with existential quantifiers.) As we shall see in the remaining two parts of the dissertation, disjunction and other specifications of partial information have an important role to play in reasoning about knowledge and action. In planning, partial information about the world underlies many useful behaviors where an agent performs a possibly redundant action, just to be safe (see section 6.5). Moreover, disjunctive characterizations of the information available to an agent allow an agent to use sensed information about the world to make conditional decisions in carrying out a plan (see section 7.1). Finally, in NLG, partial information about the hearer can be crucial to designing effective instructions that take the hearer's knowledge into account (see section 9.2). So the ability to reason more efficiently about modular disjunctive specifications directly impacts the tasks that motivate this research.

Ultimately, MPC deduction represents the simplest language for which modularity and constraints advance the state of the art. In first-order languages, exponential series which motivate constraint analyses are rampant. Problems (1.1–1.3), (1.4–1.6), (1.7–1.8) and (2.1–2.4) from [Sahlin *et al.*, 1992] illustrate four different series involving quantifiers; at the top of each scale FT requires tens of seconds to complete a proof. None of these examples cause any problem (in translation) in DIALUP; running time grows slowly and no problem takes even 100 milliseconds. With examples such as these, we can look forward to continuing to characterize the strong range of applicability of the techniques proposed in Chapters 3 and 4.

This extension to first-order problems supports the general observations about the value of exploiting modularity and constraints which we offered in section 5.2. We will return to these observations with concrete examples in Chapter 9. But first we need to look in more detail at how the specifications about action and knowledge that we need for NLG can be constructed. That is the subject of the next three chapters.

Part II

Modal Knowledge Representation

6
Action and Deliberation, Ontology and Search

Having completed our study of the technical side of modal deduction in Part I, we are brought to the question of how algorithms for modal deduction respond to the demands of concrete applications like the problem of reasoning in NLG introduced in Chapter 1 and examined in more detail in Chapter 9. This part of the dissertation provides a bridge between the modal formalism and its applications, by studying modal logic as a general tool for specifying and reasoning about problems of knowledge and action in AI.

This investigation of modal knowledge representation has three goals. One goal is to study the power of modal languages to describe planning problems according to purely expressive criteria. Planners are often based on special-purpose languages for describing actions (and occasionally knowledge, too). To provide an alternative, a modal approach must settle whether and how such descriptions can be recast in a modal language, and what inferences the modal language supports.

The second goal is to investigate the algorithms for modal proof search devised in Part I as they apply to planning problems. The results of Part I can be applied to derive a constraint-based, logic programming search strategy for reasoning about modal specifications of knowledge and action. An important and pervasive result of Part II is that such search procedures can be seen as natural generalizations of the search strategies designed and implemented in special-purpose planning algorithms.

The final goal is to lay the groundwork for reasoning in NLG by better characterizing the information for planning and acting that a hearer expects to find in an NL action description. We take the view that a plan is a complex object that describes not only a course of action to perform but also the purposes of the actions and the reasons why the actions will (or might not) achieve these purposes. In other words, plans should be characterized generally as guides to the action and deliberation of rational agents. Thus, particularly in instructions, the hearer must look for the information required to form, evaluate and adopt such a plan.

We begin this chapter by exploring the support for this view of plans and its implications for the design of a planner. These implications are taken into account later in the chapter as we write modal specifications of action and causality and deploy the techniques of modal deduction developed in Part I to reason about those specifications in planning. This chapter opens, in section 6.1, with an examination of the various uses to which plans can be put. We briefly review proposals that rely on the complex structure of plans to communicate and negotiate what to do [Ferguson, 1995; Chu-Carroll and Carberry, 1995], to ascribe plans to agents [Pollack, 1990], to adapt plans to new or changing circumstances [Kambhampati and Hendler, 1992; Hanks and Weld, 1995] and to control deliberation [Pollack, 1992].

In section 6.2, we consider how the view motivates a temporal ontology for planning that links the representation of time and the representation of inferences with the representation of action and causality.

In section 6.3, we reconcile this view with the practical task of constructing plans. We review three kinds of proposals for deriving a planner from a declarative theory of the effects of actions. The first is the use of the event calculus and negation-as-failure from [Shanahan, 1989; Missiaen *et al.*, 1995; Shanahan, 1997]; the second is Ferguson's use of explanation closure axioms and defeasible argumentation [Ferguson, 1995]; the third is the use of generalized logic programming to reason about action [Gelfond and Lifschitz, 1993; Baral and Gelfond, to appear; Baral, to appear; Lobo *et al.*, 1997]. These systems all bear a tantalizing similarity to the restricted inference of implemented planners like [McAllister and Rosenblitt, 1991; Penberthy and Weld, 1992]. In reasoning about the effects of actions, all recognize and accommodate the need to strike a balance between leaping to tentative conclusions and checking the assumptions that underlie tentative conclusions.

In section 6.4, we use the results on modal logic and modal deduction presented in Part I to describe a concrete formal system that synthesizes these conceptual, ontological and computational considerations. We provide a class of intended models that describe precisely the circumstances under which the reasoning formalism derives provably correct conclusions. By representing plans directly as modal proofs, this system gives plans the complex structure required for the varied uses of plans seen in section 6.1. Meanwhile, the desiderata of section 6.2 are met by deriving results using modal logic, which facilitates description of successive stages of deliberation and action, and then combining results using argumentation, which facilitates the recognition and report of conclusions based on the common-sense law of inertia.

Section 6.4 also establishes the kind of connection between valid reasoning and planning motivated in section 6.3. Using logic programming proof search, a constraint algorithm for reasoning about event sequences, and a system of defeasible reasoning for validating conclusions based on inertia, we find an exact parallel between the steps of modal proof search and the steps of special-purpose planning algorithms.

6.1 The structure and use of plans

A good plan is a reflection of its designer's causal knowledge. This causal knowledge ensures that the plan arranges appropriate circumstances for the execution of each action and guarantees (insofar as possible) that each action in the plan will then bring about the effect or effects intended for it. Take the simple blocks-world plan to move block A onto block B. This plan is useful only if it describes a coherent course of action, so the requirements of the domain must be taken into account in constructing it. For example, the plan must ensure that neither block has another on top when the move takes place. Similarly, since this plan will be adopted for what it achieves, the results of the plan in the domain must be accounted for. Here, these results will include the new, taller stack that the move will establish.

We adopt the view that each plan not only reflects the causal knowledge that goes into building it, but in fact explicitly records that knowledge. In other words, we model plans as data structures that encapsulate a sequence of actions that could occur, together with a justification of the possibility of those actions and a demonstration of their consequences. The motivation for this view lies in the natural account it gives of the different uses of plans. This section explains a variety of such uses for plans, both in guiding private deliberation in section 6.1.1 and in contributing to collaborative deliberation in section 6.1.2.

These different uses of plans arise for similar reasons; they share important features. A social agent in a dynamic world is always deliberating in response to these new snags and opportunities. It will need not only to build new plans, but also to change its existing ones—whether in response to changes in the world or the demands of coordination—so that the agent's plans can still be trusted to lead the agent towards the goals it planned them for. By explicitly representing causal relationships in its plans, an agent can better use and reuse them in this constant deliberation. Moreover, with an explicit causal representation, an agent can also assess the compatibility or incompatibility of independent new actions which the agent considers as new goals arise. Compatibility is a causal notion; it depends on the complex of conditions and constraints by which a plan is to achieve its intended effects. So testing compatibility also depends on giving plans a full, causal representation.

6.1.1 Plans in mental life

Pollack [Pollack, 1992], drawing on [Bratman, 1987; Bratman *et al.*, 1988], illustrates a first such role for plans in the deliberation of a bounded agent in a dynamic world: an agent can use the plans to which it is committed as a filter for reasoning. The agent seriously considers only those actions that are compatible with its plans.

As an example, Pollack describes arranging one summer to attend a distant conference while simultaneously undertaking a major move (a task whose difficulty and universality one increasingly appreciates). She began by fixing her flight dates to the conference. This facilitated further deliberation by limiting the alternative actions and new opportunities that needed to be considered. Thus, the flight plans were a guide to fleshing out the rest of the plans for the summer, for example constraining when the movers should come [Pollack, 1992, p. 51]. And the flight plans helped organize Pollack's response to unforeseen circumstances: when she was invited to give a class lecture during the middle of her planned absence, Pollack "quickly determined that giving the lecture was incompatible with my existing plans, and that it was not *prima facie* the kind of thing worthy of serious consideration nonetheless" [Pollack, 1992, p. 53].

We can see in Pollack's intuitive examples how compatibility must be measured against the entire causal complex that justifies a plan. Just assuming that the scheduled flying occurs, the movers still COULD come any time. The movers are separate agents who could be authorized to act freely and independently of Pollack. For that matter, just assuming the scheduled flying occurs, Pollack COULD still arrange to attend the class: her trip is long enough that she COULD just fly back for the class. The incompatibility is not just with the actions in the plan, but with its causal structure. To achieve planned effects, the flights come with preparatory intentions designed to ensure a successful conference—for example that preparations for the conference are to be taken, with as little disruption as possible. This IS incompatible with the movers coming too early. And the flights come with the effects for which they were undertaken—a continuous stay in Australia between them. This IS incompatible with an intervening attendance in class. If the movers must be supervised to ensure a successful move (another causal condition), this is also incompatible with the movers coming too late.

Where Pollack's argument shows how plans for one goal, richly represented, can guide deliberation for separate goals, Kambhampati and Hendler show how old plans can guide deliberation when attacking novel problems [Kambhampati and Hendler, 1992]. These new problems may be quite similar to those previously solved. Perhaps a task must be repeated; perhaps a task arises out of mistakes or exogenous events that slightly disrupt the execution of an old plan. Reusing an old plan requires assessing the compatibility of the plan with the new circumstances: again, the compatibility must respect the causal and intentional structure of the plan, not just the actions it contains. Accordingly, Kambhampati and Hendler's algorithms require plans to be represented with a VALIDATION STRUCTURE that annotates steps in the plan with concise records of what they do and why. In fact, subsequent research has shown how replanning can and should exploit exactly the same causal data structure used to construct a plan in the first place [Hanks and Weld, 1995]. It has shown, moreover, that the causal structure of plans is an important resource for the other reasoning tasks—which plan to adapt, and how to go about the adaptation—that must be solved to reuse plans effectively [Veloso, 1994; Koehler, 1994].

Kambhampati and Hendler illustrate the use of validations in refitting a blocks-world plan for a new task involving the positioning of a few extra blocks. As might be expected, refitting requires the addition of the new goals that ensure that the new blocks end up in the right place. Refitting also requires the preconditions for actions to be rechecked. To accomplish this, a validation in the plan must record when preconditions are satisfied in the initial state in one problem; that way a goal to satisfy the precondition can be inserted in a refit plan if the corresponding fact does not hold in the new problem.

It is important to observe that this rechecking is necessary even if the action could still be performed in the new situation. When executed in a changed state, the effects of the action will change. These different effects must also be checked for causal and intentional compatibility with the plan. Such examples highlight most compellingly Kambhampati and Hendler's need for a complete causal representation of plans. In their blocks-world refitting example, one such case involves the move of one block, K, onto another, J. In the new problem this still gets K onto J. However, in the earlier problem K was on the table, but now K is on block I. Because the move will now clear block I, the move has to be reconsidered. In fact, this consequence makes moving block K directly onto block J the wrong choice. Because block J eventually goes on block I, the new problem requires that block I be cleared by a different action: block K must be moved first to the table and then to block J.

6.1.2 Plans in public life

In deliberation, even a solitary agent must assess what actions are compatible with its own plans using a rich causal representation. When an agent can communicate or cooperate with other agents, it must also determine what plans are compatible with THEIR observed statements and actions. Because of the causal nature of compatibility, this PLAN RECOGNITION problem also involves not only anticipating the further sequence of the other agent's actions but also attributing a complex of beliefs and intentions to that agent [Pollack, 1990].

Pollack's dialogue in (56) illustrates both the range of attitudes that should be attributed to an agent in plan recognition and the value of doing so.

(56) a *A*: "I want to talk to Kathy, so I need to find out the phone number of St. Eligius."
b *S*: "St. Eligius closed last month. Kathy was at Boston General, but she's already been discharged. You can call her at home. Her number is 555-1238." [Pollack, 1990, p. 78]

To account for *A*'s utterance, we need to assume first that *A* intends to phone St. Eligius and that *A* intends to talk to Kathy. But we also need to assume that *A* believes that Kathy is at St. Eligius, that St. Eligius has a working phone, and hence that whoever answers the phone at St. Eligius when *A* calls will put *A* through to Kathy. These beliefs together lead *A* to the conclusion that phoning St. Eligius will lead to his talking to Kathy; the beliefs represent the causal basis for the plan.

We saw earlier why *A* would need this causal basis to guide deliberation and adapt this plan. Here we see that *S* must attribute this causal basis to reply to *A*'s question. Two components of *S*'s reply—St. Eligius closed; Kathy was at Boston General—correct the causal beliefs that *A* has used to form the plan. Without attributing causal beliefs to *A*, *S* would not recognize the need to present this information. The remainder provides not only an alternative action, calling 555-1238, but also the correct causal knowledge to account for why this alternative action will accomplish *A*'s intention of talking to Kathy. Only by recognizing that *A* must reestablish the causal basis for his plan can *S* determine that this information is necessary.

In (56), the plan recognition is intended. In formulating his contribution, *A* intends to describe the full causal basis underlying his plan. If *S* doesn't get the connection, *S* hasn't fully understood what *A* means. For example, *A*'s use of words like *so* and *need* shows *A*'s intention to make his deliberation public. Now, Pollack [Pollack, 1990] considers only plans to which an agent is committed. However, talk about hypothetical courses of action has a parallel structure and vocabulary; this indicates that such talk likewise intended to report causal connections. Consider (57)

(57) I might want to talk to Kathy, so I might need to find out the phone number of St. Eligius.

Accounts of public deliberation like [Ferguson and Allen, 1994; Ferguson, 1995; Chu-Carroll and Carberry, 1995] therefore model reports of POTENTIAL PLANS, even contributions to dialogue made hypothetically for the sake of argument, as complex objects with causal and inferential structure. As with (56), these accounts use the complex structure of contributions to discourse to better describe the relations between successive utterances and to better plan cooperative responses. An important virtue of these accounts is that they allow alternatives to weighed based on this causal structure, so that two conversational partners can arrive at a genuinely collaborative resolution of differences.

For example, Ferguson and Allen provide a formal model of the the planning discussion reported in (58). The conversation concerns a logistics problem in which freight must be transported first to the coast, by truck or rail, and then transported onward by ship.

(58) a Agent *A* suggests shipping the supplies by train;
b Agent *B* points out that the ship might leave at 4:00, and thus the supplies would miss today's ship;
c Agent *A* points out that the ship might not leave until 6:00, and thus the supplies would make today's ship. [Ferguson and Allen, 1994, (1–3)]

Each contribution is represented as a logical deduction about a hypothetical course of future events. Each deduction makes particular assumptions and exploits particular defaults; a formal system for comparing deductions on the basis of these assumptions allows the different proposals to be evaluated. In this example, as you might guess, neither agent comes out a clear winner.

[Chu-Carroll and Carberry, 1995] analyze a similar debate in a planning assistant designed to advise a student about what courses to take. Their analysis also treats a contribution to dialogue as a deduction with assumptions and defaults; they show how this allows the design of a more convincing response that challenges just the relevant assumptions on which system and user disagree.

6.2 The temporal and inferential ontology of planning

To build, extend, adapt, communicate or compare plans, we must take into account the causal connections that describe how the plan works. In describing [Ferguson and Allen, 1994; Ferguson, 1995; Chu-Carroll and Carberry, 1995], we found a representation of plans that gives these connections a central place. In this view, a plan is a formal demonstration, constructed according to some theory of events and their consequences, that a sequence of actions will achieve some goal.

In this chapter, we will use modal logic to articulate this view. We will appeal to modal operators $[\mathrm{N}]p$, meaning that p holds in the next state (after a single transition), and [H]p, meaning that p holds now and will continue to hold indefinitely into the future.[1] Using this modal language, we can write the causal theory as a set of statements T and use another set of statements I to describe the initial conditions of the planning problem (and perhaps to supply further available information about future conditions and events). More interestingly, with this language we can use a formula G—which perhaps refers to states in the future using [N]—to characterize the goal that the plan should achieve, and we can represent the actions in the plan by a set of statements P—again formulated using [N] to describe states in the future—which records our assumptions about what actions are undertaken as part of the plan and how those actions do (or don't) interact.

Thus, in our a deductive theory of plans, the demonstration that the plan achieves the goal takes the form of a deduction $\mathcal{D}$ with conclusion

(59) $T, I, P \longrightarrow G$

(59) mirrors the sequent calculus notation introduced in Chapter 2 and used to describe the state of a logic programming interpreter in Chapter 3. Here it indicates that in the deduction $\mathcal{D}$, the formulas in T, I and P are used to derive the formula G. It should be clear from this abstract presentation that the formulation of planning in (59) is substantially independent of the details of the modal language we use to describe plans, goals and causality.

The basic problem of building a plan is to find an appropriate set of actions in P, given the specification of T, I and G. As we shall see in more detail shortly, deduction from T together with the abductive assumption of premises in P can give an effective way to derive plans together with an appropriate rich causal structure.

6.2.1 Deduction, abduction and the uses of plans

Abduction is not the only formalism for describing planning in logic; in fact, abduction is contrasted unfavorably with a purely deductive view of planning in [Reiter, 1996]. In planning applications, however, I would view abduction and deduction as notational variants that differ primarily in how they organize the search space for inference. I substantiate that view here by examining the grounds for Reiter's objections to abduction. In Chapter 7, I substantiate it by presenting two formalisms, one based on deduction and the other based on abduction, which are obviously related this way.

In deductive planning, we again use the causal laws of the domain and the initial description of the world to prove that carrying out some action sequence p leads to the goal. However, we now represent those actions as a first-order term rather than a set of assumptions; the condition that the goal will obtain in a state where p has been carried out is formalized as a special statement $[p]G$. This description suffices to suggest the close relationship between abductive and deductive planning. The action term employed in deductive planning can be expanded into a set of facts describing how each component

[1]We have reasons for using an operator [N]p that does not identify which kind of transition is made; but the most important ones depend on connections between planning, modal deduction and reasoning about knowledge that we cannot yet present.

action takes place at its designated time. Conversely, if the form of abductive assumptions is restricted (as it must be) and the range of action terms is sufficiently expressive (as it too must be), then abductive action assumptions can be compacted into corresponding deductive action terms.

Despite this link, the deductive account may still seem simpler than the abductive one. As Reiter observes, abduction requires the planner to leave the deductive object-language to identify appropriate assumptions at the meta-level and to perform expensive—perhaps even impossible—consistency tests on the set of assumptions found. But these worries are misplaced.

The problem of consistency is by no means unique to abductive planning. A deductive plan that the goal will hold after the agent carries out an impossible sequence of actions is no more useful than an abductive plan that invokes inconsistent assumptions. Deduction thus involves the same checks and the same complexities as abduction. Indeed, Reiter's proposal in [Reiter, 1996] includes an explicit axiomatization describing which terms for action sequences refer to possible states, and builds in explicit and repeated checks to ensure that only such terms can figure in a deductive plan. But if we have this axiomatization of possibility, we ought to be able to use it just as well in abduction to determine which combinations of action assumptions are possible.

Likewise, the need to ascend to the meta-level to compute with plans is not a peculiarity of abduction but an inevitable concomitant of the broad use of plans reviewed in section 6.1.1. Even if a plan is built at the object-level by deduction, an agent must invoke meta-level reasoning to adapt that plan to changing goals or circumstances or to test the compatibility of that plan with others.

We can see this by considering—impressionistically—the use of demonstrations of the consequences of causal theories after these plans are first constructed. For example, in the dynamic deliberation described in [Pollack, 1992], we start with one demonstration $\mathcal{D}$ of

$$T, I, P \longrightarrow G$$

In light of some new opportunity, we recognize that it might be worthwhile to consider an elaboration of our goals—although we are committed to our present actions and their purposes. As part of this, we must construct some other demonstration $\mathcal{D}'$:

$$T, I, P' \longrightarrow G \wedge G'$$

Then we make a comparison: if $\mathcal{D}'$ is preferable to $\mathcal{D}$, we will abandon $\mathcal{D}$ and adopt $\mathcal{D}'$. Because of the similarity between these two problems, $\mathcal{D}$ can guide the construction of $\mathcal{D}'$ both by allowing the premises P of $\mathcal{D}$ to be preserved in the augmented premises P' of $\mathcal{D}'$ and, more importantly, by allowing the inferences in $\mathcal{D}$ to be preserved in $\mathcal{D}'$. Reusing inferences in this way is an inherently meta-level task. So is the application of Pollack's filters of compatibility, which, in this view of planning, rule out hypothetical actions which must disrupt these old inferences.

In adapting and reusing plans, meanwhile, we exploit the initial plan $\mathcal{D}$—which assumes one set of goals and initial conditions—to construct another demonstration $\mathcal{D}'$ whose goals and initial conditions are related but distinct:

$$T, I', P' \longrightarrow G'$$

Again, the similarity between I' and I and between G and G' suggests not only that many of the actions in P can be preserved in P' but that much of the structure and inferences of $\mathcal{D}$ can remain in $\mathcal{D}'$. In regarding a plan as deduction, we would reconstruct replanning procedures as transducing the structure of $\mathcal{D}$ into a provisional structure for $\mathcal{D}'$ in which inferences inapplicable to I', P' or G' are eliminated and replaced (when necessary) by new goals for proof search.

Consider now plan recognition, as needed to interpret discourses like (56). In (56a), it is clear that *A* describes some intended actions *P* and goals *G*. However, in light of the overall complex of interrelated beliefs that *A* means to convey in (56), it is more economical to assume that *A* in fact intends to communicate an entire deduction, which concludes:

$$T', I', P \longrightarrow G$$

In other cases where plan recognition is needed in interpretation, the speaker may intend to identify the plan only partially, or to leave the particular plan chosen to the discretion of the hearer; such possibilities are explored in [Young, 1997].

A's identification of a plan in (56a) is possible because *A* provides enough information about the causal reasoning underlying it for *S* to fill out the rest. Taking up a point introduced in Chapter 1 and revisited in Chapter 9, if an action description omits the causal laws and other facts about the world needed to to justify the choice it proposes, it is frequently because their role in the underlying plan is obvious in context. (This is particularly true in instructions that the addressee is meant to carry out.) Indeed, when conversationalists share knowledge and expertise, we will often expect the theory T' and the initial conditions I' to be given by the context of communication and planning.

As Pollack's example shows, in the presence of misconceptions the speaker's domain knowledge and assessment of the present situation may differ from that of the audience. This naturally requires more flexible and more open-ended reasoning from the audience. Conversely, when generating descriptions of plans, as in [Chu-Carroll and Carberry, 1995] for example, a speaker must be sure to give enough information that the intended deduction is easily recovered. This means taking into account the knowledge shared with the audience and the opportunities the audience has to deploy that knowledge. Assessment of shared knowledge offers another natural opportunity to exploit modal knowledge representation and modal inference; we look carefully at the one-sentence case in Chapter 9.

6.2.2 Planning and argumentation

This representation of plans as formal demonstrations obviously requires further specification and refinement. What structure do these demonstrations have? How do they represent the sequences of deliberation and action that an agent undertakes? In addressing these questions, we can find a promising starting point from previous informal work drawing out our intuitions for how people reason about the future. In the remainder of this section, we review first Konolige's motivations and suggestions for formalizing temporal reasoning in a system of defeasible argumentation [Konolige, 1988] and then Steedman's motivations and suggestions for using a dynamic logic based on the situation calculus for connecting temporal talk and temporal reasoning [Steedman, 1995; Steedman, 1997]. Then in section 6.3 we turn to the task of operationalizing this view of planning as reasoning.

It is natural to view the temporal reasoning required in plan construction, plan modification and plan recognition as DEFEASIBLE REASONING—reasoning that allows plausible conclusions to be inferred initially even though these conclusions may have to be retracted as more information arrives or more reasoning is performed. After all, as a plan is sketched out action by action, adding new actions may require some adjustments to compensate, and the successful future execution of that plan is never an absolute certainty. In particular, the fact that states affairs tend to persist unless changed by an event, seems an important but defeasible principle of temporal reasoning.

In [Konolige, 1988], Konolige suggests performing this defeasible reasoning in a DIRECT system of ARGUMENTATION. In such a system, conflicting conclusions are resolved by explicitly comparing the chain of reasoning or ARGUMENT that led to each. The argument (and conclusion) adopted is the one that fares better on the basis of general principles of reasoning or on the basis of specialized knowledge about reasoning in the domain. Konolige suggests that such argumentation is both natural and flexible.

An argument provides a natural record of temporal reasoning and planning because it concisely encodes the evidence for the success of a plan. Konolige observes that this makes an intuitive output for debugging a knowledge base. [Ferguson, 1995; Chu-Carroll and Carberry, 1995] substantiate this result by describing contributions to dialogue in terms of argumentation. More immediately, a concise record of the evidence for a course of action is precisely what is needed if this reasoning is to continue to guide further deliberation.

Argumentation is a flexible framework because the comparison of arguments so easily accommodates reasoning principles for a particular domain. Comparable specifications in indirect formalisms for default reasoning, such as circumscription [McCarthy, 1986], default logic [Reiter, 1980] and autoepistemic logic [Moore, 1985b], can be very difficult to devise. The system of section 6.4 illustrates this in its successful use of direct principles of attack for inertia and occlusion. Konolige illustrates this with a more picturesque example—which, since our later mathematics excludes it, perhaps can convey compellingly only how perilous it can be to shift assumptions about the world while motivating formalisms for common-sense reasoning. Konolige considers whether arguments for change based on causal laws are always preferable to arguments for persistence based on inertia. Generally shooting kills its victim, but what if we know of a shooting and know the victim is alive some time later? We have the argument from change that the victim should have been dead immediately after the shooting, and the argument from persistence that the victim should have been alive earlier if it was alive later. These arguments conflict. Given our knowledge of what it means to be alive and dead, we prefer the explanation that something went wrong with the shooting than the explanation that some unknown event reanimated the victim. As Konolige sees it, then, this is a special case where the domain specifies that an argument from persistence can defeat an argument from causation.

In addition to this ease of specification, another advantage of direct systems of argumentation over indirect formalisms for default reasoning is that with argumentation, causal default reasoning can be implemented using ordinary deduction as a module. For example, we can readily design a system of argumentation around the results for modal deduction described in Part I.

Following [Konolige, 1988] (and [Ferguson, 1995; Chu-Carroll and Carberry, 1995]), then, we would prefer to regard the deductions $\mathcal{D}$ involved in planning as acceptable arguments that a goal can be achieved by a specified course of action.

6.2.3 Planning, action and temporal reference

To facilitate continued deliberation, the representation in which planning is carried out should not only represent reasons for actions in a salutary way, it must also represent the actions themselves well. That is, the representation of action and time should match the choices an agent has and the framework in which it makes those choices. This section presents reasons to perform this reasoning in a modal framework, broadly inspired both by the situation calculus of [McCarthy and Hayes, 1969] and the functional translation of modal logic of [Ohlbach, 1991] presented in section 2.3.3.

It is natural to think of most, if not all, deliberation as occurring in discrete stages. In each stage, the agent draws on a stable body of information about the world to evaluate and choose among alternative courses of action as best it can. Stages are punctuated by intervals during which the information available to the agent is unreliable or in flux. During these transitions, the agent will not attempt deliberation, or will be forced to abandon effort toward deliberation in the face of unanticipated changes.

During periods of deliberation, an agent would seem to gain little from monitoring the impact of elapsed time on the information by which it evaluates its choices for action. It can view this information simply as true now. (Here we distinguish the content of the evaluation from the means by which the agent arrives at that content; clearly the agent's schedule of evaluation and choice can benefit from taking into account how much time is likely to be available.) Finer temporal indexing of facts would be required only if an agent's deliberation kept track of judgments based on pieces of information too divergent to apply to a single time. In such a case, the new information would likely trigger a new round of deliberation anyway. Such monitoring must be dispensed with to some degree regardless. An agent that had to explicitly apply the law of inertia to extend its prior thought into the present moment would probably succumb to the bookkeeping.

For these reasons, a temporal ontology which maps a set of continuously passing numerical instants each onto a distinct state might needlessly complicate an agent's reasoning for many tasks (particularly those that are characteristically deliberative). A view that breaks time down qualitatively into a series of states and transitions seems more appropriate. States encapsulate the circumstances in which the agent deliberates. They encode the invariants that an agent can exploit to select an appropriate action during those circumstances. States are linked by transitions; each transition may abstract into a single jump both changes initiated by actions selected by an agent and changes initiated by the exogenous occurrences of the agent's dynamic environment.

One implementation of this qualitative ontology is the situation calculus [McCarthy and Hayes, 1969]. The situation calculus provides a set of types which allow statements of sorted first-order logic to describe qualitative change in the world over time. Temporal states of the world are encoded in a type of situations. Properties that vary across situations are known as fluents. Actions are transitions from situations to situations. Typically, we reify actions and describe the situation reached after transitioning by action a from situation s as $result(a,s)$, using an all-purpose *result* function. We can relax this to allow nondeterministic actions—or to see terms such as a as describing types of transition which a particular transitions might instantiate—by appealing to a relation $result(a,s,s')$.

If states and relations among them provide a basic temporal ontology for planning, however, then there is a very tight fit between planning and modal languages. In fact, under the functional translation [Ohlbach, 1991], modal logic can be seen as a natural generalization of the situation calculus. The situations correspond to the worlds, and actions correspond to types or instances of the accessibility functions that relate worlds in translation-based deduction. On this scheme, if s denotes a situation then for any f in F, the term $s;f$—the result of sequencing f next in s—provides a modal path term that denotes another situation. [Steedman, 1995] explores this idea from the perspective of dynamic logic.

We will adopt and adapt this ontology for the rest of this dissertation. In using this kind of ontology, we must keep in mind the intended functions and inevitable limits of this representation. McCarthy and Hayes present the situation calculus as a general ontology for making predictions about the future; their examples range from predicting the motion of a continuously falling block to reasoning about the effects of the atomic transition of dialing the phone. The situation calculus, with its discrete representation of transitions and awkward connection to linear time, is clearly much more suited to the latter kind of reasoning than the former. There is no reason to use the situation calculus for continuous problems where little is to be gained from its strengths—representing and reasoning about points where an agent can and must make discrete deliberative choices among alternative actions. There are plenty of better formalisms for such problems, based on metric intervals, notably including [Allen, 1983; Allen and Hayes, 1989; Allen and Ferguson, 1994].

Frameworks based on the situation calculus do have clear strengths, however. An agent's choices are basically discrete. The qualitative temporal model of the situation calculus makes it quite easy to describe these successive choices in more detail, including limits on an agent's knowledge and inference. (We describe such models in Chapters 7 and 8.) The use of continuous time makes it substantially more difficult to describe these discrete stages of deliberation and choice. (In fact, I don't know how to do so.)

These strengths give us ample reason to try to accommodate a limited range of continuous reasoning within the situation calculus. [Steedman, 1995; Steedman, 1997] proposes one way to do this, using a taxonomy inspired by natural language aspectual categories. Using this taxonomy, an agent's participation in continuous activities can be broken down into stages. For example, in a given stage, a continuous event might be in progress (in a state corresponding to English progressive aspect), or it might be completed, with its effects remaining in force (in a state corresponding to the English perfect). Links between these stages are accomplished by special transitions whose interpretation matches English aspectual verbs like *start*, *stop*, *finish* and *continue*. With this added aspectual structure, transitions that might formerly have been represented as instantaneous are broken down in a qualitative way that better reflects their continuous nature. By keeping to the situation calculus, however, the aspectual approach continues to incorporate the limits in deliberation and choice inherent to the agents that execute these transitions. In particular, each stage of execution of a continuous activity offers the agent a pause for deliberation and choice. The theory offers invariants characterizing what holds at this stage and, as always, allows the agent to use these invariants to deliberate and select the appropriate next action (possibly an aspectual one).

6.3 Planning and proof search

The most immediate worry with this temporal and inferential ontology for planning outlined in section 6.2 may be whether such an abstract characterization can support efficient computation. By considering several efforts to connect deduction with planning algorithms, we can arrive at substantial optimism on this question. At the same time, we can extract some important principles for formalizing specifications of actions in such a way as to accommodate a better fit with the action of specialized planning algorithms. As we shall find in the next chapters, a still more expressive language is required to guide an agent's deliberation in the presence of information-gathering and hierarchical action. The solution to simple planning problems using deductions that we describe in this section will help us find similar deductive solutions to richer problems there.

6.3.1 Specifying a planner

To analyze the gap between logical and algorithmic planning, we must begin by considering the different ways the process of planning can be formalized. Classic descriptions of planning algorithms like SNLP [McAllister and Rosenblitt, 1991] or UCPOP [Penberthy and Weld, 1992] first define the data structures that the planner maintains, and then characterize a series of updates that can be applied nondeterministically to these data structures until a fixed point is reached. The soundness and completeness of the planner is established by proving that these fixed points describe all and only the sequences of actions that solve the input planning problem.

A logical approach puts more emphasis on its data structures—they take the form of a collection of syntactic objects, derivations, that trace how the consequences of a specification of a domain are to be calculated. The key step in showing correctness for a logical approach is to demonstrate that derivations yield exactly the conclusions true in all the models that describe how state can evolve as events occur, according to the intended meaning of the domain specification. Proofs along these lines for theories of action begin with [Lifschitz, 1987] and are now common; they can be found for example in [Lin and Shoham, 1991; Gelfond and Lifschitz, 1993; Reiter, 1991; Sandewall, 1994b]. In formalizing soundness and completeness of inference, these methods draw on and extend corresponding methods from logic, for example correspondences between modal proofs and Kripke models.

In a logical approach, the planning algorithm is then just a search procedure for constructing derivations. This procedure may be streamlined by applying optimizations analogous to those available for other proof methods. For example, we can appeal to transformations that allow derivations to built incrementally, by using a Herbrand theorem along the lines outlined in section 2.3.5. We can try to reuse repeated steps and keep proofs compact by using a cut rule or resolution, as in section 2.3.3 or 5.1.2. We can reduce nondeterminism by applying rules in fixed but general orders—such as the logic programming order studied in Chapter 3. Nevertheless, once such optimizations are justified at the level of derivations, the correctness of the search procedure is immediate because it searches by applying exactly the formal rules by which derivations are defined.

When it comes to their idealizations of the world, these two approaches are already close. Both interpret causality very strongly, by ruling out exceptions to effect axioms for actions and ruling out exceptions to inertia apart from effects of specified actions. This view of causality is built into the correctness definition of problem solutions for planning algorithms and the characterization of intended models for logical approaches. It is an obvious approximation; it fails in many common-sense world histories (like Konolige's example of shooting survival). Such approximations, although unsatisfying, seem necessary at this stage of research in order to obtain comprehensible and precise results. Sandewall anticipates that these approximations will gradually be relaxed as the field matures [Sandewall, 1994a].

These characterizations of the two different approaches lead to a concise statement of the challenge of reconciling the two frameworks: How can we see the ad-hoc data structures maintained in planning as partial representations of logical derivations? And how can we regard the incremental updates performed by planners as steps in constructing derivations?

One part of the answer can be presented immediately: planners' use of BINDINGS and CAUSAL LINKS for means-end reasoning. Planners use these data structures to encode how the effects of some actions in the plan establish the preconditions of others; a list of goals maintains the open preconditions that have not yet been accounted for in the plan. At each step, the planner may discharge a goal by adding to bindings and causal links. It is easy to imagine how this process corresponds to the construction of a logical derivation. If the successful occurrence of an action logically entails the further persistence of its effects, then these data structures and updates are just what is needed to construct a proof that each action in the plan achieves its intended effects. Thus, Penberthy and Weld write, "UCPOP is a theorem prover that resolves step preconditions against effect postconditions" [Penberthy and Weld, 1992, sec 4].

But the rest of the answer is problematic. In addition to these steps that are rather straightforwardly seen as proof search, planners perform additional steps of THREAT RESOLUTION. Whenever any change is made to the plan, a check is made as to whether some action might interfere with a needed causal link, by disrupting the desired effect before the action that depends on it takes place. Obviously, in a proof of classical logic—or modal logic, for that matter—there is no need to protect inferences against threats.

Research towards a logical reconstruction of both causal linking and threat resolution (including this one) generally involves two steps. One step is to reconstruct threat resolution in terms of one of two similar simple approaches to default reasoning: negation-as-failure and argumentation. The second step is to link these defeasible notions with established deductive techniques to derive a correct, computational account of reasoning about action. We now review several ways such reconstructions have been sketched.

6.3.2 Planning in the event calculus

The event calculus, developed in [Kowalski and Sergot, 1986; Kowalski, 1992] is a formalism for reasoning about action based on logic programming and negation-as-failure. Shanahan summarizes the basic system in two logic programming clauses.

(60) a holds-at(P,T) if
  happens(E) and $E < T$ and
  initiates(E,P) and not clipped(E,P,T).
b clipped(E,P,T) if
  happens(E') and terminates(E',P) and
  not $T \leq E'$ and not $E' < E$. [Shanahan, 1989, 1.1 and 1.2]

Clause (60a) describes when a consequence *P* persists to time *T*. There must be some earlier event *E* which we know of, and which regularly causes *P* according to the definition of the *initiates* relation in our domain theory. Moreover, we cannot know of any reason why the persistence of *P* from *E* to *T* should be disrupted or *clipped*. The appearance of negation-as-failure in this clause endows persistence with a default character.

Clause (60b) defines the disruption of persistence of *P*, by any other event E' known to occur and to disrupt *P* according to the definition of *terminates* in our domain theory, provided its occurrence is not known either to precede or to follow the interval over which *P* should persist. (These negations make disruption eager and thus make the conclusions of the calculus correctly conservative in the presence of incomplete temporal information.)

In 6.2, we introduced a characterization of planning as inference: to build a plan is to find a collection of actions whose occurrence entails the eventual truth of the goal. Taking this approach using the event calculus—as first suggested in [Eshghi, 1988] and followed up in [Shanahan, 1989; Missiaen *et al.*, 1995; Shanahan, 1997]—means assuming facts of the form $happens(E_x)$ and $E_y < E_z$ constraining the occurrence and orderings of events, so as to prove $holds\text{-}at(G,t)$. The result is a natural procedure with close affinities to the action of hand-coded planning algorithms.

Like planning algorithms, event-calculus planning involves two kinds of steps. One kind of step matches goals of $holds\text{-}at(P,t)$ with premises of the form $initiates(E,P)$ in applying clause (60a); at the same time it assumes steps and orderings between them. This of course corresponds to the resolution of preconditions with effects found in UCPOP.

In keeping with clause (60a) these connections involve a burden of showing by negation-as-failure that the persistence of effects is not disrupted. When using the event calculus to deduce the consequences of a set of known events, any of them may potentially trigger such disruption; thus in planning with the event calculus, the *not clipped* goals in the derivation of a plan must be repeatedly rechecked as more events and orderings are assumed into the plan. These checks are the second kind of step required by event-calculus planners; they correspond closely to the detection and resolution of threats performed by partial-order planning algorithms.

In particular, when a consequence depends on some *clipped* fact failing, a threat corresponds to a way of constraining the plan so that the *clipped* fact instead is provable. So constrained, the plan would no longer correspond to an event-calculus derivation. Event-calculus planners, like algorithmic planners, make assumptions to resolve threats. By assuming constraints $T \leq E'$ or $E' <$ E, the *clipped* proof can be defused at the level of clause (60b); planners also impose these constraints in the promotion and demotion steps of threat resolution. More generally, Shanahan's [Shanahan, 1997] event calculus planner can also defuse a *clipped* proof by defusing the conditions that justify the *terminates* relation in clause (60b). This corresponds to the separation strategy of UCPOP for resolving a threat caused by a conditional effect of an action.

While the event calculus offers a framework for drawing out the more general logical underpinnings of planning, it is quite a constrained formalism, particularly when it comes to reasoning with incomplete specifications of the state of the world. Such reasoning requires the possibility of case analysis to consider the alternative possibilities in which an unknown fact holds and in which it does not, and classical negation to describe the content of these alternatives. It is unclear how to add classical negation perspicuously to the event calculus—it appears rather mysteriously at certain points in [Shanahan, 1997]. It is unclear how to incorporate reasoning by cases at all. The formalisms we turn to next give examples of how such expressive power can be achieved in approaches to planning based on defeasible reasoning, but they do not achieve the simplicity or faithfulness to planning algorithms found in event calculus planning.

6.3.3 Planning with explanation closure and argumentation

Explanation closure [Schubert, 1990; Reiter, 1991; Allen and Ferguson, 1994] can be thought of as a purely deductive version of the event calculus. We follow [Allen and Ferguson, 1994] here. As in the event calculus, they assume an ontology of linear time and events whose occurrence drives change. Although events drive change in explanation closure, as in the event calculus, rules describing persistence are formulated in terms of the property that persists or changes rather than the event that changes it. Explanation closure associates with each property a statement of classical logic that specifies all the events that can change that property. The persistence of the effect of an action can then be established by proving or assuming that no events occur that would interfere. However, because explanation closure is based on classical logic, that is the only way to conclude persistence.

For example, in the blocks world, suppose a block *a* is clear over interval ι and not clear over an abutting interval ι'. Explanation closure says that some event in which another block was put on *a* must have finished as ι ended—formally:

(61) $$clear(a,\iota) \wedge \neg clear(a,\iota') \wedge \iota : \iota' \supset \exists ex.puton(e,x,a) \wedge time(e) : \iota'$$

Such axioms provide a way to reason about persistence as follows. Suppose we know that *a* starts out clear at some time, and that nothing is put on *a* before some later time. If *a* is not clear at the later time, then there must be earlier adjacent intervals where *a* goes from clear to not clear. Then the antecedent of (61) is satisfied but the consequent is not, which is absurd. Thus *a* must still be clear at the later time.

With explanation closure, then, to ensure that *a* remains clear over an interval, a planner can and must assume explicitly that nothing is put on *a* during that interval. More generally, for any explanation closure rule for property *P*, a planner can ensure that *P* does not change over the interval between ι_1 and ι_2 by adopting the assumption that the consequent of the explanation closure rule is false during the interval. Ferguson writes such assumptions in the form $NC(P,\iota,\iota')$, for nochange [Ferguson, 1995].

In planning with explanation closure, then, we view steps in the plan as assumptions about the occurrences of events, ordering constraints as assumptions about when events occur, and causal links as nochange assumptions. The action of a planner in backward chaining from goals and preconditions to effects is to construct a classical deduction that the goal will eventually hold by combining these strong assumptions with the causal axioms and explanation closure axioms that describe evolution of state in the domain.

Thus, while the construction of an explicit explanation closure deduction may follow a planning algorithm in outline, explanation closure requires detailed reasoning for tasks that planning algorithms implement with a simple atomic step. The added complexity of explanation closure is to be expected, because explanation closure axioms are designed with a richness that allows them to describe many domains and inferences that lie outside the scope of usual planning algorithms. For example, [Allen and Ferguson, 1994; Ferguson, 1995] provide explanation closure formalizations of domains where actions can have nondeterministic effects, and where actions can occur simultaneously and externally to the action of the planning agent.

The explanation closure characterization of planning may seem to leave no room for the problematic steps of threat detection and resolution. Explanation closure views plans as classical proofs, and the conclusion of a classical proof stands once and for all. Nevertheless, a problem of threat detection and resolution continues to arise under explanation closure. The reason is that not all combinations of assumptions and proofs constitute sensible plans. In some cases, an agent would have no consistent way to act in accordance with the assumptions. In explanation closure planning, threats are potential demonstrations of this inconsistency. Ferguson gives two ways to characterize this problem more precisely and to solve it using a non-monotonic formalism that uses ARGUMENTATION to rule out the consequences of inappropriate proofs.

In the first characterization [Ferguson, 1995, ch. 3.3], a threat to a plan represents a set of constraints on the ordering of events under which it becomes inconsistent to assume the successful execution of the plan. In the blocks world, for example, suppose you have assumed that block *a* is to remain clear in some interval, and that you have also assumed that at some point you put something on *a*. Then by assuming that event to occur during the protected interval, you wind up with an inconsistent plan. Argumentation can be used to rule out the consequences of proofs that can be refined inconsistently this way. Then to maintain a good argument, such threats must be addressed by adopting further assumptions so that the actions in the plan can be executed successfully as long as their occurrence respects the assumed order.

This proposal uses the explanation closure semantics of causal links to explicitly motivate the behavior of a planner in threat resolution. As with the account of the means-end reasoning of planners in explanation closure, this account interprets the relatively simple behavior of the planner in terms of potentially much more open-ended mechanisms of detecting and resolving inconsistencies.

In the second characterization [Ferguson, 1995, ch. 3.4], nochange assumptions are not explicitly interpreted by the planner. They are assumed simply as atomic formulas. Threats continue to arise, because putative plans making use of nochange assumptions can be threatened by counterarguments. Counterarguments express the possibility that something in the plan might go wrong if the events transpire in the wrong order; formally, no plan can be accepted while it has an undefeated counterargument. Counterarguments are defeated by more specific rules that show that a potential disaster cannot arise because the steps in the plan have been reordered to avoid it. Thus, at each stage the planner must check whether new assumptions permit the construction of new counterarguments to its plan. To defeat each counterargument and thereby resolve its threat, the planner is led to impose additional constraints on the plan.

This reconstruction seems like a more faithful reconstruction of the action of planners in the style of SNLP. However, it loses much of the logical transparency and immediacy of the more straightforward explanation closure account. Effectively, the data structures of the planner are encoded as premises in the logic and threat resolution is encoded in rules for arguing about these premises. So, like Ferguson's first characterization, this one is suggestive but not compelling as a logical reconstruction of implemented planning.

6.3.4 Planning with logic programming theories of action

A third strategy for reconciling logic and planning uses generalizations of logic programming as a way of striking a balance between the perspicuity of the event calculus and the expressive power of explanation closure. The idea can be seen from the clauses for inertia proposed in [Lobo *et al.*, 1997] and given in (62). When combined with axioms for the effects of actions and given an appropriate semantics as a generalized logic program, these clauses contribute to a provably correct model of inference for strict inertia.

```
(62)  a   holds(F,res(A,S)) if
                holds(F,S),
                not ab(op(F),A,S).
      b   ¬ holds(F,res(A,S)) if
                ¬ holds(F,S),
                not ab(F,A,S).
      c   holds(F,s0) or ¬ holds(F,s0).
```

Similar logic programs can also be found in [Gelfond and Lifschitz, 1993; Baral and Gelfond, to appear; Baral, to appear]. The ontology used is that of the situation calculus: from an initial situation s_0, taking action A results in a new situation $res(A, s_0)$, and so on (this use of *res* corresponds to McCarthy and Hayes's *result*). Apart from this ontological difference, clause (62a) encodes essentially the same generalization as clause (60a). The reasoner should as a default assume that positive facts persist into the future by inertia, except when it knows of something unusual—an abnormality—in the persistence of a fluent across an action.

Clause (62b) licenses complementary inferences about negative facts; the clause appeals both to negation-as-failure to make default conclusions and to classical negation $\neg$ to characterize negative facts. Finally, clause (62c) licenses case analysis for resolving partial information about the initial state of the world. These clauses address the two gaps that we observed informally in the event calculus. So it is not surprising that by computing with them, an agent will draw sound and complete consequences from its partial information.

The only difficulty lies in characterizing what it means to compute with them. Each clause in a generalized logic program takes the form shown in (63).

(63) $a_1 \textit{ or } \ldots \textit{ or } a_k \leftarrow p_1, \ldots, p_n, \textit{ not } e_1, \ldots, \textit{ not } e_m$

Each variable in (63) represents a literal—an atomic proposition or its negation. The force of such a clause is described in terms of a nondeterministic interpreter that assumes a consistent set of literals that represent a possible set of consequences for the program. If (63) is in force, then whenever the interpreter has constructed an output containing all of

$p_1, \ldots, p_n$, the interpreter either must have assumed one of the $e_1, \ldots, e_m$ for an independent reason, or must also assume one of the $a_1, \ldots, a_k$. Given a program containing many such clauses, the compatible sets of assumed literals are codified in the definition of an ANSWER SET [Gelfond and Lifschitz, 1991]. Informally, in computing an answer set, the interpreter makes exactly the assumptions it needs and no more: an answer set is a set of literals S, such that S itself constitutes the minimal set of assumptions needed to comply with the directives of each clause in the program after S is assumed. The consequences of a program are those literals present in every answer set.

This is an obscure definition with no obvious computational content. To clarify its semantics, papers like [Gelfond and Lifschitz, 1993; Baral and Gelfond, to appear; Lobo *et al.*, 1997] give an alternative characterization of the content of answer sets in terms of a set of intended models of a specification of action. These intended models rely on a domain specification in terms of a simple high-level language $\mathcal{A}$ (or one of its extensions); it is straightforward to describe the evolutions of state compatible with such a domain specification directly in terms of mathematical constraints on the situation calculus *res* function. The next step is to translate the high-level language into a logic program incorporating clauses like those in (62), and consider its answer sets. The correctness result of [Baral and Gelfond, to appear; Lobo *et al.*, 1997] is that the literals present in every answer set for the translated specification are exactly the literals present in every intended model of the original specification. This strategy validates the logic programming approach but further challenges its computational relevance; the classical specification of the intended models bears a close resemblance to result of applying predicate completion to the high-level language, or introducing explanation closure axioms. Given the resemblance, the natural approach would just be to reason with the classical characterization—such an approach to reasoning about transitions is in fact advocated in [Reiter, 1991; McCain and Turner, 1997].

There is indeed no interesting computational theory for the consequences of a generalized logic program. However, the answer set semantics for clauses like (63) does generalize the STABLE MODEL SEMANTICS of ordinary logic programs with negation-as-failure [van Gelder, 1986; Gelfond and Lifschitz, 1988; van Gelder, 1989]. There is therefore some hope that interpreters for generalized logic programming might be constructed on the basis of ordinary logic programming interpreters. Moreover, in [Baral and Gelfond, to appear], it is shown that the more expressive clauses from generalized logic programs like (62) can be omitted while maintaining a sound system for reasoning about action. The restricted programs remain complete for many reasoning tasks, in fact. Consequences of the restricted programs can be calculated using ordinary logic programming interpreters; for planning tasks, as [Baral, to appear] observes, the action of an ordinary logic programming interpreter on a restricted specification will bear a close resemblance to the behavior of a special-purpose planning algorithm. Since the restricted programs look a lot like the event calculus, this is not surprising given the results discussed in 6.3.2. In fact, the event calculus goes a step further by building in a partial-order theory of time that planners use but that is not immediately available in the situation calculus ontology used in (63).

6.3.5 Summary

The different threads summarized here converge on the following observation about planning. Planning requires leaping to conclusions about the persistence of effects. So, in lock step with the inferences needed to link effects with goals, a planner needs a threat resolution mechanism to check its assumptions and repair them if necessary. This is a common feature of the different accounts we have seen, although they accomplish it by the different mechanisms of negation-as-failure and the defeasible assumptions of explanation closure.

In justifying planners logically, we find the need and opportunity for other kinds of reasoning than planners typically do. We can see this in the classical reasoning introduced in explanation closure and the classical negation and disjunction required for correctness in logic programming theories of action. From the work we have seen, it is clearly difficult to reconcile this additional reasoning with the constrained reasoning that planners do in special cases.

Difficult, but not impossible. The reconstruction of [Ferguson, 1995, 3.4] suggests how argumentation can exploit domain knowledge to drive the reconciliation of assumptions, even with expressive reasoning; and the logic programming theories of action show how to validate such special inference mechanisms generally by relating computed conclusions to a set of intended models. Combining the strengths of both approaches should lead to a more faithful and powerful reconstruction of planning as computational logic. This is the aim of the next section.

6.4 Branching time, proof theory and validation

We now describe and validate a simple formalism for reasoning about action called ACCH. ACCH uses a framework of argumentation to reconcile competing deductions in modal logic. The presentation that follows is based on [Stone, 1997b]. The definitions and proofs of this validation owe an obvious debt to previous validations, particularly the program of logic programming theory of action discussed in section 6.3.4.

Why the combination? Modal logic represents the inferences we need explicitly but compactly. We use modal operators to describe both change and persistence; we use modal introspection axioms to capture inertia. ([Ginsberg, 1995] also suggests an analogy between frame and modal operators.) Argumentation leaves the leap to planners small, as we saw in the discussion of [Ferguson, 1995]'s parallel between argumentation and the SNLP planner. More importantly, it allows us to reason with standard proofs, and to apply the proof-theoretic results about modal logic obtained in Part I to give interesting insights into the system.

The particular formalism presented here uses prefix semantic translations for modal logic proof [Wallen, 1990; Ohlbach, 1991] and Dung's presentation of argumentation frameworks for defeasible reasoning [1993]. However, modal logic is a general logic of possible states (see [Halpern and Moses, 1985b]), while argumentation provides a general framework for defeasible reasoning (see [Lin and Shoham, 1989; Pollock, 1992; Simari and Loui, 1992]). Thus, we expect the techniques presented here to continue to apply as richer non-monotonic theories of actions are constructed and validated.

6.4.1 A Modal Semantics and a Modal Translation

This section describes a set of intended models for a theory of action; it is based roughly on the semantics of $\mathcal{A}$ from [Gelfond and Lifschitz, 1993]. We introduce a set F of fluent names and a set A of action names; a fluent literal has the form f or $\neg f$, with $f \in F$. (The opposite of a literal $\sim f$ is g if f is $\neg g$, $\neg f$ otherwise.) A domain theory is specified by a set R of causal rules and a set O of observations. A causal rule takes the form:

$$a \textbf{ causes } f \textbf{ if } P_1, \ldots, P_n$$

where a is an action name, and f and all P_i are fluent literals. An observation takes either of the forms:

$$a \textbf{ happens at } t \quad f \textbf{ holds at } t$$

where a is an action name; f is a fluent literal; and t is a natural number. For the purposes of counting times, the first moment in time is time 1, the second moment in time is time 2, and so forth. (This means that the index of a time is one more than the number of steps of change involved in reaching that time; this indexing might have been made more perspicuous by indexing the first time—perhaps counterintuitively—as time 0.)

The models we work with are Kripke models $\langle W, AF \rangle$ where W is a set of worlds and AF is a nonempty set of functions from worlds to worlds encoding accessibility. A world is represented as a pair $\langle S, E \rangle$ where S (the state) is a set of fluent names and E (the event) is a set of action names; a fluent f holds at $\langle S, E \rangle$ iff $f \in S$; $\neg f$ holds at $\langle S, E \rangle$ iff $f \notin S$; and an action a happens at $\langle S, E \rangle$ iff $a \in E$.

Definition 15 *A model $\langle W, AF \rangle$ represents an inertial model of causal rules R if each function $\alpha \in AF$ respects the following constraints for any state $\langle S, E \rangle$:*

- *For any rule in R of the form a* **causes** *f* **if** *$P_1, \ldots, P_n$: if a happens at $\langle S, E \rangle$ and P_1 through P_n hold in $\langle S, E \rangle$, then f holds at $\alpha(\langle S, E \rangle)$.*
- *Otherwise, f holds at $\alpha(\langle S, E \rangle)$ iff f holds at $\langle S, E \rangle$.*

Two actions may occur concurrently if their effects do not interfere with each other; otherwise accessibility functions respect the inertial meaning of causal rules just as transition functions do in [Gelfond and Lifschitz, 1993].

Definition 16 *A model $\langle W, AF \rangle$ is a model of an observation at a world w according to the following criteria:*

- *f* **holds at** *1 at w iff f is true at w; otherwise f* **holds at** *t iff for all $\alpha \in AF$, f* **holds at** *$t \Leftrightarrow 1$ is true at $\alpha(w)$.*

$$
\begin{aligned}
&(a\ \textbf{happens at}\ t)^T = \ [\text{N}]^{t-1}\ \mathbf{h}a\\
&(f\ \textbf{holds at}\ t)^T = \ [\text{N}]^{t-1}\ [\text{H}]\ f\\
&(a\ \textbf{causes}\ f\ \textbf{if}\ P_1,\ldots,P_n)^T\ (l) =\\
&\qquad\qquad [\text{H}]\ (P_1 \wedge \ldots \wedge P_n \wedge \mathbf{h}a \supset [\text{N}]\ [\text{H}]\ f)\\
&\qquad\qquad \wedge\ [\text{H}]\ (\mathbf{h}a \supset ab(\sim f, l))
\end{aligned}
$$

Figure 6.1: Translation $\cdot^T$ to modal logic

- *a* **happens at** *1 at w iff a happens at w; otherwise a* **happens at** *t iff for all $\alpha \in AF$, a* **happens at** *$t \Leftrightarrow 1$.*

Definition 17 *A model $\langle W, AF\rangle$ is an exact model of O at w iff it is a model of O at w, and for every finite sequence $\alpha_1,\ldots,\alpha_m$ of elements of AF (possibly empty), whenever any action a happens at $\alpha_1(\ldots(\alpha_m(w))\ldots)$, there is an observation a* **happens at** *$m+1$.*

Definition 18 *A model $\langle W, AF\rangle$ is an intended model of R and O at w, iff it is an inertial model of R and an exact model of O at w.*

Definition 19 *R and O entails an observation o iff every intended model of R and O at any w is a model of o at w.*

The truth-conditions above mirror those of modal formulas. For example, if [N] (next) is a modal operator interpreted with reference to accessibility functions *AF*, and the formula **h***a* is true of an action *a* in world *w* iff *a* happens there, then *a* **happens at** *t* corresponds to the formula $[\text{N}]^{t-1}$ **h***a*.

The role of **holds** in inertia can also be modeled as a modal operator. If *f* holds at a certain world, then *f* is true there, and *f* will continue to hold in accessible worlds until further notice. By this analogy, we introduce a modality [H] to represent *holding until further notice*; it is subject to the formal axioms $[\text{H}]\ f \supset f$, $[\text{H}]\ f \supset [\text{H}]\ [\text{H}]\ f$ and $[\text{H}]\ f \supset [\text{N}]\ f$. Since [H] only has these properties “by default”, we won’t assign models an explicit set of accessibility functions to interpret it. Instead, we simply use this intuition in designing the representations of the proof system.

The proof system represents assumed facts by a translation $\cdot^T$. Under the translation, *f* **holds at** *t* becomes $[\text{N}]^{t-1}\ [\text{H}]\ f$. Causal rules are interpreted (1) by a clause establishing effects: $[\text{H}]\ (P_1 \wedge \ldots \wedge P_n \wedge \mathbf{h}a \supset [\text{N}]\ [\text{H}]\ f)$; and (2) by a clause triggering abnormality formulas for occlusion: $[\text{H}]\ (\mathbf{h}a \supset ab(\sim f, l))$. (Each causal law is given a distinct symbol *l* to index the source of occlusion.) These translations are summarized in Figure 6.1. Meanwhile, to *prove* an observation, the weaker translation $(f\ \textbf{holds at}\ t)^Q = [\text{N}]^{t-1}\ f$ suffices.

This approach is similar to other translations of $\mathcal{A}$. The differences are the distinctive use of modal operators to encode inertia, the use of facts to encode the occurrence of actions,

$$\frac{\mu = \nu}{\Gamma, f^{\mu} \longrightarrow f^{\nu}, \Delta}\ \text{axiom} \qquad\qquad \frac{\mu\nu = \sigma}{\Gamma, f^{\mu}, \neg f^{\mu} \longrightarrow \bot^{\sigma}, \Delta} \rightarrow \bot$$

$$\frac{\Gamma \longrightarrow A \wedge B^{\mu}, A^{\mu}, \Delta \qquad \Gamma \longrightarrow A \wedge B^{\mu}, B^{\mu}, \Delta}{\Gamma \longrightarrow A \wedge B^{\mu}, \Delta} \rightarrow \wedge$$

$$\frac{\Gamma, A \wedge B^{\mu}, A^{\mu}, B^{\mu} \longrightarrow \Delta}{\Gamma, A \wedge B^{\mu} \longrightarrow \Delta} \wedge \rightarrow \qquad \frac{\Gamma \longrightarrow A \vee B^{\mu}, A^{\mu}, B^{\mu}, \Delta}{\Gamma \longrightarrow A \vee B^{\mu}, \Delta} \rightarrow \vee$$

$$\frac{\Gamma, [\textsc{h}]\, f \longrightarrow \Delta \qquad \Gamma, [\textsc{h}]\, \neg f \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}\ \text{cut}$$

$$\frac{\Gamma, A \supset B^{\mu} \longrightarrow A^{\mu}, \Delta \qquad \Gamma, A \supset B^{\mu}, B^{\mu} \longrightarrow \Delta}{\Gamma, A \supset B^{\mu} \longrightarrow \Delta} \supset\rightarrow$$

$$\frac{\Gamma, [\textsc{n}]\, A^{\mu}, A^{\mu\alpha} \longrightarrow \Delta}{\Gamma, [\textsc{n}]\, A^{\mu} \longrightarrow \Delta} [\textsc{n}] \rightarrow \qquad \frac{\Gamma, [\textsc{h}]\, A^{\mu}, A^{\mu\sigma} \longrightarrow \Delta}{\Gamma, [\textsc{h}]\, A^{\mu} \longrightarrow \Delta} [\textsc{h}] \rightarrow$$

$$\frac{\Gamma \longrightarrow [\textsc{n}]\, A^{\mu}, A^{\mu\alpha}, \Delta}{\Gamma \longrightarrow [\textsc{n}]\, A^{\mu}, \Delta} \rightarrow [\textsc{n}]^{\dagger}$$

Figure 6.2: Path-based, explicitly-scoped sequent calculus for modal logic. † For $(\rightarrow [\textsc{n}])$, α must not appear in the conclusion.

and the elimination of occurrences of negation-as-failure that appear in logic programming theories of action. The effect of negation-as-failure will be restored by an explicit operation of comparing deductions. This allows an ordinary modal logic proof system to apply in the base case.

6.4.2 Proof Theory

Figure 6.2 shows an explicitly-scoped proof system that ACCH uses to construct arguments about entailment in intended models. This calculus adopts the same conventions as the ground calculus for S4 presented in Figure 2.11; however, the calculus is specialized to the logical fragment into which causal and observational statements are translated, and takes into account the interacting behavior of the [N] and [H] operators. Like the calculus of Figure 2.11, the calculus of Figure 6.2 interleaves the steps of the functional translation of modal logic into classical logic, as in [Wallen, 1990; Ohlbach, 1991; Auffray and Enjalbert, 1992], with the inference rules for classical reasoning. Each formula is labeled with a string that represents the path of accessibility to the possible world where the formula must be shown true.

Figure 6.2 is specialized to describe the modal fragment for inertia more precisely. The axiom rule applies to any fluent literals or atomic action occurrence statement. The $(\rightarrow \bot)$ rule encodes how contradictory fluents derive a contradiction. Since the contradiction records the paths of the terms that introduce it, we must test for contradiction "when all is said and done". The cut rule formalizes the fact that at the initial state, either f or $\neg f$ is true, and whichever is true will tend to stay that way. The cut is tractable because it applies only to fluent names and can be restricted to "pseudo"-analytic uses—cases where either f or $\neg f$ is a subformula of the sequent already [D'Agostino and Mondadori, 1994]. Finally, the left modal rules build in the axioms relating them: [N] matches any constant; [H] matches any string.

Given a set of causal rules R and observations O, translated to logic via the rules in Figure 6.1 as R^T and O^T, we can analyze the structure of deductions ending with a sequent $R^T, O^T \longrightarrow \Delta$ to restrict the kinds of proofs that need to be constructed and compared, to a space that can be searched more easily, using the kind of result about proof form presented in Chapter 3. In particular, we can show that any deduction has an equivalent form where cuts occur at the root and only one formula appears on the right in sequents. In such deductions, which we shall term DIRECT, we can distinguish as an ATTACK SITE each subproof that is an input to a (cut) rule but which is not itself obtained by applying a (cut) rule. Without loss of generality, the end-sequent $\Gamma \longrightarrow A$ of an attack site gives a representative set of assumptions Γ under which any conclusion obtained anywhere in the subproof may be challenged.

6.4.3 Argumentation

Following [Dung, 1993], we define an ARGUMENTATION FRAMEWORK as a pair $\mathcal{F} = \langle AR, attacks \rangle$ where AR is a set (of arguments) and $attacks$ is a binary relation on elements of AR. For two arguments D and E, $attacks(D, E)$ means that D argues against the acceptability of E.

ACCH is an argumentation framework in this sense. In ACCH, AR is the set of direct modal proofs as given in the previous section. The relation *attacks* of ACCH is defined as the union of two relations, $attacks_{\mathrm{I}}$ for inertia, and $attacks_{\mathrm{O}}$ for occlusion.

Definition 20 *$attacks_{\mathrm{I}}(D, E)$ if D has end-sequent $\Gamma \longrightarrow ab(f, l)^{\nu}$, E has an attack site with end-sequent $\Gamma \longrightarrow \Delta$, and the attack site contains a rule application $([\mathrm{H}] \rightarrow)$ deriving fluent literal $f^{\lambda\mu}$ from $[\mathrm{H}]\, f^{\lambda}$, where λ is a prefix of ν and ν is a proper prefix of $\lambda\mu$.*

Because inertia is handled by an introspection axiom, we can propagate a fluent forward inertially from the result-situation of the action that establishes the fluent to the result of an arbitrary sequence of subsequent actions in a single step of instantiation. However, propagation into the result state of each action is subject to occlusion by abnormality; an abnormality at a given step challenges both application of inertia at that step and subsequent inertial propagation of the fluent. Definition 20 encodes this.

Definition 21 *$attacks_{\mathrm{O}}(D, E)$ if D has end-sequent $\Gamma \longrightarrow \sim P_i{}^{\mu}$ and E has an attack site with end-sequent $\Gamma \longrightarrow ab(f, l)^{\mu}$ for a rule l with precondition P_i.*

An argument that an action occludes a fluent is challenged by showing that the conditions where the action occludes the fluent are not actually met, because some fluent P_i is sure not to hold.

To describe the consequences of an argumentation framework requires the following definitions [Dung, 1993].

Definition 22 *An argument D is acceptable wrt a set S of arguments iff for each argument E in AR: if attacks*(E, D) *then there is a* D' *in S with attacks*(D', E).

Definition 23 *A set S of arguments is admissible if there are no arguments D and E in S with attacks*(D, E)*, and every argument in S is acceptable wrt S.*

Definition 24 *A preferred extension of an argumentation framework AF is a maximal (wrt set inclusion) admissible set of arguments of AF.*

Dung proves that every argumentation framework in which no infinite sequence of attacks is possible has a unique preferred extension, which can be obtained by a least fixed-point construction. This extension, denoted GE_{AF}, represents the natural consequences of the framework. Dung's theorem applies to attacks defined by definitions 20 and 21, because each argument can be assigned a finite *grade*, based on the prefixes that appear in it, such that only arguments of lower grade attack it. In particular, $ab(f,l)^{\mu}$ arguments can be attacked by arguments for fluents true at μ, but such arguments can only be attacked in turn by other $ab(f',l')^{\nu}$ arguments with ν a proper prefix of μ.

Argumentation is closely connected to logic programming. Search for an acceptable argument in GE_{AF} can be captured as a logic program by the meta-interpreter:

(64) $$\begin{array}{l} acc(D) \leftarrow not\ defeated(D). \\ defeated(D) \leftarrow attacks(D,E), acc(E). \end{array}$$

6.4.4 Validation

We can now prove the following theorem:

Theorem 12 (Correctness) *Let O be a set of observations in which the latest time mentioned is t, and let R be a set of rules. Then R, O entails every observation from a finite set* O' *iff* GE_{AF} *contains an argument with end-sequent* $R^T, O^T \longrightarrow (\bigwedge_{o \in O'} o^Q) \vee [\mathrm{N}]^t \perp$.

We present only a sketch of the proof of this result. Soundness is proved directly, by double induction on the grade of arguments and the structure of proofs. Completeness appeals to a lemma that the models of *R* and *O* can be partitioned into a finite set of types, where each type is a finite description of the initial state that determines what formulas change when during the observations. Thus we can apply the cut rule repeatedly until each attack site in the derivation specifies the type of all of the models of the attack site. We then show that by induction that if all of the models have the same type, then cut-free inference is complete: the observations can be used to determine which causal rules are triggered by each action.

6.5 Examples and problems

In this section we explore the behavior of the argument system ACCH on some interesting examples of reasoning. By presenting the structure of basic common important inferences, we can draw out how ACCH captures the intuitive ontology and reasoning of action and time. We can also compare the formal inferences by which the theory captures particular cases to the data structures typically maintained in planning for them. The assessment of the theory suggested by these examples is not uniformly positive: these examples offer motivation to enrich and streamline the approach. We will consider this task in later chapters.

6.5.1 Arguments: formal and informal

We begin with a comparison between the arguments of ACCH and the argumentation motivated by Konolige and Ferguson [Konolige, 1988; Ferguson, 1995]. Argumentation based on forward persistence is built into our system. Whenever you have a state p—whether it is given or established by causal inference—it comes as $[\mathrm{H}]p$. The $[\mathrm{H}]$ allows p to be instantiated to an arbitrary future point. It is also straightforward to construct forward arguments about the resulting state of an action given a specification of the initial state. These types of arguments exactly correspond to arguments in Konolige's system—and of course also to arguments in Ferguson's.

Konolige also has an explicit rule of backward persistence as a primitive way of constructing an argument. Backward persistence applies when a state holds at one time; it allows us to conclude that the state must also have held earlier, provided nothing caused an intervening change. In ACCH, we can also construct arguments for backward persistence, but they are not primitive. They start with case analysis based on whether the persistent fluent is true initially. If the fluent starts out false, and nothing occurs to change it, then this case will be inconsistent with the available evidence that the fluent is true later. The other case is that the fluent starts out true. Because its alternative is inconsistent, the system supports any acceptable conclusion drawn after making this assumption.

A simple example will illustrate this structure in full. Given only that a fact p holds in the third state, we want to infer by backward persistence that it must also have held in the initial state. The correctness theorem says that to establish this conclusion is to find an acceptable argument

$$[\mathrm{N}]^2 p \longrightarrow p \vee [\mathrm{N}]^2 \bot$$

The proof begins by introducing cases using the (cut) rule: we assume either $[\mathrm{H}]p$ or $[\mathrm{H}]\neg p$.

If $[\mathrm{H}]p$—if p holds until further notice—then in particular p holds in the initial state and hence $p \vee [\mathrm{N}]^2\bot$. Otherwise, we consider two arbitrary $[\mathrm{N}]$ transitions $\alpha\beta$. From the assumption $[\mathrm{H}]\neg p$ we conclude $\neg p^{\alpha\beta}$ by forward inertia. The given fact $[\mathrm{N}]^2[\mathrm{H}]p$ can likewise be instantiated to $p^{\alpha\beta}$. This establishes $\bot^{\alpha\beta}$, and hence $[\mathrm{N}]^2\bot$ and hence $p \vee [\mathrm{N}]^2\bot$.

This argument is represented by the following formal object.

(65)

$$\dfrac{\dfrac{\dfrac{\ldots p \longrightarrow p\ldots}{\ldots[\mathrm{H}]p \longrightarrow p\ldots}\,[\mathrm{H}]\rightarrow}{[\mathrm{N}]^2[\mathrm{H}]p, [\mathrm{H}]p \longrightarrow p \vee [\mathrm{N}]^2\bot}\,\rightarrow\vee \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\ldots p^{\alpha\beta}, \neg p^{\alpha\beta} \longrightarrow \bot^{\alpha\beta}\ldots}{\ldots p^{\alpha\beta}, [\mathrm{H}]\neg p \longrightarrow \bot^{\alpha\beta}\ldots}\,[\mathrm{H}]\rightarrow^*}{\ldots[\mathrm{H}]p^{\alpha\beta}, [\mathrm{H}]\neg p \longrightarrow \bot^{\alpha\beta}\ldots}\,[\mathrm{H}]\rightarrow}{\ldots[\mathrm{N}][\mathrm{H}]p^{\alpha}, [\mathrm{H}]\neg p \longrightarrow \bot^{\alpha\beta}\ldots}\,[\mathrm{N}]\rightarrow}{\ldots[\mathrm{N}]^2[\mathrm{H}]p, [\mathrm{H}]\neg p \longrightarrow \bot^{\alpha\beta}\ldots}\,[\mathrm{N}]\rightarrow}{\ldots[\mathrm{N}]^2[\mathrm{H}]p, [\mathrm{H}]\neg p \longrightarrow [\mathrm{N}]\bot^{\alpha}\ldots}\,\rightarrow[\mathrm{N}]}{\ldots[\mathrm{N}]^2[\mathrm{H}]p, [\mathrm{H}]\neg p \longrightarrow [\mathrm{N}]^2\bot\ldots}\,\rightarrow[\mathrm{N}]}{[\mathrm{N}]^2[\mathrm{H}]p, [\mathrm{H}]\neg p \longrightarrow p \vee [\mathrm{N}]^2\bot}\,\rightarrow\vee}{[\mathrm{N}]^2[\mathrm{H}]p \longrightarrow p \vee [\mathrm{N}]^2\bot}\,\text{cut}$$

As used in (65), forward persistence determines not only what tentative conclusions follow from backward persistence but also what counterarguments can be used to defeat those conclusions. Here, for example, the topmost $([\mathrm{H}] \rightarrow)$ inference is starred to indicate that it represents a default instantiation where this argument could be attacked. An attack there could establish an abnormality of $\neg p$ in the initial state or in state α.

Konolige describes a second kind of primitive argument for reasoning backward in time. These arguments draw conclusions about the initial state in which an action was performed given a description of the action's result state. For example, Konolige considers the infamous action of shooting: if a shoot occurs when the gun is loaded, the result is that the target is dead. In addition to this rule for forward argumentation, Konolige explicitly specifies a rule for backward argumentation: if the target is not dead (d) after a shoot (s), then the gun must be initially not loaded (l).

As with backward persistence, ACCH captures this causal backward reasoning by a combination of inferences for forward reasoning together with case analysis describing the initial state. For example, here is how we duplicate Konolige's backward argument that the gun must have been unloaded. Initially, the gun is either loaded or not. If it is loaded, then the shooting leads to death, by forward reasoning. This is impossible, because we know that there is still life after the shooting. Thus, the gun must not be loaded.

Again, we can supply the formal argument in ACCH that expresses this reasoning:

(66)

$$\dfrac{\dfrac{\dfrac{\ldots\neg l \longrightarrow \neg l\ldots}{\ldots[\mathrm{H}]\neg l \longrightarrow \neg l\ldots}\,[\mathrm{H}]\rightarrow}{\ldots[\mathrm{H}]\neg l \longrightarrow \neg l \vee [\mathrm{N}]\bot}\,\rightarrow\vee \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\ldots\mathbf{h}s, [\mathrm{H}]l \longrightarrow l \wedge \mathbf{h}s \qquad \dfrac{\dfrac{\ldots d^{\alpha}, \neg d^{\alpha} \longrightarrow \bot^{\alpha}, \ldots}{\ldots[\mathrm{H}]d^{\alpha}, \neg d^{\alpha} \longrightarrow \bot^{\alpha}, \ldots}\,[\mathrm{N}]\rightarrow}{\ldots[\mathrm{N}][\mathrm{H}]d, \neg d^{\alpha} \longrightarrow \bot^{\alpha}, \ldots}\,[\mathrm{N}]\rightarrow}{\ldots l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d, \mathbf{h}s, \neg d^{\alpha}, [\mathrm{H}]l \longrightarrow \bot^{\alpha}, \ldots}\,\supset\rightarrow}{\ldots l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d, \mathbf{h}s, [\mathrm{N}]\neg d, l \longrightarrow \bot^{\alpha}, \ldots}\,[\mathrm{N}]\rightarrow}{\ldots l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d, \mathbf{h}s, [\mathrm{N}]\neg d, [\mathrm{H}]l \longrightarrow \bot^{\alpha}, \ldots}\,[\mathrm{H}]\rightarrow}{[\mathrm{H}](l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d), \mathbf{h}s, [\mathrm{N}]\neg d, [\mathrm{H}]l \longrightarrow \bot^{\alpha}, \ldots}\,[\mathrm{H}]\rightarrow}{[\mathrm{H}](l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d), \mathbf{h}s, [\mathrm{N}]\neg d, [\mathrm{H}]l \longrightarrow [\mathrm{N}]\bot, \ldots}\,\rightarrow[\mathrm{N}]}{[\mathrm{H}](l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d), \mathbf{h}s, [\mathrm{N}]\neg d, [\mathrm{H}]l \longrightarrow \neg l \vee [\mathrm{N}]\bot}\,\rightarrow\vee}{[\mathrm{H}](l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d), \mathbf{h}s, [\mathrm{N}]\neg d \longrightarrow \neg l \vee [\mathrm{N}]\bot}\,\text{cut}$$

By deriving backward reasoning from forward reasoning and case analysis, ACCH offers a more parsimonious formalism that Konolige's. (As we shall see, case analysis cannot be avoided in building reliable plans from partial descriptions of the world.) The fact that such reasoning can be performed at all, however, highlights another important difference between ACCH and both Konolige's and Ferguson's frameworks for argumentation.

Both Konolige and Ferguson take it for granted that arguments that yield contradictory conclusions are incompatible. This means that whenever a derivation of $\perp$ is constructed in their systems, some component of the derivation must be rejected. This principle is difficult to reconcile with reasoning by a process of elimination. Each eliminative step consists of a derivation of $\perp$; so the reasoning system is compelled to reject some component of each step.

In ACCH, arguments involving backward reasoning and other processes of elimination are possible because ACCH completely separates contradiction from compatibility. The logical rules describing negation and $\perp$ determine contradiction. The domain-specific definition of attack determines compatibility.

Pollock's distinction between rebuttal and undercutting in argumentation can help provide an intuitive clarification of the distinction between the different frameworks [Pollock, 1992]. Two arguments rebut each other if they derive opposite conclusions. One argument undercuts another if it challenges the regularity that connects a premise of the argument and its conclusion. Pollock has developed systems of argumentation that include both rebuttal and undercutting. The reasoning framework proposed by [Simari and Loui, 1992] and used in [Ferguson, 1995] includes only rebuttal, and Konolige certainly emphasizes rebuttal (it is unclear whether he would restrict himself to it). On the other hand, the framework proposed in [Dung, 1993] and used in ACCH involves only undercutting.

In some sense, undercutting is more general that rebuttal; we can imagine adapting every rebuttal (showing that the present case is a genuine exception to a generalization) into an undercutting argument (showing that we shouldn't apply the generalization to the present case). [Dung, 1993] uses this idea to reconstruct Pollock's proposals in his framework. The ability to express rebuttal might seem crucial because of the importance of allowing an agent to handle exceptions to generalizations it knows. An agent must be resilient to the occurrence of unexpected and strange events. It can't respond by wallowing in inconsistency. Incorporating rebuttal in the agent's reasoning is a sure way of avoiding this.

Obviously, the agent must somehow respond when it finds an inconsistency in its own beliefs, but it is not clear that the right answer is to throw out conclusions, as dictated by rebuttal. Perhaps what should go is the principles those conclusions are drawn from. Rebuttal seems therefore not a general solution to inconsistency. But the success of ACCH suggests that rebuttal is not even a partial solution to the problem of inconsistency. Rebuttal gets in the way of simple, useful inferences. In fact, as soon as you specify a set of intended models for a logical theory of action, the possibility arises of describing courses of events that cannot occur in any intended model. Detecting the inconsistency of these descriptions is part and parcel of reasoning correctly in the theory. Moreover, as we have seen, often the most perspicuous way to determine that something must have held is to recognize that no alternative to this could have been possible in an intended model.

6.5.2 Argumentation and conflicting predictions

Any interesting problem of temporal reasoning involves potential conflict between defaults. The argumentation performed in ACCH resolves such conflicts in strict accord with explicit causality and inertia, in an appealingly simple way.

The classic example of the problem of conflicting defaults in temporal reasoning is the Yale shooting problem [Hanks and McDermott, 1987]. The task in this problem is to conclude that a victim, initially not dead (*d*), ends up dead when a gun starts out loaded (*l*), then there is a delay (*w*: we wait), and then the gun is fired (*s*). The causal description of this scenario is this:

$$
\begin{array}{l}
s\ \textbf{causes}\ d\ \textbf{if}\ l \\
\neg d\ \textbf{holds at}\ 1 \\
l\ \textbf{holds at}\ 1 \\
w\ \textbf{happens at}\ 1 \\
s\ \textbf{happens at}\ 2
\end{array}
$$

Using our modal translation, this corresponds to the collection of axioms below:

$$[\mathrm{H}](\mathbf{h}s \supset ab(\neg d, 1)), [\mathrm{H}](l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d), [\mathrm{H}]\neg d, [\mathrm{H}]l, \mathbf{h}w, [\mathrm{N}]\mathbf{h}s,$$

This scenario requires us to choose which of two instances of inertia to preserve. If we assume that the event of shooting succeeds in killing the victim, then this will be an exception to inertia later. On the other hand, we could assume that there is an earlier exception to inertia, so that the gun is effectively unloaded magically during the wait. Then there will be no exceptions to inertia later, because attempting to shoot won't do anything. In reasoning formalisms that have a limited capacity to resolve conflicting defaults or use global minimization of the number of exceptions, these two alternatives can seem equally valid. As a result, these frameworks give the Yale Shooting Problem an unintended or surprising treatment.

Argumentation avoids such surprising behavior by specifying how conflicting defaults are to be resolved on the basis of local information. For example, in the Yale shooting problem, we conclude that the gun stays loaded during the wait because the scenario does not allow us to build a counterargument to inertia for that fluent at that moment. Similarly, we cannot conclude that the victim stays alive after the shoot because the occurrence of the shooting forms the basis of an argument attacking, and defeating, this argument from inertia. We conclude that the victim in fact dies by combining the argument that the gun stays loaded with the effects axiom for shoot.

We briefly observe how this argumentation plays out formally. First, the argument by inertia that the victim never dies looks like this:

$$
\dfrac{\dfrac{\dfrac{\ldots, \neg d^{\alpha\beta} \longrightarrow \neg d^{\alpha\beta}}{\ldots, [\mathrm{H}]\neg d, \longrightarrow \neg d^{\alpha\beta}}\,[\mathrm{H}]\rightarrow^{*}}{\ldots, [\mathrm{H}]\neg d, \longrightarrow [\mathrm{N}]\neg d^{\alpha}}\,\rightarrow[\mathrm{N}]}{[\mathrm{H}](\mathbf{h}s \supset ab(\neg d, 1)), [\mathrm{H}](l \wedge \mathbf{h}s \supset [\mathrm{N}][\mathrm{H}]d), [\mathrm{H}]l, [\mathrm{H}]\neg d, [\mathrm{N}]\mathbf{h}s, \mathbf{h}w \longrightarrow [\mathrm{N}]^{2}\neg d}\,\rightarrow[\mathrm{N}] \tag{67}
$$

Again, we star the $([\textsc{h}] \rightarrow)$ step of instantiation that encodes the default of inertia. This is a place where the argument can be—and is—attacked. Here is the attacker:

(68)
$$\dfrac{\dfrac{\dfrac{\ldots, \mathbf{h}s^{\alpha} \longrightarrow \mathbf{h}s^{\alpha}, \ldots \qquad \ldots, ab(\neg d, 1)^{\alpha} \longrightarrow ab(\neg d, 1)^{\alpha}}{\ldots, \mathbf{h}s \supset ab(\neg d, 1)^{\alpha}, \mathbf{h}s^{\alpha}, \longrightarrow ab(\neg d, 1)^{\alpha}}\,{\supset}{\rightarrow}}{\ldots, [\textsc{h}](\mathbf{h}s \supset ab(\neg d, 1)), \mathbf{h}s^{\alpha}, \longrightarrow ab(\neg d, 1)^{\alpha}}\,[\textsc{h}] \rightarrow}{\ldots, [\textsc{h}](\mathbf{h}s \supset ab(\neg d, 1)), [\textsc{n}]\mathbf{h}s, \longrightarrow ab(\neg d, 1)^{\alpha}}\,[\textsc{n}] \rightarrow$$

This argument indicates that we expect aliveness to be abnormal at α given that a shooting occurs then. Now, either this occlusion argument must be defeated or the first inertia argument will be defeated. The only way to defeat this argument by occlusion would be to show that the conditions do not hold where shooting would cause death. That would require an argument for $\neg l^{\alpha}$. No such argument is available, so the occlusion is acceptable and the inertia is defeated.

The more interesting argument, for death, is this:

(69)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\ldots, l^{\alpha}, \mathbf{h}s^{\alpha} \longrightarrow l \wedge \mathbf{h}s^{\alpha}, \ldots \qquad \dfrac{\dfrac{\ldots, d^{\alpha\beta} \longrightarrow d^{\alpha\beta}, \ldots}{\ldots, [\textsc{h}]d^{\alpha\beta} \longrightarrow d^{\alpha\beta}, \ldots}\,[\textsc{h}] \rightarrow}{\ldots, [\textsc{n}][\textsc{h}]d^{\alpha} \longrightarrow d^{\alpha\beta}, \ldots}\,[\textsc{n}] \rightarrow}{\ldots, l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d^{\alpha}, l^{\alpha}, [\textsc{h}]\neg d, \mathbf{h}s^{\alpha}, \mathbf{h}w \longrightarrow d^{\alpha\beta}, \ldots}\,{\supset}{\rightarrow}}{\ldots, l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d^{\alpha}, [\textsc{h}]l, [\textsc{h}]\neg d, \mathbf{h}s^{\alpha}, \mathbf{h}w \longrightarrow d^{\alpha\beta}, \ldots}\,[\textsc{h}] \rightarrow^{*}}{\ldots, [\textsc{h}](l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d), [\textsc{h}]l, [\textsc{h}]\neg d, \mathbf{h}s^{\alpha}, \mathbf{h}w \longrightarrow d^{\alpha\beta}, \ldots}\,[\textsc{h}] \rightarrow}{[\textsc{h}](\mathbf{h}s \supset ab(\neg d, 1)), [\textsc{h}](l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d), [\textsc{h}]l, [\textsc{h}]\neg d, \mathbf{h}s^{\alpha}, \mathbf{h}w \longrightarrow d^{\alpha\beta}}\,[\textsc{n}] \rightarrow}{[\textsc{h}](\mathbf{h}s \supset ab(\neg d, 1)), [\textsc{h}](l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d), [\textsc{h}]l, [\textsc{h}]\neg d, [\textsc{n}]\mathbf{h}s, \mathbf{h}w \longrightarrow d^{\alpha\beta}}\,\rightarrow [\textsc{n}]}{[\textsc{h}](\mathbf{h}s \supset ab(\neg d, 1)), [\textsc{h}](l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d), [\textsc{h}]l, [\textsc{h}]\neg d, [\textsc{n}]\mathbf{h}s, \mathbf{h}w \longrightarrow [\textsc{n}]d^{\alpha}}\,\rightarrow [\textsc{n}]}{[\textsc{h}](\mathbf{h}s \supset ab(\neg d, 1)), [\textsc{h}](l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d), [\textsc{h}]l, [\textsc{h}]\neg d, [\textsc{n}]\mathbf{h}s, \mathbf{h}w \longrightarrow [\textsc{n}]^{2}d}\,\rightarrow [\textsc{n}]$$

This cumbersome notation indicates just that we expect death at $\alpha\beta$ because at α we fire a loaded gun. The only defeasible step in this structure, again starred, leaps by inertia from $[\textsc{h}]l$ to l^{α}. To attack this, we must find an event that happens in the initial state and that might occlude the fluent l. Since waiting is the only thing that happens in the initial state and waiting occludes no fluent, we cannot build such a counterargument. And hence this argument for death is also acceptable.

This discussion of the Yale shooting problem in ACCH may leave one question. As soon as we see the shooting event, we're skeptical of the victim staying alive, because we can derive the argument in (68). So what if the gun initially isn't loaded? We had better be able to show that the victim does stay alive in that case.

As before, we have a variant of arguments (67) and (68) showing inertia—you should stay alive because you were alive—and occlusion—aliveness should be abnormal when the shooting occurs:

(70)
$$\dfrac{\dfrac{\dfrac{\ldots, \neg d^{\alpha\beta} \longrightarrow \neg d^{\alpha\beta}}{\ldots, [\textsc{h}]\neg d, \longrightarrow \neg d^{\alpha\beta}}\,[\textsc{h}] \rightarrow^{*}}{\ldots, [\textsc{h}]\neg d, \longrightarrow [\textsc{n}]\neg d^{\alpha}}\,\rightarrow [\textsc{n}]}{[\textsc{h}](\mathbf{h}s \supset ab(\neg d, 1)), [\textsc{h}](l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d), [\textsc{h}]\neg l, [\textsc{h}]\neg d, [\textsc{n}]\mathbf{h}s, \mathbf{h}w \longrightarrow [\textsc{n}]^{2}\neg d}\,\rightarrow [\textsc{n}]$$

(71) $$\dfrac{\dfrac{\dfrac{\ldots, \mathbf{h}s^{\alpha} \longrightarrow \mathbf{h}s^{\alpha}, \ldots \qquad \ldots, ab(\neg d, 1)^{\alpha} \longrightarrow ab(\neg d, 1)^{\alpha}}{\ldots, \mathbf{h}s \supset ab(\neg d, 1)^{\alpha}, \mathbf{h}s^{\alpha}, \longrightarrow ab(\neg d, 1)^{\alpha}}\ {\supset}{\to}}{\ldots, [\text{H}](\mathbf{h}s \supset ab(\neg d, 1)), \mathbf{h}s^{\alpha}, \longrightarrow ab(\neg d, 1)^{\alpha}}\ [\text{H}] \to}{\ldots, [\text{H}](\mathbf{h}s \supset ab(\neg d, 1)), [\text{N}]\mathbf{h}s, \longrightarrow ab(\neg d, 1)^{\alpha}}\ [\text{N}] \to$$

These arguments are different from the earlier ones only because of the different set of premises they appeal to. However, now we also have the following argument:

(72) $$\dfrac{\ldots, \neg l^{\alpha} \longrightarrow \neg l^{\alpha}}{\ldots, [\text{H}]\neg l \longrightarrow \neg l^{\alpha}}\ [\text{H}] \to^{*}$$

Argument (72) presents an attack on argument (71) on occlusion; s depends on the precondition l to cause d, and this argument shows l does not hold when s occurs. Argument (72) involves a defeasible step which propagates $\neg l$ from the initial state to α by inertia. Since the only thing that happens in that interval is a wait, no argument challenges this step. Argument (72) is therefore acceptable. It follows that argument (71) is defeated, which reinstates argument (70). That is how inertia can be maintained even in the presence of an event that is executed safely but could have been potentially disruptive.

These examples illustrate the general way in which ACCH manages causality and inertia. With this same generality, the examples also highlight how strong the idealization is that gives us the intended models of a theory of action. The idealization elevates our specification of the domain to an absolute. Misfires and jams become absolutely impossible when not mentioned; so do all other causal failures of shooting. And there is no room for good citizens, who if they chanced upon a loaded gun would unload it, nor for any other unpredictable failures of inertia. Given the force of causality and inertia, we must never forget that they—and not some unanalyzed idea of common-sense—govern the meaning of ACCH domain specifications.

The reasoning we have just seen in the Yale shooting problem and its alternative can be recast as a problem for reasoning backward, in a variant known as the Stanford murder mystery. The reasoning required to solve the Stanford murder mystery in ACCH underscores the close connection we have observed between forward reasoning and backward reasoning in ACCH.

The Stanford murder mystery assumes the same events and the same causal laws as the Yale shooting problem. The difference is that instead of knowing initially that the gun is loaded, we know that ultimately, the victim ends up dead. We want to be able to CONCLUDE that the gun must have been loaded initially. In causal language, the scenario is this:

$$\begin{array}{c} s\ \textbf{causes}\ d\ \textbf{if}\ l \\ \neg d\ \textbf{holds at}1 \\ w\ \textbf{happens at}1 \\ s\ \textbf{happens at}2 \\ \neg alive\ \textbf{holds at}3 \end{array}$$

The argument in ACCH that solves this problem is the following formal object:

(73)
$$\begin{prooftree}
\AxiomC{$\ldots, l \longrightarrow l, \ldots$}
\RightLabel{$[\textsc{h}] \rightarrow$}
\UnaryInfC{$\ldots, [\textsc{h}]l \longrightarrow l, \ldots$}
\RightLabel{$\rightarrow \vee$}
\UnaryInfC{$\ldots, [\textsc{h}]l, \longrightarrow l \vee [\textsc{n}]^2\bot$}
\AxiomC{$\ldots, d^{\alpha\beta}, \neg d^{\alpha\beta} \longrightarrow \bot^{\alpha\beta}, \ldots$}
\RightLabel{$[\textsc{h}] \rightarrow^*$}
\UnaryInfC{$\ldots, d^{\alpha\beta}, [\textsc{h}]\neg d \longrightarrow \bot^{\alpha\beta}, \ldots$}
\RightLabel{$[\textsc{h}] \rightarrow$}
\UnaryInfC{$\ldots, [\textsc{h}]d^{\alpha\beta}, [\textsc{h}]\neg d \longrightarrow \bot^{\alpha\beta}, \ldots$}
\RightLabel{$[\textsc{n}] \rightarrow$}
\UnaryInfC{$\ldots, [\textsc{n}][\textsc{h}]d^{\alpha}, [\textsc{h}]\neg d \longrightarrow \bot^{\alpha\beta}, \ldots$}
\RightLabel{$[\textsc{n}] \rightarrow$}
\UnaryInfC{$\ldots, [\textsc{n}]^2[\textsc{h}]d, [\textsc{h}]\neg d \longrightarrow \bot^{\alpha\beta}, \ldots$}
\RightLabel{$\rightarrow [\textsc{n}]$}
\UnaryInfC{$\ldots, [\textsc{n}]^2[\textsc{h}]d, [\textsc{h}]\neg d \longrightarrow [\textsc{n}]\bot^{\alpha}, \ldots$}
\RightLabel{$\rightarrow [\textsc{n}]$}
\UnaryInfC{$\ldots, [\textsc{n}]^2[\textsc{h}]d, [\textsc{h}]\neg d \longrightarrow [\textsc{n}]^2\bot, \ldots$}
\RightLabel{$\rightarrow \vee$}
\UnaryInfC{$\ldots, [\textsc{h}]\neg l, [\textsc{n}]^2[\textsc{h}]d, [\textsc{h}]\neg d \longrightarrow l \vee [\textsc{n}]^2\bot$}
\RightLabel{cut}
\BinaryInfC{$[\textsc{h}](\mathbf{h}s \supset ab(\neg d, 1)), [\textsc{h}](l \wedge \mathbf{h}s \supset [\textsc{n}][\textsc{h}]d), [\textsc{h}]\neg d[\textsc{n}]^2 d, [\textsc{n}]\mathbf{h}s, \mathbf{h}w \longrightarrow l \vee [\textsc{n}]^2\bot$}
\end{prooftree}$$

In words, this argument suggests a separate consideration of the two cases where the gun starts out loaded and where it starts out unloaded. In the first case, clearly the gun is loaded; in the second, there is a contradiction, because the victim ought not to die.

In light of the previous discussion, it is easy to see why this argument is acceptable. According to the definition of attack, the subarguments for the two cases will be judged separately, each according to the assumptions it makes. These cases correspond to the two scenarios discussed earlier. In particular, when the gun is not loaded, we accept the argument by inertia that the victim lives. As we saw, the only potential attack against it is based on the shooting and is defeated because of precondition failure.

In ACCH, unlike most other frameworks, this computation is unaffected by the knowledge that the victim must be dead after two steps. Because the definition of conflict in ACCH is so restrictive, the contradiction between the victim being dead in two steps and the victim being alive in two steps does not constitute an attack on either conclusion.

6.5.3 Argumentation and Planning: Connections

With this appreciation about how ACCH decomposes and accomplishes reasoning tasks, we can revisit the deductive formulation of planning introduced in section 6.2, recast it explicitly in ACCH's terms, and undertake a detailed comparison with special-purpose planning algorithms.

In planning, we want devise a reason why some course of action should lead to a desired state of affairs, so that we can use this reason to guide further deliberation and action. In ACCH, we will represent the goal G as a conjunction of fluent literals $g_1 \wedge \ldots \wedge g_m$. Meanwhile, we will represent the domain by a set of causal rules R and a set of observations O about the initial state. To construct a plan to achieve G after n actions, using ACCH, we find an acceptable argument with end-sequent

$$R^T, O^T, H \longrightarrow [\textsc{n}]^n\, G$$

Here H consists of a set of at most n formulas of the form $[\textsc{n}]^k\ \mathbf{h}a$ with $k < n$. This collection H specifies the actions taken as part of the plan.

When we introduced this idea in section 6.2, we had only an informal way of talking about the kind of derivation that went into an argument, the temporal ontology the argument appealed to, and how the argument was judged good. With the development in section 6.4

and thus far in 6.5, we now have a more precise characterization of what these objects are and how they work. This precise characterization allows us to establish more concretely that these arguments match the data structures maintained by a partial-order planner, and that the actions that must be taken to flesh out an argument and ensure its acceptability correspond to the steps taken by partial-order planning algorithms.

Let's begin by recalling some observations from Part I. Proof search is undertaken by building a sequent proof from bottom up, applying applicable rules until the proof is completed or until the possible rules are exhausted. To implement such a strategy, we need to use variables to abstract the choices of instantiated terms during the constructed proof, and we need to accumulate and solve the constraints on the values of variables that are imposed in completing the proof. In argumentation, we may also need to accumulate constraints that defuse attacks on the argument that arise only under particular values of variables. To implement the strategy, we also want to consider sequent rules in a restricted but general order, so as to cut down branching in search where possible. We will see in planning a restriction on search similar to the logic programming restriction on search explored in Chapter 3.

With this in mind, let's look at the structure that ACCH gives an argument that a goal can be satisfied. The claim is that this argument will represent all of the information maintained in a partial order planner constructing a corresponding plan.

The argument records the actions that occur in the plan, using the premises H. We can think of these premises as containing action variables that are constrained as the proof is constructed, or we can suppose that these premises are actually assumed as proof search proceeds, as in abductive event-calculus planning [Shanahan, 1989]. The argument represents bindings in terms of the associated constraints on instantiations of ordinary variables. It also represents ordering constraints between actions in the plan in terms of the associated constraints on the instantiations of temporal variables—variables representing transitions of causal change and inertial persistence.

Finally, the argument records CAUSAL LINKS, by a certain recurring pattern of inference steps in the argument. In a partial-order planner like SNLP, a causal link records the fact that the effect p of action e_1 is used in the plan to achieve a precondition q of another action e_2. (This is represented using the notation $e_1 \xrightarrow{p;q} e_2$.) Whenever such a record is required in a plan, the corresponding argument will contain an inference chain of the form illustrated in 6.1.

$$\dfrac{\dfrac{\dfrac{\ldots, \mathbf{h}e_1^{\sigma} \longrightarrow q'^{\sigma}, \ldots \qquad \ldots, \mathbf{h}e_1^{\sigma} \longrightarrow \mathbf{h}e_1^{\sigma}, \ldots}{\ldots, \mathbf{h}e_1^{\sigma} \longrightarrow q' \wedge \mathbf{h}e_1^{\sigma}, \ldots} \supset\rightarrow \qquad \dfrac{\dfrac{\ldots, p^{\sigma u x} \longrightarrow q^{\tau}, \ldots}{\ldots, [\mathrm{H}]p^{\sigma u} \longrightarrow q^{\tau}, \ldots}[\mathrm{H}]\rightarrow^{*}}{\ldots, [\mathrm{N}][\mathrm{H}]p^{\sigma} \longrightarrow q^{\tau}, \ldots}[\mathrm{N}]\rightarrow}{\ldots, \mathbf{h}e_1^{\sigma}, q' \wedge \mathbf{h}e_1 \supset [\mathrm{N}][\mathrm{H}]p^{\sigma} \longrightarrow q^{\tau}, \ldots}\supset\rightarrow}{\ldots, \mathbf{h}e_1^{\sigma}, [\mathrm{H}](q' \wedge \mathbf{h}e_1 \supset [\mathrm{N}][\mathrm{H}]p) \longrightarrow q^{\tau}, \ldots}[\mathrm{H}]\rightarrow \tag{6.1}$$

The precondition q, to be established at the time τ at which e_2 occurs, appears on the right of the sequent arrow; on the left appear the fact that e_1 occurs at some time σ and the translation of the causal law according to which events such as e_1 lead to effect p under circumstances q'. The proof decomposes the causal law according to the rules for temporal and propositional connectives so as to refer to the occurrence of e_1 at σ and to link the effect p with q in an axiom; establishing the preconditions q'^{σ} of the law is left open in the leftmost node of the subproof.

The form of the causal link shown in 6.1 accounts not only for the planner's representation of the causal link itself but also for the processing that the planner undertakes when it puts the causal link in the plan. In planning by proof search, elaborating a proof to include such a causal connection imposes a number of requirements. A first-order equality constraint $p = q$ must be adopted to ensure that the rightmost axiom link is correct. Planners adopt similar bindings. The axiom also requires a temporal equation $\sigma ux = \tau$. In this equation, σ represents the time when action e_1 occurs, τ represents the time when action e_2 occurs, u represents the causal transition with e_1, and x is an inertial variable that ranges over any series of subsequent steps (in which p remains unchanged). As a constraint, this is very suggestive of the planner's ordering of e_1 before e_2 which must be adopted with the causal link. Further, this elaboration introduces a number of new open nodes in the proof, associated with the new open subproof $\ldots \longrightarrow q'^{\sigma}$. The planner posts analogous goals in its means-end analysis.

The representation of the causal link in 6.1 also includes a starred, default application of $([\mathrm{H}] \rightarrow)$. The instantiation of x in this inference represents a new point where this argument can be attacked by a counterargument. Building and defusing these counterarguments corresponds to the threat resolution phase of planning. In particular, counterarguments arise with any action e_3 in the plan that could occur at a time z to establish $ab(r, n)^z$ where $\sigma uy = z$ and $zy' = \tau$ and $r = p$.

One way to resolve such threats is to add additional constraints to e_3 in the plan to rule out these conditions. This eliminates the potential attack relationship. For example, we could constrain first-order variables using inequalities so that $r = p$ had no solution. In SNLP and UCPOP, this strategy is called separation. Alternatively, we can constrain temporal variables using inequalities, by imposing $z \leq \sigma$ or $z \not\leq \tau$. Such constraints correspond to the promotion and demotion strategies of SNLP and UCPOP.[2]

The alternative is to accept such constraints, adding to the plan $\sigma uy = z$ and $zy' = \tau$ and $r = p$. Instead, we defeat the counterargument by finding the opposite $\neg q''$ of some

[2]As is explained in [Stone, 1997b], this is slightly imprecise, in that logic programming proof search could introduce a fresh variable any time an action occurrence is used. When an attacking argument depends on the order in which some action occurs, constraining this variable would not prevent the attacker from arising again using a new fresh variable and the same instantiation. Technically, this should be resolved using constraints that refer only to the number of steps taken, like $|z| \leq |\sigma|$ or $|z| \leq |\tau|$, since the number of steps taken is independent of the instantiated variable used. This wrinkle is smoothed by the alternative representation of action assumptions developed in Chapter 7, but I leave the correct notation here.

Rules:

(1)	$puton(X, Y)$	**causes** $on(X, Y)$
		if $clear(X) \wedge clear(Y)$
(2)	$puton(X, Y)$	**causes** $\neg clear(Y)$
		if $clear(X) \wedge clear(Y) \wedge \neg table(Y)$
(3)	$puton(X, Y)$	**causes** $clear(Z)$
		if $clear(X) \wedge clear(Y) \wedge on(X, Z)$
(4)	$puton(X, Y)$	**causes** $\neg on(X, Z)$
		if $clear(X) \wedge clear(Y) \wedge on(X, Z)$

Figure 6.3: The blocks world. Causal rules are obtained by substituting distinct values in $\{a, b, c, t\}$ for *X, Y* and *Z.*

precondition of e_3 disrupting p and constructing an acceptable argument that

$$\ldots \longrightarrow \neg q''^{z}$$

As a new open node for proof, this takes the same form as any other goal and becomes part of the requirements for argumentation. This strategy corresponds to UCPOP's special separation rule for conditional effects.

A simple example

Consider how this characterization of plans describes the Sussman anomaly. We have an initial state description O in which blocks a and b are on a table t, with block c on block b. The problem is to use the theory of action consisting of the rules R in Figure 6.3 to bring about a state where a is on b, and b is on c. In fact, this can be achieved in three steps, by finding an acceptable argument D with end-sequent:

$$R^T, O^T, M, N, P \longrightarrow [\mathrm{N}]\,[\mathrm{N}]\,[\mathrm{N}]\,(on(a, b) \wedge on(b, c))$$

The actions of the plan are $M = [\mathrm{N}]^i\,\mathbf{h}puton(c, t), N = [\mathrm{N}]^j\,\mathbf{h}puton(a, b)$, and $P = [\mathrm{N}]^k\,\mathbf{h}puton(b, c)$. The basic structure of D is as follows. We must prove the goal at a path $\mu = \alpha\beta\gamma$. By propagating the initial state inertially along a path x of length i, we establish $[\mathrm{N}]\ [\mathrm{H}]\ clear(b)^x$ as a result of putting c on t. By applying inertia along a path y of length j to this result and the initial state, we show that putting a on b results in $[\mathrm{N}]\ [\mathrm{H}]\ on(a, b)^y$. This introduces the constraint $x < y$; using the result to help establish the the goal adds the constraint $y < \mu$. Meanwhile, by propagating the initial state inertially along a path z of length k, and combining this with $[\mathrm{N}]\ [\mathrm{H}]\ clear(b)^x$ (from putting c on t), we get $[\mathrm{N}]\ [\mathrm{H}]\ on(b, c)^z$ as the result of putting b on c. This finishes the proof of the goal, with the constraints $x < z < \mu$.

This proof is subject to attack, because we can use the success of putting a on b to prove $ab(clear(b), 2)^y$. This attacks the plan's inertial propagation of $[\mathrm{N}]\ [\mathrm{H}]\ clear(b)^x$ to z, if

Rules:

(0)	*hide* **causes** *enabled*	(3)	*dunkA* **causes** ¬*enabled* **if** *bombA*
$(1a)$	*hide* **causes** *bombA* **if** *hidingPlaceA*	(4)	*dunkB* **causes** ¬*enabled* **if** *bombB*
$(1b)$	*hide* **causes** ¬*bombB* **if** *hidingPlaceA*		
$(2a)$	*hide* **causes** *bombB* **if** ¬*hidingPlaceA*	**Observation:**	
$(2b)$	*hide* **causes** ¬*bombA* **if** ¬*hidingPlaceA*	(5)	*hide* **happens at** 1

Figure 6.4: The bomb in the toilet problem.

$x < y < z$. Since z cannot precede x, we must add the constraint $|z| \leq |y|$ to the plan. This ensures the linear order x, z, y, μ.

Planning for cases

This discussion of the Sussman anomaly illustrates how ACCH captures familiar reasoning from a well-known planning domain. By reconstructing planning representations in a validated logical formalism, we can now also apply them to other kinds of reasoning. For example, we saw in section 6.5.1 that this framework is compatible with reasoning for explanation as well as prediction. A related new ability of ACCH is reasoning by case analysis in planning.

Here is the motivation for such reasoning. An agent typically has limited knowledge of the world; in many domains these limits are unavoidable. The agent may be unable to determine facts about the world—perhaps even facts that are very important to its success and well-being. Just because an agent is blind in this way does not mean that it must become unable to derive and carry out successful plans. However, it needs to build its plan bearing in mind the different ways things might turn out compatible with its information. Such reasoning is not possible in SNLP or UCPOP, but the validated case analysis in ACCH allows it to capture such reasoning naturally in a simple extension of partial-order planning data structures.

An example of this reasoning, couched in a domain that underscores its importance, can be found in the famous "bomb in the toilet" problem. This scenario is defined by the rules R and observation O of Figure 6.4. A bomb is hidden in one of two packages. We must make a plan to disable this bomb (achieve ¬*enabled*) with some dunking, assuming that dunking any bomb disables it.

The solution to this planning problem is an acceptable argument D with end-sequent:

$$R^T, O^T, [\mathrm{N}]^i\, \mathbf{h}\mathit{dunkA}, [\mathrm{N}]^j\, \mathbf{h}\mathit{dunkB} \longrightarrow [\mathrm{N}]^3\, \neg \mathit{enabled}$$

The structure of D is as follows. The lowest inference is a cut, to consider separately the case where *hidingPlaceA* is true and that where ¬*hidingPlaceA* is true. Either case begins by introducing a path μ of length 3, where we show $\neg \mathit{enabled}^{\mu}$. In the first case, rule (1a) establishes *bombA* after step 1. Instantiating the occurrence of *dunkA* to a path x of length i and including step 1, $\mathbf{h}dunkA^x$ establishes [N] [H] $\neg enabled^x$. Assuming $x < \mu$, inertial instantiation then establishes $\neg enabled^{\mu}$. In the second case, rule (2a) establishes [H] *bombB* at step 1; inertial instantiation propagates this to the occurrence of *dunkB*; that achieves the goal by rule (4)—with analogous constraints.

No constraints need be imposed to show that this argument is acceptable. The only potential attackers are the use of rule (0) to occlude inertial propagation for [N] [H] $\neg enabled^x$ or [N] [H] $\neg enabled^x$. These attacks are ruled out by the constraint that the dunkings follow the hiding.

6.6 Summary

In this chapter, we have described planning as the task of finding and maintaining reasons for (and against) possible courses of action. Such reasons are needed for single agents to guide their deliberation, to update or extend their plans, or to arrive at collaborative plans with other agents. These reasons are naturally captured using direct systems of defeasible inference; moreover, since they guide deliberation, they are most naturally organized around the key points in time where an agent can initiate a deliberative action.

Formalizing these reasons calls for a delicate balance. On the one hand, for robust and general reasoning, we need a rich inventory of logical operators, including negation and disjunction, with their usual meanings (as in explanation closure or logic programming theories of action). On the other hand, we require a computationally attractive formalism for making and checking assumptions (like the use of negation-as-failure in event calculus planning). We strike this balance by developing reasons in a logical calculus with the usual proof rules but evaluating these reasons using a domain-specific regime for arbitrating between conflicting inferences. The resulting system, ACCH, builds reasons in concrete steps in the style of SNLP or UCPOP, but generalizes to draw sound and complete conclusions—including explanations—from incomplete descriptions of the state of the world.

ACCH ties together a number of threads of research into a clean package. But it lacks an important piece of expressivity. The "bomb in the toilet" problem hints at this limitation. The reasons provided by ACCH guide future deliberation, but they only take into account the information the agent has in the present state. A more flexible plan would allow the agent to choose among possible actions in a future state on the basis of its future information. Despite its "incompleteness", such a plan would continue to provide a useful guide to deliberation by limiting the actions that the agent needs to consider or the properties of the world the agent needs to assess to decide among them. We can't begin to derive such guiding reasons until we have a formal characterization of the changing knowledge the agent has and the knowledge it needs to choose. We now turn to the problem of developing such a characterization.

7
Action and Knowledge

In this chapter, we continue to explore the view of planning as ABDUCTION introduced in section 6.2. According to this view, a plan is (or at least comes with) a logical demonstration that a desired goal will be achieved, assuming the agent follows a specified course of action. To build a plan is simply to prove the goal, abductively assuming the occurrence of appropriate actions as necessary. As we saw in sections 6.3 and 6.4, this framework allows special-purpose planning algorithms, as in [McAllister and Rosenblitt, 1991; Penberthy and Weld, 1992], to be faithfully reconstructed and then extended to richer kinds of action using frameworks such as the event calculus (see e.g. [Shanahan, 1997]), explanation closure (see e.g. [Ferguson, 1995]), and, of course, defeasible argumentation.

In the abductive planning frameworks we considered in Chapter 6, and indeed most implemented planners, the plan shows that the agent can now commit to a specified sequence of actions that will achieve the goal. But a rational agent need not make all its decisions immediately. It can just as well defer choices of future actions to later steps of deliberation. Plans can and should guide these later steps of deliberation, but only if they anticipate the NEW reasons to act afforded by the agent's increased future information. This chapter, based on [Stone, 1998], describes an abductive planning model that does so.

Traditional theories of action and knowledge [Moore, 1985a; Morgenstern, 1987; Davis, 1994] suggest that searching for plans becomes vastly more complicated when information about knowledge is taken into account. The problem is that these theories are based on NAMING plans, using object-level terms that must be specified in advance without reference to the agent's knowledge. This introduces two new search problems, both of which turn out to be artificial.

The first problem is that actions must be described indirectly in the plan. For example, suppose an agent plans to look up Bill in the phone book, then call him. From the agent's point of view, when it makes the call, it will just dial some number n. But since the value of n is settled in a future situation, n cannot be included in a term that specifies the plan fully in advance. Instead, the plan must include a characterization that indirectly describes this action, like *dialing Bill's phone number*. This means that even after the right action is found, a planner still has to search to find an independent description strong enough to show that the action achieves its intended effects.

The second problem is that planners must reason about FOLLOWING THE PLAN, not simply about acting in the world. Not every description corresponds to an action or plan that the agent can carry out: the description might appeal to a fact that the agent will not know. To avoid this, plan reasoning must map out the control structure of the plan in advance and compare the knowledge required by that control structure and the knowledge the agent can expect to have, at each step of execution.

One approach is to avoid these problems using heavy limitations on the syntax and semantics of parameterized actions [Levesque, 1996; Goldman and Boddy, 1996; Golden and Weld, 1996]. This chapter takes a very different approach. We simply add the idea of CHOICE directly into the characterization of achieving a goal. Any future situation offers the agent a number of concrete actions to take. To choose one of these, an agent simply consults its knowledge of these actions to find a good one. Thus, in our basic formulation, a plan is a demonstration that a goal state will follow a series of such feasible choices.

This definition allows plans to be constructed in which each choice is represented as it will be made. This is even true for the new, parametric actions that become available with more information. This account thus dispenses with object-level descriptions of actions and reasoning about following plans; instead, the account parameterizes actions using local Skolem constants, corresponding to the run-time variables of implemented planners. At the same time, the proof itself specifies how choices depend on one another. For example, conditional plans are realized as proofs that use case analysis to reason separately about alternative states of knowledge for the agent. Thus, proof search allows the control structure of the plan to be derived incrementally—in a way that mirrors the introduction and exploration of branches of alternative executions in implemented planners.

The structure of this chapter is as follows. Section 7.1 describes a new way to account for an agent's future information and deliberation explicitly in the logical representation of a plan. Section 7.2 examines the logical underpinnings of the new account. It first motivates the principles of inference that the account should respect and then outlines a model theory and inference techniques that realize these principles of inference. In contrast to previous approaches, we separate the ontology of knowledge and time, yielding models that might seem awkward but in fact admit more straightforward search.

Section 7.3 describes how the reasoning formalism presented in sections 7.1–7.2 yields an abductive planning framework; in this framework, the dependence of actions and reason on an agent's information is realized as a simple constraint on the possible form of assumptions about action occurrences in a planning argument. We conclude this chapter in section 7.4 with illustrations of the formalism at work.

7.1 Choice and Future Reasons to Act

In this section, we introduce a characterization of reasons to act that explicitly refers to the successive stages of information in which an agent deliberates and chooses its future actions. This characterization can be formulated in intuitive language, as follows. A reason to choose a particular next action consists of a demonstration that this choice is the first step in a sequence of steps of deliberation and action—where the agent knows at each state what action to do next, and does it—which allows the agent to achieve its goals, thanks to a specified set of causal connections.

7.1.1 Choice

We derive and formalize our characterization by a running example, the "bomb in the toilet" problem, described earlier in section 6.5. It goes as follows: Given that one of two packages is a bomb, and that an agent *R* can defuse a bomb-package by dunking it, how can *R* defuse the bomb? The solution is for *R* to successively dunk both packages (two actions); the one-action plan in which *R* dunks whatever package is the bomb is not a solution, because *R* cannot choose to carry it out.

In the purely temporal theory of Chapter 6, a plan is a formal demonstration, constructed according to some theory of events and their consequences, that a sequence of actions will achieve some goal. We continue to assume that this demonstration takes the form of a deduction $\mathcal{D}$ with conclusion:

$$T, I, P \longrightarrow G \tag{7.1}$$

Again, this notation indicates that in the deduction $\mathcal{D}$, the formulas in *T*, *I* and *P* are used to derive the formula *G*. (The deductive approach matches previous work on knowledge and action, and suggests an explanation-closure approach to reasoning about inertia, as in [Reiter, 1991; Scherl and Levesque, 1993] for example.) *G* is a logical statement that some goal or goals hold at various points in the future. *T* is a theory describing the causal effects of actions in the domain. *I* describes the initial conditions for the planning problem (and perhaps further available information about future conditions and events). *P* records assumptions describing the occurrence and interactions of actions in the plan. In this framework, the basic problem of building a plan is to find an appropriate set of actions in *P* by abductively assuming premises, given the specification of *T*, *I* and *G*.

(7.1) provides a model in which the agent makes a SINGLE CHOICE OF ACTION, and evaluates the consequences of that choice of action in the different possibilities compatible with what it knows. Given a choice of *P*, we can make assumptions for the sake of argument; for example we can consider the different cases for which package is the bomb. However, assumptions in *P* are made once-and-for-all and cannot depend on what is assumed for the sake of argument; thus, *P* cannot name *whatever package is the bomb*.

A logic of knowledge can make the idea of choice explicit. We treat a single-choice, single-step plan as a proof of

$$[\textsc{k}]T, [\textsc{k}]I \longrightarrow \exists a[\textsc{k}]([\textsc{k}]\mathbf{h}a \supset [\textsc{n}][\textsc{k}]G) \tag{7.2}$$

($[\textsc{k}]p$ represents that the agent knows p; $[\textsc{n}]p$, that p is true after one step of time; $\mathbf{h}a$ means that a is the next event to happen.) The goal formula says that the agent knows, of some concrete action *a*, that if *a* patently occurs, then as a result *G* will patently hold—in philosophical shorthand: the agent KNOWS WHAT WILL ACHIEVE *G* [Hintikka, 1971]. The assumed theory *T* and facts *I* are now explicitly represented as part of the agent's knowledge.

(7.2) matches (7.1) because of the wide scope of $\exists a$ with respect to the operator $[\textsc{k}]$.

Our discussions of the modularity of modal logic in Chapters 2 and 3 give us the intuitions we need to understand the correspondence. In the consequent of (7.2), the quantifier $\exists a$ invites us first to choose an action in the real world, at root scope. Then the modal operator $[\textsc{k}]$ creates a new modular subgoal for proof. Because of the modularity of this goal, and because of the parallel modularity of the formulas that can be used to prove the goal, no ambiguities introduced in this modular subproof can affect the choice of a. Thus, like P in (7.1), a in (7.2) must specify a concrete action that cannot depend on assumptions in the argument assessing a's known result.

Moore's definition of ability to act [Moore, 1985a] also works by giving a quantifier wide scope over a modal operator. However, Moore's definition also includes a requirement that an agent must knowingly select its concrete action under a given abstract description, d. Moore's condition can be reformulated in our notation for comparison:

$$[\textsc{k}]T, [\textsc{k}]I \longrightarrow \exists a[\textsc{k}](a = d \wedge ([\textsc{k}]\mathbf{h}a \supset [\textsc{n}][\textsc{k}]G)) \qquad (7.3)$$

The equation $a = d$ greatly increases the complexity of building a proof, by introducing cumbersome reasoning about known equalities between terms. Since we have grounded (7.2) in (7.1), we discover that it is not essential to derive an abstract description d, and then perform the equational reasoning to show $a = d$; naming the action abstractly does not help analyze an agent's ability to choose an appropriate action from among its concrete options.

7.1.2 Dependent Choice

The definition in (7.2) allows us to specify not only ambiguities in the state of the world but also ambiguities in the information that the agent might have. If what the agent knows is specified partially, proofs will permit an agent's chosen action to depend on its knowledge. By supporting correct reasoning about these dependent choices, proofs already allow us to describe conditional plans and plans with parameterized actions. There is thus no need for explicit, object-level constructs describing the structure of complex plans. Moreover, we continue to avoid the explicit abstract description of actions and plans using terms in the language. This allows us to maintain a very simple definition of achieving a goal by a sequence of choices.

To describe our representation of dependent choices, we return to the "bomb in the toilet" scenario. If we suppose that the agent knows which package is the bomb, we can conclude that the agent knows enough to defuse the bomb. The agent can dunk the package it knows is the bomb.

Formally, this inference might play out in one of two ways. We can add the condition $[\textsc{k}]b1 \vee [\textsc{k}]b2$ to say that the agent knows whether package one is the bomb or whether package two is. Using case analysis, we can prove

$$[\textsc{k}]T, [\textsc{k}]I, [\textsc{k}]b1 \vee [\textsc{k}]b2 \longrightarrow \exists a[\textsc{k}]([\textsc{k}]\mathbf{h}a \supset [\textsc{n}][\textsc{k}]G)$$

If the agent knows package one is the bomb, it can conclude that dunking package one will defuse the bomb; otherwise, it must know that package two is the bomb and be able to conclude that dunking it will defuse the bomb.

This proof instructs the agent to make a CONDITIONAL choice of action, depending on what it knows. To see that this proof implicitly represents a conditional choice, imagine how the agent might use the proof directly to select an action while executing a plan. According to our specification, the agent will have one of two facts as part of its concrete knowledge: either $[\text{K}]b1$ or $[\text{K}]b2$. The proof maps out the reasoning that shows what to do in either case. Thus, the agent need only match its concrete knowledge against the cases in the proof to find which applies, then extract the appropriate component. For practical execution, we might want to use such analysis in advance, to recover an explicit conditional from a proof. Nevertheless, for efficient search, we must represent dependent choices implicitly rather than explicitly. Case analysis can be performed incrementally in proof search, so it is straightforward to derive the conditions for performing different actions piece-by-piece, as needed. Moreover, logical case analysis always interacts correctly with scope of quantifiers, so there is no possibility of proposing a conditional expression that could not form the basis of the agent's choice.

The other alternative is to add the condition $\exists x[\text{K}]bomb(x)$, to say that the agent knows what the bomb is. Then we prove that the agent has a plan by picking a witness c that the agent knows is a bomb and showing that the agent knows dunking c defuses the bomb. We can regard this proof as instructing the agent to make a PARAMETERIZED choice of action, depending on what it knows.

Again, as with an abstract, symbolic description of a parameterized action, this proof has enough information for the agent to choose a successful action. If the partial specification of its knowledge is correct, its concrete knowledge includes a fact $[\text{K}]bomb(x)$ for some object x. The proof spells out what to do with that value x: use it in place of the arbitrary witness c that the proof assumed. Doing so allows the agent to derive from the proof a concrete reason for a specific action. Again, by comparison with an explicit description, we see that the logical treatment, in terms of scope, naturally guarantees that only information the agent has can affect its choices. There could be no possibility of proposing a described action whose referent the agent did not know.

7.1.3 Sequenced Choice

We can call these arguments indirect assessments of an agent's plan. Indirect assessments allow an agent to determine the options available to itself in the future. Here is an example. Suppose we equip an agent R with a bomb-detector in the initial bomb-in-the-toilet scenario. R can describe what would hold after it used the bomb detector in the indefinite way just outlined: R would know which package is the bomb. Therefore, in the next step, R could choose to defuse the bomb. Thus, R already knows that in two steps of deliberation and action (choosing first to detect and then to dunk) the bomb will be defused.

This argument gives R an indirect reason to use the bomb detector now. The proofs we accept as plans must have a staged structure to reflect this staged introduction of future reasons to act. We should represent *R*'s goal thus:

$$\exists a[\textsc{k}]([\textsc{k}]\mathbf{h}a \supset [\textsc{n}]\exists a'[\textsc{k}]([\textsc{k}]\mathbf{h}a' \supset [\textsc{n}][\textsc{k}]G))$$

This fits *R*'s argument. *R* first chooses *a* based on what *R* knows now. *R*'s choice of *a* must enable *R* to choose a good action a' in the next step, based on what *R* knows then. There, *R* will choose a' by reasoning that a' brings about *G*. In all, *G* is nested under three [K] operators. Each inserts a boundary corresponding to a new stage of deliberation as *R* assesses its progress toward the goal. Each may be preceded by an existential quantifier for any action selected at that stage.

We can generalize this to longer plans using a recursive definition. At each step, we identify an action to do next based on information then available, and assume this action occurs; we then make sure that any remaining actions will be identified when needed, until the goal is finally achieved. We use $can(G,n)$ to denote the condition whose proof constitutes a plan to achieve the goal *G* in *n* further steps of action; $can(G,n)$ is defined inductively:

$$can(G,0) \equiv [\textsc{k}]G$$
$$can(G,n+1) \equiv \exists a_n[\textsc{k}]([\textsc{k}]\mathbf{h}a_n \supset [\textsc{n}]can(G,n))$$

This recursive definition directly reflects the staged process by which successive actions are selected and taken.

In describing the knowledge an agent needs to follow a plan *p*, [Davis, 1994] uses a similar staged definition. Simplifying somewhat, and adapting the notation of (7.3), the agent satisfies *can*(*G*,*p*) to follow *p* and achieve *G*:

$$\exists a_n[\textsc{k}](a_n = next(p) \wedge ([\textsc{k}]\mathbf{h}a_n \supset [\textsc{n}]can(G, rest(p))))$$

As with Moore, this presupposes an overall abstract description of the course of action being carried out and appeals to complicated reasoning to determine the *next* action to match that course of action. We have seen that we can give a logical analysis of what an agent can choose to do without separately constructing or reasoning about such a description of a plan.

7.2 Logical Foundations

In the model of section 7.1, a (*k*-step) plan is a proof of

$$[\textsc{k}]T, [\textsc{k}]I \longrightarrow can(G,k)$$

This section describes the underlying logic in which such proofs are to be constructed. In 7.2.1, we look at the idealizations of knowledge involved in the planning task. In planning, inferences about knowledge represent not an empirical description of an agent's mental state but a specification in advance of the mental actions needed for carrying out the plan. If the planning agent can perform these actions, the corresponding axioms should be available;

we thus argue for a strong model of an agent's knowledge in planning.

In 7.2.2, we provide a model-theory for a monotonic language that satisfies these axioms. It is as easy as section 7.1 suggested: we interpret temporal operators by reified translation to a first-order language and introduce an S4 logic of knowledge with increasing domains on top of this extended first-order language. As with the temporal ontology of Chapter 6, this modal ontology for knowledge can be understood informally as describing a state qualitatively, in terms of the transitions that take you there. And, as with the temporal ontology of Chapter 6, we can expect to apply results for modal proof theory like those outlined in Chapter 2 and derived in Chapters 3 and 4. We contrast this model theory with some prior approaches in 7.2.3.

7.2.1 The axioms of knowledge and time

In anticipating future reasons to act, it is appropriate to use an idealization of the information on which future deliberation will be founded. This idealization involves these four second-order principles: deduction; memory; positive introspection; and veridicality. If $[\text{K}]A$ represents the condition that the agent knows a formula A, and $[\text{N}]A$ represents the condition that A is true after the next action takes place, then these conditions can be formalized as follows.

$[\text{K}]A \wedge [\text{K}](A \supset B) \supset [\text{K}]B$	DEDUCTION
$[\text{K}][\text{N}]A \supset [\text{N}][\text{K}]A$	MEMORY
$[\text{K}]A \supset [\text{K}][\text{K}]A$	POSITIVE INTROSPECTION
$[\text{K}]A \supset A$	VERIDICALITY

These conditions are sometimes viewed as controversial, because it is clear that no agent with finite resources could satisfy them in their full generality in all cases. However, the way these principles are used in planning defuses such objections. Only particular instantiations of these principles need be used in a plan; those instantiations that are used completely specify the mental actions that need to be performed in carrying out the plan. Since a bounded agent can keep up with the limited idealizations required in any particular plan, it makes sense to leave open the full space of idealizations in searching for a plan. That's what adopting these second-order principles does.

Consider the deduction axiom. No finite agent can exhibit perfect deduction for all theories at all times. However, any particular instance of deduction represents a correct inference that the agent could draw if it devoted its resources to it. A plan that makes use of a particular instance of the deduction axiom can be seen as a plan to make the particular needed deduction at the proper point in executing the plan. This is likely to be well within the capacity of a bounded agent, especially if the agent devised the plan in the first place. At the same time, representing knowledge explicitly and reasoning about it using the deduction axiom ensures that all inferences that an agent needs to make are included in its plans. This will actually help a bounded agent, by allowing it to know in advance (to some extent) interactions it need not think about in executing the plan.

The memory axiom works similarly. Finite agents must forget. Still, we want an agent to be able to recognize that a particular fact now known will be needed again in the future, and to take the necessary mental steps to ensure that that fact therefore is not forgotten. Particular assumptions of memory used in a plan-proof identify such facts in advance so that this process can take place. At the same time, a plan also includes facts for which memory is not required; a bounded agent can use that information to know in advance (to some extent) what it can safely forget.

Positive introspection plays a role in hypothetical reasoning about knowledge in the present not unlike the role memory plays in reasoning about knowledge in the future. It is used in a plan to show that an agent will know A under certain assumptions by showing that the agent already does know A. So again the particular inferences licensed by the axiom are not unreasonable for a real agent to make or plan to make—even though the recursive form of the axiom suggests that it requires the agent to know an infinite collection of facts.

Veridicality, finally, is the condition that gives this characterization of planning teeth. By veridicality, an agent's knowledge that it knows what plan to follow to achieve a goal entails that the agent will realize its goal if it follows the plan. Conversely, in adopting the plan, the agent commits to a set of instances of veridicality that it depends on for success. As with the other axioms, formalizing these commitments and recording them can guide the agent in keeping up the fiction that its reasoning is ideal. This is compatible with the inevitability that a real agent will be mistaken about some of what it thinks it knows.

Many logics of knowledge also adopt the principle of NEGATIVE INTROSPECTION: $\neg[\text{K}]A \supset [\text{K}]\neg[\text{K}]A$. Moore refrains from adopting it, claiming it is FALSE: an agent that believes A erroneously seems intuitively to lack negative introspection about failing to know A. I don't know how seriously to take that argument, given the idealizations about error we have already made. But I too will omit this axiom, partly because of the considerations of modularity observed in section 2.1.3, and partly just because negative introspection does not seem to buy very much in planning. If you might not know, the thing to do is find out, regardless of whether you actually know you don't know.

7.2.2 A model for the axioms

Because we accept the deduction axiom, we can build a MODAL LOGIC of knowledge and time to satisfy these principles. (Recall from the discussion of Hilbert systems for modal proof in 2.3.1 that all modal logics respect the deduction axiom.) By making [K] an S4 modal operator—constraining its underlying accessibility relation to be reflexive and transitive—we account for positive introspection and veridicality.

This leaves only memory to account for. In other approaches, memory represents a special stipulation on the model (see 7.2.3). Here, however, we propose to exploit the close connection between modal operators and quantifiers to reduce memory to the ordinary principles governing quantifiers under increasing domains.

Recall (from 2.2.2) the content of the increasing domain constraint: if an object e exists at world w and Rww', then e exists at world w', but the converse is not necessarily so. This makes (74a) a strictly stronger statement than (74b).

(74) a $[\text{K}]\forall x P(x)$
b $\forall x [\text{K}] P(x)$

Both statements are true at w only when some set S of entities all share property P at accessible worlds w'. The difference is that for (74a), S is $D(w')$ (the entities that exist at w') whereas for (74b), S is $D(w)$. (74a) is stronger because $D(w) \subseteq D(w')$.

Now, we already have cause to view the temporal operators [N] and [H] as special kinds of quantifiers: universal quantifiers ranging over the action transitions between temporal states. Suppose then that we treat temporal states and action transitions as ordinary first-order entities in a modal model subject to increasing domains. (That means each world will specify a full and distinct branching structure of alternative futures.) Then the relationship of the sentences (75) in the memory axiom will mirror the relationship of the sentences (74).

(75) a $[\text{K}][\text{N}]P$
b $[\text{N}][\text{K}]P$

That is, on this assumption, memory—like deduction, veridicality and positive introspection—will fall out of the model.

There is a certain paradox to this approach. The model postulates a single timeless epistemic accessibility relation between worlds, yet nevertheless accounts for an agent's memory and the possibility for growth in an agent's information. How can this work?

We can best see by considering the following simple example. Our hero knows initially that the next action to affect the world will be taken by his arch nemesis and will be to produce a penny, turned either to heads up or to tails up. Our hero doesn't know which will occur, nor does he know whether his nemesis will hide his action or not. In fact, his nemesis cannot hide the action. Thus, after the action is taken our hero will know what his nemesis did.

This scenario can be represented using a the model shown in Figure 7.1. Each dot in the picture represents a state in a possible world; a common state, labeled 0, is the initial state in each possible world. Arrows between dots indicate a transition relationship between one state and another by some action; the labels of arrows indicate the correspondences between transitions across worlds. In this case, we can read the action that the adversary performs off the label (*h* or *t*) that indicates whether the penny is heads up or tails up the resulting situation; each result is also labeled with a proposition *p* indicating that the coin in question is a penny.

Each tree in the model represents a possible world; open dots represent parts of the real world—the entire leftmost tree represents the alternative possible courses of reality. The two rightmost trees represent epistemic alternatives to the real world. The nested box structure is a reminder that these two worlds have each other as their only alternatives; you cannot return to the real world once you have left it. As dictated by the increasing domain constraint, the accessibility functions *a* and *b* reappear in each of these two worlds.

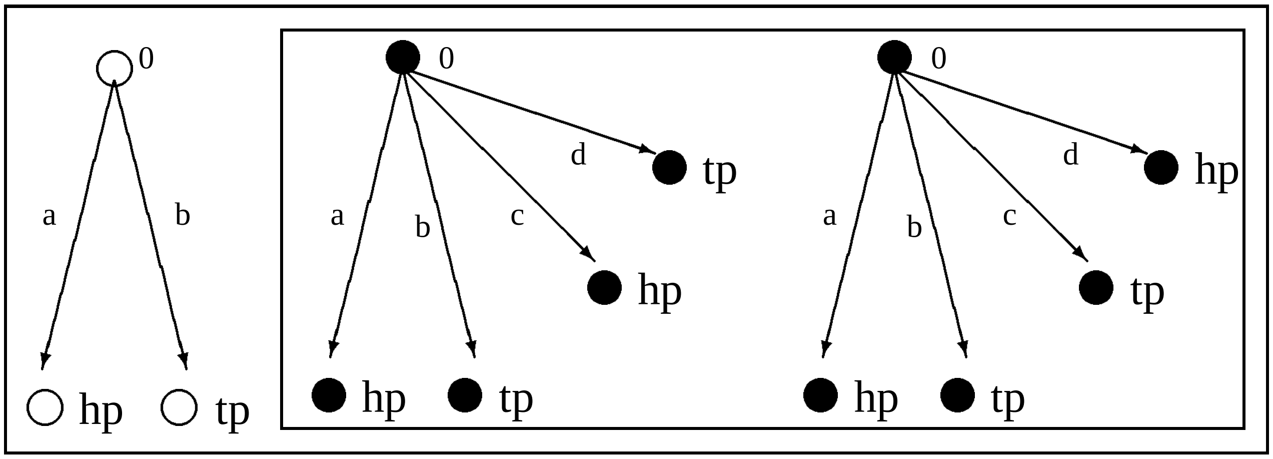


Figure 7.1: Model of knowledge and action using reified temporal transitions

Moreover, as made possible by the increasing domain constraint, two new accessibility functions, *c* and *d*, are introduced.

Suppose we start at point 0 in the real world. To determine that the agent will know that his nemesis shows heads if his nemesis does show heads, we evaluate the formula $[\text{N}](h \supset [\text{K}]h)$. First we obtain all the worlds reachable by local temporal accessibility, 0; *a* and 0; *b*. We narrow to 0; *a*, since *h* is true there, and consider its epistemic alternatives—the states found along the path 0; *a* in all three worlds. Since *h* is true in all of them, the overall formula is true.

On the other hand, to determine that the agent does not yet know this, we evaluate the formula $[\text{K}][\text{N}](h \supset [\text{K}]h)$. We find the epistemic alternatives, and consider the possible temporal continuations in all of them: *a* and *b* applied to 0 in all three worlds, AS WELL AS *c* and *d* applied to 0 in the two new worlds. Again, we eliminate the *t* worlds, leaving the states reached by 0; *a* in all three worlds (where as we have seen our hero knows *h*) AS WELL AS the states reached by 0; *c* in the second world and 0; *d* in the third. The epistemic alternatives to these worlds include the states 0; *c* in the third world and 0; *d* in the second world. In these worlds, *h* is false. So the whole sentence is false.

We can adapt the definitions of models and truth from section 2.2.2 to describe this ontology in a variety of ways. The simplest is just to describe a modal logic whose first-order component is sorted so that temporal formulas like $[\text{N}]p$ (and if appropriate $[\text{H}]p$) are interpreted as abbreviations for sorted first-order quantifiers.[1]

We briefly sketch the technical development of this approach now. We need separate

[1]An alternative would be to have multi-modal logic where modalities had different dimensions. The advantage of this alternative would be to allow a first-order domain to be assigned to each state of each world separately, so that the existence of entities could be restricted by time as well as by possibility. The disadvantage of the alternative is that the corresponding mathematics is less well-explored.

sorts for ordinary entities **e**, temporal states **t**, and temporal transitions (indexed when appropriate by temporal operators) $\mathbf{a}_i$. This sorting requires some extensions to the language—we want a signature that includes a binary function symbol **;** combining a state and a transitions to yield a state and that assigns sorts to constant symbols and sorts for the arguments of relation symbols. Relation symbols will take all first-order arguments except for a distinguished temporal state argument (which we continue to indicate by superscript terms).

The model tuple $M = \langle W, R, D, F \rangle$ is interpreted with slightly more structure. D is broken down into a set of functions $D_{\mathbf{e}}$, $D_{\mathbf{t}}$ and $D_{\mathbf{a}_i}$; each associates worlds with nonempty increasing sets of objects of the appropriate sort. We extend F to include a function interpreting the elements of transition type as rigid designators for transitions between states. The argument of this function will be a pair in $\cup\{D_{\mathbf{t}}(w) | w \in W\} \times \cup\{D_{\mathbf{a}_i}(w) | w \in W; \textit{i an operator}\ \}$; its result will be an element of $\cup\{D_{\mathbf{t}}(w) | w \in W\}$. We also extend F so that the interpretation of constant and relation symbols respects the sorts specified by the signature. The semantics and truth-definition is otherwise unchanged.

We can define a translation to eliminate temporal operators as in [Ohlbach, 1991]. At subterms, translation of p depends on a term t of type **t** and is denoted $T(p, t)$. If $\Box$ is a temporal operator then $T(\Box p, t)$ is $\forall x T(p, t; x)$ where x is a new variable not occurring in p or t. Other logical and modal operators are unaffected; e.g., $T([\textsc{k}]p, t) \equiv [\textsc{k}]T(p, t)$. Atomic relations make explicit their dependence on t: R becomes R^t. Using some a variable or constant 0 for the initial state, we translate a formula p as $T(p, 0)$.

Now, since [K] is an ordinary modal operator, reasoning with [K] is correctly described for example by the proof rules given in Figure 3.10 in section 3.4.1:

$$\frac{\Gamma, [\textsc{k}]A^{t,\mu}, A^{t,\mu x} \longrightarrow \Delta}{\Gamma, [\textsc{k}]A^{t,\mu} \longrightarrow \Delta}\ [\textsc{k}] \rightarrow (\text{LV } x{:}\textsc{k})$$

$$\frac{\Gamma \longrightarrow [\textsc{k}]A^{t,\mu}, A^{t,\mu f(X)}, \Delta}{\Gamma \longrightarrow [\textsc{k}]A^{t,\mu}, \Delta}\ \rightarrow [\textsc{k}]\ (\text{SK } f(X){:}\textsc{k})$$

(Recall that LV and SK abbreviate the steps taken to rewrite the sequent to introduce a new logic variable or Skolem term.) The dependence on t anticipates that we can use proof rules for temporal operators that combine the translation with ordinary first-order proof rules. Of course, these look much like what we already have—for example:

$$\frac{\Gamma, [\textsc{n}]A^{t,\mu}, A^{t,\mu t} \longrightarrow \Delta}{\Sigma \triangleright \Gamma, [\textsc{n}]A^{t,\mu} \longrightarrow \Delta}\ [\textsc{n}] \rightarrow (\text{LV } t{:}\textsc{n}(\mu))$$

$$\frac{\Gamma \longrightarrow [\textsc{n}]A^{t,\mu}, A^{t,\mu f(X)}, \Delta}{\Gamma \longrightarrow [\textsc{n}]A^{t,\mu}, \Delta}\ \rightarrow [\textsc{n}]\ (\text{SK } f(X){:}\textsc{n}(\mu))$$

The main difference is that we must now keep track of both the sort of a temporal transition and the possible world where it first is known to exist.

Because the temporal model theory and proof theory is no different from that outlined in Chapter 6, we can continue to regard this ontology as describing time in a qualitative way that promises to fits well with linguistic analysis. Since we also reason about [K] in terms of paths of accessibility, we have a parallel ontology describing the evolution of the agent's information in a qualitative way. If anything, this ontology is even more natural for information.

In [Stone, 1998], I go a step further and illustrate the double application of prefix theorem-proving techniques for modal logic to this system. Both temporal and epistemic modal operators are replaced by explicit quantifiers. After translation, modal reasoning follows directly from the classical case.

The only trick in the translation is the handling of the increasing domains of individuals across possible worlds. We use compound terms *t@w* where *w* names the world at which the referent for *t* is first defined—the DOMAIN of *t*. This representation in fact mirrors the constraint treatment of domain restrictions described in section 4.3. As arguments of relations involving individuals and states, any constant symbol or free variable *t* is immediately translated as $t@w_0$, where w_0 represents the real world. Bound variables are assigned an appropriate domain as quantifiers are translated. This translation depends on whether the quantifier is instantiated or Skolemized. At a world *w*, Skolemized quantifiers introduce a term *t@w* that cannot be assumed to exist before *w*. So at Skolemized quantifiers we replace (argument occurrences of) the old bound variable *x* by a new term *x@w*. Other quantifiers are instantiated; at a world *w* they may take on any value *t@u*, provided that this *t* exists at *w* (given it first exists at *u*). To meet the proviso, we must find a path $u; v = w$, showing that *u* is a prefix of *w*, written $u \leq w$. So at instantiated quantifiers we replace the old bound variable *x* by a new term *x@u* where *u* is a new restricted variable over worlds.

For completeness, the translation that we have just outlined informally is given precisely in Figure 7.2. The translation turns a modal formula *A* into a classical formula $[A]^{s_0@w_0,w_0,\pm}$ depending on the initial state s_0, the real world w_0 and whether *A* is assumed (+) or to be proved (-). It looks more complicated than it is: the translation just annotates terms and quantifiers with explicit domains and annotates atomic relations with an explicit world and state of evaluation. The translation requires us to reason with the equations

$$E \equiv w; (\alpha \star \beta) = w; \alpha; \beta, \quad w; 1 = w$$

Using this translation, a plan is just a classical deduction with the following conclusion:

$$E, [[\text{K}]T, [\text{K}]I]^{s_0@w_0,w_0,+} \longrightarrow [can(G,k)]^{s_0@w_0,w_0,-}$$

7.2.3 Some other representations of knowledge and action

Prior integrated models of time and knowledge have not made available the streamlined principles described in section 7.2.2. This section looks at two notable examples. We begin with [Moore, 1985a]; Moore explicitly sets out to devise models that respect the axiomatization of knowledge and time from section 7.2.1. Moore develops a logic where

$$
\begin{aligned}
&[R(t_1,\ldots,t_k)]^{d,w,\pm} \equiv R(t_1,\ldots,t_k,d,w)\\
&[A \wedge B]^{d,w,\pm} \equiv [A]^{d,w,\pm} \wedge [B]^{d,w,\pm}\\
&[\neg A]^{d,w,\pm} \equiv \neg[A]^{d,w,\mp}\\
&[[\textsc{k}]A]^{d,w,\pm} \equiv \forall\alpha[A]^{d,w;\alpha,\pm}\\
&[[\textsc{n}]A]^{s@v,w,+} \equiv \forall\tau\forall u(u \le w \supset [A]^{s;\tau@u,w,+})\\
&[[\textsc{n}]A]^{s@v,w,-} \equiv \forall\tau[A]^{s;\tau@w,w,-}\\
&[\forall xA]^{d,w,+} \equiv \forall e\forall u(u \le w \supset [A[e@u/x]]^{d,w,+})\\
&[\forall xA]^{d,w,-} \equiv \forall e[A[e@w/x]]^{d,w,-}
\end{aligned}
$$

Figure 7.2: Translation $[\cdot]^{\cdot,\cdot,\pm}$ to classical logic

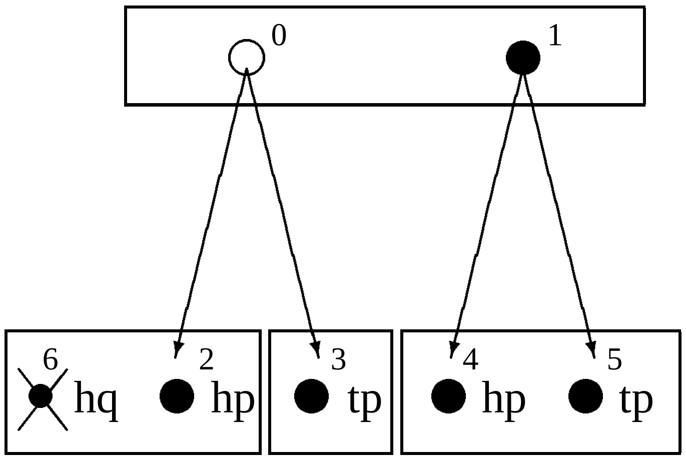


Figure 7.3: Moore's model.

worlds represent snapshots of a possible history and are subject to different accessibility relations representing knowledge and time. In this approach knowledge and time interact on the same level of a model. At each moment in time, knowledge is described by a reflexive transitive accessibility relation to alternative time-slices.

How the world w changes when action e is performed is represented by a new world $res(e,w)$, as in the situation calculus [McCarthy and Hayes, 1969]. This is reconciled with modal terminology by requirement that there be a unique e such that $\mathbf{h}e$ is true at each world, and that there be at most one world w' for each world w and each action e such that $w' = f(w)$ for any $f \in AF_n$ and $\mathbf{h}e$ is true at w'.

In this setup, our simple scenario will be represented in terms of a model such as that shown in Figure 7.3.

Moore's basic ontology is consistent with forgetting. World 6 in Figure 7.3 shows this. In world 0, our hero thinks that either world 0 or world 1 is possible; following a step of time, we discover that our hero thinks that any of worlds 2, 3, 4 or 5 could come next. In all of these worlds p is true, so the agent initially knows that p will be true. On the other hand, if the nemesis transforms reality to world 2, our hero will have one accessible world, namely world 2 itself, where p is true, but another accessible world, namely world 6, where q is true instead (the coin is a quarter, say). So our hero could forget p.

Therefore, Moore imposes an additional principle to guarantee memory. It is a commutativity condition: anywhere you get by transitioning on time and then knowledge, you could have gotten to by transitioning on knowledge and then time. Logically, for any world w' with $K(a, res(e, w), w')$, there must be a world w'' with $K(a, w, w'')$ and $w' = res(e, w'')$. World 6 doesn't meet this condition, and gets crossed out. Moore's condition takes the model further from a free structure, and requires complicated equational reasoning to capture [Gasquet, 1993].

With the deletion of world 6, the model of Figure 7.3 captures the scenario fully. In particular, note that world 2 is a world where our hero knows that h holds, world 3 is a world where our hero knows that t holds, and worlds 4 and 5 are worlds where our hero does not know which holds. Since worlds 2 and 3 are the real possibilities, our hero will know which action his nemesis takes. But since worlds 4 and 5 are epistemic possibilities for our hero, he doesn't know that he will know.

Fagin, Halpern, Moses and Vardi present an alternative modal logic of knowledge and time [Fagin *et al.*, 1995]. It applies many of the same principles as Moore's logic, but works by combining knowledge with linear time rather than the situation calculus. Formally, each model or SYSTEM is divided into a series of RUNS, each of which consists of a series of states ordered on a linear time line. In their notation, (w, t) represents the state at time t in run w. Because time is linear, each run specifies a unique and complete evolution of history. However, many several runs may have compatible histories up to some later moment of divergence. This gives the formalism a notion of branching: the choices of an agent or nondeterministic events are modeled in terms ignorance about which run the system is on.

As in Moore's proposal, an agent's knowledge is represented by a time-varying accessibility relation between points. Two points are related by $K(a, (w, t), (w', t'))$ when agent a cannot distinguish between its internal information in time t of run w and in time t' of run w'. On this view, $K(a, \Leftrightarrow, \Leftrightarrow)$ must be an equivalence relation. Thought of in terms of accessibility functions, this requires that each $f \in AF_k$ be a constant function.

The picture this gives for our scenario is given in Figure 7.4, which follows the same conventions as Figure 7.3. Now both world 0 and world 1 are represented by open circles, because for all anyone knows before the nemesis acts, either run could represent the real world. Given this characterization of reality, states 4 and 5 are the true possible next states—just as two worlds are true possible next states in Figure 7.3. In each of these two possible next states, our hero can see what his nemesis has done. In Figure 7.4, as in Figure 7.3, there are additional futures that our hero thinks possible but which are not actually possible. Those are the ones where his nemesis hides the coin from him. So Figure 7.4 correctly reflects a case where the agent learns something unexpected, and does

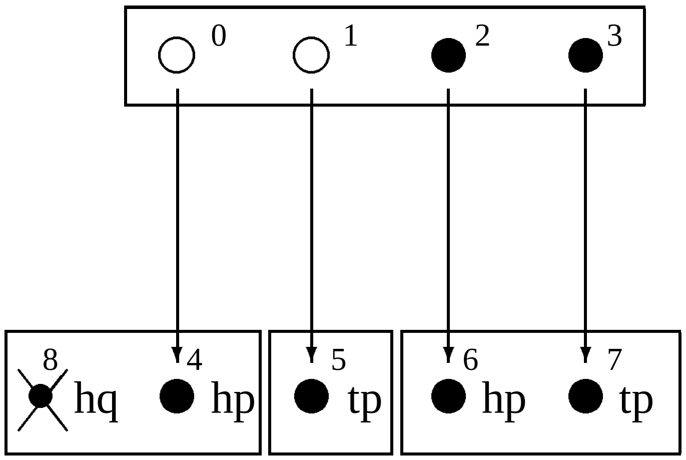


Figure 7.4: Fagin, Halpern, Moses and Vardi's model

so in much the same way that Figure 7.3 does.

If Fagin, Halpern, Moses and Vardi's model has a similar structure to Moore's model, it has similar drawbacks for our purposes. In particular, it happily accommodates cases where the agents forget, as illustrated by the presence in Figure 7.4 by state 8. Once again, the only option is to impose an additional, and possibly expensive, commutativity constraint to implement perfect memory.

7.3 A New Abductive Presentation of Planning

In this section, we recast this DEDUCTIVE approach to planning as an ABDUCTIVE problem, in which action occurrences are assumed as needed. The recursive definition of *can* already outlines a sequence of assumptions with a common content: at a particular stage of action and deliberation, the agent selects and performs an appropriate action. More precisely, proving $can(G,k)$ introduces, in lock-step with the introduction of temporal transitions, action assumptions that all take the form

$$\forall\beta.\mathbf{h}(\underbrace{e_i@u_i}_{\textit{real action}}, \underbrace{t_i@m_i}_{\textit{real state}}, \underbrace{m_i;a_i;\beta}_{\textit{known world}}) \tag{7.4}$$

Here $u_i \leq m_i$. m_i represents the agent's view of what is REAL when the action is chosen; $t_i@m_i$ is a real state introduced at m_i by a goal quantifier; and e_i is some real action. Meanwhile, because the assumption is applicable at any world $m_i;a_i;\beta$, it can contribute only to what is KNOWN at m_i. By encoding the evaluation of a CONCRETE action for KNOWN effects, this form concisely distills the notion of choice.

Because the assumptions are indistinguishable, we can make them as needed. Thus, we can offer a purely abductive presentation of the proof search problem for building a plan to achieve G after k steps of deliberation and action. We simply prove $([\text{K}][\text{N}])^k[\text{K}]G$, making action assumptions of the form in (7.4) where necessary. This abductive approach eliminates ambiguities in proof search: there is only one way to assume a new action but there are many ways to match a sequence of uninstantiated actions (assumed independently). It also helps strengthen the connection between this theory and implemented planners: implemented planners also add actions one by one, as necessary.

The derivation of this abductive characterization in part depends on how formulas are represented using Skolem terms, logic variables and unification, according to a particular theorem-proving technique. Not surprisingly, we prefer to follow a LOGIC PROGRAMMING proof search strategy, as characterized in Chapter 3. In logic programming proofs, the first actions taken are always to decompose goals; this matches the strategies of special-purpose planning algorithms, and moreover allows modal operators in planning goals to be processed by introducing fresh constants independent of actions. (On the use of constrained constants for Skolem terms more generally, see [Bibel, 1982].) As we prove $can(G, k)$ by this strategy, the formula is decomposed level-by-level. Level n requires us to decompose a goal translated from $\exists a[\text{K}]([\text{K}]\mathbf{h}a \supset [\text{N}]\ldots)$; the implication introduces a new assumption of the form in (7.4).

This explains the source of the assumptions in (7.4). But is abduction sufficiently restricted? Suppose an assumption instantiates the sequence t_i to a particular time $s_j@w_j$. Then e_i first exists at some world $u_i \leq w_j$. If the assumption contributes to the ultimate proof of G, moreover, a_i can only equal α_j. Thus the instantiated assumption could just as well have been explicitly made in decomposing the formula $can(G, k)$.

7.4 Key Examples

The last three sections have outlined a logical approach to planning based on an analysis of an agent's ability to choose. To plan, an agent describes its goal in a form that indicates that the goal can be reached after a sequence of steps not only of action (corresponding to temporal updates) but also of deliberation and choice (corresponding to modal transitions). At each step, an agent must choose a concrete next action based on its known properties; this restriction corresponds directly to constraints which distinguish the possible worlds where actions and times are defined from the worlds where action assumptions can be used.

7.4.1 Run-time variables and knowledge preconditions

In this section, we first show how our framework allows the results of one action to provide parameters for later actions—unlike [Levesque, 1996; Goldman and Boddy, 1996]. We return to the example of the bomb-in-the-toilet, formalized as in figure 7.5. The agent knows there is a bomb, knows it has a detector and knows it can dunk. The agent must defuse something in two steps. (In figure 7.5, $[\text{H}]p$ abbreviates that p is true indefinitely; explanation closure axioms are omitted as this proof goes through without them.) This

$$\begin{array}{ll}
1 & [\textsc{k}]\exists b.\mathit{bomb}(b) \\
2 & \exists a[\textsc{k}][\textsc{h}]\forall x(\mathit{bomb}(x) \wedge \mathbf{h}a \supset [\textsc{n}][\textsc{k}]\mathit{bomb}(x)) \\
3 & [\textsc{k}]\forall x \exists d[\textsc{k}][\textsc{h}](\mathit{bomb}(x) \wedge \mathbf{h}d \supset [\textsc{n}]\mathit{defused}(x))
\end{array}$$

Figure 7.5: Bomb-in-the-toilet with detector.

translates into the goal:

$$\mathit{defused}(b(\alpha)@w_0;\alpha,\ s_0;\tau;\tau'@w_0;\alpha;\alpha',\ w_0;\alpha;\alpha';\beta)$$

$b(\alpha)$ Skolemizes b; other first-order terms will be Skolemized similarly; here, $b(\alpha)$ could also be found by unification during proof search. The proof requires two actions:

$$\begin{array}{l}
\forall\beta.\mathbf{h}(a@w_0,\ s_0@w_0,\ w_0;\alpha;\beta) \\
\forall\beta.\mathbf{h}(d(\alpha, b(\alpha))@w_0;\alpha,\ s_0;\tau@w_0;\alpha,\ w_0;\alpha;\alpha';\beta)
\end{array}$$

The first assumption considers the result of the immediate real action a of using the detector, assessed in worlds $w_0;\alpha$ compatible with what we know initially. The second assumption considers the result of dunking the hypothetical object $b(\alpha)$—a real action in world $w_0;\alpha$—in worlds $w_0;\alpha;\alpha'$ compatible with what we know after one step. The reader can readily flesh out this proof along the outlines suggested earlier, after first computing the translation and Skolemization of the clauses of figure 7.5.

Note how we represent the choice of dunking $b(\alpha)$ directly. The agent will learn from using the detector that $b(\alpha)$ is a bomb; the proof relies on the fact that the agent has this knowledge. Encoding this into the proof is enough—for, as we saw earlier, this is enough information to allow the agent later to extract what to do, by matching its concrete knowledge against the abstract knowledge the proof supposes. So we need not describe the dunking, as do [Moore, 1985a; Morgenstern, 1987; Davis, 1994].

A comparison is instructive with a representative implemented planning language with similar plans, SADL [Golden and Weld, 1996]. In SADL plans for such examples, sensing introduces a RUN-TIME VARIABLE storing the observed value; these run-time variables can be then appear as arguments to later actions. The terminology suggests some inherent departure from logic. On the contrary, such variables correspond exactly to Skolem terms like $b(\alpha)$, naming new abstract entities that exist only at remote worlds. Recognizing this logical status for run-time variables explains why such variables are treated existentially and why—in view of the "knowledge precondition" that only concrete actions can be chosen—they can serve as parameters only to actions chosen in future deliberation. At the same time, it confirms our contention that an agent's internal representation of its future actions need not be a timeless, abstract description of that action.

$$
\begin{array}{ll}
4 & [\textsc{k}]\forall s v \exists o [\textsc{k}][\textsc{h}](closed(s) \land combo(s,v) \land \mathbf{h}o \supset \\
 & \qquad\qquad\qquad [\textsc{n}][\textsc{k}]open(s)) \land \\
 & \qquad [\textsc{k}][\textsc{h}](closed(s) \land \neg combo(s,v) \land \mathbf{h}o \supset \\
 & \qquad\qquad\qquad [\textsc{n}][\textsc{k}]closed(s)) \\
5 & [\textsc{k}](closed(d_0)) \\
6 & [\textsc{k}]([\textsc{h}]combo(d_0,n_0) \lor [\textsc{h}]\neg combo(d_0,n_0)) \\
7 & [\textsc{k}]\forall s[\textsc{h}]\neg(open(s) \land closed(s))
\end{array}
$$

Figure 7.6: Safe problem

7.4.2 Knowledge preconditions for actions and plans

The next example shows, as [Golden and Weld, 1996] argue, that actions may have to be performed with different knowledge in different circumstances. However, it also shows that this variation is a natural component of a logical approach to planning—not an argument against it.

Consider a domain with a safe. If the agent dials the combination to the safe, the safe patently opens; if the agent dials something else, the safe patently remains closed. The safe starts out closed, has a constant combination, and can't be open and closed at once. We formalize the situation in figure 7.6. Suppose the agent wants to open the safe d_0 in one step—in our theory, to build this plan requires proving:

$$open(d_0@w_0,\ s_0;\tau@w_0;\alpha,\ w_0;\alpha;\alpha')$$

This cannot be proved abductively unless the agent knows the combination to the safe. Let's add that assumption:

$$\exists v[\textsc{k}]combo(d_0,v)$$

Then we can assume the real action of dialing this combination for assessment according to what the agent knows:

$$\forall\beta.\mathbf{h}(o(1,d_0,v)@w_0,\ s_0@w_0,\ w_0;\alpha;\beta)$$

This allows us to complete the plan straightforwardly, by applying the first rule of clause 4.

By comparison, suppose the agent merely wants to determine in one step whether the combination to the safe is n_0 or not. This goal is represented in modal logic as $[\textsc{k}]combo(d_0,n_0) \lor [\textsc{k}]\neg combo(d_0,n_0)$. It translates into a planning problem to prove

$$
\begin{array}{l}
combo(d_0@w_0,\ n_0@w_0,\ s_0;\tau@w_0;\alpha,\ w_0;\alpha;\alpha';\beta) \lor \\
\neg combo(d_0@w_0,\ n_0@w_0,\ s_0;\tau@w_0;\alpha,\ w_0;\alpha;\alpha';\gamma)
\end{array}
$$

This is a weaker statement than the goal for the previous problem—for starters, it contains disjunction. We can prove this abductively—without assuming knowledge of what the combination of the safe is—by considering the known consequences of attempting to dial n_0:

$$\forall\beta.\mathbf{h}(o(1,d_0,n_0)@w_0,\ s_0@w_0,\ w_0;\alpha;\beta)$$

The proof is interesting. At world $w_0;\alpha$, we perform the case analysis licensed by clause 6, and consider separately what happens when the combination is right and when it's wrong. In either case, we prove our goal by ANTICIPATING our ability to EXPLAIN the observed results of dialing the combination. For example, suppose we have the combination right. Then by clause 4, we can derive the safe's opening as part of our final knowledge:

$$\mathit{open}(d_0@w_0,\ s_0;\tau@w_0;\alpha,\ w_0;\alpha;\alpha';\beta)$$

Now, we perform the case analysis licensed by clause 6 afresh at world $w_0;\alpha;\alpha';\beta$: we have seen the door open, what will we then think about the combination? If the combo is right, we know the combo is right; that takes care of the first case. But the combo cannot be wrong: we could apply clause 4 to conclude

$$\mathit{closed}(d_0@w_0,\ s_0;\tau@w_0;\alpha,\ w_0;\alpha;\alpha';\beta)$$

(As known facts, the occurrence of the action and the initial closure of the safe remain available.) But this is impossible by clause 7. By parallel reasoning, we can show that if the combo is in fact wrong, after acting we will know it.

7.5 Summary

In this chapter, we have added an ontology of qualitative transitions in information to the ontology of qualitative temporal transitions presented in Chapter 6. This extension allows an agent to calculate reasons for present action that take into account the new choices the agent will be able to make in later states. The development of this extension has been dense. However, a summary will emphasize the continuity of the planning framework developed here with that developed in Chapter 6.

In both cases, plans are proofs that record the causal connections by which an agent can choose actions to fulfill goals. In both cases, the proofs introduce the goal as a conclusion to be derived after a series of steps. Now those steps are recorded both in a sequence of action transitions that describe how physical actions contribute to the goal and in a sequence of epistemic transitions that describe how steps of deliberation and choice contribute to the goal. Each goal and conclusion in the proof is labeled by sequences of both types. These parallel kinds of entities provide an abstract ontology for planning.

In both cases, the proofs begin by allowing a domain theory and a set of initial conditions to be used as assumptions. Then they allow action statements of a special form to be assumed to represent what the agent will do as part of the plan. As in Chapter 6, the form of these assumptions guarantees that a concrete action is always chosen, in that the parameters of the action can and will be settled by deliberation before the action takes place. We check this by enforcing the temporal and first-order entities in the action to take concrete values—as represented by the earliest world where they are known to exist—in the step of deliberation when the action is considered—represented by a more distant world (containing additional new, abstract objects).

In Chapter 6, we were able to provide a detailed reconstruction of the data structures, operation, and expressive power of implemented planners. By contrast, the planning framework developed in this chapter has substantial gaps. We can regard the abductive scheme of section 7.3 as providing a way to construct arguments about the consequences of the action and deliberation of a single agent. To match the presentation of Chapter 6, we should also provide and validate a system for reasoning correctly but defeasibly using these arguments. This remains an important topic for future work.

8
Modular Specifications of Knowledge

The modal account of planning with information-gathering presented in Chapter 7 depends on the modularity of modal logic, as follows. The account describes how an agent's actions are determined by successive stages of deliberation. Modularity is used to isolate these stages of deliberation. By introducing each stage using a strongly modular modal operator, the account ensures that any deliberation makes recourse only to the information that the agent has at that stage, and that any deliberation can only impact future decisions.

We saw in Chapter 3 that modal operators are useful not only to correctly describe a domain, but also to encode knowledge about the structure of proofs and proof-search in a domain. Viewing the treatment of planning from Chapter 7 as an example of a modal description of a domain, then, suggests that this description could be augmented so as to specify the structure of and search for plans more precisely. This chapter presents some speculations about how this might be done, drawing on an analogy between modular specifications and the HIERARCHICAL specifications of actions and plans commonly used in practical planners.

The planning formalisms of Chapters 6 and 7 allow only one general way to achieve a goal, namely, to orchestrate appropriate circumstances for the execution of multiple actions by sequencing actions together. Because of this uniform structure, these plans can be described as FLAT. In hierarchical planning, the task of achieving a goal can be accomplished by another strategy: the task can be recursively decomposed into a series of smaller problems at a lower level of abstraction. Representative hierarchical planners include NOAH [Sacerdoti, 1975], SIPE [Wilkins, 1988] and O-PLAN [Currie and Tate, 1991].

Hierarchical planning offers improvements in search along at least two dimensions. First, the set of actions required to perform an action can be specified directly in a hierarchical decomposition. This can avoid the search a flat planner takes to identify these actions by backward chaining through their effects and preconditions. Second, in many cases, hierarchical planners are designed to solve subproblems independently. Because the interactions among subproblems are limited, hierarchical planners can avoid much of the search for possible threats and threat-resolutions that a flat planner would carry out to reconcile sets of actions with one another. These advantages for search make hierarchical planning very attractive.

In this chapter, I establish the link between hierarchical planning and modular specifications by looking closely at the knowledge that must underlie a hierarchical plan. Recall from Chapter 6 that any plan should reference the causal generalizations that allow the plan to work; this causal structure is required to allow plans to play a broad role in an agent's deliberation. For hierarchical plans, this knowledge must in part establish causal relations between actions at different levels in the hierarchy. Pollack was the first to point this out, in her work on the knowledge attributed to agents in virtue of their plans [Pollack, 1990]. A modular structure is imposed on plans in deriving a proof in modal logic that these requirements are satisfied. Because such plans exploit the modularity of modal logic in a new way, this formalization of hierarchical plans can result in exponential reductions in the size of the proofs that represent plans and the complexity of finding them.

By connecting with logical modularity, this reconstruction of hierarchical planning retains both of the improvements (introduced above) that hierarchical planning can provide. In contrast, previous proposals for hierarchical extensions of flat planners [Yang, 1990; Young *et al.*, 1994; Barrett and Weld, 1994] accommodate only the first potential improvement in search. These proposals essentially build ordinary flat plans, and treat hierarchical decompositions as restrictions on the search space that constrain which actions may appear in the plan, and which order actions may appear in. These restrictions in search do not allow these planners to attack subproblems independently, however. All pairs of actions must still be checked for possible interactions. This compromises these planners' ability to capture the full intended effect or the full empirical advantages of hierarchical action decompositions.

8.1 Motivating Hierarchy

HIERARCHICAL plans are plans in which the actions to be taken are specified at various LEVELS. Clark provides a handy example ladder of actions at different levels ([Clark, 1996], p. 147):

(76)

Level	Action in progress
5	*A* is getting an "up" elevator to come
4	*A* is calling an "up" elevator
3	*A* is activating the "up" button
2	*A* is depressing the "up" button
1	*A* is pressing the right index finger against the "up" button

In this ladder, terms at higher levels factor out or presuppose causal connections that lower-level terms encode explicitly. In this sense, actions at different levels are distinguished by the abstraction of the causal reasoning which governs them. In Clark's example, the basic level 1, with *pressing*, describes the action only at the level of the primitive forces applied in the action. The next level, with *depressing*, packages the event at a coarser granularity, which explicitly signals the kinematic change that the force is intended to bring about. Level 3, with *activating*, describes an intended change to the internal state of the button and abstracts away from the movement that triggers that state change. Level 4 abstracts further, describing the action in terms of its intended effect on the internal state of the elevator system as a whole. Finally, level 5 describes the action as it affects *A*'s intended interaction with the elevator system.

Organizing actions in hierarchical levels as in (76) seems to help people plan and carry out more complicated tasks. Writing computer programs is a good example. The programming practice of modular design involves breaking down large tasks into successively more fine-grained ones, in just this way. The ultimate result of this process is a complete program with a modular, hierarchical structure. The hierarchical structure allows lower levels of the implementation to be encapsulated, providing an interface to some needed functionality but hiding how that functionality is actually achieved. This encapsulation enables programmers to construct programs more easily, and to make those programs easier to debug, maintain and extend.

In programming—and in deriving and maintaining large plans, generally—a crucial contribution of hierarchical organization is to constrain the interactions among actions that are possible in a large plan. Let us review why interactions cause problems, and consider why we might want to eliminate the possibility of interactions by strengthening the specification of a planning domain.

The formal accounts of planning presented in Chapters 6 and 7 involve two difficult cases for managing interactions among actions. Threats to plans, as arise when one action in the plan interferes with the desired effects of another, are one computationally problematic case. As described in Chapter 6, detecting threats requires an exhaustive examination of the possible consequences of each action in the plan. And resolving threats requires adopting further constraints on the plan in one of several ways that must be explored nondeterministically. Meanwhile, choices in plans, as arise when an agent's partial information about the world will be resolved in one of several different ways, also induce explosive interactions among actions. In conditional plans, interactions can induce dependencies among choices. In general, any choice can influence all subsequent ones; thus, when there are multiple choices to make, the size of the plan will grow exponentially in the number of choices.

The search problem associated with these interactions can be eliminated by specifying the independence of actions and decisions in advance as part of the domain. In particular, organizing actions into levels signals such independence by abstracting and encapsulating key features of causal reasoning. In assessing whether sets of actions can be executed in a consistent sequence, a planner can avoid searching for potential threats when the actions have been specified as independent. (At the same time, of course, it avoids search to resolve threats too.) In a hierarchical action domain, for example, actions might be specified as independent in this way when they contribute to orthogonal subtasks of a decomposed high-level action; thus, the planner can avoid considering threats that cross subtasks. Similarly, in planning for partial information, the planner need not propose decisions that take all previous decisions into account—it can restrict itself to independent decisions that are made on the basis of restricted or encapsulated information. In a hierarchical action domain, decisions that arise in one subtask may specified as independent from decisions that arise in other subtasks. Then the flow of control of the plan can be collapsed from a single large and nested branching process to a sequence of small decisions.

When considering ways such as these to streamline a planner's response to interactions within a plan, it is vital for any independence in the domain to be explicitly specified. For, the mere possibility of interaction among actions—whether as threats or as dependencies in choice—directly impacts the search space for planning. A search space that takes account of potential interactions remains large even when actions happen not to interact. So, an organization of actions into levels can signal independence by abstracting and encapsulating key features of causal reasoning—it can thereby lead to more tractable planning problems—but this organization can do so only if this independence is an explicit part of its content.

We close this section by considering how these arguments play out in a concrete example. Consider subway travel in New York City. Certain lines run two kinds of trains in the same direction on parallel tracks. On one side of the platform, local trains come; on the other side of the platform, express trains come. For many destinations, you can take either kind, and the sensible thing to do is just to take the first train that arrives. Suppose your journey might involve a number of changes in lines, and you wish to codify your policy for getting on trains into a plan. (As a tourist in a big city, one may, at the very least, derive some considerable peace of mind from a conclusive demonstration that one will reach one's destination.) Without knowledge of what choices interact, this plan must be constructed by first supposing that the express train comes first and planning the remainder of the journey in that case, and then supposing that the local train comes first and planning the remainder of the journey in that case. As the plan extends from stop to stop, it grows exponentially in size.

In this example, however, it is obvious that choices cannot interact in the way this plan allows for. The decision criteria you adopt in selecting the local or express train at a particular station are independent of the choices of trains you took on previous legs of the journey.

Informally, a hierarchical specification of actions in the subway gives rise to a planning strategy that respects this independence. Suppose the domain describes two levels of action. One level describes the high-level actions of getting from one station to another. Elaborating a plan for a journey at this level means outlining a sequence of legs leading you through a sequence of station stops and line changes from your current position to your final destination. The other level describes the lower-level actions and decisions involved in finding an appropriate platform in each station, and in deciding on, boarding, and disembarking from an appropriate train. Each leg of the journey can be expanded into a conditional plan that sequences these lower-level actions together. In particular, part of each plan could include a decision to take the first train—local or express—that arrives at the platform while you wait there. Once the plans for the lower-level goals are included at the appropriate spots in the higher-level plan, one arrives at a detailed prescription for the whole journey. But because this plan has been built so as to respect the independence of local decisions, the size of this plan grows linearly with the number of legs in the journey.

The most salient feature of this hierarchical reasoning is its use of DECOMPOSITION to transform the problem of planning a journey into a series of problems of planning smaller legs. More generally, in decomposition, actions are fleshed out level by level and the domain rules specify directly what should be the SUBACTIONS of each action—the actions at the immediately lower level that contribute the it. Specification and reasoning about decomposition is characteristic of hierarchical planning; the research of [Young *et al.*, 1994; Barrett and Weld, 1994] presents alternative methods of implementing decomposition in formal, partial-order planners. In [Young *et al.*, 1994], operators can directly specify a decomposition whose effect is to add additional actions, preconditions and effects to the plan under construction. Apart from a requirement to realize the intended higher-level action, these actions, preconditions and effects are subject to ordinary plan reasoning (including search for threats). In [Barrett and Weld, 1994], ordinary action specifications are complemented by a context-free specification of possible plans against which any possible plan must be parsed. Parsing the plan corresponds to recognizing how decomposition could in fact sequence the actions in the plan. Again, actions at lower levels in different subtasks may interact arbitrarily.

In the subway plan, decomposition can show a system that its plan for a journey across town should consist of a series of plans for legs, where each leg describes a trip between stations on a single line. But this in itself does not distinguish a large tree-like plan in which decisions about trains interact from the smaller linear plan with independent decisions. In fact, the fully branching plan will still specify how the journey is to be broken up—conditionally—into a series of legs. Thus, any execution trace of this branching plan will consist of a sequence of actions that can be grouped according to the decomposition.

What allows the subway plan to be kept concise is that the decomposition not only specifies subtasks to perform but specifies how those subtasks do and do not interact. The planner must know not only that a subway journey can be accomplished by a sequence of legs of travel along a single line but also that each leg of travel stands on its own. Choices made on that leg of the journey are independent of others, and need neither be anticipated nor remembered. Since other accounts of hierarchical planning, like [Young *et al.*, 1994; Barrett and Weld, 1994], do not model this independence, one might expect this feature of specifications to be very difficult to represent. Fortunately, we shall find otherwise: once the relationship of actions at different levels in a hierarchical specification—as an agent must rely on it in planning—is spelled out precisely, it naturally allows planning problems at lower levels to be attacked by independent, modular search.

8.2 Hierarchy and Causality

Recall from Chapter 6 that complete plans include the causal connections by which actions achieve their intended effects. In particular, a plan which contains a ladder of actions must specify the causal connections among actions at different levels. So the first step in describing a hierarchical plan is to describe this causal connection. We provide such a specification in this section.

The relation between actions in a hierarchy is known as GENERATION [Goldman, 1970]—

or more properly conditional generation. Pretheoretically, generation names the relation that holds between an agent *a*'s low-level action *l* and *a*'s high-level action *h* when the statement *a did h by doing l* holds. This kind of statement indicates not only that *a* did *l* and that *a* also did *h*; it indicates that there is some systematic, general regularity in the world that leads from occurrences of *l* to occurrences of *h*. The notion of CONDITIONAL GENERATION makes the grounds or conditions for this regularity explicit.

Formally, Goldman characterizes a conditional law-like relationship between *l* and *h* in part using the formula in (77).

(77) $\exists C . C \wedge [\textsc{h}](C \wedge \mathbf{h}l \supset \mathbf{h}h)$

(Here we have reformulated Goldman's requirement to use the notation introduced in Chapters 6 and 7.) The formula identifies a proposition *C* which holds at the current time and about which we can make a further general statement. At any other time (here for economy restricted to the future using the operator [H]), if those conditions were to hold, and *l* were to occur (**h***l*), then *h* would also occur (**h***h*).

The condition in (77), which we abbreviate by $\textsc{cgen}(h, l, s)$, must be further refined in order to describe a real law-like relationship between *l* and *h*. In fact, if *l* and *h* both occur, (77) can be verified just by taking *C* to be the implication $\mathbf{h}l \supset \mathbf{h}h$. So we have not ruled out a purely accidental relationship between *l* and *h*. Apparently the choice of *C* must be restricted so that *C* includes only the kind of information about the present situation on which law-like generalizations can be based.

Goldman accomplishes this restriction by defining conditional generation in terms of (77) and a pair of further components. These are: first, if the agent had not done *l*, the agent would not have done *h*; and second, if *C* had not obtained, then the agent would not have done *h* despite doing *l*. To motivate the first requirement, Goldman considers the case of piano-playing that puts Smith to sleep but awakens Brown. The piano-playing should conditionally generate both results, but awakening Brown should not generate putting Smith to sleep. Formally, it could, if the conditions *C* could specify that the only kind of Brown-awakening that might occur at the relevant time is the piano-playing kind. The first additional requirement patches the gap. If Brown had stayed asleep Smith still could have fallen asleep too. After all, the piano piece could have been a mellower one. We can describe the effect of the first requirement as ensuring that *C* is not so specific as to explain *h* not based on *l* but instead based on inappropriate features of what actually happened.

To motivate the second requirement, Goldman considers the problem of the DIRECTION of conditional generation. There is no bound on how specific we can make *C*. So suppose the conditions *C* could be specified so precisely that, given *C*, the occurrence of *l* was EQUIVALENT to the occurrence of *h*. Then Brown-awakening would generate piano-playing just as surely as piano-playing generated Brown-awakening. The second counterfactual rules this out, because even if this stronger *C* didn't hold (but some weaker regularity did), the agent could have done both *l* and *h*. So this requirement forces *C* to be weaker than the conditions of this extreme example—indeed, it forces *C* to be as weak as possible. Only then will it be true that S could not have done h by doing l WITHOUT C.

If we see Goldman's definitions primarily as ways to constrain C, we are free to accomplish the restriction by other means than the addition of counterfactual clauses. We need only a logical expression that holds of a proposition C when C provides a true characterization of the situation in which l and h take place, in terms of appropriate features that establish a law-like relationship between l and h. In the setting provided by our earlier study of the use of modal logic to structure common-sense specifications, a natural avenue is to rely on modality to supply such an expression: $\Box C$. We regard $\Box$ here as indicating the context or module in which the relationship between l and h is assessed.

To make sense of this idea, let us recall from Chapter 2 the link between modality and the modular or contextual structure of a body of knowledge. Suppose we intend to evaluate a specific query Q in a system for common-sense inference, and want to more directly control the performance of a system and the results it finds. We can accomplish this by restricting the premises which the system can access in evaluating the query. One approach is to divide up the system's knowledge into modules, as in modal logic programming [Miller, 1989; Giordano and Martelli, 1994]; another approach is to divide the knowledge into contexts or microtheories, as in [Guha and Lenat, 1990; Guha, 1991]. With these strategies, we allow the system to use only facts from applicable modules or contexts when proving the query. As suggested in Chapter 2, we can leverage such ideas by viewing the query as a modal, modular query $\Box Q$ and then specifying selected premises in the form $\Box P$, to indicate that they alone can be applied towards this modular query.

To review: Goldman's generation relation requires us to identify a restricted class of information to use in assessing the hierarchical relationship among actions. We need to identify all and only the conditions C that support a law-like relationship between low-level and high-level actions. But to structure any common-sense knowledge base, we already expect to make use of contexts or modules, whose function is to restrict the premises that are available to the system when assessing common-sense relations like causality. This modularity is already organized to group together the factors in a situation that support causal generalizations. So we can factor this structure into our treatment of the generation relation between actions.

To elucidate, it is illustrative to follow CYC's terminology of microtheories and consider the use of microtheories in inference. The general flavor is given by Guha's description of different microtheories that might be used when modeling a device with multiple components [Guha, 1991, p. 117ff.]. We are given a query about the device's behavior. Based on the query, we recursively construct a problem-solving context which provides a specialized microtheory of the device. Constructing the theory involves ignoring or abstracting some components of the device, ignoring and abstracting some behaviors of the components, and considering only causal properties of the components and behaviors that are relevant to the query.

Such a specialized theory naturally brings to bear exactly the regularities and features of the device that impact the query. This theory is identified modally (or contextually) as a set of facts of the form $[\textsc{psc}]A$. Now, suppose that the query describes a possible generation relation between a lower-level action and a higher-level action. We can now formulate this query as simply as $[\textsc{psc}](\mathbf{h}l \supset \mathbf{h}h)$. By assessing the query with respect to this specialized microtheory, it is as though the microtheory supplies precisely the background C for conditional generation. Grounding C in independent causal knowledge gives a way to avoid the overly weak or overly strong conditionalizations for which Goldman introduces his additional counterfactual conditions.

We can abstract away from Guha's notions of constructing a problem-solving context, and restate this idea purely at the level of the modal logic of structured specifications. We describe an agent's knowledge of the world in terms of a series of SOURCES OF INFORMATION, each of which is represented by a modal operator. Each fact that an agent knows comes with a specification of the source or sources of information from which it has been or can be obtained. For example, a fact about the agent's environment that it learned through visual perception would be recorded as a result of this perceptual source. More interestingly, causal laws are partitioned based on the experience, training or institution from which the agent derived its knowledge of those laws. By relating the operators associated with these kinds of information by inclusion axiom schemes, we can specify useful combinations of facts. In particular, for each type of high-level action we can assume a source that combines the various kinds of information needed to relate that high-level action to its low-level realizations. By combining this type of information with information that specifies the events that actually occur, we obtain a complete specification of the causal interactions.

Notate by $[\textsc{k}]_h$ the agent's source of knowledge to be taken into account in determining the relation of action h to lower actions. Then the condition that the agent knows that h is conditionally generated by l in s simplifies to (78)

(78) $[\textsc{k}]_h(\mathbf{h}l \supset \mathbf{h}h)$

Here, instead of existentially quantifying some proposition C—subject, as we have seen, to further conditions—and asserting it of an arbitrary accessible situation, we identify C as the totality of information available in the present by knowledge-source $[\textsc{k}]_h$. Since we thus obtain a restricted characterization of the current situation, there is no need to introduce an arbitrary alternative one. Moreover, we can eliminate the need for Goldman's further restrictions on C by setting up $[\textsc{k}]_h$ in the right way.

To see this, let's return to Goldman's piano example. To determine what generates changes in sleep, we invoke the theory of sleep and the relevant factors about sleep available from the knowledge-source $[\textsc{zzz}]$. In Goldman's example, as the piano-playing is about to begin, this source includes facts such as: that Brown sleeps while Smith is awake; that Brown is disposed to hear piano music as an engaging disturbance while Smith is disposed to hear it as a soothing rhythmic sound; and that their dispositions will lead to the expected changes in wakefulness upon the initiation of piano-playing. Formally, if simply, we might have:

$$\begin{array}{l}
[\textsc{zzz}]\mathit{sleeps}(\mathit{brown}) \\
[\textsc{zzz}]\mathit{wakes}(\mathit{smith}) \\
[\textsc{zzz}][\textsc{h}](\mathbf{h}(\mathit{piano}) \supset \mathbf{h}(\mathit{bugs}(\mathit{brown}))) \\
[\textsc{zzz}][\textsc{h}](\mathbf{h}(\mathit{piano}) \supset \mathbf{h}(\mathit{soothes}(\mathit{brown}))) \\
[\textsc{zzz}][\textsc{h}]\forall x(\mathbf{h}(\mathit{bugs}(x)) \supset [\textsc{n}]\mathit{wakes}(x)) \\
[\textsc{zzz}][\textsc{h}]\forall x(\mathbf{h}(\mathit{soothes}(x)) \supset [\textsc{n}]\mathit{sleeps}(x))
\end{array}$$

It is a logical consequence of this that that the piano wakes Brown up:

$$[\textsc{zzz}](\mathbf{h}(\mathit{piano}) \supset [\textsc{n}]\mathit{wakes}(\mathit{brown}))$$

and that the piano puts Smith to sleep:

$$[\textsc{zzz}](\mathbf{h}(\mathit{piano}) \supset [\textsc{n}]\mathit{sleeps}(\mathit{smith}))$$

(Equivalently, by considering a wider source of information $[\textsc{zzzr}]$ that includes both our theory of sleep and the real event of piano playing that in fact happens here, we derive $[\textsc{zzzr}][\textsc{n}]\mathit{wakes}(\mathit{brown})$ and $[\textsc{zzzr}][\textsc{n}]\mathit{sleeps}(\mathit{smith})$.) Now, the specification itself identifies what the conditions C should be for this judgment of generation, namely $\bigwedge_p[\textsc{zzz}]p$. Thus, as long as the theory of sleep and current data give no role to one person's waking in enabling another's sleep or vice versa (as it will not), the judgments of generation under the conditions thus identified will fall in line with both Goldman's additional tests and with naive judgments.

Why, meanwhile, does the sleeping not generate the piano-playing? Again, what generates piano playing is computed according to a theory of musical performance and relevant data in the current situation, as captured by a knowledge-source $[^\flat_\sharp]$. This source provides that our pianist, John, intends to play a certain piece and has some requisite skill; and that anyone who hit the keys similarly would indeed play the piano. Formally, if simply:

$$\begin{array}{l}
[^\flat_\sharp]\mathit{musical\text{-}intentions}(\mathit{john}) \\
[^\flat_\sharp]\mathit{piano\text{-}skilled}(\mathit{john}) \\
[^\flat_\sharp][\textsc{h}]\forall x(\mathit{musical\text{-}intentions}(x) \wedge \mathit{piano\text{-}skilled}(x) \wedge \\
\qquad \mathbf{h}(\mathit{hits\text{-}keys}(x)) \supset \mathbf{h}(\mathit{plays\text{-}piano}(x)))
\end{array}$$

So it is hitting the keys that generates playing the piano. And as long as the wakefulness of the audience does not figure in $[^\flat_\sharp]$ (as it will not), there can be no counterintuitive judgments of awakening generating piano-playing.

8.3 Knowledge, Hierarchy and Choice

We now have a natural modal formalism for the causal connections between hierarchical levels of actions. We need to integrate this formalism into the framework introduced in Chapter 7 for describing an agent's knowledge and choice. Recall that this framework is based on a recursive definition of an agent's ability to achieve a goal G in some number k of steps of decision and choice:

$$\begin{aligned} &can(G, 0) \equiv [\text{K}]G \\ &can(G, n+1) \equiv \exists a_n[\text{K}]([\text{K}]\mathbf{h}a_n \supset [\text{N}]can(G, n)) \end{aligned}$$

This definition describes a series of sequenced choices of atomic actions. Further knowledge is required when actions in the plan are taken at different levels at different times. [Pollack, 1990] arrives at a characterization of this knowledge from the beliefs people must assume to underlie one another's hierarchical plans, given their contributions to dialogue. Pollack argues that when people adopt a plan for doing h by l, they know that l conditionally generates h. In fact, they typically assume a generating expansion of h all the way down to atomic actions. Of course, it may be useful for an agent to commit during deliberation to a plan that counts as partial under this definition—perhaps as an intention to perform a specific high-level action [Bratman, 1987]. However, Pollack gives examples of dialogues that show that quite detailed planning decompositions are assumed by conversationalists.

Why should agents need to establish knowledge of generation as part of knowing that their plans will work? The different abstractions of causal knowledge involved in generation suggest an answer. Each sequence of high-level actions represents some abstract changes in the world, but obliterates the more concrete changes of lower-level actions. Such abstraction might be designed to suppress irrelevant changes, or to allow the concrete realizations of successive hierarchical actions to be described by different properties and relations whose mutual impact can be left unspecified. To ensure that a hierarchical decomposition will work, an agent must know that the low-level sequence accomplishes the intended abstract effect no matter how these unknown lower-level details are resolved. This is nothing other than the knowledge of generation just presented.

To formalize this recursive requirement of expansion, we want to define a predicate $spec(h)$ which is true when the agent can fully specify how to carry out h. The base case is when the agent knows that h is an atomic action in the agent's repertoire. To handle the recursive case, it is convenient to specialize the definition of generation presented in (78). We assume that each high-level action h is defined in terms of some condition F_h that we plan h for. We will leave any causal connection between h and F_h implicit in the pragmatics of planning (if F_h is already true, why plan h to achieve it?), and define:

$$\mathbf{h}h \equiv [\text{N}]F_h$$

F_h complements the specification of a source of information $[\text{K}]_h$ that describes information to be taken into account for h. Both specifications can be encapsulated in the following clause defining what it takes do flesh out h:

(79) $[\text{K}]complex(h) \wedge [\text{K}]_h \exists n.can(F_h, n) \supset \text{SPEC}(h)$

This just says that the agent knows that h is complex, and has a plan—at the lower level of abstraction governed by the modular theory $[\textsc{k}]_h$—to achieve the result of $h\ F_h$.

The overall definition of *spec* is thus:

$$[\textsc{k}]atomic(a) \lor ([\textsc{k}]complex(a) \land [\textsc{k}]_h \exists n.can(F_h, n))$$

Finally, we revise the definition of *can* to reflect the requirement of specifying the component actions:

$$can(G, 0) \equiv [\textsc{k}]G$$
$$can(G, n+1) \equiv \exists a_n(spec(a_n) \land [\textsc{k}]([\textsc{k}]\mathbf{h}a_n \supset [\textsc{n}]can(G, n)))$$

Inspection of these definitions reveals that they call for hierarchical relationships not only among the actions but also among the causal theories governing them. Each barrier between one level of action to the next lowest one imposes a restriction on available information. Treatments of levels are nested, so the barriers of information accumulate as deeper levels are examined. Lower-level theories must therefore be included in the information that can be taken into account in higher-level theories. The reader should bear this in mind.

8.4 Modality, Modularity and Disjunction

We now have a logical characterization of hierarchical planning. The question to ask is whether this characterization supplies information about interaction that can be operationalized to reduce the size of proofs. This section invokes our earlier study of modal proof theory and logic programming to show that the answer is yes. The key reason is that our new specification of planning invokes strongly modular modal operators in two ways in building plan proofs. We retain the modal operators that segment the stages of deliberation in which an agent makes successive choices at one level in the plan. But we also introduce nested modal operators to implement the definition of conditional generation when expanding actions. This added modular structure breaks up what would otherwise be a single flat sequence of stages of choice into independent subsequences, in just the way suggested above by our informal discussion of planning for the subway. Indeed, the two subsections of this section are organized so as to present that subway example in its full formal detail.

8.4.1 The construction of a conditional plan

We begin by considering the plan for a single leg of the journey. The domain is described in Figure 8.1. Here we have class of situations in which a conditional plan may be necessary to achieve the goal of moving to the next station. We use *exp* to represent the condition that an express train comes first and *loc* to represent the condition that a local one comes. For now, we will build a flat plan; however, the use of the modality $[\textsc{train}]$ to encode a source of the agent's knowledge anticipates use of the specification in the hierarchical setting. Since the plan involves two steps, the planning problem corresponds to the formula

1 $[\text{TRAIN}][\text{H}](\mathit{exp} \lor \mathit{loc})$
2 $[\text{TRAIN}][\text{H}](\mathit{exp} \land \mathbf{h}(\mathit{look}) \supset [\text{N}][\text{TRAIN}]\mathit{exp})$
3 $[\text{TRAIN}][\text{H}](\mathit{loc} \land \mathbf{h}(\mathit{look}) \supset [\text{N}][\text{TRAIN}]\mathit{look})$
4 $[\text{TRAIN}][\text{H}](\mathit{exp} \land \mathbf{h}(\mathit{take\text{-}exp}) \supset [\text{N}][\text{TRAIN}]g)$
5 $[\text{TRAIN}][\text{H}](\mathit{loc} \land \mathbf{h}(\mathit{take\text{-}loc}) \supset [\text{N}][\text{TRAIN}]g)$

Figure 8.1: Domain specification for conditional planning

$\mathit{can}(g, 2)$, abbreviated for flat plans to the following statement:

$$\exists a[\text{K}]([\text{K}]\mathbf{h}a \supset [\text{N}]\exists a'[\text{K}]([\text{K}]\mathbf{h}a' \supset [\text{N}][\text{K}]g)$$

This formalizes that you know what action a to do, such that after doing a you will know what action a' to do, such that doing a' achieves g.

The solution to the problem is a plan to test whether *exp* or *loc* is true using *look*, and then if *exp* is true doing *take-exp* and if *loc* is true doing *take-loc*. The formal proof corresponding to this solution goes as follows (from the perspective of the agent who will execute it). We show there is an a by using witness *look*. Consider only what I know, but assume also that I know I will *look*. I know either *exp* or *loc* is true by (1); consider the cases separately. In the first case, if *exp*, then after I *look* I will know *exp*. Thus there is a b, namely *take-exp* such that I know that *take-exp* achieves g. For, considering only what I know, assuming I know I *take-exp*, I use rule (2) to establish that (I know) I achieve g. In the second case, if *loc*, the same reasoning applies for *take-loc*. ■

The role of knowledge is to force the disjunction between *exp* and *loc* to be treated in the scope in which the test *look* is assumed. If the disjunction is used at root scope to describe the real world, it does not make available the premises on which *look* depends, because those premises must be part of the agent's knowledge.

The analysis of knowledge preconditions uses the modularity of S4 modal logic to link the scope in which alternatives are considered, the scope at which testing actions are assumed, and the scope at which the results of the tests are used to make decisions. The modular structure induced by proving a plan correct does not create a modular plan, however. The results of a test make an ambiguity known only within a scope, but in fact all subsequent actions in the plan are nested within that scope, so arbitrary interactions remain possible. This is precisely where the difference will arise from the encoding of a hierarchical plan.

8.4.2 The construction of a hierarchical plan

To describe a space of hierarchical planning problems, we can combine the rules in Figure 8.1 with those in Figure 8.2. An overall journey is now described as a sequence of transitions of type *go*(*x*,*y*), following links, that get you from *at* your start to *at* your destination. We associate the action $\mathit{go}(\Leftrightarrow, \Leftrightarrow)$—defined as complex by clause 11—with the

6 $[\textsc{k}]at(p_1)$
7 $[\textsc{k}][\textsc{h}](link(p_1, p_2))$
8 $[\textsc{k}][\textsc{h}](link(p_2, p_3))$
9 $[\textsc{k}][\textsc{h}](link(p_3, p_4))$
10 $[\textsc{k}][\textsc{h}]\forall xy(link(x, y) \wedge at(x) \wedge \mathbf{h}(go(x, y)) \supset [\textsc{n}]at(y))$
11 $[\textsc{k}][\textsc{h}]\forall xy(complex(go(x, y)))$

Figure 8.2: Definitions for a Hierarchical Specification

modality [TRAIN], which describes the theory of how $go(\Leftrightarrow, \Leftrightarrow)$ actions are generated, and with the fluent g, which holds after the action succeeds. Thus, each *go* transition is realized by solving a planning problem in terms of the causal theory of lines. Since we know that those plans have length two at worst, we can invoke the planning formula presented above, and resolve to a specialized rule which governs $spec(go(\Leftrightarrow, \Leftrightarrow))$ in this domain:

$$[\textsc{train}][\textsc{h}]\forall xy([\textsc{train}]\exists a[\textsc{k}]([\textsc{k}]\mathbf{h}a \supset [\textsc{n}]\exists a'[\textsc{k}]([\textsc{k}]\mathbf{h}a' \supset [\textsc{n}][\textsc{k}]g)) \\ \supset \textsc{spec}(go(x, y)))$$

Meanwhile, to construct a two step plan to travel to p_3 we obtain the following condition and construct a proof of it:

$$\exists a(spec(a) \wedge [\textsc{k}]([\textsc{k}]\mathbf{h}a \supset \\ [\textsc{n}]\exists a'(spec(a') \wedge [\textsc{k}]([\textsc{k}]\mathbf{h}a' \supset [\textsc{n}][\textsc{k}]at(p_3)))))$$

As with the example of section 8.4.1, there is a unique way to prove this formula given this specification. Not surprisingly, this proof involves instantiating the high-level actions a and a' respectively to $go(p_1, p_2)$ and $go(p_2, p_3)$. It can then be shown directly from the knowledge of *link* facts, the knowledge of the occurrences of the actions, and rule (10) of Figure 8.2 that finally we know $at(p_3)$. Each SPEC fact, meanwhile, can only be derived by the reasoning discussed in section 8.4.1. As before, the disjunction introduces an ambiguity that persists for the remainder of the plan AT THAT LEVEL. However, because the remainder of the higher-level plan is sequenced in the scope of the higher-level action, rather than its decomposition, this low-level ambiguity is limited to the scope of the low-level plan.

8.5 Conclusion

Hierarchical plans have a distinctive content. The relationship between actions at different levels of abstraction is a causal relationship which must be known before a hierarchical plan can be successfully undertaken. The fact that agents must obtain this knowledge explains why hierarchical plans have a more constrained structure than flat plans. This structure may be exploited for efficiency.

The next step for both theoretical and practical development is to adapt the ideas presented here to a more efficient theory of time, using explanation closure [Schubert, 1990; Reiter, 1991] or defeasible reasoning [Lin and Shoham, 1991; Sandewall, 1994b]. Both strategies have yet to be extended to logics of knowledge, so there is a lot of work to do. Still, it seems like this approach already offers a promising first step toward a practical system based on these ideas.

Part III

Modal Logic for NLG

9 Modal Logic and Reasoning in NLG

In Chapters 6, 7 and 8, we investigated how reasoning tasks involving action and knowledge can be formalized in terms of modal logic and modal deduction. The techniques developed in Part I suggest simple, predictable procedures for evaluating these specifications while exploiting the essential modularity of modal reasoning. We thus have an expressive but constrained framework in which to describe both the subject matter and knowledge state of a conversation. This chapter returns to the problem of reasoning in NLG, first introduced in Chapter 1, and shows how this framework can be applied in constructing concise and precise descriptions of actions.

The context for this demonstration is the SPUD system for sentence generation [Stone and Doran, 1996; Stone and Doran, 1997; Stone, 1997a; Stone and Webber, 1998; Bourne, 1998]. SPUD adopts a view of sentence generation as goal-directed activity, like [Appelt, 1985; Dale, 1992] before it. On this view, the task of the generator is to use the words and constructions of the language to design a message that fulfills a set of communicative intentions. SPUD works with two kinds of intentions in particular: intentions to uniquely identify the entity designated by a referring expression, and intentions to establish a proposition as part of the content of the conversation. SPUD fulfills these intentions incrementally, using a grammar in the LTAG formalism [Joshi *et al.*, 1975; Schabes, 1990] to add units of meaning and syntax word-by-word into an incomplete sentence.

This chapter begins by introducing SPUD; section 9.1 describes informally how SPUD works and why it is particularly suited to generating appropriate descriptions of actions under realistic assumptions about input and context. This introduction, adapted from [Stone and Webber, 1998], highlights three tasks that SPUD must solve: evaluating possibilities for reference, assessing the content it has managed to communicate, and determining which of its options best fits the context. Section 9.2 then shows how DIALUP can supply answers to these tasks from specifications of language and context formulated declaratively as modal logic programs. Finally, in section 9.3, we discuss some concrete, detailed examples of the uses of SPUD and DIALUP in generating concise, contextually-appropriate descriptions.

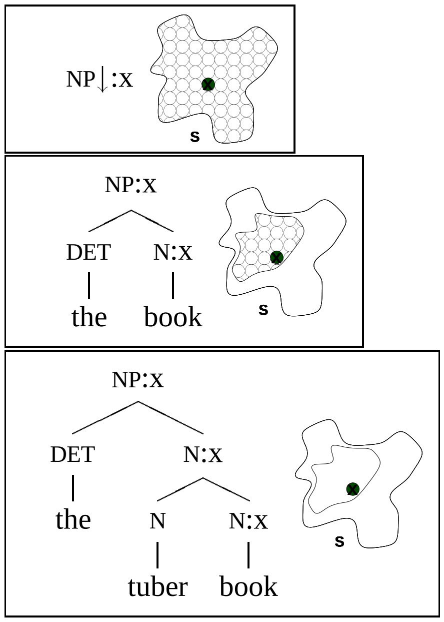
9.1 An Overview of SPUD

An NLG system must satisfy at least three constraints in mapping the content planned for a sentence onto the string of words that realize it [Dale, 1989; Meteer, 1991; Rambow and Korelsky, 1992]. Any fact to be communicated must be fit into an abstract grammatical structure, including lexical items. Any reference to a domain entity must be elaborated into a description that distinguishes the entity from its DISTRACTORS—the salient alternatives to it in context. Finally, a surface form must be found for this conceptual material.

In one architecture for NLG systems that is becoming something of a standard [Reiter and Dale, 1997], these tasks are performed in separate stages. For example, to refer to a uniquely identifiable entity *x* from the common ground, first a set of concepts is identified that together single out *x* from its distractors in context. Only later is the syntactic structure that realizes those concepts derived.

SPUD integrates these processes in generating a description—producing both syntax and semantics simultaneously, in stages, as illustrated in (80).

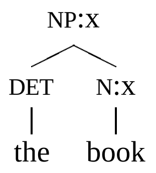
(80) a b c



Each step adds to the representation a lexicalized entry encoded as an elementary tree in Feature-based Lexicalized Tree-Adjoining Grammar (LTAG) [Schabes, 1990]. Each such tree is paired with logical formulae that, by referring to a rich discourse model, characterize the semantic and pragmatic contribution that the element makes to the sentence.

For example, (80a) represents the overall task in planning an NP to refer to entity *x*. In the initial pair, the syntactic component is represented by a tree consisting of a single node NP↓:*x*. As indicated by the ↓ notation, this node is a SUBSTITUTION SITE; this indicates that another tree must be supplied to specify the structure of the NP before the derivation is complete. As indicated by the label or index *x*, this node refers to and supplies information about entity *x*. Meanwhile, the semantic component in the initial pair locates this entity *x* within a set *S* of distractors that has been supplied by the discourse context and which is otherwise unconstrained.

According to the grammar of English, the first step in generating the NP is to choose a head noun which indicates the type of *x*, and a determiner which indicates its INFORMATION STATUS in the sense of [Gundel *et al.*, 1993]. Such a choice is specified by the entry for *the* *book* specified in (81).

(81) a Syntax: 

b Semantics: $book(x)$

c Pragmatics: $unique\text{-}id(x)$

This entry contributes the lexical item *book* to the syntactic tree. (Here and elsewhere we can use LTAG trees that contain multiple lexical items.) Simultaneously, the entry supplies *book* as *x*'s type and *uniquely identifiable* as *x*'s information status. As described in [Stone and Doran, 1997], the concrete choices available to the SPUD generator always consist of entries that contribute syntactic structure, lexical elements, and semantic and pragmatic constraints. The triple in (81) is a representative example.

For conciseness and efficiency, however, SPUD accepts separate lexical and grammatical specifications. An entry like (81) is assembled dynamically from these two kinds of information. In particular, the word *book* is specified as in (82), with a lexeme, semantics, and TREE FAMILY.

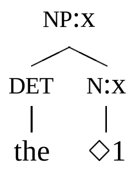
(82) a Lexeme: book

b Semantics: $book(x)$

c Trees: DEF-NP(x), . . .

The tree family specifies a range of syntactic constructions that the word can be used with; included in this range are pragmatic variations like the choice of definite versus indefinite forms for NPs, the choice of unmarked versus marked word-order for VPs, etc. Elements of this tree family include specifications like that given in (83) for the DEF-NP tree used in (81).

(83) a Tree: DEF-NP(x)

b Syntax: 

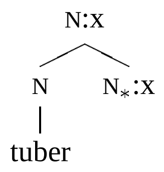
c Pragmatics: $unique\text{-}id(x)$

The notation ◇1 indicates that the (first) lexeme from the corresponding lexical entry should appear at this site as an ANCHOR to the syntactic tree.

Thus, the entry in (81) is straightforwardly obtained by combining (82) with (83). When this entry SUBSTITUTED into (80a), the result is the state shown in (80b). (The operation of substitution identifies the root of one tree, called an INITIAL tree, with a substitution site in another tree.) The tree is expanded to include new syntactic material at the same time as the set of distractors is narrowed to incorporate the new semantic constraint.

Once the head noun of the NP is selected, further steps can incorporate modifiers into the NP as needed. The modifier chosen in (80c) is the entry for *tuber* given in (84).

(84) a Syntax:



b Semantics: *concerns-tubers*(x)

c Pragmatics: (always applicable)

Not surprisingly, this entry adds the semantic constraint that its referent concern tubers at the same time as it adds the lexical and syntactic specification for *tuber* to a sentence. This entry, we assume, has no pragmatic requirements: it is always applicable. (We will suppress such empty requirements in the future.)

In TAG, any node in a tree potentially offers a point at which further modification of a sentence can take place. This modification is accomplished by the syntactic operation of ADJUNCTION. Syntactically, (84) associates *tuber* with an AUXILIARY tree that contains a distinguished leaf, the FOOT node. This node is marked with a $*$ in (84). In adjunction, the old tree is rewritten at a distinguished node, the ADJUNCTION SITE. The subtree rooted at the adjunction site is substituted into the foot node in the auxiliary tree, and then the old tree is rewritten so that the new extended version of the auxiliary tree replaces the old subtree rooted at the adjunction site. (80c) shows what happens once the entry for *tuber* adjoins at the N node in (80b). In tandem with this syntactic change, the requirement that *x* be about tubers can be factored in as a constraint on reference; (80c) supposes that this distinguishes *x* from its distractors in context.

The steps presented in (80) fall out from the general execution of the SPUD algorithm, which is summarized in Figure 9.1.

The steps of this algorithm refer to abstract notions—ambiguity of reference, information conveyed, appropriateness and specificity of descriptors—whose implementation depends on an approach to meaning and interpretation based on logic.[1] Section 9.2 describes why modal specifications of knowledge and action, together with reasoning methods like those implemented in DIALUP, provide an attractive computational framework for deriving such representations. Before diving into these technicalities, we first sketch some of the motivations for developing an algorithm like SPUD.

In contrast to systems that use a cascade of special-purpose mechanisms to construct sentences, SPUD derives several advantages from its direct use of LTAG operators to identify

[1]Indeed, the algorithm given in Figure 9.1 is sufficiently general that SPUD can use similar steps to construct both definite and indefinite referring forms. The main difference lies in how alternatives are evaluated. When an indefinite referring form is used to refer to a BRAND-NEW generalized individual [Prince, 1981] (an object, for example, or an action in an instruction), the object is marked as new and does not have to be distinguished from others because the hearer creates a fresh "file card" for it. However, because the domain typically provides features needed in an appropriate description for the object, SPUD continues its incremental addition of content to convey them.

- Start with a tree with one node (e.g., S, NP) and one or more referential or informational goals.
- While the current tree is incomplete, or its references are ambiguous to the hearer, or its meaning does not fully convey the informational goals (provided progress is being made):
  - consider the trees that extend the current one by the addition (using LTAG operations) of a true and appropriate lexicalized descriptor;
  - rank the results based on local factors (e.g., completeness of meaning, distractors for reference, unfilled substitution sites, specificity of licensing conditions);
  - make the highest ranking new tree the current tree.

Figure 9.1: An outline of the SPUD algorithm

and combine semantic and syntactic elements simultaneously. These advantages include:

- SYNTACTIC CONSTRAINTS ARE HANDLED EARLY AND NATURALLY. In the problem illustrated in (80), SPUD directly encodes the syntactic requirement that a description should have a head noun—missing from the concept-level account—using the NP substitution site.
- THE ORDER OF ADDING CONTENT IS FLEXIBLE. Because an LTAG derivation allows modifiers to adjoin at any step (unlike a top-down CFG derivation), there is no tension between providing what the syntax requires and going beyond what the syntax requires.
- GRAMMATICAL KNOWLEDGE IS STATED ONCE ONLY. All operations in constructing a sentence are guided by LTAG's lexicalized grammar; by contrast, with separate processing, the lexicon is split into an inventory of concepts (used for organizing content or constructing descriptions) and a further inventory of concepts in correspondence with some syntax (for surface realization).

These advantages are observed in [Stone and Doran, 1997], but are echoed in previous work on using TAG in NLG, such as [Joshi, 1987] and [Yang *et al.*, 1991].

In fact, however, SPUD makes the heaviest demands on its logical representations, and draws the greatest benefit from them, as it keeps track of its incremental progress towards multiple goals in generating descriptions. The advantage that accrues from this process is examined in [Stone and Webber, 1998], where it is observed that SPUD naturally supports TEXTUAL ECONOMY. Textual economy refers to sentences where a speaker achieves communicative goals indirectly, by exploiting the hearer's recognition of inferential links to material elsewhere within a sentence. Textual economy leads to efficient descriptions

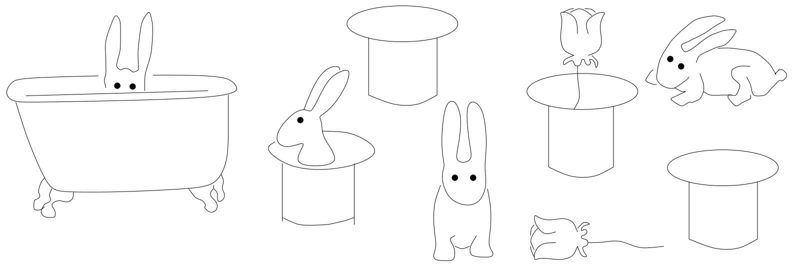


Figure 9.2: “Remove the rabbit from the hat.”

because the material that supports such inferences has been included to satisfy independent communicative goals, and is therefore OVERLOADED in the sense of Pollack [Pollack, 1991].

To see how SPUD supports textual economy, consider first how SPUD might derive (85) as an instruction for the hearer to carry out against the scene pictured in Figure 9.2.

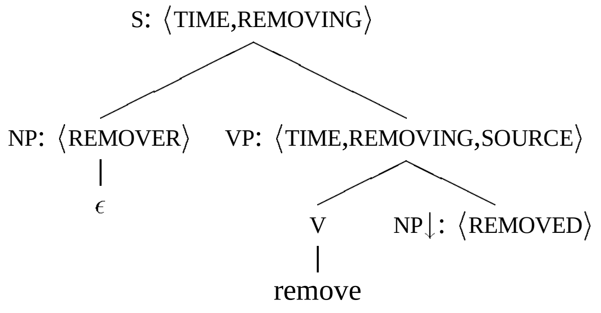
(85) Remove the rabbit from the hat.

Even though there are several rabbits, several hats, and even a rabbit in a bathtub and a flower in a hat, this command is sufficient to identify the rabbit currently in the hat and the hat that currently contains the rabbit. It suffices because one of the semantic features of the verb *remove*—that its object (here, the rabbit) starts out in the source (here, the hat)—distinguishes the intended rabbit and hat in Figure 9.2 from the other ones. Because of the inferential link between the identification of the rabbit and the specification of the type of action to be performed, (85) is an illustration of textual economy. (85) suggests that textual economy is a very important feature of concise, natural descriptions of actions.

In its derivation of (85), we assume that SPUD is given a general goal of describing a new action that the hearer is to perform, by making sure the hearer can identify the key features that allow its performance. For (85), then, SPUD is given two features of the action to be described: it involves motion of an intended object by the agent, and its result is achieved when the object reaches a place decisively away from its starting point. In explaining how SPUD realizes these goals in (85), our explanation assumes SPUD makes a nondeterministic choice from among available lexical entries. While we can in fact account for SPUD’s choices more precisely (see for example section 9.3.2), such an account would distract from our main point at present: to illustrate how SPUD can realize the textual economy of this example.

The first time through the loop of Figure 9.1, SPUD must expand an S node. One of the applicable moves is to substitute a lexical entry for the verb *remove*. Of the elements in the verb’s LTAG tree family, the one that fits the instructional context is the entry with imperative syntax, as shown in (86).

(86) a Syntax:



b Semantics:
nucleus(PREP, REMOVING, RESULT) ∧
in(PREP, start(TIME), REMOVED, SOURCE) ∧
caused-motion(REMOVING, REMOVER, REMOVED) ∧
away(RESULT, end(TIME), REMOVED, SOURCE)

The tree given in (86a) specifies that *remove* syntactically SATISFIES a requirement to include an S, REQUIRES a further NP to be included (describing what is removed), and ALLOWS the possibility of an explicit VP modifier that describes what the latter has been removed from.

The semantics in (86b) consists of a set of features, formulated in an ontologically promiscuous semantics, as advocated in [Hobbs, 1985]. It follows [Moens and Steedman, 1988] in viewing events as consisting of a preparatory phase, a transition, and a result state (what is called a *nucleus* in [Moens and Steedman, 1988]). The semantics in (86b) describes all parts of a *remove* event: In the preparatory phase, the object (REMOVED) is in/on SOURCE. It undergoes motion caused by the agent (REMOVER), and ends up away from SOURCE in the result state.

Semantic features are used by SPUD in one of two ways. Some make a SEMANTIC CONTRIBUTION that specifies new information—these add to what new information the speaker can convey with the structure. Others simply impose a SEMANTIC REQUIREMENT that a fact must be part of the conversational record—these figure in ruling out distractors.

For this instruction, SPUD treats the CAUSED-MOTION and AWAY semantic features as semantic contributions. It therefore determines that the use of this item communicates the needed features of the action. At the same time, it treats the IN feature—because it refers to the shared initial state in which the instruction will be executed—and the NUCLEUS feature—because it simply refers to our general ontology—as semantic requirements. SPUD therefore determines that the only ⟨REMOVED,SOURCE⟩ pairs that the hearer might think the instruction could refer to are pairs where REMOVED starts out in/on SOURCE as the action begins.

Thus, SPUD derives a triple effect from use of the word *remove*—increasing syntactic satisfaction, making semantic contributions and satisfying semantic requirements—all of which contribute to SPUD's task of completing an S syntactic constituent that conveys needed content and refers successfully. Such multiple effects make it natural for SPUD to achieve textual economy. Positive effects on any of the above dimensions can suffice to merit inclusion of an item in a given sentence. However, the effects of inclusion may go beyond this: even if an item is chosen for its semantic contribution, its semantic requirements can still be exploited in establishing whether the current lexico-syntactic description is sufficient to identify an entity, and its syntactic contributions can still be exploited to add further content.

Since the current tree is incomplete and referentially ambiguous, SPUD repeats the loop of Figure 9.1, considering trees that extend it. One option is to adjoin at the VP the entry corresponding to *from the hat*. In this compound entry, *from* matches the verb and *the* matches the context; *hat* carries semantics, requiring that SOURCE be a hat. After adjunction, the requirements reflect both *remove* and *hat*; reference, SPUD computes, has been narrowed to the hats that have something in/on them (the rabbit, the flower).

Another option is to substitute the entry for *the rabbit* at the object NP; this imposes the requirement that REMOVED be a rabbit. Suppose SPUD discards this option in this iteration, making the other (perhaps less referentially ambiguous) choice. At the next iteration, *the rabbit* still remains an option. Now combining with *remove* and *hat*, it derives a sentence that SPUD recognizes to be complete and referentially unambiguous, and to satisfy the informational goals.

9.2 A Modal Perspective on Conversational State

SPUD's derivation of sentences such as (85) makes use of detailed reasoning—reasoning that enables the generator to assess quickly and reliably at any stage how the hearer will interpret the current sentence, with its (incomplete) syntax and semantics. SPUD's incremental assessment and search involves a range of key questions about the sentence, particularly these:

- what (generalized) individuals might the hearer take the sentence to refer to?
- what would the sentence invite the hearer to conclude about those individuals?
- how tight a link does the sentence establish to the ongoing conversation?

SPUD answers these questions using a declarative specification of the conversational state. We describe this process in several stages. We begin, in section 9.2.1, by describing the logical principles by which such a specification can be designed, following [Clark and Marshall, 1981; Clark, 1996]. As section 9.2.2 argues, these principles provide a framework that meshes closely with the design of DIALUP. In particular, they systematically exploit descriptions of nested knowledge and indefinite information, in a way which fits DIALUP's expressive power. We use the remainder of this section to show how DIALUP allows such specifications to be accessed efficiently to settle the questions that SPUD raises.

9.2.1 A Modal Framework for the Common Ground

The management of conversation depends particularly on information that is part of the COMMON GROUND. Clark, following Lewis [Lewis, 1969], characterizes this information as in (87).

(87) *p* is common ground for members of community *C* if and only if:

1. every member of C has information that basis b holds;
2. b indicates to every member of C that every member of C has information that b holds;
3. b indicates to members of C that p. ([Clark, 1996], p. 94)

Clause 2 of (87) is self-referential; it indicates, to whatever agent applies it, that all three clauses are available as premises for inference for all the other agents in the group. This recursive structure matches the needs of agents that need to coordinate in tasks like reference; [Clark and Marshall, 1981] formulate an ingenious series of thought experiments to illustrate why the recursion is required. In determining the intended referent of an expression, speakers and hearers should always consider each other's perspectives. Failing to do so raises the possibility of misunderstanding. This recursive requirement can be satisfied only if agents formulate referring acts based on knowledge—such as that characterized in (87)—which can be assumed in each's model of the other to any level of embedding.

To embed the definition in (87) in logic, we need to fix a model of the basis that the definition refers to. For [Barwise, 1988], the basis can be represented directly in situation semantics as a situation with a circular structure [Barwise and Etchemendy, 1987]. But little is known about what proof search with such a representation would be like.

In a more standard formalization, the basis is represented indirectly; common knowledge is viewed as a particular fixed-point. Suppose $[\textsc{e}]A$ represents that everyone in a group knows that A holds. Then the condition that A is common knowledge within the group, $[\textsc{c}]A$, is governed by the two axiom schemes in (88).

(88) a $[\textsc{c}]A \Leftrightarrow [\textsc{e}](A \wedge [\textsc{c}]A)$
b $B \wedge [\textsc{c}](B \supset [\textsc{e}](B \wedge A)) \supset [\textsc{c}]A$

(88a), the fixed-point scheme, ensures that if a fact A is common knowledge, then everyone knows A, everyone knows everyone knows A, and so forth. (88b), the induction scheme, ensures that agents recognize as common knowledge everything that follows from their shared knowledge of what everyone knows. (See [Fagin *et al.*, 1995] for more details.) We can compare the induction scheme with the self-referential requirement proposed in (87). In the induction axiom, the proposition B, together with the shared inference from B to $[\textsc{e}](B \wedge A)$, constitutes the basis on which common knowledge of A depends. The induction scheme for common knowledge allows the basis to rest on any arbitrary proposition; thus, like any induction scheme, it gives rise to computationally explosive, open-ended search problems. Before eliminating the possibility that A is common knowledge, a theorem-prover must rule out all possible strengthenings of A to $B \wedge A$.

For DIALUP we will adopt a different formalization, following an idea explored already by [Clark and Marshall, 1981]. Clark and Marshall observe that, in conversation, the content that can count as the basis in (87) is derived in a small number of stereotypical ways. For example, one such basis is the shared perceptual environment in which conversants find themselves; another is the broad background that conversants derive from shared interests, experiences or community membership. Inferences about common knowledge in conversation therefore have a simpler form than suggested by the induction scheme. The inferences just combine information that is recognized to be available from SHARED SOURCES; they call these shared sources TYPES OF COPRESENCE.

This idea can be realized directly in DIALUP, by defining an appropriate theory of interrelated modal operators. We complement the knowledge [S] and [A] of speaker and addressee by a modality [CP]; [CP]A represents that A follows from the content available to speaker and hearer from their shared sources. Because this modality represents a kind of knowledge, it is subject to (VER) and (PI). In addition, the (INC) scheme applies, with [CP]$A \supset$ [S]A and [CP]$A \supset$ [A]A.

This modal theory already suffices to establish [CP] as a special case of common knowledge. For example, we can argue as follows to see that copresent information is available in each agent's model of the other at any level of nesting. We can use (PI) (or (VER)) inferences to reason from a proposition [CP]A into a proposition $[\mathrm{CP}]^k A$ in which the [CP] operator is nested arbitrarily deeply. Then the (INC) schemes can be used to replace each [CP] with either [S] or [A] at our discretion.

The copresence account avoids the pitfalls of arbitrary induction by using a more structured specification to flesh out the content of [CP]. Clark suggests that human speakers and hearers can and do arrive at a recognition of their common ground just by identifying the various sources of information to which both have access. These sources include the knowledge of a variety of communities to which speaker and hearer both belong. The different communities may have their origin for example in the different residence, occupation, interests, etc. of the two participants. Such sources provide the best concrete examples of types of copresence.

Here is the formalization of this. Each particular type of copresence is assigned a distinct modality $[\mathrm{CP}_i]$; the formula $[\mathrm{CP}_i]A$ says that the information available from source i entails A. Because all these modalities describe kinds or sources of knowledge, all are subject to (VER) and (PI).

To exploit these modalities, we establish an appropriate relationship between them and [CP]. For example, suppose speaker and hearer recognize themselves to share [CR] describing their common region of residence, [CJ] describing a common job, and [CI] describing some common interest. Then they can draw the correct inferences about what they share by indicating by inclusion schemes that [CP] can take into account any of this knowledge. Formally, [CR]$A \supset$ [CP]A, [CJ]$A \supset$ [CP]A and [CI]$A \supset$ [CP]A.

Copresence is much more tractable than an inductive definition of common knowledge because only conclusions of A are needed in showing [CP] A—not more general conclusions of $B \wedge A$. In DIALUP, the constraint algorithm for modal prefix matching makes this reasoning particularly efficient. DIALUP requires no more choice points and no larger proofs for a copresence theory than Prolog requires for the corresponding first-order knowledge-base. This is true even for any of the nestings of knowledge that common knowledge entails. As I show next, handling these nestings efficiently and correctly is a real advantage. By contrast, models based on compiling knowledge into partitions [Ballim and Wilks, 1991; Kobsa and Pohl, 1995; Taylor *et al.*, 1996]—and then maintaining separate databases for different nestings of operators—achieve efficiency for simple cases at the cost of significant penalties in completeness of inference and in defining new nested belief spaces.

9.2.2 Specifying the Common Ground

DIALUP offers an attractive set of resources for constructing specifications in terms of copresence. One simple strategy is already very powerful. This is to lift any formula in a first-order knowledge-base by prefixing it by a modal operator that abstracts how one would come to know the formula. Detailed, accurate stereotypes of classes of users can then be obtained by listing the modalities those users will be familiar with.

By establishing modal relationships among sources of copresence, these simple specifications can incorporate the fact, emphasized by Clark, that communities form nested sets. Clark's example contrasts the communities of San Franciscans (whose shared knowledge is given by $[\text{SF}]$), Los Angelinos (by $[\text{LA}]$) and Californians (by $[\text{CA}]$). We can indicate the epistemic consequences of community nesting by inclusion schemes among sources of copresence. In California, $[\text{CA}]A \supset [\text{SF}]A$ and $[\text{CA}]A \supset [\text{LA}]A$: shared knowledge of San Franciscans includes shared knowledge of Californians.

For natural language interaction, characterizing the limits of the common ground can be as important as itemizing concrete shared facts. Speakers, in formulating questions and answers to extend the common ground, depend on these characterizations of how their hearers' knowledge may go beyond or fall beneath their own. It is in this domain that nested and indefinite specifications of knowledge come into their own, and where the expressive advantages of DIALUP become most compelling.

Our knowledge of limits in the common ground is evidenced in Clark's contrast between two kinds of information people can have.

(89) a INSIDE INFORMATION of a community is particular information that members of the community mutually assume is possessed by members of the community.
b OUTSIDE INFORMATION of a community is types of information that outsiders assume is inside information for that community. ([Clark, 1996], p. 101)

He gives an example contrasting the inside knowledge of New Zealanders—we'll represent this by $[\text{NZ}]$—with the information outsiders have about it—which we represent using a more inclusive source of knowledge $[\text{O}]$.

There is a first, obvious difference between insiders and outsiders. If u is an outsider but v is an insider for community C (New Zealanders for example), then $[\text{C}]A \supset [\text{V}]A$ but not $[\text{C}]A \supset [\text{U}]A$. This prevents u and v from taking information supplied by C as shared, even when each happens to know that the other knows it. That's correct.

This exclusion is not the end of the story for an outsider, however. The outsider can still know quite a bit about insiders' information; such outsider knowledge can in fact be shared with insiders. To formalize the different kinds of knowledge involved, we appeal to Hintikka's formulation of knowing the answer to questions using wide-scope quantifiers [Hintikka, 1971], as in Chapter 7. In Clark's example, outsiders know that New Zealanders know who the prime minister of New Zealand is. Here's what this looks like:

(90) $[\mathrm{O}]\forall x(pm(x,nz) \supset [\mathrm{NZ}]pm(x,nz))$

This is outside information, as it looks like. Suppose *u* knows that Jim Bolger is the Prime Minister of New Zealand:

(91) $[\mathrm{U}]pm(bolger,nz)$

Then *u* can conclude that *v* knows who the Prime Minister is (and in fact that this is shared knowledge among New Zealanders):

(92) $[\mathrm{U}]\exists x[\mathrm{V}]pm(x,nz)$

On the other hand, if *v* knows the same thing, *v* knows that the fact is part of HIS shared knowledge with other New Zealanders. It's inside information.

We can easily imagine simple reasons why *u* might derive judgments like (92) in conversation. For instance, *u* could use (92) in deciding to ask *v* who the prime minister of New Zealand is, or in deciding to ask a wh-question rather than a yes/no question. Both of these uses presuppose that *u* has only indefinite information.

The formalization of action and knowledge introduced in Chapter 7 suggests an even broader range of application for indefinite specifications and queries. Using the techniques of Chapter 7, statements that describe what an agent knows how to do will be formulated using existential quantifiers and nested knowledge operators. For example, in instructions, the speaker must often rely on an outsider's description of the abilities of the addressee, using statements with a similar form to (90). Such descriptions cover cases where the instruction is executed in a future situation which the addressee will be able to perceive directly but which the speaker can now only characterize abstractly; they cover cases where the addressee has access to private knowledge that the speaker lacks. In these cases, only by representing and reasoning with an indefinite description of the addressee's knowledge can a speaker knowingly formulate an instruction the addressee will know how to follow. We give an example of this in section 9.2.4.

With generalizations like (90), we can also inventory the information that internal sources provide. This also seems to have its uses in NLG. As a QUERY, (90) asks whether a source of information can settle a general issue. Speculatively, some linguistic constructions may depend on the ability to settle an issue in this way. (I am particularly thinking of cases of reference where the speaker relies on the hearer's inference to determine whether a property should contribute to distinguishing an entity from its alternatives.) In such cases, queries and specifications like (90) will be needed to determine whether these constructions can be appropriately used.

We now have seen, in sections 9.2.1 and 9.2.2, how the specification of the common ground not only falls within DIALUP's range but allows all of that range to be exploited.

In sections 9.2.3, 9.2.4 and 9.2.5, we look at the queries about these specifications that SPUD needs DIALUP to assess. This analysis reveals not only a continued close fit between DIALUP's expressive power and the needs of NLG, but also a match between DIALUP's logic programming search and the practical requirements of an NLG system.

9.2.3 Assessing Reference in SPUD *Using Modal Logic*

SPUD's model of reference begins with an explicit representation of the exact content by which speaker and hearer agree on the referents of referring expressions. We shall call this part of the meaning of a sentence its PRESUPPOSITION. SPUD relates the presupposition to a modal specification of common ground, as well as to a modal specification of the attentional state of the conversation.

In this section, we explain the model of presupposition as implemented in SPUD, and show how it gives rise to logic programming queries in DIALUP. In SPUD the results of these queries are managed more efficiently by adopting classic techniques from NLG, like the use of constraint networks for reference resolution. A logical foundation is straightforwardly compatible with these optimizations.

Our use of the term presupposition follows the theoretical perspective of [van der Sandt, 1992] that presuppositions are ANAPHORS that are resolved against an evolving model of discourse. This view modulates the received view of a presupposition, from Frege and Russell, as a statement of the uniqueness conditions under which a sentence refers successfully. For example, the received view famously represents the uniqueness condition as the presupposition (93b) of (93a):

(93) a The King of France is bald.
b $\exists x(kof(x) \wedge \forall y(kof(y) \supset x = y))$
c $kof(x)$

The anaphoric view replaces the formula (93b), with its logical complexity, by the formula (93c) and a complex process of RESOLUTION against the context. Resolving (93c) requires not only showing that the logical condition is satisfied in the common ground but also providing a discourse referent from the context that can serve as the value for the variable x. For (93c), the uniqueness derives from the fact that the speaker and the hearer must agree on how the presupposition is resolved; thus this resolution of presupposition parallels the resolution of other anaphoric elements in the sentence.

Proposals like [van der Sandt, 1992] are developed within the formal semantics of discourse representation theory [Heim, 1982; Kamp, 1981; Kamp and Reyle, 1993] or dynamic semantics [Groenendijk and Stokhof, 1990b; Muskens, 1996]. Let us reconstruct this idea in a computational setting designed around natural language generation, and investigate its consequences.

First, we describe the interface with sentence meaning. We design the grammar to deliver the content of each sentence in two parts: the PRESUPPOSITION $P\mathbf{x}$—an open formula containing free occurrences only of the variables in the sequence $\mathbf{x}$—which is resolved anaphorically and the ASSERTION $N\mathbf{x}$—another open formula containing only occurrences of variables in the sequence $\mathbf{x}$—which contributes new information to the evolving discourse.

Second, we consider what counts as a possible resolution of the presupposition $P\mathbf{x}$.

(94) A RESOLUTION of the presupposition $P\mathbf{x}$ is a proof of $[\text{CP}]\exists\mathbf{x}P\mathbf{x}$.

It is clear why (94) appeals to the modal operator $[\text{CP}]$—this implements the requirement that only the shared common ground can be taken into account in resolving a presupposition. (In cases of accommodation where presupposition and common ground seem to disagree [Lewis, 1979], we can follow Lewis in assuming it is the content of the common ground that is adjusted, not the requirement imposed by the presupposition.)

The narrow scope of $\exists\mathbf{x}$ in (94) may be more puzzling, however. This choice indicates that the resolution requires us to find mere discourse referents to satisfy the presupposition—not concrete entities in the world, as would be required if the quantifier was given wide scope. To see the difference, consider a conversation that takes place during a murder mystery. The murder has occurred, so we know there is a murderer: $[\text{CP}]\exists x.\mathit{murderer}(x)$. But the case is not solved; we do not know who the murderer is. This would be expressed by $\exists x[\text{CP}]\mathit{murderer}(x)$, a formula which must be false in this context. Suppose one party to this conversation wishes to tell the other *the murderer used a wrench*; the sentence will carry the presupposition $\mathit{murderer}(x)$. By (94), there is a resolution to this presupposition, where x is instantiated to whoever our indefinite shared knowledge tells us the murderer must be. If we were required to prove $\exists x[\text{CP}]\mathit{murderer}(x)$ there would be no resolution to the presupposition.

Anticipating the use of Skolemization and unification to derive resolutions to presuppositions, we can regard a resolution as supplying terms that denote discourse referents for each of the variables in the presupposition. (Under this view, it is easiest to assume, as does the implementation, that case analysis is irrelevant to the resolution of presuppositions. In fact, this assumption is an oversimplification, but in light of the flexibility of pronominal reference in discourses which describe disjunctive sets of cases [Stone, 1992], we should expect a proof-theoretic view to persist even after this assumption is relaxed.)

Given the state of the common ground in conversation, some presuppositions cannot be resolved at all, while others might potentially be resolved in many ways, not all of which would equally respond to the goals and structure of the ongoing conversation. This difference among alternative resolutions can be formalized by appeal to the SALIENCE of discourse referents. The salience of a discourse referent measures the extent to which the ongoing discourse puts that discourse referent at the center of speaker's and hearer's attention; influential computational accounts of salience in discourse include [Grosz and Sidner, 1986; Grosz *et al.*, 1995]. Salience constrains reference; a referring expression picks out the just the most salient referent that the expression describes. We can apply the same idea to the resolution of presuppositions.

First, we use modal formulas to define a salience ranking among entities. A fact $[\text{CP}]d(t,t')$ indicates that t' is a distractor for t. (Here t and t' are either concrete entities, explicit Skolem terms, or bound variables within the scope of an appropriate quantifier.) Accordingly, to refer to t, we must distinguish t from t'. Equivalently, $[\textsc{cp}]d(t,t')$ says that t' is more salient than t. (Obviously in general these facts should be indexed by time or utterance.)

This ranking among entities determines a ranking among resolutions. Given a presupposition $P\mathbf{x}$ depending on a sequence of variables $\mathbf{x}$, one resolution is more salient than another if for any variable x that appears in $\mathbf{x}$, the value the first supplies to x is more salient than the value the second supplies. Given this ranking, we say a presupposition is SUPPORTED if it has a unique, maximally-salient resolution. For an utterance to be felicitous, its presupposition must be supported.

This characterization of presupposition is attractive for NLG, because it accounts for reference in descriptions of action in a simple and elegant theoretical model. For example, let us return to the scene of rabbits, hats, flowers and bathtubs depicted in Figure 9.2. Against this scene, we can refer successfully using the NP *the rabbit in the hat*. Following the example of (93c), we view the presupposition of this expression as a flat formula relating multiple contextually-determined variables, something like (95):

(95) $\mathit{rabbit}(x) \wedge \mathit{hat}(y) \wedge \mathit{in}(x,y)$

Recall that this formula is viewed as a anaphor which must be matched against the context rather than as a self-contained statement of the conditions that must hold for *the rabbit in the hat* to be felicitous. In particular, given the technical definition of support above, to support this presupposition (and thereby obtain a felicitous use of the NP) requires the three combined constraints to be derivable from the shared representation of the common ground in only one (salient) way. This supporting derivation provides the pair of values for x and y in which the rabbit and hat referred to are uniquely identified. The value for x is the rabbit in the hat, and that for y is the hat with the rabbit in it.

Compositional semantics can compose the anaphoric constraint given in (95) straightforwardly. Meanwhile, the theory does not require self-contained conditions on felicity of reference to be built up compositionally. Thus, in contrast to the Russellian account of reference, there is no need to identify an intermediate syntactic scope—either in the referring expression or its logical presupposition—at which the uniqueness operators for the different entities apply.

More generally, if other elements besides noun phrases contribute anaphoric presuppositions to a sentence (and of course they will), these additional constraints will figure into the resolutions of referring expressions in the sentence. Indeed, we can see the instruction of (85)—*remove the rabbit from the hat*—as contributing essentially the same presupposition as shown in (95). Theoretically, then, we can account for the support of its presupposition and the felicity of its reference in exactly the same way.

This model can also be implemented for NLG in an attractive way using DIALUP. (SPUD contains such an implementation.) The implementation rests on two observations. First, as observed in Chapter 5, DIALUP's constraint logic programming search makes it sensible simply to count proofs in place of accumulating the distinct combinations of values for variables under which proofs are possible.

Second, with this model, DIALUP's logic programming search goes hand in hand with techniques like constraint satisfaction that are already used for efficient reference resolution in NLG. Consider a case where DIALUP attempts to find resolutions for a conjunctive presupposition $P\mathbf{x} \wedge Q\mathbf{x}$. Following logic programming search on (94), DIALUP introduces a fresh $[\text{CP}]$ transition α, and appropriate new logic variables $\mathbf{u}$ with domain α. Then it attempts two independent goals: showing $P\mathbf{u}$ at α, and showing $Q\mathbf{u}$ at α.

Such independent goals are not one of DIALUP's strengths. Because of backtracking, the interpreter must iterate through successive combinations of solutions for P and solutions for Q. This search strategy may be more expensive than needed. Moreover, backtracking means that solutions for P and for Q cannot be obtained separately and then combined. Such reuse of work is a natural response to the incremental construction of interpretation that generators such as SPUD use.

To address these difficulties, we can apply simple constraint-satisfaction algorithms [Mackworth, 1987]. These techniques are already used to maintain distractors compactly and efficiently in generation, for example in [Dale and Haddock, 1991]. Using $\vdash A$ to indicate that A is derivable from an input specification (in DIALUP), the basic observation is that $\{\mathbf{b} \mid \vdash [\text{C}]P\mathbf{b} \wedge Q\mathbf{b}\} = \{\mathbf{b} \mid \vdash [\text{C}]P\mathbf{b}\} \cap \{\mathbf{b} \mid \vdash [\text{C}]Q\mathbf{b}\}$. Using this observation, we can perform the two queries separately using DIALUP, to obtain a pair of constraints. Then we can combine and resolve constraints as part of a separate, special-purpose module. Now, a programmer can easily ensure that each query will result in a manageable set of tuples that puts a relatively small load on the constraint-satisfaction process.

9.2.4 Assessing Content in SPUD *Using Modal Logic*

Section 9.2.3 showed how possibilities for reference can be maintained declaratively, incrementally and efficiently using DIALUP. In this section we look at the complementary problem of assessing what a partial sentence contributes to the conversation. Again, DIALUP offers a natural tool to make this assessment in an expressive and efficient way.

Recall that the semantic contribution that the sentence makes to the common ground is represented by its assertion, a formula $N\mathbf{x}$. The interpretation of this assertion depends on how the variables in $\mathbf{x}$ are resolved by the presupposition. The most straightforward case arises when the presupposition associates $\mathbf{x}$ with a sequence of concrete entities $\mathbf{r}$. Then asserting $N\mathbf{x}$ makes a bid that the common ground be updated so that (96) holds.

(96) $[\text{CP}]N\mathbf{r}$

This is the simple account that is implemented in the present version of SPUD. Anticipating possible extensions to (96), however, in the rest of this discussion we will use a placeholder R to represent the new information asserted by a sentence.

Any generator can use a logic programming query, formulated in terms of R, to assess whether a given fact F will be taken as part of the common ground after R is added to the common ground. The logical representation for this query implements the following algorithm. Consider the content of the common ground beforehand. Suppose this content is extended so that the content of the assertion is also part of the common ground. Then check whether F now follows from that extended common ground. This algorithm is how DIALUP processes the query $[\text{CP}](R \supset [\text{CP}]F)$.

SPUD uses such queries repeatedly, as it monitors the incremental construction of sentence semantics. Here is how. At each stage, SPUD maintains a list G of facts that it should communicate but has not yet communicated. So SPUD composes a new query to DIALUP $[\text{CP}](R \supset [\text{CP}]g)$ for each g in G. The results of these queries tell SPUD which goals in S it has now satisfied; SPUD retains the goals in G and not in S for further iterations of adding content.

Logic programming leaves the complexity of these queries firmly in the hands of the designer of the domain. In the simplest case, g is a primitive fact; there are no (shared) rules in the knowledge base describing how g might be derived by inference. Then DIALUP will only consider facts from R in trying to derive g. This reduces to a simple unification test between g and the facts asserted in the sentence. This is a simple test indeed, especially when the logic programming engine implements discriminating and efficient indexing methods to retrieve applicable clauses.

Even if g can potentially be established by complex chains of reasoning, logic programming methods are available to reduce the cost of repeated queries. For example, rules can be rewritten to fail more quickly by first making sure all the new information needed for the inference has been included in the assertion of the sentence and only after this information is obtained consulting the shared background to complete the inference.

There is no requirement of simplicity on the goal g. It can be any goal formula accepted by DIALUP. This offers an exciting opportunity to exploit DIALUP's nested and indefinite information to more flexibly control the content of a sentence. For example, it is possible to consider a goal that the hearer know the answer to a question. For the hearer to know what satisfies p, we use the goal $\exists x[\text{A}]p(x)$. This gives rise to the incremental query $[\text{CP}](R \supset [\text{CP}]\exists x[\text{A}]p(x))$. The query is set to exploit not just the concrete specification of facts in the common ground but also indefinite characterizations of the hearer's knowledge.

When we look at assertions that describe indefinite discourse markers, however, further precision seems to be required to represent the contribution R of the assertion. In the rest of this section, we describe a natural way to extend SPUD to better represent asserted contributions, and provide a simple example of how this extension would work. With indefinite discourse markers, the presupposition picks out terms for entities that may be defined only at the world α introduced in the proof as representative of the content of the common ground. Let us write these terms as $\mathbf{r}^{\alpha}$, indicating explicitly their dependence on the world α. Observe that the assertion cannot refer to the world α itself, because α is merely assumed for the sake of argument during the resolution of the presupposition and goes out of scope afterwards.

The interpretation of markers $\mathbf{r}^{\alpha}$ in assertion must instead follow the information that the hearer has about these markers. The information, and hence the interpretation, will vary during the incremental construction of a sentence. One possibility is that the presupposition P of the sentence remains incomplete, in that the context provides alternative salient resolutions to P besides $\mathbf{r}^{\alpha}$. In this case, we do not yet know whether the description in the sentence will allow the hearer to identify the referents $\mathbf{r}$ on any concrete basis (even using private knowledge). Thus, we should transfer $\mathbf{r}^{\alpha}$ over to the world where the assertion is made, so as to obtain a condition we write as (97).

(97) $[\textsc{cp}]N\mathbf{r}^{\dagger}$

The notation $\mathbf{r}^{\dagger}$ in (97) indicates that dependencies on α in each occurrence of a term in $\mathbf{r}$ must be replaced by references to the world where the matrix of the $[\textsc{cp}]$ formula is evaluated. For example, suppose the presupposition of a sentence is $p(x, y)$ and the assertion is $r(x, y)$. Suppose the presupposition refers in context to a concrete individual a and an indefinite individual f that depends on a and varies from world to world. That means that the resolution for the presupposition has the form $q(a, f(a, \alpha))^{\alpha}$—using α to represent an arbitrary world representative of the content of the common ground. Then the assertion, as specified by (97), is treated as though it had the following functional translation at world μ:

(98) $\forall x : \textsc{cp}.r(a, f(a, \mu x))^{\mu x}$

According to (98), this sentence adds to the common ground the fact that a stands in the relation r to whatever individual f happens to assign to a.

In contrast, once the presupposition identifies $\mathbf{r}$ uniquely, we can use the presupposition itself to record the basis which might allow the hearer to identify $\mathbf{r}$. The world named by α can be replaced by any path term which leads to the world where the $[\textsc{cp}]$ formula is evaluated. We write this as in (99).

(99) $[\textsc{cp}](P\mathbf{r}^{*} \supset N\mathbf{r}^{*})$

The idea behind (99) is to shortcut an inference that might otherwise have to be specified using equational reasoning. We use (99) when it is part of the common ground that a unique (salient) sequence of discourse markers satisfies P. In these circumstances, it may happen that some participants in the conversation have independent information about concrete individuals that satisfy P. Call these individuals $\mathbf{a}$. By accepting the meaning of the sentence, these participants therefore discover concrete entities $\mathbf{a}$ that satisfy N. To draw this conclusion using (97), we must use participants' knowledge that $\mathbf{r} = \mathbf{a}$ together with the axiom of indiscernibility of identicals. As we shall see, this inference typically follows automatically from the management of path terms in (99).

Again, we can adapt the example of $p(x, y)$ and $r(x, y)$ for concreteness. When the presupposition is resolved as $p(a, f(a, \alpha))^{\alpha}$, then the assertion as specified by (99) is treated as though it had the following functional translation at world μ:

(100) $\forall x : \textsc{cp}.\forall y \leq \mu x.(p(a, f(a, y))^{\mu x} \supset r(a, f(a, y))^{\mu x}$

(100) makes a claim of any object u that f happens to assign to a at some world: it is part of the common ground that if a and u are related by p, then a and u are related by r. (We can streamline the notation of (100) as $[\textsc{cp}](p(a, f(a,*)) \supset r(a, f(a,*)))$.)

An example of an indefinite instruction can illustrate the precision that a link between a generator and DIALUP enables. We consider evaluating the instruction in (101).

(101) Type your social security number.

One of the speaker's intentions in formulating this instruction is that the hearer know what number to type.

(102) $\exists n[\textsc{a}]\textit{should-type}(n)$

Importantly, we do not assume that the speaker would know how to carry out this instruction concretely; the speaker may not know what number to type.

The common ground for this example must include the two facts formalized in (103). Everybody (in the domain) has a social security number, formalized as in (103a); and, at least as applied to the hearer in (103b), one knows one's own social security number.

(103) a $[\textsc{cp}]\forall x \exists n.\textit{ssn}(x, n)$
b $[\textsc{cp}]\forall n(\textit{ssn}(a, n) \supset [\textsc{a}]\textit{ssn}(a, n))$

With the referring expression *your social security number,* the instruction in (101) carries the presupposition in (104).

(104) $\textit{hearer}(u) \wedge \textit{ssn}(u, v)$

We can suppose that this has a unique resolution; $u = a$ and the term for v is a discourse referent formulated in terms of the Skolem function f introduced for the quantifier in (103a) and abstracting a variable possible world path by $*$—i.e, $v = f(a,*)$.

Thus, when we reach (101), the representation of the content contributed by in the instruction is:

(105) $\textit{hearer}(a) \wedge \textit{ssn}(a, f(a,*)) \supset \textit{should-type}(f(a,*))$

Overall then, we post this query to DIALUP:

(106) $[\textsc{cp}]([\textsc{cp}](\textit{hearer}(a) \wedge \textit{ssn}(a, f(a,*)) \supset \textit{should-type}(f(a,*))) \supset$
$[\textsc{cp}]\exists x[\textsc{a}]\textit{should-type}(x))$

DIALUP uses the following reasoning to show that this query is true. Decomposing this goal introduces transitions α and β of type [CP], a first-order logic variable y of domain $\alpha\beta$ and a further transition γ of type $[\textsc{a}]$. Then we're looking to prove $\textit{should-type}(y)$ at world $\alpha\beta\gamma$. We can apply the content of the sentence here, by specializing the parameter $*$ to refer to a prefix of $\alpha\beta$ so that the f term respects the domain constraint of the variable y; we can take $\alpha\beta$ itself as representative (although of course in DIALUP this is abstracted by a constraint).

This means we are now looking at the goals *hearer*(a) at $\alpha\beta\gamma$ and *ssn*$(a, f(a, \alpha\beta)$ at $\alpha\beta\gamma$. The first goal immediately follows from the specification of the conversational setting: it is part of the common ground who is the hearer now. For the second goal, we can apply clause (103b), since γ is an $[\text{A}]$-transition. This reduces the goal to showing *ssn*$(a, f(a, \alpha\beta)$ at world $\alpha\beta$. This just follows from clause (103a).

9.2.5 *Guiding Choices in* SPUD *Using Modal Logic*

When SPUD looks to extend a sentence, it uses the queries described in sections 9.2.3 and 9.2.4 to assess how much progress towards SPUD's goals each option allows. The greedy strategy outlined in Figure 9.1 dictates that SPUD adopt whatever option fares best under this evaluation. In particular, in the current implementation, SPUD first narrows consideration to the options that leave the fewest goals for further content to later steps of execution; among this set, SPUD considers only those that leave variables' distractor-sets for reference the smallest.

These criteria do not always determine a unique entry for SPUD to use, however. Before guessing randomly, SPUD makes a further determination of which options link most tightly with the context. This determination, like SPUD's others, involves a modal logic query against a declarative specification.

At this stage, SPUD compares two lexical entries, each of which represents a possible extension of the current sentence at the current stage. Both of these entries are associated with the presupposition that they impose, which we will write $P\mathbf{u}$ and $P'\mathbf{v}$. Intuitively, the one with the most specific presupposition makes the most precise demands of the context.

To effect this test in modal logic, we first need to determine what information should be taken into account. We shouldn't use $[\text{CP}]$. $[\text{CP}]$ contains too much information to compare these entries meaningfully. By hypothesis, both entries apply and so both presuppositions are provable in the current common ground.

Instead we need to consider a collection of general background knowledge, which we can identify with the contents of the modal operator $[\text{GK}]$. This knowledge can include generalizations about the meanings of words and their relationships, and can include generic regularities in the world, but should not describe the current situation.

We can determine whether P makes a better fit to the context than P' by considering just the information in $[\text{GK}]$. If we look in $[\text{GK}]$ at every case where P applies, and see that it is also a case of P', then we know that the first lexical item is no weaker a link to the context than the second. Logically, the query here is (107).

(107) $$[\text{GK}](\exists \mathbf{u} P\mathbf{u} \supset \exists \mathbf{v} P'\mathbf{v})$$

(This query also involves nested implication and an indefinite program clause, though one hardly as impressive as (106).) An advantage of this query is that is determined only by general and lexical information. The result of this query in SPUD is therefore precompiled and built into the structure of the lexicon.

A similar query is possible to compare the assertions of different lexical items. Now it is less clear whether it should take only general information into account or should be assessed relative to the overall common ground. For now, SPUD uses the general query, for purely computational reasons.

9.3 Worked Examples

This section provides two sets of detailed examples that illustrate how modal specifications are supplied to SPUD and what SPUD does with them. The first, presented in section 9.3.1, investigates a descriptive setting; it focuses on how SPUD can adjust the facts it conveys and the constructions it uses to respect different specifications of the kinds of information SPUD shares with its addressee. The second, presented in section 9.3.2, returns to the F-16 maintenance instructions described in Chapter 1. It focuses on how SPUD can produce concise descriptions of actions by using its logical model of the common ground and sentence interpretation to recognize when optional descriptive details of path and manner can be omitted.

[Bourne, 1998] reports another case study of generating instructions in SPUD. She focuses on generating an appropriate realization of the TERMINATION CONDITIONS which indicate to an addressee when to stop an instructed activity. Describing termination concisely and naturally demands reasoning about interpretation, because when an activity should stop can be indicated not only by an explicit temporal modifier but can also follow from how objects of verbs, paths and purposes of events are described. When termination information is available from these other sources, it is much less likely to be described explicitly. Bourne provides a computational account using SPUD, based on appealing to simple and general logic programming rules that describe the internal structure of events and relationships among events.

9.3.1 Sources of Information and Customized Text

The first example is an exploration of how to specify the content of the common ground in a modular way and then reuse this specification to provide different descriptions of an event to different audiences. The content of the example—an event in a murder mystery story—is somewhat facetious; the range of variation among addressees—what modality they communicate in and what background knowledge they have access to—most certainly is not.

Here is the scene. The speaker and the addressee share mutual familiarity with a certain country mansion; the speaker is there, engaged with the addressee in a process of communication (about which more later). Suddenly, from the study, just a few doors down the hall, an explosive boom blasts out. It could only have come from a gun (this being an incipient murder mystery). The speaker decides to describe this event to the addressee.

The scenario now ramifies into six alternatives, depending on the medium by which the speaker is engaged in communication with the addressee and the addressee's background knowledge. For condition one (C1), the speaker is writing a letter to the addressee; the addressee can be assumed to have no access to the speaker's present situation. For condition two (C2), the speaker is talking to the addressee over the phone; the addressee will doubtless have heard the blast, but may not have perceived more about it. Finally, for condition three (C3), the speaker and the addressee are engaged in a face-to-face discussion; the addressee has just the same sensory capabilities as the speaker does. Modulating each of these conditions is whether or not the addressee has the cosmopolitan (or cynical) upbringing required to know that a blast under these circumstances can only mean a gun.

The differences and commonalities among these alternatives can be abstracted using four sources of information. Following section 9.2.1, each source is assigned a modal operator. There a source [M] encoding familiarity with the mansion—providing information shared under all alternatives—as well as three others whose status varies: [G], for knowledge about guns; [O], for awareness of the occurrence of sounds in the environment; and [L], for awareness of the localization of sounds in space.

Specified in [M], we find that the room in a certain place in the mansion is a study, and that this is the only study there. In [G] we find that any boom must be caused by a gun. In [O] we find that a boom has in fact just occurred, and that there has been nothing else like it recently. In [L] we find that the boom came from the study.

Our six alternatives are now identified simply by the combinations of modal operators which constitute the common ground in that case. For C1, we have [M] (+[G]); for C2, [M] + [O] (+[G]); for C3, [M] + [O] + [L] (+[G]). When SPUD is given this specification and instructed to describe the boom event—making sure the addressee knows the features the speaker has just learned—the six sentences in (108) are output. ((108c) involves two sentences because a quirk of the little grammar used here prevents the two facts from being combined into a single sentence.)

(108) a ([M]) There was a boom made by a gun from the study.
b ([M] + [G]) There was a boom from the study.
c ([M] + [O]) The boom was from the study. The boom was made by a gun.
d ([M] + [O] + [G]) The boom was from the study.
e ([M] + [O] + [L]) The boom was made by a gun.
f ([M] + [O] + [L] + [G]) [silence]

Both the amount of information provided and the linguistic constructions used to present that information depend on the common ground and change in parallel with variation in the scenario. (108b) in particular is abbreviated dynamically, using a query that assesses its incremental interpretation and reveals that the inference to the gun will already be made by this addressee.

This example is intuitive rather than immense. It aims to illustrate that the notion of source of information—motivated for dialogue in [Clark and Marshall, 1981] and linked to modal logic one way here—is simple and flexible. Programmers need not to be intimidated by the modal notion; with a little patience, they can easily explore it to capture whatever organizations of information arise from the structure of the domain or the composition and variation in texts about the domain.

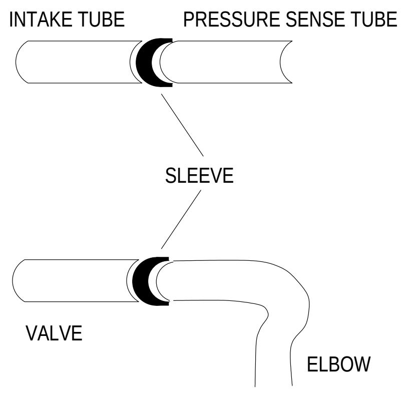


Figure 9.3: A piping schematic for some instructions.

9.3.2 Semantics, Reference and Lexical Choice

The second example illustrates an interaction of lexical semantics and world knowledge in NLG that can be modeled using declarative characterizations of interpretation. The goal of the example is to mirror patterns of description found in the F-16 maintenance manual (first described in section 1.1). It represents part of a larger effort to characterize the syntax, semantics and discourse of the F-16 manual as well as some of the logistics of aircraft repair [Badler and others, 1998].

A common maintenance operation in the fuel system involves accessing and adjusting fittings between pipes. (Section 1.1 also alluded to this.) The joint between a pair of pipes is typically sealed using a sleeve and then secured using a coupling that surrounds the sleeve. Before maintenance, the coupling is removed and the sleeve is slid away from the joint. Afterwards, the sleeve is positioned over the joint once more and a coupling is installed around it. The F-16 manual is very consistent in its use of these verbs and nouns to describe this operation.

In formulating and posing queries to DIALUP that compare lexical semantics and a domain specification, SPUD offers a declarative model of the decisions that go into the construction of these consistent descriptions of actions. In this section, we will illustrate this for two instructions, meant to be executed in succession against the scene illustrated in Figure 9.3. The instructions are characteristic of actual instructions and the schematic is a simplified version of a real one. We start with a logical specification of the shared knowledge schematized in the figure. Note that this specification includes some facts about the geometry and function of objects in the schematic whose role (and derivation) may not immediately evident. The two sleeves are shown in the configurations where they fulfill their usual function of sealing joints. (Only such configurations count as *positions* for the maintenance manual.) The sleeves are in contact with the pipes and will stay in contact with pipes if moved horizontally. The specification also provides a simple theory for describing the consequences of motion events in the domain; I used a modal presentation of explanation closure for reasoning about change (since I have not yet implemented the defeasible inference described in Chapter 6).

The first instruction is (109).

(109) Slide the sleeve on the elbow.

The key to the instruction is the choice of verb and its impact on reference. Of course, we need to identify the intended object and path for the hearer to be able to execute this instruction. However, this setting involves only definite knowledge and in fact the lexical semantics for any motion verbs will refer to the object and path. So the requirements of planning reduce to the requirements of reference.

Now SPUD first calls DIALUP to determine which verbs are compatible with this action. Most are rejected. The case of *position* is interesting: because of function and geometry in the domain, the configuration after carrying out the instruction in (109) does not have the sleeve in position. Of the entries that remain, *slide*—by contrast with potential alternatives like *move* and even *place* that could also describe the needed action—imposes a lexical constraint on its arguments. To slide, the object must move along a surface throughout the path. This constraint is not only met in the schematic, it contributes to the presupposition and is therefore factored into reference resolution by SPUD (and DIALUP). The more constrained reference problem provides a decisive reason for SPUD to select *slide* initially.

The next step adjoins in *on the elbow*. This distinguishes the elbow from its alternatives—it is the only elbow in the schematic—and thereby also distinguishes the path we are taking. Then it suffices to describe the object as *the sleeve*. While there are two sleeves in Figure 9.3, only one meets the rest of the presupposition maintained during sentence construction. Only for it does the path onto the elbow keep it in contact with a surface.

The second action is described as in (110).

(110) Reposition the sleeve.

The instruction is undertaken in a new context where the current state reflects the result of performing (109) and the history records the action and its earlier state. Again the key task is the choice of verb, and again SPUD makes this choice principally by assessing progress towards reference resolution in conjunction with DIALUP. Again, general verbs like *move* and *place* leave relatively unconstrained problems for reference, while the more precise lexical entries *slide*, *position* and *reposition* carry presuppositions that immediately narrow the referential possibilities. In particular, *slide* describes the relation of contact during the event between the moved object and another surface; *position* describes the relation of geometry and function between the moved object and the end of its trajectory. But *reposition* not only demands this functional relation, thanks to *re-* it imposes a further presupposition that the object have been moved from its position. Not surprisingly, SPUD chooses *reposition*—its lexical position is sufficient to latch on directly to the sleeve moved in the previous instruction and the position it started from as that instruction began. That leaves only the syntactic task of substituting an appropriate object description for *the sleeve*. (The current version of SPUD has no lexical items for pronouns, as in fact befits this domain.)

In these examples, the lexical meaning for verbs like *slide* and *position* allows the generator to determine when the entries apply, what the hearer will interpret from them, and thus which lexical entry to use at each step in augmenting the sentence. These different uses for the same information testify to the beauty and simplicity of adopting a declarative framework for NLG and the importance of being able to construct queries that explicitly access a range of different sources of information in NLG.

9.4 Summary

Chapter 1 argued that NLG is a reasoning task. This chapter has explored what kind of reasoning is involved. Reasoning is required to assess alternatives for reference—a particular theme of sections 9.1, 9.2.3 and 9.3.2. The assessment can benefit from access to a structured, declarative modal description of the common ground. Different reasoning is required to determine the contribution of new information—a particular theme of sections 9.2.4 and 9.3.1. This reasoning benefits from queries and specifications involving nested modal operators and indefinite constructs. Finally, reasoning must test the fit of constructions to context, as described in section 9.2.5. The abstract discussion and implemented examples of this chapter show how the studies of reasoning in Part I and representation in Part II together provide systematic and promising support for the exciting and challenging application of NLG.

10 Conclusion

As observed from Chapter 1 to Chapter 9, natural language generation (NLG) requires substantial reasoning. This dissertation documents a systematic preliminary effort to show how inference in modal logic can be used in practice to support this reasoning.

A generation system can produce a concise and precise sentence only by drawing inferences about how the hearer will interpret that sentence as dependent on, and as an update to, the common ground. Such inference is naturally founded on a declarative framework, like modal logic, for describing the effects of actions in the domain and the knowledge of the participants in the conversation. From a theoretical perspective, modal logic seems particularly attractive because it so easily accommodates the partial and indefinite information that underlies so much of conversation and action. This dissertation has engaged the goal of bringing this theoretical advantage to bear more directly to bear in the construction of NLG systems. This chapter summarizes our results on this problem, and concludes with a statement of some of the outstanding problems and questions raised by these results.

10.1 Overview of Results

The key contribution of this dissertation lies in describing how specifications in modal logic—even and especially those that appeal to indefinite constructs—can be stated naturally and executed, using streamlined search techniques, to guide NLG systems. This section outlines the principles for the execution of indefinite modal specifications derived in Part I and the principles for constructing indefinite modal specifications derived in Part II.

10.1.1 Executing Indefinite Modal Specifications

To execute modal specifications requires leveraging both the flexibility of efficient classical theorem-proving and the distinctive modularity of modal logic. This is a significant problem—as shown in Chapter 2—because the two are at odds. On the one hand, flexible search strategies impose no constraints on the relationships among inferences and, by thus ignoring modularity, leave open hopelessly wild possibilities for search. On the other hand, brute-force modular systems place such overbearing constraints on the order in which search must proceed that it becomes impossible to guide that search intelligently. The need to balance these opposing principles is why neither structural tableau methods—heavy on modularity—nor semantics-based resolution methods—heavy on flexibility—offer the last best word on modal deduction; opportunities to balance the principles will remain at the heart of research on the subject.

One strategy for balancing the flexibility of goal-directed search with the modularity of modal logic is the design of a modular inference system based on logic programming search. Such a system, called DIALUP, is derived in Chapter 3. DIALUP combines the use of logic variables and unification for flexible search with an optimized representation of possible worlds—as distinct constants—that exploits the eager search of the logic programming interpreter. The proof system permits the use of constraint algorithms, described in Chapter 4, which efficiently impose modularity directly in the structure of proofs.

The distinctive advantage of this search strategy, in contrast to previous efficient and modular designs for logic programming or contextual reasoning, is in allowing nested, modular existential quantifiers and disjunctions. These indefinite constructs get a simple and powerful treatment in DIALUP; this treatment underlies the construction and executions of indefinite specifications throughout the dissertation. For a start, as outlined in Chapter 5, DIALUP's treatment of these constructs makes for highly competitive performance on benchmark theorem proving problems involving modularity and disjunction.

10.1.2 Constructing Indefinite Modal Specifications

The task of reasoning about action and knowledge, explored in Part II, finds a range of further uses for indefinite specifications. For example, some sequences of actions achieve their effects despite possible variation in the state in which they occur. Think of dunking two packages—one of which is a bomb. As shown in Chapter 6, an indefinite characterization of this variation, using disjunction, may be necessary to calculate the effects of these actions.

Domains with information-gathering actions, meanwhile, provide another motivation for indefinite specifications. As described in Chapter 7, partial information, encoded using existential quantifiers and disjunctions, is needed in such domains to abstract the knowledge an agent can obtain from sensing actions. These characterizations provide the logical foundation for an agent's ability to make choices later in the plan that draw on the information the agent has learned earlier.

Indefinite characterizations of knowledge allow one agent to assess information that it lacks but that another agent has. Planning is a special case of this—where an agent assesses the information available to itself in the future. Conversation requires this ability more generally. We ran across two simple examples of this in Chapter 9. To ask a question and expect an answer requires knowing in detail how one's partner's knowledge may surpass one's own. Likewise, providing an instruction that one's partner knows how to execute may involve describing the instruction in terms that the partner will be able to interpret concretely but that remain indefinite to oneself.

As conversational agents start to think ahead about what they say and what they do, reasoning about one's own knowledge for planning and reasoning about other's knowledge will both come into play. Ultimately, a flexible conversational agent, of the kind envisioned for example [Cassell *et al.*, 1994a], must combine both types of reasoning. With such reasoning, it will be able to trade off independent action in the world (including ordinary actions and sensing actions) against joint actions agreed to (or, in the case of questions and answers, accomplished) in conversation. In formulating and choosing among such options, the conversational agent's reasoning can be expected to build on and extend the indefinite characterizations of planning, knowledge and action developed here.

Modularity itself has a further role in planning. Modular specifications can describe the independence of planning tasks in the domain and can reduce the number and kinds of interactions among tasks that an agent needs to plan for. Chapter 8 showed how modularity can reduce the search space in planning and facilitate the construction of more compact, natural plans.

10.1.3 Putting Indefinite Modal Specifications into Practice

We saw in Chapter 9 how the guidelines for developing indefinite modal specifications developed in Part II and the algorithms for executing those specifications developed in Part I can be put into practice. We examined the particular case of the SPUD system for NLG. SPUD represents the common ground in conversation using modal logic. This specification describes the sources of information that constitute the private expertise and shared background of participants in conversation. The specification includes not just concrete details but indefinite statements that describe the information available to one agent but not to the other. SPUD invokes DIALUP at every stage of generation, to consult this specification, monitor its progress towards its goals, and check the fit between its utterance and the context. SPUD's queries involve not just simple checks of what is true, but complex assessments formulated in terms of hypothetical updates to the common ground, nested modal operators and indefinite constructs. SPUD uses all of DIALUP's expressive range.

SPUD's incremental appeals to modal inference enable brevity in descriptions of actions; they allow SPUD to check that what it intends to communicate will be correctly recovered by the hearer from a compact description. Indefinite descriptions and queries, in particular, allow reference to discourse markers and make it possible for the speaker, in identifying abstract actions, to recognize that the hearer can realize those actions concretely. The resulting flexibility means that SPUD can start from independently-motivated descriptions of actions and still output concise, interesting sentences.

10.2 Issues and Problems

The breadth of this investigation inevitably precludes a definitive treatment of many of the points it raises. I mention here only a few of the most pressing issues and problems.

One problem is to achieve a more systematic understanding of the relative advantages of modularity and flexible search order across a range of problems in modal deduction. The results of Chapter 5 suggest that such understanding can come only with a thorough study of the different sources of difficulty in modal search—as encountered both in random problems and in the specifications designed for concrete applications. Thus it can only develop in parallel with the development of new ways to design hard model deduction problems and growth in use of modal languages in practice.

A different challenge is to reconcile default reasoning with the epistemic reasoning needed to describe conditional and parameterized actions in plans. Here one tack would be to follow the formalism of argumentation introduced in Chapter 6—with the aim of connecting with threat-based approaches to reasoning about conditional and parameterized actions as in [Golden and Weld, 1996]. To nail down such an extension would require not just extending relations among arguments, such as attack, but also redefining the intended models which exhibit common-sense inertia in the epistemic arena. To account for the modularity of independent planning subproblems will require still further extensions. Indeed, hardly any research has investigated the balance between ignoring what may have been true and ignoring what may have happened that must go into a description of inertia across modular subproblems. Faced with the difficulty of these successive reformulations of argumentation, an alternative would be to build on a different approach to default reasoning, with the hope of obtaining definitions and results which generalize more straightforwardly across domains. [Thomason, 1998] represents a preliminary study of one such framework, based on applying circumscription to a multimodal logic.

Future work is also needed to put SPUD to broader use, exploring the range of input representations, linguistic descriptions and models of interpretation that can be informed by modal specifications and reasoning. A relatively straightforward starting point would be to build a larger grammar fragment and look at the treatment of a greater variety of actions and instructions. Ongoing analysis of the F-16 instruction corpus for SPUD (referenced in Chapter 1), conducted with Martha Palmer, Christine Doran and Tonia Bleam, encompasses only a dozen verbs. There is obviously scope for more, and plenty of room for surprises in extending the preliminary success of SPUD to a more substantial domain of actions.

A more ambitious goal is to reconcile SPUD's incremental construction and evaluation of sentence interpretation with a default theory of knowledge and action. This may raise new problems and require new mechanisms, for example by introducing the possibility of inaccurate inferences that can be signaled by the absence of information. For such problems, a formalism based on abduction—in which hearer and speaker explicitly represent and agree on their shared assumptions about interpretation—may be preferable to a default logic based on arbitration among explicit defeasible rules. [Thomason and Hobbs, 1997] offer a first look at using abduction in NLG, but their proposal is silent on important computational concerns. Reference resolution is a good example. On the abductive view of [Hobbs *et al.*, 1988], a referring expression can potentially be explained by any assumption which supplies a discourse entity for it to refer to. The correct referent is found only because most of these hypotheses are inconsistent. But detecting and managing these inconsistencies is a much more open-ended computational task than the intersective constraint resolution required for efficient NLG. For problems such as reference, then, explicit reference to state of common ground, as represented in SPUD and DIALUP, seems to provide a complementary expressive resource that can be usefully combined with an abductive view of interpretation.

Substantial and different as these projects are, all fit into an overarching and ultimate one: to provide a computational account of the generation of precise, concise and natural sequences of sentences—describing reasons to carry out sequences of actions in the coherent language people use and expect. This project not only relies on the more basic efforts just described; it also brings exciting issues of its own to explore. Among the most important is how to lift and connect approaches to content and inference from sentences to discourse more generally. To do so, we must represent the inferential connections we recover in discourse at the level of content, perhaps by continuing to explore a connection between plans and reasoning and formalizations of argument as in [Ferguson, 1995]. We must also represent the basis for these inferential connections at the level of semantics and other linguistic structures—a view of cue words and other connectives as referential descriptions, as in [Knott, 1996], suggests that we can go far by emphasizing the continuity between sentences and broader discourse. By exploring these connections between such representations—in an ever more concrete and computational way—we can hope to more tightly reconcile the representations of language and computational logic, and to thereby obtain a more thorough and compelling model of what people put into conversation, and what they get out of it.

10.3 Closing Statement

The different sections of this work draw on and extend the contributions of researchers in different communities. Part I is mathematics: it shows how entities governed by an equational theory can serve as indicators of modularity in modal proofs, and how in key cases proof systems can exploit this abstraction to implement the modularity of modal logic simply, powerfully and flexibly. Part II is artificial intelligence: it describes and motivates tools using these algorithms that can be used for practical reasoning. And Part III is computational linguistics: it shows how the computation of sentence interpretation can benefit not only from declarative representations of lexicon and grammar but also from declarative representations of the underlying domain of knowledge and action which the sentence describes.

These algorithms thus connect various theoretical observations to the real requirements of scaling up reasoning and language to NLG problems of practical size. With this connection, the trends coalesce to form a single, coherent argument about knowledge representation and reasoning in natural language processing: formal foundations, linguistic explanations, efficient deduction and natural specifications do not have to be sacrificed to achieve logical representations with simple syntax and simple translations from natural language.

Bibliography

[Aho *et al.*, 1981] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal of Computation*, 10(3):405–421, 1981.

[Allen and Ferguson, 1994] James F. Allen and George Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–579, 1994.

[Allen and Hayes, 1989] James F. Allen and Patrick J. Hayes. Moments and points in an interval-based temporal logic. *Computational Intelligence*, 5(4):225–238, 1989.

[Allen, 1983] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[Andreoli, 1992] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[Andrews, 1981] Peter B. Andrews. Theorem proving via general matings. *Journal of the Association for Computing Machinery*, 28(2):193–214, 1981.

[Appelt, 1985] Douglas Appelt. *Planning English Sentences*. Cambridge University Press, Cambridge England, 1985.

[Auffray and Enjalbert, 1992] Yves Auffray and Patrice Enjalbert. Modal theorem proving: an equational viewpoint. *Journal of Logic and Computation*, 2(3):247–295, 1992.

[Badler and others, 1998] Norm Badler et al. Automating maintenance instructions. Final Report to Air Force HRGA, March 1998.

[Baldoni *et al.*, 1993] Matteo Baldoni, Laura Giordano, and Alberto Martelli. A multimodal logic to define modules in logic programming. In *ILPS*, pages 473–487, 1993.

[Baldoni *et al.*, 1996] Matteo Baldoni, Laura Giordano, and Alberto Martelli. A framework for modal logic programming. In M. Maher, editor, *JICSLP 96*, pages 52–66. MIT Press, 1996.

[Ballim and Wilks, 1991] Afzal Ballim and Yorick Wilks. *Artificial Believers: the ascription of belief*. Lawrence Erlbaum, Hillsdale, NJ, 1991.

[Ballim *et al.*, 1991] Afzal Ballim, Yorick Wilks, and John Barnden. Belief ascription, metaphor, and intensional identification. *Cognitive Science*, 15:133–171, 1991.

[Baral and Gelfond, 1993] Chitta Baral and Michael Gelfond. Representing concurrent actions in extended logic programming. In *IJCAI*, pages 866–871, 1993.

[Baral and Gelfond, to appear] Chitta Baral and Michael Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, to appear.

[Baral, to appear] Chitta Baral. Relating logic programming theories of actions and partial order planning. *Annals of Mathematics and Artificial Intelligence*, to appear.

[Barrett and Weld, 1994] Anthony Barrett and Daniel S. Weld. Task-decomposition via plan parsing. In *AAAI*, 1994.

[Barwise and Etchemendy, 1987] Jon Barwise and John Etchemendy. *The Liar: An Essay on Truth and Circularity*. Oxford, New York, 1987.

[Barwise, 1988] Jon Barwise. Three views of common knowledge. In *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 365–379. Morgan Kaufmann, 1988.

[Bibel, 1982] Wolfgang Bibel. *Automated Theorem Proving*. Vieweg, Braunschweig, 1982.

[Bibel, 1993] Wolfgang Bibel. *Deduction: Automated Logic*. Academic Press, London, 1993.

[Biermann *et al.*, 1993] Alan W. Biermann, Curry I. Guinn, Richard Hipp, and Ronnie W. Smith. Efficient collaborative discourse: A theory and its implementation. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 177–181, March 1993.

[Bourne, 1998] Juliet Bourne. Generating effective instructions: Knowing when to stop. PhD Thesis Proposal, Department of Computer & Information Science, University of Pennsylvania, July 1998.

[Boyer and Moore, 1972] R. S. Boyer and J. S. Moore. The sharing of structure in theorem-proving programs. In B. Meltzer and D. Michie, editors, *Machine Intelligence 7*, pages 101–116. Edinburgh University Press, 1972.

[Bratman *et al.*, 1988] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

[Bratman, 1987] Michael E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambrdige, MA, 1987.

[Buvač *et al.*, 1995] Saša Buvač, Vanja Buvač, and Ian A. Mason. Metamathematics of contexts. *Fundamenta Informaticae*, 23(3), 1995.

[Carenini *et al.*, 1994] Giuseppe Carenini, Vibhu O. Mittal, and Johanna D. Moore. Generating patient specific interactive explanations. In *Proceedings of Fourth International Conference on User Modeling*, Hyannis, MA, 1994.

[Cassell *et al.*, 1994a] Justine Cassell, Catherine Pelachaud, Norm Badler, Mark Steedman, Brett Achorn, Tripp Becket, Brett Douville, Scott Prevost, and Matthew Stone. Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *SIGGRAPH*, pages 413–420, 1994.

[Cassell *et al.*, 1994b] Justine Cassell, Matthew Stone, Brett Douville, Scott Prevost, Brett Achorn, Mark Steedman, Norm Badler, and Catherine Pelachaud. Modeling the interaction between speech and gesture. In *Proceedings of the Cognitive Science Society*, 1994.

[Chellas, 1980] Brian F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, Cambridge, 1980.

[Chu-Carroll and Carberry, 1995] Jennifer Chu-Carroll and Sandra Carberry. Response generation in collaborative negotiation. In *Proceedings of ACL*, pages 136–143, 1995.

[Clark and Marshall, 1981] Herbert H. Clark and Catherine R. Marshall. Definite reference and mutual knowledge. In Aravind K. Joshi, Bonnie Lynn Webber, and Ivan Sag, editors, *Elements of Discourse Understanding*, pages 10–63. Cambridge University Press, Cambridge, 1981.

[Clark, 1996] Herbert H. Clark. *Using Language*. Cambridge University Press, Cambridge, UK, 1996.

[Clocksin and Mellish, 1994] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer, New York, 4th edition, 1994.

[Currie and Tate, 1991] K. Currie and A. Tate. O-plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[D'Agostino and Mondadori, 1994] Marcello D'Agostino and Marco Mondadori. The taming of the cut. classical refutations with analytic cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[D'Agostino, 1992] Marcello D'Agostino. Are tableaux an improvement on truth-tables. *Journal of Logic, Language and Information*, 1:235–252, 1992.

[Dale and Haddock, 1991] Robert Dale and Nicholas Haddock. Content determination in the generation of referring expressions. *Computational Intelligence*, 7(4):252–265, 1991.

[Dale, 1989] Robert Dale. *Generating Referring Expressions in a Domain of Objects and Processes*. PhD thesis, Centre for Cognitive Science, University of Edinburgh, 1989.

[Dale, 1992] Robert Dale. *Generating Referring Expressions*. MIT Press, Cambridge MA, 1992.

[Davis, 1994] Ernest Davis. Knowledge preconditions for plans. *Journal of Logic and Computation*, 4(5):721–766, 1994.

[Debart *et al.*, 1992] Françoise Debart, Patrice Enjalbert, and Madeleine Lescot. Multimodal logic programming using equational and order-sorted logic. *Theoretical Computer Science*, 105:141–166, 1992.

[Douville *et al.*, 1996] Brett Douville, Libby Levison, and Norm Badler. Task level object grasping for simulated agents. *Presence*, 5(4):416–430, 1996.

[Dung, 1993] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotic reasoning and logic programming. In *IJCAI*, pages 852–857, 1993.

[Dyckhoff, 1992] Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57(3):795–807, 1992.

[Eshghi, 1988] K. Eshghi. Abductive planning with event calculus. In *Fifth International Conference on Logic Programming*, page 562, 1988.

[Etzioni *et al.*, 1992] Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In *Proceedings of KR-92*, pages 115–125, 1992.

[Fagin *et al.*, 1995] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge.* MIT Press, Cambridge MA, 1995.

[Fariñas del Cerro, 1986] Luis Fariñas del Cerro. MOLOG: A system that extends PROLOG with modal logic. *New Generation Computing*, 4:35–50, 1986.

[Ferguson and Allen, 1994] George Ferguson and James F. Allen. Arguing about plans: Plan representation and reasoning for mixed-initiative planning. In Kristian Hammond, editor, *Proceedings of the Second International Conference on A.I. Planning Systems*, pages 43–48, 1994.

[Ferguson, 1995] George M. Ferguson. *Knowledge Representation and Reasoning for Mixed-Initiative Planning*. PhD thesis, University of Rochester, 1995.

[Fitting, 1972] M. Fitting. Tableau methods of proof for modal logics. *Notre Dame Journal of Formal Logic*, 13(2), 1972.

[Fitting, 1983] Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library*. D. Reidel, Dordrecht, 1983.

[Frisch and Scherl, 1991] Alan M. Frisch and Richard B. Scherl. A general framework for modal deduction. In *Proceedings of KR*, pages 196–207. Morgan Kaufmann, 1991.

[Frisch, 1991] Alan M. Frisch. The substitutional framework for sorted deduction: fundamental results on hybrid reasoning. *Artificial Intelligence*, 49:161–198, 1991.

[Gallier, 1986] Jean H. Gallier. *Logic for Computer Science: Foundations of Automated Theorem Proving*. Harper and Row, New York, 1986.

[Gallier, 1993] Jean Gallier. Constructive logics. I. A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110(2):249–339, 1993.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[Gasquet, 1993] Olivier Gasquet. Automated deduction for a multi-modal logic of time and knowledge. In *Automated Deduction in Nonstandard Logics*, pages 38–45, Menlo Park, CA, October 1993. AAAI, AAAI Press.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, 1988.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–387, 1991.

[Gelfond and Lifschitz, 1993] Michael Gelfond and Vladimir Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2–4):301–321, 1993.

[Gelfond, 1994] Michael Gelfond. Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*, pages 98–116, 1994.

[Gentzen, 1935] Gerhard Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210; 405–431, 1935.

[Gertner and Webber, 1998] Abigail S. Gertner and Bonnie L. Webber. TraumaTIQ: Online decision support for trauma management. *IEEE Intelligent Systems*, 13(1), 1998.

[Ginsberg, 1995] Matthew L. Ginsberg. Approximate planning. *Artificial Intelligence*, 76:89–123, 1995.

[Giordano and Martelli, 1994] Laura Giordano and Alberto Martelli. Structuring logic programs: A modal approach. *Journal of Logic Programming*, 21:59–94, 1994.

[Giunchiglia and Sebastiani, 1996] Fausto Giunchiglia and Roberto Sebastiani. Building decision procedures for modal logics from propositional decision procedures—the case study of modal *k*(*m*). In *CADE-13*. Springer, 1996.

[Giunchiglia *et al.*, 1997] Fausto Giunchiglia, Marco Roveri, and Roberto Sebastiani. A new method for testing decision procedures in modal logics. In *CADE 14*. Springer, 1997.

[Giunchiglia *et al.*, 1998] Enrico Giunchiglia, Fausto Giunchiglia, Roberto Sebastiani, and Armando Tacchella. More evaluation of decision procedures for modal logics. In *KR-98*, 1998.

[Golden and Weld, 1996] Keith Golden and Daniel Weld. Representing sensing actions: the middle ground revisited. In *KR*, 1996.

[Goldman and Boddy, 1996] Robert P. Goldman and Mark S. Boddy. Expressive planning and explicit knowledge. In *AIPS*, pages 110–117, 1996.

[Goldman, 1970] Alvin I. Goldman. *A Theory of Human Action*. Prentice Hall, Englewood Cliffs, NJ, 1970.

[Groenendijk and Stokhof, 1990a] Jeroen Groenendijk and Martin Stokhof. Dynamic Montague grammar. In L. Kálmán and L Pólos, editors, *Papers from the Second Symposium on Logic and Language*, pages 3–48, Budapest, 1990. Akadémiai Kiadó.

[Groenendijk and Stokhof, 1990b] Jeroen Groenendijk and Martin Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1990.

[Grosz and Sidner, 1986] Barbara Grosz and Candace Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12:175–204, 1986.

[Grosz *et al.*, 1995] Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–225, 1995.

[Guha and Lenat, 1990] Ramanathan V. Guha and Douglas B. Lenat. Cyc: A midterm report. *AI Magazine*, 11(3):32–59, 1990.

[Guha, 1991] Ramanathan V. Guha. *Contexts: A Formalization and Some Applications*. PhD thesis, Stanford, 1991. TR STAN-CS-91-1399.

[Gundel *et al.*, 1993] Jeanette K. Gundel, Nancy Hedberg, and Ron Zacharski. Cognitive status and the form of referring expressions in discourse. *Language*, 69(2):274–307, 1993.

[Halpern and Moses, 1985a] Joseph Y. Halpern and Yoram Moses. A guide to the modal logics of knowledge and belief: preliminary draft. In *Proceedings of the Ninth IJCAI*, pages 480–490, 1985.

[Halpern and Moses, 1985b] Joseph Y. Halpern and Yoram Moses. A guide to the modal logics of knowledge and belief: preliminary draft. In *9th International Joint Conference on Artificial Intelligence*, pages 480–490, 1985.

[Hanks and McDermott, 1987] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3), 1987.

[Hanks and Weld, 1995] Steve Hanks and Daniel S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, 1995.

[Heim, 1982] Irene Heim. *The Semantics of Definite and Indefinite Noun Phrases*. PhD thesis, University of Massachusetts, Amherst, 1982. Published 1987 by Garland Press.

[Herbrand, 1971] Jacques Herbrand. *Logical Writings*. Harvard University Press, Cambridge, 1971.

[Hintikka, 1962] Jaakko Hintikka. *Knowledge and Belief*. Cornell University Press, 1962.

[Hintikka, 1971] Jaakko Hintikka. Semantics for propositional attitudes. In Linsky, editor, *Reference and Modality*, pages 145–167. Oxford, 1971.

[Hobbs *et al.*, 1988] Jerry R. Hobbs, Mark Stickel, Douglas Appelt, and Paul Martin. Interpretation as abduction. In *Proceedings of ACL*, pages 95–103, 1988.

[Hobbs, 1985] Jerry R. Hobbs. Ontological promiscuity. In *Proceedings of ACL*, pages 61–69, 1985.

[Hodas and Miller, 1994] Joshua S> Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.

[Hopcroft and Ullman, 1973] J. E. Hopcroft and J. D. Ullman. Set merging algorithms. *SIAM Journal of Computation*, 2:294–303, 1973.

[Howard, 1980] W. A. Howard. The formulae-as-types notion of construction. In *To H. B. Curry: essays on combinatory logic, lambda calculus, and formalism*, pages 479–490. Academic Press, New York, 1980.

[Hughes and Cresswell, 1968] G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic*. Methuen, London, 1968.

[Hustadt and Schmidt, 1997] Ullrich Hustadt and Renate A. Schmidt. On evaluating decision procedures for modal logic. Technical Report MPI-I-97-2-003, Max-Plank-Institut für Informatik, Febrary 1997.

[Jackendoff, 1990] Ray S. Jackendoff. *Semantic Structures*. MIT Press, Cambridge, MA, 1990.

[Jackson and Reichgelt, 1987] Peter Jackson and Han Reichgelt. A general proof method for first-order modal logic. In *Proceedings of IJCAI*, pages 942–944, 1987.

[Joshi *et al.*, 1975] Aravind K. Joshi, L. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of the Computer and System Sciences*, 10:136–163, 1975.

[Joshi, 1987] Aravind K. Joshi. The relevance of tree adjoining grammar to generation. In Gerard Kempen, editor, *Natural Language Generation*, pages 233–252. Martinus Nijhoff Press, Dordrecht, The Netherlands, 1987.

[Kambhampati and Hendler, 1992] Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Aritificial Intelligence*, 55:193–258, 1992.

[Kamp and Reyle, 1993] Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer, Boston, 1993.

[Kamp, 1981] Hans Kamp. A theory of truth and semantic representation. In Jeroen Groenendijk, Theo Janssen, and Martin Stokhof, editors, *Truth, Interpretation and Information*. Foris, Dordrecht, 1981.

[Kanger, 1957] Stig Kanger. *Provability in Logic*, volume 1 of *Stockholm Studies in Philosophy*. Almqvist and Wiksell, Stockholm, 1957.

[Kanovich, 1990] Max I. Kanovich. Efficient program synthesis in computational models. *Journal of Logic Programming*, 9:159–177, 1990.

[Kapur and Narendran, 1986] Deepak Kapur and Paliath Narendran. NP-completeness of the set unification and matching problems. In *CADE 8*, 1986.

[Kapur and Narendran, 1992] Deepak Kapur and Paliath Narendran. Complexity of unification problems with associative-commutative operators. *Journal of Automated Reasoning*, 9:261–288, 1992.

[Kleene, 1951] Stephen C. Kleene. Permutation of inferences in Gentzen's calculi LK and LJ. In *Two papers on the predicate calculus*, pages 1–26. American Mathematical Society, Providence, RI, 1951.

[Knott, 1996] Alistair Knott. *A Data-Driven Methodology for Motivating a Set of Coherence Relations*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1996.

[Kobsa and Pohl, 1995] Alfred Kobsa and Wolfgang Pohl. The user modeling shell system BGP-MS. *User Modeling and User-Adapted Interaction*, 4(2):59–106, 1995.

[Koehler, 1994] Jana Koehler. Avoiding pitfalls in case-based planning. In *AIPS*, pages 104–109. AAAI, 1994.

[Konolige, 1988] Kurt Konolige. Defeasible argumentation in reasoning about events. In Z. W. Ras and L. Saitta, editors, *Methodologies for Intelligent Systems 3*, pages 380–390, 1988.

[Korn and Kreitz, 1997] Daniel S. Korn and Christoph Kreitz. Deciding intuitionistic propositionallogic via translation into classical logic. In William McCune, editor, *CADE-14*, number 1249 in LNAI, pages 131–145, Berlin, 1997. Springer.

[Kowalski and Sergot, 1986] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[Kowalski, 1992] Robert Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146, 1992.

[Kripke, 1963] Saul A. Kripke. Semantical analysis of modal logic. I. Normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[Ladner, 1977] Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977.

[Levesque, 1996] Hector J. Levesque. What is planning in the presence of sensing? In *Proceedings of AAAI*, pages 1139–1146, 1996.

[Levison, 1996] Libby Levison. *Connecting Planning and Acting via Object-Specific Reasoning*. PhD thesis, University of Pennsylvania, 1996. MS-CIS-96-08.

[Lewis, 1969] David K. Lewis. *Convention: A Philosophical Study*. Harvard University Press, Cambridge, MA, 1969.

[Lewis, 1979] David Lewis. Scorekeeping in a language game. In *Semantics from Different Points of View*, pages 172–187. Springer Verlag, Berlin, 1979.

[Lifschitz, 1987] Vladimir Lifschitz. Formal theories of action. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence*, pages 35–57. Morgan Kaufmann, 1987.

[Lin and Shoham, 1989] Fangzhen Lin and Yoav Shoham. Argument systems: a uniform basis for nonmonotonic reasoning. In *KR*, pages 245–255, 1989.

[Lin and Shoham, 1991] Fangzhen Lin and Yoav Shoham. Provably correct theories of actions: preliminary report. In *AAAI*, pages 349–354, 1991.

[Lincoln and Shankar, 1994] P. D. Lincoln and N. Shankar. Proof search in first-order linear logic and other cut-free sequent calculi. In *LICS*, pages 282–291, 1994.

[Lobo *et al.*, 1997] Jorge Lobo, Gisela Mendez, and Stuart R. Taylor. Adding knowledge to the action description language $\dashv$. In *Proceedings of AAAI*, pages 454–459, 1997.

[Loveland, 1991] Donald W. Loveland. Near-horn Prolog and beyond. *Journal of Automated Reasoning*, 7:1–26, 1991.

[Mackworth, 1987] Alan Mackworth. Constraint Satisfaction. In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 205–211. John Wiley and Sons, 1987.

[Martelli and Montanari, 1982] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[McAllister and Rosenblitt, 1991] David McAllister and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI*, pages 634–639, 1991.

[McCain and Turner, 1997] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proceedings of AAAI*, pages 460–467, 1997.

[McCarthy and Buvač, 1994] John McCarthy and Saša Buvač. Formalizing context (expanded notes). Technical Report STAN-CS-TN-94-13, Stanford University, 1994.

[McCarthy and Hayes, 1969] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 473–502. Edinburgh University Press, Edinburgh, 1969.

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[McCarthy, 1987] John McCarthy. Generality in artificial intelligence. *Communications of the ACM*, 30(12):1030–1035, 1987.

[McCarthy, 1993] John McCarthy. Notes on formalizing context. In *IJCAI*, pages 555–560, 1993.

[McCune, 1994] William W. McCune. Otter 3.0 reference manual and guide. Technical Report ANL–94/6, Argonne National Laboratory, 1994.

[Mellish *et al.*, 1998] Chris Mellish, Michael O'Donnell, Jon Oberlander, and Alistair Knott. An architecture for opportunistic text generation. In *9th International Workshop on Natural Language Generation*, 1998.

[Meteer, 1991] Marie W. Meteer. Bridging the generation gap between text planning and linguistic realization. *Computational Intelligence*, 7(4):296–304, 1991.

[Miller *et al.*, 1991] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Miller, 1989] Dale Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6(1–2):79–108, 1989.

[Miller, 1994] Dale Miller. A multiple-conclusion meta-logic. In S. Abramsky, editor, *Proceedings of the International Symposium on Logics in Computer Science*, pages 272–281, 1994.

[Mints, 1992] Grigori Mints. *A Short Introduction to Modal Logic*. Number 30 in CSLI Lecture Notes. CSLI, 1992.

[Missiaen *et al.*, 1995] Lode Missiaen, Maurice Bruynooghe, and Marc Deneker. CHICA, an abductive planning system based on event calculus. *Journal of Logic and Computation*, 5(5):579–602, 1995.

[Mitchell *et al.*, 1992] D Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *AAAI 92*, pages 459–465, 1992.

[Moens and Steedman, 1988] Marc Moens and Mark Steedman. Temporal ontology and temporal reference. *Computational Linguistics*, 14(2):15–28, 1988.

[Moore, 1985a] Robert C. Moore. A formal theory of knowledge and action. In Jerry R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*, pages 319–358. Ablex, Norwood NJ, 1985.

[Moore, 1985b] Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1), 1985.

[Morgenstern, 1987] Leora Morgenstern. Knowledge preconditions for actions and plans. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 867–874, Milan Italy, 1987.

[Muskens, 1996] Reinhard Muskens. Combining Montague semantics and discourse representation. *Linguistics and Philosophy*, 19(2):143–186, 1996.

[Nadathur and Loveland, 1995] Gopalan Nadathur and Donald W. Loveland. Uniform proofs and disjunctive logic programming. In *LICS*, pages 148–155, 1995.

[Ohlbach, 1991] H. J. Ohlbach. Semantics-based translation methods for modal logics. *Journal of Logic and Computation*, 1(5):691–746, 1991.

[Ohlbach, 1993] Hans Jürgen Ohlbach. Optimized translation of multi modal logic into predicate logic. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning*, volume 698 of *LNCS*, pages 253–264. Springer, Berlin, 1993.

[Onishi and Matsumoto, 1957] M. Onishi and K. Matsumoto. Gentzen method in modal calculi I. *Osaka Mathematical Journal*, 9:113–130, 1957.

[Onishi and Matsumoto, 1959] M. Onishi and K. Matsumoto. Gentzen method in modal calculi II. *Osaka Mathematical Journal*, 11:115–120, 1959.

[Otten and Kreitz, 1996] Jens Otten and Christoph Kreitz. T-string-unification: unifying prefixes in non-classical proof methods. In *TABLEAUX 96*, volume 1071 of *LNAI*, pages 244–260, Berlin, 1996. Springer.

[Pelletier, 1986] Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986.

[Penberthy and Weld, 1992] J. S. Penberthy and D. S. Weld. UCPOP: a sound, complete partial order planner for ADL. In *KR*, pages 103–114, 1992.

[Pollack, 1990] Martha E. Pollack. Plans as complex mental attitudes. In Philip Cohen, Jerry Morgan, and Martha Pollack, editors, *Intentions in Plans and Communication*, pages 77–103. MIT Press, Cambridge MA, 1990.

[Pollack, 1991] Martha Pollack. Overloading intentions for efficient practical reasoning. *Noûs*, 25:513–536, 1991.

[Pollack, 1992] Martha E. Pollack. The uses of plans. *Artificial Intelligence*, 57:43–68, 1992.

[Pollock, 1992] John L. Pollock. How to reason defeasibly. *Artificial Intelligence*, 57(1):1–42, 1992.

[Prince, 1981] Ellen Prince. Toward a taxonomy of given-new information. In P. Cole, editor, *Radical Pragmatics*. Academic Press, 1981.

[Prior, 1967] Arthur N. Prior. *Past, Present and Future*. Clarendon Press, Oxford, 1967.

[Rambow and Korelsky, 1992] Owen Rambow and Tanya Korelsky. Applied text generation. In *ANLP*, pages 40–47, 1992.

[Reed *et al.*, 1992] David W. Reed, Donald W. Loveland, and Bruce T. Smith. The near-horn approach to disjunctive logic programming. In *Proceedings of the Second Workshop on Extensions of Logic Programming*, Berlin, 1992. Springer Verlag.

[Reiter and Dale, 1997] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3:57–88, 1997.

[Reiter *et al.*, 1995] Ehud Reiter, Chris Mellish, and John Levine. Automatic generation of technical documentation. *Applied Artificial Intelligence*, pages 259–287, 1995.

[Reiter, 1980] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 12:81–132, 1980.

[Reiter, 1991] Raymond Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[Reiter, 1994] Ehud Reiter. has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Seventh International Workshop on Natural Language Generation*, pages 163–170, June 1994.

[Reiter, 1996] Ray Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *KR*, 1996.

[Robinson, 1965a] J. A. Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.

[Robinson, 1965b] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–45, 1965.

[Sacerdoti, 1975] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of IJCAI*, pages 206–214, 1975.

[Sahlin *et al.*, 1992] Dan Sahlin, Torkel Franzén, and Seif Haridi. An intuitionistic predicate logic theorem prover. *Journal of Logic and Computation*, 2(5):619–656, 1992.

[Sandewall, 1994a] Erik Sandewall. *Features and Fluents: Representation of Knowledge about Dynamical Systems*. Oxford University Press, New York, 1994.

[Sandewall, 1994b] Erik Sandewall. The range of applicability of some non-monotonic logics for strict inertia. *Journal of Logic and Computation*, 4(5):581–615, 1994.

[Schabes, 1990] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, Computer Science Department, University of Pennsylvania, 1990.

[Schank and Abelson, 1977] Roger Schank and Robert Abelson. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum, Hillsdale, N.J., 1977.

[Scherl and Levesque, 1993] Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In *AAAI*, pages 689–695, 1993.

[Schmidt, 1996] Renate A. Schmidt. Resolution is a decision procedure for many propositional modal logics. In *AiML*, 1996.

[Schmidt, 1998] Renate A. Schmidt. E-Unification for subsystems of S4. In *Rewriting Techniques and Applications*, 1998.

[Schubert, 1990] Lenhart K. Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H. E. Kyburg, R. P. Loui, and G. N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, Boston, 1990.

[Schulz, 1993] Klaus U. Schulz. Word unification and transformation of generalized equations. *Journal of Automated Reasoning*, 11(2):149–184, 1993.

[Shanahan, 1989] Murray Shanahan. Prediction is deduction but explanation is abduction. In *Proceedings of IJCAI*, pages 1055–1060, 1989.

[Shanahan, 1997] Murray Shanahan. Event calculus planning revisited. In *Proceedings of European Conference on Planning*, pages 390–402, 1997.

[Shieber *et al.*, 1990] Stuart Shieber, Gertjan van Noord, Fernando Pereira, and Robert Moore. Semantic-head-driven generation. *Computational Linguistics*, 16:30–42, 1990.

[Shoham, 1991] Yoav Shoham. Varieties of context. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press, 1991.

[Simari and Loui, 1992] Guillermo R. Simari and Ronald P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53:125–157, 1992.

[Slaney, 1994] John Slaney. MINLOG. Technical Report TR-ARP-12-94, Australian National University, 1994.

[Smullyan, 1973] Raymond M. Smullyan. A generalization of intuitionistic and modal logics. In Hugues Leblanc, editor, *Truth, Syntax and Modality*, pages 274–293. North-Holland, Amsterdam, 1973.

[Steedman, 1995] Mark Steedman. Dynamic semantics for tense and aspect. In *Proceedings of IJCAI*, pages 1292–1298, 1995.

[Steedman, 1997] Mark Steedman. Temporality. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 895–935. Elsevier, 1997.

[Stone and Doran, 1996] Matthew Stone and Christine Doran. Paying heed to collocations. In *International Natural Language Generation Workshop*, pages 91–100, 1996.

[Stone and Doran, 1997] Matthew Stone and Christine Doran. Sentence planning as description using tree-adjoining grammar. In *Proceedings of ACL*, pages 198–205, 1997.

[Stone and Webber, 1998] Matthew Stone and Bonnie Webber. Textual economy through close coupling of syntax and semantics. In *Proceedings of INLG*, 1998.

[Stone, 1992] Matthew Stone. ‘or’ and anaphora. In Chris Barker and David Dowty, editors, *SALT II*, Columbus, 1992. Ohio State University.

[Stone, 1997a] Matthew Stone. Applying theories of communicative action in natural language generation using logic programming. In *AAAI Fall Symposium on Communicative Action*, 1997.

[Stone, 1997b] Matthew Stone. Partial order reasoning for a nonmonotonic theory of action. In *AAAI Workshop on Theories of Action*, Providence, RI, 1997.

[Stone, 1998] Matthew Stone. Abductive planning with sensing. In *AAAI*, Madison, WI, 1998.

[Stone, to appear] Matthew Stone. Representing scope in intuitionistic deductions. *Theoretical Computer Science*, to appear. IRCS TR 97-14.

[Taylor *et al.*, 1996] Jasper Taylor, Jean Carletta, and Chris Mellish. Combining power with tractability in belief models. Technical Report HCRC RP-76, Human Communication Research Centre, 1996.

[Tennant, 1992] Neil Tennant. *Autologic*. Number 9 in Edinburgh Information Technology Series. Edinburgh University Press, Edinburgh, 1992.

[Thistlethwaite *et al.*, 1987] P. B. Thistlethwaite, M. A. McRobbie, and R. K. Meyer. *Automated Theorem-Proving in Non-classical Logics*. Wiley, New York, 1987.

[Thomason and Hobbs, 1997] Richmond H. Thomason and Jerry R. Hobbs. Interrelating interpretation and generation in an abductive framework. In *AAAI Fall Symposium on Communicative Action*, 1997.

[Thomason, 1998] Richmond H. Thomason. Intra-agent modality and nonmonotonic epistemic logic. In *Theoretical Aspects of Rationality and Knowledge (TARK)*, 1998.

[Troelstra and van Dalen, 1988] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics*, volume 1. North-Holland, Amsterdam, 1988.

[van Benthem, 1983] Johan F. A. K. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Naples, 1983.

[van Benthem, 1984] Johan van Benthem. Modal correspondence theory. In *Handbook of Philosophical Logic*, volume 2, pages 167–247. D. Reidel, Dordrecht, 1984.

[van der Sandt, 1992] Rob van der Sandt. Presupposition projection as anaphora resolution. *Journal of Semantics*, 9(2):333–377, 1992.

[van Gelder, 1986] Allen van Gelder. Negation as failure using tight derivations for general logic programs. In *Proceedings of the International Symposium on Logic Programming*, pages 127–139. IEEE Computer Society,, The Computer Society Press, 1986.

[van Gelder, 1989] Allen van Gelder. Negation as failure using tight derivations for general logic programs. *Journal of Logic Programming*, 6(1):109–133, 1989.

[Veloso, 1994] Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer, 1994.

[Walker, 1993] Lyn Walker. *Informational redundancy and resource bounds in dialogue*. PhD thesis, Department of Computer & Information Science, University of Pennsylvania, 1993. Institute for Research in Cognitive Science report IRCS-93-45.

[Wallen, 1990] Lincoln A. Wallen. *Automated Proof Search in Non-Classical Logics: Efficient Matrix Proof Methods for Modal and Intuitionistic Logics*. MIT Press, Cambridge, 1990.

[Ward, 1994] Nigel Ward. *A Connectionist Language Generator*. Ablex, Norwood, NJ, 1994.

[Webber *et al.*, 1995] Bonnie Webber, Norm Badler, Barbara Di Eugenio, Chris Geib, Libby Levison, and Michael Moore. Instructions, intentions and expectations. *Artificial Intelligence*, 73:253–259, 1995.

[Webber, 1998] Bonnie Webber. Instructing animated agents: Viewing language in behavioral terms. In Harry Bunt, Robert-Jan Beun, and Tijn Borghuis, editors, *Multimodal Human-Computer Communication: Systems, Techniques and Experiments*, volume 1374 of *LNAI*, pages 89–100. Springer, 1998.

[Wiedenbach *et al.*, 1996] Christoph Wiedenbach, Bernd Gaede, and Georg Rock. SPASS and FLOTTER, version 0.42. In M. A. McRobbie and J. A. Slaney, editors, *13th International Conference on Automated Deduction, CADE-13*, number 1104 in LNAI, pages 141–145, Berlin, 1996. Springer.

[Wilkins, 1988] D. E. Wilkins. *Practical Planning: Extending the Classical AI Paradigm*. Morgan Kaufmann, San Mateo, CA, 1988.

[Yang *et al.*, 1991] Gijoo Yang, Kathleen F. McCoy, and K. Vijay-Shanker. From functional specification to syntactic structures: systemic grammar and tree-adjoining grammar. *Computational Intelligence*, 7(4):207–219, 1991.

[Yang, 1990] Q. Yang. Formalizing planning knowledge for a hierarchical planner. *Computational Intelligence*, 6(1):12–24, 1990.

[Young *et al.*, 1994] R. Michael Young, Martha E. Pollack, and Johanna D. Moore. Decomposition and causality in partial-order planning. In *AIPS*, 1994.

[Young, 1997] R. Michael Young. *Generating Concise Descriptions of Complex Activities*. PhD thesis, University of Pittsburgh, 1997.