Efficient Constraints on Possible Worlds for Reasoning about Necessity

Matthew Stone

University of Pennsylvania
3401 Walnut Street, Suite 400A
Philadelphia, PA 19104-6228

May 1997

Site of the NSF Science and Technology Center for Research in Cognitive Science

IRCS Report 97--07

Efficient Constraints on Possible Worlds for Reasoning about Necessity[1]

Matthew Stone
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd St, Philadelphia PA 19104
matthew@linc.cis.upenn.edu

Summary

Modal logics offer natural, declarative representations for describing both the modular structure of logical specifications and the attitudes and behaviors of agents. The results of this paper further the goal of building practical, efficient reasoning systems using modal logics. The key problem in modal deduction is reasoning about the world in a model (or scope in a proof) at which an inference rule is applied—a potentially hard problem. This paper investigates the use of partial-order mechanisms to maintain constraints on the application of modal rules in proof search in restricted languages. The main result is a simple, incremental polynomial-time algorithm to correctly order rules in proof trees for combinations of K, K4, T and S4 necessity operators governed by a variety of interactions, assuming an encoding of negation using a scoped constant $\perp$. This contrasts with previous equational unification methods, which have exponential performance in general because they simply guess among possible intercalations of modal operators. The new, fast algorithm is appropriate for use in a wide variety of applications of modal logic, from planning to logic programming.

Content area: Reasoning Techniques—deduction, efficiency and complexity.

1 Introduction

The necessity operator of modal logic provides a natural declarative construct for specifying both content and search control: it can be used to describe change over time, to specify attitudes of agents like knowledge and belief, or simply to enforce modularity in complex specifications (Moore, 1985; Halpern and Moses, 1985; Giordano and Martelli, 1994). Automatic interpretation of modal specifications requires efficient reasoning mechanisms for modal logic. Despite recent advances by (Wallen, 1990; Ohlbach, 1991), efficient modal reasoning remains elusive. This paper identifies important new opportunities for performing tractable inference in modal logic. These results show for the first time how automatic systems for program synthesis, planning, and logic programming can use modal logic as a practical representation.

[1]Thanks to Mark Steedman for extensive comments and Tandy Warnow and Dale Miller for helpful discussion. This work was supported by an NSF graduate fellowship, and an IRCS graduate fellowship, as well as NSF grant IRI95-04372, ARPA grant N6601-94-C6043, and ARO grant DAAH05-94-G0426. This paper has been submitted to the *Journal of Logic and Computation*. May 12, 1997.

As reviewed in section 2, the key difficulty in modal reasoning is to capture the *scope* discipline on the use of formulas in modal proofs. Attempting to prove a necessity truth, or to reason from a possible truth, creates a scope in a deduction in which only other necessary truths can be used. Classic descriptions of these scopes, as in e.g. (Chellas, 1980), lead to an explosion in the search space for automated deduction. Systems using Hilbert-type axiom systems manage scope by a protocol that transfers valid inferences step-by-step from top-level into nested scopes; Gentzen-type sequent calculi perform all inference in each scope inside a contiguous region of a proof. These treatments of scope force proof-search algorithms to guess among alternatives before they have the information to determine which alternatives are possibly relevant. Some difficulties can be avoided by translating modal formulas into classical logic using the worlds and accessibility relations of Kripke semantics for modal logic (Kripke, 1963), but explicitly deriving full proofs of accessibility between worlds is still often unmanageable.

This paper investigates another description of modal inference. This system, pioneered by (Fitting, 1972; Smullyan, 1973; Wallen, 1990; Ohlbach, 1991), assigns a proof-theoretic abstraction of scope explicitly to formulas. Each formula is labeled by a string recording the sequence of embedded operators along the path to the scope where the formula holds. By allowing the label of a formula to be partially instantiated by unification as the formula is used during proof search, this procedure avoids the most severe drawbacks of deduction in earlier systems. However, this method involves complex and expensive equational unification processes which limit its practical use.

In this paper, I show how significantly better results can be achieved for K, K4, T and S4 modal logics by encoding negation and possibility in terms of a scoped propositional constant $\perp$. With this encoding, $\Box$ becomes the only modal connective: the encoding creates $\Box$-only languages. By analyzing the unification problems for proofs in $\Box$-only languages, I show that solutions respect the order in which terms representing scopes are introduced. In K, K4, T, and S4, this constraint resolves all essential ambiguities in unification of paths of accessibility. 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 path-based explicitly-scoped proof system plays an integral role in the statements and proofs of these results.

The organization of the rest of the paper is as follows. In the next section, I give an introduction to the proof theory of modal logic, its motivations, pitfalls and complexities. In section 3, I present the main proof-theoretic observation that underlies my results. A constraint algorithm exploiting this observation is presented in section 4. In section 5, I finish by considering the impact of these algorithms for a range of practical problems, including the synthesis of functional programs, temporal reasoning and automatic planning, and reasoning about agents.

2 Proofs in Modal Logic

A variety of formal systems describe the proofs of modal logics; the choice of proof system can have a profound impact on the difficulty of automatic proof construction. 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. 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 been misleadingly optimistic. The purpose of this section is to review these results briefly, to connect the intuitions behind new proof systems with the intuitions behind old ones, and to highlight the distinctive complexities that remain in modal deduction and how those relate to standard characterizations of the complexity of modal deduction. 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.

I begin in section 2.1 with an informal example intended to motivate modal scoping mechanisms and to introduce a key theme: how the meaning of modal operators comes from assumptions about *the relationships* among the scopes these operators introduce. Section 2.2 provides the inevitable technical necessities about language and notation, and introduces the different proof systems for modal logic concretely. Section 2.2.1 reviews the familiar axiomatic method for modal inference and its computational limitations. Section 2.2.2 describes structurally-scoped proof systems in the style of Gentzen, and the role of *impermutabilities* of inference in these systems in hindering automated deduction. Next, in 2.2.3, explicitly-scoped proof systems are introduced by way of a ground system, which is then *lifted* to a system using unification in 2.2.4. In this system, explicit scoping and unification frees search engines from the unnecessary commitment inherent in structurally-scoped proof methods or relational translations to classical logic. The lifted, explicitly-scoped system will be our focus for the remainder of the paper.

As section 2.3 shows, the success of this system in supporting techniques from classical theorem-proving is offset by the fact that the unification procedure it relies on to resolve scopes is intractable both in principle and in practice. The intractability of scope unification is a *local* problem that *compounds* the *global* intractability, implicit in PSPACE-completeness results (Ladner, 1977; Halpern and Moses, 1985), that proofs of propositional modal logic must in some cases be unreasonably large. While global problems with the possible size of proofs are familiar from ordinary first-order deduction without equality, this local problem has no analogue there. In fact, it is likely to be a more serious problem in practice. 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 assigns scopes to rules. Moreover, the ambiguities involved arise as a result of the equational theory governing scope terms, and are therefore difficult to avoid by judicious reformulation of logical statements (of the sort familiar to Prolog programmers).

2.1 Motivation

Formulas in first-order classical logic specify information about ordinary entities, while formulas in more expressive logics can also constrain how such information is to be used in reasoning. Many proposals for such expressive logics start, informally, by imposing a notion of *scope* on deductions. The force of this notion of scope is that two formulas must lie in the same scope to be combined in reasoning. Scopal restrictions on the use of formulas in inference play two important roles in knowledge representation. First, the expressive power helps describe complex domains concisely and correctly. For example, a scoped specification can describe agents' propositional attitudes, by ensuring that two facts will be combined only when they describe the content of a single attitude of a single agent. Each formula's scope records the agent and attitude it describes. Similarly, a scoped specification can describe multiple moments in time, by guaranteeing that only facts true at the same time can be combined in inference. The time at which a fact holds determines its scope. Second, the expressive power of imposing scopes in proofs offers a method to ensure that scoped specifications are modular and reusable. In a scoped language, when a body of knowledge forms a module, its logical interactions can be limited to facts in the same or compatible modules. On this view, the scope of a formula depends on the module in which it resides.

The different proposals of (Halpern and Moses, 1985; Morgenstern, 1987; Ballim et al., 1991; McCarthy and Buvač, 1994) offer methods to realize various notions of scopes in deductions in a computational setting. Although formalized differently (and to different degrees), 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. These features achieve strikingly similar effects across the different formalisms. It follows that reasoning about scopes is a central problem in implementing any of them.

A concrete example, adapted from (McCarthy and Buvač, 1994), illustrates the motivation for and the behavior of scopes in proofs. We will use it to introduce modal logic as a particular scoped representation, but at the same time to emphasize that practical applications of modal logic depend on the availability of flexible range of reasoning principles. That is why the discussion of modal proof systems in section 2.2 is parametrized for different modal operators and reasoning principles right from the start.

Example 1. 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. 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. To formalize this, one might choose a representation like this:

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

In this formula, $[\textsc{list}]A$ indicates that A is to be proved in a special scope where only the information in the company's catalogue can be taken into account (and likewise $[\textsc{spec}]$ for the Navy's specifications). Similarly, we can represent that GE and the Navy may have different, partial information about what is in the list and the specification, by introducing operators $[\textsc{ge}]$ and $[\textsc{navy}]$: $[\textsc{ge}]A$ indicates that A is to be proved taking only GE's information into account.

One way to model the operators $[\textsc{list}]$, $[\textsc{spec}]$, $[\textsc{ge}]$ and $[\textsc{navy}]$ is as necessity operators in modal logic. As section 2.2 substantiates, if $[\textsc{list}]$ is a necessity operator, a modal proof of $[\textsc{list}]A$ in the simplest modal logic, K, is precisely a proof of A that takes only other $[\textsc{list}]$ formulas into account.

The expressive power of scoped representations becomes particularly attractive when we *relate* the information that can be used in different scopes. For purposes of practical inferences in real applications, this streamlines the statement of commonalities, facilitating maintenance and reuse. For knowledge representation, this may make possible a range of natural inferences, including important inferences about nested scopes, that would be difficult or impossible to describe otherwise.

For example, since both the list and the specification in the above example represent kinds of accounting information, many parallel inferences may be required both in the scope of $[\textsc{list}]$ and in the scope of $[\textsc{spec}]$. These common inferences motivate an operator $[\textsc{acct}]$ for specifying facts about price that list and specification share. One simple example might be the fact that prices are measured in dollars:

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

For such statements to play their intended role in reasoning, we need a way to to transfer results from one scope to another. To draw inferences about the units in which list and specification record prices using (2), we need to be able to infer $[\textsc{list}]A$ and $[\textsc{spec}]A$ when we have $[\textsc{acct}]A$.

An operator $[\textsc{both}]$ that records information that GE and the Navy share will also be needed to capture commonalities in $[\textsc{ge}]$ and $[\textsc{navy}]$. It might be used, for example, to record that the two organizations are aware of the method of calculating prices described by (1):

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

Again, to use (3) as intended, we need a way to infer $[\textsc{ge}]A$ and $[\textsc{navy}]A$ from $[\textsc{both}]A$.

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 $[\textsc{list}]\mathit{price}(x,p)$ and $[\textsc{spec}]\mathit{price}(x,p)$. This illustrates how a scoped representation can be more natural and convenient than a corresponding complex first-order representation, such as *list-price*(x,p) and *spec-price*(x,p), for which common generalizations like the measurement of price in dollars must be spelled out explicitly and separately.

With modal operators, this kind of reasoning can be described formally and investigated mathematically by introducing axiom schemas that can be applied in constructing proofs. For example, the (INC) axiom relating $\Box_i$ to $\Box_j$ by inclusion:

$$(\textsc{inc}) \quad \Box_i A \supset \Box_j A$$

can capture the import of the $[\textsc{both}]$ and $[\textsc{acct}]$ modalities; (INC) can relate $[\textsc{both}]$ to the $[\textsc{ge}]$ and $[\textsc{navy}]$ operators and $[\textsc{acct}]$ to $[\textsc{list}]$ and $[\textsc{spec}]$.

A different strategy for relating operators relies on establishing relationships between outer scopes and more deeply nested scopes. The need for this strategy in one direction is already present with (3). To use (3) as we would use (1), we need a way to infer A from $[\textsc{both}]A$. The need for this strategy in the other direction arises in modeling the *hypothetical* reasoning of agents. For example, suppose what the Navy knows is specified as in (4).

(4) $[\textsc{navy}]$ $[\textsc{list}]$ $\mathit{price}(\mathit{fx22\text{-}engine}, 3600) \wedge$
$[\textsc{navy}]$ $[\textsc{spec}]$ $\mathit{spares}(\mathit{fx22\text{-}engine}, \mathit{fx22\text{-}fan\text{-}blades})$

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. From this, and (3), we might expect to be able to conclude that the Navy could determine its specification price if it knew the list price for FX22 fan blades. This requires hypothetical reasoning about the reasoning of the Navy. The fact that must be derived is:

(5) $\forall p$ $[\textsc{navy}]$ $([\textsc{navy}]$ $[\textsc{list}]$ $\mathit{price}(\mathit{fx22\text{-}fan\text{-}blades}, p) \supset$
$[\textsc{navy}]$ $[\textsc{spec}]$ $\mathit{price}(\mathit{fx22\text{-}engine}, 3600+p))$

We would like to derive (5) as a consequence of (3), but our goal has the form $[\textsc{navy}]$ $[\textsc{navy}]$ $\mathit{price}(x,p)$, with a double embedding. Our strategy is appeal to an inference from $[\textsc{navy}]A$ to $[\textsc{navy}][\textsc{navy}]A$—what the Navy knows, it knows that it knows. This inference expresses a different, natural relationship between scopes. We continue by establishing

(6) $[\textsc{navy}]\ [\textsc{navy}]\ [\textsc{list}]\ \mathit{price}(\mathit{fx22\text{-}engine}, 3600)\ \wedge$
$[\textsc{navy}]\ [\textsc{navy}]\ [\textsc{navy}]\mathit{spares}(\mathit{fx22\text{-}engine}, \mathit{fx22\text{-}fan\text{-}blades})\ \wedge$
$[\textsc{navy}]\ [\textsc{navy}]\ [\textsc{ge}]\mathit{price}(\mathit{fx22\text{-}fan\text{-}blades}, p)$

The first two must be obtained indirectly from (4), by the same inference about nested scopes. We get the third because it is assumed in proving the implication. This establishes the result.

This discussion shows that we need two additional axiom schemas to formalize this reasoning in modal logic: (VER) and (PI):

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

End example.

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. In addition to the axioms (INC), (VER) and (PI) introduced above, the axioms (CON) and (NI) are widely used:

(CON) (consistency)	$\neg \Box_i (A \wedge \neg A)$
(NI) (negative introspection)	$\neg \Box_i A \supset \Box_i \neg \Box_i A$

(As we shall explain subsequently, these additional two axioms are not immediately compatible with the framework developed in this paper.) For example, the combination of (CON) and (PI), known as KD4, has been argued to give rise to a sensible model of belief, because a normative agent's beliefs are consistent (in keeping with (CON)), and because an agent believes it believes any proposition it believes (in keeping with (PI)). 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)). The modeling of the attitudes of agents in modal logics begins with (Hintikka, 1962); subsequent work is reviewed in (Lenzen, 1978); (Fagin et al., 1995) offers a recent introduction and case studies. Not surprisingly, the choice of which axioms should be used to describe different kinds of attitudes are controversial. For example, it may be appropriate to incorporate (NI) into models of belief or knowledge over finite domains—giving KD45 and S5 respectively; it may not always be appropriate to incorporate (CON) into models of belief—giving K4. Modalities are called D when governed just by (CON); T is for (VER).

2.2 The Range of Modal Proof Systems

This paper will consider a family of first-order modal languages, $\mathcal{L}^{\Box}_{m,T}$. $\mathcal{L}^{\Box}_{m,T}$ is parametrized 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 (INC), (VER) and (PI) axioms. As usual, we presume a signature describing the arity of functions and predicates, and thus a set of atomic formulas of the form

$p(t_1, \ldots, t_n)$. Schematizing such formulas as P, the formulas of $\mathcal{L}^{\Box}_{m,T}$ are described as A by the following grammar:

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

In formal manipulations, we will keep to the concise $\Box_i$ notation, but we will continue to use the [NAME] notation seen in the example, in contexts where necessity operators are profitably assigned intelligible and legible names. Note that 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. Note however that many of the proof theoretic devices presented in this section can be modified straightforwardly to handle the alternative *varying* and *constant* domain systems.

2.2.1 Hilbert Systems

Inference in modal logic is most succinctly and intuitively characterized by Hilbert Systems. In these systems, a proof is sequence of formulas where each formula is either an instance of an axiom, or derivable from earlier formulas by the action of simple inference rules. For the simplest propositional modal logic ($K_{(i)}$ from (Halpern and Moses, 1985)), there are three axiom schemas:

$A1$.	*Any tautology of classical propositional logic*
$A2$.	$\Box_i A \wedge \Box_i(A \supset B) \supset \Box_i B$
$A3$.	$\Diamond_i A \equiv \neg\Box_i \neg A$

These are combined by two rules of inference:

$R1$.*(modus ponens)*	From A and $A \supset B$ infer B.
$R2$.*(necessitation)*	From A infer $\Box_i A$.

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

It is relatively straightforward to see how this proof system imposes a scope discipline on the modal operators, so that in any scope, logic can be used to combine all and only information explicitly asserted there. In this system, each formula in a proof is a theorem that holds in the real world, at root scope. Conclusions cannot depend on additional assumptions made for the sake of argument; the rule of necessitation should only apply to theorems. When a formula in a proof takes the form $\Box_i A$, it predicates A of the ith more-deeply nested scope. Any tautology can be established in any nested scope, by introducing it at root scope as an instance of axiom A1, and then applying R2 to introduce the necessary nestings. Axiom A2, together with the rule of modus ponens, allows the action of modus ponens in nested scopes. As a result, nested scopes are closed under logical consequence, the same way the root scope is. On the other hand, A2 and R2 are the only logical means of introducing formulas of the form $\Box_i A$. To derive a contingent conclusions of the form $\Box_i A$ (for example in the consequent of an implication), we must combine explicit assumptions of the form $\Box_i A$ (made for example in the antecedent of the implication) with the action of A2.

Hilbert Systems seem intuitive, and their use can sometimes facilitate mathematical study of modal systems, for example in proofs of soundness and completeness. However, Hilbert Systems are computationally unattractive. A key 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 only instantiations of subformulas of Γ are used. In general, the use of 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).

2.2.2 *Structurally Scoped Sequent Calculi*

A modal proof system that does satisfy the subformula property is shown in Figure 1. This proof system extends the sequent calculus of classical logic with rules governing modal operators; the modal rules are governed by parameters which vary in order to encode relationships between scopes. This sequent calculus represents a sound and complete inference system for the same semantics as the Hilbert System characterizes: it is an equivalent system. 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 of the form $\Gamma \longrightarrow \Delta$, where Γ and Δ are multisets of formulas; this represents a derivation of the disjunction of the Δ formulas, using the Γ formulas as assumptions. The label of the root of a proof is called its end-sequent.

Any instantiation of the axiom rule $\Gamma, A \longrightarrow A, \Delta$ is a proof (so a conclusion 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 1, 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. There are no other proofs. 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 1, 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.

An informal justification of how this system creates and maintains scopes in proofs is as follows. Each sequent appears in a scope that corresponds to its position in the proof tree. In the proof, applications of $(\rightarrow \Box_i)$ and $(\Diamond_i \rightarrow)$ mark the boundaries between scopes. The entire subproof *above* each application is more deeply *nested* in scope, by the application of one $\Box_i$ operator. The $(\Box_i \rightarrow)$ and $(\rightarrow \Diamond_i)$ rules represent applying necessary information in the current scope. In

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

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

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

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

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

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

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

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

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

$$\frac{(\Gamma, \Box_i A)^{\Box_i \to}, A \longrightarrow \Delta^{\to \Diamond_i}}{\Gamma, \Box_i A \longrightarrow \Delta}\ \Box_i \to$$

$$\frac{\Gamma^{\to \Box_i} \longrightarrow A, (\Box_i A, \Delta)^{\Diamond_i \to}}{\Gamma \longrightarrow \Box_i A, \Delta}\ \to \Box_i$$

$$\frac{\Gamma^{\Box_i \to} \longrightarrow A, (\Diamond_i A, \Delta)^{\to \Diamond_i}}{\Gamma \longrightarrow \Diamond_i A, \Delta}\ \to \Diamond_i$$

$$\frac{(\Gamma, \Diamond_i A)^{\to \Box_i}, A \longrightarrow \Delta^{\Diamond_i \to}}{\Gamma, \Diamond_i A \longrightarrow \Delta}\ \Diamond_i \to$$

$$\frac{\Gamma, \forall x.A, A[t/x] \longrightarrow \Delta}{\Gamma, \forall x.A \longrightarrow \Delta}\ \forall \to$$

$$\frac{\Gamma \longrightarrow A[a/x], \forall x.A, \Delta}{\Gamma \longrightarrow \forall x.A, \Delta}\ \to \forall^{\dagger}$$

$$\frac{\Gamma, \exists x.A, A[a/x] \longrightarrow \Delta}{\Gamma, \exists x.A \longrightarrow \Delta}\ \exists \to^{\dagger}$$

$$\frac{\Gamma \longrightarrow A[t/x], \exists x.A, \Delta}{\Gamma \longrightarrow \exists x.A, \Delta}\ \to \exists$$

Figure 1: Structurally-scoped, cut-free sequent calculus for modal logic. † For $(\to \forall)$, and $(\exists \to)$, a must not appear in the conclusion.

some cases these rules may introduce boundaries as well: in some logics, necessary information can be applied *only* in nested scopes.

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 is accomplished by operators $^{\Box\rightarrow}$, $^{\rightarrow\Box}$, $^{\Diamond\rightarrow}$, and $^{\rightarrow\Diamond}$ that relate sequents above and below modal rules. The intent of filtering functions is this: Only those formulas that describe the nested scope survive the transition from below the application of $(\rightarrow \Box_i)$ to above it. Above the transition, surviving formulas are modified to reflect their strength in the new, nested scope. The filtering functions are thus the distinctive feature of the structurally-scoped modal proof system.

Filtering functions vary in a way that indirectly encodes the relationships between scopes as given in Hilbert Systems by axioms (INC)–(PI). For K modalities, we take scopes at face value: necessary formulas are formulas that apply with ordinary force in nested scopes. Thus, the $(\rightarrow \Box)$ and $(\Diamond \rightarrow)$ transitions eliminate all assumptions except those of the form $\Box A$ and eliminate all (potential) conclusions except those of the form $\Diamond A$. In the nested scope we remove the outer $\Box$ on assumptions and the outer $\Diamond$ on conclusions. How do we apply necessary information in K? K scopes need not be consistent—necessity does not imply possibility—so we allow necessary information to be brought to bear only as a side effect of creating a scope using $(\rightarrow \Box)$ and $(\Diamond \rightarrow)$. Thus, for K, we dispense with $(\Box \rightarrow)$ and $(\rightarrow \Diamond)$ rules.

Positive introspection and inclusion are modeled by changes in the $(\Box \rightarrow)$ and $(\Diamond \rightarrow)$ filters. To achieve the effect of (PI), surviving assumptions appear both as A, so they are true in the current scope, and as $\Box A$, so they will be true in future nested scopes. Surviving conclusions likewise appear both as A and as $\Diamond A$. In S4, $\Box A$ implies A, so in S4 the same effect is achieved by passing just $\Box A$ and $\Diamond A$. When (INC) relates modality i and modality j, it impacts the transition into a $\Box_j$ scope. On the one hand, any $\Box_i$ formula will be at least as strong entering a $\Box_j$ scope as it is entering a $\Box_i$ scope. This means ensuring that the results of usual filter for $\Box_i$ scopes appear above the transition. At the same time, the $\Box_i$ formula must also be as strong entering the $\Box_j$ scope as any $\Box_j$ formula would be. This means applying the usual $\Box_j$ filter to the $\Box_i$ formulas and ensuring that the results are also available in the nested scope.

Veridicality is modeled by changes in the $(\rightarrow \Box)$ and $(\Diamond \rightarrow)$ filters. With veridicality, necessary assumptions can be used and possible conclusions demonstrated in the current scope. Since there is no change in scope, the formulas above the rule application are the same as those below: the $(\rightarrow \Box)$ and $(\Diamond \rightarrow)$ filters for the (VER) logics, T and S4, are identities.

A formal description of these filtering functions follows, for completeness. (As the formal presentation plays little role in what follows, the uninterested reader may safely skip ahead to the examples of structurally-scoped proofs.) We assume that

m_i	$\Gamma^{\Box_i^{m_i}\rightarrow}$	$\Gamma^{\rightarrow\Box_i^{m_i}}$
K	*none*	$\{A \vert \Box_i A \in \Gamma\}$
T	Γ	$\{A \vert \Box_i A \in \Gamma\}$
K4	*none*	$\{A \vert \Box_i A \in \Gamma\} \cup \{\Box_i A \vert \Box_i A \in \Gamma\}$
S4	Γ	$\{\Box_i A \vert \Box_i A \in \Gamma\}$

m_i	$\Delta^{\rightarrow\Diamond_i^{m_i}}$	$\Delta^{\Diamond_i^{m_i}\rightarrow}$
K	*none*	$\{A \vert \Diamond_i A \in \Delta\}$
T	Δ	$\{A \vert \Diamond_i A \in \Delta\}$
K4	*none*	$\{A \vert \Diamond_i A \in \Delta\} \cup \{\Diamond_i A \vert \Diamond_i A \in \Delta\}$
S4	Δ	$\{\Diamond_i A \vert \Diamond_i A \in \Delta\}$

Figure 2: Primitive rules governing changes between scopes in structurally-scoped modal logics.

each $\Box_i$ is assigned a primitive type m_i from among K, T, K4, and S4, and that the modalities are related by a partial order $\geq$ such that $i \geq j$ whenever we have the inclusion $\Box_i A \supset \Box_j A$ for all formulas A. We start with the primitive functions shown in figure 2, which show how to alter the sequent to build in particular modal theories. Note that when the entry in the table is *none*, sequent rules which invoke the value of that entry do not apply in the logic—regardless of what inclusions are available. To determine an appropriate overall change for $\Box_i$ we combine the primitive functions with the effects of inclusions, according to the following definition:

$$\begin{aligned}
\Gamma^{\Box_i\rightarrow} &= \Gamma^{\Box_i^{m_i}\rightarrow} \\
\Delta^{\rightarrow\Diamond_i} &= \Delta^{\rightarrow\Diamond_i^{m_i}} \\
\Gamma^{\rightarrow\Box_i} &= \bigcup_{j\geq i, j\neq i}(\Gamma^{\rightarrow\Box_j}) \cup \bigcup_{j\geq i}(\Gamma^{\rightarrow\Box_j^{m_i}}) \\
\Delta^{\Diamond_i\rightarrow} &= \bigcup_{j\geq i, j\neq i}(\Delta^{\Diamond_j\rightarrow}) \cup \bigcup_{j\geq i}(\Gamma^{\Diamond_j^{m_i}\rightarrow})
\end{aligned}$$

Example 2. Figures 3, 4 and 5 show proofs in this system of three sequents involving a single S4 modality:

$$\begin{gathered}
\Box(a \supset \Box b) \longrightarrow \Box\Box(a \supset b) \\
\Box(a \supset \Box b) \longrightarrow \Box(a \supset \Box b) \\
\Box(a \supset \Box b) \longrightarrow a \supset \Box\Box b
\end{gathered}$$

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

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

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

Figure 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 4: Example theorem 2 in a structural system.

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 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

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

Figure 5: Example theorem 3 in a structural system.

not pass the filtering of higher $(\rightarrow \Box)$ rules. *End example.*

The significance of the relative positions of rules in a proof represents a problematic departure from 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, tableau (Smullyan, 1968) and matrix (Andrews, 1981; Bibel, 1982) theorem-proving methods can be seen as optimizations of sequent calculi which eliminate this redundancy. The difficulty that arises in the absence of free permutabilities is this. 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 scope—at the right time—automated methods must be prepared to apply a rule before they know whether the application will even be needed! The regime for imposing scope on proofs means that proofs can no longer be constructed in a goal-directed manner.

2.2.3 Explicitly-scoped Sequent Calculi

To overcome this limitation, we must represent rules in a notation that can give rules 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. In so doing, we capture the scope of each rule application in the labels of its principal and side formulas. 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). An explicitly-scoped sequent calculus is presented in figure 6.

In the calculus of figure 6, each sequent takes the form

$$\Sigma \triangleright \Gamma \longrightarrow \Delta$$

The formulas in Γ and Δ are labeled with *strings* from a distinguished alphabet of scope variables—terms composed from scope variables out of 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.) Further, a multiset of auxiliary premises Σ is associated with each sequent, and specifies the *types* of the free scope and first-order variables in the sequent; I will call Σ a typing context. (The $\triangleright$ notation is common in programming language theory to identify premises used in typing.) For a scope variable—which represents a transition of one level of nesting of some modal operator—the type records which operator it is. For a first-order variable—introduced by a quantifier rule at some scope—the type records the string representation of that scope. Thus, Σ is a multiset of pairs of the form $\alpha : i$ for scope variables and $x : \mu$ for first-order variables. The information in Σ can be combined to derive *judgments* that complex scope representations and first-order terms take particular types. For scope terms, the judgment $\Sigma \triangleright \nu : i$ indicate that ν describes a

$$\frac{\mu = \nu}{\Sigma \triangleright \Gamma, A^{\mu} \longrightarrow A^{\nu}, \Delta}\ \text{axiom}$$

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

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

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

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

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

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

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

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

$$\frac{\Sigma \triangleright \sigma : i \qquad \Sigma \triangleright \Gamma, \Box_i A^{\mu}, A^{\mu\sigma} \longrightarrow \Delta}{\Sigma \triangleright \Gamma, \Box_i A^{\mu} \longrightarrow \Delta}\ \Box_i \rightarrow$$

$$\frac{\Sigma, \alpha : i \triangleright \Gamma \longrightarrow \Box_i A^{\mu}, A^{\mu\alpha}, \Delta}{\Sigma \triangleright \Gamma \longrightarrow \Box_i A^{\mu}, \Delta}\ \rightarrow \Box_i^{\dagger}$$

$$\frac{\Sigma \triangleright \sigma : i \qquad \Sigma \triangleright \Gamma \longrightarrow \Diamond_i A^{\mu}, A^{\mu\sigma}, \Delta}{\Sigma \triangleright \Gamma \longrightarrow \Diamond_i A^{\mu}, \Delta}\ \rightarrow \Diamond_i$$

$$\frac{\Sigma, \alpha : i \triangleright \Gamma, \Diamond_i A^{\mu}, A^{\mu\alpha} \longrightarrow \Delta}{\Sigma \triangleright \Gamma, \Diamond_i A^{\mu} \longrightarrow \Delta}\ \Diamond_i \rightarrow^{\dagger}$$

$$\frac{\Sigma \triangleright t : \mu \qquad \Sigma \triangleright \Gamma, \forall x.A^{\mu}, A[t/x]^{\mu} \longrightarrow \Delta}{\Sigma \triangleright \Gamma, \forall x.A^{\mu} \longrightarrow \Delta}\ \forall \rightarrow$$

$$\frac{\Sigma, a : \mu \triangleright \Gamma \longrightarrow A[a/x]^{\mu}, \forall x.A^{\mu}, \Delta}{\Sigma \triangleright \Gamma \longrightarrow \forall x.A^{\mu}, \Delta}\ \rightarrow \forall^{\ddagger}$$

$$\frac{\Sigma, a : \mu \triangleright \Gamma, \exists x.A^{\mu}, A[a/x]^{\mu} \longrightarrow \Delta}{\Sigma \triangleright \Gamma, \exists x.A^{\mu} \longrightarrow \Delta}\ \exists \rightarrow^{\ddagger}$$

$$\frac{\Sigma \triangleright t : \mu \qquad \Sigma \triangleright \Gamma \longrightarrow A[t/x]^{\mu}, \exists x.A^{\mu}, \Delta}{\Sigma \triangleright \Gamma \longrightarrow \exists x.A^{\mu}, \Delta}\ \rightarrow \exists$$

Figure 6: Path-based, explicitly-scoped, cut-free sequent calculus for modal logic. † For $(\rightarrow \Box_i)$, $(\Diamond_i \rightarrow)$, α must not appear in the conclusion. ‡ For $(\rightarrow \forall)$, $(\exists \rightarrow)$, a must not appear in the conclusion.

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

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

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

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

Figure 7: Deriving the judgment $\Sigma \triangleright \nu : i$.

transition that modality i matches; such judgments are derived according to the rules shown in Figure 7, which realize axioms (INC), (VER) and (PI) as rules of inference amalgamating and reclassifying terms. Similarly, judgments of the form $\Sigma \triangleright t : \mu$ indicates 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 assignment $x : \nu \in \Sigma$, ν is a prefix of μ.*

The explicitly-scoped calculus imposes the same scope discipline as the earlier systems by its manipulation of terms. The rules for the connectives of ordinary first-order logic identify the scopes of principal and side formulas. The axiom rule in this system requires the labels of formulas to match, as well as the formulas themselves. Thus, an atomic conclusion can be established in a nested scope only in virtue of an assumption introduced into that scope by some lower modal rule: The $(\Box_i \rightarrow)$ and $(\rightarrow \Diamond_i)$ rules apply a result in a nested scope by appending an additional term ν to the label of the side formula. These terms are constrained to match the strength of the principal formula by imposing a judgment $\Sigma \triangleright \nu : i$. Meanwhile, the $(\rightarrow \Box_i)$ and $(\Diamond_i \rightarrow)$ rules create a *new* nested scope by appending a new variable (representing the next type of nesting) to the label of the side formula. Since the new variable introduced at a $(\rightarrow \Box_i)$ rule does not appear in any scope fixed to that point, necessary assumptions will have to be instantiated by this new variable for the new scope to figure in an axiom. This strategy for isolating nested formulas from outer formulas is subtler but ultimately similar to the strategy used in the structurally-scoped sequent calculus. Finally, to ensure that first-order terms do not escape their scopes, we require the judgment $\Sigma \triangleright t : \mu$ when $(\rightarrow \exists)$ and $(\forall \rightarrow)$ are applied in scope μ with instantiation t.

Because of its explicit scoping, this new system is somewhat more expressive than either of the two previous modal proof systems. 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 1 if and only if there is a derivation with end-sequent $\emptyset \triangleright \Gamma \longrightarrow \Delta$ in the explicitly-scoped sequent calculus (ie., every assumption and conclusion is labeled with ϵ).

This result is typically established by showing that the explicitly-scoped calculus is also sound and complete for the usual semantics of modal logic (Kripke, 1963). Recall that a modal frame consists of a set W of worlds and binary relations R_i for each pair of operators $\Box_i$ and $\Diamond_i$; the axiom schemas correspond to properties which the relations must satisfy (e.g., for (VER), reflexivity; for (PI), transitivity). The truth of a formula is relativized to a world w; in particular, $\Box_i A$ is true at w if and only if A is true at every v such that $R_i(w, v)$. Since the explicitly-scoped system labels formulas with the point where they are to be evaluated, and manipulates those scopes using logical rules analogous to quantifiers, it is obviously quite close to the semantics. In fact, the annotations can be thought of as paths of accessibility between possible worlds. Ohlbach has devised a model-theory for modal logic that takes paths as primitive: the idea is to replace every relation R_i with a set of functions AF_i such that $R_i(u, v)$ if and only if $\exists f \in AF_i . v = f(u)$. For this model-theory, the explicitly-scoped sequent calculus is nothing more than the translation of modal logic into classical logic given by the semantics (Ohlbach, 1991; Ohlbach, 1993).

Nevertheless, the explicitly-scoped calculus offers a number of advantages for deduction over reasoning with a traditional semantics—so called reified methods for modal deduction (Moore, 1985; Jackson and Reichgelt, 1987; Frisch and Scherl, 1991). First, it is more expressive: equations between terms can encode axioms about necessity which cannot be captured using first-order axioms about accessibility relations (van Benthem, 1983; Ohlbach, 1993). (We will not consider such cases here.) Second, it is more efficient. Encoding scopes as terms and reasoning by equality makes proofs more compact and search more constrained than reasoning about relations. As we shall see in the next section, equational unification makes it relatively simple to work with partially-specified paths of accessibility; working simply and efficiently with worlds and *partially-specified proofs of relatedness* is much more involved. Such advantages are well-known from general uses of equality in theorem-proving (Plotkin, 1972).

Moreover, the use of an explicitly-scoped calculus need not be regarded as a semantic method, despite the apparent similarity. (Stone, 1996) 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-scoped calculus can be considered a purely proof-theoretic optimization. Further, as in this paper, the explicitly-scoped calculus can be studied fruitfully as a proof-theoretic object in its own right (see (Schmidt, 1996) for another example).

Example 3. Consider again the three theorems of example two. Proofs identical

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

Figure 8: Example theorem 1 in an explicitly-scoped system.

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

Figure 9: Example theorem 2 in an explicitly-scoped system.

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 8, 9 and 10. Note how the labels encode the scopes of the different $(\Box \rightarrow)$ applications. In figure 8, the side formula of the lower $(\Box \rightarrow)$ gets $\alpha\beta$, indicating the double nesting; likewise, in figure 9, it gets α; and in figure 10, the empty string. As with the structurally-scoped system, these $(\Box \rightarrow)$ rules cannot be permuted down across the remaining $(\rightarrow \Box)$ rules. Otherwise, they would violate the eigenvariable condition that says that when a scope is introduced by a $(\rightarrow \Box)$ rule, it cannot appear anywhere in the sequent. On the other hand, in the explicitly-scoped system, the proof can still be constructed if the $(\Box \rightarrow)$ 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. *End example.*

2.2.4 A Lifted System

Using general proof-theoretic techniques (as in eg. (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.

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

Figure 10: Example theorem 3 in an explicitly-scoped system.

Second, requirements for variables to be new are replaced by the use of Herbrand (or Skolem) terms. Herbrand terms contain as subterms all values that would have to appear on the sequent where the variable was introduced, taking into account possible permutations. By ruling out circular terms by an occur-check in unification, we ensure that a variable can be chosen in place of the Herbrand function and the proof reordered so that the variable is new. This eliminates the remaining impermutabilities of the calculus.

Figure 11 shows the final, lifted system, $LE\Box$, which the remainder of this paper addresses. 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:

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

(as always, formulas in Γ and Δ are labeled by terms explicitly indicating scope). Because terms and variables are introduced globally, the typing context must grow throughout the proof: Σ represents an input context, while Σ' represents an output context enriched to describe the new variables and terms introduced in the subproof above. Similarly, 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. Note that binary inferences propagate this list *first* to the *right* subproof, and *second* to the *left* subproof. Later sections will exploit the overall ordering of equations that results, in which equations from left subproofs always follow equations from right subproofs.

Each formula in a sequent is associated with a list of free variables schematized by a subscript X in the inference rules of figure 11; quantifier and modal rules which introduce a variable add the variable to this list. Herbrand terms involve function symbols associated uniquely with quantifiers and modal operators (as indicated by subscripting); we build a Herbrand term as a placeholder for a fresh eigenvariable by applying this function symbol to the list of free variables on the

$$\frac{}{\Sigma/\Sigma; C/C, A=B, \mu=\nu \rhd \Gamma, A^{\mu}_{X} \longrightarrow B^{\nu}_{X}, \Delta}\ \text{axiom}$$

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

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

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

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

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

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

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

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

$$\frac{\Sigma, x : i/\Sigma'; C/C' \rhd \Gamma, (\Box_i A)^{\mu}_{X}, A^{\mu x}_{X,x} \longrightarrow \Delta}{\Sigma/\Sigma'; C/C' \rhd \Gamma, (\Box_i A)^{\mu}_{X} \longrightarrow \Delta}\ \Box_i\rightarrow^{\dagger}$$

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

$$\frac{\Sigma, x : i/\Sigma'; C/C' \rhd \Gamma \longrightarrow (\Diamond_i A)^{\mu}_{X}, A^{\mu x}_{X,x}, \Delta}{\Sigma/\Sigma'; C/C' \rhd \Gamma \longrightarrow (\Diamond_i A)^{\mu}_{X}, \Delta}\ \rightarrow\Diamond_i^{\dagger}$$

$$\frac{\Sigma, \alpha(X) : i/\Sigma'; C/C' \rhd \Gamma, (\Diamond_{i\alpha} A)^{\mu}_{X}, A^{\mu\alpha(X)}_{X} \longrightarrow \Delta}{\Sigma/\Sigma'; C/C' \rhd \Gamma, (\Diamond_{i\alpha} A)^{\mu}_{X} \longrightarrow \Delta}\ \Diamond_i\rightarrow$$

$$\frac{\Sigma, u : \mu/\Sigma'; C/C' \rhd \Gamma, (\forall x.A)^{\mu}_{X}, (A[u/x])^{\mu}_{X,u} \longrightarrow \Delta}{\Sigma/\Sigma'; C/C' \rhd \Gamma, (\forall x.A)^{\mu}_{X} \longrightarrow \Delta}\ \forall\rightarrow^{\dagger}$$

$$\frac{\Sigma, h(X) : \mu/\Sigma'; C/C' \rhd \Gamma \longrightarrow (A[h(X)/x])^{\mu}_{X}, (\forall_h x.A)^{\mu}_{X}, \Delta}{\Sigma/\Sigma'; C/C' \rhd \Gamma \longrightarrow (\forall_h x.A)^{\mu}_{X}, \Delta}\ \rightarrow\forall$$

$$\frac{\Sigma, h(X) : \mu/\Sigma'; C/C' \rhd \Gamma, (\exists_h x.A)^{\mu}_{X}, (A[h(X)/x])^{\mu}_{X} \longrightarrow \Delta}{\Sigma/\Sigma'; C/C' \rhd \Gamma, (\exists x.A)^{\mu}_{X} \longrightarrow \Delta}\ \exists\rightarrow$$

$$\frac{\Sigma, u : \mu/\Sigma'; C/C' \rhd \Gamma \longrightarrow (A[u/x])^{\mu}_{X,u}, (\exists x.A)^{\mu}_{X}, \Delta}{\Sigma/\Sigma'; C/C' \rhd \Gamma \longrightarrow (\exists x.A)^{\mu}_{X}, \Delta}\ \rightarrow\exists^{\dagger}$$

Figure 11: Lifted path-based, explicitly-scoped, cut-free sequent calculus for modal logic, *LE*$\Box$. † 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 12: Example theorem 1 in the lifted system.

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

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

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 and from first-order variables to first-order terms—satisfying certain conditions. As usual, we write $t\theta$ to describe the action on term t of the term homomorphism induced by θ. Further, we abbreviate by $\Sigma\theta$ the set of modal assignments of the form $\alpha\theta : i$ for $\alpha : i \in \Sigma$ and α a Herbrand term—$\Sigma\theta$ thus gives the types of exactly the ground scope transitions introduced in the proof. In this notation, the conditions θ must satisfy are the following. First, for every declaration of a first-order variable $x : \mu$ in Σ and for every Herbrand term t with $t : \nu$ in Σ and $t\theta$ a subterm of $x\theta$, we must have that $\nu\theta$ is a prefix of $\mu\theta$. This ensures that the values of first-order variables respect the scopes where the variables are introduced. Second, for every declaration of a modal variable $x : i$ in Σ, we can derive $\Sigma\theta \triangleright x\theta : i$ using the inference rules of figure 7. This ensures that the transitions made at modal rules respect the strengths of the modal statements.

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, as in (Lincoln and Shankar, 1994), the proof gives explicit transformations between the derivations of the two systems (cf. also (Frisch and Scherl, 1991)).

Example 4. Proofs in the lifted system of our three S4 theorems appear in figures 12, 13 and 14. The figures present *uniform proofs* (Miller et al., 1991), as an illustration of how the lifted system facilitates systematic, goal-directed proof search. In all three proofs, we 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

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

Figure 13: Example theorem 2 in the lifted system.

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

Figure 14: Example theorem 3 in the lifted system.

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 $\alpha\beta$. *End example.*

2.3 *The Problem*

Using this system, 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.

For example, in constructing the proofs shown in figures 12, 13 and 14, a search engine will likely begin by constructing the right branch, and solving the 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. 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, resolving scopes in modal deduction 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, 1985) 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 scopes in a modal proof; when statements of possibility create different scopes, 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 resolving scopes by unification is easy. For example, since K variables can only be instantiated to single terms, scope 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 scopes continue to apply. In fact, the complexity of scope unification is likely to pose the most significant obstacle to the use of modal logic in practical applications, because alternatives for unifying scopes arise because of the very axioms for relating scopes that make modal logic attractive as a representation in the first place.

It is also noteworthy that the complexity of resolving scopes by unification 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 equations between modal scopes (cf. (Wallen, 1990; Auffray and Enjalbert, 1992)). For each scope variable or Herbrand term μ (other than ϵ) there is a term π_μ such that each occurrence of μ in an equation 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 and Herbrand terms occur uniquely; equating terms means identifying the nodes the terms designate. In fact, the unique prefix property may be imposed on ground proofs as well without loss of generality. Too see why this is, observe that we can obtain a new ground proof from any other ground proof by of substituting a fresh scope eigenvariable α for all and only those occurrences of a scope variable β that are preceded by a given prefix μ.

The unique prefix property makes reasoning about annotation equations 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 the general case. Moreover, since an efficient algorithm can determine whether a set of strings are equal under a polynomial size substitution, the problem of resolving scopes by unification is in NP.

Nevertheless, the problem is hard.

Theorem 1 *In LE$\Box$, the problem of determining whether there is a proof containing a given derivation as its first component is NP-hard.*

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.

The typing context will contain $\mu : 1$ for each, assigning each to a T modality 1 (governed by the introspection 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 $G\theta = Q_{S_{11}}Q_{S_{12}}Q_{S_{13}}K_1 \dots Q_{S_{m1}}Q_{S_{m2}}Q_{S_{m3}}K_m$. 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.) Each $\Diamond_1$ introduces a fresh variable, while each $\Box_1$ introduces a Herbrand term with a unique head function constant. Thus, proving this formula ensures that each p_a is established in a scope 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 in a scope 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 Herbrand terms. Now proving $\Gamma \longrightarrow \Delta$

generates a proof attempt in which each goal p_a in scope G is matched with the assumption of p_a in scope Q_a. This is precisely the unification problem considered above. ■

3 □-only Logic and Variable Introduction

The recent proof systems reviewed in section 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 Herbrand terms in unifiers: When either α must nest in β or β must nest in α, the one that nests is the one that is introduced *later* in the proof. This theorem is presented in section 3.1. The restriction on negation is not as dire as it may seem, as shown in section 3.2: 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 scopes into alternative axiomatic links in proof search, so that the remaining scope alternatives can be managed efficiently.

Why does the invariant on the introduction of terms hold? The formal argument is given in section 3.1, but informally, the invariant is combined effect of two properties of □-only proofs in $LE\Box$. First, the terms representing the scope of a formula can only grow through the application of modal rules. Accordingly, all the terms labeling the scope of 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 scope paths, and can fail in accounts of additional axiomatic relationships between scopes. 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 in any scope whatsoever.

Second, when □ alone appears in the proof, variables are introduced on annotations precisely when annotations change in *left* rules, while Herbrand 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 can only be unified with the corresponding right term using a ground string of Herbrand terms introduced earlier in the proof. Herbrand terms, meanwhile, are introduced only on right terms, so there is no way for a newer Herbrand term to represent a scope in which an older one is nested. This constraint rules out or resolves search ambiguities such as those investigated in section 2.3.

3.1 Theorem on Variable Introduction

The proofs of this section require three easy corollaries of the simple structure of *LE*□.

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

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

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

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

by 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 LE□ derivation $\mathcal{D}$ with end-sequent*

$$\Sigma/\Sigma'; C/C' \triangleright \Gamma, A^{\mu}_{X}, A^{\mu}_{X} \longrightarrow \Delta$$

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

$$\Sigma/\Sigma'; C/C' \triangleright \Gamma, A^{\mu}_{X} \longrightarrow \Delta$$

by eliminating one occurrence of A^{μ}_{X} on the left throughout $\mathcal{D}$; we can likewise eliminate duplicate formulas from Δ.

Lemma 3 (monotonicity) *For any LE□ derivation $\mathcal{D}$ with end-sequent*

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

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. ■

Lemma 4 (simplicity) *For any $LE\Box$ 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. ■

Let C be a list of annotation equations resulting from a $LE\Box$ 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. We have the following definition:

Definition 2 *$\mathcal{D}$ 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 $j < i$.*

The variable introduction theorem states the key observation true of all *derivations* in $LE\Box$, regardless of whether there is a solution to the list of equations associated with them.

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

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

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

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

where Σ'' contains only elements from Σ', C'' contains only elements from C', and Γ' contains only elements from Γ, and where $\mathcal{D}'$ enjoys the variable introduction property.

Proof. Consider a proof attempt $\mathcal{D}$ with end-sequent:

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

We say a formula A^μ is *linked* 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 linked, we say A^μ is *unlinked* 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 $LE\Box$ proof attempts shows that there is a proof attempt $\mathcal{D}'$ corresponding to $\mathcal{D}$ satisfying the conditions of the statement of the lemma, and where one of two further properties holds of each unlinked 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 "unlinked" in $\mathcal{D}$ should "fall off" of $\mathcal{D}'$.

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

$$\Sigma/\Sigma; C/C, A = B, \mu = \nu \triangleright \Gamma, A^{\mu}_{X} \longrightarrow B^{\nu}_{X}, \Delta$$

We construct the axiom:

$$\Sigma/\Sigma; C/C, A = B, \mu = \nu \triangleright \Gamma_1, A^{\mu}_{X} \longrightarrow B^{\nu}_{X}, \Delta$$

Here Γ_1 consists of the formulas D^{η} of Γ such that either D^{η} is linked in the axiom or η is a prefix of μ. Axioms introduce no variables, so any axiom satisfies the variable introduction property. The equations and typings are unchanged. Unlinked formulas whose annotations are not prefixes of μ do not appear in any left equation term. They are correctly eliminated in the new derivation. The labels of the other unlinked formulas appear first on the left here, because by definition their labels are not prefixes of ν.

Now suppose the claim is true for derivations of height h or less, and consider derivations of height h+1. The five right rules that do not alter annotations are straightforward. In each case, we apply the induction hypothesis to the immediate subderivation(s), observe that the principal and side formulas are available in the new derivation(s), and apply the right rule to the new results. The induction hypothesis and the monotonicity lemma ensure that the resulting derivation meets the needed conditions. This takes care of cases for $(\rightarrow \wedge)$, $(\rightarrow \vee)$, $(\rightarrow\supset)$, $(\rightarrow \forall)$, and $(\rightarrow \exists)$.

The five left cases that do not alter annotations are somewhat more involved. (The rules are $(\wedge \rightarrow)$, $(\vee \rightarrow)$, $(\supset\rightarrow)$, $(\forall \rightarrow)$, and $(\exists \rightarrow)$.) We consider the case of $(\supset\rightarrow)$ in detail as a key illustration. Consider a derivation $\mathcal{D}$ ending in $(\supset\rightarrow)$, as below:

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

The same multiset of formulas Δ appears above the rule-application in the right subderivation and below the rule-application. Hence the unlinked formulas in the whole derivation are all unlinked in right subderivation. We apply the induction hypothesis to the *right* subderivation; we thereby eliminate or find first left uses for all these unlinked formulas. In particular, if our principal formula $(A \supset B)^{\mu}_{X}$ is unlinked, 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

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

in which neither $(A \supset B)^\mu_X$ 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, we have two cases (principal formula linked, principal formula unlinked but preserved in the right subderivation) with parallel structure. We apply the induction hypothesis to the left subderivation—observing that it applies only to those unlinked formulas in the overall derivation that are *not* labeled by prefixes of μ. We apply the weakening lemma to each new derivation and the unlinked 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:

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

The annotations of formulas weakened onto either subderivation are first used in equations on the left in this overall deduction, by the induction hypothesis (these annotations have a first left use in one subderivation and no use in the other). Meanwhile, for any unlinked 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''.

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

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

Observe that the unlinked formulas in the immediate subderivation are exactly those that are unlinked in the overall derivation. For, consider any unlinked 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 $\Box A$. 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 $A^{\mu\alpha(X)}_X$. By the simplicity lemma, we may assume this is not so without loss of generality.

So the induction hypothesis applies to the subderivation with the same unlinked formulas. Applying $(\rightarrow \Box)$ to the result (as below) gives a derivation with the needed properties:

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

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

$$\frac{\Sigma, x : i/\Sigma'; C/C' \triangleright \Gamma, (\Box_i A)^{\mu}_{X}, A^{\mu x}_{X,x} \longrightarrow \Delta}{\Sigma/\Sigma'; C/C' \triangleright \Gamma, (\Box_i A)^{\mu}_{X} \longrightarrow \Delta} \Box_i \to$$

Here x is a fresh scope variable, different from other variables and Herbrand terms. In the immediate subderivation, $A^{\mu x}_{X,x}$ is unlinked, because μx cannot be the prefix of the annotation of any Δ formula. Apply the induction hypothesis. If $A^{\mu x}_{X,x}$ disappears, the subderivation suffices. Otherwise, use the new subderivation to construct a derivation ending:

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

Here, μx—along with all surviving unlinked 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 unlinked formulas. ■

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

Lemma 5 (substitution ordering) *Suppose $\mathcal{D}$ is a LE$\Box$ proof attempt 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 Herbrand terms, and if $x\theta$ contains Herbrand term c there is a Herbrand 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 $i-1$ equations of C and consider a solution θ for the first i equations of C. Naturally, θ is a solution to the first $i-1$ equations, so by the induction hypothesis, variables introduced in the last $i-1$ equations are bound to earlier Herbrand 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 Herbrand terms; θ must associate any new variable in l_i with some of them; and $l_i\theta$ can contain no new Herbrand terms. ■

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

Theorem 3 (constant ordering) *For any proof $\mathcal{D}, \theta$ in LE$\Box$ 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 scope equations imposed in $\mathcal{D}$, and a subset of these equations are imposed in $\mathcal{D}'$, θ also unifies the scope equations imposed in $\mathcal{D}'$. By the substitution ordering lemma, θ assigns strings of Herbrand terms to each scope variable (that appears in the equations of $\mathcal{D}'$). Every scopal Herbrand term and 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 all modal Herbrand terms to be represented as constants that are distinguished before unification (even though they technically are function applications with free variables). The mating theorem-proving method does this via multiplicities (Andrews, 1981; Bibel, 1982). *Uniform proof search* (Miller et al., 1991; Miller, 1994), an abstraction of backward chaining, does this by applying left rules only when right rules are not applicable. Modal Herbrand functions must be unified only when applications of $(\rightarrow \Box)$ must be permuted lower in the proof and collapsed; this is never needed in uniform proof search because right rules already apply as early as possible.

These techniques allow the definition of an ordering on Herbrand constants in advance of solving unification equations:

Definition 3 ($\sqsubset$) *Let the equations corresponding to a proof attempt be ordered as before, and let c and d be arbitrary constants 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.

3.2 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, distracting.) The encoding ^{t} 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:

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

The right subproof is an instance of the new $\perp$ 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:

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

Since the $\perp$ rule is in fact the only rule that can establish $\perp$ on the right and it can establish anything, the addition of $\perp$ on the right does not change the provability of the end-sequent of the immediate subderivation. The new constant $\perp$ 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 scopes, it in fact suffices to introduce a *scoped* constant $\perp$ governed by the rule:

$$\Sigma \triangleright \Gamma, \perp^\mu \longrightarrow A^\mu, \Delta$$

The use of this rule is clearly sound, because it is a specialization of the more general unscoped $\perp$ 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 scopes by strings of eigenvariables introduced lower. Under this circumstance, whenever we establish $\perp^\mu$ on the left, there will in fact be some formula A^μ on the right. Therefore no generality is lost by the scoped $\perp$ rule. However, with the scoped $\perp$ rule, the variable introduction theorem goes through, and the hence the algorithms of section 4 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 $\perp$. But first I want to show that there is no magic involved in the translation. In the original proof, there are ambiguities in which scope 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 $\perp$ is used to deduce which conclusion of $\perp$. In accordance with the constant ordering theorem, the scope 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 the scopes in 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 points.

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 7, which only combine the terms declared in Σ into longer strings. Because of the unique prefix property (cf. section 2.3), 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 scope that is already under consideration.

We exploit this observation in the following lemma.

Lemma 6 (encoding negation) *To any proof $\mathcal{D}$ with end-sequent $\Gamma \longrightarrow \Delta$ in the ground, explicitly scoped system of figure 6, 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:

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

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:

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

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 Constructing Trees from Constraints

With the constant ordering theorem, we have established an invariant that eliminates one source of nondeterminism in the unification of scope equations. Given Herbrand terms α and β which appear on scope 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, scope equations may still have an exponential number of unifiers; the constant ordering theorem leaves open how strings of Herbrand terms should be partitioned among matching *variables*. We have already seen a concrete example of this, in the unification problem common to the proofs of figures 12, 13 and 14: $xy = \alpha\beta$. 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 4.1, 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 12, 13 and 14 in 4.2. The complications come in 4.3, where it is observed that inclusion axioms may introduce hard problems even into variable ambiguities. Accordingly, in 4.4 we consider restrictions on interaction axioms to rule out the problematic cases observed in 4.3, and provide a constraint solver for multimodal logics under these restrictions which uses the algorithm from 4.1 as a subroutine.

4.1 Monomodal 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 $u \leq v$, 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 $u \not\leq v$, 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 Herbrand terms can be determined in advance of unification, as in the matrix or uniform proof methods. 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 scope Herbrand 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 canoncial 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 Herbrand terms are mapped to themselves under this induced substitution. We impose on t constraints of the form $d \not\leq c$ whenever $c \sqsubset d$ (as earlier $\sqsubset$ refers to the order of introduction of Herbrand terms in the proof). Since $\sqsubset$ is a total order, these constraints ensure that any pair of Herbrand 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 Herbrand 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 $u \leq v$ and $v \not\leq u$. 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*, $u \leq v$ for *S4* and $u \leq_{\leq 1} v$ for *T*. $u \leq_{\leq 1} v$—meaning v is u or a child of u—can be represented as the conjunction of $u \leq v$ and $v \Rightarrow u$.

As a final step, we should stipulate arbitrary symbols to correspond to each node in the tree to which no Herbrand 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 Herbrand term f is introduced at some scope 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 least-nested scope 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 scope 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 Herbrand 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 $\not\leq$.[2] 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, 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. $(i,j) \not\leq (k,l)$. 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 7 (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 nonunit cell to which they belong.

Theorem 4 (correctness) *Any tree t so constructed satisfies the constraint set C.*

[2]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.

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 8 (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 5 (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 between scopes 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.

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.

4.2 An Example

Let us return to the simple example of figures 12, 13 and 14. 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 15. 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.

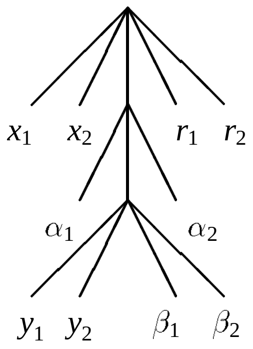

Figure 15: Tree for $xy = \alpha\beta$.

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 Problematic Interactions in Multimodal Languages

Efficient multimodal 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 multimodal language remain quite simple.

Lemma 9 (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 multimodal 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:

$$[\text{Y}]\,A \supset [\text{YT}]\,A \qquad [\text{Y}]\,A \supset [\text{YF}]\,A$$
$$[\text{Y*}]\,A \supset [\text{Y*}]\,[\text{Y*}]\,A \qquad [\text{Y*}]\,A \supset A$$
$$[\text{Y*}]\,A \supset [\text{Y}]\,A$$

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:

$$[\text{Y}]\ [\text{Y*}]\ C\ \wedge\ [\text{Y*}]\ ([\text{YF}]\ C \supset [\text{Y*}]\ B)\ \wedge\ [\text{YT}]\ B \supset A$$

To prove A, we backchain against $[\text{YT}]\ B \supset A$, 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 in scope $u\beta$. The first clause introduces variables yz that reach this scope. 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 monomodal 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 multimodal 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 multimodal 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 6 *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:

$$([\mathrm{I}]\, A \supset [\mathrm{UT}]\, A) \wedge ([\mathrm{I}]\, A \supset [\mathrm{VF}]\, A) \wedge ([\mathrm{I}]\, A \supset [\mathrm{YT}]\, A)$$

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:

$$u_i : *, \quad t_i : \mathrm{I}, \quad v_i : *$$

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 ith 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 kth letter by adding the equations:

$$m_{k1} u_k v_k = \sigma_{k-1} \alpha_k, \quad m_{k2} y_k z_k = m_{k1} u_k \beta_k$$

Under the typing context:

$$m_{k1} : *, \quad m_{k2} : *, \quad u_k : \mathrm{Y*}_k, \quad v_k : \mathrm{Y*}_k, \quad y_k : \mathrm{Y}_k, \quad z_k : \mathrm{Y*}_k, \quad \alpha_k : \mathrm{YT}_k, \quad \beta_k : \mathrm{YF}_k,$$

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:

$$[*]\ [\text{Y}]\ (A_z \supset [\text{Y*}]\ C_y) \wedge [*]\ [\text{Y*}]\ ([\text{YF}]\ C_y \supset [\text{Y*}]\ B_y) \wedge [*]\ ([\text{YT}]\ B_y \supset A_y)$$

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:

$$[*][\text{I}][*]D_i$$

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 scope introduces equations $\delta_i = \sigma$. ■

The practical relevance of this result is unclear, as the axioms involved in this construction are rather pathological. Example 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.

4.4 Restrictions for Multimodal Languages

Problems arise in multimodal 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.

4.4.1 Typing by Length Constraints

We first observe that a simple but useful syntactic restriction of *homogeneity* on the multimodal language allows algorithm $\mathcal{A}$ to be used directly. The restriction starts from a distinguished negative modality $\Box_N$. The theory T specifying relations between modal operators is homogeneous just in case it contains an inclusion $\Box_i\, A \supset \Box_N\, A$ 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:

$$\begin{array}{rl} P ::= & A \mid P \vee P \mid P \wedge P \mid N \supset P \mid \Box_i P \mid \forall x.P \mid \exists x.P \\ N ::= & A \mid N \vee N \mid N \wedge N \mid P \supset N \mid \Box_N\, N \mid \forall x.N \mid \exists x.N \end{array}$$

(A schematizes over atomic formulas.) Thus, any modality may appear in positive positions, but only $\Box_N$ may appear in negative positions.

If $\Gamma \longrightarrow \Delta$ is homogeneous, then all modal Herbrand terms in the proof must represent arbitrary $\Box_N$ transitions. If also T is homogeneous, then each of these Herbrand 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 $\Box_N$ 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 $\Box_N$ 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 4.1. Algorithm $\mathcal{A}$ computes a solution or reports that none exists in time $O(K^2 \log K)$.

4.4.2 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:

$$\begin{array}{rcl}
G & ::= & P \mid G \vee G \mid G \wedge G \mid D \supset G \mid \Box_k G \mid \forall x.G \mid \exists x.G \\
D & ::= & P \mid D \vee D \mid D \wedge D \mid G \supset D \mid \Box_i\, D_{i,\{\}} \mid \Box_k\, D\ [k \neq i] \mid \forall x.G \mid \exists x.G \\
D_{i,S} & ::= & P \mid D_{i,S} \vee D_{i,S} \mid D_{i,S} \wedge D_{i,S} \mid G \supset D_{i,S} \mid \forall x.G \mid \exists x.G \\
& & \Box_k\, D\ [i \text{ separate from } k\text{, each } j \text{ in } S \text{ disjoint from } k] \mid \\
& & \Box_k\, D_{i,S\cup\{k\}}\ [\mathrm{K} \text{ simple}]
\end{array}$$

(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 *Herbrand 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 monomodal languages. Formally, we have:

Theorem 7 (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 Herbrand 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 Herbrand terms, and nothing to prove.

Suppose the claim is true for the first $n-1$ 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 Herbrand terms in the the first $n-1$ 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 Herbrand terms. From the induction hypothesis, we know that the same forced Herbrand 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 Herbrand 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 Herbrand 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 Herbrand term, any $(\pi_x x\vec{v})\theta$ must contain forced Herbrand 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 Herbrand term from $x\theta$ not to appear in $x\theta'$. Continuing, we find $\vec{v}\theta' = \vec{c}$, for some string of n Herbrand 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 8 (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 $n-1$ 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:

$$x_i\theta = ((lx_1 \dots x_i)\theta' \cap (lx_1 \dots x_i)\theta'') \text{ minus } (lx_1 \dots x_{i-1})\theta$$

$x_i\theta$ either only contains Herbrand terms from $x_i\theta'$ or only contains Herbrand terms from $x_i\theta''$. Otherwise there would be a Herbrand term a that appears in $(lx_1 \dots x_{i-1})\theta$ but not $(lx_1 \dots x_{i-1})\theta'$ and a Herbrand term b that appears on $(lx_1 \dots x_{i-1})\theta'$ but not $(lx_1 \dots x_{i-1})\theta$, and where moreover a and b appear in both $(lx_1 \dots x_i)\theta$ and $(lx_1 \dots x_i)\theta'$. 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 Herbrand 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. ■

4.4.3 Relaxation for Least-Commitment Multimodal 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 typechecking. 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 4.4 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 Herbrand 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 Herbrand 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 Herbrand 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 multimodal 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 9 (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 fixpoint 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 4.1, the proof of correctness of algorithm $\mathcal{A}$ can be adapted to show that if 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 $k-1$ equations and consider solving k equations. Following algorithm $\mathcal{B}$, we first use this induction hypothesis to solve the first $k-1$ 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 $O(N^4)$.

5 Conclusion and Applications

Previous research on efficient deduction in modal logic devised an explicitly-scoped calculus describing modal provability. The calculus has nice formal properties but checking the axioms link in the proof represents an intractable problem. This paper has identified a new invariant for deduction problems in modal logic, and shown how this invariant leads to fast algorithms for correctly applying axioms in modal proofs. Presentations of modal logic in terms of scope equations thus provide both a theoretical tool for analyzing proofs and proof search in new ways and a practical tool for implementing fast deduction.

Adding scope to logical representations is an important goal for this paper has developed tractable solutions. The algorithms described here apply in a variety of domains that call for natural means of describing modular inference, time, and agents. To conclude, I briefly sketch the role of the three algorithms from sections 4.1, 4.4.1 and 4.4.3 in developing practical applications of deduction.

5.1 Automatic Synthesis of Functional Programs

Intuitionistic theorem proving and the related problem of automated synthesis of functional programs represents a natural domain in which to apply the constraint algorithm from section 4.1. The problem and application relies on the following observations. Functional programs can be characterized in terms of the types of their inputs and the types of their outputs. These types can in turn be represented as formulas in a logical language: the formula $p \Rightarrow q$ represents the type of functions from objects of type p to objects of type q. The correspondence between propositions and types—known as the Curry-Howard isomorphism (Howard, 1980)—at the same time identifies programs with proofs. For example, application of a function to an argument is recorded in the inference from the type $p \Rightarrow q$ of the function and the type p of the argument to the type q of the result. This parallel underlies a number of systems for the synthesis of functional programs (Martin-Löf, 1982; Constable et al., 1986).

In the type $p \Rightarrow q$, $\Rightarrow$ denotes the implication of intuitionistic logic, where the assumption p must be used only to derive q. This scope discipline can in fact be characterized by a correspondence between intuitionistic formulas and proofs and formulas and proofs of a modal logic, S4 (Gödel, 1986; Rasiowa and Sikorski, 1953; Maehara, 1954). The correspondence is achieved by translating an intuitionistic formula p to an S4 formula $T(p)$ as follows:

$$\begin{array}{lll}
T(A) & = & \Box A \; A \text{ atomic} \\
T(p \wedge q) & = & T(p) \wedge T(q) \\
T(p \vee q) & = & T(p) \vee T(q) \\
T(p \Rightarrow q) & = & \Box(T(p) \supset T(q)) \\
T(\forall x p) & = & \Box \forall x(T(p)) \\
T(\exists x p) & = & \exists x(T(p))
\end{array}$$

The correspondence between programs and proofs also holds in explicitly scoped proof system based on semantic translation, such as the one this paper has investigated (Stone, 1996). Explicitly-scoped S4 deduction, together with the basic $O(N^2 \log N)$ constraint algorithm described in section 4.1, can thus be used to derive proofs and programs.

5.2 Constraint Reasoning for Branching Time

The constraint technique proposed in section 4.4.1, meanwhile, applies in a variety of problems in temporal and causal reasoning, including typical ways of deriving particular predictions about the future on the basis of causal generalizations. The use of constraints to represent the ordering of intervals and events is a well-known technique in AI (Sacerdoti, 1975; Allen, 1983; Dean and Boddy, 1988), but has focused on linear models of time, where all events must ultimately be temporally ordered. AI theorists, however, have often preferred to work in terms of branching models of time, such as the situation calculus (McCarthy and Hayes, 1969). The algorithm described in section 4.4.1 can be taken as a description of a range of cases where efficient constraint methods can check consistency of conjunctions of ordering constraints between points in branching-time causal reasoning.

The ontology for causal theories consists of a homogeneous theory (cf. section 4.4.1) involving a pair of modal operators $\bigcirc$ and $\Box$. The negative operator is $\bigcirc$; $\bigcirc\ P$ represents that P is true after the next event occurs. Without further constraints on this operator, time can branch. Persistence of effects is captured by the modal operator $\Box$; $\Box\ P$ represents that P holds, and will continue to hold until further notice. Thus, $\Box$ is described by the axioms $\Box P \supset P$, $\Box P \supset \Box\Box P$ and $\Box P \supset \bigcirc P$. Meanwhile, the predicate $\mathbf{h}a$ gives a way to talk about the occurrence of actions: $\mathbf{h}a$ is true if the next event will include an action that can be characterized as an a.

This setup allows us to represent a prediction as a modal deduction with end-sequent $R, H \longrightarrow G$. R is the causal theory of the world, expressed in terms of $\bigcirc$, $\Box$ and $\mathbf{h}$; H is a series of statements describing future events; and G is a formula ensuring that some prediction holds after a series of events. Such deductions encode a homogeneous problem as long as $\Box$ is only needed in positive positions, to allow a proposition established at one time to be propagated forward in time in a single step of instantiation to the later time when the proposition is needed. This is the typical case in planning, prediction and explanation in AI. Because of the simplicity of the deductive framework, the results of section 4.4.1 can be used to solve the modal equations in these deductions efficiently—in time $O(N^2 \log N)$.

(In fact, the proof theory of simple modal languages can continue to be leveraged in an interesting way even in the presence of defeasible inertia (Stone, 1997a). Using defeasible argumentation (Pollock, 1992; Dung, 1993) permits proofs to retain a simple form that omits the disjunctive and negative conditions needed to handle inertia in classical logic (Schubert, 1990; Reiter, 1991).)

5.3 Executing Specifications of Agents

Finally, we return to the example of section 2.1: executing modal specifications of agents. The specification described in 2.1 is representative of a broad and useful class of formal theories in which the multimodal constraint algorithm of section 4.4.3 applies. In this class, we specify a set of S4 modalities interacting with one another, and a set of K modalities interacting with one another, but have no interactions linking the two kinds of modalities. I will call these S4/K specifications.

Recall that section 4.4 identified restrictions on multimodal reasoning: Every modality is required to be either clean or unambiguous, according to a technical definition characterizing the modal interactions T and the sequent $\Gamma \longrightarrow \Delta$ to be proved. In the technical language of section 4.4, the K modalities are clean because they are simple; the S4 modalities are clean because they have no inclusions to forced modalities. For S4/K specifications, no further restrictions on interactions or formulas are needed to ensure that scope equations are easy to solve by the techniques of section 4.4.3. Nevertheless, S4/K specifications describe the key reasoning problems needed for specifications like the Navy/GE specification of section 2.1.

In the Navy/GE example, the S4 family consists of operators [GE] describing GE's knowledge, [NAVY] describing the Navy's knowledge, and [BOTH] describing the organizations' shared knowledge. The K family consists of operators [LIST] describing GE's list, [SPEC] describing the Navy's specification, and [ACCT] describing accounts in general.

The S4 family allows for efficient specifications not just of agents' knowledge but also of their common knowledge. Common knowledge is a crucial component of coordination and agreement (Fagin et al., 1995). Inclusion axioms model shared knowledge in the GE/Navy example of section 2.1: the modality [BOTH] in the example was subjected to inclusion axioms [BOTH]$A \supset$ [NAVY]A and [BOTH]$A \supset$ [GE]A. These inclusions allow any nesting of [NAVY] and [GE] operators to be derived from a single [BOTH] operator. Thus [BOTH] in fact describes common knowledge to GE and the Navy.

The K family of operators provides a complementary tool to structure specifications. Following (Giordano and Martelli, 1994; Baldoni et al., 1993), K modal operators in some cases provide a good description of modularity in logic programs. As with the [LIST], [SPEC] and [ACCT] modalities in the Navy/GE example, modal operators can allow concepts to be described as distinct (like [LIST]*price* and [SPEC]*price*) while compactly describing bodies of knowledge that apply uniformly to both concepts. In fact, as (Schild, 1991) shows, a similar use of K modal operators can provide a general encoding of terminological knowledge.

Thanks to the constraint algorithms described in section 4.4.3, specifications combining these features can be designed like Prolog programs to offer both efficient execution and declarative semantics. For, logic programs have a restricted syntax and a restricted proof procedure that ensures that problematic scope ambiguities never arise. Without scope ambiguities, modal logic programs will define search problems whose nondeterminism depends solely on the number of alternative clauses that could establish a goal, just as in Prolog programs. (Stone, 1997b) extends the general framework from (Miller et al., 1991) to develop a modal logic programming language, DIALUP, using the partial-order mechanisms described in section 4. DIALUP provides a concrete environment in which to explore the algorithms described in this paper and the efficient specifications of agents they make possible.

References

Aho, A. V., Sagiv, Y., Szymanski, T. G., and Ullman, J. D. (1981). Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal of Computation*, 10(3):405–421.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Andrews, P. B. (1981). Theorem proving via general matings. *Journal of the Association for Computing Machinery*, 28(2):193–214.

Auffray, Y. and Enjalbert, P. (1992). Modal theorem proving: an equational viewpoint. *Journal of Logic and Computation*, 2(3):247–295.

Baldoni, M., Giordano, L., and Martelli, A. (1993). A multimodal logic to define modules in logic programming. In *ILPS*, pages 473–487.

Ballim, A., Wilks, Y., and Barnden, J. (1991). Belief ascription, metaphor, and intensional identification. *Cognitive Science*, 15:133–171.

Bibel, W. (1982). *Automated Theorem Proving*. Vieweg, Braunschweig.

Bibel, W. (1993). *Deduction: Automated Logic*. Academic Press, London.

Boyer, R. S. and Moore, J. S. (1972). The sharing of structure in theorem-proving programs. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 7*, pages 101–116. Edinburgh University Press.

Chellas, B. F. (1980). *Modal Logic: An Introduction*. Cambridge University Press, Cambridge.

Constable, R. L. et al. (1986). *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Englewood Cliffs N.J.

Dean, T. and Boddy, M. (1988). Reasoning about partially ordered events. *Artificial Intelligence*, 36:375–399.

Debart, F., Enjalbert, P., and Lescot, M. (1992). Multimodal logic programming using equational and order-sorted logic. *Theoretical Computer Science*, 105:141–166.

Dung, P. M. (1993). On the acceptability of arguments and its fundamental role in nonmonotic reasoning and logic programming. In *IJCAI*, pages 852–857.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning About Knowledge*. MIT Press, Cambridge MA.

Fitting, M. (1972). Tableau methods of proof for modal logics. *Notre Dame Journal of Formal Logic*, 13(2).

Fitting, M. (1983). *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library*. D. Reidel, Dordrecht.

Frisch, A. M. and Scherl, R. B. (1991). A general framework for modal deduction. In *Proceedings of KR*, pages 196–207. Morgan Kaufmann.

Gallier, J. H. (1986). *Logic for Computer Science: Foundations of Automated Theorem Proving*. Harper and Row, New York.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: a guide to the theory of NP-completeness*. W. H. Freeman, San Francisco.

Giordano, L. and Martelli, A. (1994). Structuring logic programs: A modal approach. *Journal of Logic Programming*, 21:59–94.

Gödel, K. (1986). Eine interpretation des intuitionistischen Aussagenkalküls. In Feferman, S., editor, *Kurt Gödel Collected Works*, volume 1, pages 296–303. Oxford. With introduction by A. S. Troelstra.

Halpern, J. Y. and Moses, Y. (1985). A guide to the modal logics of knowledge and belief: preliminary draft. In *9th International Joint Conference on Artificial Intelligence*, pages 480–490.

Hintikka, J. (1962). *Knowledge and Belief*. Cornell University Press.

Hopcroft, J. E. and Ullman, J. D. (1973). Set merging algorithms. *SIAM Journal of Computation*, 2:294–303.

Howard, W. A. (1980). 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.

Jackson, P. and Reichgelt, H. (1987). A general proof method for first-order modal logic. In *Proceedings of IJCAI*, pages 942–944.

Kanovich, M. I. (1990). Efficient program synthesis in computational models. *Journal of Logic Programming*, 9:159–177.

Kapur, D. and Narendran, P. (1986). NP-completeness of the set unification and matching problems. In *CADE 8*.

Kapur, D. and Narendran, P. (1992). Complexity of unification problems with associative-commutative operators. *Journal of Automated Reasoning*, 9:261–288.

Kleene, S. C. (1951). 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.

Kripke, S. A. (1963). Semantical analysis of modal logic. I. Normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96.

Ladner, R. E. (1977). The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480.

Lenzen, W. (1978). Recent work in epistemic logic. *Acta Philosophica Fennica*, 30(1):1–219.

Lincoln, P. D. and Shankar, N. (1994). Proof search in first-order linear logic and other cut-free sequent calculi. In *LICS*, pages 282–291.

Maehara, S. (1954). Eine Darstellung der intuitionistischen Logik in der Klassischen. *Nagoya mathematical journal*, 7:45–64.

Martelli, A. and Montanari, U. (1982). An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282.

Martin-Löf, P. (1982). Constructive mathematics and computer programming. In Cohen, L. J., editor, *Logic, Methodology and Philosophy of Science VI*, pages 153–175, Amsterdam. North-Holland.

McCarthy, J. and Buvač, S. (1994). Formalizing context (expanded notes). Technical Report STAN-CS-TN-94-13, Stanford University.

McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, volume 4, pages 473–502. Edinburgh University Press, Edinburgh.

Miller, D. (1994). A multiple-conclusion meta-logic. In Abramsky, S., editor, *Proceedings of the International Symposium on Logics in Computer Science*, pages 272–281.

Miller, D., Nadathur, G., Pfenning, F., and Scedrov, A. (1991). Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157.

Mints, G. (1992). *A Short Introduction to Modal Logic*. Number 30 in CSLI Lecture Notes. CSLI.

Moore, R. C. (1985). A formal theory of knowledge and action. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, pages 319–358. Ablex, Norwood NJ.

Morgenstern, L. (1987). Knowledge preconditions for actions and plans. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 867–874, Milan Italy.

Ohlbach, H. J. (1991). Semantics-based translation methods for modal logics. *Journal of Logic and Computation*, 1(5):691–746.

Ohlbach, H. J. (1993). Optimized translation of multi modal logic into predicate logic. In Voronkov, A., editor, *Logic Programming and Automated Reasoning*, volume 698 of *LNCS*, pages 253–264. Springer, Berlin.

Otten, J. and Kreitz, C. (1996). T-string-unification: unifying prefixes in non-classical proof methods. In *TABLEAUX 96*, volume 1071 of *LNAI*, pages 244–260, Berline. Springer.

Plotkin, G. (1972). Building in equational theories. *Machine Intelligence*, 7:73–90.

Pollock, J. L. (1992). How to reason defeasibly. *Artificial Intelligence*, 57(1):1–42.

Rasiowa, H. and Sikorski, R. (1953). Algebraic treatment of the notion of satisfiability. *Fundamenta mathematicae*, 40:62–95.

Reiter, R. (1991). The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press.

Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of IJCAI*, pages 206–214.

Schild, K. (1991). A correspondence theory for terminological logics: preliminary report. In *IJCAI*, pages 466–471.

Schmidt, R. A. (1996). Resolution is a decision procedure for many propositional modal logics. In *AiML*.

Schubert, L. K. (1990). Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In Kyburg, H. E., Loui, R. P., and Carlson, G. N., editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, Boston.

Schulz, K. U. (1993). Word unification and transformation of generalized equations. *Journal of Automated Reasoning*, 11(2):149–184.

Smullyan, R. M. (1968). *First-order Logic*, volume 43 of *Ergebnisse der Mathematik und ihere Grenzgebeite*. Springer-Verlag, Berlin.

Smullyan, R. M. (1973). A generalization of intuitionistic and modal logics. In Leblanc, H., editor, *Truth, Syntax and Modality*, pages 274–293. North-Holland, Amsterdam.

Stone, M. (1996). Representing scope in intuitionistic deductions. Submitted, University of Pennsylvania.

Stone, M. (1997a). Partial order reasoning for a nonmonotonic theory of action. In *AAAI Workshop on Theories of Action*, Providence, RI.

Stone, M. (1997b). Reasoning in natural language generation through fast modal logic programming. Submitted, University of Pennsylvania.

van Benthem, J. F. A. K. (1983). *Modal Logic and Classical Logic*. Bibliopolis, Naples.

Wallen, L. A. (1990). *Automated Proof Search in Non-Classical Logics: Efficient Matrix Proof Methods for Modal and Intuitionistic Logics*. MIT Press, Cambridge.