Towards a Computational Account of Knowledge, Action and Inference in Instructions (Extended Notes)

Matthew Stone
Department of Computer Science and
Center for Cognitive Science
Rutgers, the State University of New Jersey
mdstone@cs.rutgers.edu

Abstract

I consider *abstract instructions*, which provide indirect descriptions of actions in cases when a speaker has key information that a hearer can use to identify the right action to perform, but the speaker alone cannot identify that action. Principled generation of abstract instructions requires a system to assess the inferences about action a user will draw from an instruction. I sketch a framework for specifying, computing, and accessing those assessments in natural language generation.

1 Motivation

This work investigates the inferences hearers need to draw from utterances in dialogue in order to choose appropriate actions to take in the world. Long-studied examples of *indirect speech acts* [Gri75, Sea75] not only call attention to these inferences, but bring home the remarkable fact that speakers *rely on* hearers to make them.

(1) a It's cold in here.
b Close the window!

Thus, under the right circumstances, a hearer has not understood (1a) without drawing the inference to the implicit instruction for action (1b).

It is a mistake to regard such inferential expectations as marginal—as mere polite indirection, for example. For one thing, the utility of inference is pervasive: dialogue proceeds most *efficiently* when contributions accomplish some actions explicitly while selectively omitting others as inferable. [Wal93, GC94, CCC95, You97, DJMT98] offer arguments and models for this. Take the question-answer pair in (2a) and (2b), through which the proposal in (2c) is communicated by inference.

(2) a What should we put in the living room?
b I have a blue sofa for $500.

c Let's put this sofa in the living room.

An indirect utterance like (2b) can improve on a direct proposal like (2c) in that the indirect utterance not only contributes the proposal itself, but also summarizes the proposal's costs (here, the cost in dollars) and its benefits (here, the decorative benefit of a possible color match). This puts the dialogue participants in a better position to evaluate, adopt and carry out the proposal.

What's more, adopting inference as a central mechanism for *implementations* for modules such as natural language generation (NLG) results both in more *streamlined* architectures and in more *flexible* behavior. In [App85], for instance, Appelt proposes to construct sentences by *planning* the hearer's interpretation of them—including the inferences to be drawn from them. The framework treats the whole range of tasks involved in formulating a sentence uniformly and simultaneously—and thereby exploits opportunities for synergy among them:

(3) Remove the bolt with the wrench from the tool box.

In (3), one compact and natural sentence describes the goal of an action (removal), its method (wrench), and the knowledge required to accomplish it (the wrench's location). The sentence planning using description (SPUD) system [SD97, SW98] I describe in section 3.3 also assigns inference about interpretation a central role in generation, albeit a more constrained role than Appelt.

Despite its advantages, inference *is* often viewed as marginal—even dispensable—particularly in practical NLG systems. Such systems rarely aspire to the faithful account of agreement and coordination among participants that make dialogue agents like [Pow77, Hou86, CPB+94] hopelessly explicit and redundant without inference. Having eschewed such complexities of interaction, they can cut corners on the way to credible output. In avoiding inference, they then incur only the drawback of the cumbersome directness and choppy inflexibility suggested in (4):

(4) a I propose we put my sofa in the living room. It is blue. It costs $500.
b Remove the bolt from the housing. Use the wrench. The wrench is in the tool box.

This seems a small price for a feasible implementation. Inference is notoriously difficult to model.

My purpose in this work is twofold. First, in section 2, I will describe how inference is *essential* to explain how certain instructions, *abstract instructions*, achieve the communicative goal of informing the hearer of what to do. Abstract instructions require the hearer to *choose* an appropriate next action by *combining* the information conveyed in the instruction with the hearer's *private* background knowledge. Because the speaker *lacks* this private knowledge, abstract instructions cannot be reformulated to achieve their communicative goal directly. For a practical system to generate instructions without inference in these situations means abandoning a goal-directed view of NLG, with all its advantages.

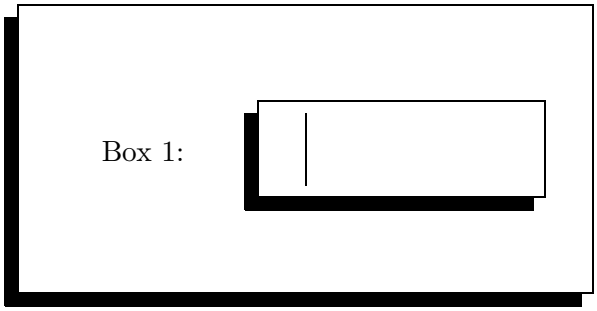

Figure 1: A window where a user might act.

Second, in section 3, I will sketch a computational model of these inferences based on *modal logic programming*. A modal first-order logic of knowledge provides a language in which specifications and queries about choices of action an agent can make have a natural form involving existential quantifiers and hypothetical implications. The logic programming proof procedure for this language allows these queries to be posed and evaluated against these specifications with predictable search. To plan and validate abstract instructions in NLG, we can therefore create a modal specification of the private information of the hearer and the shared information in the context and perform an incremental assessment of sentence interpretation using logic programming queries. With its emphasis on expressive languages and predictable search, this model suggests that accounts of inference in dialogue need not rest on contrived representations or incur unpredictable and unmanageable computational complexity. Thus, a principled account of abstract instructions may soon be a practical one.

2 Examples

Consider a dialogue agent whose task is to assist a user in completing administrative forms correctly. For concreteness, we presume that the process is mediated by a graphical interface which allows the form to be filled out and submitted electronically. At a certain stage, the user might be confronted with some form, including (among other things) the entry box schematized in Figure 1. Suppose the user has asked for instruction about what to do next. The form might make (5) a correct response for the dialogue agent to provide:

(5) Enter your user ID in box 1.

We can easily imagine the user successfully carrying out this instruction, and the dialogue naturally moving on to address other aspects of the form.

This brief scenario masks a range of complexities, starting with the user's action itself. The simple description of (5) notwithstanding, the user's response to the instruction in fact consists of a sequence of concrete steps in which the user moves the mouse to intended locations, and presses intended buttons and keys. We can also regard such actions as organized into broader routines, such as routines to select an intended text box or to type an intended string of

characters, which are fleshed out regularly into intended component actions with mouse and keyboard. This organization might help to account for the choices that the user takes together or reports together; and it might usefully delineate our requirements for knowledge representation.

Either characterization of action places the same key requirement on the user carrying out the instruction. In deciding what to do, the user must select a *particular action* for specific effects that the user *knows* and intends that action to have. In carrying out an action like (5), for instance, this might mean selecting a primitive like *pressing j* next, or a routine like *typing jdoe*, because the user knows a way these actions will contribute to the user's ID (*jdoe*, say) being in box 1.

We see now how (5) requires inference for the user. (5) cannot be regarded as a name for a particular concrete action: the user's interface does not come equipped with a *name* button which automatically ensures that whoever presses its gets their name entered in box 1. In this respect, we can contrast (5) with an alternative such as (6) which might correspond directly to a concrete action for the user.

(6) Type *jdoe* at the cursor.

Obviously, in the case of (5), the user must work backward from the effect specified in the instruction to identify concrete actions which accomplish that effect. In fact, once we fix a framework for representing these concrete actions, we can be quite precise about the knowledge that the user brings to bear in this process. For example, if the user knows a routine to enter a box in a string, it suffices that the user knows her user ID as a string i, and knows which box b is box 1. Using this knowledge, the user should be able to work out that applying the routine to i and b will enter her user ID in box 1.

If *interpreting* (5) requires inference for the user, then *generating* (5) requires the system—or at least its designers—to *anticipate* this inference. The inference itself is inextricably tied to the content the system needs to provide to the user. There might be times when an instruction can be issued in a way that specifies a concrete action directly, as in (6). For example, a system might produce (6) instead of (5), *if it knew* that the user's ID was *jdoe*. However, it seems unlikely that the system *will* know the user's ID when it plans an instruction like (5). For one thing, if the system has the necessary information, it ought to be capable of entering what is needed on its own! So from the system's point of view, the instruction is *abstract*; it determines a concrete action which the user will be able to identify in advance, but which the speaker cannot yet identify. Since the system only knows how to describe the action indirectly, the user's interpretation of the instruction must rely on inference.

If this inference is not anticipated, the result will be a blind system, which pays no heed to whether it gives its users instructions they know how to carry out. Imagine a setting in which user IDs, unbeknownst to the public, also serve as administrative index keys, or in which interface items are given internal identifiers. Then a blind system could offer (7), baffling its users, when it should provide the unproblematic (5).

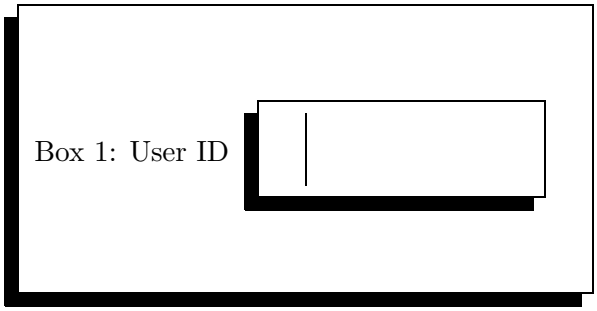

Figure 2: An alternative window.

(7) Enter your administrative index key in widget 55a8.

There are further advantages to making the anticipated inference *explicitly* available to the dialogue system, not just an implicit part of the system's design. It then becomes possible to automatically adjust the formulation of instructions to accommodate the different knowledge of different users at different times. One illustration of this is just to change the prompts in the interface. Figure 2 suggests an alternative labeling for the window of Figure 1. In a large form containing Figure 2, when the user asks what to do next, it becomes possible for the system to reply:

(8) Complete box 1.

With the new label, it becomes part of the common ground that box 1, when completed, will contain the user's ID. The user can therefore select the same concrete action to satisfy (8) as to satisfy (5). Of course, in a large form containing Figure 1, (8) is almost entirely unhelpful. Without the added background, box 1 could be completed by anything.

An explicit representation of inference also enables more flexible interaction. For example, people often signal their failure to understand or accept others' contributions to dialogue by asking a question in response. For a system to reply appropriately to such a followup question, the system must relate the question to the information the system's utterance was intended to convey [LC92, Moo94, Loc95, Loc94]. Information conveyed inferentially is no exception. Here are two ways the user might respond to (5):

(9) a How do I enter my user ID somewhere?
b How will my user ID be stored?

With an explicit representation of the inference required to interpret (5), the system can recognize from (9a) that the user lacks some of the background knowledge required to carry out the instruction. This is the first step in selecting the appropriate additional background and formulating a reply to present it to the user. Meanwhile, that explicit representation could also suggest to the system that (9b) does *not* address the user's ability to carry out the instruction (for which no means of storage is required), but raises another objection—concern over whether the action is sufficiently secure, perhaps. Obviously, an

appropriate response to this concern would require the system to present very different information from background on how to use interfaces.

3 Specifying and Accessing Inferences

Section 2 described *abstract instructions*—a general class of utterances whose generation and interpretation require inferences about knowledge and action. Abstract instructions arise when the speaker *has* key information, from which the user can infer what to do next, but the speaker also *lacks* key information, and cannot formulate an explicit, direct statement of which action to choose. This section sketches how these inferences can be specified (section 3.1), carried out (section 3.2), and accessed in NLG (section 3.3), using first-order multi-modal logic programming.

3.1 Modal representations

First-order multi-modal logic extends the syntax of first-order classical logic by introducing propositional operators of *necessity* and *possibility*. These operators extend classical logic in a very general way. Formulas in first-order classical logic characterize ordinary entities once-and-for-all. In contrast, the necessary formulas of modal logic provide characterizations of entities that take only restricted information into account. For example, a *temporal* modal logic might use $[\text{N}]p$ to introduce a characterization p that takes into account only what will be true after the next action takes place. Meanwhile, an *epistemic* modal logic might use $[\text{K}]p$ to indicate that p takes into account only the information that an agent knows. (Thus, I write necessity operators in the form [NAME].) To characterize a conversation from the point of view of a dialogue agent, we can use three modal operators: [S] for what the system knows, [U] for what the user knows, and [CG] for the content of the common ground in the conversation.[1]

We will combine these epistemic modal operators with a formalization of action inspired by the situation calculus to describe agents' ability [Moo85, Sto98a]. We introduce terms for situations and actions (executed, we assume, by the user); we write $do(s, \alpha, s')$ if situation s' can result if action α occurs in situation s. Recall from section 2 that in deciding what to do, the user must select a *particular action* for specific effects that the user *knows* and intends that action to have. Suppose the effect is a predicate E of situations; we can now elaborate this condition as (10), using the ontology of the situation calculus and modal logic.

[1] [Fit83, FM98] provide good technical introductions to modal proof, while [FHMV95] is an introduction to modal logic in AI. We assume an idealization of conversation in which [S] [U] and [CG] satisfy these axiom schemes:

$$[\text{S}]p \supset p \quad [\text{U}]p \supset p \quad [\text{CG}]p \supset p$$
$$[\text{S}]p \supset [\text{S}][\text{S}]p \quad [\text{U}]p \supset [\text{U}][\text{U}]p \quad [\text{CG}]p \supset [\text{CG}][\text{CG}]p$$
$$[\text{CG}]p \supset [\text{S}]p \quad [\text{CG}]p \supset [\text{U}]p$$

(10) In situation s, there is an action α about which the user knows, that E is true in any situation s' resulting from doing α in s.

This corresponds to the formula in (11).

(11) $\exists\alpha\ [\text{U}]\ \forall s'\ (do(s, \alpha, s') \supset E(s'))$

Consider the problem of interpreting instruction (5) by reasoning about knowledge and action. Informally, we used these premises to characterize the shared background on which this reasoning depends.

(12) a The user knows her user ID as a string.
b The user (jointly with the system) knows which is box 1.
c The user can enter a string in a box.

(13) provides the corresponding formalizations, drawing on the definition of ability in (11) and making explicit the shared status of this information.

(13) a $[\text{CG}]\ \exists i\ [\text{U}]\ (userid(u, i) \wedge string(i))$
b $\exists b\ [\text{CG}](box(b) \wedge number(b, 1))$
c $[\text{CG}]\ \forall sib\ (string(i) \wedge box(b) \supset \exists\alpha\ [\text{U}]\ \forall s'\ (do(s, \alpha, s') \supset in(i, b, s')))$

Meanwhile, to anticipate the interpretation of the instruction, the system needs the conclusion that it is part of the common ground that the user can enter her user ID in box 1; we draw on (11) to formalize it thus:

(14) $[\text{CG}]\ \exists ib\alpha\ [\text{U}]\ (userid(u, i) \wedge box(b) \wedge number(b, 1)) \wedge$
$[\text{U}]\ (\forall s'\ (do(s, \alpha, s') \supset in(i, b, s')))$

The semantics of modal logic will ensure that (14) always holds when (13) does; the logic programming proof system for modal logic we explore in section 3.2 provides a way of demonstrating this automatically.

3.2 Modal logic programming

Logic programming languages embody simple, specific search procedures for building proofs. At each step in logic programming search, the goal is to find a way to use the available assumptions to establish a specific query. If the query is complex, its logical structure directly determines the available alternatives for search. Thus, logical symbols in queries can be seen as instructions for decomposing and transforming the search problem that the interpreter faces. Similarly, the atomic formulas that an assumption can be used to derive—the *head* (or heads) of that assumption—serve as indexes that regulate whether an assumption can be applied. And the logical structure of the assumption provides an instruction for creating a set of new search problems whenever the assumption is used.

This general perspective on logic programming has been formalized and analyzed under the framework of *abstract logic programming languages* [MNPS91]. Mathematically, this formalization begins by establishing a correspondence between search problems posed to the logic programming interpreter and *sequents* in a proof calculus.

A sequent is an expression of the form $\Gamma \longrightarrow \Delta$ where Γ and Δ are finite multisets of formulas. For the purposes of proof, a sequent represents the judgment that if *all* of the *assumption* formulas in Γ are true, then *some* of the *conclusion* formulas in Δ must also be true. A *sequent proof* is a tree of sequents of a distinguished form. The leaves take the form $\Gamma, A \longrightarrow \Delta, A$—so that some assumption is directly matched against some conclusion. The interior nodes instantiate inference figures, which relate proof problems of larger logical formulas to proof problems for constituent formulas. For a simple example, the figure for a conjunctive conclusion is:

$$\frac{\Gamma \longrightarrow A, \Delta \qquad \Gamma \longrightarrow B, \Delta}{\Gamma \longrightarrow A \wedge B, \Delta} \tag{15}$$

Read upward, it says that you can conclude that $A \wedge B$ (or Δ) must be true (on assumptions Γ) provided that you can conclude that A (or Δ) must be true (on assumptions Γ) and provided that you can conlude that B (or Δ) must be true (on assumptions Γ).

A logic programming search problem can be represented as a sequent because the *program* can be represented with the assumptions Γ and the *goal* or *query* can be represented with the conclusion(s) Δ. The action of the interpreter in transforming search problems can then be seen as the construction of a proof in a restricted sequent calculus—for example, breaking down a conjunctive goal into two separate subgoals in a logic programming interpreter corresponds to an application of the sequent rule of (15). The rules of the logic programming calculus are specialized so as to model the constrained search entertained by the logic programming interpreter. The key result required to show the correctness of a logic programming language in this framework is then to show that the restricted sequent calculus permits a derivation of a goal exactly when the goal is provable in a general sequent calculus for the logic.

[Fit83] presents a number of modal sequent calculi. (Strictly speaking, he presents *tableau* calculi, a notational variant of sequent calculi.) A calculus such as his *prefixed* tableau calculus is required to implement a logic programming language with programs that specify partial information, including the program with existential quantifiers proposed in (13). A version of this calculus, specialized for logic programming search, is developed in [Sto98b].

The resulting logic programming interpreter derives (14) from (12) as follows. It begins by decomposing the goal: it introduces a world w representative of the common ground, introduces variables I, B and A for individuals that exist at w, and (treating the first conjunct) introduces an additional world w' representative of what the user knows at w. The goal now decomposes into atomic formulas $\mathit{userid}(u, I)$, $\mathit{box}(B)$ and $\mathit{number}(B, 1)$, which are established at w' by matching against the corresponding components of the background clauses and solving the resulting simple constraints on values of variables and on possible worlds. Next (treating the second conjunct), the interpreter introduces a new world w'', introduces a fresh situation $c_{s'}$ at w'' to interpret the universal quantifier, and assumes $do(s, A, c_{s'})$. The goal is now $in(I, B, c_{s'})$ at w''; we match against the clause describing the user's general knowledge of entering strings in boxes (again solving a simple resulting constraint on worlds

and variables). This yields three more atomic modal goals—*string*(I), *box*(B) (at some variable world x), and $(s, A, c_{s'})$ at w''—which are dispatched in turn. In fact, for this simple query and simple program, logic programming proof search is completely deterministic.

A key feature of the logic programming calculus is its *modular* treatment of modal connectives. A necessary goal [x]g can be seen as a *modular* goal because, in modal logic, only program clauses of the form [x]p can contribute to its proof. Under certain constraints on modal semantics (met here), modularity also brings locality: a goal [x]$(p \supset g)$ introduces a *local* assumption p that can only contribute to the proof of g. At the same time, an indefinite program [x]$(p \vee q)$ introduces a *local* ambiguity that may be restricted to the proof of some modular goal [x]g. This modularity is part and parcel of correct modal reasoning—it is part of what it means for modal formulas to take only restricted information into account. But modularity is useful in its own right, to constrain logic programming search and describe the modular structure of logical specifications [Mil89, GM94].

3.3 Modal queries in sentence planning

Sections 3.1 and 3.2 have shown how inferences about knowledge and action can be specified and derived. It remains to be seen how these conclusions might actually be *accessed* in sentence planning. In this section, I suggest one answer, inspired[2] by the SPUD system for sentence generation [SD97, SW98].

SPUD adopts a view of sentence generation as goal-directed activity, like [App85, Dal92] before it. On this view, the task of the generator is to use the words and constructions of the language to design a message that fulfills a set of communicative intentions. SPUD works with two kinds of intentions in particular: intentions to uniquely identify the entity designated by a referring expression, and intentions to establish a proposition as part of the content of the conversation. SPUD fulfills these intentions incrementally, using a grammar in the LTAG formalism [JLT75, Sch90] to add units of meaning and syntax word-by-word into an incomplete sentence. Thus, unlike [App85], SPUD does not attempt the exhaustive search of the grammar performed in unrestricted planning.

The SPUD algorithm is outlined in Figure 3. The steps of this algorithm refer to abstract notions—ambiguity of reference, information conveyed, appropriateness and specificity of descriptors—whose implementation depends on an approach to meaning and interpretation based on inference in logic programming. In this approach, the meaning of a sentence is represented in the same terms as the background modal specification of conversational state. Key stages of NLG reason about sentence interpretation by constructing and evaluating logic programming queries in which these meanings are combined with background information. What follows describes this approach in more detail.

I begin with the representation of sentence meaning. The grammar is designed to deliver the content of each sentence in two parts: the *presupposition*

[2] SPUD itself cannot yet generate (5), because the implementation makes some assumptions about discourse referents which (5) does not satisfy.

- Start with a tree with one node (e.g., S, NP) and one or more referential or informational goals.
- While the current tree is incomplete, or its references are ambiguous to the hearer, or its meaning does not fully convey the informational goals (provided progress is being made):
  - consider the trees that extend the current one by the addition (using LTAG operations) of a true and appropriate lexicalized descriptor;
  - rank the results based on local factors (e.g., completeness of meaning, distractors for reference, unfilled substitution sites, specificity of licensing conditions);
  - make the highest ranking new tree the current tree.

Figure 3: An outline of the SPUD algorithm

$P\mathbf{x}$—an open formula containing free occurrences only of the variables in the sequence $\mathbf{x}$—which places a requirement on the context and the *assertion* $N\mathbf{x}$—another open formula containing only occurrences of variables in the sequence $\mathbf{x}$—which contributes new information to the evolving discourse.

The use of the term presupposition follows the theoretical perspective of [vdS92] that presuppositions are *anaphors* that are resolved against an evolving model of discourse. This view modulates the received view of a presupposition, from Frege and Russell, as a statement of the uniqueness conditions under which a sentence refers successfully. For example, the received view famously represents the uniqueness condition as the presupposition (16b) of (16a):

(16) a The King of France is bald.
b $\exists x(\mathit{kof}(x) \wedge \forall y(\mathit{kof}(y) \supset x = y))$
c $\mathit{kof}(x)$
d $\mathit{bald}(x)$

The anaphoric view replaces the formula (16b), with its logical complexity, by the formula (16c) and a complex process of *resolution* against the context. Resolving (16c) requires not only showing that the logical condition is satisfied in the common ground but also providing a discourse referent from the context that can serve as the value for the variable x. For (16c), the uniqueness derives from the fact that the speaker and the hearer must agree on how the presupposition is resolved; thus this resolution of presupposition parallels the resolution of other anaphoric elements in the sentence.

This view of the resolution of presupposition also explains why the assertions of sentences contain free variables. Variables in the assertion provide a way of recovering and commenting on the values of variables evoked by the presupposition. For example, the assertion of (16a) can be represented as (16d): once we find a value k for x by resolving the presupposition, we can take x to

refer to k in adding the assertion to the context.

For (5), we can analyze the instruction in terms of presupposition and assertion as if its meaning is *you should take the action of entering your user ID in box 1.* That is, the presupposition is (17).

(17) a action α enters user ID i of yours in box b numbered 1
b $P(\alpha, i, b) \equiv userid(u, i) \wedge box(b) \wedge number(b, 1)) \wedge$
$\forall s' \; (do(s, \alpha, s') \supset in(i, b, s'))$

The assertion is (18)

(18) a you should do action α
b $o(\alpha)$

Having laid out the representation of meaning, we turn to logical queries that use this representation to assess sentence interpretation. We begin with the interpretation of presupposition.

(19) a A *possible resolution* of the presupposition $P\mathbf{x}$ is a proof of $[\textsc{cp}]\exists\mathbf{x}P\mathbf{x}$.
b A presupposition is resolved successfully when there is a unique (maximally salient) possible resolution.

It is clear why (19) appeals to the modal operator [CP]—this implements the requirement that only the shared common ground can be taken into account in resolving a presupposition. (In cases of accommodation where presupposition and common ground seem to disagree [Lew79], we can follow Lewis in assuming it is the content of the common ground that is adjusted, not the requirement imposed by the presupposition.) The narrow scope of $\exists\mathbf{x}$ in (19), meanwhile, indicates that the resolution requires us to find mere discourse referents to satisfy the presupposition—not concrete entities in the world, as would be required if the quantifier was given wide scope. The difference is relevant in (5): the system knows there must be an action α that enters the user's name, but does not know a specific action that does so. The proof outlined in section 3.2 effectively provides a possible resolution of the presupposition (17); nothing changes when we dispense with the introduction of worlds for $[\textsc{u}]p$ and prove p instead. As we observed that there was no nondeterminism in search, we can see that this proof is in fact a successful resolution as well.

Finally, we turn to the assertion. In assessing sentence interpretation, the system will have to determine whether the sentence (with assertion $A\mathbf{x}$ and presupposition $P\mathbf{x}$) will license the inference to some particular proposition q. To make that assessment, the system can envisage the consequences of communicating the sentence. That is, the system restricts attention to developments of the situation compatible with what the system knows, as represented by the content of [S]. The system supposes further that the content of the sentence has been communicated. Since the sentence content links to whatever discourse referents are evoked by the presupposition, that content can be represented by $\forall\mathbf{x} \; (P\mathbf{x} \supset A\mathbf{x})$. The system's supposition is thus that $[\textsc{cg}] \; \forall\mathbf{x} \; (P\mathbf{x} \supset A\mathbf{x})$ holds. Then, in that hypothetical context, the system tests whether q can also

be taken as part of the common ground. These operations correspond to the query

(20) $[\text{S}]\ ([\text{CG}]\ \forall \mathbf{x}\ (P\mathbf{x} \supset A\mathbf{x}) \supset [\text{CG}]q)$

So proving (20) suggests that the inference to q is licensed.

In the case of (5), the key inference is that the user knows what she should do next: $\exists\alpha\ [\text{U}]\ o(\alpha)$. Thus we apply (20) to (5) as (21).

(21) $[\text{S}]\ ([\text{CG}]\ \forall \alpha i b\ (P(\alpha, i, b) \supset o(\alpha)) \supset [\text{CG}]\exists\alpha\ [\text{U}]\ o(\alpha))$

(21) is straightforwardly provable. After matching the goal $o(\alpha)$ against the hypothesized inference rule, the proof simply repeats the argument given in section 3.2 that the user knows what will enter her name in box 1.

4 Conclusion

Enter your user ID in box 1 is a very simple instruction. Yet it places an almost paradoxical requirement on a system issuing it. The system means this description to identify an action to the user. Thus, to be confident in the instruction, the system must know that the user can select the right action using the description. Yet, normally the system will not know what the user will enter to fulfill the directive. In this sense, the system cannot know what the user will do.

Reconciling these requirements depends on the system's representing different states of knowledge explicitly and reasoning about them correctly. I have suggested one possible such reconciliation, based on using logic programming inference in first-order multi-modal logic to evaluate semantic interpretation.

Acknowledgments

This paper draws heavily from my dissertation [Sto98b], with all the input that entails, particularly from my advisor Mark Steedman and committee members Aravind Joshi, Rich Thomason, Bonnie Webber and Scott Weinstein. This presentation benefits from input of audiences at the University of Edinburgh, University of Brighton and AT&T Labs Research, and was made possible by a postdoctoral fellowship from RUCCS and the organizers of ICOS-1.

References

[App85] Douglas Appelt. *Planning English Sentences*. Cambridge University Press, Cambridge England, 1985.

[CCC95] Jennifer Chu-Carroll and Sandra Carberry. Response generation in collaborative negotiation. In *Proceedings of ACL*, pages 136–143, 1995.

[CPB+94] Justine Cassell, Catherine Pelachaud, Norm Badler, Mark Steedman, Brett Achorn, Tripp Becket, Brett Douville, Scott Prevost, and Matthew Stone. Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *SIGGRAPH*, pages 413–420, 1994.

[Dal92] Robert Dale. *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes*. MIT Press, Cambridge MA, 1992.

[DJMT98] Barbara Di Eugenio, Pamela W. Jordan, Johanna D. Moore, and Richmond H. Thomason. An empirical investigation of proposals in collaborative dialogue. In *Proceedings of COLING-ACL*, 1998.

[FHMV95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, Cambridge MA, 1995.

[Fit83] Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library*. D. Reidel, Dordrecht, 1983.

[FM98] Melvin Fitting and Richard L. Mendelsohn. *First-order Modal Logic*, volume 277 of *Synthese Library*. Kluwer, Dordrecht, 1998.

[GC94] Nancy Green and Sandra Carberry. A hybrid reasoning model for indirect answers. In *Proceedings of ACL*, pages 58–65, 1994.

[GM94] Laura Giordano and Alberto Martelli. Structuring logic programs: A modal approach. *Journal of Logic Programming*, 21:59–94, 1994.

[Gri75] H. P. Grice. Logic and conversation. In P. Cole and J. Morgan, editors, *Syntax and Semantics III: Speech Acts*, pages 41–58. Academic Press, New York, 1975.

[Hou86] George Houghton. *The Production of Language in Dialogue: A Computational Model*. PhD thesis, University of Sussex, 1986.

[JLT75] Aravind K. Joshi, L. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of the Computer and System Sciences*, 10:136–163, 1975.

[LC92] Lynn Lambert and Sandra Carberry. Modeling negotiation subdialogues. In *Proceedings of ACL*, pages 193–200, 1992.

[Lew79] David Lewis. Scorekeeping in a language game. In *Semantics from Different Points of View*, pages 172–187. Springer Verlag, Berlin, 1979.

[Loc94] Karen E. Lochbaum. *Using Collaborative Plans to Model the Intentional Structure of Discourse*. PhD thesis, Harvard University, 1994.

[Loc95] Karen E. Lochbaum. The use of knowledge preconditions in language processing. In *Proceedings of IJCAI*, pages 1260–1266, 1995.

[Mil89] Dale Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6(1–2):79–108, 1989.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Moo85] Robert C. Moore. A formal theory of knowledge and action. In Jerry R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*, pages 319–358. Ablex, Norwood NJ, 1985.

[Moo94] Johanna Moore. *Participating in Explanatory Dialogues*. MIT Press, Cambridge MA, 1994.

[Pow77] Richard Power. The organisation of purposeful dialogues. *Linguistics*, 17:107–152, 1977.

[Sch90] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, Computer Science Department, University of Pennsylvania, 1990.

[SD97] Matthew Stone and Christine Doran. Sentence planning as description using tree-adjoining grammar. In *Proceedings of ACL*, pages 198–205, 1997.

[Sea75] John R. Searle. Indirect speech acts. In P. Cole and J. Morgan, editors, *Syntax and Semantics III: Speech Acts*, pages 59–82. Academic Press, New York, 1975.

[Sto98a] Matthew Stone. Abductive planning with sensing. In *AAAI*, pages 631–636, Madison, WI, 1998.

[Sto98b] Matthew Stone. *Modality in Dialogue: Planning, Pragmatics and Computation*. PhD thesis, University of Pennsylvania, 1998.

[SW98] Matthew Stone and Bonnie Webber. Textual economy through close coupling of syntax and semantics. In *Proceedings of INLG*, pages 178–187, 1998.

[vdS92] Rob van der Sandt. Presupposition projection as anaphora resolution. *Journal of Semantics*, 9(2):333–377, 1992.

[Wal93] Marilyn A. Walker. *Informational redundancy and resource bounds in dialogue*. PhD thesis, Department of Computer & Information Science, University of Pennsylvania, 1993. Institute for Research in Cognitive Science report IRCS-93-45.

[You97] R. Michael Young. *Generating Concise Descriptions of Complex Activities*. PhD thesis, University of Pittsburgh, 1997.