Belief Representation and Quasi-Indicators

William J. Rapaport

**Department of Computer Science and Engineering,
Department of Philosophy, Department of Linguistics,
and Center for Cognitive Science
State University of New York at Buffalo, Buffalo, NY 14260-2000**

rapaport@cse.buffalo.edu

http://www.cse.buffalo.edu/∼rapaport/

August 1984

Abstract

This thesis is a study in "knowledge" representation, specifically, how to represent beliefs expressed by sentences containing quasi-indicators. An *indicator* is a personal or demonstrative pronoun or adverb used to make a strictly demonstrative reference. A *quasi-indicator* is an expression that occurs within an intentional context and that represents a use of an indicator by another speaker. E.g., if John says, "I am rich", then if *we* say, "John believes that he himself is rich", our use of 'he himself' is quasi-indexical. Quasi-indicators pose problems for natural-language question-answering systems, since they cannot be replaced by any co-referential noun phrases without changing the meaning of the embedding sentence. Therefore, the referent of the quasi-indicator must be represented in such a way that no invalid co-referential claims are entailed.

I discuss the origin of the problem of quasi-indicators in philosophy of language, and the lack of recognition of its importance by researchers in knowledge representation. I show why one natural treatment of such pronouns in knowledge representation is both philosophically and logically incorrect. I then propose a new treatment, show that it is correct, and modify an augmented transition network grammar to parse appropriate sentences into a semantic-network representation (using the Semantic Network Processing System). This is then linked to the inference mechanism of a question-answering system, giving it the capability of correctly handling information about people's beliefs about themselves and others.

The data base of the system is assumed to contain information about various people (including its users), among which is information about the beliefs (and desires, wants, or other intentional attitudes) of these people. Such a data base constitutes the "knowledge" (more accurately, the beliefs) of the system about these people and about their beliefs. The objects of intentional attitudes are intensional (i.e., non-extensional) entities. Moreover, the entities represented in the data base are the objects of the system's beliefs, and, so, are also intentional, hence intensional. A data structure ideally suited to dealing with such entities is a propositional semantic network. Thus, a second issue concerns testing the hypothesis that the nodes of semantic networks represent intensional entities by investigating the appropriateness of such networks for dealing with representations of beliefs.

Among the interrelated issues in knowledge-representation that can be raised in such a context are those of multiple reference and the proper treatment of pronouns. Solutions to problems of multiple reference turn out to be a side effect of a solution to the subtler problem of pronominal—in particular, quasi-indexical—reference.

Note: This document is a LaTeX version of a thesis submitted to the Faculty of the Graduate School of State University of New York at Buffalo in partial fulfillment of the requirements for the degree of Master of Science. Some spelling errors have been corrected (and hopefully no new ones introduced!) and some updated bibliographic information has been added. The illustrations are copied from the original document and appear at the end of this version. This document originally appeared (in a slightly different format) as:

> Rapaport, William J. (1984), "Belief Representation and Quasi-Indicators", *Technical Report 215* (Buffalo: SUNY Buffalo Department of Computer Science, August 1984).

and was subsequently published in a revised version as:

> Rapaport, William J. (1986), "Logical Foundations for Belief Representation", *Cognitive Science* 10: 371–422.

A sequel was published as:

> Rapaport, William J.; Shapiro, Stuart C.; & Wiebe, Janyce M. (1997), "Quasi-Indexicals and Knowledge Reports", *Cognitive Science* 21: 63–107.

PREFACE

It is my intention in this essay to illustrate the importance of philosophy for research in Artificial Intelligence and the correlative importance of a knowledge of Artificial Intelligence for philosophical research. It is, thus, an essay in the spirit of Dennett's recommendations:

> Philosophers, I have said, should study AI. Should AI workers study philosophy? Yes, unless they are content to reinvent the wheel every few days. When AI reinvents a wheel, it is typically square, or at best hexagonal, and can only make a few hundred revolutions before it stops. Philosopher's wheels, on the other hand, are perfect circles, require *in principle* no lubrication, and can go in at least two directions at once. Clearly a meeting of minds is in order. (Dennett 1978: 126.)

I wish to express my gratitude to State University of New York, College at Fredonia, for a leave of absence that enabled me to begin the work culminating in this research; to my former colleagues in the Department of Philosophy at Fredonia for their inspiration, encouragement, and philosophical companionship; to Hector-Neri Castañeda for his wisdom and for teaching me philosophy; to Stuart C. Shapiro for his advice and for teaching me computer science and AI; and to my past and present colleagues in the Semantic Network Research Group for their comments, criticism, and help with hardware and software, especially Ernesto Morgado, Terry Nutter, and Joao Martins.

This thesis is dedicated to my parents.

1 AN OVERVIEW

1.1 Belief Representation and Knowledge Representation

This is an essay on belief representation.

The branch of artificial intelligence (AI) and computer science known as "knowledge representation" is concerned with how to represent "knowledge of the application environment … and knowledge of the intended audience" in a computer system (cf. McCalla & Cercone 1983: 12). The name of this field is fundamentally misleading. A more neutral term would be something like 'information representation' or 'data representation'. When knowledge representation is taken to be more generally computer-science oriented than AI-oriented, either of these terms would be more appropriate.

But when it is taken as a subject concerned with the representation of information in AI systems, its epistemic connotation comes to the fore. In this context, however, more than mere knowledge is being represented. For what is represented are such things as objects, properties, situations, and propositions. While these *can* be the objects of knowledge (one can know an object, or know what it is; one can know what properties it has; one can know that a certain proposition is true, etc.), they are, more generally, the objects of *belief* and *acquaintance*—in general, the objects of *thought*.

The distinction is an important one, for one can think about things that do not exist and one can believe propositions that are, in fact, false (cf. Meinong 1904, Rapaport 1978). But one cannot *know* a false proposition. Yet, if an AI system is to simulate a mind (or perhaps *be* a mind) or merely interact with humans, it must be provided with ways of representing non-existents and falsehoods. Since belief is a part of knowledge and has a wider scope than knowledge, the term *belief representation* is a more appropriate one in an AI context. This is one sense in which this is an essay on belief representation.

More centrally, it is also an essay on how to represent the beliefs of intelligent agents. The ultimate goal is the construction of an AI system that can reason about the cognitive states of intelligent agents. These agents include people (e.g., the system's users), other AI systems (e.g., interacting ones), and itself. (Cf. Nilsson 1983: 9.) The cognitive states include goals, desires, hopes, and knowledge, in addition to beliefs. Here, I shall be concerned only with beliefs, partly because belief plays a central logical and psychological role in the network of cognitive states, partly because it is in many ways simpler to analyze than other cognitive states, and partly to be able to build on the large philosophical and computational literature about belief.

1.2 A Belief-Representation System

The sort of AI system that is of concern here is to be a deductive question-answering system whose data base contains information about the world and about various cognitive agents. In order for the system to learn more about these agents (and the world)—to expand its "knowledge" base—it should contain information about their beliefs and be able to reason about them. Such a data base constitutes the beliefs of the system about these agents and about their beliefs.

Since each of the agents is in fact such a system itself, each has beliefs about the beliefs of the others. Thus, our system must be able to represent (i.e., have beliefs about) beliefs about beliefs and to reason about these. Such beliefs have variously been referred to as *iterated*, *nested*, or *recursive* beliefs.

A belief-representation system must also be sensitive to the *intensionality of belief* and to the associated phenomenon of *referential opacity.* I shall have more to say about this in Section V. For now, it suffices to note two important points:

First, the intensionality of belief puts constraints on the system's deduction mechanism. For instance, given the system's beliefs that an agent, *A*, believes some proposition *p* and that *p* is logically equivalent to another proposition *q*, the system should not infer that *A* believes *q*, in the absence of further information.

Second, an agent can have inconsistent beliefs about an object. For instance, *A* might believe both that the Evening Star is a planet and that the Morning Star is not a planet, even though the Morning Star *is* the Evening Star. This can happen as long as *A* does not believe that the Morning Star is the Evening Star. In this case, *A*'s "data base" contains *two* items, one for the Morning Star, one for the Evening Star. Such items are *intensional objects*, and our AI system must be able to deal with them.

The central part of this essay is a discussion of another requirement for such an AI system—one that has not been discussed in previous computational literature: The system must be sensitive to the *indexicality* of certain beliefs, in particular, to the phenomenon of *quasi-indexicality.* This, too, will be dealt with in much greater detail later (in §4). Briefly, it is a feature that is at the core of *self-referential* beliefs—i.e., beliefs about oneself—and their expression by others. Thus, the belief that *A* would express by "I am rich" must be reported by someone else thus: "*A* believes that *he* is rich"; it clearly should not be reported as "*A* believes that *I* am rich".[1]

Finally, the system ought to be able to expand and refine its beliefs by interacting with users in ordinary conversational situations. This is not strictly a requirement. Users could be required to learn a rigid, canonical language for unambiguously expressing beliefs and to use this language with the system. Indeed, the first stage of the present system will be like this, and it is the one presented here. But if the system is to be considered as a cognitive agent, and especially if it is to be used as a tool in understanding *our* belief-representation mechanisms, it ought to interpret ordinary statements about belief, expressed in (grammatical) natural language, the way humans do. Thus, we would want the system to make reasonable or plausible interpretations of users' belief reports—based on such things as subject matter and prior beliefs (including beliefs about the user and the user's beliefs)—and to modify its initial representation as more information is received.

1.3 Methodology

The methodology I shall follow is, roughly, to divide, simplify, conquer, complicate, and then re-conquer. I shall begin by examining ways of representing two distinct kinds of belief reports (*de re* and *de dicto* reports) and the special report (*de se* reports) involving quasi-indexical reference. These reports will be given in English using canonical representations and will be translated into a semantic-network representation. The translations can be done (see §8 and the Appendix) using the Semantic Network Processing System (SNePS; Shapiro 1979) and an ATN parser-generator with a deductive question-answering capability (Shapiro 1982). The result of this is a computational study of the *logic of belief.*

It is important to note that here I am only concerned with the *logic* of belief reports expressed in a canonical language; thus, I shall *not* be concerned with the *pragmatic* problems (in one sense of that word) of determining from context and prior belief *which* canonical representation was intended by a speaker. It is also important to note that here I am only concerned with the logic of *belief*; thus, I shall not deal with the issue of how different objects of belief—different topics of conversation—might affect the interpretation of a natural-language belief report.

I view these issues roughly as issues of *performance*, as opposed to the issues of *competence* that, I feel, must be dealt with first. Once we see how to represent beliefs in clear cases, we can then turn to the more complex linguistic, psychological, and pragmatic issues of the interpretation of ordinary language.

1.4 Outline of This Essay

In the next section, I return to motivational issues, discussing the importance of reasoning about beliefs and the importance of nested beliefs. Following this, I survey some of the previous computational literature. I next turn to a discussion of the nature and importance of quasi-indexical reference. This will set the stage for the presentation of the representation of beliefs and a discussion of inference, followed by a presentation of the current implementation of the system. I conclude with a look at some problems and the tasks that remain unfinished. An appendix contains the ATN grammar and LISP functions that translate between (a canonical fragment of) English and SNePS networks.

[1]Nor should it be reported as "*A* believes 'I am rich' ". For arguments to this effect, see Church 1950, Feldman 1977, and Cresswell 1980.

2 MOTIVATION

In this section, several arguments will be presented for the importance of representing and reasoning about beliefs in general and about nested beliefs in particular. We shall also briefly look at the importance of beliefs involving quasi-indicators (although they will be considered in more detail in §4).

2.1 The Importance of Beliefs

Beliefs play at least three important roles, both for humans and in AI systems: They have an *evidentiary* role, a role in *explaining behavior*, and a role in *producing* (appropriate) *behavior*.[2]

2.1.1 The Evidentiary Role of Beliefs

Beliefs, as well as other cognitive attitudes (such as knowledge), can play an evidentiary role. That is, one's own beliefs, as well as the beliefs of others, can be used as evidence or premises for coming to believe propositions.

As an example,[3] to decide whether I should believe *p*, I might reason as follows:

> John believes *q*.
> I believe what John believes.
> I believe that *p* is logically equivalent to *q*.
> Therefore, I should believe *p*.

Thus, if an AI system is going to be able to increase and refine its data base—its "beliefs"—it ought to be able to represent and reason about its own beliefs as well as those of its users.

2.1.2 The Explanatory Role of Beliefs

Actions can be either intentional or unintentional. That is, an action can be the result of a (conscious) decision, or it might be merely accidental. In the former case, it would be the end result of a chain of reasoning that would include beliefs. An AI system (perhaps an intelligent robot) that would be capable of performing actions, or even a system that would be capable of recommending actions to its users, would thus need to be able to deal with beliefs.

When a belief is a cause of a person's actions, we are not only interested in *what* the person believes, but also in *how* the person believes it. That is, we are not only interested in a third-person characterization of the agent's beliefs, but also in the agent's *own* characterization of those beliefs.[4] The distinction between these two ways of reporting beliefs is captured by means of the distinction between *de re* and *de dicto* belief representations. Thus, an AI system that is capable of explaining or recommending behavior must be able to distinguish between these two kinds of belief reports by having two distinct means of representing them. I shall describe these two kinds of belief reports in more detail in §5.1.2.

2.1.3 The Behavior-Producing Role of Beliefs

Several AI systems (e.g., HAM-ANS (Wahlster 1984, Marburger *et al.* 1984), GRUNDY (Rich 1979), UC (Wilensky, *et al.*, 1984)) employ the technique of *user modeling* in order to produce appropriate natural-language output. An interactive dialogue system can build a profile of each user in order to tailor its output to the user's needs. This profile might include such information as:

(1) the user's degree of familiarity with the topic (as in UC, where the naive user must be distinguished from the expert),

(2) the user's current "state of knowledge" (more accurately, the user's current set of beliefs, whether or not they accurately reflect the world),

[2]The first two of these were stressed by John Perry in his lectures on Situation Semantics during the COLING-84 Summer School, Stanford University, 1984.

[3]Adapted from Perry's lectures.

[4]Some of the points in this section are adapted from Perry's lectures.

(3) the user's interests, and

(4) the mutual (or shared) beliefs of the system and user (in order to be able to deal with such questions as whether they are talking about the same objects or have the same beliefs).

(Cf. Wahlster 1984). Not only would such a representation of the user's beliefs (and other cognitive attitudes) be used in determining the *level* of the dialogue (e.g., whether the system's output is to be used by a novice or an expert), but also in determining the *quality* and *quantity* of the information output (e.g., whether the system should provide only literal answers, whether it should provide more discursive ones that give slightly more—or less—information in order not to be misleading; cf. Joshi *et al.*, 1984).

2.1.4 Belief Spaces

The discussions in the previous sections lead naturally to the notion of *belief spaces* (Martins 1983), or *views* (Konolige 1984), or *belief contexts.* The central idea behind each of these notions is the set of beliefs of an agent. In this essay, I consider the set of propositions believed by an agent, together with the set of items about which the agent has beliefs, *relativized to a "reporter" of these beliefs*, to be the *belief space of the agent in the context of the "reporter".*

The items about which an agent has beliefs need not exist; they might be co-referential (if they do exist)—as in the Morning Star/Evening Star case—and they might be the same as or different (or differently represented) from the items in another agent's belief space. Moreover, with the possible exception of the system's own beliefs, the belief space of an agent *as represented by the system* will contain, *not* the agent's (own) representations of the objects of his beliefs, but the *system's* representations of the agent's representations of them. And, in the case of nested beliefs, the objects of an agent's beliefs would be represented, not as the agent represents them, but as another agent would represent (report) them.

Thus, some care must be taken in the choice of a representational scheme for an AI system that can reason about beliefs.

2.2 The Importance of Nested Beliefs

Discussions of nested beliefs such as

(5) John believes that Mary believes that Lucy is rich

occasionally result in the observation that the speaker must be joking, that no one really talks that way. While there are, no doubt, performance limitations on the allowable depth of such nesting, our linguistic competence clearly allows for the grammatical (if not acceptable) formulation of such sentences. Any sentence can be prefixed by an operator of the form '*A* believes that' (where '*A*' names a cognitive agent). Indeed, Kant held that *all* statements were within the scope of an implicit 'I think that' operator (Kant 1787: B131), so even as simple a statement as

(6) John believes that $1+1=2$

is really of the form

(6K) I (the speaker) think that John believes that $1+1=2$.

But the performance limitation must be taken seriously. The feeling that a sentence such as (5) must be a joke raises the question of how deep the nesting actually can be in ordinary cases. As Dennett puts it,

> A *first-order* intentional system has beliefs and desires (etc.) but no beliefs and desires *about* beliefs and desires. ... A *second-order* intentional system is more sophisticated; it has beliefs and desires ... about beliefs and desires ... both those of others and its own. ... A *fourth-order* system might *want* you to *think* it *understood* you to be *requesting* that it leave. How high can we human beings go? "In principle," forever, no doubt, but in fact I suspect that you wonder whether I realize how hard it is for you to be sure that you understand whether I mean to be saying that you can recognize that I can believe you to want me to explain that most of us can keep track of only about five or six orders, under the best of circumstances. (Dennett 1983: 345.)

There are some clear cases where up to three occurrences of ‘believes that’ are natural and of importance in explaining behavior. For instance, as Perry has pointed out, each of the participants in the Situation Semantics course at COLING-84 attended because of their beliefs about Barwise and Perry’s beliefs, as well as because of their beliefs about the other students’ beliefs about Barwise and Perry’s beliefs (e.g., that they believed that the theory was worthwhile, hence I ought to believe it, too).

Another natural case is the following: Suppose that I tell Mary (truthfully) that John thinks that she doesn’t like him. In that case, I believe that John believes that Mary believes that he is dislikable. (Actually, I tell Mary that I believe that John believes that Mary believes that he is dislikable; here, there are *four* levels of nesting of propositional attitudes.)

Konolige (1984) cites McCarthy’s use of the “Wise Man Puzzle” as a situation in which one must reason about another person’s beliefs about the beliefs of yet another person. The puzzle concerns three people with white dots painted on their foreheads; they each know that at least one of them has a white dot; and they must deduce the color of their own dots. As Konolige observes,

> The reasoning involved . . . hinges on the ability of the wise men to reason about one another’s beliefs. . . . [The third wise man] must reason about the second wise man’s reasoning about the first wise man’s beliefs. . . . Reasoning about beliefs about beliefs about beliefs . . . can quickly become confusing. (Konolige 1984: 6–7.)

The confusion engendered by such nested beliefs is, no doubt, one of the reasons for their humorous quality; but, given their importance, it is also a reason to have a means of representing them accurately.

Finally, nested propositional attitudes also occur when one considers the structure of an information-seeking dialogue. Here, a user might come to an AI system in the hopes that it will provide information. If the user is not clear about the exact nature of the information needed, the attitude of the system must be to *wonder* what the user *wants to know* (cf. Carberry 1984).

I shall have more to say about nested beliefs, and especially their interaction with quasi-indicators, in §3.

2.3 Pronominal Reference

One of the more difficult issues in natural-language understanding is how to determine the reference of pronouns. There is an important case of pronominal reference that occurs only within belief (or other intentional) contexts (i.e., within the scope of a ‘believes that’ or other intentional-attitude operator). An example is the occurrence of ‘he’ in

(7) John believes that he is rich.

This *quasi-indexical* use of ‘he’ is distinct from other uses (as we shall see in §4). Yet no other AI system that deals with beliefs is sensitive to its unique qualities. One of the main purposes of this essay is to show how an AI system for reasoning about beliefs can, and must be able to, handle quasi-indexical reference.

3 EARLIER WORK

Research on AI systems that are capable of representing and reasoning about beliefs falls into three (not necessarily mutually exclusive) categories:

(I) Research concerned with the *logic* of knowledge and belief or with *intensionality* (the focus of my research).

(II) Research concerned with *common-sense*, or *ordinary*, or *plausible* reasoning about beliefs (i.e., research concerned with *performance limitations*).

(III) Research concerned with *belief revision.*

3.1 Category III: Belief Revision

Belief-revision systems have typically been concerned with the problem of revising a system's data base in the light of new information. This is not simply a matter of adding new information, since the new information might be inconsistent with the old information. (Cf. Doyle 1979.)

If the system is to be capable of reasoning about the beliefs of others, or if several users might contribute information (new beliefs) to the system's data base, the system must be able to keep track of the source of each belief. (Cf. Martins 1983 for an interesting approach using a form of relevance logic.)

Much of this work on belief revision is not immediately relevant to the representational issue we shall be looking at. However, the system under consideration here *is* capable of certain kinds of belief revision, namely *node merging*: the process of "identifying" two nodes (see §7). Moreover, since belief-revision systems must be capable of representing and reasoning about beliefs, they must be sensitive to the issues treated by researchers in category (I), including the present work.

3.2 Category II: Performance Limitations

Research in the area of performance limitations must also take cognizance of the results from category (I), since these are *competence* issues. Thus, representational aspects of the problem of nested beliefs—as well as their interaction with quasi-indicators—are especially relevant here.

3.2.1 Wilks and Bien

Yorick Wilks and Janusz Bien argue "that there can be a very general algorithm for the construction of beliefs about beliefs about beliefs . . . " (Wilks & Bien 1983: 96) but feel

> that a belief manipulating system, which is to be psychologically and computationally plausible, must have built into it some limitations on processing, so as to accord with the fact that deep nestings of beliefs (while well formed in some "competence sense") are in practice incomprehensible. (Wilks & Bien 1983: 97.)

They explicitly distance themselves from the "logic-based approaches" (Wilks & Bien 1983: 97)—i.e., the approach of category (I).

Nevertheless, they are concerned with a number of representational issues of importance to us. For instance, they make it clear that one must distinguish between an agent's beliefs as the *agent* would represent them, and the agent's beliefs as the *system* represents them; they are especially concerned about dealing with this complication in the case of nested beliefs.

They are also concerned with what they call *self-embedding* (Wilks & Bien 1983: 114–117): dealing with the system's beliefs about itself or the system's beliefs about a user's beliefs about himself. Yet there is no indication of how their system would handle quasi-indexical reference. This, as I suggested in §2.3 and as I shall emphasize in §4, is one of the most important problems in self-referential beliefs.

3.2.2 Konolige

Kurt Konolige is concerned, *inter alia*, with reasoning about the beliefs of others. He provides "a ... formal model of belief ... for representing situations in which belief derivation is logically incomplete" (Konolige 1984: 2). In this model, each agent has his own set of inference rules; the system can reason about an agent's beliefs by using the agent's rules. Thus, it can distinguish between the beliefs an agent would have if the agent were "ideal" (i.e., if the agent believed all the logical consequences of his beliefs) from the beliefs that the agent actually has.

Even though Konolige's theory is supposed to deal with realistic limitations on belief systems, he apparently identifies the system's beliefs with facts about the real world (Konolige 1984: 38). But the system ought to be treated the same as any other agent: As I noted in §2.2, the system's beliefs are all implicitly prefixed by an "I believe that ... ", and the objects of its beliefs are not items in the real world. And, as with Wilks and Bien, it is not clear how Konolige's system would deal with quasi-indicators—if it deals with them at all: One of his axioms (M4; Konolige 1984: 56) states that if agent *S* believes *p*, then *S* believes that *S* believes *p*; yet he paraphrases this with a sentence involving quasi-indicators: "if an agent believes *p*, then he believes that he believes it." As I shall point out in §4, these two formulations are not equivalent.

3.2.3 Cohen and Perrault

While Philip R. Cohen and C. Raymond Perrault's work on speech acts (1979) is not directly concerned with performance limitations (their axioms for belief are for an idealized believer; Cohen & Perrault 1979: 480 fn. 5), they are concerned with plans and actions. They recognize the importance of belief over action (Cohen & Perrault 1979: 480), the importance of nested beliefs in such communicative acts as slamming a door (the slammer intends that the observer believes that the slammer intends to insult him; cf. Cohen & Perrault 1979: 478), and the consequent importance of representing the *agent's* model of *another* agent's beliefs (Cohen & Perrault 1979: 480).

But they miss the need for quasi-indicators. Their axiom B.2 (Cohen & Perrault 1979: 480 fn. 5) is:

aBELIEVE(P) ⇒ aBELIEVE(aBELIEVE(P)),

but the third occurrence of 'a' should be a quasi-indicator, not 'a' (cf. my discussion of Maida & Shapiro's (1982) Rule 2 in §6.1, below). Similarly, their definitions of such "operators" as:

$\underline{\text{MOVE(AGT, SOURCE, DESTINATION)}}$
CANDO.PR: LOC(AGT, SOURCE)
WANT.PR: AGT BELIEVE AGT WANT move-instance
EFFECT: LOC(AGT, DESTINATION)

have a WANT.PR that requires that the AGT know his own name: The second occurrence of 'AGT' should be a quasi-indicator.

What is of significance here is the importance of quasi-indicators in *nested* belief contexts.

3.2.4 Clark and Marshall

The work of Herbert H. Clark and Catherine R. Marshall (1981) also points up—by its absence—the importance of quasi-indicators in nested belief contexts. They emphasize the importance of "shared knowledge" (which, incidentally, requires the use of some sort of "co-referentiality" mechanism across belief spaces) and, in particular, of "mutual belief": a sort of infinite nesting of beliefs (Clark & Marshall 1981: 12).

But by not taking quasi-indicators into account, their analysis of the requirements for the use of definite referring expressions is incomplete. For instance, their clause (3) is:

Ann knows that Bob knows that Ann knows that *t* is *R*.

But they do not point out that this also requires the condition that

Ann knows that Bob knows that she herself is Ann,

the 'she herself' here being a quasi-indicator (see §4). A simpler alternative, one not needing this extra belief but still needing a quasi-indicator, is:

Ann knows that Bob knows that she herself knows that *t* is *R*.

3.3 Category I: Intensional-Logic Approaches

3.3.1 Moore

One of the first AI researchers to recognize the importance of an AI system capable of reasoning about knowledge was Robert C. Moore:

> AI systems need to understand what knowledge [they and the systems or people they interact] with have, what knowledge is needed to achieve particular goals, and how that knowledge can be obtained. (Moore 1977: 223.)

He also recognized

> how intimately the concept of knowledge is tied up with action. ...[T]he real importance of such information is usually that it tells us something about what ... [the agent] can do or is likely to do. (Moore 1977: 223.)

Note, however, his talk of *knowledge* rather than of *belief.* "Surely," he says, "one of the most important aspects of a model of another person is a model of what he knows" (Moore 1977: 223). But far more important is a model of what he *believes:* It is people's *beliefs*—which *includes* their knowledge, as well as mistaken beliefs—that enable them to have goals and perform actions.

Moore also recognized the importance of being able to deal with referentially opaque contexts, even though his method for so dealing (use of quotation) leaves much to be desired (cf. §1.2, fn. 1). He recommended the use of Hintikka's (1962) logic of knowledge (without, apparently, implementing it). While this is far superior to, say, (re)inventing his own such logic, buying into someone else's theory carries certain risks, notably the inability—well-known at the time—of Hintikka's system to deal with quasi-indicators (cf. Castañeda 1966: 130 fn. 1; 1967: 11ff; Hintikka 1967). It should be noted, however, that Moore's system does not handle nested knowledge claims, much less knowledge claims about one's own knowledge (though this merely gets him off one hook only to impale him on a sharper one).

3.3.2 McCarthy

John McCarthy (1979) emphasized the importance of using what he called *individual concepts*[5] (a kind of intensional entity) in computational analyses of knowledge. However, his intensional stance was not very thoroughgoing, since he mixed intensional and extensional entities together.

Like Moore, he also emphasized the importance of knowledge for action:

> A computer program that wants to telephone someone must reason about who knows the number. More generally, it must reason about what actions will obtain needed knowledge. (McCarthy 1979: 145.)

—as well as, presumably, about what knowledge will lead to desired actions. But, again like Moore, he concentrated on *knowledge* rather than belief.

It is also curious—and unfortunate—that McCarthy's work shows almost no familiarity with closely related contemporary and prior work in philosophy (e.g., the work of Montague or of deontic logicians). Nor, as is by now to be expected, does he show how to deal with quasi-indicators.

3.3.3 Creary and Barnden

A bit more awareness of philosophical issues is evident in the work of Lewis G. Creary (1979) and John A. Barnden (1983).

Creary points out that AI systems ought to reason about knowledge and belief for the simple reason that humans do. He cites four main problems that a belief system ought to be able to deal with: the philosophical problems of referential opacity, quantifying in (i.e., embedding belief reports in the scope of existential quantifiers), and nested beliefs; and the computational problem of making realistic inferences. His treatment seems to be a thoroughly intensional one: Knowledge and belief are taken to be relations between agents and *propositions*, and propositions

[5]This term, like many others in logical and philosophical studies of intensionality, means different things to different authors; cf. §5.1.3.1. Since most computational researchers have not tried to make their uses of such terms precise, I shall not examine them in detail.

appear to be constituted by relations holding among intensional entities, thus allowing him to quantify into belief contexts (cf. Creary 1979: 177). He also *seems* to recognize that an item referred to within a nested belief context is not necessarily the same as an item referred to outside the context (Creary 1979: 178), but his notation and explanations are far from clear. Similarly, he points to ambiguities in the expression of beliefs, but his discussion is so unclear that it is hard to tell if he has the *de re/de dicto* or some other distinction in mind. One point that *is* clear, from his very first example, is the lack of sensitivity of his system to the problem of quasi-indicators: His analysis of 'Mike wants to meet Jim's wife as such' is:

wants(mike, MeetMike, Wife Jim)

(Creary 1979: 177). The capitalized 'Mike' is a concept of the lower-case 'mike'—but this is not Mike's *self-* concept, as it ought to be (see §4).

Barnden (1983) tries to repair some of the deficiencies in Creary's system and notation, especially with respect to the representation of nested beliefs. But he allows the representation of extensional entities, he assumes (as a simplifying move—but one that is really an *over-* simplification) that the objects of an agent's beliefs are *not* relative to the agent (Barnden 1983: 282), and—of course—he takes no note of quasi-indicators.

3.3.4 Maida and Shapiro

The research to be presented in this essay is a direct outgrowth of the work reported by Anthony S. Maida and Stuart C. Shapiro (1982). In particular, it began with an attempt to make a small correction to their analysis of 'John knows that he is taller than Bill' (Maida & Shapiro 1982: 316)–once again, there was the problem of quasi-indicators. I shall discuss this aspect of their work in §4.

The thrust of Maida & Shapiro 1982 is to argue for a thoroughgoing intensionalism. I shall not rehearse their arguments here. Rather, I shall cite some of the principles that I have adopted from their work:

The data base of an AI system capable of reasoning about the beliefs of itself and others must be such that:

(MS.1) all items represented are intensional,

(MS.2) intensionally distinct items must have distinct representations,

(MS.3) distinct representations correspond to intensionally distinct items

(points (MS.2) and (MS.3) can be referred to jointly as the Uniqueness Principle)

(MS.4) intensionally distinct items that are extensionally the same are linked by a "co-referentiality" mechanism. (I shall have more to say about this in §9.)

Maida (1983) has gone on to apply some of these techniques to reasoning about knowledge, though in a somewhat different style from the approach taken here and concentrating on somewhat different problems.

3.4 Conclusions

It should be clear from the above survey that there have been two chief problems with systems of category (I). With scattered exceptions, there has been little attempt to seriously apply the insights of philosophers on the logic of belief; this will become clearer in what follows, as we look at some of the philosophical issues. And none of the earlier systems has shown how to deal with (much less shown any awareness of) the problem of quasi-indexical reference.

I now turn to the promised discussion of that phenomenon.

4 QUASI-INDICATORS

4.1 The Nature of Quasi-Indexical Reference

Beginning in the 1960s, Hector-Neri Castañeda wrote a series of papers in which he introduced and elaborated a theory of quasi-indexical reference (Castañeda 1966, 1967ab, 1968ab, 1970, 1975c, 1980, 1983; Adams & Castañeda 1983). The theory has been subjected to a great deal of philosophical scrutiny, but it has generally emerged unscathed, and its importance has never been questioned (cf., e.g., Hintikka 1967; Perry 1979, 1983, forthcoming; Lewis 1979; Boër & Lycan 1980; Stalnaker 1981; Brand 1984).

Following Castañeda (1967b: 85), an *indicator* is a personal or demonstrative pronoun or adverb used to make a strictly demonstrative reference, and a *quasi-indicator* is an expression within an intentional context (typically, one within the scope of a verb of propositional attitude,[6] such as 'believes that') that represents a use of an indicator by another person.

Consider the following statement by person *A* addressed to person *B* at time *t* and place *p: A* says, "I am going to kill you here now." Person *C*, who overheard this, calls the police and says, "*A* said to *B* at *p* at *t* that he* was going to kill him* there* then*." The starred words are quasi-indicators representing uses by *A* of the indicators 'I', 'you', 'here', and 'now'.

There are two properties (among many others) of quasi-indicators that must be taken into account when making representation decisions for our AI system:

(QI.1) They occur only *within* intentional contexts.

(QI.2) They cannot be replaced *salva veritate*[7] by any co-referential expressions.

The general question they are intended to help answer is: "How can we attribute indexical references to others?" (Castañeda 1980: 794).

The specific cases that I am concerned with are exemplified in the following scenario. Suppose that John has just been appointed editor of *Byte*, but that John does not yet know this. Further, suppose that, because of the well-publicized salary accompanying the office of *Byte's* editor, which is traditionally immediately deposited in the new editor's account,

(8) John believes that the editor of *Byte* is rich.

And suppose finally that, because of severe losses in the stock market,

(9) John believes that he himself is not rich.

Suppose that the system had information about each of the following: John's appointment as editor, John's (lack of) knowledge of this appointment, and John's belief about the wealth of the editor. We would *not* want the system to infer

(10) John believes that he* is rich

because (9) is consistent with the system's information. The 'he himself' in (9) is a quasi-indicator, for (9) is the sentence that *we* use to express the belief that *John* would express as 'I am not rich'. Someone pointing to John, saying,

(11) He [i.e., that man there] believes that he* is not rich

could just as well have said (9). The first 'He' in (11) is not a quasi-indicator: It occurs outside the believes-that context, and it can be replaced by 'John' or by 'the editor of *Byte*', *salva veritate.* But the 'he*' in (11) and the 'he himself' in (9) could not be thus replaced by 'the editor of *Byte*'—given our scenario—*even though* John is the editor of *Byte.* And if poor John also suffered from amnesia, it could not be replaced by 'John' either.

[6]Another kind of attitude is a *practitional* attitude: In 'John intends Mary to sing', John has the attitude of intending to the "practition" 'Mary to sing', and in 'John intends to sing', he has that attitude to the practition 'he* to sing'. Cf. Castañeda 1975c.

[7]I.e., preserving truth value.

4.2 The Importance of Quasi-Indexical Reference

4.2.1 Inference and Action

Clearly, a deductive question-answering system capable of reasoning about an agent's beliefs must be able to handle quasi-indicators if it is not to draw faulty conclusions.

A number of philosophers, from John Perry (1979) to, most recently, Myles Brand (1984), have emphasized the importance of such beliefs for explaining and producing actions. Suppose that Mary is the only woman in Muir Woods (at some time *t*), suppose that a large tree is about to fall on her, and suppose that she comes to believe this (say, because someone tells her). The propositional "content" of her belief—that is, the internal make-up of the object of her belief—will determine whether or not she takes evasive action. (More precisely, it will be of use in explaining her subsequent behavior, whether evasive or not.) To see why, consider the following two reports of Mary's belief state:

(12) Mary believes that a tree is about to fall on the only woman in Muir Woods.

(13) Mary believes that a tree is about to fall on her*.

Mary is unlikely to take evasive action if she believes what (12) reports her as believing, *unless she also believes that she* is the only woman in Muir Woods.* But those two beliefs taken together would (normally) produce in Mary the belief state reported in (13). Both the belief in case (13) and the additional belief required for action in case (12) contain quasi-indicators in their third-person reports ('her*' in the former, 'she*' in the latter).

4.2.2 Consequences of Ignoring Quasi-Indicators

Theories that do not take quasi-indexical reference into account do so at the expense of being unable to represent an important category of beliefs, namely, beliefs about oneself.[8] There are two theories in AI that face this inadequacy.

4.2.2.1 Konolige

The first, as I have already mentioned, is Konolige's. He says:

> we might argue that, if an agent *S* believes a proposition *P*, then he believes that he[*] believes it. All he has to do to establish this is query his[*] belief subsystem with the question, "Do I believe *P*?" If the answer comes back "yes," then he should be able to infer that he[*] does indeed believe *P*, *i.e.* $[S][S]P$ is true if $[S]P$ is. (Konolige 1984: 38. '$[S]P$' is Konolige's notation for '*S* believes *P*'. I have indicated quasi-indicators by '[*]'.)

But $[S]P$ should not imply $[S][S]P$, since *S* might believe *P* and believe that *he** believes *P*, yet *not* believe that *S* believes *P*, since *S* might not believe that he* is *S*. (It is also important that, when *S* queries a belief subsystem, he query *his own* subsystem, and not merely the subsystem of someone named '*S*'.)

4.2.2.2 Schank

Roger Schank's Conceptual Dependency theory also fails to take note of the importance of quasi-indicators. The primitive actions MTRANS and MBUILD are the ones that need quasi-indicators as slot fillers. For instance, one CD analysis of

(14) John promised to give Mary a book.

is

(14CD)

actor	:	John		
action	:	MTRANS		
object	:	actor	:	John
		action	:	ATRANS
		object	:	book
		to	:	Mary
		from	:	John
from	:	John		

[8] Such beliefs also play a role in purely philosophical speculation, such as in discussions of the Cartesian *cogito*; cf. Rapaport 1976a: 63, 67n1.

(cf. Schank & Riesbeck 1981: 19–20.) But this is not fine-grained enough. A similar analysis of

John promised that Lucy would give Mary a book

would be something like:

actor	:	John		
action	:	MTRANS		
object	:	actor	:	Lucy
		action	:	ATRANS
		object	:	book
		to	:	Mary
		from	:	Lucy
from	:	John		

But without a quasi-indicator as the filler of the object:actor slot, (14CD) is the analysis, not of (14), but of

(15) John promised that John would give Mary a book.

And, as our study of quasi-indicators in §4.1 showed, (14) and (15) are not equivalent. (A similar argument against Schank is given in Brand 1984: 209–212.)

It is only fair to note that these criticisms are by no means fatal to Konolige or to Schank. What is important is that any theory dealing with beliefs that aspires to representational adequacy must have a way of handling quasi-indicators.

In the next section, I show how this can be done.

5 THE REPRESENTATION OF BELIEF

In this, the central section of the essay, we shall look briefly: at philosophical theories of, and computational needs for, intensionality; at various distinctions between *de re* and *de dicto* beliefs; at SNePS as a "knowledge"-representation language; and, finally, at an adequate representation, in SNePS, of *de re* and *de dicto* beliefs, nested beliefs, and quasi-indicators.

5.1 Intensionality

There has been much controversy in philosophy over the meaning of 'intension'. Although I shall not attempt to define it here, intensional approaches to problems in logic and language may be roughly characterized as ones that place more of an emphasis on "meanings" (or, perhaps, mental states and events—things in the mind) than on truth-values or denotations (on non-mental entities—things in the external world). (Cf. Brody 1967: 64, 67.)

5.1.1 Intensional Contexts

Included among *intensional contexts* are such modalities as 'it is necessary that ...', '*A* believes that ... ', '*A* is looking for ... , etc.

Among the intensional contexts are the *intentional* ones, those, such as belief contexts, that involve intentional—or psychological—verbs.[9] Typically, the objects of (psychological acts expressed by) intentional verbs need not exist or be true. Thus, one can think about unicorns or believe false propositions.

Among the intentional verbs are those expressing *propositional attitudes*: 'believe', 'know', etc. These express attitudes that a cognitive agent might take towards a proposition. 'Look for', 'think about', etc., are attitudes whose linguistic expression form intentional contexts, but they are not *propositional* attitudes.

Another kind of intentional context is provided by what can be called *practitional* operators (cf. Castañeda 1975c, and §4.1, fn. 1, above): 'it is wrong (to) ...', 'John intends (to) ...', etc. For instance, if John has promised his wife not to speak to Lucy, has also promised his best friend to speak to his (the best friend's) fiancee, and—unknown to John—Lucy *is* his best friend's fiancee, then while John ought not to speak to Lucy, he ought to speak to his best friend's fiancee.

5.1.2 Referential Opacity and Quantifying In

Two problems that must be faced by anyone dealing with intensional language are those of referential opacity and "quantifying in".

A linguistic context is *referentially opaque* if substitution of co-referential constituents does not preserve truth value, and is *referentially transparent* otherwise.[10]

One must also take care when *quantifying into* intensional contexts. This can also be viewed as a substitutional issue: the replacement of one of the constituents by a bound variable. One's allegiance to intensional vs. extensional approaches to logic, language, and ontology can often be determined by one's attitude towards one aspect of this problem: Suppose Lucy, after having read a fantasy story, is looking for a unicorn. Is there a unicorn that she is looking for? If you are inclined to answer 'No' *on the grounds that unicorns do not exist*, then you might look askance at intensional contexts, because they cannot be thus *existentially* quantified into. But if you are inclined to answer 'Yes' (or perhaps to hesitate), on the grounds that she is (or might be) looking for the (specific) unicorn she read about, then you are willing to admit *intensional entities* into your ontology.

5.1.3 Intensional Theories in Philosophy

At the risk of oversimplifying, we might say that most contemporary theories of intensional entities can be traced back, through Church (1951, 1973–1974) and Carnap (1956), to Frege's 1892 essay, "On Sense and Reference", or else to Meinong's 1904 essay, "The Theory of Objects".

[9]This is, roughly, Brentano's sense of 'intentional', in which "intentionality" is a distinguishing mark of mental phenomena (cf. Brentano 1874, Chisholm 1967, and §5.1.3.2, below). By 'intentional' verbs, I do *not* mean such verbs as 'kick', as in 'John hated Lucy, so he (intentionally) kicked her'—i.e., he kicked her *on purpose*.

[10]This can, perhaps, be made more precise: A context is referentially opaque if its extension (its truth-value in the case of a sentence, its referent in the case of an NP, etc.) is not a function of the extensions of its parts. However, the rough characterization above should suffice.

5.1.3.1 The Fregean approach Frege distinguished between the *Sinn* (sense, meaning) and the *Bedeutung* (denotation, reference, referent, meaning) of words, phrases, and sentences. Each word or phrase *expresses* a sense, and each sense *determines* a referent. For instance, the sense expressed by 'the Morning Star' is, roughly, "the last starlike object visible in the morning sky"; this, in turn, determines a referent, namely, a certain astrophysical object, which is also called 'Venus'.[11] Frege introduced this distinction chiefly in order to eliminate it from his logical foundations for mathematics, which, he claimed, could be handled *without* recourse to senses. For mathematics, all that counted was the referents of sentences[12] and their constituents. The referent of a sentence is its *truth value*, the referent of an NP is an *object*, and (arguably) the referent of a predicate is a *concept*.[13] The sense of a sentence is a *thought* (or *proposition*). However, since Frege wanted the referent of a sentence to be a function of the referents of its constituents, he had to complicate matters when it came to intensional sentences:[14] The *referent* of a sentence that occurs within an intensional context is its ordinary *sense.*

Various philosophers have attempted to formalize these notions, e.g., Church *(op. cit.)*, Carnap *(op. cit.)*, and Montague (1974). I shall not go into these theories here, except to note that senses (and their more formal theoretical counterparts) are *intensional* entities. All words and phrases have senses, even 'unicorn', but not all have referents.

5.1.3.2 The Meinongian approach Alexius Meinong, a student of Brentano[15] took a more psychological approach. He analyzed psychological experiences into three components: a psychological *act* (e.g., such acts as believing, desiring, or thinking); an *object* of the act (e.g., that which is believed, or desired, or thought about); and a *content* of the act, which "directs" it to its object. (For details, see Meinong 1904; Findlay 1963; Rapaport 1976b, 1978, 1979.) All psychological acts are directed to objects; this is the Thesis of Intentionality, first adumbrated by Brentano (1874) as a distinguishing mark of mental, as opposed to physical, phenomena. But all objects do not exist, or, as Meinong humorously put it: "There are objects of which it is true that there are not such objects" (Meinong 1904: 490). That is, I can think about unicorns *in exactly the same way* that I can think about cats, and I can believe that $1 + 1 = 3$, even though the object of my belief is false (or does not exist, as Meinong would have put it).[16] The object of a propositional attitude is called an *objective*; the object of a thought is called an *objectum.* Meinongian objectives and objecta are intensional entities.

Contemporary theories that are Meinongian in spirit include those of Castañeda (1972, 1975abc, 1977, 1979; cf. Rapaport 1976b, 1978), Terence Parsons (1980; cf. Rapaport 1976b, 1978, 1985), Richard Routley (1979; cf. Rapaport 1984), and myself (Rapaport 1976b, 1978, 1979, 1981, 1982).

All of these theories can handle referential opacity and quantifying in. Typically, substitution of co-referentials (more generally: of identicals) in intentional contexts is blocked on the grounds that the items to be substituted are intensionally *distinct.* And most cases of quantifying in are allowed, since quantifiers are presumed to range over *intensional* entities. (The sort of case that is not so easily handled is the one where, just because Lucy is looking for a unicorn, it does not follow that there is (even in the non-existentially-loaded sense of 'there is') a *particular* unicorn that she is looking for—any one will do. But this problem is beyond our present scope. Cf. Rapaport 1976b for more detail.)

5.1.4 The Need for Intensional Entities in "Knowledge" Representation

As has been forcefully argued by Woods (1975), Brachman (1977), Shapiro (1981), and Maida & Shapiro (1982), an AI system that is going to interact with a human, or that is intended to be a model (or simulation) of a human mind, must be able to represent and reason about intensional entities. And, as Maida and Shapiro have stressed, they *only* need to deal with intensional entities. I would also claim, in the Kantian spirit (cf. §2.2), that they *cannot* deal with *extensional* entities. What is lacking from such schemes is a full-blown theory of intensional entities. Any of the ones mentioned here would no doubt suffice, though I favor the Meinongian approach, because of its psychological underpinnings. (Castañeda's theory is especially appropriate for SNePS.)

[11]Note that the term 'Venus' expresses a *different* sense, but that sense determines the *same* referent as the one determined by the sense of 'the Morning Star'.

[12]Strictly: the referent determined by the sense expressed by the sentence!

[13]Cf. Frege 1891: 30–31; 1982a: 43, esp. fn.; Dummett 1967: 231. It is important to note that a Fregean "concept" is a technical notion. Cf. §3.3.2, n. 1.

[14]Which, he claimed, do not occur in mathematics, and so these problems could be avoided there.

[15]As was Edmund Husserl, the founder of phenomenology.

[16]Actually, he would have said that it "lacked being", in particular, it did not "subsist". For details and more motivation, see Rapaport 1976b, 1978.

5.2 De Re and De Dicto

In the philosophical literature, a belief *de dicto* is treated as a psychological act of belief whose object is a proposition—a *"dictum"*—and a belief *de re* is treated as a psychological act of belief whose object is (to use the Meinongian term) an objectum.

For example, suppose that Ralph sees the person whom he knows to be the janitor stealing some government documents, and suppose—unknown to Ralph—that the janitor has just won the lottery. Then Ralph believes *de dicto* that the janitor is a spy, and he believes *de re* that the lottery winner is a spy. That is, if asked, Ralph would assent to the proposition 'The janitor is a spy'; but he merely believes *of the man* who *we* know to be the lottery winner that he is a spy—Ralph would not assent to 'The lottery winner is a spy'.

Much of the philosophical literature on *de re* and *de dicto* belief concerns the relations between these, conceived as *ways* of believing. I do not believe that there are two such distinct modes of belief. But be that as it may, it seems clear to me that there *are* two distinct ways of *reporting* an agent's beliefs, which may, for better or worse, be called *de re* and *de dicto*.[17]

Thus, if what we are interested in is expressing or communicating the actual "content" of Ralph's belief, we would need to report it in a *de dicto* fashion. If, however, we are *not* concerned to, or *cannot*, communicate his belief in that manner, then we can (or must) report it in a *de re* fashion.

Traditionally viewed, a belief *de dicto* is a referentially opaque context, whereas a belief *de re* is referentially transparent. Thus, the inference

(16)

Ralph believes [*de dicto*] that the janitor is a spy.
The janitor = the lottery winner.
____
Ralph believes [*de dicto*] that the lottery winner is a spy.

is invalid. Moreover, its conclusion not only presents *false* information, it represents a *loss* of information, namely, of the information about the propositional "content" of Ralph's belief. On the other hand,

(17)

Ralph believes [*de re*] of the janitor that he is a spy.
The janitor = the lottery winner.
____
Ralph believes [*de re*] of the lottery winner that he is a spy.

is valid. But the conclusion conveys just as little information about Ralph's *actual* belief *de dicto* as does the first premise.

Castañeda (1970: 167ff) prefers to distinguish between *propositionally transparent* and *propositionally opaque* constructions: The former display the internal make-up of the proposition; the latter don't. What I call a *de dicto belief report* is referentially opaque but propositionally transparent, while what I call a *de re belief report* is referentially transparent but propositionally opaque. It is the propositional kind of opacity and transparency that is important for communication and for representational issues in AI.

Ordinary language, however, does not distinguish these. That is, without an explicit device (e.g., as in (16) and (17) above), belief sentences can be interpreted as *either de re or de dicto.* This will pose a pragmatic problem for the ultimate project. At this stage of investigation, however, I shall adopt the following conventions from Castañeda 1970: 174ff: Where *A* names or describes an agent and *p* a proposition, and *x* ranges over NPs,

(C1) Any sentence of the form

A believes that *p*

will be the canonical representation of a *de dicto* belief report.

(C2) Any sentence of the form

A believes of *x* that *p*

will be the canonical representation of a *de re* belief report, where *x* names or describes the *res*.

[17] Parsons 1980: 46 fn. 10 can be read as taking this interpretation.

An alternative to (C2) is

(C3) *x* is believed by *A* to be F

where F names or describes the property predicated of *x*. While (C3) has the advantage that the *res* is outside of the belief context, (C2) has the computational advantage that its grammatical structure is a generalization of (C1)'s structure, as well as the philosophical advantage that in both (C1) and (C2), the objective of the belief can be treated as being propositionally transparent. (Cf. Castañeda 1970: 174ff.)

Finally, a *de se* belief may be taken to be a belief about oneself (cf. Lewis 1979: 521). Such a belief can be reported as being *de re or de dicto*. Consider the following:

(DD.DS) John believes that he* is rich.

(DR.DS) John believes of himself that he is rich.

The first will be taken here by convention as a *de dicto* report of John's *de se* belief; all such reports involve the use of quasi-indicators. The second will be taken here by convention as a *de re* report of John's *de se* belief.[18] Both (DD.DS) and (DR.DS) are mutually consistent: for (DR.DS) might be true because John might believe that the editor of *Byte* is rich, yet not believe that he* is the editor of *Byte*. We, who know that he *is* the editor, can report his belief about the editor by the *de re/de se* sentence (DR.DS). On the other hand, (DD.DS) is *in*consistent with John's believing that he* is not rich.

5.3 The Semantic Network Processing System (SNePS)

SNePS (Shapiro 1979) is a facility for building semantic networks and retrieving and deducing information from them. A SNePS network consists of nodes linked by labeled, directed arcs. The nodes and arcs have the following features, among many others:

(S.1) Each constant node represents a unique concept.

(S.2) Each concept represented in the network is represented by a unique node.

(S.3) Arcs represent non-conceptual, binary relations.

(S.4) Deduction rules are propositions, and so are represented by nodes.

(S.5) Non-standard, set-oriented logical connectives are used.

(S.6) Universal and existential quantifiers are available.

In the context of the sort of system considered here, *non-dominated* nodes—i.e., nodes with no arcs pointing to them—represent *beliefs of the system*. *All* nodes, whether dominated or not, are in the "mind" of the system. To use Meinongian terminology, all nodes represent Meinongian objects of the *system's* psychological acts: Dominated nodes represent objecta; non-dominated nodes represent objectives.[19]

As a simple example, the sentence

(18) John is rich.

could be represented in SNePS by the network of Figure 1.

The WHICH-ADJ case frame is used to represent simple subject-predicate propositions. So, the non-dominated node m5 represents the system's belief that something (viz., whatever is represented by node m3) has the property represented by m4.

The LEX arc points from a node to a tag used by the ATN parser-generator to attach a word to the node. (The node labeled 'rich', however, could also be considered to be the system's concept of the *word* 'rich'. Cf. Maida & Shapiro 1982: 303 for details on the LEX arc.)

[18] Because of the use of 'himself', it might be hard to "hear" (DR.DS) as *de re*. It is to be understood as 'John believes of the person who *we* know to be he that he (that person) is rich'. But this is too long to be a canonical representation.

[19] The complete set of all possible nodes—whether actually in the network or not—corresponds neatly to Meinong's notion of *Aussersein*. Cf. Rapaport 1976b, 1978.

The NAME-NAMED case frame is used to identify objecta by name. So, the non-dominated node m2 represents the system's belief that something (viz., whatever is represented by node m3) is named by the name whose concept is m1.

More idiomatically, (18) has been analyzed as

(18A) Something is named 'John' and it is rich.

or, perhaps,

(18B) $\exists x$[Name(x, 'John') & Rich(x)],

where the quantifier ranges over objecta in the system's belief space.

There is nothing sacrosanct about this analysis. We could just as well have represented (18) by the network of Figure 2. Here, m7 represents the system's belief that the entity represented by m3 is a person, and m9 represents the system's belief that the entity represented by m3 is male.

Clearly, the amount of detail one wishes to build into the network analysis depends on one's ontological and linguistic theories. Rather than worry about such details here, I shall use the analysis of Figure 1, since it contains the essential information for my purposes.

To represent the system's beliefs about the beliefs of others, an AGENT-VERB-OBJECT case frame is used. Thus

(19) John believes that p

will be represented by the network fragment of Figure 3. Here, m6 represents the system's belief that the person whom the system believes to be named 'John' believes m5, where m5—in a concrete case—will be a node representing, *roughly*, the system's *de dicto* report of John's belief.

I say 'roughly', because, in fact, the system can never represent John's belief *exactly*; it can only represent John's belief using its *own* concepts. I shall return to this point shortly (cf. §2.1.4, above, and §§5.4.2,5.4.3, below), but it should be noted that humans have precisely the same limitations.

5.4 SNePS Representations of Beliefs

I can now be a bit more precise. However, to make things easier for the reader, I shall not present a general scheme for representation, but only representations of actual—though representative—sentences.

5.4.1 De Re and De Dicto Beliefs

The *de dicto* belief report

(20) John believes that Lucy is sweet

will be represented by the network of Figure 4. A predicate-logic analysis of (20) might be:

(20A) $\exists x$[Name(x, 'John') & Believes(x, $\exists y$[Name(y, 'Lucy') & Sweet(y)]].

That is, the system believes three things: that someone is named 'John', that he believes that someone is named 'Lucy', and that he believes that she (i.e., the person he believes to be named 'Lucy') is sweet.

To simplify the graphical notation, which will quickly become complex, Figure 4 can also be drawn as in Figure 5. The idea here is that, since m8 and m11 are both nodes in an AGENT-VERB-OBJECT case frame with the same AGENT and the same VERB, we can eliminate the redundant arcs from the *graphical representation* of the network by using the box notation. *This is a notational convenience only.*

A *de re* belief report,

(21)] John believes of Lucy that she is sweet,

will be represented by the network of Figure 6. A predicate-logic analysis of (21) might be:

$\exists x \exists y$[Name(x, 'John') & Name(y, 'Lucy') & Believes(x, Sweet(y))].

That is, the system believes three things: that someone is named 'John', that someone is named 'Lucy', and that he (John) believes of her (Lucy) that she is sweet. Note that here the system has no beliefs about how John represents Lucy.[20]

We can also combine these. But here we must be careful. In the absence of prior knowledge of co-referentiality, the entities *within* a belief context should be represented *separately* from entities *outside* the context that might be co-referential with them.

Suppose that the system's beliefs include that a person named 'Lucy' is young and that John believes that a (possibly different) person named 'Lucy' is rich. This is represented by the network of Figure 7. The section of network dominated by nodes m10 and m14 is the system's *de dicto* representation of John's belief. That is, m14 is the *system's* representation *of* a belief that *John* would express by 'Lucy is rich', and it is represented *as* one of John's beliefs. Such nodes are considered as being in the system's representation of John's "belief space". If it is later determined that the "two" Lucies are the same, then a node of co-referentiality would be added, as in Figure 8 (node m16; cf. Maida & Shapiro 1982: 303–304 and §9.2.1, below, for a discussion of the EQUIV case frame).

5.4.2 Nested Beliefs

The representational scheme presented here can also handle sentences involving iterated belief contexts. Consider

(22) Bill believes that Stu believes that Hector is philosophical.

The interpretation of this that I am most interested in representing treats (22) as the system's *de dicto* representation of Bill's *de dicto* representation of Stu's *de dicto* belief that Hector is philosophical. On this interpretation, we need to represent the *system's* Bill, the system's representation of *Bill's* Stu, and the system's representation of Bill's representation of *Stu's* Hector. In the implementation, these are represented, respectively, by: (Bill system), (Stu Bill system), and (Hector Stu Bill system). Sentence (22) is represented by the network of Figure 9.

In the implementation (cf. §8 and the Appendix), such a network is built recursively as follows: The parser maintains a stack of "believers"; the first element on the stack is the system. Each time a belief sentence is parsed, it is made the object of a belief of the previous believer in the stack. Structures are shared wherever possible. This is accomplished by use of a new SNePS User Language function, *forb-in-context* (find-or-build in a belief context), which, given a description of a node and a belief context (a stack of believers), either finds the node in that context if it exists there (i.e., if it is already in the believer's belief space, or else builds it in that context (i.e., represents the fact that it is now in the believer's belief space). Thus,

(23) Bill believes that Stu believes that Hector is Guatemalan

would modify the network of Figure 9 by adding new beliefs to (Bill system)'s belief space and to (Stu Bill system)'s belief space, but would use the same nodes to represent Bill, Stu, and Hector. (See the sample run in §8 and the definition of *forb-in-context* in the Appendix.)

5.4.3 Some Comments

Before turning to the representation of quasi-indicators, it is worth discussing two representational choices that could have been made differently.

First, in Figures 4–6, I am assuming that the system's concept of sweetness (Figs. 4–5, node m9; Fig. 6, node m8) is also the system's concept of (Lucy system)'s concept of sweetness. This assumption seems warranted, since *all* nodes are in the system's belief space. If the system had reason to believe that *its* concept of sweetness differed from Lucy's, this could—and would have to—be represented.

Second, there are, *prima facie*, at least two other possible case frames to represent situations where an agent has several beliefs (of which the *de dicto* case is merely one instance). For example, instead of the network of Figure 4 or 5, we could have used either of the networks of Figures 10 or 11.

But the case frame for node m10 in Figure 10 makes no sense linguistically: Syntactically, a belief sentence is an SVO sentence (or, it has an agent-verb-object case frame); it is not an SVOO, SVOOO, etc., sentence. Having a single OBJECT arc to a "supernode", in the manner of Hendrix 1979: 64, might be better, but still odd linguistically.

[20]These analyses, although arrived at independently, bear a structural similarity to analyses by Castañeda (1970: 176ff) and by Chisholm (1976: 13).

In addition, as new beliefs are added, the case frame would have to change. A related disadvantage to this alternative (as Shapiro has pointed out in discussion) is that m10 is a *single* proposition with two objects (i.e., there are two Meinongian objectives of the act of believing), rather than two propositions each with a single object (i.e., two acts of believing, each with a single objective). Thus, a question like "Who believes *p*?" or an addition to the data base that *one* of the reports about John's beliefs is false would apply to *all* of the beliefs, not just to one of them.

In Figure 11, node m10 is the SNePS representation of the conjunction of m6 and m9. One objection to this format is that the nodes in *John's* belief space are explicitly conjoined, unlike the nodes in the *system's* belief space; there seems no good reason for this structural dissimilarity. Moreover, retrieval of the information that John believes that someone is sweet would be difficult, since that information is not explicitly represented in Figure 11.

5.4.4 De Se Beliefs

5.4.4.1 The representation To adequately represent *de dicto* reports of *de se* beliefs, we need the strategy of separating entities in different belief spaces (see §5.4.1).

Consider the possible representation of

(3) John believes that he* is rich

shown in Figure 12 (adapted from Maida & Shapiro 1982: 316).

This suffers from three major problems. First, it is ambiguous: It could be the representation of (3) or of

(24) John believes that John is rich.

But, as we have seen, (3) and (24) express quite different propositions; thus, they should be separate items in the data base.

Second, Figure 12 cannot represent (24). For then we would have no easy or uniform way to represent (3) in the case where John does not know that he is named 'John': Figure 12 says that the person (m3) who is named 'John' and who believes m6, believes that that person is rich; and this would be false in the amnesia case.

Third, Figure 12 cannot represent (3) either, for it does not adequately represent the quasi-indexical nature of the 'he' in (3): Node m3 represents both 'John' and 'he', hence is both inside and outside the intentional context, contrary to both of the properties of quasi-indicators discussed in §4.1.

Finally, because of these representational inadequacies, the system would invalidly "infer" (iii) from (i)–(ii):

(i) John believes that he is rich.
(ii) He = John.
(iii) John believes that John is rich.

simply because premise (i) would be represented by the same network as conclusion (iii). (I shall return to this in §6.1; see the sample run in §8.)

Rather, the general pattern for representing such sentences is illustrated in Figure 13. The role that 'he*' plays in the English sentence is represented by node m5; its quasi-indexical nature is represented by means of node m7.

That nodes m3 and m5 must be distinct follows from the separation principle. But, since m5 is the system's representation of John's representation of himself, it must be within the system's representation of John's belief space; this is accomplished via nodes m7 and m6, representing John's belief that m5 is his "self-representation". Node m6, with its EGO arc to m5, represents, roughly, the proposition 'm5 is me'. (Cf. §5.4.4.2.)

This representation of quasi-indexical *de se* sentences is thus a special case of the general schema for *de dicto* representations of belief sentences. When a *de se* sentence is interpreted *de re*, it does not contain quasi-indicators, and can be handled by the general schema for *de re* representations. Thus,

(25) John is believed by himself to be rich

would be represented by the network of Figure 12.

As a final example, consider the following:

(26) John believes that he* is not rich.

(8) John believes that the editor of *Byte* is rich.

(27) John is the editor of *Byte*.

(28) John believes of himself that he is rich.

If John is unaware of the truth of (27), then the system ought to be able to infer—and, hence, to represent—(28). We can represent the data base resulting from the input of this sequence of information and inference by the network of Figure 14. Here, nodes m11 and m7 represent (26) (node m10 represents a SNePS negation of node m9); nodes m15 and m17 represent (8); m18 represents (27); and m20 represents (28).

5.4.4.2 The EGO arc

One representational issue requires discussion: the EGO arc. The node pointed to by the EGO arc is the system's representation of what might be called John's *self concept*, i.e., John's "model" of himself (cf. Minsky 1968).

Again, let us consider alternative representations. The network of Figure 15, which is the simplest alternative, uses a COGNITIVE-AGENT/SELF-CONCEPT case frame to "identify" m5 as being m3's (John's) self concept (a kind of "merger" of m5 with m3). But m5 needs to be *inside John's* belief space (to be on a par with Figure 13), which this network does not do. Note that in Figure 13, node m7 also links John (m3) with his self concept (m5),[21] as does Figure 15's m9, but it does so while placing m5 within John's belief space.

Another alternative is the network of Figure 16. Here, we do have m5 within John's belief space, but we also have John there—*as well as* outside it—which violates the principle of the separation of belief spaces. And, again, any viable role played by such a case frame can also be played by node m7 of Figure 13.

Finally, consider the network of Figure 17. Here, the idea is to have a case frame analogous to the NAME-NAMED one, which might be called the QI-EGO case frame. Node m7 represents the proposition (believed by John) that m5 is he himself. But, of course, that is *not* the proposition believed by John. Rather, he believes a proposition that *he* would express as 'm5 is me'. And that is precisely what the original EGO arc of Figure 13 is intended to capture.

At least two philosophers have suggested analyses of *de dicto/de se* beliefs that are structurally similar to the lone EGO arc. Chisholm would say that such a belief report conveys the information that John has an "individual essence" m5 and that he believes that whatever has m5 is rich (cf. Chisholm 1977: 169). And Perry introduces an *ego function* that maps a person to that person's "special sense"; a Perry-style analysis of our *de dicto/de se* belief would be:

$$\exists s[s = \text{ego(John) \& Believes(John, Rich(}s\text{))}]$$

(cf. Perry 1983: 19, 25).

There are difficulties with both of these suggestions, which I shall not go into here (cf., e.g., Castañeda 1983: 326f), and there are also more complicated cases that need to be examined. But these are topics for future investigation (see §9.3.2).

[21] By means of the path from (i.e., the relative product of) the converse-EGO arc from Fig. 11's m5 to m6, through the converse-OBJECT arc from m6 to m7, to the AGENT arc to m3.

6 INFERENCES

6.1 Inferences Using the Maida and Shapiro Network

Recall the problem with the Maida & Shapiro network for ‘John believes that he* is rich’ (§5.4.4.1). It was ambiguous in the sense that a parser taking English sentences as input and producing (SNePS) networks as output would produce the network of Figure 12 as the parse of *both* of the following sentences:

(3) John believes that he* is rich.

(24) John believes that John is rich.

But, as we have seen, each should be parsed differently: For if John does not believe that he* is John, then it might be the case that (24) is true while it is *not* the case that (3) is true, or vice versa.

Thus, the system would “infer” (24) from (3) together with

(25) He is John.

But the “inference” would be “*static*” rather than “*dynamic*”; i.e., rather than having to use an inference rule to *build a new piece of network* corresponding to (24), the system would merely *retrieve* (3) (“find” it, in SNePS terminology), since there would only *be one* net for (3) and (24). (See the sample run in §8.)

This, it is to be emphasized, is a problem with Maida and Shapiro’s *representation*, not their rules. Indeed, the point is precisely that no rules are involved here.

Nevertheless, given the insensitivity of their networks to quasi-indexical reference, the actual rules they present are more powerful than they should be. The observations that follow hold for any representations that do not recognize the need for quasi-indicators, including those of Cohen & Perrault 1979 and Clark & Marshall 1981 (cf. §3, above).

For instance, consider Maida and Shapiro’s Rule 2 and one of its proffered instances (paraphrased from Maida & Shapiro 1982: 330; my emphases, and my use of ‘he*’):

(Rule 2) $(\forall x, y, z, \mathrm{R})[\mathrm{Believes}(x, \mathrm{R}xy)\ \&\ \mathrm{Believes}(x, \mathrm{Equiv}(y, z)) \rightarrow \mathrm{Believes}(x, \mathrm{R}xz)]$

(P) If John believes that Jim’s wife is Sally’s mother and *he* (John) believes that *he** wants to meet Jim’s wife, then *he* (John) believes that *he** wants to meet Sally’s mother (and this is so regardless of whether Jim’s wife *in fact is* Sally’s mother).

The problem is that (P) is *not* an instance of Rule 2, because the ‘*x*’ in ‘R*xy*’ cannot be replaced by a quasi-indicator whose antecedent is the ‘*x*’ in the first argument place of ‘Believes’; it can only be replaced by the *same* value that ‘*x*’ gets. Thus, the following *would* be a legitimate instance of Rule 2:

(P1) If John believes that Jim’s wife is Sally’s mother and he (John) believes that *John* wants to meet Jim’s wife, then he (John) believes that *John* wants to meet Sally’s mother.

There can be no legitimate use of Rule 2 to sanction (P). For if there were, it would also sanction the following:

(P2) If John believes that Jim’s wife is Sally’s mother and he (John) believes that *he** wants to meet Jim’s wife, then he (John) believes that *John* wants to meet Sally’s mother.

(P3) If *he** believes that Jim’s wife is Sally’s mother and *he** believes that *he** wants to meet Jim’s wife, then *he** believes that *he** wants to meet Sally’s mother.

But these, as we have seen, should not be sanctioned.

6.2 The Maida & Shapiro Approach to Inference

However, Maida and Shapiro take an *approach* to rules of inference for sentences involving propositional attitudes that is quite appropriate to the position (discussed in §5.2) that emphasizes the priority of communication:

> A system that conforms to the Uniqueness Principle [cf. §3.3.4, above] does not need the substitutivity of equals for equals as a basic reasoning rule, because no two distinct nodes are equal. Co-referentiality between two nodes must be asserted by a proposition. It requires inference rules to propagate assertions from one node to another node which is co-referential with it. Thus, *intensional representation implies that referential opacity is the norm* and transparency must be explicitly sanctioned by an inference process . . . . (Maida & Shapiro 1982: 300; my emphasis.)

Referential opacity is the norm. That is, *propositional transparency is the norm.* Inferences such as those just discussed *should* be blocked, because, as noted, they represent a *loss* of information.

6.3 Adequate Inference Rules

What inference rules do we want? For one thing, we want some rule to sanction (P). The following will do this:[22]

(R1) $(\forall x, y, z, w, \mathrm{R})$[Believes($x$, R$wy$) & Believes($x$, Equiv($y, z$)) $\rightarrow$ Believes(x, Rwz)].

The difference between Rule 2 and (R1) is the extra variable, w, which is needed in order to make (R1) a rule of inference that *preserves propositional transparency.* Both (P) and (P1) are instances of (R1), but (P2) and (P3) are not. ((P3) is not, because the quasi-indicator 'he*' cannot be a value of 'x'.)

Similarly, if we want the following inference to be valid:[23]

(29) (i) John believes that the editor of *Byte* is rich.
(ii) John believes that he* is the editor of *Byte.*
(iii) John believes that he* is rich.

as well as the inference with (29iii) and (29i) switched, we would need the following propositionally transparent rule:

(R2) $(\forall x, y, z, \mathrm{F})$[Believes($x$, F$y$) & Believes($x, z = y$) $\rightarrow$ Believes(x, Fz)].

To sanction inferences involving referentially transparent transitive predicates, further rules are needed. Consider the following valid inferences:

(30)
(i) John is taller than Bill.
(ii) Bill is the editor of *Byte*.
(iii) John is taller than the editor of *Byte*.

(31)
(i) The editor of *Byte* is rich.
(ii) John is the editor of *Byte.*
(iii) John is rich.

(32)
(i) John wrote a memo to the editor of *Byte.*
(ii) John is the editor of *Byte.*
(iii) John wrote a memo to (a) John.
(b) himself.

[22] Note that what is ultimately needed is a rule *schema*, since, in the general case, R will vary in arity.

[23] James Moor (personal correspondence) is skeptical of this desire: For it is perfectly possible for John to have the beliefs expressed by (29i) and (29ii), but fail to draw the proper inference. Clearly, I am using an idealized notion of belief here. The issue Moor points out is related to the sorts of issues discussed by Konolige, and is at the performance level. Cf. §3.3.

(Note that 'wrote a memo to' might be considered to have a referentially opaque reading; here, it is being used referentially transparently.)

To sanction (30), for instance, we need an additional premise to the effect that *being taller than* is referentially transparent:

RefTransp(tall)

(this could be part of the lexical entry for 'tall'), and we need a rule for EQUIV for relations:

(R3) $(\forall x, y, z,$R)[RefTransp(R) & Rxy & Equiv$(y,z) \rightarrow$ Rxz].

The other inferences would be handled similarly.

It should be noted that in SNePS, (R1)–(R3) would not be second-order rules, even though the predicate-logic formulations I have given here quantify over predicates. A more adequate notation for the network of Figure 1, for example, might be

$\exists x[\{$Named(x), Name(Lex(John))$\}$ & $\{$Which(x), Adj(Lex(rich))$\}]$,

where '$\{\}$' is, roughly, an operator mapping a set of arc-node pairs into network fragments. Or, treating arc labels as (the only) binary predicates, Figure 1 might be represented thus:[24]

$(\exists v, w, x, y, z)$[Named(y,x) & Name(y,z) & Lex(z, 'John') & Which(w,x) & Adj(w,v) & Lex(v,rich)].

Using a notation like either of these makes it clearer to see that, e.g., 'R' in (R3) is never in predicate position and, hence, can be quantified over without ascending to second-order logic. (Cf. Shapiro 1979: 192–193, which offers a third notation, and cf. SNePS feature (S.4), mentioned in §5.3, above.)

[24]I owe this suggestion to Shapiro.

7 BELIEF REVISION

The system should also be capable of handling sequences of new information that might require it to revise its beliefs.[25] For instance, suppose that the system is given the following information at three successive times:

> at time t1: (33) The system's Lucy believes that Lucy's Lucy is sweet.
> at time t2: (34) The system's Lucy is sweet.
> at time t3: (35) The system's Lucy = Lucy's Lucy.

Then it should build the networks of Figures 18-20, successively.

At time t1 (Fig. 18), node m3 represents the system's Lucy: (Lucy system) and m7 represents Lucy's Lucy: (Lucy Lucy system).

At time t2 (Fig. 19), m13 is built, representing the system's belief that (Lucy system) (who is not yet believed to be—and, indeed, might *not* be—(Lucy Lucy system)) is sweet. (Cf. §5.4.3.)

At time t3 (Fig. 20), m14 is built, representing the system's new belief that there is really only one Lucy. This is a merging of the two "Lucy"-nodes. From now on, all properties of (Lucy system) will be inherited by the (Lucy Lucy system), by means of an inference rule for the EQUIV case frame (similar to rule (R3); cf. §6.3, and Maida & Shapiro 1982: 330)—but not *vice versa*, since properties that (Lucy system) believes (Lucy Lucy system) to have are not necessarily properties that (Lucy system) is believed by the system to have.

[25] The material in this section is based on conversations with Shapiro.

8 IMPLEMENTATION AND SAMPLE RUNS

A fragment of the system described here has been implemented using SNePS and an ATN parser-generator with a question-answering capability, augmented by a collection of LISP functions (MYFORB) that enable the ATN to build appropriate representations (as described in §§5–7). The implementation runs on a VAX 750 under UNIX at the SUNY Buffalo Department of Computer Science. The ATN, lexicon, and MYFORB package are listed in the appendix.

For details on SNePS, see Shapiro 1979 and Shapiro *et al.* 1983; for details on the ATN parser-generator, see Shapiro 1982. Understanding of some of the more cryptic listings below should be facilitated by the following fragments of the SNePS User Language (SNePSUL):

> (build <arc-1> <node-1> … <arc-n> <node-n>) is the SNePSUL command to build a network consisting of a non-dominated node with <arc-1> pointing to <node-1> and … and <arc-n> pointing to <node-n>.
>
> (find <arc-1> <node-1> … <arc-n> <node-n>) is the SNePSUL command to retrieve a node (in general, a set of nodes) with <arc-1> to <node-1> and … and <arc-n> to <node-n>; any <node-i> may be a variable, and any <arc-i> may be a *path* of arcs.
>
> (deduce <arc-1> <node-1> … <arc-n> <node-n>) is the SNePSUL command to find nodes satisfying the given description and to use already-built deduction rules together with existing nodes to build (infer) new nodes satisfying the description; the description may include variable nodes.
>
> (: <sentence>) is a SNePSUL command to parse the <sentence> using the ATN grammar. The system finds or builds a network representing the parse of the sentence; it then generates an English sentence that describes the relevant part of the network. If <sentence> is a question, the system uses its inference package to deduce a node, whose surface English generation is an answer to the question.
>
> (desc <nodes>) is a SNePSUL command to print a description of the <nodes>, showing all nodes with descending arcs together with their arcs and the nodes they point to; if <nodes> is '*nodes', the entire data base is described.
>
> Lines beginning with a semicolon are comments.

On the following pages are sample runs illustrating the capabilities (and some of the limitations) of the present implementation.[26]

[26]Added December 2006: This PDF version has been edited to save space by deleting some blank lines and other minor reformatting of the output.

```
Script started on Mon Aug  6 15:35:23 1984
OK-What next? snalps
Franz Lisp, Opus 38.79
Mon Aug  6 15:35:43 1984
The following arcs have been defined:

(before after)
(stime stime-)
(etime etime-)
(agent agent-)
(verb verb-)
(object object-)
(member member-)
(class class-)
(name name-)
(named named-)
(ego ego-)
(equiv equiv-)
(which which-)
(adj adj-)
(lex lex-)

using grammar gr.jul16.2
grammar atnined
lexicon lexined
forb.jul13 loaded
MYFORB loaded

Grammar for English Parsing, Question Answering and Generation based on Shapiro,
S. C., Generalized Augmented Transition Network Grammars for Generation from
Semantic Networks, AJCL 8,1 (Jan 1982), 12-25.

enter #now at first SNePS prompt to begin

use (ˆ(parse 7)) to set tracing package

sneps
*(demo parse.demo)
File parse.demo is now the source of input stream
nil
exec: 0.03 sec    gc: 0.00 sec
```

```
;=========================================================
; Demonstration of sentences parsed by the new ATN parser-generator.
;=========================================================
;
; To save space, only additions to the data base will be shown.
; Assigning *nodes to oldnodes will facilitate this:
;
* (*nodes = oldnodes)

nil
exec: 0.10 sec    gc: 0.00 sec

;---------------------------------------------------
; All parses must begin by telling the system that the time is now:
;
* #now
(b1)
exec: 0.08 sec    gc: 0.00 sec
;---------------------------------------------------
;
; A simple, subject-predicate sentence (cf. Fig. 1):
;
* (: John is tall)

(i understand that John is tall)
exec: 5.40 sec    gc: 2.96 sec

;
; The network representation of ‘John is tall’ (the pair of SNePSUL commands
; below allows presentation of the newly-built nodes only):
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m6 tall m5 be m4 m3 m2 John m1 b1 now)
exec: 0.13 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m6 (adj (m5 (lex (tall)))) (which (m3)))
(be)
(m4 (lex (be)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(b1)
(now (:val (b1)))
(dumped)
exec: 0.40 sec    gc: 0.00 sec

;
; (Node m4 is used for generation purposes only.)
```

```
;--------------------------------------------------
; More information about John is added to the beliefs of the system:
;
* (: John is poor)

(i understand that tall John is poor)
exec: 5.73 sec    gc: 0.00 sec

;
; The system’s beliefs now include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m8 poor m7)
exec: 0.13 sec    gc: 2.98 sec

;
* (desc *oldnodes)

(m8 (adj (m7 (lex (poor)))) (which (m3)))
(dumped)
exec: 0.21 sec    gc: 0.00 sec

;
;--------------------------------------------------
; An example of the question-answering capability of the system (in English):
;
* (ˆ (setq infertrace ’surface))

(surface)
exec: 0.05 sec    gc: 0.00 sec

;
* (: who is tall)

i wonder if
something is tall

i know
poor tall John is tall

(poor John is tall)
exec: 11.76 sec    gc: 0.00 sec
```

```
;--------------------------------------------------
; Another example:
;
* (: who is poor)

i wonder if
something is poor

i know
poor tall John is poor

(tall John is poor)
exec: 11.55 sec    gc: 2.91 sec

;
;--------------------------------------------------
; Information about Lucy is added:
;
* (: Lucy is young)

(i understand that Lucy is young)
exec: 5.51 sec    gc: 0.00 sec

;
; The system’s new beliefs are:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m13 young m12 m11 m10 Lucy m9 x m6 tall m5 be m4 m3 m2 John m1 b1 now)
exec: 0.21 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m13 (adj (m12 (lex (young)))) (which (m11)))
(m10 (name (m9 (lex (Lucy)))) (named (m11)))
(x (:val (q00013)))
(m6 (adj (m5 (lex (tall)))) (which (m3)))
(be)
(m4 (lex (be)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(b1)
(now (:val (b1)))
(dumped)
exec: 0.68 sec    gc: 0.00 sec

;
; (Node x was built as part of the question-answering process.)
```

```
;-----------------------------------------------------
; A de dicto report of one of John’s beliefs about ; someone he believes to be
; named ‘Lucy’ (cf. Fig. 4):
;
* (: John believes that Lucy is sweet)

(i understand that poor tall John believes that Lucy is sweet)
exec: 14.81 sec    gc: 12.65 sec

;
;  The system’s new beliefs are:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m22 m21 sweet m20 m19 m18 m17 m16 etm m15 believe m14 m8 poor m7)
exec: 0.30 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m22 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m21 (adj (m20 (lex (sweet)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m18 (object (m17 (name (m9 (lex (Lucy)))) (named (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(etm (:val (m15)))
(m8 (adj (m7 (lex (poor)))) (which (m3)))
(dumped)
exec: 0.83 sec    gc: 0.00 sec

;
; (The ETIME, STIME, and BEFORE arcs are part of the system’s ability to deal
; with tenses; note that the ‘‘now pointer’’ is BEFORE the E(nding)TIME of m22.
; These features of the system are not directly relevant to my project,
; and have generally been ignored.)
;
;-----------------------------------------------------
; The system is told that its Lucy is (also) sweet (cf. Fig. 7):
;
* (: Lucy is sweet)

(i understand that young Lucy is sweet)
exec: 5.63 sec    gc: 0.00 sec
```

```
; Now the system’s beliefs include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m23 m13 young m12 m11 m10 Lucy m9 x m6 tall m5 be m4 m3 m2 John m1 b1 now)
exec: 0.28 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m23 (adj (m20 (lex (sweet)))) (which (m11)))
(m13 (adj (m12 (lex (young)))) (which (m11)))
(m10 (name (m9 (lex (Lucy)))) (named (m11)))
(x (:val (q00013)))
(m6 (adj (m5 (lex (tall)))) (which (m3)))
(be)
(m4 (lex (be)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(b1 (before (m15)))
(now (:val (b1 (before (m15)))))
(dumped)
exec: 0.76 sec    gc: 0.00 sec

;---------------------------------------------------
; A de dicto report of one of John’s de se beliefs (cf. Fig. 13):
;
* (: John believes that he is rich)

(i understand that poor tall John believes that he* is rich)
exec: 11.18 sec    gc: 3.11 sec

;
; Now the system’s beliefs include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m28 m27 rich m26 m25 m24 b2 QI m22 m21 sweet m20 m19 m18
  m17 m16 etm m15 believe m14 m8 poor m7)
exec: 0.36 sec    gc: 0.00 sec
```

```
* (desc *oldnodes)

(m28 (object (m27 (adj (m26 (lex (rich)))) (which (b2))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m25 (object (m24 (ego (b2)))) (verb (m14 (lex (believe))))
     (agent (m3)))
(QI (:val (b2)))
(m22 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m21 (adj (m20 (lex (sweet)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m18 (object (m17 (name (m9 (lex (Lucy)))) (named (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(etm (:val (m15)))
(m8 (adj (m7 (lex (poor)))) (which (m3)))
(dumped)
exec: 1.26 sec    gc: 2.96 sec

;
;---------------------------------------------------
; A nested belief (cf. Fig. 9):
;
* (: Bill believes that Stu believes that Hector is philosophical)

(i understand that Bill believes that Stu believes that Hector is philosophical)
exec: 31.36 sec    gc: 23.61 sec

;
; Now the system’s beliefs include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m44 m43 m42 philosophical m41 m40 m39 m38 m37 Hector m36
  m35 m34 m33 Stu m32 m31 m30 Bill m29 m23 m13 young m12
  m11 m10 Lucy m9 x m6 tall m5 be m4 m3 m2 John m1 b1 now)
exec: 0.70 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m44 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m43 (etime (m15))
           (stime (m16 (before (b1 (before (m15))) (m15))))
           (object (m42 (adj (m41 (lex (philosophical))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m39 (object
      (m38 (object (m37 (name (m36 (lex (Hector))))
                        (named (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m34 (object (m33 (name (m32 (lex (Stu)))) (named (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m30 (name (m29 (lex (Bill)))) (named (m31)))
(m23 (adj (m20 (lex (sweet)))) (which (m11)))
(m13 (adj (m12 (lex (young)))) (which (m11)))
(m10 (name (m9 (lex (Lucy)))) (named (m11)))
(x (:val (q00013)))
(m6 (adj (m5 (lex (tall)))) (which (m3)))
(be)
(m4 (lex (be)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m15)))))
(dumped)
exec: 2.00 sec    gc: 0.00 sec

;
;---------------------------------------------------
;
; Another nested belief, adding new information to (Bill system)'s belief space
; and to (Stu Bill system)'s belief space, but using the same nodes to represent
; Bill, Stu, and Hector:
;
* (: Bill believes that Stu believes that Hector is Guatemalan)

(i understand that Bill believes that Stu believes that Hector is Guatemalan)
exec: 16.78 sec    gc: 6.63 sec

;
; Now the system's beliefs include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m48 m47 m46 Guatemalan m45 m28 m27 rich m26 m25 m24 b2 QI
  m22 m21 sweet m20 m19 m18 m17 m16 etm m15 believe m14 m8 poor m7)
exec: 0.88 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m48 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m47 (etime (m15))
           (stime (m16 (before (b1 (before (m15))) (m15))))
           (object (m46 (adj (m45 (lex (Guatemalan))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m28 (object (m27 (adj (m26 (lex (rich)))) (which (b2))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m25 (object (m24 (ego (b2)))) (verb (m14 (lex (believe))))
     (agent (m3)))
(QI (:val (b2)))
(m22 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m21 (adj (m20 (lex (sweet)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m18 (object (m17 (name (m9 (lex (Lucy)))) (named (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(etm (:val (m15)))
(m8 (adj (m7 (lex (poor)))) (which (m3)))
(dumped)
exec: 1.91 sec    gc: 0.00 sec

;
;---------------------------------------------------
; Another de dicto report of one of John’s beliefs about his Lucy:
;
* (: John believes that Lucy is rich)

(i understand that poor tall John believes that Lucy is rich)
exec: 11.80 sec    gc: 0.00 sec

;
; Note that the generator should, but does not, produce
; ‘‘i understand that poor tall John believes that sweet Lucy is rich’’.
; This is a modification that has not yet been made to the generation grammar,
; since the main concern in the present project has been representation,
; not generation.
;
; Now the system’s beliefs include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m50 m49 m44 m43 m42 philosophical m41 m40 m39 m38 m37
  Hector m36 m35 m34 m33 Stu m32 m31 m30 Bill m29 m23 m13
  young m12 m11 m10 Lucy m9 x m6 tall m5 be m4 m3 m2 John m1 b1 now)
exec: 0.80 sec    gc: 0.00 sec
```

```
* (desc *oldnodes)

(m50 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m49 (adj (m26 (lex (rich)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m44 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m43 (etime (m15))
           (stime (m16 (before (b1 (before (m15))) (m15))))
           (object (m42 (adj (m41 (lex (philosophical))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m39 (object
      (m38 (object (m37 (name (m36 (lex (Hector))))
                        (named (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m34 (object (m33 (name (m32 (lex (Stu)))) (named (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m30 (name (m29 (lex (Bill)))) (named (m31)))
(m23 (adj (m20 (lex (sweet)))) (which (m11)))
(m13 (adj (m12 (lex (young)))) (which (m11)))
(m10 (name (m9 (lex (Lucy)))) (named (m11)))
(x (:val (q00013)))
(m6 (adj (m5 (lex (tall)))) (which (m3)))
(be)
(m4 (lex (be)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m15)))))
(dumped)
exec: 2.35 sec    gc: 3.18 sec
```

```
;---------------------------------------------------
; A nested belief:  A de dicto report of Bill’s belief
; about Stu’s belief about Stu’s Lucy:
;
* (: Bill believes that Stu believes that Lucy is sweet)

(i understand that Bill believes that Stu believes that Lucy is sweet)
exec: 18.80 sec    gc: 6.85 sec

;
; Now the system’s beliefs include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m57 m56 m55 m54 m53 m52 m51 m48 m47 m46 Guatemalan m45
  m28 m27 rich m26 m25 m24 b2 QI m22 m21 sweet m20 m19 m18
  m17 m16 etm m15 believe m14 m8 poor m7)
exec: 1.11 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m57 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m56 (etime (m15))
           (stime (m16 (before (b1 (before (m15))) (m15))))
           (object (m55 (adj (m20 (lex (sweet))))
                        (which (m54))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m53 (object
      (m52 (object (m51 (name (m9 (lex (Lucy))))
                        (named (m54))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m48 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m47 (etime (m15))
           (stime (m16 (before (b1 (before (m15))) (m15))))
           (object (m46 (adj (m45 (lex (Guatemalan))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m28 (object (m27 (adj (m26 (lex (rich)))) (which (b2))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m25 (object (m24 (ego (b2)))) (verb (m14 (lex (believe))))
     (agent (m3)))
(QI (:val (b2)))
(m22 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m21 (adj (m20 (lex (sweet)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m18 (object (m17 (name (m9 (lex (Lucy)))) (named (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(etm (:val (m15)))
(m8 (adj (m7 (lex (poor)))) (which (m3)))
(dumped)
exec: 2.83 sec    gc: 0.00 sec

;
;-----------------------------------------------------
;
; A de dicto report of one of the system’s Lucy’s beliefs about Lucy’s John:
;
* (: Lucy believes that John is rich)

(i understand that sweet young Lucy believes that John is rich)
exec: 14.00 sec    gc: 3.33 sec

;
; Now the system’s beliefs include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m62 m61 m60 m59 m58 m50 m49 m44 m43 m42 philosophical m41
  m40 m39 m38 m37 Hector m36 m35 m34 m33 Stu m32 m31 m30
  Bill m29 m23 m13 young m12 m11 m10 Lucy m9 x m6 tall m5
  be m4 m3 m2 John m1 b1 now)
exec: 1.05 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m62 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m61 (adj (m26 (lex (rich)))) (which (m60))))
     (verb (m14 (lex (believe))))
     (agent (m11)))
(m59 (object (m58 (name (m1 (lex (John)))) (named (m60))))
     (verb (m14 (lex (believe))))
     (agent (m11)))
(m50 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m49 (adj (m26 (lex (rich)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m44 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m43 (etime (m15))
           (stime (m16 (before (b1 (before (m15))) (m15))))
           (object (m42 (adj (m41 (lex (philosophical))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m39 (object
      (m38 (object (m37 (name (m36 (lex (Hector))))
                        (named (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m34 (object (m33 (name (m32 (lex (Stu)))) (named (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m30 (name (m29 (lex (Bill)))) (named (m31)))
(m23 (adj (m20 (lex (sweet)))) (which (m11)))
(m13 (adj (m12 (lex (young)))) (which (m11)))
(m10 (name (m9 (lex (Lucy)))) (named (m11)))
(x (:val (q00013)))
(m6 (adj (m5 (lex (tall)))) (which (m3)))
(be)
(m4 (lex (be)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m15)))))
(dumped)
exec: 2.91 sec    gc: 0.00 sec

;
;----------------------------------------------------
; If the system is given the same de dicto report again,
; it does not modify its beliefs.
;
; The entire current data base is:
;
* (desc *nodes)

(m62 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m61 (adj (m26 (lex (rich)))) (which (m60))))
     (verb (m14 (lex (believe))))
     (agent (m11)))
(m59 (object (m58 (name (m1 (lex (John)))) (named (m60))))
     (verb (m14 (lex (believe))))
     (agent (m11)))
(m57 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m56 (etime (m15))
           (stime (m16 (before (b1 (before (m15))) (m15))))
           (object (m55 (adj (m20 (lex (sweet))))
                        (which (m54))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m53 (object
      (m52 (object (m51 (name (m9 (lex (Lucy))))
                        (named (m54))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m50 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m49 (adj (m26 (lex (rich)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m48 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m47 (etime (m15))
           (stime (m16 (before (b1 (before (m15)))
                               (m15))))
           (object (m46 (adj (m45 (lex (Guatemalan))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m44 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m43 (etime (m15))
           (stime (m16 (before (b1 (before (m15))) (m15))))
           (object (m42 (adj (m41 (lex (philosophical))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m39 (object
      (m38 (object (m37 (name (m36 (lex (Hector))))
                        (named (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m34 (object (m33 (name (m32 (lex (Stu)))) (named (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m30 (name (m29 (lex (Bill)))) (named (m31)))
(m28 (object (m27 (adj (m26 (lex (rich)))) (which (b2))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m25 (object (m24 (ego (b2)))) (verb (m14 (lex (believe))))
     (agent (m3)))
(QI (:val (b2)))
(m23 (adj (m20 (lex (sweet)))) (which (m11)))
(m22 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m21 (adj (m20 (lex (sweet)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m18 (object (m17 (name (m9 (lex (Lucy)))) (named (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(etm (:val (m15)))
(m13 (adj (m12 (lex (young)))) (which (m11)))
(m10 (name (m9 (lex (Lucy)))) (named (m11)))
(x (:val (q00013)))
(m8 (adj (m7 (lex (poor)))) (which (m3)))
(m6 (adj (m5 (lex (tall)))) (which (m3)))
(be)
(m4 (lex (be)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m15)))))
(oldnodes
 (:val (m62 (etime (m15))
            (stime (m16 (before (b1 (before (m15)))
                                (m15))))
            (object (m61 (adj (m26 (lex (rich))))
                         (which (m60))))
            (verb (m14 (lex (believe))))
            (agent (m11)))
       (m61 (adj (m26 (lex (rich)))) (which (m60)))
       (m60)
       (m59 (object (m58 (name (m1 (lex (John))))
                         (named (m60))))
            (verb (m14 (lex (believe))))
            (agent (m11)))
       (m58 (name (m1 (lex (John)))) (named (m60)))
       (m50 (etime (m15))
            (stime (m16 (before (b1 (before (m15)))
                                (m15))))
            (object (m49 (adj (m26 (lex (rich))))
                         (which (m19))))
            (verb (m14 (lex (believe))))
            (agent (m3)))
       (m49 (adj (m26 (lex (rich)))) (which (m19)))
       (m44 (etime (m15))
            (stime (m16 (before (b1 (before (m15)))
                                (m15))))
            (object
             (m43 (etime (m15))
                  (stime (m16 (before (b1 (before (m15)))
                                      (m15))))
                  (object
                   (m42 (adj (m41 (lex (philosophical))))
                        (which (m40))))
                  (verb (m14 (lex (believe))))
                  (agent (m35))))
            (verb (m14 (lex (believe))))
            (agent (m31)))
       (m43 (etime (m15))
            (stime (m16 (before (b1 (before (m15)))
                                (m15))))
            (object (m42 (adj (m41 (lex (philosophical))))
                         (which (m40))))
            (verb (m14 (lex (believe))))
            (agent (m35)))
       (m42 (adj (m41 (lex (philosophical))))
            (which (m40)))
       (philosophical)
       (m41 (lex (philosophical)))
       (m40)
       (m39 (object
             (m38 (object (m37 (name (m36 (lex (Hector))))
                               (named (m40))))
                  (verb (m14 (lex (believe))))
                  (agent (m35))))
            (verb (m14 (lex (believe))))
            (agent (m31)))
       (m38 (object (m37 (name (m36 (lex (Hector))))
                         (named (m40))))
            (verb (m14 (lex (believe))))
            (agent (m35)))
       (m37 (name (m36 (lex (Hector)))) (named (m40)))
       (Hector)
       (m36 (lex (Hector)))
       (m35)
       (m34 (object (m33 (name (m32 (lex (Stu))))
                         (named (m35))))
            (verb (m14 (lex (believe))))
            (agent (m31)))
       (m33 (name (m32 (lex (Stu)))) (named (m35)))
       (Stu)
       (m32 (lex (Stu)))
       (m31)
       (m30 (name (m29 (lex (Bill)))) (named (m31)))
       (Bill)
       (m29 (lex (Bill)))
       (m23 (adj (m20 (lex (sweet)))) (which (m11)))
       (m13 (adj (m12 (lex (young)))) (which (m11)))
       (young)
```

```
       (m12 (lex (young)))
       (m11)
       (m10 (name (m9 (lex (Lucy)))) (named (m11)))
       (Lucy)
       (m9 (lex (Lucy)))
       (x (:val (q00013)))
       (m6 (adj (m5 (lex (tall)))) (which (m3)))
       (tall)
       (m5 (lex (tall)))
       (be)
       (m4 (lex (be)))
       (m3)
       (m2 (name (m1 (lex (John)))) (named (m3)))
       (John)
       (m1 (lex (John)))
       (b1 (before (m15)))
       (now (:val (b1 (before (m15)))))))
(dumped)
exec: 10.51 sec    gc: 0.00 sec

;
; The same report repeated:
;
* (: Lucy believes that John is rich)

(i understand that sweet young Lucy believes that John is rich)
exec: 12.95 sec    gc: 0.00 sec

;
; Its beliefs are the same as before:
;
* (desc *nodes)

(m62 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m61 (adj (m26 (lex (rich))))
                  (which (m60))))
     (verb (m14 (lex (believe))))
     (agent (m11)))
(m59 (object (m58 (name (m1 (lex (John))))
                  (named (m60))))
     (verb (m14 (lex (believe))))
     (agent (m11)))
(m57 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m56 (etime (m15))
           (stime (m16 (before (b1 (before (m15)))
                               (m15))))
           (object (m55 (adj (m20 (lex (sweet))))
                        (which (m54))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m53 (object
      (m52 (object (m51 (name (m9 (lex (Lucy))))
                        (named (m54))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m50 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m49 (adj (m26 (lex (rich))))
                  (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m48 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m47 (etime (m15))
           (stime (m16 (before (b1 (before (m15)))
                               (m15))))
           (object (m46 (adj (m45 (lex (Guatemalan))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m44 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m43 (etime (m15))
           (stime (m16 (before (b1 (before (m15)))
                               (m15))))
           (object (m42 (adj (m41 (lex (philosophical))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m39 (object
      (m38 (object (m37 (name (m36 (lex (Hector))))
                        (named (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m34 (object (m33 (name (m32 (lex (Stu))))
                  (named (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m30 (name (m29 (lex (Bill)))) (named (m31)))
(m28 (object (m27 (adj (m26 (lex (rich))))
                  (which (b2))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m25 (object (m24 (ego (b2))))
     (verb (m14 (lex (believe)))) (agent (m3)))
(QI (:val (b2)))
(m23 (adj (m20 (lex (sweet)))) (which (m11)))
(m22 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m21 (adj (m20 (lex (sweet))))
                  (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m18 (object (m17 (name (m9 (lex (Lucy))))
                  (named (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(etm (:val (m15)))
(m13 (adj (m12 (lex (young)))) (which (m11)))
(m10 (name (m9 (lex (Lucy)))) (named (m11)))
(x (:val (q00013)))
(m8 (adj (m7 (lex (poor)))) (which (m3)))
(m6 (adj (m5 (lex (tall)))) (which (m3)))
(be)
(m4 (lex (be)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m15)))))
(oldnodes
 (:val (m62 (etime (m15))
            (stime (m16 (before (b1 (before (m15)))
                                (m15))))
            (object (m61 (adj (m26 (lex (rich))))
                         (which (m60))))
            (verb (m14 (lex (believe))))
            (agent (m11)))
       (m61 (adj (m26 (lex (rich)))) (which (m60)))
       (m60)
       (m59 (object (m58 (name (m1 (lex (John))))
                         (named (m60))))
            (verb (m14 (lex (believe))))
            (agent (m11)))
       (m58 (name (m1 (lex (John)))) (named (m60)))
       (m50 (etime (m15))
            (stime (m16 (before (b1 (before (m15)))
                                (m15))))
            (object (m49 (adj (m26 (lex (rich))))
                         (which (m19))))
            (verb (m14 (lex (believe))))
            (agent (m3)))
       (m49 (adj (m26 (lex (rich)))) (which (m19)))
       (m44 (etime (m15))
            (stime (m16 (before (b1 (before (m15)))
                                (m15))))
            (object
             (m43 (etime (m15))
                  (stime (m16 (before (b1 (before (m15)))
                                      (m15))))
                  (object
                   (m42 (adj (m41 (lex (philosophical))))
                        (which (m40))))
                  (verb (m14 (lex (believe))))
                  (agent (m35))))
            (verb (m14 (lex (believe))))
            (agent (m31)))
       (m43 (etime (m15))
            (stime (m16 (before (b1 (before (m15)))
                                (m15))))
            (object (m42 (adj (m41 (lex (philosophical))))
                         (which (m40))))
            (verb (m14 (lex (believe))))
            (agent (m35)))
       (m42 (adj (m41 (lex (philosophical))))
            (which (m40)))
       (philosophical)
       (m41 (lex (philosophical)))
       (m40)
       (m39 (object
             (m38 (object (m37 (name (m36 (lex (Hector))))
                               (named (m40))))
                  (verb (m14 (lex (believe))))
                  (agent (m35))))
            (verb (m14 (lex (believe))))
            (agent (m31)))
       (m38 (object (m37 (name (m36 (lex (Hector))))
                         (named (m40))))
            (verb (m14 (lex (believe))))
            (agent (m35)))
       (m37 (name (m36 (lex (Hector)))) (named (m40)))
       (Hector)
       (m36 (lex (Hector)))
       (m35)
       (m34 (object (m33 (name (m32 (lex (Stu))))
                         (named (m35))))
            (verb (m14 (lex (believe))))
            (agent (m31)))
       (m33 (name (m32 (lex (Stu)))) (named (m35)))
       (Stu)
       (m32 (lex (Stu)))
       (m31)
       (m30 (name (m29 (lex (Bill)))) (named (m31)))
       (Bill)
       (m29 (lex (Bill)))
       (m23 (adj (m20 (lex (sweet)))) (which (m11)))
       (m13 (adj (m12 (lex (young)))) (which (m11)))
       (young)
       (m12 (lex (young)))
       (m11)
```

```
       (m10 (name (m9 (lex (Lucy)))) (named (m11)))
       (Lucy)
       (m9 (lex (Lucy)))
       (x (:val (q00013)))
       (m6 (adj (m5 (lex (tall)))) (which (m3)))
       (tall)
       (m5 (lex (tall)))
       (be)
       (m4 (lex (be)))
       (m3)
       (m2 (name (m1 (lex (John)))) (named (m3)))
       (John)
       (m1 (lex (John)))
       (b1 (before (m15)))
       (now (:val (b1 (before (m15)))))))
(dumped)
exec: 10.38 sec    gc: 0.00 sec

;
;----------------------------------------------------
; A nested belief containing a de dicto report of Lucy’s John’s de se belief:
;
* (: Lucy believes that John believes that he is rich)

(i understand that sweet young Lucy believes that John
  believes that he* is rich)
exec: 21.73 sec    gc: 10.30 sec

;
; Now the system’s beliefs include:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m68 m67 m66 m65 m64 m63 b3 m57 m56 m55 m54 m53 m52 m51 m48
  m47 m46 Guatemalan m45 m28 m27 rich m26 m25 m24 b2 QI m22
  m21 sweet m20 m19 m18 m17 m16 etm m15 believe m14 m8 poor m7)
exec: 1.28 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m68 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m67 (object (m66 (adj (m26 (lex (rich))))
                        (which (b3))))
           (verb (m14 (lex (believe))))
           (agent (m60))))
     (verb (m14 (lex (believe))))
     (agent (m11)))
(m65 (object
      (m64 (object (m63 (ego (b3))))
           (verb (m14 (lex (believe))))
           (agent (m60))))
     (verb (m14 (lex (believe))))
     (agent (m11)))
(m57 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m56 (etime (m15))
           (stime (m16 (before (b1 (before (m15)))
                                   (m15))))
           (object (m55 (adj (m20 (lex (sweet))))
                        (which (m54))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m53 (object
      (m52 (object (m51 (name (m9 (lex (Lucy))))
                        (named (m54))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m48 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object
      (m47 (etime (m15))
           (stime (m16 (before (b1 (before (m15)))
                                   (m15))))
           (object (m46 (adj (m45 (lex (Guatemalan))))
                        (which (m40))))
           (verb (m14 (lex (believe))))
           (agent (m35))))
     (verb (m14 (lex (believe))))
     (agent (m31)))
(m28 (object (m27 (adj (m26 (lex (rich)))) (which (b2))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m25 (object (m24 (ego (b2)))) (verb (m14 (lex (believe))))
     (agent (m3)))
(QI (:val (b3)))
(m22 (etime (m15))
     (stime (m16 (before (b1 (before (m15))) (m15))))
     (object (m21 (adj (m20 (lex (sweet)))) (which (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(m18 (object (m17 (name (m9 (lex (Lucy)))) (named (m19))))
     (verb (m14 (lex (believe))))
     (agent (m3)))
(etm (:val (m15)))
(m8 (adj (m7 (lex (poor)))) (which (m3)))
(dumped)
exec: 3.36 sec    gc: 0.00 sec
```

```
;
;=========================================================

End of "parse.demo" demonstration
(t)
exec: 252.50 sec    gc: 84.68 sec

| text file input completed|
*(exit)
No files updated.
OK-What next?
script done on Mon Aug  6 15:46:22 1984
```

```
sneps
*(demo MandS.demo)
File MandS.demo is now the source of input stream
nil
exec: 0.06 sec    gc: 0.00 sec

;==========================================================
;
; Demonstration of the failure of the Maida \& Shapiro
; network to block the invalid inference
;
;     John believes that he is rich.
;     He = John.
;     -------------------------------
;     John believes that John is rich.
;
;==========================================================
;
; Build the first premise:
;
;     John believes that he is rich.
;
* (desc
  (build
    agent (build named- (build name (build lex John)))
    verb (build lex believe)
    object (build which (find (named- name lex) John)
                  adj (build lex rich))))

(m7 (object (m6 (adj (m5 (lex (rich)))) (which (m3))))
    (verb (m4 (lex (believe))))
    (agent (m3)))
(dumped)
exec: 0.71 sec    gc: 0.00 sec

;
; Show current data base:
;
* (dump *nodes)

(m7 (object (m6)) (verb (m4)) (agent (m3)))
(m6 (adj (m5)) (which (m3)) (object- (m7)))
(rich (lex- (m5)))
(m5 (lex (rich)) (adj- (m6)))
(believe (lex- (m4)))
(m4 (lex (believe)) (verb- (m7)))
(m3 (named- (m2)) (which- (m6)) (agent- (m7)))
(m2 (name (m1)) (named (m3)))
(John (lex- (m1)))
(m1 (lex (John)) (name- (m2)))
(dumped)
exec: 0.13 sec    gc: 0.00 sec
```

```
;---------------------------------------------------
; Build the second premise:
;
;     He = John.
;
* (desc (build equiv (find (named- name lex) John) equiv he))

(m8 (equiv (m3) (he)))
(dumped)
exec: 0.25 sec    gc: 0.00 sec

; Show current data base:
* (dump *nodes)

(he (equiv- (m8)))
(m8 (equiv (m3 he)))
(m7 (object (m6)) (verb (m4)) (agent (m3)))
(m6 (adj (m5)) (which (m3)) (object- (m7)))
(rich (lex- (m5)))
(m5 (lex (rich)) (adj- (m6)))
(believe (lex- (m4)))
(m4 (lex (believe)) (verb- (m7)))
(m3 (named- (m2)) (which- (m6)) (agent- (m7))
    (equiv- (m8)))
(m2 (name (m1)) (named (m3)))
(John (lex- (m1)))
(m1 (lex (John)) (name- (m2)))
(dumped)
exec: 0.16 sec    gc: 0.00 sec

;---------------------------------------------------
; Try to deduce the conclusion:
;
;     John believes that John is rich.
;
* (surface
  (deduce agent (find (named- name lex) John)
          verb (find lex believe)
          object %x))

i wonder if
(t00005 (object (q00004)) (verb (m4 (lex (believe))))
        (agent (m3)))

i know
(t00005 (object (q00004 (:val (m6 (adj (m5 (lex (rich))))
                                  (which (m3))))))
        (verb (m4 (lex (believe))))
        (agent (m3)))

John believes that John is rich
(dumped)
exec: 7.08 sec    gc: 2.95 sec
```

```
;
; Note that no premises about EQUIV or about ‘believe’ are necessary,
; simply because the first premise and the conclusion are represented
; by the same network.
;
; Hence, the ‘‘inference’’ is ‘‘static’’, not ‘‘dynamic’’;
; no new nodes are built:
;

* (dump *nodes)

(x (:val (q00004)))
(he (equiv- (m8)))
(m8 (equiv (m3 he)))
(m7 (object (m6)) (verb (m4)) (agent (m3)))
(m6 (adj (m5)) (which (m3)) (object- (m7)))
(rich (lex- (m5)))
(m5 (lex (rich)) (adj- (m6)))
(believe (lex- (m4)))
(m4 (lex (believe)) (verb- (m7)))
(m3 (named- (m2)) (which- (m6)) (agent- (m7))
    (equiv- (m8)))
(m2 (name (m1)) (named (m3)))
(John (lex- (m1)))
(m1 (lex (John)) (name- (m2)))
(dumped)
exec: 0.20 sec    gc: 0.00 sec

;
;==========================================================

End of "MandS.demo" demonstration
(t)
exec: 9.35 sec    gc: 2.95 sec

| text file input completed|
*(exit)
No files updated.
OK-What next?
script done on Mon Aug  6 15:57:22 1984
```

```
sneps
*(demo MSblock.demo)
File MSblock.demo is now the source of input stream
nil
exec: 0.05 sec    gc: 0.00 sec

;==========================================================
;
; Demonstration that the inference
;
;     John believes that he is rich.
;     He = John.
;     -------------------------------
;     John believes that John is rich.
;
; is blocked when the new representations are used, with or without rule (R2)
;
;==========================================================
;
; Build the first premise:
;
;     John believes that he is rich.
;
* #now
(b1)
exec: 0.16 sec    gc: 0.00 sec

;
* (: John believes that he is rich)

(i understand that John believes that he* is rich)
exec: 12.65 sec    gc: 9.08 sec

;
; Show current data base:
;
* (desc *nodes)

(m12 (object (m11 (adj (m10 (lex (rich)))) (which (b2))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(be)
(m9 (lex (be)))
(m8 (object (m7 (ego (b2)))) (verb (m4 (lex (believe))))
    (agent (m3)))
(QI (:val (b2)))
(m6 (before (b1 (before (m5))) (m5)))
(etm (:val (m5)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m5)))))
(dumped)
exec: 0.81 sec    gc: 0.00 sec
```

```
;----------------------------------------------------
;
; Build the second premise:
;
;     He = John.
;
* (desc (build
         equiv (find (named- name lex) John)
         equiv (find (ego- object- agent named- name lex) John)))

(m13 (equiv (m3) (b2)))
(dumped)
exec: 0.31 sec    gc: 0.00 sec

;
; Show current data base:
;
* (desc *nodes)

(m13 (equiv (m3) (b2)))
(m12 (object (m11 (adj (m10 (lex (rich)))) (which (b2))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(be)
(m9 (lex (be)))
(m8 (object (m7 (ego (b2)))) (verb (m4 (lex (believe))))
    (agent (m3)))
(QI (:val (b2)))
(m6 (before (b1 (before (m5))) (m5)))
(etm (:val (m5)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m5)))))
(dumped)
exec: 0.88 sec    gc: 0.00 sec

;----------------------------------------------------
;
; Try to deduce the conclusion:
;
;     John believes that John is rich.
;
* (ˆ (setq infertrace ’surface))

(surface)
exec: 0.10 sec    gc: 0.00 sec

;
* (surface
  (deduce agent (find (named- name lex) John)
          verb (find lex believe)
          object %x))

i wonder if
```

```
John believes something

i know
John believes that he* is rich

i know
John believes t00015

John believes m7 and
John believes that he* is rich
(dumped)
exec: 20.20 sec    gc: 0.00 sec

;
;---------------------------------------------------
;
; Build rule (R2):
;
;     For all x,y,z,F: If x believes F(y)
;                          and x believes z = y
;                         then x believes F(z).
;
* (desc
  (build
   avb ($x $y $z $F)
   &ant (build agent *x
               verb (findorbuild lex believe)
               object (build which *y
                             adj (build lex *F)))
   &ant (build agent *x
               verb (find lex believe)
               object (build equiv *y
                             equiv *z))
   cq (build agent *x
             verb (find lex believe)
             object (build which *z
                           adj (find lex *F)))))

(m21 (cq
      (m20 (object (m19 (adj (m14 (lex (v4))))
                        (which (v3))))
           (verb (m4 (lex (believe))))
           (agent (v1))))
     (&ant (m16 (object (m15 (adj (m14 (lex (v4))))
                             (which (v2))))
                (verb (m4 (lex (believe))))
                (agent (v1)))
           (m18 (object (m17 (equiv (v2) (v3))))
                (verb (m4 (lex (believe))))
                (agent (v1))))
     (avb (v4) (v3) (v2) (v1)))
(dumped)
exec: 1.83 sec    gc: 2.95 sec
```

```
;
;---------------------------------------------------
;
; Try again to infer the conclusion:
;
;     John believes that John is rich.
;
* (surface
  (deduce agent (find (named- name lex) John)
          verb (find lex believe)
          object %x))


i wonder if
John believes something

i know
John believes that he* is rich

i know
John believes t00029

i wonder if
John believes that something is v4

i wonder if
John believes t00060


John believes m7 and
John believes that he* is rich
(dumped)
exec: 31.66 sec    gc: 2.95 sec

;
;=========================================================

End of "MSblock.demo" demonstration
(t)
exec: 69.65 sec    gc: 17.95 sec

| text file input completed|
*(exit)
No files updated.
OK-What next?
script done on Mon Aug  6 16:25:07 1984
```

```
sneps
*(demo r2.demo)
File r2.demo is now the source of input stream
nil
exec: 0.03 sec    gc: 0.00 sec

;=========================================================
;
; Demonstration of inference rule (R2)
;
;=========================================================
;
; The inference:
;
;     John believes that he is rich.
;     John believes that he = the editor of Byte.
;     ------------------------------------------
;     John believes that the editor of Byte is rich.
;
; should be blocked without (R1).
;
;---------------------------------------------------
; Assign *nodes to oldnodes:
;
* (*nodes = oldnodes)

nil
exec: 0.20 sec    gc: 0.00 sec

;
;---------------------------------------------------
; Build the network for the first premise
; (by parsing it):
;
* #now
(b1)
exec: 0.06 sec    gc: 0.00 sec

;
* (: John believes that he is rich)

(i understand that John believes that he* is rich)
exec: 13.05 sec    gc: 9.31 sec

;
; Current data base:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m12 m11 rich m10 be m9 m8 m7 b2 QI m6 etm m5 believe m4 m3 m2 John m1 b1 now)
exec: 0.11 sec    gc: 0.00 sec

;
```

```
* (desc *oldnodes)

(m12 (object (m11 (adj (m10 (lex (rich)))) (which (b2))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(be)
(m9 (lex (be)))
(m8 (object (m7 (ego (b2)))) (verb (m4 (lex (believe))))
    (agent (m3)))
(QI (:val (b2)))
(m6 (before (b1 (before (m5))) (m5)))
(etm (:val (m5)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m5)))))
(dumped)
exec: 0.85 sec    gc: 0.00 sec

;
;---------------------------------------------------
; Build the network for the second premise, analyzed as:
;
;  (i)  John believes that someone is the editor of Byte.
; (ii)  John believes that he = that someone.
;
; Build (i):
;
* (surface
  (build agent (find (named- name lex) John)
         verb (find lex believe)
         object
          (build which #edByte
                adj (build lex |the editor of Byte|))))

John believes that b3 is the editor of Byte
(dumped)
exec: 5.45 sec    gc: 0.00 sec

;
; Data base now includes:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m15 m14 the editor of Byte m13 b3 edByte)
exec: 0.25 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m15 (object (m14 (adj (m13 (lex (|the editor of Byte|))))
                  (which (b3))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(edByte (:val (b3)))
```

```
(dumped)
exec: 0.31 sec    gc: 0.00 sec

;
; Build (ii):
;
* (surface
  (build
   agent (find (named- name lex) John)
   verb (find lex believe)
   object
    (build
      equiv
       (find (ego- object- agent named- name lex) John)
      equiv
       (find (which- object- agent named- name lex) John
             (which- adj lex) |the editor of Byte|))))

John believes m16
(dumped)
exec: 3.31 sec    gc: 0.00 sec

;
; Data base now includes:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m17 m16 m12 m11 rich m10 be m9 m8 m7 b2 QI m6 etm m5
  believe m4 m3 m2 John m1 b1 now)
exec: 0.20 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m17 (object (m16 (equiv (b2) (b3))))
     (verb (m4 (lex (believe)))) (agent (m3)))
(m12 (object (m11 (adj (m10 (lex (rich))))
                  (which (b2))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(be)
(m9 (lex (be)))
(m8 (object (m7 (ego (b2)))) (verb (m4 (lex (believe))))
    (agent (m3)))
(QI (:val (b2)))
(m6 (before (b1 (before (m5))) (m5)))
(etm (:val (m5)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m5)))))
(dumped)
exec: 1.00 sec    gc: 0.00 sec

;
```

```
;--------------------------------------------------
; Try to deduce the conclusion:
;
* (^ (setq infertrace 'surface))
(surface)
exec: 0.08 sec    gc: 0.00 sec

;
* (surface
  (deduce agent (find (named- name lex) John)
          verb (find lex believe)
          object %x))


i wonder if
John believes something

i know
John believes t00011

i know
John believes that b3 is the editor of Byte

i know
John believes that he* is rich

i know
John believes t00022


John believes m7 and
John believes that he* is rich and
John believes that b3 is the editor of Byte and
John believes m16
(dumped)
exec: 39.85 sec    gc: 6.05 sec

;
; Note that only the premises are retrieved.
;
;--------------------------------------------------
; Now add rule (R2):
;
* (desc
  (build
   avb ($x $y $z $F)
   &ant (build agent *x
               verb (findorbuild lex believe)
               object (build which *y
                             adj (build lex *F)))
   &ant (build agent *x
               verb (find lex believe)
               object (build equiv *y
                             equiv *z))
   cq (build agent *x
             verb (find lex believe)
             object (build which *z
                           adj (find lex *F)))))

(m25 (cq
      (m24 (object (m23 (adj (m18 (lex (v4))))
                        which (v3))))
           (verb (m4 (lex (believe))))
           (agent (v1))))
     (&ant (m20 (object (m19 (adj (m18 (lex (v4))))
                             (which (v2))))
                (verb (m4 (lex (believe))))
                (agent (v1)))
           (m22 (object (m21 (equiv (v2) (v3))))
                (verb (m4 (lex (believe))))
                (agent (v1))))
     (avb (v4) (v3) (v2) (v1)))
(dumped)
exec: 1.96 sec    gc: 0.00 sec

;
; Current data base now includes the rule:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m25 m24 m23 m22 m21 m20 m19 m18 v4 F v3 z v2 y v1 x m15
  m14 the editor of Byte m13 b3 edByte)
exec: 0.43 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m25 (cq
      (m24 (object (m23 (adj (m18 (lex (v4))))
                        (which (v3))))
           (verb (m4 (lex (believe))))
           (agent (v1))))
     (&ant (m20 (object (m19 (adj (m18 (lex (v4))))
                             (which (v2))))
                (verb (m4 (lex (believe))))
                (agent (v1)))
           (m22 (object (m21 (equiv (v2) (v3))))
                (verb (m4 (lex (believe))))
                (agent (v1))))
     (avb (v4) (v3) (v2) (v1)))
(F (:val (v4)))
(z (:val (v3)))
(y (:val (v2)))
(x (:val (v1)))
(m15 (object
      (m14 (adj (m13 (lex (|the editor of Byte|))))
           (which (b3))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(edByte (:val (b3)))
(dumped)
exec: 1.26 sec    gc: 0.00 sec

;
;---------------------------------------------------
; Now the deduction of the consequent occurs;
; not only are the premises retrieved, but the consequent is built:
;
* (surface
  (deduce agent (find (named- name lex) John)
          verb (find lex believe)
          object %x))

i wonder if
John believes something

i know
John believes t00032

i know
John believes that b3 is the editor of Byte

i know
John believes that he* is rich

i know
John believes t00043

i wonder if
John believes that something is v4

i wonder if
John believes t00074

since
John believes t00083 and
John believes that he* is rich

i infer
John believes that b3 is rich

since
John believes t00094 and
John believes that b3 is the editor of Byte

i infer
John believes that b3 is the editor of Byte

since
```

```
John believes t00105 and
John believes that he* is rich

i infer
John believes that he* is rich

since
John believes t00116 and
John believes that b3 is the editor of Byte

i infer
John believes that he* is the editor of Byte

since
John believes that he* is the editor of Byte and
John believes t00133

i infer
John believes that b3 is the editor of Byte

since
John believes that he* is the editor of Byte and
John believes t00144

i infer
John believes that he* is the editor of Byte

since
John believes that b3 is rich and
John believes t00155

i infer
John believes that b3 is rich

since
John believes that b3 is rich and
John believes t00166

i infer
John believes that he* is rich


John believes m7 and
John believes that he* is rich and
John believes that b3 is the editor of Byte and
John believes m16 and
John believes that he* is the editor of Byte and
John believes that b3 is rich
(dumped)
exec: 192.00 sec    gc: 30.78 sec

;
; Note also that not only is the desired consequent inferred, but so is the
; further consequent that John believes that he has the property of being the
; editor of Byte.
;
; Data base now includes:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m29 m28 m27 m26 m17 m16 m12 m11 rich m10 be m9 m8 m7 b2
  QI m6 etm m5 believe m4 m3 m2 John m1 b1 now)
exec: 0.50 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m29 (object (m28 (adj (m10 (lex (rich)))) (which (b3))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(m27 (object (m26 (adj (m13 (lex (|the editor of Byte|))))
                  (which (b2))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(m17 (object (m16 (equiv (b2) (b3))))
     (verb (m4 (lex (believe)))) (agent (m3)))
(m12 (object (m11 (adj (m10 (lex (rich)))) (which (b2))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(be)
(m9 (lex (be)))
(m8 (object (m7 (ego (b2)))) (verb (m4 (lex (believe))))
    (agent (m3)))
(QI (:val (b2)))
(m6 (before (b1 (before (m5))) (m5)))
(etm (:val (m5)))
(m2 (name (m1 (lex (John)))) (named (m3)))
(now (:val (b1 (before (m5)))))
(dumped)
exec: 1.40 sec    gc: 0.00 sec

;
;==========================================================

End of "r2.demo" demonstration
(t)
exec: 264.66 sec    gc: 49.11 sec

| text file input completed|
*(exit)
No files updated.
OK-What next?
script done on Wed Aug  8 09:59:10 1984
```

```
sneps
*(demo BelRev.demo)
File BelRev.demo is now the source of input stream
nil
exec: 0.08 sec    gc: 0.00 sec

;==========================================================
; Demonstration of belief revision
;==========================================================
;
; Assign *nodes to oldnodes:
;
* (*nodes = oldnodes)
nil
exec: 0.08 sec    gc: 0.00 sec

;
;----------------------------------------------------
; At time t1, the system is told that Lucy believes that Lucy is sweet,
; i.e., that (Lucy system) believes that (Lucy Lucy system) is sweet:
;
* #now
(b1)
exec: 0.10 sec    gc: 0.00 sec

;
* (: Lucy believes that Lucy is sweet)

(i understand that Lucy believes that Lucy is sweet)
exec: 15.66 sec    gc: 12.10 sec

;
; At time t1, the data base is:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m13 m12 sweet m11 be m10 m9 m8 m7 m6 etm m5 believe m4 m3 m2 Lucy m1 b1 now)
exec: 0.11 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m13 (etime (m5))
     (stime (m6 (before (b1 (before (m5))) (m5))))
     (object (m12 (adj (m11 (lex (sweet)))) (which (m9))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(be)
(m10 (lex (be)))
(m8 (object (m7 (name (m1 (lex (Lucy)))) (named (m9))))
    (verb (m4 (lex (believe))))
    (agent (m3)))
(etm (:val (m5)))
```

```
(m2 (name (m1 (lex (Lucy)))) (named (m3)))
(now (:val (b1 (before (m5)))))
(dumped)
exec: 0.91 sec    gc: 0.00 sec

;
;-------------------------------------------------
; At time t2, the system is told that Lucy is sweet,
; i.e., that (Lucy system) is sweet:
;
* (: Lucy is sweet)

(i understand that Lucy is sweet)
exec: 5.01 sec    gc: 0.00 sec

;
; At time t2, the data base includes:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m14)
exec: 0.20 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m14 (adj (m11 (lex (sweet)))) (which (m3)))
(dumped)
exec: 0.16 sec    gc: 0.00 sec

;
;-------------------------------------------------
;
; At time t3, the system is told that (Lucy system) is (Lucy Lucy system):
;
* (desc
  (build
   equiv (find (named- name lex) Lucy
               (agent- verb lex) believe)
   equiv
   (find (named- name lex) Lucy
         (named- object- agent named- name lex) Lucy)))

(m15 (equiv (m3) (m9)))
(dumped)
exec: 0.38 sec    gc: 0.00 sec

;
; At time t3, the data base includes:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m15 m13 m12 sweet m11 be m10 m9 m8 m7 m6 etm m5 believe
  m4 m3 m2 Lucy m1 b1 now)
exec: 0.13 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m15 (equiv (m3) (m9)))
(m13 (etime (m5))
     (stime (m6 (before (b1 (before (m5))) (m5))))
     (object (m12 (adj (m11 (lex (sweet))))
                  (which (m9))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(be)
(m10 (lex (be)))
(m8 (object (m7 (name (m1 (lex (Lucy)))) (named (m9))))
    (verb (m4 (lex (believe))))
    (agent (m3)))
(etm (:val (m5)))
(m2 (name (m1 (lex (Lucy)))) (named (m3)))
(now (:val (b1 (before (m5)))))
(dumped)
exec: 0.91 sec    gc: 0.00 sec

;
;---------------------------------------------------
; At time t4, the system is told that (Lucy system) is rich:
;
* (: Lucy is rich)

(i understand that sweet Lucy is rich)
exec: 5.71 sec    gc: 0.00 sec

;
; At time t4, the data base includes:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m17 rich m16 m14)
exec: 0.25 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m17 (adj (m16 (lex (rich)))) (which (m3)))
(m14 (adj (m11 (lex (sweet)))) (which (m3)))
(dumped)
exec: 0.26 sec    gc: 0.00 sec

;
;---------------------------------------------------
; At time t5, the system is told that (Lucy system)
; believes that (Lucy Lucy system) is philosophical:
```

```
;
* (: Lucy believes that Lucy is philosophical)

(i understand that rich sweet Lucy believes that Lucy is philosophical)
exec: 12.06 sec    gc: 3.13 sec

;
; At time t4, the data base includes:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m20 m19 philosophical m18 m15 m13 m12 sweet m11 be m10
  m9 m8 m7 m6 etm m5 believe m4 m3 m2 Lucy m1 b1 now)
exec: 0.20 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m20 (etime (m5))
     (stime (m6 (before (b1 (before (m5))) (m5))))
     (object (m19 (adj (m18 (lex (philosophical))))
                  (which (m9))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(m15 (equiv (m3) (m9)))
(m13 (etime (m5))
     (stime (m6 (before (b1 (before (m5))) (m5))))
     (object (m12 (adj (m11 (lex (sweet)))) (which (m9))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(be)
(m10 (lex (be)))
(m8 (object (m7 (name (m1 (lex (Lucy)))) (named (m9))))
    (verb (m4 (lex (believe))))
    (agent (m3)))
(etm (:val (m5)))
(m2 (name (m1 (lex (Lucy)))) (named (m3)))
(now (:val (b1 (before (m5)))))
(dumped)
exec: 1.28 sec    gc: 0.00 sec

;
;---------------------------------------------------
;
; Build the rule for the indiscernibility of identicals:
;
;     For all x, y, F:  If F(x) and x = y, then F(y).
;
* (desc
  (build avb ($x $y $F)
         &ant (build which *x
                     adj (build lex *F))
         &ant (build equiv *x
                     equiv *y)
         cq (build which *y
                   adj (find lex *F))))

(m25 (cq (m24 (adj (m21 (lex (v3)))) (which (v2))))
     (&ant (m22 (adj (m21 (lex (v3)))) (which (v1)))
           (m23 (equiv (v1) (v2))))
     (avb (v3) (v2) (v1)))
(dumped)
exec: 1.11 sec    gc: 0.00 sec

;
; Now the data base includes the rule:
;
* (*nodes - *oldnodes - oldnodes = oldnodes)

(m25 m24 m23 m22 m21 v3 F v2 y v1 x m17 rich m16 m14)
exec: 0.41 sec    gc: 0.00 sec

;
* (desc *oldnodes)

(m25 (cq (m24 (adj (m21 (lex (v3)))) (which (v2))))
     (&ant (m22 (adj (m21 (lex (v3)))) (which (v1)))
           (m23 (equiv (v1) (v2))))
     (avb (v3) (v2) (v1)))
(F (:val (v3)))
(y (:val (v2)))
(x (:val (v1)))
(m17 (adj (m16 (lex (rich)))) (which (m3)))
(m14 (adj (m11 (lex (sweet)))) (which (m3)))
(dumped)
exec: 0.78 sec    gc: 2.91 sec

;
;---------------------------------------------------
; Now, (Lucy system) believes that (Lucy Lucy system) is sweet and philosophical,
; the system believes that (Lucy system) is sweet and rich,
; and the system believes that (Lucy system) = (Lucy Lucy system).
;
; Hence, the system ought to believe that (Lucy Lucy system) is sweet and rich:
;
* (surface
  (deduce which
          (find (named- object- agent named- name lex)
                Lucy)
          adj %x))

i wonder if
(t00005 (adj (q00004)) (which (m9)))

i wonder if
(m22 (adj (m21 (lex (v3 (:val (v3))))))
     (which (v1 (:val (v1)))))

i wonder if
(m23 (equiv (v1 (:val (v1))) (v2 (:val (m9)))))

since
(m23 (equiv (v1 (:val (m3))) (v2 (:val (m9)))))
(m22 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v1 (:val (m3)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v2 (:val (m9)))))

since
(m23 (equiv (v1 (:val (m3))) (v2 (:val (m9)))))
(m22 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v1 (:val (m3)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v2 (:val (m9)))))

since
(m22 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v1 (:val (m9)))))
(m23 (equiv (v1 (:val (m9))) (v2 (:val (m9)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v2 (:val (m9)))))

since
(m22 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v1 (:val (m9)))))
(m23 (equiv (v1 (:val (m9))) (v2 (:val (m9)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v2 (:val (m9)))))

i wonder if
(m23 (equiv (v1 (:val (v1))) (v2 (:val (v2)))))

since
(m23 (equiv (v1 (:val (m3))) (v2 (:val (m9)))))
(m22 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v1 (:val (m3)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v2 (:val (m9)))))
```

```
since
(m23 (equiv (v1 (:val (m3))) (v2 (:val (m9)))))
(m22 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v1 (:val (m3)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v2 (:val (m9)))))

since
(m23 (equiv (v1 (:val (m9))) (v2 (:val (m9)))))
(m22 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v1 (:val (m9)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v2 (:val (m9)))))

since
(m23 (equiv (v1 (:val (m9))) (v2 (:val (m9)))))
(m22 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v1 (:val (m9)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v2 (:val (m9)))))

since
(m23 (equiv (v1 (:val (m3))) (v2 (:val (m3)))))
(m22 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v1 (:val (m3)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v2 (:val (m3)))))

since
(m23 (equiv (v1 (:val (m3))) (v2 (:val (m3)))))
(m22 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v1 (:val (m3)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v2 (:val (m3)))))

since
(m23 (equiv (v1 (:val (m9))) (v2 (:val (m3)))))
(m22 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v1 (:val (m9)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (rich))))))
     (which (v2 (:val (m3)))))
```

```
since
(m23 (equiv (v1 (:val (m9))) (v2 (:val (m3)))))
(m22 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v1 (:val (m9)))))

i infer
(m24 (adj (m21 (lex (v3 (:val (sweet))))))
     (which (v2 (:val (m3)))))

rich Lucy is sweet and
Lucy is rich and
rich Lucy is sweet and
Lucy is rich
(dumped)
exec: 35.38 sec    gc: 3.00 sec

;
;---------------------------------------------------
; However, the system should not believe that (Lucy system) is philosophical
; just because Lucy believes it (after all, Lucy might have a false belief).
; Nor should it believe that (Lucy Lucy system) is philosophical
; (after all, (Lucy Lucy system) might not be philosophical;
; (Lucy system) merely believes that (Lucy Lucy system) is philosophical):
;
* (surface
  (deduce which %x
          adj (find lex philosophical)))

i wonder if
(t00046 (adj (m18 (lex (philosophical)))) (which (q00045)))

(dumped)
exec: 0.81 sec    gc: 0.00 sec

;
; That is, no one is believed by the system to be philosophical.
;
;---------------------------------------------------
; But the system should believe that (Lucy system) believes that
; (Lucy Lucy system ) is philosophical:
;
* (desc
  (find (agent named- name lex) Lucy
        (verb lex) believe
        object (find which ?x
                     (adj lex) philosophical)))

(m20 (etime (m5))
     (stime (m6 (before (b1 (before (m5))) (m5))))
     (object (m19 (adj (m18 (lex (philosophical))))
                  (which (m9))))
     (verb (m4 (lex (believe))))
     (agent (m3)))
(dumped)
exec: 0.61 sec    gc: 0.00 sec

;
;---------------------------------------------------
;
; Notice the sentence generated from this node:
;
* (surface
  (find (agent named- name lex) Lucy
        (verb lex) believe
        object (find which ?x
                     (adj lex) philosophical)))

rich sweet Lucy believes that rich sweet Lucy is philosophical
(dumped)
exec: 8.65 sec    gc: 3.11 sec

;
; This is an incorrect de dicto report, because (Lucy system), who is believed
; by the system to be rich and sweet, does not (necessarily) believe that
; (Lucy Lucy system) is rich and sweet.
;
; This is a bug in the generation grammar, which is to be expected, since the
; project thus far has been concerned with representation, not generation.
;
;---------------------------------------------------
; Note the following curious parse, also due to the same generation bug:
;
* (: Lucy believes that Lucy is philosophical)

(i understand that rich sweet Lucy believes that rich
  sweet Lucy is philosophical)
exec: 12.35 sec    gc: 0.00 sec

;
;=========================================================

End of "BelRev.demo" demonstration
(t)
exec: 106.36 sec    gc: 27.18 sec

| text file input completed|
*(exit)
No files updated.
OK-What next?
script done on Wed Aug  8 10:50:39 1984
```

9 FUTURE WORK

There are several directions for future modifications. These can be classified into implementation issues, representation issues, extensions of the representation scheme, and extensions of the system as a whole.

9.1 Implementation Issues

Three major sections of the system in its current state have yet to be implemented:

(I) The ATN does not yet handle *de re* belief sentences. The decision to use

A believes of *x* that *p*

as the canonical form of *de re* belief sentences should facilitate this.

(II) Nor does the ATN currently handle English questions of such forms as:

What is believed by *A*?
What does *A* believe?
Who is believed to be F?
Who is believed by *A* to be F?
Whom does *A* believe to be F?

(III) The ATN generator has not yet been modified to handle all sentences that the parser can currently handle. Thus, many of the system's responses in the demonstrations in §8 contain occurrences of node identifiers, rather than being in English.

It should be noted that these are issues in natural-language processing, and not representational or SNePS-implementational issues.

Other important implementation modifications that need to be made include the following: The *forb-in-context* function needs to be augmented to allow a context to be evaluated, e.g.:

```
(forb-in-context <arcnodelist>
                 context ((find (named- name lex) John) system))
```

It would also be nice to allow the generator to generate adjectives within belief contexts, in the appropriate way (see, e.g., the demonstrations of parsing and of belief revision in §8). Finally, a deduce-in-context function is needed: Many of the deductions in the sample runs in §8 should be able to be expressed using the following format:

```
(deduce-in-context <arcnodelist> context <believer>)
```

9.2 Representation Issues

9.2.1 The EQUIV Arc

First, the node-merging mechanism of the EQUIV case frame with its associated rule needs to be generalized: Its current interpretation is co-referentiality; but if the sequence (33)–(35) were embedded in someone else's belief space, then *extensional co-referentiality* would be incorrect. What is needed is a notion of "co-referentiality-within-a-belief-space".

George Lakoff (1968), discussing a similar problem, has suggested using David Lewis's (1968) notion of *counterparts.* Thus, e.g., (Lucy Lucy system) might be a counterpart in her belief space of (Lucy system) or of (Lucy John system). An alternative is the relation of *consociation* (Castañeda 1972, 1975b). This seems, *prima facie*, preferable, since the entire framework of quasi-indicators and intensional entities meshes more naturally with Castañeda's system than with Lewis's.

It should be noted that other systems also have co-referentiality mechanisms. For instance, HAM-ANS (cf. §2.1.3) links nodes in different "belief networks" by a single CO-REF arc, but its implementers do not seem to realize that the nodes so joined must represent intensional objects.

Shapiro has pointed out[27] that there is also a problem with the EQUIV case frame as introduced in Maida & Shapiro 1982: 304ff. Presumably, a node representing an intensional object that has an extensional referent (that has

[27]In discussion.

a Sein-correlate, in the terminology of Rapaport 1978) can have a lone EQUIV arc pointing to it. Suppose that the system is told, incorrectly, that the Morning Star is Mercury (see Figure 19). If it is later told that this is false, Figure 19 would be modified to Figure 20.

But this not only negates m7, it also negates the still-true fact that m3 is self-EQUIV. This may be a problem merely for the EQUIV arc, which could be obviated by using counterparts or consociation. Or it might be a more serious problem. It needs to be resolved.

9.2.2 The EGO Arc

I have likened the EGO arc to Perry's ego function (cf. §5.4.4.2). But Castañeda has raised some objections to the ego function:

> Perry constructs the following rule:
>
> He*.2: With t as antecedent,
>
> [H*] *A believes that he* is so-and-so*
>
> has the sense of
>
> [S.E] *There is a sense s such that s = ego(t) and A believes that [s] is so and so.*
>
> [T]his rule is incorrect .... The whole point of the quasi-indicator 'he*' in [H*] is to depict the fact that the subject of the proposition that is the accusative of what the person A believes is ego(des(t)) [i.e., the ego of the designation of t]; obviously this is *not* represented in [S.E]. ... Hence, the two sentences must have different senses. (Castañeda 1983: 326-27.)

It is not clear whether the EGO arc is infested with this difficulty, or, if it is, whether it is as serious for our representational scheme as Castañeda feels it is for Perry.

Assuming that the EGO arc, or something like it, has no special philosophical problems, it has some potential further applications in the representation of self-knowledge. For instance, it can be used to represent the system's belief that it* is (or is not!) a computer, as in Figure 21. The implications of this for such claims as that "Machines lack an irreducible first-person perspective .... Therefore, machines are not agents" (Baker 1981: 157), especially in light of Castañeda's quasi-indexical theory of the first-person (Castañeda 1968), are exciting and worth exploring.

9.2.3 Quasi-Indicators in Nested Beliefs

There is a final representational issue that is, perhaps, more of an implementational problem. Currently, the parser treats a quasi-indicator in a nested belief context as having for its antecedent the *nearest* believer. Thus,

John believes that Ralph believes that he* (i.e., Ralph) is rich

can be parsed correctly, but

John believes that Ralph believes that he* (i.e., John) is rich

would be parsed the same way. Here, the 'he*'s *are* quasi-indicators, and, so, the representational formalism *can* handle these cases. But the determination of their antecedents must be handled contextually.

9.3 Extensions of the Representation

9.3.1 Knowledge

As hinted in §6.3, the representation needs to be extended to other cognitive as well as non-cognitive verbs. The case of 'know' poses special problems, for a number of reasons. For one thing, a *de dicto* knowledge report such as

(36) John knows that Bill is tall

is not fully propositionally transparent: While it might communicate the content of the proposition that John *believes*, it does not indicate whether John is aware that what he believes is true. That is, under what circumstances can it be inferred from (36) that John knows that he* knows that Bill is tall?

A second problem with 'know' concerns the generally accepted rule

(RK) $(\forall x, p)[\text{Knows}(x, p) \rightarrow p]$

But if p contains a quasi-indicator, (RK) fails: From 'John knows that he* is tall', we would not want the *system* to infer that he* is tall, since the system probably does not believe that he* (or it*) is tall, or even that John's self-concept is tall. Rather, it should infer that *John* is tall. Rules of inference for such cases need to be formulated and implemented. (Castañeda 1966: 152ff; 1967: 93ff discusses this.)[28]

9.3.2 Perry's Challenge

Perry (1983: 35; his numbering) offers to

> Those who would still like to pursue Castañeda's line ... a couple of challenging examples to chew on that haven't otherwise come up:
>
> **(25)** Sheila and I both believe that I am wanted on the telephone.
>
> **(26)** Sheila and I each believe that she and I are wanted on the telephone.
>
> **(27)** I believe that I am wanted on the telephone and Sheila believes it too.

Clearly, the system must be able to rise to these challenges.

9.4 Extensions to the System

There are two major extensions that must eventually be made to the system in order to meet the goal, discussed at the outset, of building a system that is pragmatically adequate (or "performatively competent"). For one thing, decisions need to be made about the deduction rules for belief. Should it be a rule that believers believe all the logical consequences of their beliefs? Probably not; but what rules should there be? At the other extreme is the position that there are no appropriate rules (cf. Moor's comment, mentioned in §6.3, fn. 1). Perhaps each believer should have his or her (or its) own set of rules, determined empirically (cf. Konolige 1984).

Finally, rather than treating all sentences of the form

(37) *A* believes that *x* is F.

as canonically *de dicto* and all sentences of the form

(38) *x* is believed by *A* to be F

as canonically *de re*, the system needs to be more flexible. In ordinary conversation, both sentences can be understood in either way, depending on context, including prior beliefs as well as idiosyncrasies of particular predicates. For instance, given

(8) John believes that the editor of *Byte* is rich,

and the fact that John is the editor of *Byte*, most people would infer

(7) John believes that he is rich.

But given[29]

(39) John believes that all identical twins are conceited.

(40) Unknown to John, he is an identical twin.

[28] Note added 2006: This problem has been solved. See Rapaport, William J.; Shapiro, Stuart C.; & Wiebe, Janyce M. (1997), "Quasi-Indexicals and Knowledge Reports", *Cognitive Science* 21: 63–107.

[29] I owe this example to Carol Glass.

most people would *not* infer

(41) John believes that he* is conceited.

Thus, we want to allow the system to make the most “reasonable” or psychologically plausible (*de re* vs. *de dicto*) interpretations of users’ belief reports, based on prior beliefs and on subject matter, and to modify its initial representation as more information is received.

APPENDIX

This appendix contains listings of the ATN parser-generator, the package of LISP functions (called ‘MYFORB’) that enable the system to handle nested beliefs and quasi-indicators, and the lexicon.

```
THE ATN PARSER-GENERATOR:
;==========================================================
;
; This is gr.jul16.2, modified from the grammar for
; English parsing, question answering, and generation in
; Shapiro 1982.
; Last modified: 16 jul 84
;
;==========================================================

(^ prin3 "The following arcs have been defined:" <><>)

(^ define before after stime stime- etime etime-
          agent agent- verb verb- object object-
          member member- class class-
          name name- named named-
          ego ego-
          equiv equiv-
          which which- adj adj- lex lex-)

(^ de add-indef (phrase)
     (cond ((member
             (cond ((equal (car (explode phrase)) "(" )
                           (cadr (explode phrase)))
                    (t (car (explode phrase))))
                   ’(a e i o))
            (pconcat ’an phrase))
           (t (pconcat ’a phrase))))

(^ de mylength (s)
     (cond ((null s) 0)
           ((atom s) 1)
           (t (add1 (mylength (cdr s))))))

(s ; parse a sentence and generate a response.
  (push sp t (sendr believer.reg ’(system))
             (jump respond)))

(respond; generate the response represented by the
        ; semantic node in *.
         (jump g (and (getr *) (overlap (getr type) ’d))
             ; the input was a statement represented by *.
             (setr string ’(i understand that)))
         (jump g (and (getr *) (eq (getr type) ’q))))
             ; the input was a question answered by *.
```

```
(sp ; parse a sentence.
    (wrd (who what) t
    ; if it starts with ‘who’ or ‘what’, it’s a question.
         (setr type ’q) (liftr type)
         (setr subj %x) (to v))
    (push npp t
    ; a statement starts with a noun phrase--its subject.
          (sendr type ’d) (sendr p-att) (sendr nh2)
          (sendr believer.reg)
          (setr type (cond ((nullr stype) ’d)
                           (t ’(d s))))
          (liftr type) (setr subj *)
          (to v)))

(v (cat v t ; the next word must be a verb.
          (setr p-att (getf prop-att))
          (setr vb (findorbuild lex (ˆ (getr *))))
          (setr tns (getf tense))
          (liftr nh2) (liftr stype)
          (to compl)))

(compl ; consider the word after the verb.
       (cat v (and (getf pprt)
                   (overlap (getr vb) (geta lex- ’be)))
              ; it must be a passive sentence.
              (setr obj (getr subj)) (setr subj nil)
              (setr vc ’pass)
              (setr vb (findorbuild lex (ˆ (getr *))))
              (to sv))
       (cat adj (overlap (getr vb) (geta lex- ’be))
                ; a predicate adjective.
                (setr adj (findorbuild lex (ˆ (getr *))))
                (liftr stype)
                (to svc))
       (jump sv *))

(sv ; start building the temporal structure.
    (jump o (and (getr *) (eq (getr type) ’q)))
            ; ignore the tense of a question.
    (jump o (and (getr *) (eq (getr tns) ’pres))
            ; present means starting before and ending
            ; after now.
          (setr
           stm
           (findorbuild before *now
                        before
                        (findorbuild after *now) = etm)))
    (jump o (and (getr *) (eq (getr tns) ’past))
            ; past means starting and ending before now.
          (setr stm
                (build before
                       (build before *now) = etm))))
```

```
(o ; parse what follows the verb group.
   (wrd that (getr p-att)
      ; a that-sentence will be NP + V of prop. att + that
      (addl believer.reg (getr subj)) (to svt))
   (wrd by (eq (getr vc) ’pass)
      ; a passive sentence will have ‘by np’.
      (to pag))
   (push npp t
      ; an active sentence will have an object np.
      (sendr type) (sendr p-att) (sendr pats)
      (setr obj *) (liftr vc)
      (to svo)))

(pag ; parse the subject np of a passive sentence.
     (push npp t
           (sendr type) (sendr pats)
           (setr subj *) (liftr vc)
           (to svo)))

(svo ; return a semantic node.
     (pop ; a de dicto belief statement
          (findorbuild agent (ˆ (getr subj))
                       verb (ˆ (getr vb))
                       object (ˆ (getr obj))
                       stime (ˆ (getr stm)) etime *etm)
          (eq (getr type) ’d))
     (pop ; a de se belief statement
          (findorbuild agent (ˆ (getr subj))
                       verb (ˆ (getr vb))
                       object (ˆ (getr obj)))
          (overlap (getr type) ’s))
     (pop (deduce agent (ˆ(getr subj))
                  verb (ˆ(getr vb))
                  object (ˆ(getr obj)))
          (and (nullr pats) (eq (getr type) ’q)))
     (pop ; an agent-verb-object question.
          (excpt
           (find
            arg-
             (deduce
               min 2 max 2
               arg (tbuild agent (ˆ (getr subj))
                           verb (ˆ(getr vb))
                           object (ˆ(getr obj)))
               arg (tbuild min (ˆ(mylength (getr pats)))
                           max (ˆ(mylength (getr pats)))
                           arg (ˆ(getr pats)))))
           ’(arg))
          (eq (getr type) ’q)))
```

```
(svt ; parse the M-object of the propositional attitude
  (push sp t
        (sendr p-att) (sendr believer.reg) (sendr type)
        (sendr pats) (sendr nh2)
        (setr obj *)
        (liftr vc) (liftr nh2) (liftr stype)
        (to svo)))

(svc
  (pop ; a noun-be-adj statement that is not embedded
       ; i.e., is believed by the system
       (eval (buildq (forbtop which + adj +) subj adj))
       (and (eq (getr type) ’d)
            (eq (popr (getr believer.reg)) ’system)))
  (pop ; a noun-be-adj statement that is embedded
       ; i.e., is within a belief-space
       (eval (buildq (forbnotop which + adj +) subj adj))
       (and (overlap (getr type) ’d)
            (pop2r (getr believer.reg))))
  (pop ; a noun-be-adj question.
       (deduce which (ˆ (getr subj)) adj (ˆ (getr adj)))
       (eq (getr type) ’q)))

(npp ; parse a noun phrase.
     (wrd (a an) t (setr indef t) (to npdet))
     (jump npdet *))

(npdet ; parse a np after the determiner.
  (cat adj t ; hold adjectives for later.
       (hold ’adj (findorbuild lex (ˆ (getr *))))
       (to npdet))
  (cat n (and (getr indef) (overlap (getr type) ’d))
       ; ‘a noun’ means a new member of the class noun.
       (setr
        nh
        (build member-
               (build class
                      (findorbuild lex (ˆ (getr *))))))
       (to npa))
  (cat n (and (getr indef) (eq (getr type) ’q))
       ; ‘a noun’ in a question refers to a known noun.
       (setr nh %y)
       (addr pats
             (tbuild member *y
                     class (tbuild lex (ˆ (getr *)))))
       (liftr pats) (to npaq))
  (cat npr t
       ; a proper noun is someone’s name, but it’s a new
       ; name if within a believes-context unless the name
       ; already exists in that context
       (setr
        nh
        (findorbuild named-
                     (forb-in-context
                      name (findorbuild lex (^(getr *)))
                      context (^(getr believer.reg)))))
       (setr nh2 nh) (liftr nh2)
       (to npa))
  (cat pron (getr p-att)
       ; a quasi-indicator within a de dicto
       ; belief context
       (setr
        nh
        (find
         ego-
         (^(cond ((find-in-context
                    ego ?QI
                    context (^(getr believer.reg))))
                 (t (build-in-context
                     ego #QI
                     context (^(getr believer.reg))))))))
       (setr stype 's) (liftr stype)
       (to npa)))

(npa
  ; remove all held adjectives and build which-adj
  ; propositions.
  (vir adj t
       (forbtop which (^ (getr nh)) adj (^ (getr *)))
       (liftr nh2) (liftr stype)
       (to npa))
  (pop nh t))

(npaq
  ; remove held adjectives and add patterns to a query np
  (vir adj t
       (addr pats (tbuild which (^(getr nh))
                              adj (^(getr *))))
       (liftr pats) (to npaq))
  (pop nh t))

(gs
  ; generate a sentence to express the semantic node in *.
  (group
   (jump gs1 (and (geta object) (overlap (getr vc) 'pass))
         ; a passive sentence is ‘object verb by agent’.
         (setr subj (geta object)) (setr obj (geta agent))
         (setr prep 'by))
   (jump gs1 (and (geta agent) (disjoint (getr vc) 'pass))
         ; an active sentence is ‘agent verb object’.
         (setr subj (geta agent)) (setr obj (geta object))
         (setr vc 'act))
   (jump gs1 (and (geta which)) (setr subj (geta which))
         ; a which-adj sentence is ‘which be adj’.
         (setr obj (geta adj)) (setr vc 'act))
   (call np
         ; a member-class sentence is ‘member be a class’
         (geta member) (geta member) (addr done *)
         (sendr done) reg (addr string reg)
         (jump gmemcl))))

(gmemcl
  (call pred * t ; generate a verb group using ‘be’
        (sendr numbr ’sing) (sendr vc ’act)
        (sendr vb ’be) reg (addr string reg)
        (jump gmemcl-cl)))

(gmemcl-cl
  (call np (geta class) t ; generate a np for the class
        (sendr done) (sendr indef t) (sendr numbr ’sing)
        * (addr string *) (to end)))

(gs1
  (call np subj t ; generate a np to express the subject.
        (addr done *) (sendr done) (sendr numbr) reg
        (addr string reg) (jump svb)))

(svb ; generate a verb group.  Use ‘be’ if no other verb.
  (call pred * t
        (sendr numbr) (sendr vc)
        (sendr vb (or (geta lex (geta verb)) ’be))
        reg (addr string reg) (jump surobj)))

(surobj
  ; generate a np to express the obj if there is one.
  (call np obj obj
        (sendr done) * (addr string prep *) (to end))
  (to (end) t))

(pred ; figure out the proper tense.
  (call past (geta etime) t tense (to genvb))
      ; past tense depends on ending time.
  (call futr (geta stime) t tense (to genvb))
      ; future tense depends on starting time.
  (to (genvb) t (setr tense ’pres)))
      ; present tense is the default.

(genvb ; return the verb group.
  (pop (verbize (getr numbr) (getr tense) (getr vc)
                (getr vb)) t))

(past ; if we can get to *now by before arcs,
      ; it is past tense.
      (to (pastend) (overlap * *now))
      (to (past (geta before)) t))

(pastend (pop ’past t))
```

```
(futr ; if we can get to *now by after arcs,
      ; it is future tense.
      (to (futrend) (overlap * *now))
      (to (futr (geta after)) t))

(futrend (pop ’futr t))

(np
  (group ; first see if there’s a propositional attitude
         (push g (or (geta which) (geta verb))
               ; generate an S to express the M-objective
               ; of a propositional attitude if there is
               ; one.  If the obj-register contains a
               ; propositional node, then that node has
               ; either a which-arc or a verb-arc
               (sendr done) (addr string ’that *)
               (to end))
         ; else: the proper number is pl for a class,
         ; sing for an individual
         (jump np1 numbr)
         (jump np1
               (or (geta sub-) (geta sup-) (geta class-))
               (setr numbr ’pl))
         (jump np1
               (not
                (or (geta sub-) (geta sup-) (geta class-)))
               (setr numbr ’sing))))

(np1 (call adjs (pconcat (geta which-) ’<>) (geta which-)
           (sendr done) string (jump npadj))
     (jump npadj *))

(npadj  ; generate a np to express *.
  (to (nppop) (and (geta :var) (geta :val))
      ; use the value of a variable
      (addr string (wrdize (getr numbr) (geta :val))))
  (to (nppop) (geta :var)
      ; use ‘something’ for a free variable
      (addr string ’something))
  (to (nppop) (geta lex)
      ; use the word at the end of the lex arc if present.
      (addr string (wrdize (getr numbr) (geta lex))))
  (to (nppop) (geta ego-)
      ; use ‘he*’ for quasi-indicators
      (addr string ’|he*|))
  (call names (pconcat (geta named-) ’<>) (geta named-)
        ; use its name if it has one
        (sendr done) reg (addr string reg) (to nppop))
  (call classes (pconcat (geta member-) ’<>)
        (geta member-)
        ; use its class if it has one
        (sendr done) (sendr numbr) reg (setr indef t)
        (addr string reg) (to nppop))
  (to (nppop) t ; use its identifier if nothing else
      (addr string *)))

(nppop (pop (add-indef (getr string)) indef)
       (pop string (nullr indef)))

(adjs
  ; generate a string of adjectives,
  ; one for each which-adj node in *.
  (wrd <> t (to endpop))
  (call np (geta adj) (and (disjoint * done)
                           (top\? (getr *)))
        ; only call np if * not done
        ; and embedded sentence is top
        (sendr done) * (addr string *) (to adjs))
  (to (adjs) t)
  (pop string t))

(names ; use the first usable name
  (call np (geta name) (and (disjoint * ’<>)
                            (disjoint * done))
        string (to flush))
  (to (names) (and (disjoint * ’<>) (overlap * done))))

(flush
  (wrd <> t (to endpop))
  (to (flush) (disjoint * ’<>)))

(classes ; use the first usable class
  (call np (geta class) (and (disjoint * ’<>)
                             (disjoint * done))
        (sendr numbr) string (to flush))
  (to (classes) (and (disjoint * ’<>) (overlap * done))))
```

```
;==========================================================
;=============== rulegrm 2/5/81 ===========================
;========== written by Stuart C. Shapiro =================
;
;  This is a grammar for generating English from SNePS
;  deduction rules.  To use it do the following:
;
;  1.  Add a generation grammar for atomic assertions.
;      This grammar must start at the state ‘gs’, must
;      build a sentence in the register ‘string’, and must
;      transfer to the state ‘end’, having consumed its
;      node.  The register ‘neg’ will be set to ‘t’, if
;      the sentence is to be negated.
;
;  2.  Only the top level of your network should terminate
;      at the state ‘end’.  Levels you call or push to
;      should terminate at some other state, such as
;      ‘endpop’.
;
;  3.  Start the parser at state ‘g’, or use the SNePS
;      function, ‘surface’.
;
;  4.  To trace inferences in English,
;      (setq infertrace ’surface).
;
;==========================================================

(g (jump grule * (setr conj ’and) (setr tab 0)))

(g1
  (group (jump grule (and (getr *) (getr count))
                (addr string (getr count) ’\))
         (setr count (add1 (getr count))))
  (jump grule (and (getr *) (nullr count)))))

(grule
  (group
   (jump grulec
         (and (getr *) (not (or (geta avb) (geta evb)))))
   (jump grulec (and (geta avb) (atom (geta avb)))
         (addr string "for every"
               (pack(snoc(unpack(geta avb))’\,))))
   (jump grulec (and (geta evb) (atom (geta evb)))
         (addr string "there exists a" (geta evb)
                      "such that"))
   (jump grulec
         (and (getr *) (eq (mylength (geta avb)) 2))
         (addr string "for every" (car(geta avb)) "and"
               (pack
                (snoc (unpack (cadr (geta avb)))’\,))))
   (jump grulec
         (and (getr *) (eq (mylength (geta evb)) 2))
         (addr string "there exists a" (car (geta evb))
               "and a" (cadr (geta evb)) "such that"))
   (jump grulec
         (and (getr *) (greaterp (mylength (geta avb)) 2))
         (addr string "for every"
               ((lambda (vbls)
                        (append (mapcar
                                 (function
                                  (lambda(v)
                                         (pack
                                          (snoc
                                           (unpack v)
                                           '\,))))
                                 (cddr vbls))
                                (list
                                 (cadr vbls) 'and
                                 (pack
                                  (snoc
                                   (unpack(car vbls))
                                   '\,)))))
                (geta avb))))
   (jump grulec
         (and (getr *) (greaterp (mylength(geta evb)) 2))
         (addr string
               "there exists"
               ((lambda (vbls)
                        (append
                         (mapconc
                          (function
                           (lambda (v)
                                  (list
                                   'a (pack
                                       (append
                                        (unpack v)
                                        '(\,))))))
                          (cddr vbls))
                         (list 'a (cadr vbls)
                               "and a"(car vbls))))
                (geta evb)) "such that"))))

(grulec
  (group
   (jump gorent (geta ant))
   (jump gnumquant (geta pevb) (setr emax (geta emax))
         (setr emin (geta emin)))
   (jump g&ent (or (geta cq) (geta dcq)))
   (jump gthresh (geta thresh)
         (setr tot (mylength (geta arg)))
         (setr thresh (geta thresh)))
   (jump gandor (geta max)
         (setr tot (mylength (geta arg)))
         (setr min (geta min)) (setr max (geta max)))
   (jump gs *))
  (to (end) t
      (prin3
       <>"======error -- no generation grammar for" <>)
      (describe (^ (getr \*)))))

(gandor
  (group (jump gandorn neg)
         (jump gandorp (nullr neg))))

(gandorn
  (group
   (jump gandorp-n1n (overlap min tot) (setr min 1))
   (jump gandorp-nnn
         (and (overlap min 1) (overlap max tot))
         (setr min tot))
   (call gandorp (pconcat (geta arg) ’<>) t
         (sendr conj ’and)
         (sendr tab (plus (getr tab) 3)) str
         (addr string "it is not the case that" ’<>
               ’% (plus 3 (getr tab)) str) (to end))))

(gandorp
  (group
   (call grule (pconcat (geta arg) ’<>) (overlap max 0)
         (sendr conj ’and)(sendr neg (nullr neg))
         (sendr tab) * (addr string *) (to end))
   (jump gandorp-nnn (overlap min tot))
   (jump gandorp-n1n
         (and (overlap min 1) (overlap max tot)))
   (call grule (pconcat (geta arg) ’<>)
         (and (overlap min 1) (overlap max 1))
         (sendr conj ’or) (sendr tab (plus 7 (getr tab)))
         (sendr neg) str
         (addr string "either" str) (to end))
   (call g1 (pconcat (geta arg) ’<>) (overlap max tot)
         (sendr neg) (sendr count 1)
         (sendr tab (plus 3 (getr tab))) str
         (addr string "at least" min
               "of the following:" ’<>
               ’% (plus 3 (getr tab)) str) (to end))
   (call g1 (pconcat (geta arg) ’<>) (overlap min 0)
         (sendr count 1) (sendr tab (plus 3 (getr tab)))
         (sendr neg) str
         (addr string "at most" max "of the following:"
               ’<> ’% (plus 3 (getr tab)) str) (to end))
   (call g1 (pconcat (geta arg) ’<>) (overlap min max)
         (sendr count 1) (sendr tab (plus 3 (getr tab)))
         (sendr neg) str
         (addr string "exactly" min "of the following:"
               ’<> ’% (plus 3 (getr tab)) str) (to end))
   (call g1 (pconcat (geta arg) ’<>) t (sendr count 1)
       (sendr tab (plus 3 (getr tab))) (sendr neg) str
       (addr string ’between min ’and max
             "of the following:" ’<>
```

```
            ’% (plus 3 (getr tab)) str) (to end))))

(gandorp-nnn
  (call grule (pconcat (geta arg) ’<>) t
        (sendr conj ’and) (sendr neg)
        (sendr tab) * (addr string *) (to end)))

(gandorp-n1n
  (call grule (pconcat (geta arg) ’<>)t
        (sendr conj ’or) (sendr tab (plus 7 (getr tab)))
        (sendr neg) str
        (addr string "either" str
              "(or" (cond ((overlap max 2) "both)")
                    (t (list ’all (getr max) ’\)))))
  (to end)))

(gnumquant
  (call grule (pconcat (geta &ant) (geta cq) ’<>)
        (nullr neg)
        (sendr conj ’and) (sendr tab (plus 3 (getr tab)))
        str (addr string "there is at most" (getr emax)
                  (geta pevb) "such that" ’<> ’%
                  (plus 3(getr tab)) str)
        (to end)))

(g&ent
  (group
   (call grule (pconcat (geta &ant) ’<>) (geta &ant)
         (sendr conj ’and) (sendr tab (plus 3 (getr tab)))
         (sendr string ’if) str
         (addr string
               (if (getr neg)
                   "it is not the case that" ’<>)
               str ’<> ’% (plus 3 (getr tab))) (jump gcq))
   (call grule (pconcat (geta cq) ’<>) t
         (sendr tab) (sendr conj ’and)
         * (addr string *) (to end))))

(gorent
  (call grule (pconcat (geta ant) ’<>) t (sendr conj ’or)
        (sendr tab (plus 3 (getr tab))) (sendr string ’if)
        str (addr string
                  (if (getr neg) "it is not the case that"
                      ’<>)
                  str ’<> ’% (plus 3 (getr tab)))
        (jump gcq)))

(gcq
  (call grule (pconcat (geta cq) ’<>) (geta cq)
        (sendr conj ’and) (sendr string ’then)
        (sendr tab (plus 8 (getr tab)))
        * (addr string *) (to end)))
```

```
(gthresh
  (group
   (jump gandorp neg
         (setr min thresh) (setr max (sub1 (getr tot))))
   (call grule (pconcat (geta arg) ’<>)
         (and (overlap tot 2) (overlap thresh 1))
         (sendr tab (plus 3 (getr tab)))
         (sendr conj "if and only if")
         str (addr string str) (to end))
   (call g1 (pconcat (geta arg) ’<>) (overlap thresh 1)
         (sendr count 1) (sendr tab (plus 3 (getr tab)))
         (sendr conj ’and) str
         (addr string
               "the following are equivalent:" ’<> str)
         (to end))
   (call g1 (pconcat (geta arg) ’<>) t
         (sendr count 1) (sendr conj ’and)
         (sendr tab (plus 3 (getr tab))) str
         (addr string "if any of the following are true,"
               "they all are:" ’<> str) (to end))))

(end
  (group
   (wrd <> t (to endpop))
   (jump g1 * (addr string conj ’<> ’% (getr tab)))
   (pop string t)))

(endpop (pop string t))

(ˆ prin3 <> "using grammar gr.jul16.2" <>)
```

MYFORB:

```
;==========================================================
;
; THIS IS forb.jul13, containing Franz LISP functions for
; forb-in-context.
; Last modified: 16 jul 84
;
;==========================================================
;
; Data Structures:
; ================
;
;   <arcnodeset>    ::= <arc><nodeset>
;   <arcnodelist>   ::= <arcnodeset>+
;   <believed-node> ::= <arcnodelist>[ context <Ct> ]
;   <Ct>            ::= <believer>
;
; A <believer> is the contents of the believer.reg
; register created by the ATN.  The register is a stack
; of all believers, with the stack-top the current
; believer and the 2nd element the ‘‘previous’’ or
; meta-believer.  E.g., (Lucy John system) is the
; (system’s concept of John)’s concept of Lucy; i.e.,
; Lucy
;     John
;         system
;
;==========================================================
;
; Functions:
;
;==========================================================
;
; forb-in-context
; ===============
;
; Takes a <believed-node> and finds it in context or
; builds it in context
;
(def forb-in-context
  (macro (believed-node)
         ‘(or (find-in-context ,@(cdr believed-node))
              (build-in-context ,@(cdr believed-node)))))
;----------------------------------------------------
;
; find-in-context
; ===============
;
; Takes a <believed-node>, i.e., an <arcnodelist> and a
; <context>; if it finds a node corresponding to
; <arcnodelist> in the <context>, then it returns it;
; else returns nil.
```

```
;
(def find-in-context
  (macro
   (believed-node)
    `(cond ((find ,@(arcnodelist-of (cdr believed-node)))
            (find-in-context1
             (find ,@(arcnodelist-of (cdr believed-node)))
             ',(believer believed-node)))
           (t nil))))
;---------------------------------------------------
;
; find-in-context1
; ================
;
(de find-in-context1 (nodeset belief-stack)
  (cond ((null nodeset) nil)
        ((equal (context-of (firstnode nodeset))
                belief-stack)
         (list (firstnode nodeset)))
        (t
         (find-in-context1 (cdr nodeset) belief-stack))))
;---------------------------------------------------
;
; context-of
; ==========
;
; Takes a node and returns the stack of its believers,
; defaulting to nil.
;
(de context-of (node)
  (cond ((top\? node) '(system))
        ((is-obj-of-bel node)
         (mk-bel-stack
          (agent-of node)
          (context-of (firstnode (bel-space-of node)))))))
;---------------------------------------------------
;
; is-obj-of-bel
; =============
;
; Takes a node & returns non-nil if the node is the object
; of an agent-believes-object frame, else nil.
;
(de is-obj-of-bel (node)
  (cond ((find object (^ node) (verb lex) believe) t)
        (t nil)))
;---------------------------------------------------
;
; mk-bel-stack
; ============
;
; Pushes a new believer onto a stack of believers
;
```

```
(de mk-bel-stack (agent-node believer-stack)
  (cons agent-node believer-stack))
;--------------------------------------------------
;
; agent-of
; ========
;
; Takes a node and returns the agent who believes it.
;
(de agent-of (node)
  (cond ((top\? node) ’system)
        (t (firstnode (find (agent- verb lex) believe
                            (agent- object) (ˆ node))))))
;--------------------------------------------------
;
; bel-space-of
; ============
;
; Takes a node and returns the node representing its
; belief space, i.e., (system) or the node representing
; the proposition that some agent believes it.
;
(de bel-space-of (node)
  (cond ((top\? node) ’(system))
        (t (find object (ˆ node) (verb lex) believe))))
;--------------------------------------------------
;
; nextnode
; ========
;
; Takes a nodeset and returns its second node.
;
(de nextnode (nodeset)
  (firstnode (cdr nodeset)))
;--------------------------------------------------
.bp
;--------------------------------------------------
;
; build-in-context
; ================
;
; Takes a <believed-node>, i.e., an <arcnodelist> and a
; <context>, and returns a built node in that context.
; (Written by SCS.)
;
(def build-in-context
  (nlambda (args)
         (let ((prop))
              (funcall ’build-in-context1 args)
              prop)))

(def build-in-context1
  (macro (believed-node)
         (cond ((eq (believer-of believed-node) ’system)
                ‘(newsetq
                  prop
                  ,(cons ’build
                         (arcnodelist-of believed-node))))
               (t
                ‘(funcall
                  ’build-in-context1
                  ’(agent ,(believer-of believed-node)
                    verb (findorbuild lex believe)
                    object
                     (ˆ (newsetq
                         prop
                         (build
                          ,@(arcnodelist-of
                             believed-node))))
                    context
                    ,(meta-believer believed-node)))))))

(defmacro newsetq (sym value)
  ‘((lambda (v) (cond (,sym v) (t (setq ,sym v))))
    ,value))
;---------------------------------------------------
;
; believer-of
; ===========
;
; Takes a <believed-node> and returns the first element
; of its context (the current, or local, believer).
;
(de believer-of (believed-node)
  (cond ((member ’context believed-node)
         (popr (believer believed-node)))
        (t ’system)))
;---------------------------------------------------
;
; believer
; ========
;
; Takes a <believed-node> and returns its context,
; defaulting to ’system.
;
(de believer (believed-node)
  (cond ((member ’context believed-node)
         (sneval (rac believed-node)))
        (t ’(system))))
;---------------------------------------------------
;
; rac
; ===
;
; Takes a list and returns its last element.
;
```

```
(de rac (l)
 (car (reverse l)))
;---------------------------------------------------
;
; popr
; ====
;
; Takes a stack implemented as a list and returns its top.
;
(de popr (stack)
  (cond ((listp stack) (car stack))
        (t stack)))
;---------------------------------------------------
;
; arcnodelist-of
; ==============
;
; Takes a <believed-node> and returns its <arcnodelist>.
;
(de arcnodelist-of (believed-node)
 (cond ((member ’context believed-node)
        (rddc believed-node))
       (t believed-node)))
;---------------------------------------------------
;
; rddc
; ====
;
; Takes a list and returns a list of all but the last two
; elements.
;
(de rddc (l)
 (reverse (cddr (reverse l))))
;---------------------------------------------------
;
; meta-believer
; =============
;
; Takes a <believed-node> and returns its meta-believer;
; i.e., the cdr of its context.
;
(de meta-believer (believed-node)
  (cond ((member ’context believed-node)
         (cond ((eq (believer believed-node) ’(system))
                ’(system))
               (t (cdr (believer believed-node)))))
        (t ’(system))))
;---------------------------------------------------
;
; pop2r
; =====
;
; Takes a stack implemented as a list and returns its
; second element.
;
(de pop2r (stack)
  (cadr stack))
;----------------------------------------------------
;
; snfuncs
; =======
;
; Modifies snfuncs (the SNePS oblist).
;
(setq snfuncs
  '(build find # $ % * : add deduce def-path define
          define-aux delrel demo describe desc dump erase
          findorbuild gfind insys lisp nfind outsys remvar
          resume surface tbuild forbtop forbnotop
          find-in-context build-in-context forb-in-context
          intext clear-infer exit myequal))
;==========================================================
(prin3 <> "forb.jul13 loaded" <>)
```

LEXICON for use with the grammar
(modified from the lexicon in Shapiro 1982):

```
(a ((ctgy . det)))
(an ((ctgy . det)))
(ant ((ctgy . n)))
(animal ((ctgy . n)))
(bat ((ctgy . n)))
(be ((ctgy . v)))
(believe ((ctgy . v)(prop-att . t)))
(Bill ((ctgy . npr)(gen . m)))
(car ((ctgy . n)))
(Carol ((ctgy . npr)(gen . f)))
(cat ((ctgy . n)))
(dog ((ctgy . n)))
(elk ((ctgy . n)))
(fox ((ctgy . n)))
(girl ((ctgy . n)))
(gnu ((ctgy . n)))
(Guatemalan ((ctgy . adj)))
(he ((ctgy . pron)))
(Hector ((ctgy . npr)(gen . m)))
(hen ((ctgy . n)))
(is ((ctgy . v)(root . be)(num . sing)(tense . pres)))
(it ((ctgy . pron)))
(John ((ctgy . npr)(gen . m)))
(Lucy ((ctgy . npr)(gen . f)))
(man ((ctgy . n)(plur . men)))
(Mary ((ctgy .  npr)(gen . f)))
(men ((ctgy . n)(root . man)(num . plur)))
(mortal ((ctgy . adj)))
(old ((ctgy . adj)))
(owl ((ctgy . n)))
(pear ((ctgy . n)))
(pet ((ctgy . n)))
(people ((ctgy . n)(root . person)(num . plur)))
(person ((ctgy . n)(plur . people)))
(philosophical ((ctgy . adj)))
(pick ((ctgy . v)))
(poor ((ctgy . adj)))
(rat ((ctgy . n)))
(rich ((ctgy . adj)))
(saw ((ctgy . n)(root . saw1))
     ((ctgy . v)(root . see)(tense . past)))
(saw1 ((ctgy . n)(root . saw)))
(see ((ctgy . v)(past . saw)(pastp . seen)))
(seen ((ctgy . v)(root . see)(tense . pastp)(pprt . t)))
(she ((ctgy . pron)))
(smart ((ctgy . adj)))
(Socrates ((ctgy . npr)(gen . m)))
(sow ((ctgy . n)))
(Stu ((ctgy . npr)(gen . m)))
(sweet ((ctgy . adj)))
```

```
(tall ((ctgy . adj)))
(that ((ctgy . conj)))
(the ((ctgy . det)))
(the-editor-of-Byte ((ctgy . npr)(gen m f)))
(what ((ctgy . pron)))
(was ((ctgy . v)(root . be)(num . sing)(tense . past)))
(who ((ctgy . pron)))
(wise ((ctgy . adj)))
(yak ((ctgy . n)))
(young ((ctgy . adj)))
```

References

Robert M. Adams and Hector-Neri Castañeda, "Knowledge and Belief: A Correspondence," in J.E. Tomberlin (ed.), *Agent, Language, and the Structure of the World* (Indianapolis: Hackett, 1983): 293–309.

Lynne Rudder Baker, "Why Computers Can't Act," *American Philosophical Quarterly* 18(1981)157–63.

John A. Barnden, "Intensions as Such: An Outline," *Proc. 8th International Joint Conference on Artificial Intelligence* (IJCAI-83), Vol. 1(1983): 280–86.

Steven E. Boer and William G. Lycan, "Who, Me?," *Philosophical Review* 89(1980)427–66.

Ronald J. Brachman, "What's in a Concept: Structural Foundations for Semantic Networks," *International Journal for Man-Machine Studies* 9(1977)127–52.

Myles Brand, *Intending and Acting* (Cambridge, MA: MIT Press, 1984).

Franz Brentano, "The Distinction between Mental and Physical Phenomena" (1874), trans. D.B. Terrell, in R.M. Chisholm (ed.), *Realism and the Background of Phenomenology* (New York: Free Press, 1960): 39–61.

Boruch A. Brody, "Glossary of Logical Terms," in P. Edwards (ed.), *Encyclopedia of Philosophy*, Vol. 5 (New York: Macmillan and Free Press, 1967): 57–77.

M. Sandra Carberry, "Understanding Pragmatically Ill-Formed Input," *Proc. 10th International Conference on Computational Linguistics* (COLING-84) (Morristown, NJ: Assoc. for Computational Linguistics, 1984): 200–206.

Rudolf Carnap, *Meaning and Necessity*, 2nd ed. (Chicago: University of Chicago Press, 1956).

Hector-Neri Castañeda, " 'He': A Study in the Logic of Self-Consciousness," *Ratio* 8(1966)130–57.

Hector-Neri Castañeda, "On the Logic of Self-Knowledge," *Noûs* 1(1967a)9–21.

Hector-Neri Castañeda, "Indicators and Quasi-Indicators," *American Philosophical Quarterly* 4(1967b)85–100.

Hector-Neri Castañeda, "On the Logic of Attributions of Self-Knowledge to Others," *Journal of Philosophy* 54(1968a)439–56.

Hector-Neri Castañeda, "On the Phenomeno-Logic of the I," *Proc. 14th International Congress of Philosophy* (Vienna: Herder, 1968b): 260–66.

Hector-Neri Castañeda, "On the Philosophical Foundations of the Theory of Communication: Reference" (originally written in 1970) *Midwest Studies in Philosophy* 2(1977)165–86.

Hector-Neri Castañeda, "Thinking and the Structure of the World," *Philosophia* 4(1974)3–40. Originally written in 1972. Reprinted in 1975 in *Critica* 6(1972)43–86.

Hector-Neri Castañeda, "Individuals and Non-Identity: A New Look," *American Philosophical Quarterly* 12(1975a)131–40.

Hector-Neri Castañeda, "Identity and Sameness," *Philosophia* 5(1975b)121–50.

Hector-Neri Castañeda, *Thinking and Doing* (Dordrecht: D. Reidel, 1975c).

Hector-Neri Castañeda, "Perception, Belief, and the Structure of Physical Objects and Consciousness," *Synthese* 35(1977)285–351.

Hector-Neri Castañeda, "On the Philosophical Foundations of the Theory of Communication: Reference," *Midwest Studies in Philosophy* 2(1977b)165–86.

Hector-Neri Castañeda, "Fiction and Reality: Their Basic Connections," *Poetica* 8(1979)31–62.

Hector-Neri Castañeda, "Reference, Reality, and Perceptual Fields," *Proc. and Addresses of the American Philosophical Assoc.* 53(1980)763–823.

Hector-Neri Castañeda, "Reply to John Perry: Meaning, Belief, and Reference," in J.E. Tomberlin (ed.), *Agent, Language, and the Structure of the World* (Indianapolis: Hackett, 1983): 313–27.

Roderick M. Chisholm, "Intentionality," in P. Edwards (ed.), *Encyclopedia of Philosophy*, Vol. 4 (New York: Macmillan and Free Press, 1967): 201–04.

Roderick M. Chisholm, "Thought and Its Reference," *American Philosophical Quarterly* 14(1977)167–72.

Alonzo Church, "On Carnap's Analysis of Statements of Assertion and Belief" (1950), in L. Linsky (ed.), *Reference and Modality* (London: Oxford University Press, 1971): 168–70.

Alonzo Church, "A Formulation of the Logic of Sense and Denotation," in P. Henle *et al.* (eds.), *Structure, Method, and Meaning* (New York: Liberal Arts Press, 1951).

Alonzo Church, "Outline of a Revised Formulation of the Logic of Sense and Denotation," Part I, *Noûs* 7(1973)24–33; Part II, *Noûs* 8(1974)135–56.

Herbert H. Clark and Catherine R. Marshall, "Definite Reference and Mutual Knowledge," in A.K. Joshi, B.L. Webber, and I.A. Sag (eds.), *Elements of Discourse Understanding* (Cambridge: Cambridge University Press, 1981).

Philip R. Cohen and C. Raymond Perrault, "Elements of a Plan-Based Theory of Speech Acts" (1979), reprinted in B.L. Webber and N.J. Nilsson (eds.), *Readings in Artificial Intelligence* (Palo Alto: Tioga, 1981): 478–495.

Lewis G. Creary, "Propositional Attitudes: Fregean Representation and Simulative Reasoning," *Proc. 6th International Joint Conference on Artificial Intelligence* (IJCAI-79), Vol. I (1979): 176–81.

M.J. Cresswell, "Quotational Theories of Propositional Attitudes," *Journal of Philosophical Logic* 9(1980)17–40.

Daniel C. Dennett, "Artificial Intelligence as Philosophy and as Psychology," in D.C. Dennett, *Brainstorms: Philosophical Essays on Mind and Psychology* (Montgomery, VT: Bradford Books, 1978): 109–26.

Daniel C. Dennett, "Intentional Systems in Cognitive Ethology: The 'Panglossian Paradigm' Defended," *Brain and Behavioral*

Sciences 6(1983)343–390.
Jon Doyle, "A Truth Maintenance System," *Artificial Intelligence* 12(1979)231–72.
Michael Dummett, "Gottlob Frege," in P. Edwards (ed.), *Encyclopedia of Philosophy*, Vol. 3 (New York: Macmillan and Free Press, 1967): 225–37.
Richard H. Feldman, "Belief and Inscriptions," *Philosophical Studies* 32(1977)349–53.
J.N. Findlay, *Meinong's Theory of Objects and Values*, 2nd ed. (Oxford: Clarendon Press, 1963).
Gottlob Frege, "Function and Concept" (1891), trans. P.T. Geach, in P. Geach and M. Black (eds.), *Translations from the Philosophical Writings of Gottlob Frege* (Oxford: Basil Blackwell, 1970): 21–41.
Gottlob Frege, "On Concept and Object" (1982a), trans. P.T. Geach, in P. Geach and M. Black (eds.), *Translations from the Philosophical Writings of Gottlob Frege* (Oxford: Basil Blackwell, 1970): 42–55.
Gottlob Frege, "On Sense and Reference" (1892b), trans. M. Black, in P. Geach and M. Black (eds.), *Translations from the Philosophical Writings of Gottlob Frege* (Oxford: Basil Blackwell, 1970): 56–78.
Gary G. Hendrix, "Encoding Knowledge in Partitioned Networks," in N.V. Findler (ed.), *Associative Networks* (New York: Academic Press, 1979): 51–92.
Jaakko Hintikka, *Knowledge and Belief* (Ithaca, NY: Cornell University Press, 1962).
Jaakko Hintikka, "Indexicals, Possible Worlds, and Epistemic Logic," *Noûs* 1(1967)33–62.
Aravind Joshi, Bonnie Webber, and Ralph M. Weischedel, "Preventing False Inferences," *Proc. 10th International Conference on Computational Linguistics* (COLING-84) (Morristown, NJ: Assoc. for Computational Linguistics, 1984): 134–38.
Immanuel Kant, *Critique of Pure Reason*, 2nd ed. (1787), trans. N. Kemp Smith (New York: St. Martin's Press, 1929).
Kurt Konolige, "Belief and Incompleteness," Center for the Study of Language and Information Report CSLI-84-4 (Stanford: Stanford University, 1984).
George Lakoff, "Counterparts, or the Problem of Reference in Transformational Grammar" (Bloomington: Indiana University Linguistics Club, 1968).
David Lewis, "Counterpart Theory and Quantified Modal Logic," *Journal of Philosophy* 65(1968)113–26.
David Lewis, "Attitudes *De Dicto* and *De Se*," *Philosophical Review* 88(1979)513–43.
Anthony S. Maida, "On a Humanly Understandable Knowledge Representation for Reasoning about Knowledge," unpublished ms. (October 1983).
Anthony S. Maida and Stuart C. Shapiro, "Intensional Concepts in Propositional Semantic Networks," *Cognitive Science* 6(1982)291–330.
H. Marburger, K. Morik, and B. Nebel, "The Dialog System HAM-ANS: Natural Language Access to Diverse Application Systems," lecture presented at CSLI, Stanford University (29 June 1984); abstract in *CSLI Newsletter* 38(28 June 1984)1–2.
Joao Martins, *Reasoning in Multiple Belief Spaces* Doctoral dissertation, SUNY Buffalo Dept. of Computer Science Technical Report 203 (1983).[30]
Gordon McCalla and Nick Cercone, "Guest Editors' Introduction: Approaches to Knowledge Representation," *Computer* 16(October 1983)12–18.
J. McCarthy, "First Order Theories of Individual Concepts and Propositions," in J.E. Hayes, D. Michie, and L. Mikulich (eds.), *Machine Intelligence* 9(Chichester, Eng.: Ellis Horwood, 1979): 129–47.
Alexius Meinong, "Ueber Gegenstandstheorie" (1904), in R. Haller (ed.), *Alexius Meinong Gesamtausgabe*, Vol. II (Graz, Austria: Akademische Druck- u. Verlagsanstalt, 1971): 481–535. English translation ("The Theory of Objects") by I. Levi *et al.*, in R.M. Chisholm (ed.), *Realism and the Background of Phenomenology* (New York: Free Press, 1960): 76–117.
Marvin L. Minsky, "Matter, Mind, and Models," in M. Minsky (ed.), *Semantic Information Processing* (Cambridge, MA: MIT Press, 1968): 425–32.
Richard Montague, *Formal Philosophy*, ed. R.H. Thomason (New Haven: Yale University Press, 1974).
Robert C. Moore, "Reasoning about Knowledge and Action," *Proc. 5th International Joint Conference on Artificial Intelligence* (IJCAI-77) (1977): 223–27.
Nils J. Nillson, "Artificial Intelligence Prepares for 2001," *AI Magazine* 4.4(Winter 1983)7–14.
Terence Parsons, *Nonexistent Objects* (New Haven: Yale University Press, 1980).
John Perry, "The Problem of the Essential Indexical," *Noûs* 13(1979)3–21.
John Perry, "Castañeda on He and I," in J.E. Tomberlin (ed.) *Agent, Language, and the Structure of the World* (Indianapolis: Hackett, 1983): 15–42.
John Perry, "Perception, Action, and the Structure of Believing," in R. Grandy and R. Warner (eds.), festschrift for Paul Grice (forthcoming).[31]
William J. Rapaport, "On *Cogito* Propositions," *Philosophical Studies* 29(1976a)63–68.
William J. Rapaport, *Intentionality and the Structure of Existence.* Ph.D. dissertation, Indiana University, 1976b.
William J. Rapaport, "Meinongian Theories and a Russellian Paradox," *Noûs* 12(1978)153–80; errata, *Noûs* 13(1979)125.
William J. Rapaport, "An Adverbial Meinongian Theory," *Analysis* 39(1979)75–8l.

[30]Note added 2006: Published as: Martins, João, & Shapiro, Stuart C. (1988), "A Model for Belief Revision," *Artificial Intelligence* 35: 25–79.
[31]Now published in Grandy & Warner (eds.) *Philosophical Grounds of Rationality* (Oxford: Oxford University Press): 330–359.

William J. Rapaport, “How to Make the World Fit Our Language: An Essay in Meinongian Semantics,” *Grazer Philosophische Studien* 14(1981)1–21.

William J. Rapaport, “Meinong, Defective Objects, and (Psycho-)Logical Paradox,” *Grazer Philosophische Studien* 18(1982)17–39.

William J. Rapaport, Critical Notice of Routley 1979, *Philosophy and Phenomenological Research* 44(1984)539–52.

William J. Rapaport, “To Be and Not to Be,” *Noûs* (1985).

Elaine Rich, “User Modeling via Stereotypes,” *Cognitive Science* 3(1979)329–54.

Richard Routley, *Exploring Meinong’s Jungle and Beyond* (Canberra: Australian National University, Research School of Social Sciences, Dept. of Philosophy, 1979).

Roger C. Schank and Christopher K. Riesbeck (eds.), *Inside Computer Understanding* (Hillsdale, NJ: Lawrence Erlbaum Associates, 1981).

Stuart C. Shapiro, “The SNePS Semantic Network Processing System,” in N.V. Findler (ed.), *Associative Networks* (New York: Academic Press, 1979): 179–203.

Stuart C. Shapiro, “What do Semantic Network Nodes Represent?” Paper presented at Conference on Foundational Threads in Natural Language Processing, SUNY Stony Brook (27–29 July 1981). Semantic Network Research Group Technical Note No. 7 (Buffalo: SUNY Buffalo Dept. of Computer Science).

Stuart C. Shapiro, “Generalized Augmented Transition Network Grammars For Generation From Semantic Networks,” *American Journal of Computational Linguistics* 8(1982)12–25.

Stuart C. Shapiro and the SNePS Implementation Group, “SNePS User’s Manual” (Buffalo: SUNY Buffalo Dept. of Computer Science, 1983).

Robert C. Stalnaker, “Indexical Belief,” *Synthese* 49(1981)129–51.

Wolfgang Wahlster, “User Models in Dialog Systems,” lecture presented at the 10th International Conference on Computational Linguistics (COLING-84), Stanford University (2 July 1984).

R. Wilensky, Y. Arens, and D. Chin, “Talking to UNIX in English: An Overview of UC,” *Communications of the Assoc. for Computing Machinery* 27(1984)574–93.

Yorick Wilks and Janusz Bien, “Beliefs, Points of View, and Multiple Environments,” *Cognitive Science* 7(1983)95–119.

William A. Woods, “What’s in a Link: The Semantics of Semantic Networks,” in D.G. Bobrow and A.M. Collins (eds.), *Representation and Understanding* (New York: Academic Press, 1975): 35–79.

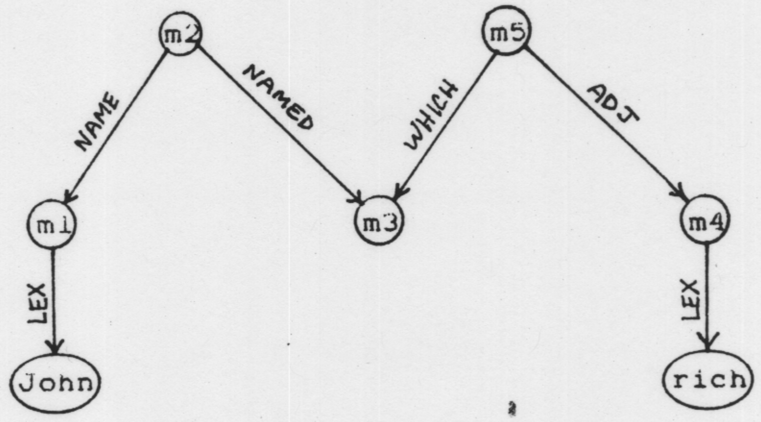

Figure 1. SNePS network for 'John is rich'.

The LEX arc points from a node to a tag used by the ATN parser-generator to attach a word to the node. (The node labeled 'rich', however, could also be considered to be the system's concept of the *word* 'rich'. Cf. Maida and Shapiro 1982: 303 for details on the LEX arc.)

The NAME-NAMED case frame is used to identify objecta by name. So, the non-dominated node m2 represents the system's belief that something (viz., whatever is represented by node m3) is named by the name whose concept is m1.

More idiomatically, (18) has been analyzed as

(18A) Something is named 'John' and it is rich.

or, perhaps,

(18B) $\exists x$[Name(x, 'John') & Rich(x)],

where the quantifier ranges over objecta in the system's belief space.

There is nothing sacrosanct about this analysis. We could just as well have represented (18) by the network of Figure 2. Here, m7 represents the system's belief that

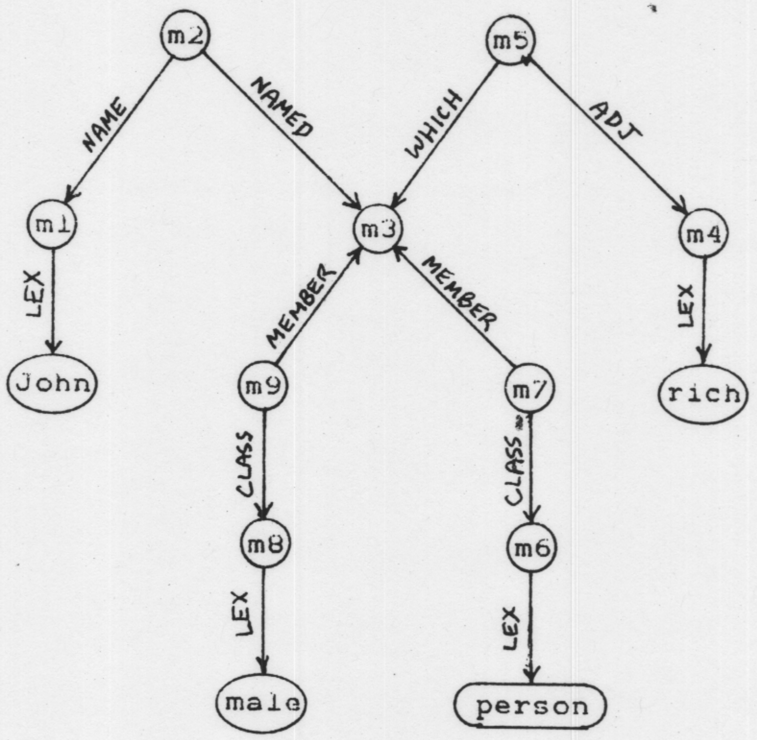

Figure 2. Another SNePS network for 'John is rich'.

the entity represented by m3 is a person, and m9 represents the system's belief that the entity represented by m3 is male.

Clearly, the amount of detail one wishes to build into the network analysis depends on one's ontological and linguistic theories. Rather than worry about such details here, I shall use the analysis of Figure 1, since it contains the essential information for my purposes.

To represent the system's beliefs about the beliefs of others, an AGENT-VERB-OBJECT case frame is used. Thus

(19) John believes that *p*

will be represented by the network fragment of Figure 3. Here, m6 represents the system's belief that the person whom the system believes to be named 'John' believes m5, where m5—in a concrete case—will be a node representing, *roughly*, the system's *de dicto* report of John's belief.

I say 'roughly', because, in fact, the system can never represent John's belief *exactly;* it can only represent John's belief using its *own* concepts. I shall return to this point shortly (cf. Sect. II.1.4, above, and Sects. 4.2, 4.3, below), but it should be noted that humans have precisely the same limitations.

4. SNePS Representations of Beliefs.

I can now be a bit more precise. However, to make things easier for the reader, I shall not present a general scheme for representation, but only representations of actual—though representative—sentences.

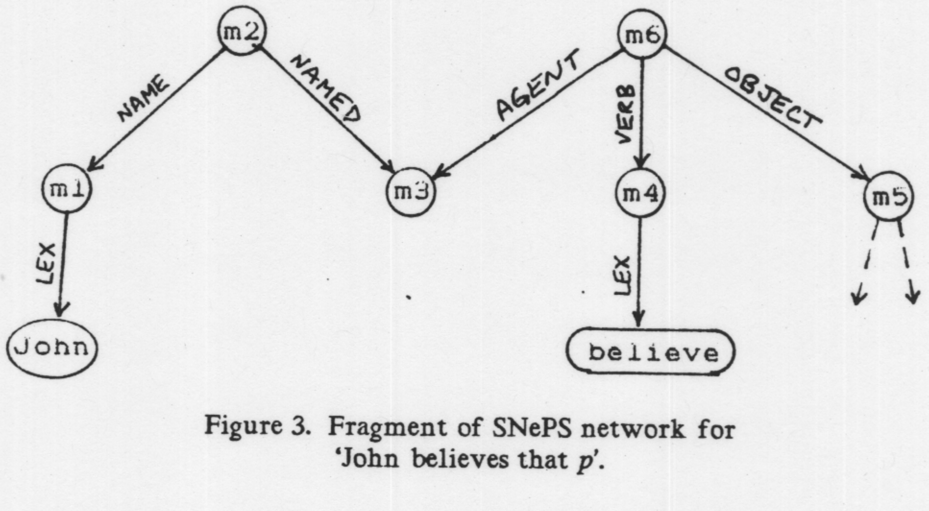

Figure 3. Fragment of SNePS network for 'John believes that *p*'.

4.1. De Re and De Dicto Beliefs. The *de dicto* belief report

(20) John believes that Lucy is sweet

will be represented by the network of Figure 4. A predicate-logic analysis of (20) might be:[12]

(20A) ∃*x*[Name(*x*, 'John') & Believes(*x*, ∃*y*[Name(*y*, 'Lucy') & Sweet(*y*)]].

That is, the system believes three things: that someone is named 'John', that he believes that someone is named 'Lucy', and that he believes that she (i.e., the person he believes to be named 'Lucy') is sweet.

To simplify the graphical notation, which will quickly become complex, Figure 4 can also be drawn as in Figure 5. The idea here is that, since m8 and m11 are both nodes in an

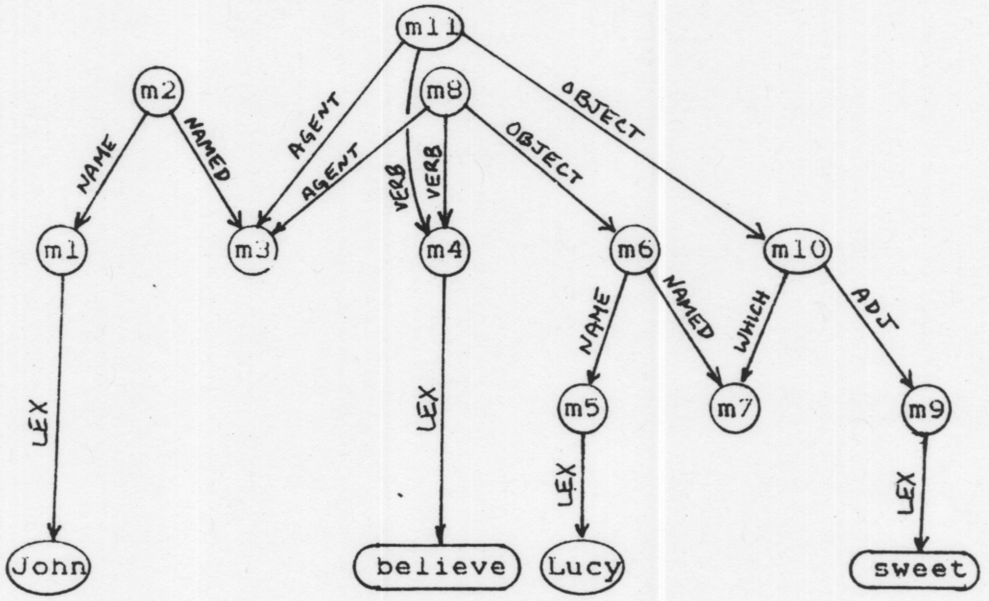

Figure 4. A SNePS network for the *de dicto* belief report 'John believes that Lucy is sweet'.

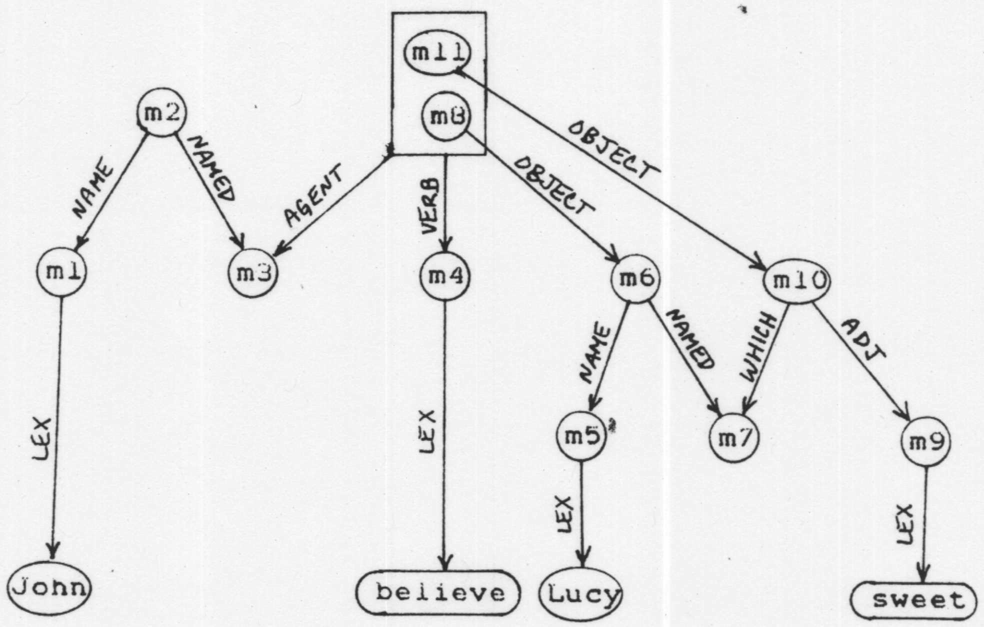

Figure 5. An alternative graphical representation of the network in Figure 4.

AGENT-VERB-OBJECT case frame with the same AGENT and the same VERB, we can eliminate the redundant arcs from the *graphical representation* of the network by using the box notation. *This is a notational convenience only.*

A *de re* belief report,

(21) John believes of Lucy that she is sweet,

will be represented by the network of Figure 6. A predicate-logic analysis of (21) might be:

(21A) $\exists x \exists y$[Name(x, 'John') & Name(y, 'Lucy') & Believes(x, Sweet(y))]

That is, the system believes three things: that someone is named 'John', that someone is named 'Lucy', and that he (John) believes of her (Lucy) that she is sweet. Note that here the system has no beliefs about how John represents Lucy.[13]

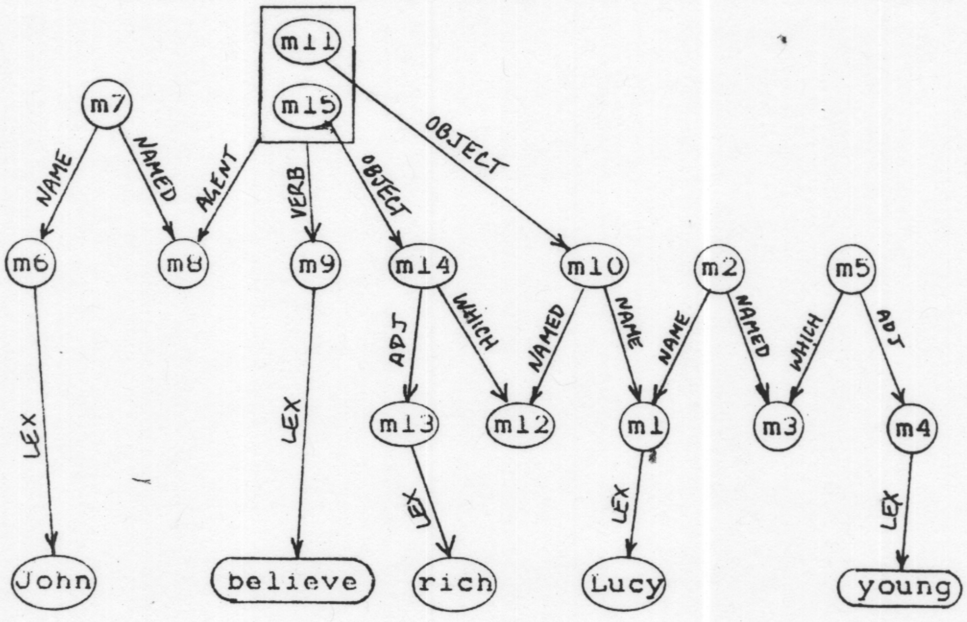

Figure 7. SNePS network representing that Lucy is young and that John believes that someone named 'Lucy' is rich.

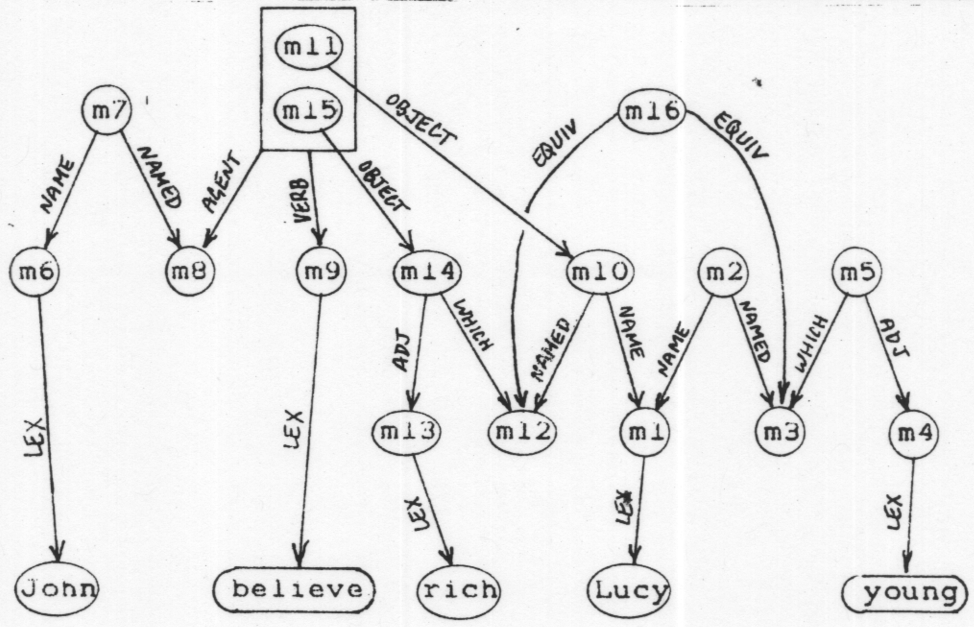

Figure 8. SNePS network representing that Lucy is young, that John believes that someone named 'Lucy' is rich, and that John's Lucy is the system's Lucy.

4.2. Nested Beliefs. The representational scheme presented here can also handle sentences involving iterated belief contexts. Consider

(22) Bill believes that Stu believes that Hector is philosophical.

The interpretation of this that I am most interested in representing treats (22) as the system's *de dicto* representation of Bill's *de dicto* representation of Stu's *de dicto* belief that Hector is philosophical. On this interpretation, we need to represent the *system's* Bill, the system's representation of *Bill's* Stu, and the system's representation of Bill's representation of *Stu's* Hector. In the implementation, these are represented, respectively, by: (Bill system), (Stu Bill system), and (Hector Stu Bill system). Sentence (22) is represented by the network of Figure 9.

In the implementation (cf. Sect. VIII and the Appendix), such a network is built recursively as follows: The parser maintains a stack of "believers"; the first element on the

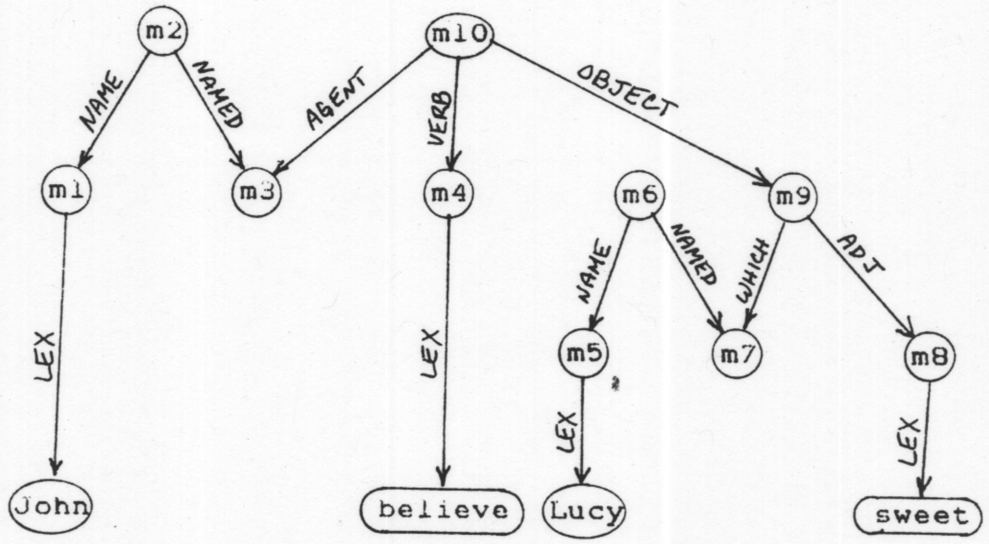

Figure 6. A SNePS representation of the *de re* belief report 'John believes of Lucy that she is sweet'.

We can also combine these. But here we must be careful. In the absence of prior knowledge of co-referentiality, the entities *within* a belief context should be represented *separately* from entities *outside* the context that might be co-referential with them.

Suppose that the system's beliefs include that a person named 'Lucy' is young and that John believes that a (possibly different) person named 'Lucy' is rich. This is represented by the network of Figure 7. The section of network dominated by nodes m10 and m14 is the system's *de dicto* representation of John's belief. That is, m14 is the *system's* representation *of* a belief that *John* would express by 'Lucy is rich', and it is represented *as* one of John's beliefs. Such nodes are considered as being in the system's representation of John's "belief space". If it is later determined that the "two" Lucies are the same, then a node of co-referentiality would be added, as in Figure 8 (node m16; cf. Maida and Shapiro 1982: 303-04 and Sect. IX.2.1, below, for a discussion of the EQUIV case frame).

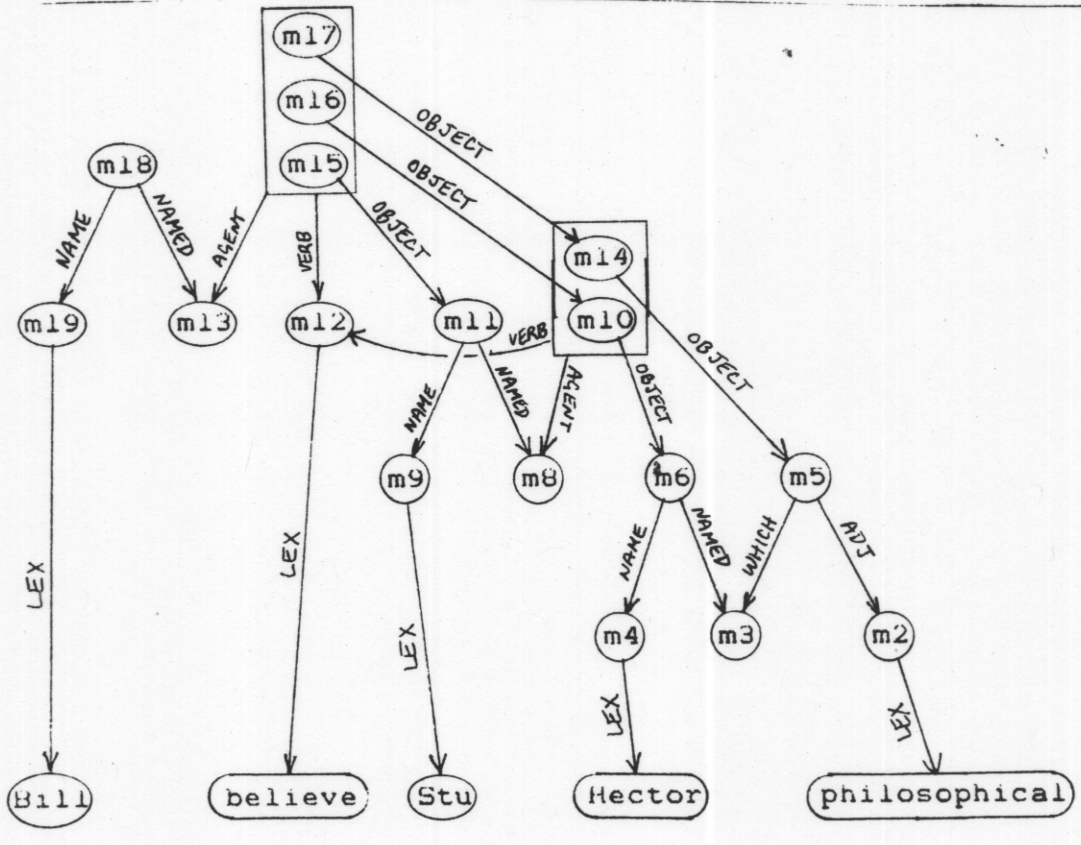

Figure 9. SNePS network for 'Bill believes that Stu believes that Hector is philosophical'.

stack is the system. Each time a belief sentence is parsed, it is made the object of a belief of the previous believer in the stack. Structures are shared wherever possible. This is accomplished by use of a new SNePS User Language function, *forb-in-context* (find-or-build in a belief context), which, given a description of a node and a belief context (a stack of believers), either finds the node in that context if it exists there (i.e., if it is already in the believer's belief space, or else builds it in that context (i.e., represents the fact that it is now in the believer's belief space). Thus,

(23) Bill believes that Stu believes that Hector is Guatemalan

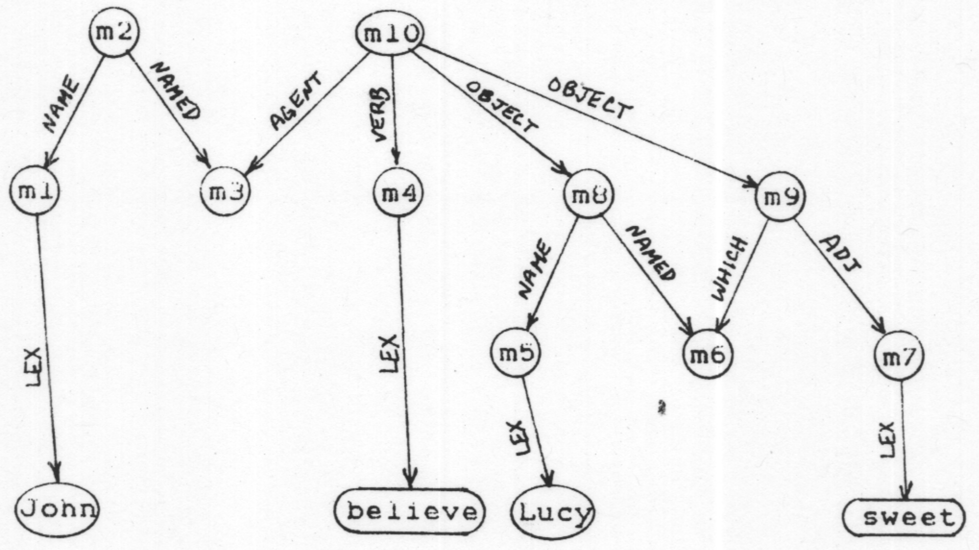

Figure 10. Alternative SNePS network for 'John believes that Lucy is sweet'.

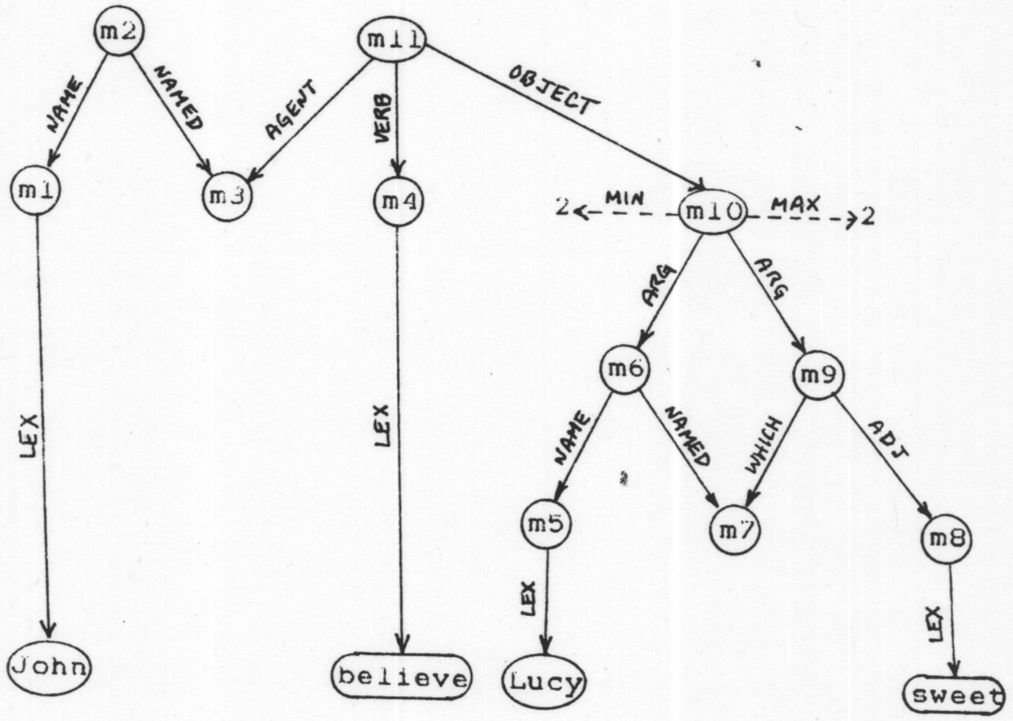

Figure 11. Another alternative SNePS network for 'John believes that Lucy is sweet'.

In Figure 11, node m10 is the SNePS representation of the conjunction of m6 and m9. One objection to this format is that the nodes in *John's* belief space are explicitly conjoined, unlike the nodes in the *system's* belief space; there seems no good reason for this structural dissimilarity. Moreover, retrieval of the information that John believes that someone is sweet would be difficult, since that information is not explicitly represented in Figure 11.

4.4. De Se Beliefs.

4.4.1. The representation. To adequately represent *de dicto* reports of *de se* beliefs, we need the strategy of separating entities in different belief spaces (see Sect. 4.1).

Consider the possible representation of

(3) John believes that he* is rich

shown in Figure 12 (adapted from Maida and Shapiro 1982: 316).

This suffers from three major problems. First, it is ambiguous: It could be the representation of (3) or of

(24) John believes that John is rich.

But, as we have seen, (3) and (24) express quite different propositions; thus, they should be separate items in the data base.

Second, Figure 12 cannot represent (24). For then we would have no easy or uniform way to represent (3) in the case where John does not know that he is named 'John': Figure 12 says that the person (m3) who is named 'John' and who believes m6, believes that that person is rich; and this would be false in the amnesia case.

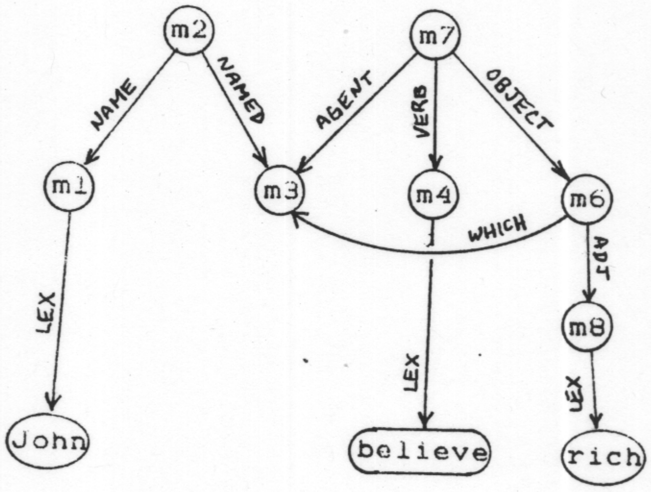

Figure 12. A possible SNePS network for 'John believes that he* is rich'.

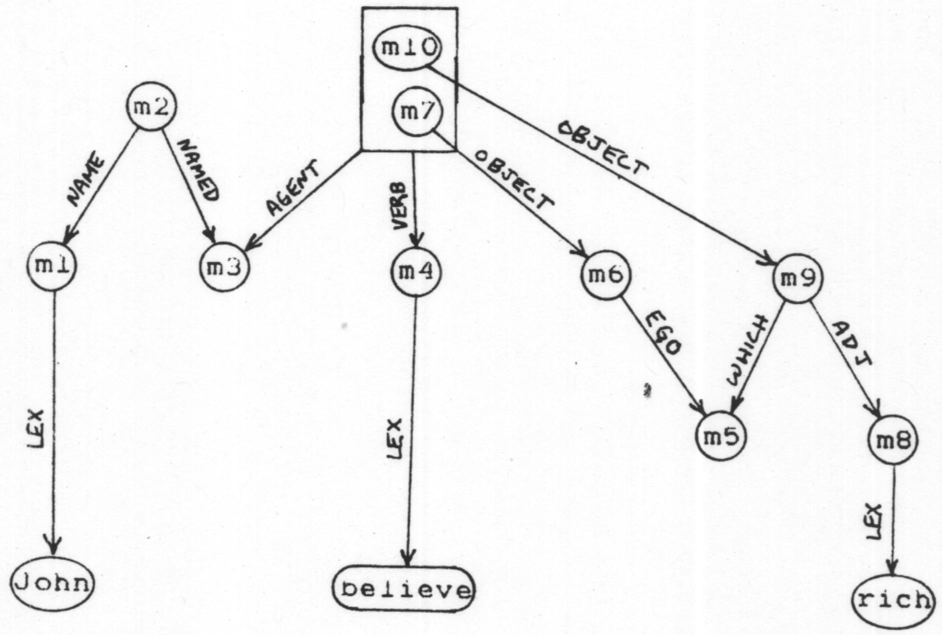

Figure 13. SNePS network for 'John believes that he* is rich'. (Node m5 is the system's representation of John's "self concept", expressed by John as 'I' and by the system as 'he*').

As a final example, consider the following:

(26) John believes that he* is not rich.

(8) John believes that the editor of *Byte* is rich.

(27) John is the editor of *Byte*.

(28) John believes of himself that he is rich.

If John is unaware of the truth of (27), then the system ought to be able to infer—and, hence, to represent—(28). We can represent the data base resulting from the input of this sequence of information and inference by the network of Figure 14. Here, nodes m11 and m7 represent (26) (node m10 represents a SNePS negation of node m9); nodes m15 and m17 represent (8); m18 represents (27); and m20 represents (28).

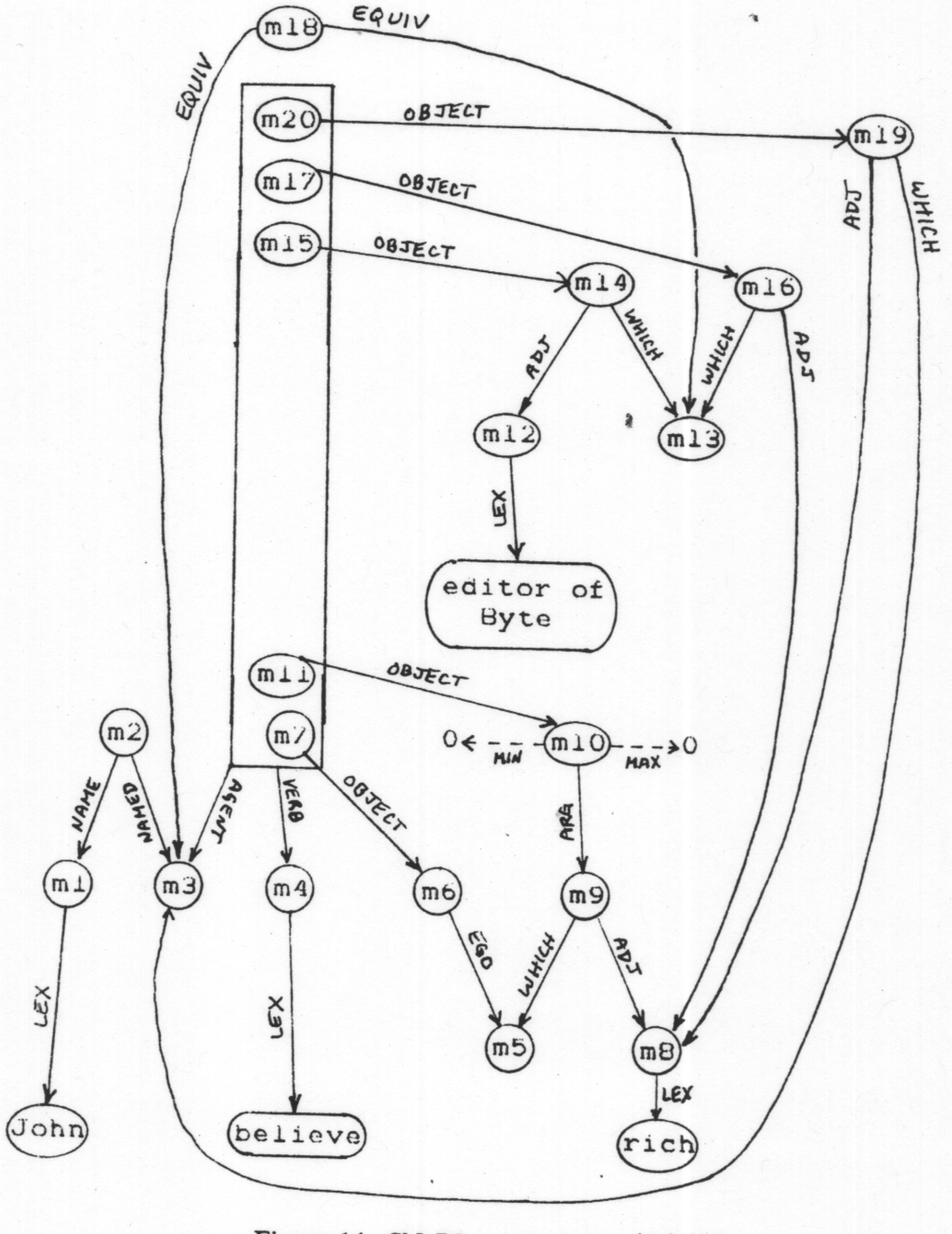

Figure 14. SNePS network for (26)-(28)

4.4.2. The EGO arc. One representational issue requires discussion: the EGO arc. The node pointed to by the EGO arc is the system's representation of what might be called John's *self concept*, i.e., John's "model" of himself (cf. Minsky 1968).

Again, let us consider alternative representations. The network of Figure 15, which is the simplest alternative, uses a COGNITIVE-AGENT/SELF-CONCEPT case frame to "identify" m5 as being m3's (John's) self concept (a kind of "merger" of m5 with m3). But m5 needs to be *inside John's* belief space (to be on a par with Figure 13), which this network does not do. Note that in Figure 13, node m7 also links John (m3) with his self concept (m5),[14] as does Figure 15's m9, but it does so while placing m5 within John's belief space.

Another alternative is the network of Figure 16. Here, we do have m5 within John's belief space, but we also have John there—*as well as* outside it—which violates the principle of the separation of belief spaces. And, again, any viable role played by such a case frame can also be played by node m7 of Figure 13.

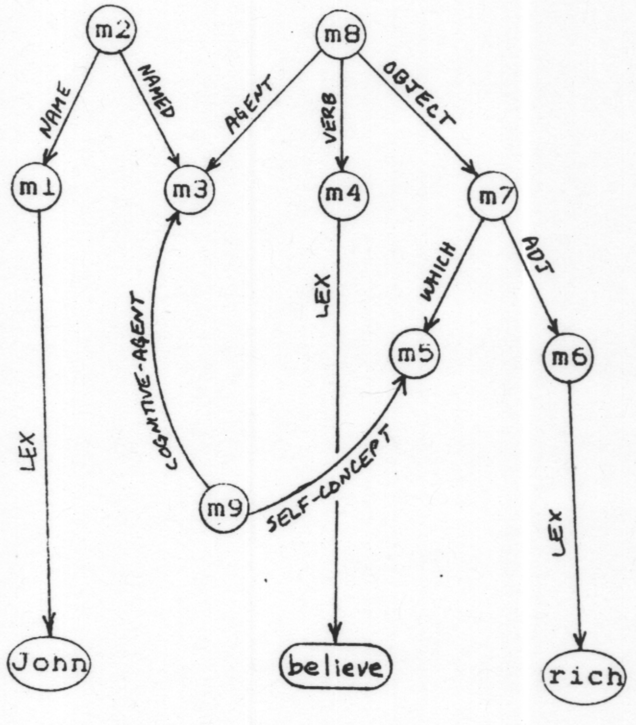

Figure 15. Alternative SNePS network for
'John believes that he* is rich'.

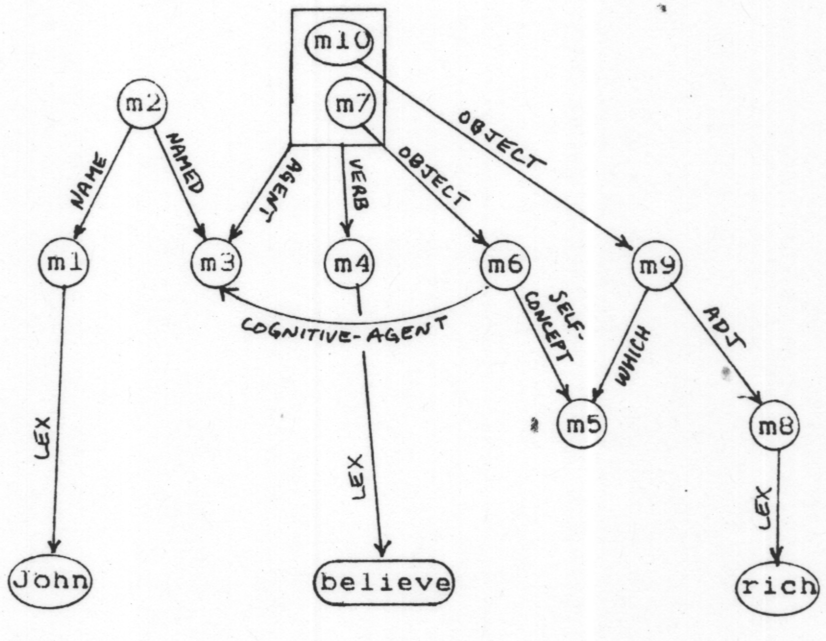

Figure 16. Another alternative SNePS network for
'John believes that he* is rich'.

Finally, consider the network of Figure 17. Here, the idea is to have a case frame analogous to the NAME-NAMED one, which might be called the QI-EGO case frame. Node m7 represents the proposition (believed by John) that m5 is he himself. But, of course, that is *not* the proposition believed by John. Rather, he believes a proposition that *he* would express as 'm5 is me'. And that is precisely what the original EGO arc of Figure 13 is intended to capture.

At least two philosophers have suggested analyses of *de dicto/de se* beliefs that are structurally similar to the lone EGO arc. Chisholm would say that such a belief report conveys the information that John has an "individual essence" m5 and that he believes that whatever has m5 is rich (cf. Chisholm 1977: 169). And Perry introduces an *ego function* that maps a person to that person's "special sense"; a Perry-style analysis of our *de dicto/de se* belief would be:

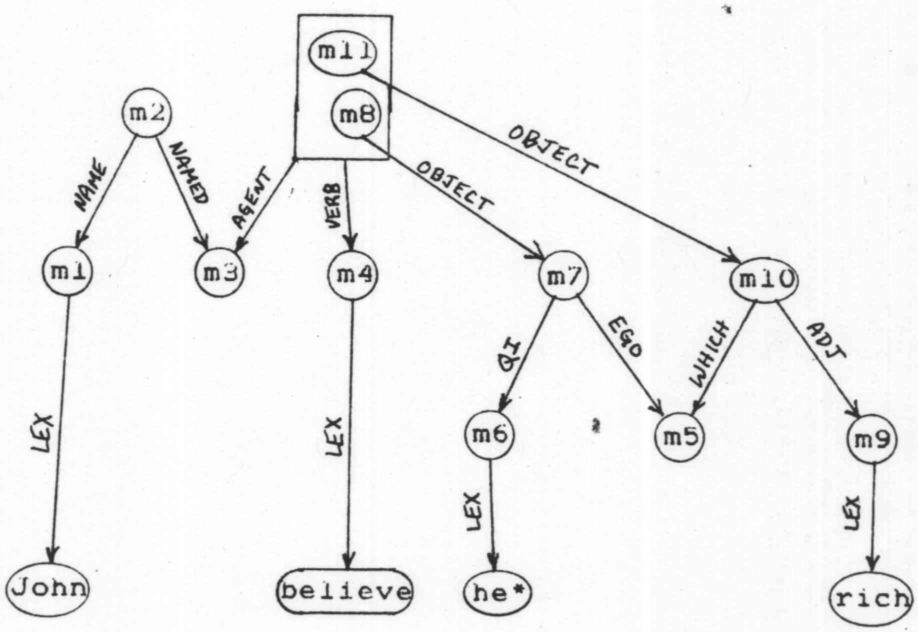

Figure 17. A third alternative SNePS network for 'John believes that he* is rich'.

$\exists s[s = \text{ego}(\text{John})\ \&\ \text{Believes}(\text{John}, \text{Rich}(s))]$

(cf. Perry 1983: 19, 25).

There are difficulties with both of these suggestions, which I shall not go into here (cf., e.g., Castañeda 1983: 326f), and there are also more complicated cases that need to be examined. But these are topics for future investigation (see Sect. IX.3.2).

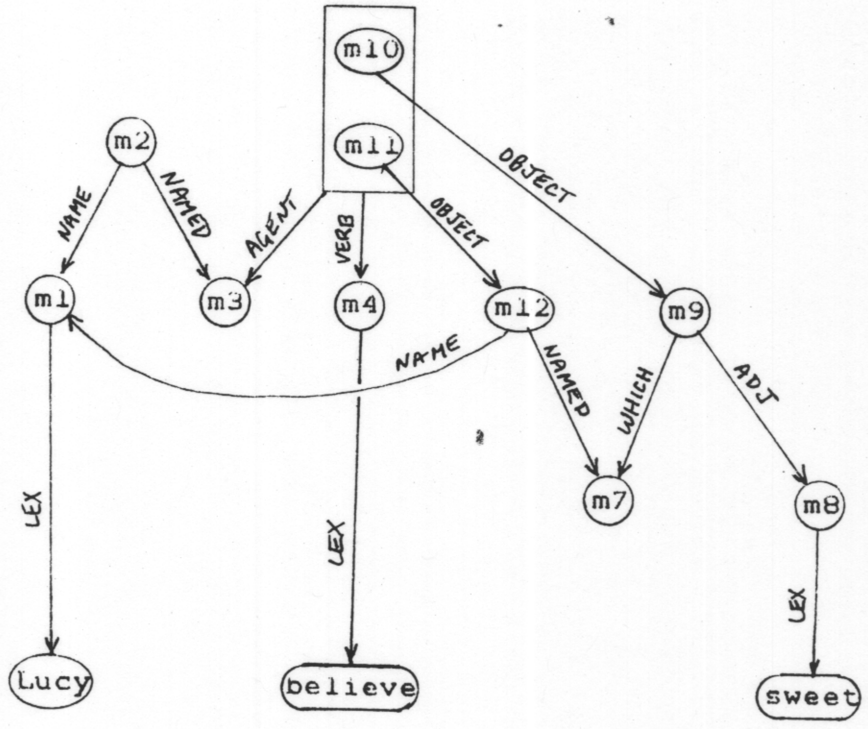

Figure 18. SNePS network at time t1, representing 'Lucy believes that Lucy is sweet'.

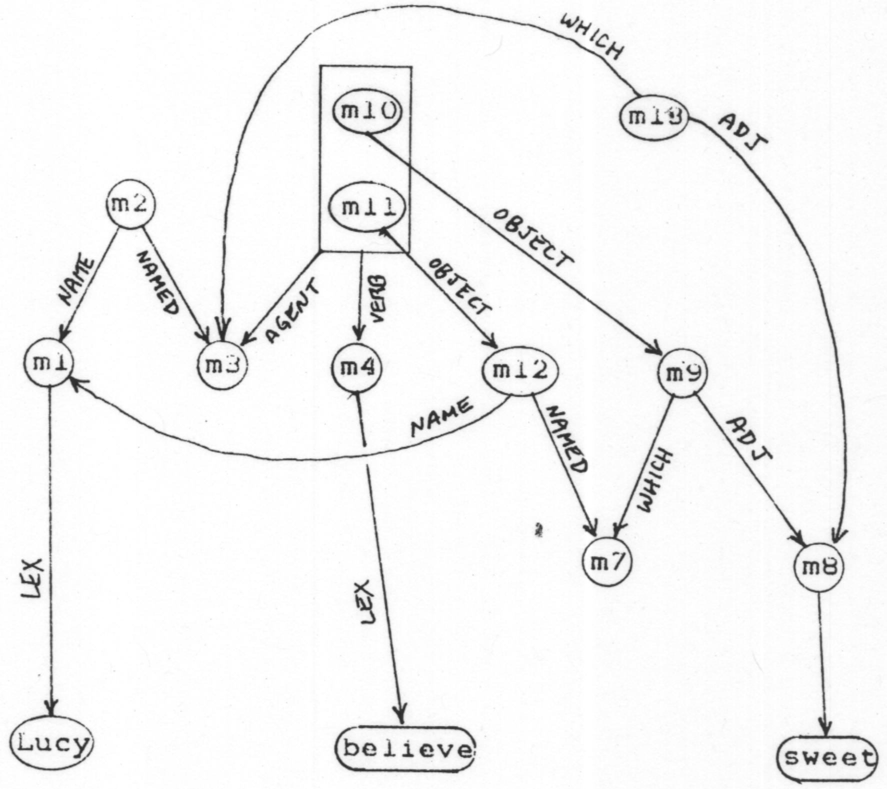

Figure 19. The SNePS network at time t2, modified from the network of Fig. 18, representing 'Lucy believes that Lucy is sweet', and that Lucy (the believer) is sweet.

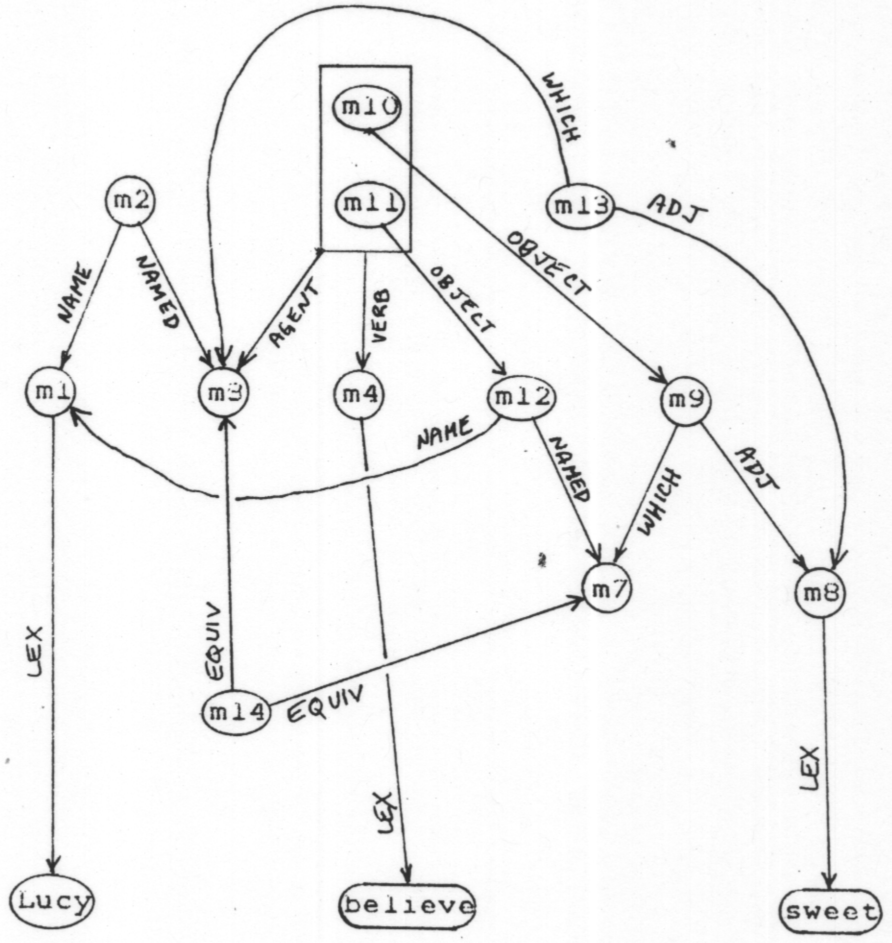

Figure 20. The SNePS network at time t3, modified from the network of Fig. 19, representing: 'Lucy believes that Lucy is sweet', that Lucy (the believer) is sweet, and that the system's Lucy is Lucy's Lucy.

intensional object that has an extensional referent (that has a Sein-correlate, in the terminology of Rapaport 1978) can have a lone EQUIV arc pointing to it. Suppose that the system is told, incorrectly, that the Morning Star is Mercury (see Figure 21). If it is later told that this is false, Figure 21 would be modified to Figure 22.

But this not only negates m7, it also negates the still-true fact that m3 is self-EQUIV. This may be a problem merely for the EQUIV arc, which could be obviated by using counterparts or consociation. Or it might be a more serious problem. It needs to be resolved.

2.2. The EGO Arc. I have likened the EGO arc to Perry's ego function (cf. Sect. V.4.4.2). But Castaneda has raised some objections to the ego function:

Perry constructs the following rule:

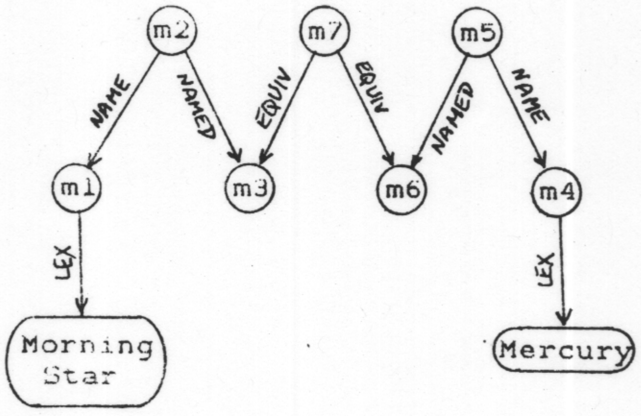

Figure 21. SNePS network for 'The Morning Star is Mercury'.

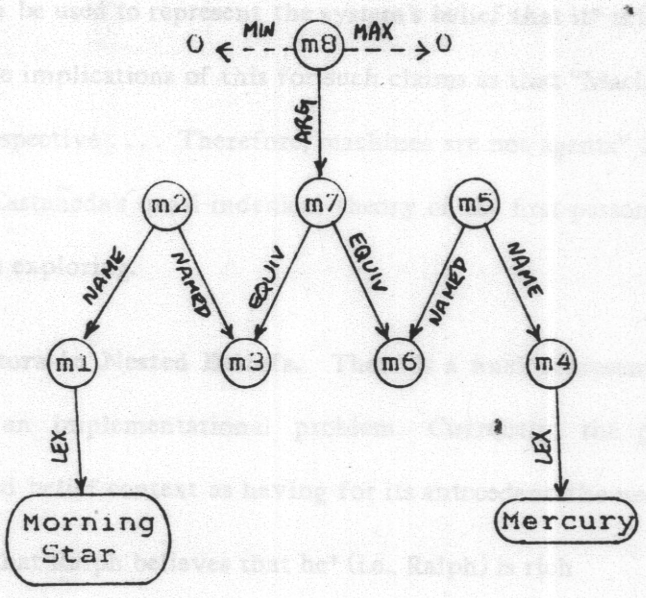

Figure 22. SNePS network for 'The Morning Star is not Mercury'.

He*.2: With t as antecedent,

[H*] *A believes that he* is so-and-so*

has the sense of

[S.E] *There is a sense s such that s = ego(t) and A believes that [s] is so and so.*

> [T]his rule is incorrect . . . . The whole point of the quasi-indicator 'he*' in [H*] is to depict the fact that the subject of the proposition that is the accusative of what the person A believes is ego(des(t)) [i.e., the ego of the designation of t]; obviously this is *not* represented in [S.E]. . . . Hence, the two sentences must have different senses. (Castañeda 1983: 326-27.)

It is not clear whether the EGO arc is infested with this difficulty, or, if it is, whether it is as serious for our representational scheme as Castañeda feels it is for Perry.

Assuming that the EGO arc, or something like it, has no special philosophical problems, it has some potential further applications in the representation of self-knowledge. For instance, it can be used to represent the system's belief that it* is (or is not!) a computer, as in Figure 23. The implications of this for such claims as that "Machines lack an irreducible first-person perspective .... Therefore, machines are not agents" (Baker 1981: 157), especially in light of Castañeda's quasi-indexical theory of the first-person (Castañeda 1968), are exciting and worth exploring.

2.3. Quasi-Indicators in Nested Beliefs. There is a final representational issue that is, perhaps, more of an implementational problem. Currently, the parser treats a quasi-indicator in a nested belief context as having for its antecedent the *nearest* believer. Thus,

John believes that Ralph believes that he* (i.e., Ralph) is rich

can be parsed correctly, but

John believes that Ralph believes that he* (i.e., John) is rich

would be parsed the same way. Here, the 'he*'s *are* quasi-indicators, and, so, the

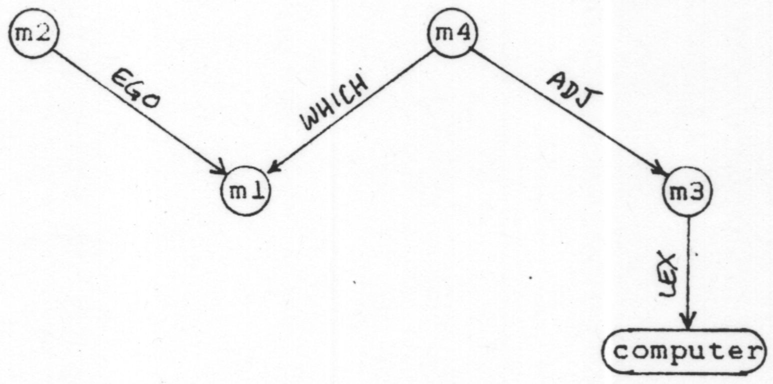

Figure 23. SNePS network for the system's belief that it* is a computer.