SITUATED MODELING OF EPISTEMIC PUZZLES

Murat Ersan and Varol Akman

BILKENT UNIVERSITY

Department of Computer Engineering and Information Science

Technical Report BU-CEIS-94-17

Situated Modeling of Epistemic Puzzles

Murat Ersan and Varol Akman
Department of Computer Engineering and Information Science
Bilkent University, Bilkent, 06533 Ankara, Turkey
Fax: +90 (312) 266–4126
Email: {ersan,akman}@bilkent.edu.tr

Situation theory is a mathematical theory of meaning introduced by Jon Barwise and John Perry. It has evoked great theoretical and practical interest and motivated the framework of a few 'computational' systems. Unfortunately, there is a lack of 'real life' applications on these systems and this paper is a preliminary attempt to remedy this situation. Here, we solve a class of epistemic puzzles, introduced by Raymond Smullyan, using the situation-theoretic programming language PROSIT.

1 Introduction

Situation theory is a principled programme to develop a mathematical theory of meaning which aims to clarify and resolve some tough problems in the study of language, information, logic, and philosophy [3]. It was introduced by Jon Barwise and John Perry in the Eighties and has stimulated great interest [4]. The theory matured within the last ten years or so [1, 7, 8, 15, 23, 24] and various versions of it have been applied to a number of linguistic issues [9]. This was followed by assorted studies on the computational aspects of the theory, which gave birth to a group of computational systems based on situation theory [5, 6, 12, 13, 14, 20, 21, 22].

PROSIT (PROgramming in SItuation Theory), developed by Nakashima et al. [12, 13, 14], is *the* pioneering work in this direction. PROSIT seems to be especially suitable for writing programs simulating human-like (commonsense) reasoning [10, 11]. Unfortunately, there have been very few attempts to employ PROSIT in this style. Such a study is, however, of great importance, and would help us see where and why we should utilize systems based on situation theory, and how we should formulate a situation-theoretic programming paradigm. In fact, as far as we know the only remarkable application in which PROSIT has been effectively exploited is the "Three Wisemen Problem" [12]. This is a problem involving common knowledge in a multi-agent setting. Pinning our faith upon situation theory, we tried to make use of PROSIT in the solution of what we came to call 'epistemic puzzles.' These puzzles were introduced by Raymond Smullyan in his book *Forever Undecided: A Puzzle Guide to Gödel* [16]. (Also cf. [17, 18, 19] for similar puzzles.) Here, the main goal is to distinguish the liars and truth-tellers living on an island (using the statements they make). Throughout this paper the nature of epistemic puzzles and how they are handled via a situation-theoretic world-view will be analyzed. A short introduction to PROSIT will be offered at the outset to motivate the presentation in the sequel. (This can be skipped by readers familiar with PROSIT.)

2 Situation Theory

The three major concepts of situation theory [1, 4, 8, 9, 10] are infons, situations, and constraints. Infons are the basic informational units and are denoted as $\ll P, a_1, \ldots a_n, i \gg$ where P is an n-place relation, $a_1, \ldots a_n$ are objects appropriate for the respective argument places of P, and i is the polarity (0 or 1). It is possible to use spatial and temporal locations in the argument places of relations.

Situations are 'first-class' citizens of the theory. There is no clear definition of what a situation exactly is. Rather, a situation is considered to be a structured part of the Reality that the agent somehow manages to pick out (individuate). The only definition given at this level is that of the *supports* relation:

> s *supports* α ($s \models \alpha$) means that α is an item of information that is true of s.

It is desirable to have some computational tools to handle situations. Abstract situations are the mathematical constructs with which we can abstract analogs of real situations. They are more amenable to mathematical manipulation. An abstract situation is defined as a set of infons. Given a real situation s the set $\{\alpha \mid s \models \alpha\}$ is a corresponding abstract situation.

In situation theory, the flow of information is carried out by constraints. A situation s will carry information relative to the constraint $C = [S \Rightarrow S']$, if $s : S[f]$, where f anchors the parameters in S and S'. Hence, the information carried by s relative to C is that there is a situation s', possibly extending s, of type $S'[f]$.

An important feature of situation theory is the existence of types. Types are higher-order uniformities which cut across uniformities like individuals, relations, situations, and spatial and temporal locations. Just as individuals, temporal locations, spatial locations, relations, and situations, types are also (higher-order) uniformities that are discriminated by agents. In this framework, relations may have their argument places filled either with individuals, situations, locations, and other relations or with types of individuals, situations, locations, and relations. For example, if an agent sees smoke he can conclude that there is fire. For he is aware of the constraint which links situations where there is smoke to those where there is fire. This constraint is not particular to a certain instance, but holds in general. Actually the constraint links types of situations, viz. smoky-type of situations to ones with fire.

The development of types brings the requirement of devices for making reference to arbitrary objects of a given type. Therefore for each type T, an infinite collection of *parameters* $T_1, T_2, \ldots$ is introduced. For example IND_3 is an IND-parameter (parameters of type IND).

These parameters bring about some computational power, but we need more than that. Rather than parameters ranging over all individuals, we need parameters that range over a more restricted class, e.g. all men kicking footballs. Such parameters are called *restricted parameters*. For example,

$$\dot{r1} = \dot{a} \uparrow \ll kicking, \dot{a}, \dot{b}, 1 \gg$$
$$\dot{a} = IND3 \uparrow \ll man, IND3, 1 \gg$$
$$\dot{b} = IND2 \uparrow \ll football, IND2, 1 \gg$$

Once defined, $\dot{r1}$ ranges over all men kicking footballs.

In addition, it is possible to obtain new types using a parameter, s, and a set, I, of infons (in the form $[s \mid s \models I]$). For example,

$$[SIT \mid SIT \models \ll kicking, \dot{a}, \dot{b}, 1 \gg]$$

represents a situation-type where a man is kicking a football and

$$[\dot{a} \mid SIT \models \ll kicking, \dot{a}, \dot{b}, 1 \gg]$$

denotes the type of men kicking a football.

3 PROSIT

PROSIT is a declarative language in which both programs and data are just sets of declarative elements called *infons*. This feature makes PROSIT akin to Prolog, but PROSIT is based on situation theory rather than Horn clauses. The motivation behind the design of this new language rests on the following gross features, each of which is supported by the theory:

- The use of partially specified objects and partial information
- Situations as first-class citizens
- Informational constraints
- Self-referential expressions

In PROSIT, an infon (a discrete item of information) is represented as a list whose first element is the symbol for a relation and whose remaining elements are the objects for which the relation holds:

$$(relation\ object_1\ \ldots\ object_n)$$

For example, the infon

```
(listening-to John Mary)
```

states that the relation `listening-to` holds between the objects represented by the parameters `John` and `Mary`, i.e., John is listening to Mary.

One can assert infons and query a knowledge base incorporating, among other things, such as infons. Unlike Prolog, all infons are local to situations. For example, to assert the infon mentioned above in situation `sit1` the following expression is used:

```
(!= sit1 (listening-to John Mary))
```

Expressions in PROSIT are Lisp-style objects (i.e., atoms or lists). Atoms that are numbers or strings are considered to be *constants*. Symbols starting with a character other than "`*`" are *parameters*. They are used to represent things in the world, such as individuals, situations, and relations. Usually, different parameters correspond to different entities. Parameters can be used in any infon (including queries and constraints); their scope is global. A third kind of expression is a *variable*. Variables are represented with symbols starting with "`*`". They can only occur in queries and constraints. They can stand for any PROSIT expression, yet their scope is local to the constraint or query they participate in.

In PROSIT, there exists a tree hierarchy among all situations, where the situation `top` is at the root of the tree. `top` is the global situation and the 'owner' of all the other situations generated. One can traverse the 'situation tree' using the predicates `in` and `out`. Although it is possible to issue queries from any situation about any other situation, the result will depend on where the query is made. If a situation `sit2` is defined in the current situation, say `sit1`, then `sit1` is said to be the owner of `sit2`, or equivalently:

- `sit2` is a part of `sit1`, or
- `sit1` describes `sit2`

The owner relation states that if `(!= sit2 infon)` holds in `sit1`, then `infon` holds in `sit2`, and conversely, if `infon` holds in `sit2` then `(!= sit2 infon)` holds in `sit1`. `in` causes the interpreter to go to a specified situation which will be a part of the 'current situation' (the situation in which the predicate is called) and `out` causes the interpreter to go to the owner of the current situation.

Similar to the owner relation there is the 'subchunk' relation. It is denoted as`([_ sit1 sit2)`, where `sit1` is a subchunk of `sit2`, and conversely, `sit2` is a 'superchunk' of `sit1`. When a situation, say `sit1`, is asserted to be the subchunk of another situation, say `sit2`, it means that `sit1` is totally described by `sit2`. A superchunk is like an owner (except that `out` will always cause the interpreter to go to the owner, not to a superchunk).

PROSIT has two more relations that can be defined between situations. These are the 'subtype' and the 'subsituation' relations. When the subtype relation (denoted by `(@< sit1 sit2)`) is asserted, it causes the current situation to describe that `sit2` supports `i` for every infon `i` valid in `sit1` and that `sit2` respects every constraint that is respected by `sit1`, i.e., `sit1` becomes a subtype of `sit2`. The subsituation relation is denoted as

`(<-- sit1 sit2)` and is the same as `(@< sit1 sit2)` except that only infons, but no constraints, are inherited. Both relations are transitive.

A distinguishing feature of PROSIT is that the language allows circularity [2]. The fact that PROSIT permits situations as arguments to infons makes it possible to represent self-referential statements. Consider a card game (`sit`) where there are two players. John has the ace of spades and Mary has the queen of spades. When both players display their cards the following infons will be factual:

```
(!= sit (has John ace-of-spades))
(!= sit (has Mary queen-of-spades))
(!= sit (see John sit))
(!= sit (see Mary sit))
```

The notion of informational constraints is a distinguishing feature that encouraged the design of PROSIT. Constraints can be considered as a special type of information and 'generate' new facts. They are just a special case of infons, and therefore, are also situated. A constraint can be specified using either of the three relations `=>`, `<=`, and `<=>`. Constraints specified with `=>` are forward-chaining. They are of the form `(=>` $fact\ head_1\ head_2\ \ldots\ head_n$`)`. If *fact* is asserted to the situation then all of the head facts are also asserted to that situation. Constraints specified with `<=` are backward-chaining. They are of the form `(<=` $head\ fact_1\ fact_2\ \ldots\ fact_n$`)`. If each of the facts from 1 to n are supported by the situation, then the head fact is also supported (though not asserted) by the same situation. Finally, constraints specified with `<=>` should be considered as both backward- and forward-chaining.

Now, if there is a constraint stating that everything that smiles is happy in situation `sit1`, viz.,

```
(resp sit1 (=> (smiles *X) (happy *X)))
```

then the assertion of

```
(smiles John)
```

in `sit1` will force PROSIT to assert the following infon in `Sit1`, too:

```
(happy John)
```

4 Epistemic Puzzles

An epistemic puzzle is one which mainly involves knowledge and belief. This can be either in the form of individual knowledge or common knowledge (mutual information) [1] in a multi-agent setting (e.g., the Three Wisemen Problem, see presently). A computational

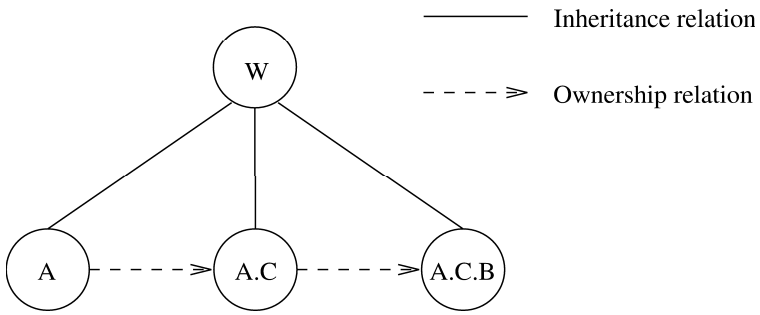

Figure 1: The Three Wisemen Problem. The facts that all wisemen know are kept in situation W. The facts that A knows are kept in situation A. The facts that A knows that C knows are kept in situation A.C. The facts that A knows that C knows that B knows are kept in situation A.C.B.

solution to such puzzles requires a system that is able to perform human-like reasoning [10, 11]. Including certain situation-theoretic constructs, PROSIT is highly eligible for implementing and solving these puzzles.

4.1 The Three Wisemen Problem

The solution of the "Three Wisemen Problem" [12] in PROSIT is, to our best knowledge, the only serious attempt to use situation-theoretic constructs in resolving epistemic puzzles. The main aim is to show how to use common knowledge computationally in solving problems involving cooperation of multiple agents. The situation-theoretic aspects of PROSIT (reasoning *about* situations and *in* situations) generated an intuitive and simple solution for this hypothetical problem, stated as follows [12, p. 79]:

> "Three wisemen are sitting at a table, facing each other, each with a white hat on his head. Someone tells them that each of them has a white or red hat but that there is at least one white hat. Each wiseman can see the others' hats but not his own. If a fourth person asks them whether they know their own color, then the first two wisemen will answer no, but, after that, the third one will answer yes."

The existing facts in the problem can be categorized into two groups: facts that all wisemen are aware of, and facts that are known individually. Facts such as that there are three agents A, B, and C, that all agents are wise, and that each agent is wearing either a white or a red hat are known by all three wisemen. On the other hand, the fact that B and C are wearing white hats is known only by A.

There are two ways for an agent to decide that his hat is white. The first is when the other two wisemen have red hats. The second is when his assumption of having a red hat causes a contradiction. The approach in [12] is to use the latter in order to solve this problem. A assumes that he has a red hat. After B and C answers no, A concludes that

C should have said yes (because from B's answer C concludes that at least one of A and C is wearing a white hat) if he were wearing a red hat. So he knows that he is wearing a white hat. The key observation used in the solution is the one causing the transfer of knowledge, i.e., if A knows something about B, then he also knows that C knows it, too! PROSIT's tree hierarchy of situations makes it rather easy to represent this (Figure 1).

4.2 Smullyan's Puzzles

In [16] Smullyan introduces a number of puzzles about liars and truth-tellers to warm up the readers with symbolic logic. Most of the events in the puzzles take place on an island, viz., the Island of Knights and Knaves. On this imaginary island the following three propositions hold:

1. Knights always make true statements.
2. Knaves always make false statements.
3. Every inhabitant is either a knight or a knave.

The puzzles are epistemic in the sense that knights 'reflect' their individual knowledge and beliefs while knaves 'reflect' the contrary of them. A simple puzzle of this type is the one about the census taker Mr. McGregor [16, pp. 15–16]:

> "The census taker Mr. McGregor once did some fieldwork on the Island of Knights and Knaves. On this island, women are also called knights and knaves. McGregor decided on this visit to interview married couples only. McGregor knocked on one door; the husband partly opened it and asked McGregor his business. "I am a census taker," replied McGregor, "and I need information about you and your wife. Which, if either, is a knight, and which, if either, is a knave?"
> "We are both knaves!" said the husband angrily as he slammed the door. What type is the husband and what type is the wife?"

The solution is as follows [16, p. 16]:

> "If the husband were a knight, he would never have claimed that he and his wife were both knaves. Therefore he must be a knave. Since he is a knave, his statement is false; so they are not both knaves. This means his wife must be knight. Therefore he is a knave and she is a knight."

As it can be seen from the solution of the puzzle, when a reasoner is asked to solve this puzzle he first makes assumptions, then based on these assumptions he considers a hypothetical world and then tries to find out if there are any inconsistencies in this hypothetical world. If an inconsistency exists he concludes that his assumption is wrong

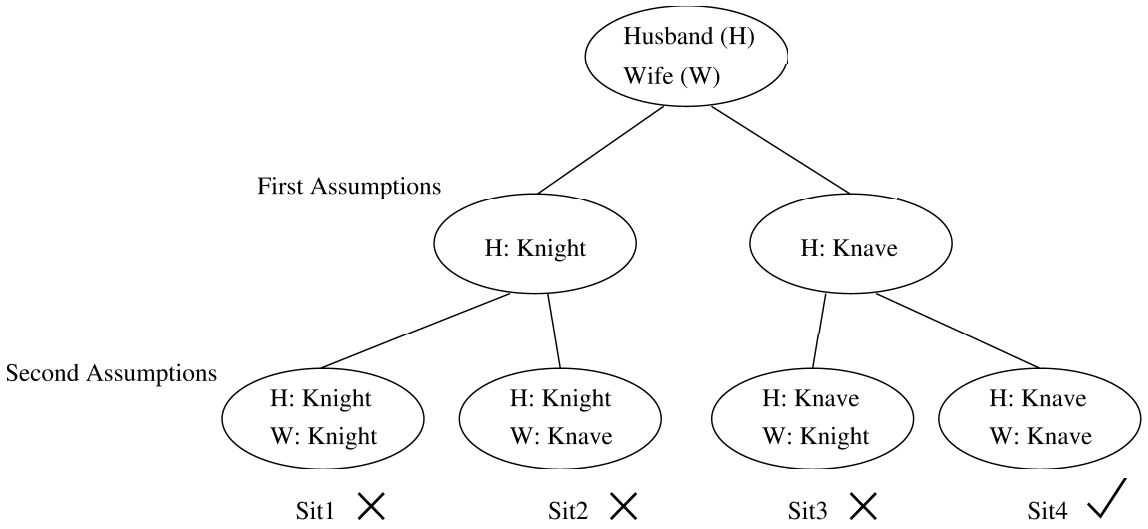

Figure 2: The hypothetical worlds created by the reasoner for the census taker problem. There is only one world (Sit4) consistent with the statement the husband uttered.

and totally forgets about that hypothetical world. The reasoner continues to make new assumptions (while learning something from the previous failures) until he finds all the solutions of the puzzle, i.e., the consistent hypothetical worlds (Figure 2). In the puzzle above, first it was assumed that the husband is a knight, but this assumption led to failure because a knight can never claim that he is a knave (an inconsistency). So it was decided that the husband is a knave. The next section explains how these puzzles are handled in PROSIT.

4.3 Epistemic Puzzles in PROSIT

Examining the structure of these puzzles one will notice properties that are suitable for a situation-theoretic representation. These are as follows:

- Actions always take place in a clearly defined context, i.e., the Island of Knights and Knaves.
- There are abstract individuals, properties, and relations (e.g., being a knight, being on the island, and so on).
- There are well-defined rules that invariably hold on the island (e.g., knights always make true statements).

As mentioned previously, a system to solve these puzzles should be able to make human-like reasoning. There are three main properties that enable PROSIT to simulate human-like reasoning. The first one is situated programming, i.e., infons and constraints are local to situations. The second is PROSIT's situation tree structure, with which we can represent nested knowledge/belief (e.g., "A thinks that B believes that C knows ...").

```
; testing the consistency of a situation requires a
; translation of the uttered sentences to what they
; really mean
(!  (resp island (<= (consistent)
      (means P1 *sentence *translation)
      (means P2 *sentence2 *translation2)
      (and *translation *translation2))))
; every sentence uttered by a knight is true
(!  (resp island (<= (means *x *sentence *sentence)
      (says *x *sentence)
      (knight *x))))
; any sentence uttered by a knave is false
(!  (resp island (<= (means *x *sentence (no *sentence))
      (says *x *sentence)
      (knave *x))))
```

Figure 3: Three main constraints of the PROSIT implementation.

The third is the use of inconsistency to generate new information. Now, it is time to see how PROSIT solves these puzzles. The following puzzle [16, pp. 23–24] will be exploited to explain our approach:

> "This is the story of a philosopher—a logician, in fact—who visited the cluster of islands and fell in love with a bird-girl named Oona. They were married. His marriage was a happy one, except that his wife was too flighty! For example, he would come home late at night for dinner, but if it was a particularly lovely evening, Oona would have flown off to another island. So he would have to paddle around in his canoe from one island to another until he found Oona and brought her home. [...] On one occasion, the husband came to an island in search of Oona and met two natives A and B. He asked them whether Oona had landed on the island. He got the following responses:
> A: B is a knight, and Oona is on this island.
> B: A is a knave, and Oona is on this island.
> Is Oona on this island?"

The solution of this puzzle will make use of various properties of PROSIT, including inheritance. As the solution is based on creating hypothetical situations and testing their consistency, it is useful to have a situation, say *island*, from which all the hypothetical situations will inherit some essential facts that will not change from one situation to another. For example, the fact that the native A says "B is a knight, and Oona is on this island" will hold in every hypothetical situation. Therefore this fact is kept in the *island* situation. Similarly, the rules stating that knights always make true statements and that knaves always make false statements are kept in the *island* situation. The three main constraints used in the solution of this puzzle are shown in Figure 3. The first step of the solution, i.e., making assumptions about the natives, is simulated by creating

hypothetical situations. Each hypothetical situation represents a different combination of assumptions. A reasoner can assume the native A to be a knight or a knave, the native B to be a knight or a knave, and Oona to be on the island or not. So, the program will generate eight (2^3) hypothetical situations. The following are two hypothetical situations (`Sit1, Sit2`) that we will be examining throughout this section:

```
Sit1:  (knight A) (knave B) (on-island Oona)
Sit2:  (knave A) (knave B) (not-on-island Oona)
```

The next step is to generate the infons that hold in the hypothetical situations. If a knight makes a statement, it means that this statement holds in that situation. On the other hand, if a statement is made by a knave, it is concluded that the negation of that statement holds in the situation. So the following infons hold in the hypothetical situations `Sit1` and `Sit2`:

```
Sit1: (and (knight B) (on-island Oona))
      (no (and (knave A) (on-island Oona)))
Sit2: (no (and (knight B) (on-island Oona))
      (no (and (knave A) (on-island Oona)))
```

The final step is to check the hypothetical situations and to discard the inconsistent ones. The consistent situations are the solutions of the puzzle. In the previous case `Sit1` is one of the inconsistent hypothetical situations to be discarded, and `Sit2` is a solution (in fact, the only solution):

`Sit1`: `(and (knight B) (on-island Oona))` (from the second step)
`(knave B)` (from the first step)
Inconsistency!
`Sit2`: Consistent, therefore A and B are knaves and Oona is not on the island.

Smullyan's solution is as follows [16, p. 26]:

> "A couldn't possibly be knight, for if he were, then B would be a knight (as A said), which would make A a knave (as B said). Therefore A is definitely a knave. If Oona is on the island we get the following contradiction: It is then true that A is a knave and Oona is on the island, hence B made a true statement, which makes B a knight. But then A made a true statement in claiming that B is a knight and Oona is on the island, contrary to the fact that A is a knave! The only way out of the contradiction is that Oona is *not* on the island. So Oona is not on this island (and, of course, A and B are both knaves)."

The simpler puzzle given earlier, i.e., the one about Mr. McGregor, is solved in a similar fashion. There are two natives, H and W, in the puzzle. Each can be either a knight or a knave. So there will be four hypothetical situations (Figure 2):

```
; if a native is a knight, he definitely is not a knave
(!  (resp island (=> (knight *x)
        (no (knave *x)))))
; if a native is a knave, he definitely is not a knight
(!  (resp island (=> (knave *x)
        (no (knight *x)))))
; (no (and *st1 *st2)) is equivalent to (or (no *st1) (no *st2))
(!  (resp island (<= (means *x (or (no *st1) (no *st2)))
        (says *x (and *st1 *st2))
        (knave *x))))
; (no (or *st1 *st2)) is equivalent to (and (no *st1) (no *st2))
(!  (resp island (<= (means *x (and (no *st1) (no *st2)))
        (says *x (or *st1 *st2))
        (knave *x))))
```

Figure 4: The constraints about negative knowledge.

```
Sit1:  (knight H) (knight W)
Sit2:  (knight H) (knave W)
Sit3:  (knave H) (knight W)
Sit4:  (knave H) (knave W)
```

After the generation of new infons using the statement uttered by H, the hypothetical situations will consist of the following:

```
Sit1:  (knight H) (knight W) (and (knave H) (knave W))
Sit2:  (knight H) (knave W) (and (knave H) (knave W))
Sit3:  (knave H) (knight W) (no (and (knave H) (knave W)))
Sit4:  (knave H) (knave W) (no (and (knave H) (knave W)))
```

Among these hypothetical situations the only consistent one is `Sit3`, which states that H is a knave and W is a knight.

It is time to examine how PROSIT finds out about these inconsistencies. As it is seen from the examples above a distinguishing feature of PROSIT is that it allows inconsistency in situations. The assertion of an infon i in a situation does not prevent it to be supported by that situation. A situation may support both i and `(no` i`)`. This should not be considered as a contradiction in the system, but merely a contradiction in the situation, which means that the situation is inconsistent. This kind of inconsistency can be adequately used to get new information. In the example above, there is a situation (`Sit1`) that supports both `(knight H)` and `(knave H)`. `(knave H)` is equivalent to `(no (knight H))` (using the rules in Figure 4), therefore both `(knight H)` and its negation are supported by the situation. The situation is inconsistent and the assumptions have failed. One final comment on PROSIT is that it does not apply the predicate `no` over the predicates `and`

and `or`, therefore two additional constraints should be explicitly defined in order to achieve this (Figure 4).

5 Concluding Remarks

This study shows that situation theoretic languages are suitable means for human-like reasoning. PROSIT is especially appropriate for problems involving knowledge and belief. Self-referential expressions, and situations as arguments of infons are two very powerful features. These features can efficiently be used in representing one's knowledge and beliefs. It is left for future work to improve the algorithms and rules so that fewer number of hypothetical worlds are manipulated. This can be achieved once we decide to use some statements as shortcuts, e.g., a knight can never state that he is a knave, or a knave can never utter a statement such as "If I am a knight, then ...". These facts can be employed to limit the number of hypothetical worlds.

This paper should be considered as a preliminary study on 'real life' situation-theoretic applications. We hope that situation-theoretic systems will be used more seriously in general knowledge representation applications.

References

[1] J. Barwise, *The Situation in Logic*, CSLI Lecture Notes, No. 17, Center for the Study of Language and Information, Stanford University, Stanford, CA, 1989.

[2] J. Barwise and J. Etchemendy, *The Liar: An Essay on Truth and Circularity*, Oxford University Press, New York, N.Y., 1987.

[3] J. Barwise and J. Etchemendy, "Model-Theoretic Semantics," in M. I. Posner, editor, *Foundations of Cognitive Science*, MIT Press, Cambridge, MA, pp. 207–243, 1989.

[4] J. Barwise and J. Perry, *Situations and Attitudes*, MIT Press, Cambridge, MA, 1983.

[5] A. W. Black, *Constraints in Computational Situation Theory*, Lecture Notes circulated during *Logic, Language, and Information Summer School*, Saarbrücken, Germany, May 1991.

[6] A. W. Black, *An Approach to Computational Situation Semantics*, Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, U.K., April 1993.

[7] R. Cooper, "Three Lectures on Situation Theoretic Grammar," in M. Filgueiras, L. Damas, N. Moreira, and A. P. Thomás, editors, *Natural Language Processing*, Lecture Notes in Artificial Intelligence, Vol. 476, Springer-Verlag, Berlin, Germany, pp. 102–140, 1991.

[8] R. Cooper, *A Working Person's Guide to Situation Theory*, Human Communication Research Centre (HCRC) Publications, RP–24, University of Edinburgh, Edinburgh, U.K., October 1991.

[9] K. Devlin, *Logic and Information*, Cambridge University Press, New York, N.Y., 1991.

[10] D. B. Lenat et al., "Cyc: Toward Programs with Common Sense," *Communications of the ACM*, 33(8): 30–49, August 1990.

[11] J. McCarthy, "Programs with Common Sense," in V. Lifschitz, editor, *Formalizing Common Sense: Papers by John McCarthy*, Ablex, Norwood, N.J., 1990.

[12] H. Nakashima, S. Peters, and H. Schütze, "Communication and Inference through Situations," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '91)*, pp. 76–81, 1991.

[13] H. Nakashima, H. Suzuki, P. Halvorsen, and S. Peters, "Towards a Computational Interpretation of Situation Theory," in *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS–88)*, Tokyo, Japan, pp. 489–498, 1988.

[14] H. Schütze, "The PROSIT Language," Version 0.4, CSLI Memo, Center for the Study of Language and Information, Stanford University, Stanford, CA, August 1991.

[15] J. Seligman, "Perspectives in Situation Theory," in R. Cooper, K. Mukai, and J. Perry, editors, *Situation Theory and Its Applications*, Vol. 1, CSLI Lecture Notes, No. 22, Center for the Study of Language and Information, Stanford, CA, pp. 147–191, 1990.

[16] R. Smullyan, *Forever Undecided: A Puzzle Guide to Gödel*, Oxford University Press, New York, N.Y., 1987.

[17] R. Smullyan, *Alice in Puzzle-land: A Carrollian Tale for Children Under Eighty*, Penguin Books, New York, N.Y., 1984.

[18] R. Smullyan, *The Lady or The Tiger? (and Other Logic Puzzles)*, Knopf, New York, N.Y., 1982.

[19] R. Smullyan, *What is The Name of This Book? (The Riddle of Dracula and Other Logic Puzzles)*, Prentice-Hall, Englewood Cliffs, N.J., 1978.

[20] E. Tın and V. Akman, "Computational Situation Theory," Manuscript, Department of Computer Engineering and Information Science, Bilkent University, Bilkent, Ankara, Turkey, 1993.

[21] E. Tın and V. Akman, "BABY-SIT: Towards a Situation-Theoretic Computational Environment," Extended Abstract, in C. Martín-Vide, editor, *Proc. I. International Conference on Mathematical Linguistics*, Barcelona, Spain, pp. 97–99, 1993.

[22] E. Tın and V. Akman, "BABY-SIT: A Computational Medium Based on Situations," Abstract, in P. Dekker and M. Stokhof, editors, *Proc. 9th Amsterdam Colloquium*, Institute for Logic, Language, and Computation (ILLC), Department of Philosophy, University of Amsterdam, Amsterdam, The Netherlands, p. 29, 1993.

[23] D. Westerståhl, "Parametric Types and Propositions in First-Order Situation Theory," in R. Cooper, K. Mukai, and J. Perry, editors, *Situation Theory and Its Applications*, Vol. 1, CSLI Lecture Notes, No. 22, Center for the Study of Language and Information, Stanford, CA, pp. 193–230, 1990.

[24] E. N. Zalta, "Twenty-five Basic Theorems in Situation and World Theory," *Journal of Philosophical Logic* 22: 385–428, 1993.