Situated Modeling of Epistemic Puzzles

MURAT ERSAN, *Computer Science Department, Brown University, Providence, Rhode Island 02912, USA. E-mail: me@cs.brown.edu*

VAROL AKMAN, *Department of Computer Engineering and Information Science, Bilkent University, Bilkent, Ankara 06533, Turkey. E-mail: akman@cs.bilkent.edu.tr*

Abstract

Situation theory is a mathematical theory of meaning introduced by Jon Barwise and John Perry. It has evoked great theoretical interest and motivated the framework of a few 'computational' systems. PROSIT is the pioneering work in this direction. Unfortunately, there is a lack of real-life applications on these systems and this study is a preliminary attempt to remedy this deficiency. Here, we solve a group of epistemic puzzles using the constructs provided by PROSIT.

Keywords: computational situation theory, epistemic puzzles, common knowledge, PROSIT

1 Introduction

Situation theory is a principled programme to develop a mathematical theory of meaning which aims to clarify and resolve some formidable problems in the study of language, information, logic, and philosophy of mind. It was introduced by Jon Barwise and John Perry in their *Situations and Attitudes* [3] and stimulated great interest. The theory matured within the last ten years or so [6, 7] and various versions of it have been applied to a number of linguistic issues [8], resulting in what is commonly known as *situation semantics*. This was accompanied by assorted studies on the computational aspects of the theory, which gave birth to a small collection of programming languages based on situation theory; cf. [24] for a recent survey.

PROSIT (PROgramming in SItuation Theory) [14, 15, 5] is *the* pioneering work in this direction. PROSIT seems to be especially suitable for writing programs simulating human-like (commonsense) reasoning [12]. 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 go about formulating a situation-oriented programming paradigm [23, 22]. Pinning our faith upon situation theory, we tried to make use of PROSIT in the solution of what we came to call *epistemic puzzles* [13, 19, 20]. Thus, throughout this paper the nature of epistemic puzzles and their modeling via a situation-theoretic world-view will be analyzed.

A short introduction to situation theory will be offered in the next section. This will be followed by a review of PROSIT. Then we'll explain what epistemic puzzles are like, how they have been solved using classical approaches, and why the situated standpoint fits best to model and solve them. The discussion will be supported by a variety of puzzles, some of which were introduced by Raymond Smullyan in his *Forever Undecided: A Puzzle Guide to Gödel* [20]. (Also cf. [16, 17, 18] for similar puzzles.)

2 Situation theory

Situation theory is a mathematical theory of meaning with considerable depth. The present introduction will cover only those aspects of the theory required for an appraisal of the ideas in the sequel.

Barwise and Perry were aware of the limitations of classical logic and contended that the standard view of logic is inappropriate for many of the uses to which it has been put by semanticists. Some semantic theories emphasized the power of language to classify minds, i.e., the mental significance of language, while others focused on the connections between language and the described world, i.e., the external significance of language. However, Barwise and Perry claim that for an expression to have meaning, it should convey information. This is possible only if the expressions have a link with the kinds of events they describe and also a link with the states of mind. They develop a theory of situations and of meaning as a relation between situations. The theory provides a system of abstract objects that describe the meaning of expressions and mental states in terms of the information they carry about the world.

The information-based approach to the semantics of natural languages has resulted in what is known as situation semantics. The primary idea is that language is used to convey information about the world. Two sentences with the same interpretation—describing the same situation—can carry different information. Context-dependence, which was underestimated in classical approaches to semantics, is the essential hypothesis of situation semantics. Indexicals, demonstratives, tenses, and other linguistic devices rely heavily on context for their interpretation. Therefore, a sentence can be used over and over again in different situations to say different things. Its interpretation is subordinate to the situation in which the sentence is uttered.

The framework of situation theory mainly consists of the things an (intelligent) agent is able to discriminate using his cognitive abilities. The basic ingredients include

- *individuals*, viz. entities that are individuated as 'objects',
- *properties* that hold or fail to hold for these individuals,
- *relations* that hold or fail to hold among these individuals, and
- *spatial* and *temporal locations*, viz. regions of space and time.

Two key notions of situation theory are *infons* and *situations*. Infons are the basic informational units and are considered to be discrete items of information. Infons 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).

Situations are 'first-class' citizens of the theory. There is no clear-cut definition of what a situation exactly is. Rather, a situation is considered to be a structured part of the Reality that a cognitive agent somehow manages to pick out (individuate). Situations support facts:

s *supports* α (denoted as $s \models \alpha$) if α is an infon that is true of situation s.

Truth or falsity do not depend on $\models$ but instead are handled by the notion of polarity. For example, $SIT_1 \not\models \ll$running,Bob,1$\gg$ does not imply $SIT_1 \models \ll$running,Bob,0$\gg$.

Abstract situations are the mathematical constructs with which we can represent real situations. They are more amenable to manipulation. An abstract situation is defined to be a set of infons: given a real situation s, $\{\alpha \mid s \models \alpha\}$ is the corresponding abstract situation.

Types are higher-order uniformities which cut across individuals, relations, situations, and spatial and temporal locations. For each type T, an infinite collection of *parameters* $T_1, T_2, \ldots$ is introduced. For example, IND_1 is an IND-parameter (parameters of type individual). Given a SIT-parameter, SIT_i, and a set of infons, I, $[SIT_i \mid SIT_i \models I]$ denotes a situation-type, the type of situation in which conditions in I are satisfied. For example, $[SIT_1 \mid SIT_1 \models \ll$running,Bob,$LOC_1, TIM_1, 1\gg]$ denotes the type of situation in which Bob is running at some location and at some time.

Frequently, rather than parameters ranging over all individuals, we need parameters that range over a more limited class, i.e., *restricted parameters*. Given a parameter, v, and a condition, D, on v, a restricted parameter $v \uparrow D$ is defined. This is of the same basic type as v and satisfies the requirements imposed by D. For example, in the following, $\dot{b}$ ranges over all footballs and $\dot{a}$ over all men kicking footballs:

$\dot{b} = IND_2 \uparrow \ll$football,$IND_2, 1\gg$
$\dot{a} = IND_3 \uparrow \{\ll$man,$IND_3, 1\gg, \ll$kicking,$IND_3, \dot{b}, 1\gg\}$

In addition, it is possible to obtain new types (in the form $[s \mid s \models I]$) using a parameter and a set of infons. For example, $[SIT_1 \mid SIT_1 \models \ll$kicking,$\dot{a}, \dot{b}, 1\gg]$ represents a situation-type where a man is kicking a football; $[\dot{a} \mid SIT_1 \models \ll$kicking,$\dot{a}, \dot{b}, 1\gg]$ denotes the type of men kicking a football.

In situation theory, the flow of information is realized by a certain group of infons called *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 $S'[f]$. In the following example, an agent who is aware of C will infer that there is a fire whenever he perceives smoke:

$S_0 = [\dot{s_0} \mid \dot{s_0} \models \ll$smoke-present,$\dot{l}, \dot{t}, 1\gg]$
$S_1 = [\dot{s_1} \mid \dot{s_1} \models \ll$fire-present,$\dot{l}, \dot{t}, 1\gg]$
$C = [S_0 \Rightarrow S_1]$.

3 PROSIT

Currently, there are three computational systems based on situation theory. PROSIT, developed by Nakashima et al. [14, 15, 5], is the earliest system. It was followed by the ASTL system of Black [4]. Another computational medium called BABY-SIT is currently being built by Akman and Tın [23, 22, 21]. PROSIT is primarily aimed at problems of knowledge representation whereas ASTL is intended for experiments in natural language processing. On the other hand, BABY-SIT will hopefully handle problems of both sorts. A considerably detailed comparative study on PROSIT, ASTL, BABY-SIT, and other systems with a situation-theoretic flavor has recently appeared [24]. Therefore, we won't go into the details of ASTL and BABY-SIT.

However, since the epistemic puzzles in this paper were implemented in PROSIT, this section will be devoted to that language.

PROSIT [14, 15, 5] is a declarative language in which both programs and data are just sets of *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 language rests on the following desirable features, each of which is supported by the theory:

- Partially specified objects and partial information.
- Situations as first-class citizens.
- Situatedness of information and constraints.
- Informational constraints.
- Self-referential expressions.

3.1 Syntax and semantics

Expressions in PROSIT are either atoms or lists. Atoms that are numbers or strings are considered to be *constants*; atoms that are symbols are regarded either as *parameters* or *variables*. Lists are similar to Lisp lists.

Parameters are Lisp symbols starting with a character other than `*`. They are used to represent things that cannot be captured by PROSIT constants, such as objects, 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. Symbols starting with `*` are variables. Variables are place-holders that stand for any PROSIT expression. They only appear in constraints and queries; their scope is local to the constraint or query they participate in.

In PROSIT, an infon 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. For example, the infon `(loves John Mary)` expresses that the relation `loves` holds between the objects represented by the parameters `John` and `Mary`. PROSIT has no polarity argument in infons, but handles this using the predicate `no`. Thus, `(no` *infon*`)` stands for the 'dual' of *infon*.

One can assert infons, and query a knowledge base incorporating, among other things, infons. Unlike Prolog, infons are local to situations. For example, to assert the infon mentioned above into a situation, `sit1`, we write `(!= sit1 (loves John Mary))`.

In PROSIT, there exists a tree hierarchy among situations, with the situation `top` at the root of the tree. Thus `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`. `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). `out` causes the interpreter to go to the owner of the current situation. 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`. (We can also say that `sit2` is a part of `sit1`, or that `sit1` describes `sit2`.) Briefly, 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`.

Similar to the owner relation there is the *subchunk* relation, denoted by `([_ sit1 sit2)`, where `sit1` is a subchunk of `sit2` (or `sit2` is a *superchunk* of `sit1`). When `sit1` is asserted to be the subchunk of `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: *subtype* and *subsituation*. When the subtype relation, denoted by `(@< sit1 sit2)`), is asserted, it causes the current situation to describe that `sit2` supports each infon 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, denoted by `(<-- sit1 sit2)`, 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 of infons makes it possible to write self-referential statements. Consider a card game (`sit1`) between two players. John has the ace of spades and Mary has the queen of spades. When both players display their cards we have:

```
(!= sit1 (has John a-of-spd))
(!= sit1 (has Mary q-of-spd))
(!= sit1 (sees John sit1))
(!= sit1 (sees Mary sit1))
```

Clearly the last two infons are circular, viz. `sit1` supports facts in which it appears as an argument.

3.2 Inference

The notion of informational constraints is a distinguishing feature that shaped the design of PROSIT. Constraints can be considered as special types of information that '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 `(=>` *head* $result_1 \ldots result_n$`)`. If *head* is asserted to the situation then all of the results are also asserted to that situation. Constraints specified with `<=` are backward-chaining. They are of the form `(<=` *head* $goal_1 \ldots goal_n$`)`. If each of the goals is supported by the situation, then *head* is also supported (though not asserted) by the same situation. Finally, constraints specified with `<=>` should be considered as both backward- and forward-chaining (bidirectional).

If there is a constraint "A smiling person must be a happy person" in `sit1`, i.e.,

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

then the assertion of `(smiling John)` in `sit1` will force PROSIT to assert `(happy John)` in `sit1`. Here `resp` stands for "respects" and causes `sit1` to respect the constraint about smiling.

When an expression, *expr*, is queried, PROSIT tries to evaluate the query, binding values to the variables in the query as the interpreter goes through the database.

If this process fails at any stage, PROSIT backtracks to the previous stage in the search of a solution, and undoes all the bindings made along the incorrect path. The search will succeed if (i) *expr* unifies with an expression that is explicitly asserted in the current situation or its subsituations, or (ii) *expr* unifies with the *head* of a backward-chaining constraint `(<=` *head* $goal_1 \ldots goal_n$`)` and finds a solution to all of the goals when queried in order.

PROSIT offers two types of unification. One is variable unification (V-unification), the other is parameter unification (P-unification). V-unification is the one familiar from Prolog and binds variables to objects. It occurs only in the query mode and its effects are undone when PROSIT backtracks. P-unification occurs only in the assertion mode. It is performed by explicitly stating that two parameters stand for the same object and can be unified. P-unification is one of the major differences between PROSIT and Prolog (in which atoms never unify).

3.3 Applications

Although it offers a variety of constructs that can be used for simulating human-like reasoning, there have been few attempts to employ PROSIT in this style. One of the applications in which PROSIT was used is the treatment of *identity* which we consider next.

Parameters are the means to keep track of the correspondence between concepts in the mind and real objects in the world, cf. [10]. The idea can be exemplified by the following. The famous Roman orator Cicero's first name is Tully. For someone who knows this identity, the answer to the question "Is Tully an orator?" would be "Yes". However, for someone who is not aware of this identity it is not possible to give the same answer.

In PROSIT, it is easy to express the difference between someone who knows the identity of Cicero and Tully, and someone who does not. If an individual is aware of the identity of Cicero and Tully, his knowledge will be classified by `sit1` where

```
(!= sit1 (= cicero tully))
(!= sit1 (orator cicero))
```

Here, the former is a P-unification which states that Cicero and Tully are the same person; the latter states that Cicero is an orator. On the other hand, the knowledge of someone who does not know this identity is classified by `sit2` where

```
(!= sit2 (orator cicero))
```

The system's response for each case will be as follows:

```
(!= sit1 (orator tully))
yes.
(!= sit2 (orator tully))
unknown.
```

A rather recent study on communication and inference through situations [14] was the most serious attempt to make use of PROSIT. It was mainly aimed at a problem ("The Three Wisemen") that requires cooperation in a multi-agent setting. Situation theory was used as a framework to represent common knowledge [1]. The idea behind this choice was to exploit the foundations of situation theory for analyzing information flow.

4 Situations and epistemic puzzles

4.1 Epistemic puzzles

Epistemic puzzles deal with knowledge—either in the form of individual knowledge or common knowledge (mutual information). The ontology of these puzzles include the *agents* whose knowledge we try to represent, $A = \{a, b, \ldots\}$, the *knowledge* each agent has, $K = \{K_a, K_b, \ldots\}$, and the *facts* mentioned in the statement of the puzzle. If we let all the facts in a puzzle make up the set $F = \{f_1, f_2, \ldots\}$ (where each f_i is a relation that holds among the agents and objects that are present in the puzzle), then each K_i is a subset of F. The primary question to be answered in epistemic puzzles is generally about the facts that the agents are aware of. So a puzzle might ask if an agent, say x, is aware of the fact f_i, i.e., whether $f_i \in K_x$. However, this representation fails to handle two fundamental properties of knowledge [1, 2]: circularity (i.e., if a knows f_3, then he knows that he knows f_3, ad infinitum) and deductive omniscience (i.e., if a knows that p and p entails q, then a knows that q). For this representation to handle circularity of knowledge it should be extended such that each K_i is an element of itself. So if a knows the facts f_1, f_3, and f_4, then $K_a = \{f_1, f_3, f_4, K_a\}$. To achieve deductive omniscience the definition of the *facts* should be extended. In addition to simple relations that hold among agents, rules of the form "If ... then ..." should also be considered as belonging to the *facts*.

To elucidate the definitions above, we show how they can be used to represent common knowledge. In a card game, John has the ace of spades and Mary has the queen of spades. Jack comes and looking at her cards announces that Mary has the queen of spades. At this point, each agent's knowledge is represented as follows:

$K_{john} = \{\ll \text{has,john,a-of-spd,1} \gg, K_{john}, K_{common}\}$
$K_{mary} = \{K_{mary}, K_{common}\}$
$K_{common} = \{\ll \text{has,mary,q-of-spd,1} \gg, K_{common}\}$

We now describe, via an example, what an epistemic puzzle looks like:

> Two logicians place cards on their foreheads so that what is written on the card is visible only to the other logician. Consecutive positive integers have been written on the cards. The following conversation ensues:
> A: I don't know my number.
> B: I don't know my number.
> A: I don't know my number.
> B: I don't know my number.
> ... n statements of ignorance later ...
> A or B: I know my number.
> What is on the card and how does the logician know it?

Note that the facts that we are after are restricted. We are only interested in the numbers on the cards, not in the colors or the shapes of the cards. Here, both A and B know some facts, and as the conversation proceeds they generate new facts. At the end, one of them finds out what the number on his forehead is. The aim of this study is to simulate the way the agent holds information about the situation he happens to be in and the way he reasons about this information.

There have been many attempts in AI to deal with knowledge and information. The most common tool used in tackling the fundamental problems posed by these concepts was classical (predicate) logic or its extensions such as modal, temporal, and deontic logics [9, 11]. All these attempts were of a mathematical nature, and therefore within the existing pure mathematics paradigm. On the other hand, situation theory emerged as a realistic theory of information. First, an empirical study of information was made [8]. This was followed by the application of the existing mathematical techniques and the development of new mathematical tools. In that respect, situation theory is tailor-made for problems involving knowledge and information.

4.2 Previous approaches

We'll examine three different approaches used in solving epistemic puzzles. First, we'll analyze how Smullyan solves his knights-and-knaves puzzles using symbolic logic. In [20], Smullyan introduces a number of puzzles about liars and truth-tellers [2]. Most of the events in the puzzles take place on the Island of Knights and Knaves where 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 aim of the puzzles is to decide whether an inhabitant is a knight or a knave using the statements he makes. Assume that P is a native of the Island of Knights and Knaves. Let k be the proposition that "P is a knight". Suppose P utters a proposition X. In Smullyan's puzzles the reasoner knows neither the truth value of k nor the truth value of X, i.e., he does not know whether the native is a knight or a knave, and he does not know whether the asserted proposition is true or false. The only thing he knows is that P is a knight if and only if X is true. So he knows that the proposition $k \equiv X$ is true. So the sentence "P asserts X" is translated as $k \equiv X$. We'll show how this fact helps in the solution of a particular problem that will be detailed in Section 4.3.2: a native of the island, P_1, declares to a census-taker that he and his wife, P_2, are both knaves; can he be telling the truth? Now, k_1 is the proposition that P_1 is a knight, and $\neg k_1$ that he is a knave. Similarly $\neg k_2$ is the proposition that P_2 is a knave. Translating to symbolic logic, the reasoner knows that $k_1 \equiv (\neg k_1 \wedge \neg k_2)$. At this point, the domain of the problem changes from knowledge to symbolic logic: given two propositions k_1 and k_2 such that $k_1 \equiv (\neg k_1 \wedge \neg k_2)$, what are the truth values of k_1 and k_2? Using a truth table one can easily verify that the only case in which $k_1 \equiv (\neg k_1 \wedge \neg k_2)$ is when k_1 is false and k_2 is true. Hence P_1 was lying.

Although there is a very interesting translation here from the domain of knowledge to the domain of symbolic logic, the question "Is this the way an intelligent agent handles such problems?" should be carefully considered. Would an intelligent agent use a truth table to decide who is lying and who is telling the truth?

In the second approach, rather than explaining the way an agent reasons throughout the puzzle, it is proven that the final result that the agent has reached is correct. For example, in the puzzle about cheating husbands this is done using induction [13, p. 168]:

> The queens of the matriarchal city-state of Mamajorca, on the continent of Atlantis, have a long record of opposing and actively fighting the male infidelity problem. Ever since technologically-primitive days of Queen Henrietta I, women in Mamajorca have been required to be in perfect health and pass an extensive logic and puzzle-solving exam before being allowed to take a husband. The queens of Mamajorca, however, were not required to show such competence.
>
> It has always been common knowledge among the women of Mamajorca that their queens are truthful and that the women are obedient to the queens. It was also common knowledge that all women hear every shot fired in Mamajorca. Queen Henrietta I awoke one morning with a firm resolution to do away with the infidelity problem in Mamajorca. She summoned all of the women heads-of-households to the town square and read them the following statement:
>
> *There are (one or more) unfaithful husbands in our community. Although none of you knew before this gathering whether your own husband was faithful, each of you knows which of the other husbands are unfaithful. I forbid you to discuss the matter of your husband's fidelity with anyone. However, should you discover your husband is unfaithful, you must shoot him on the midnight of the day you find about it.*
>
> Thirty nine silent nights went by, and on the fortieth night, shots were heard.

How did the wives decide on the infidelity of their husbands? As a solution to this problem, a theorem stating that if there are n unfaithful husbands they will be shot on the midnight of the nth day, is proven [13, p. 169]. For $n = 1$ there would be one unfaithful husband. His wife would immediately realize that he is the unfaithful one, just after hearing the queen's statement, because she definitely knows that there is no other unfaithful husband. Assume that the claim holds for $n = k$, i.e., if there are k unfaithful husbands they would be shot on the kth night. We prove that the claim also holds if there are $k+1$ unfaithful husbands. In that case, every cheated wife would know k unfaithful husbands. As all the cheated wives are logically competent, they know that if there are k unfaithful husbands then those husbands will be shot on the kth night. As none of the cheated wives can prove that their husband is unfaithful, no shots are fired during the first k nights. Because no shots are fired on the kth night, the cheated wives decide that there are more than k unfaithful husbands and that their own husband is unfaithful too. So the unfaithful husbands are shot on the $k + 1$st night. It must be noted that, rather than explicating how the cheated wives decide that their husbands are unfaithful, this proof demonstrates that their decision is correct.

The third approach used in solving these puzzles is the most realistic one. It explains how the agents in these puzzles reason about the situations they find themselves in. A slight blemish of this approach is that, it is informal. For example, the solution of the puzzle where the native P_1 states that he and his wife, P_2, are both knaves is given as follows [20, 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.

This informal solution seems to be the right one to handle these puzzles. What we'll try to do in the sequel is in some sense to formalize it using situation-theoretic concepts.

4.3 The situated approach

Situation theory, as mentioned earlier, is tailor-made for the problems involving knowledge and information. It provides a group of features that motivated the design of PROSIT—a language especially suitable for writing commonsense reasoning programs [12].

One of these features is that situations are first-class citizens of the theory. This feature combines reasoning *in* situations and reasoning *about* situations. More specifically, situations can be arguments to relations. Therefore situation theory should not only be considered as a theory of relations *in* situations, but also of relations *among* situations.

Both individual and common knowledge are represented as situations. These situations consist of a number of infons representing the facts an agent is aware of. So if Jack knows that John has the ace of spades and that Mary has the queen of spades, then the situation representing Jack's individual knowledge, i.e., `jk`, will consist of two infons (discarding any unrelated stuff):

```
(!= jk (has john a-of-spd))
(!= jk (has mary q-of-spd))
```

An important advantage of using situations to represent knowledge is that it is possible to express some statements that are not expressible in logic. For example, the statement "I know a man who drinks wine every night" can be most closely rendered in (implicitly sorted) predicate logic by the following formula:

$$(\exists x)[know(I, x) \wedge drinkswineeverynight(x)]$$

However, this formula can also be interpreted as: "I know a man, and that man drinks wine every night. (I don't know whether he drinks wine every night.)" Using situations to represent knowledge, we would write the following infons to express the original statement:

```
(!= i-know (man *x))
(!= i-know (drinks-wine-every-night *x))
```

On the other hand, if I didn't know that he drinks wine every night, then the second infon would not hold.

Another feature of situation theory that helps formalize epistemic concepts is the general treatment of partial information. As mentioned previously, situations are partial, i.e., they do not define the truth or falsity of all relations on all objects in the domain. Assume that Mary and Jack are facing each other in a room, and that there is a cat behind Mary. Jack is able to see the cat, so the situation that models his knowledge will support the fact that there is a cat in the room:

```
(!= jk (in-room cat))
```

However, in the situation representing Mary's knowledge there should be no infon to that effect. Hence, when a query is made about the cat in the situation representing Mary's knowledge, the answer should not be `no` but rather `unknown`.

Logical omniscience is supported by informational constraints in situation theory. For example, if Bob is attuned to the constraint that every smiling human being is happy, and knows that John is smiling, then he deduces John is happy. This is represented as follows:

```
(resp bk (=> (smiling *X) (happy *X)))
```

The circularity of knowledge is modeled via "self-referential expressions" which is another important feature of situation theory. Situations are members par excellence of the ontology of situation theory; therefore, they can be used as constituents of infons. For example, if `common` stands for the situation holding the facts that are common to every agent, i.e., the common knowledge situation, and `jk` stands for Jack's individual knowledge, then either of the following expressions states that Jack knows everything that is common knowledge:

```
(!= jk common)
([_ common jk)
```

The second expression illustrates the use of the subchunk relation. It can be translated as "`common` is fully described by the infons in `jk`". So if *infon* holds in `common`, then `(!= common` *infon*`)` holds in `jk`.

Using the subchunk relation it is straightforward to define circularity. `([_ jk jk)` will generate a self-referential situation, and as `jk` stands for the individual knowledge of Jack, it will provide for the desired circularity. So if Jack knows that Mary has the queen of spades, then all of the following will hold:

```
(!= jk (has mary q-of-spd))
(!= jk (!= jk (has mary q-of-spd)))
(!= jk (!= jk (!= jk (has mary q-of-spd))))
...
```

The final feature of situation theory that led us to using it as a framework for epistemic puzzles is the "situatedness of information and constraints". Each infon or constraint exists in a situation (more formally, is supported by a situation). Consequently, each infon or constraint has an interpretation according to the situation it exists in. This can be considered as context-dependence.

To clarify the argument above, consider a constraint that deduces facts about the height of individuals. Let both Mike and John be 185 cm tall. Both are aware of the fact that if someone is higher than 185 cm, then that individual is taller than Mike and John. The following represents the constraint that is supported by Mike's (or John's) knowledge:

```
(resp mk
(<= (taller *y *x) (me *x) (height *y *h) (> *h 185)))
```

If John knows that Bob is 175 cm tall and Mike knows that Bill is 195 cm tall, then Mike will deduce the fact that Bill is taller than himself, viz.

```
(!= mk (taller bill mike))
```

but John will not be able to deduce anything using the previous constraint.

The same argument holds for infons. Imagine a case which Holmes and Watson are working on: a theft in which the door of the flat that the thief broke into is not fractured and that the windows are closed. Although both Holmes and Watson are aware of these facts, viz.

```
(!= hk (intact door) (closed windows))
(!= wk (intact door) (closed windows))
```

only Holmes is able to deduce that the thief had the key to the door. This is because only Holmes finds out that the thief has a key, using the following constraint:

```
(resp hk
(=> (and (intact door) (closed windows) (has thief key))))
```

This example demonstrates that the same infon can generate different facts in different contexts; a system should simulate this capability if it is trying to perform human-like reasoning [12].

It can be concluded that the main advantage of using situation theory in representing knowledge is the conceptual clarity it offers. Epistemic puzzles can be modeled without much effort; all the tools required for this are already present in situation theory. This is due to the fact that situation theory is a 'natural' theory of information.

4.3.1 The Three Wisemen problem

The solution of the "Three Wisemen Problem" [14] in PROSIT is, to our best knowledge, the only serious attempt to use situation-theoretic constructs in the resolution of epistemic puzzles. It turns out that the situation-theoretic aspects of PROSIT (reasoning *about* situations and *in* situations) offer an intuitive and simple solution for this hypothetical problem [14, 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 available facts in the problem can be categorized into two groups: facts that are known by all wisemen 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 say, 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

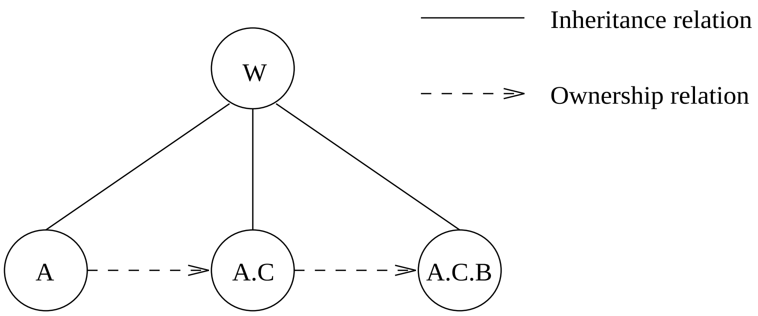

FIG. 1: The Three Wisemen Problem. The facts known to all wisemen are kept in situation W. The facts that A knows are kept in situation A. The facts that A knows that C knows are kept in situation A.C. The facts that A knows that C knows that B knows are kept in situation A.C.B.

red hat causes a contradiction. The approach followed by [14] 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 (A) were wearing a red hat. So A knows that he is wearing a white hat. PROSIT's tree hierarchy of situations makes it rather easy to represent this (Figure 1).

4.3.2 Smullyan's puzzles

These 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 following [20, 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, already given in Section 4.2, is repeated here for convenience [20, 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, when a reasoner is asked to solve this puzzle he first makes some assumptions. Then, based on these assumptions, he considers a

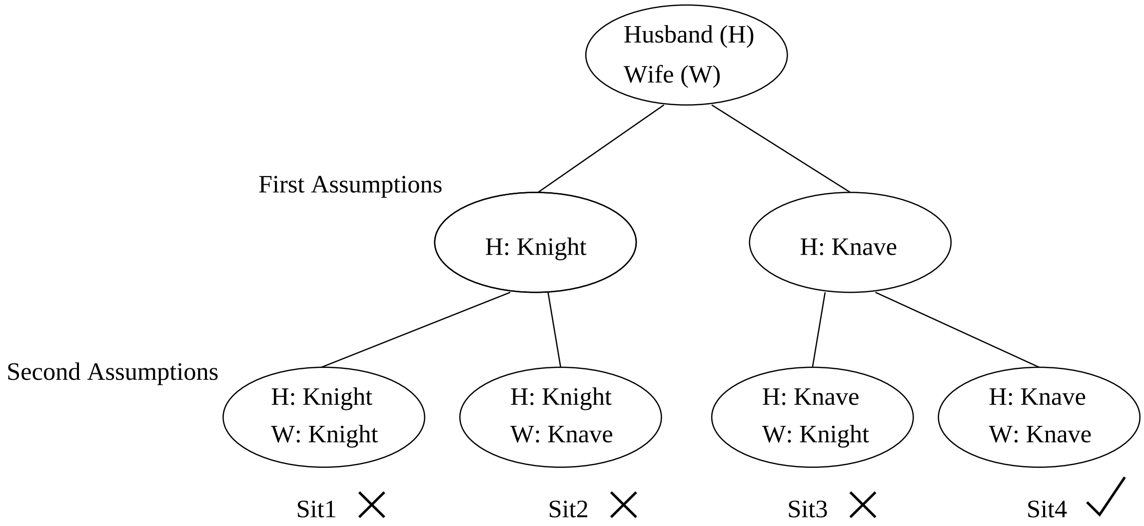

FIG. 2: The hypothetical worlds created by the reasoner for the census-taker problem. There is only one world (Sit4) coherent with the statement the husband uttered.

hypothetical world and tries to find out whether there are any incoherences in this world. If an incoherence is detected he concludes that his assumption is wrong and forgets about that world. The reasoner continues to make new assumptions (while learning something from the previous failures) until he finds all the solutions, i.e., the coherent 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. So it was decided that the husband is a knave.

Examining the structure of these puzzles one must notice properties that are suitable for a situated treatment:

- Actions always take place in a clearly defined context (e.g., 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, for a system to solve these puzzles it should be able to make human-like reasoning. There are three main properties that enable PROSIT to simulate human-like reasoning. The first is situated programming, i.e., infons and constraints are local to situations. The second is PROSIT's situation tree structure, with which one can represent iterated knowledge/belief (e.g., "A knows that B believes that C knows ..."). The third is the use of incoherence to generate new information. Now, we'll see how PROSIT solves these puzzles. The following puzzle [20, 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

```
; Testing the coherency of a situation requires a
; translation of the uttered sentences to what they
; really mean.
(!  (resp island (<= (coherent)
        (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))))
```

FIG. 3. Three main constraints of the puzzle about Oona.

> 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 coherency, 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 A says "B is a knight, and Oona is on this island" will hold in every hypothetical situation. Therefore this is kept in *island*. Similarly, the rules stating that knights always make true statements and that knaves always make false statements are kept in *island*. 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 A to be a knight or a knave, 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:

```
Sit1:  (knight A) (knave B) (on-island Oona)
Sit2:  (knave A) (knave B) (no (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 ones that are incoherent. The coherent situations are then the solutions of the puzzle. In the previous case, `Sit1` is one of the incoherent 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)
Incoherent!
`Sit2`: Coherent, therefore A and B are knaves and Oona is not on the island.

Smullyan's solution is as follows [20, 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 census-taker, 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):

```
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 coherent 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 incoherences. As it is seen from the examples above, a distinguishing feature of PROSIT is that it allows

```
; 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)))))
; N.B. (no (and *st1 *st2)) is equiv.  to (or (no *st1) (no *st2))
(!  (resp island (<= (means *x (or (no *st1) (no *st2)))
         (says *x (and *st1 *st2)) (knave *x))))
; N.B. (no (or *st1 *st2)) is equiv.  to (and (no *st1) (no *st2))
(!  (resp island (<= (means *x (and (no *st1) (no *st2)))
         (says *x (or *st1 *st2)) (knave *x))))
```

FIG. 4. The constraints about negative knowledge.

incoherence in situations. A situation may support both an infon and its dual. This should not be considered as a contradiction in the system, but merely a contradiction in the situation, viz. the situation is incoherent (cannot be actual). This kind of incoherence 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 dual are supported by the situation. The situation is incoherent and the assumptions have failed. One final comment on PROSIT is that it does not distribute `no` over `and` and `or`, therefore two additional constraints should be defined in order to achieve this (Figure 4).

4.3.3 The Cheating Husbands puzzle

The cheating husbands puzzle is well-known from folklore and has long been the primary example to illustrate the subtle relationship between knowledge, communication, and action in a distributed environment. The puzzle involves an initial step in which a set of facts is announced publicly, thereby becoming common knowledge.

Moses et al. [13], using a number of variants of the puzzle, describe what happens when synchronous, asynchronous, and ring-based communication channels are used to send a protocol to be followed, e.g., announce the orders of the queen. The area of distributed computing is mainly interested in the types of protocols, the delays and bounds in communication, and whether the communication is fault-tolerant or not. For example, instead of making an announcement at the town-square, the queen may send letters to all wives; this makes the communication asynchronous. Similarly, to test whether the system is fault-tolerant, another version of the puzzle in which wives are disobedient, i.e., they gossip about their husbands' fidelity, is used.

On the other hand, we are interested in the way agents reason about knowledge, assuming that communication is totally synchronous and reliable. We are using the puzzle to illustrate how intelligent agents reason in a multi-agent system, and how they represent each others' knowledge.

The three wisemen problem is a special case of this puzzle where the number of agents is restricted to three. In this puzzle all the wives in Mamajorca know each other. They know that a husband is either faithful or unfaithful. On the other hand, none of the wives know whether their husbands are faithful or not. Because all of these

```
; A wife knows that her husband is unfaithful if the
; assumption that her husband is faithful results in
; an incoherent situation.
(!  (resp wives (<= (unfaithful *x *time)
                (me *x)
                (wife *y)
                (wife *z)
                (not (= *x *y))
                (not (= *z *x))
                (not (= *z *y))
                (!  ([_ wives *y))
                (!  (@< wives *y))
                (bind-lisp *pre (- *time 1))
                (!  (!= *y (faithful *x *pre)))
                (transfer-knowledge-about-third *y *z)
                (incoherent *y))))
```

FIG. 5: The constraint that decides the faithfulness of a husband by making assumptions and searching for incoherences.

facts are common to all the wives in Mamajorca, they are supported by the situation `wives` which holds the infons that are common to all individuals in the puzzle. Some of the relations require an argument that indicates a temporal location. The temporal location is represented by an integer, n, which indicates the nth night after the queen has made the announcement. Every silent night after the announcement is regarded as the wives' not being able to decide about their husbands' fidelity.

We now analyze the case where there are three unfaithful husbands. After the second silent night following the announcement, `b`'s (a wife) not knowing whether her husband is faithful or unfaithful is represented as

```
(!= wives (no (!= b (faithful b 2))))
(!= wives (no (!= b (unfaithful b 2))))
```

Let `a` be one of the wives whose husband is unfaithful. Throughout the two silent nights she knows who the other cheated wives are (say, `b` and `c`):

```
(!= a (unfaithful b 1))
(!= a (unfaithful c 1))
(!= a (unfaithful b 2))
(!= a (unfaithful c 2))
```

A wife whose husband is unfaithful would realize this fact either if none of the other wives are cheated (the queen declared that there are some unfaithful husbands) or if her assumption that her husband is faithful generates a contradiction. The latter can be considered as proof-by-contradiction.

The way a wife decides that her husband is unfaithful is via making assumptions and checking whether an assumption causes any incoherences (Figure 5). Let `a` be the wife who is reasoning. In the constraint that models the way a wife would reason in such a situation, the premise `(me *x)` would bind `*x` to the situation this constraint

```
; A hypothetical situation is incoherent if it
; supports a fact we know it does not support.
(!  (resp wives (<= (incoherent *y)
        (no (!= *y *x)) (!= *y *x))))
; If the wives *x and *y know the character of the
; third wife's (*z) husband, they know that each of them
; knows it.  So if (!= *x (character *z)), then
; (!= *x (!= *y (character *z))) should be asserted.
(!  (resp wives (<= (transfer-knowledge-about-third *y *z)
             (or
             (and
             (character *character)
             (*character *z *time)
             (bind-lisp *pre (- *time 1))
             (!  (!= *y (*character *z *pre))))
             (true)))))
```

FIG. 6: Constraints that are used to find the incoherences and to transfer the knowledge about the third party.

is activated in, e.g., `a`. The next two premises bind the variables `*y` and `*z` to the other cheated wives `b` and `c`, respectively. The premises

```
(!  ([_ wives *y))
(!  (@< wives *y))
```

indicate that `b`, i.e., the individual bound to the variable `*y`, knows that the facts supported by `wives` are common to all wives (subchunk relation), and is aware of all the facts that are supported by `wives` (subtype relation). Next `a` assumes that her husband is faithful. She knows that if her husband were faithful, the other wives would know it. In the program, this assumption is made by asserting the fact that `a`'s husband is faithful in the situation that holds the facts that `a` knows that `b` knows, via the premise

```
(!  (!= *y (faithful *x *pre)))
```

The variable `*pre` is assigned to the value `*time`-1 (using `bind-lisp` which makes use of Lisp functions), where `*time` indicates the night on which the reasoning is made. Moreover, the facts that `a` knows about `c`'s husband are also asserted into the situation supporting the facts that `a` knows that `b` knows, because what `a` knows about `c`'s husband, `b` knows it too. This is achieved by the constraint `transfer-knowledge-about-third`. The final step is to check if the assumption `a` made would cause any incoherence. This is realized by the constraint `incoherent` which checks if a situation supports a fact we know it does not support. It should be noted that this rule implicitly expresses the fact that if someone is not faithful, he is unfaithful.

The constraint that transfers knowledge about the third individual (Figure 6) is a good example of the use of the situation tree hierarchy. If `a` knows on the second night after the announcement that `b`'s husband is unfaithful then she knows that `c` knows it too:

```
(!= a (unfaithful b 2))
(!= a (!= c (unfaithful b 2)))
```

So an infon supported by the situation `a` is copied to another situation `a.c` using the procedure `transfer-knowledge-about-third`.

The constraint `incoherent` (Figure 6) checks whether a situation is coherent or not. This is achieved by searching for an infon that is supported by that situation with both positive and negative polarities.

To clarify the explanations made above, consider how `a` would reason until she finds out that her husband is unfaithful. `a` knows that `b` and `c` are being cheated:

```
(!= a (unfaithful b 3))
(!= a (unfaithful c 3))
```

`a` wishes to learn whether her husband is faithful or not. She assumes that her husband is faithful. She knows that if her assumption were true, then `b` would be aware of this fact. She also knows that `b` knows the fact that `c` is being cheated.

```
; a's assumption
(!= a (!= b (faithful a 2)))
; transferred knowledge about c
(!= a (!= b (unfaithful c 2)))
```

`b` did not shoot her husband on the second night, i.e., she did not know that her husband was unfaithful. So, an incoherence would occur if she shot her husband on the second night, i.e., if she knew that her husband was unfaithful. (This would make `a`'s assumption false, and mean that `a`'s husband is unfaithful.) To decide on the truth of her assumption, `a` should learn whether `b` could have decided that her husband is faithful or unfaithful. `b` could decide about her husbands faithfulness, just like `a` did. `b` would also assume that her husband was faithful. Then `b` would know that `c` would also know this fact on the first night after the announcement was made. `b` would also know that `c` would have known that `a`'s husband is faithful.

```
; b's assumption
(!= a (!= b (!= c (faithful b 1))))
; transferred knowledge about a
(!= a (!= b (!= c (faithful a 1))))
```

However this assumption of `b` would lead to a contradiction, because if `c` had known that both `a`'s and `b`'s husbands are faithful, then she would have immediately decided that her husband is unfaithful and shoot him.

Because of this incoherence, `b` must decide on the second night that her husband is unfaithful and shoot him. In other words, if `a`'s assumption about her husband were true then `b` would have shot her husband on the second night. But this did not happen, which means that `a`'s assumption that her husband is faithful fails. `a`'s husband is unfaithful and she shoots her husband on the third night after the announcement was made. Making the same reasoning `b` and `c` also shoot their husbands.

4.3.4 The Facing Logicians puzzle

The facing logicians puzzle (cf. Section 4.1) is another puzzle which can be considered to be epistemic. Assume that the first logician, A, has the number 4 on the card on

```
; If the number on the other logician's forehead is n and
; if the logician knows that the number on his forehead
; is not n-1, then the number on his forehead is n+1.
(!  (resp common (<= (know *x)
                 (me *x)
                 (logician *y)
                 (not (= *x *y))
                 (num *y *z)
                 (bind-lisp *a (- *z 1))
                 (no (num *x *a))
                 (bind-lisp *k (+ *z 1))
                 (!  (num *x *k)))))
```

FIG. 7. The constraint with which a logician finds out the number on his forehead.

his forehead, and the other logician, B, has the number 3. A knows that the number on the forehead of B is 3, while the B knows that A has the number 4 on his forehead. It is common knowledge to both that the numbers on their foreheads are positive. (Both are also aware of the fact that common knowledge *is* common.)

```
(!= a (num b 3))
(!= b (num a 4))
(!= common (no (num a 0)))
(!= common (no (num b 0)))
([_ common a)
(@< common a)
([_ common b)
(@< common b)
```

Facts that are common knowledge are known by all the individuals and it is known that these facts are common (Figure 7). The subchunk relation, `[_`, is used to indicate that the individual knows that the facts supported by the situation `common` are common. The subtype relation, `@<`, on the other, indicates that any infon that is supported by the situation `common` is also supported by the situation representing the individual's knowledge.

It is also common knowledge that the numbers are consecutive. So if a logician knows that the number on the forehead of the other logician is n and if he also knows that the number on his own forehead is not $n-1$, then he definitely knows that the number on his forehead is $n+1$.

Assume that B is the one who is asked whether he knows what the number on his forehead is. B would answer "no" because he does not have enough knowledge to make a decision. He could only answer "yes" if the number on the forehead of A were 1. Then he could easily deduce the fact that the number on his forehead was 2. B's answer, however, will make A to learn that the number on his (A's) forehead is not 1.

How does A come to such a decision? Well, he makes assumptions about the number on his forehead. He assumes that the number on his forehead is 1. If it were so, then B would know it. In the program this fact is asserted to the situation `a.b` which holds the facts that A knows that B knows. A continues reasoning: "If B knew that the

```
(!  (resp common (<= (no (know *x))
                     (me *x)
                     (logician *y)
                     (not (= *x *y))
                     (not (!= common (num *y *k)))
                     (no (num *y *z))
                     (bind-lisp *a (+ *z 1))
                     (!  (!= *y (num *x *a)))
                     (!  ([_ common *y))
                     (!  (@< common *y))
                     (incoherent *y)
                     (clear *y)
                     (not (num *x *s))
                     (!  (!= common (no (num *x *a)))))))
```

FIG. 8: The constraint that generates the numbers that cannot be on the forehead of a logician.

number on my forehead were 1, would everything be as it is now? Would it cause any contradiction?" So A tries to find a contradiction in the facts that he knows that B knows. From his previous answer A knows that B does not know what the number on his (B's) forehead is. So A tries to prove that B would know the number on his (B's) forehead, if the number on A's forehead were 1, and reaches a contradiction. B would know what the number on his (B's) forehead is using the rule in Figure 7. This kind of reasoning using incoherences in situations is performed by the constraint in Figure 8.

To elucidate the way the program deduces the facts about the number on the forehead of a logician, we examine in detail the situation in which `a` has 4 on his forehead, and `b` has 3. In the beginning, it is common knowledge that none of the logicians have the number 0 on their foreheads:

```
(!= common (no (num a 0)))
(!= common (no (num b 0)))
```

The situation tree is illustrated in Figure 9 where `a` and `b` are the situations that support the facts known by A and B, respectively, and `common` denotes the situation supporting the facts that are common to both agents. The dashed arrows indicate that both `a` and `b` are inheriting the infons supported by `common`, i.e., both agents are aware of the facts that are common, and know that these facts are common.

After B says "I don't know my number," the fact that the number on A's forehead is not 0 becomes common knowledge (Figure 10):

```
(!= common (no (num a 1)))
```

Next, A says "I don't know my number," implying that the number on B's forehead is neither 1 nor 2 (Figure 11):

```
(!= common (no (num b 1)))
(!= common (no (num b 2)))
```

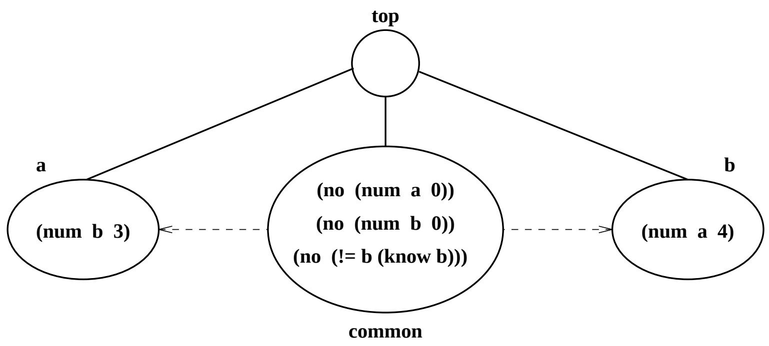

FIG. 9: The situation tree shows the facts that A knows, B knows, and those that are common.

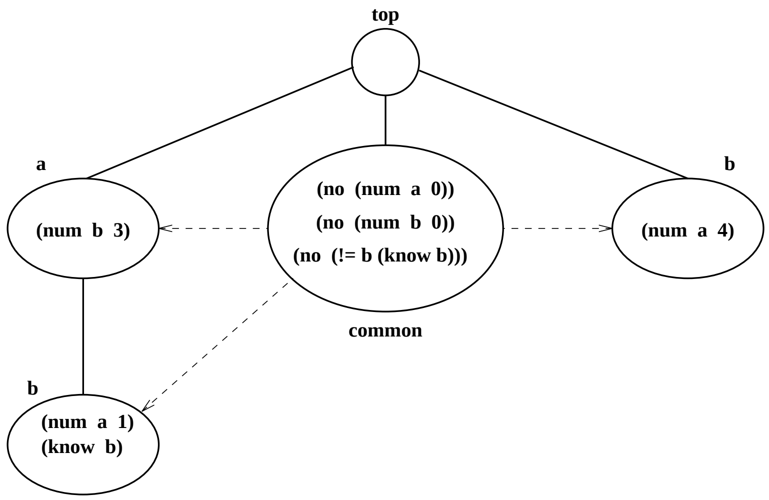

FIG. 10: A makes the assumption that the number on his forehead is 1, and reaches to an incoherence.

Then, B once again says "I don't know my number," and it is concluded that the number on A's forehead is neither 2 nor 3 (Figure 12):

```
(!= common (no (num a 2)))
(!= common (no (num a 3)))
```

At this moment, A deduces that the number on his forehead is 4, because he knows that the numbers are consecutive, that B's number is 3, and that the number on his own forehead is not 2 (Figure 13):

```
(!= a (num a 4))
```

Note that the logicians are making intelligent assumptions. If it is known that the number on the forehead of A is not n, then B assumes that the number on his

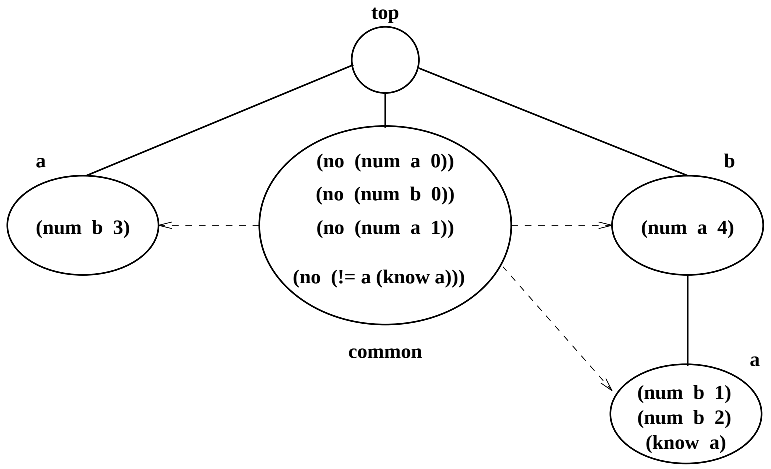

FIG. 11: B makes the assumption that the number on his forehead is 1 or 2, and each time is led to an incoherence.

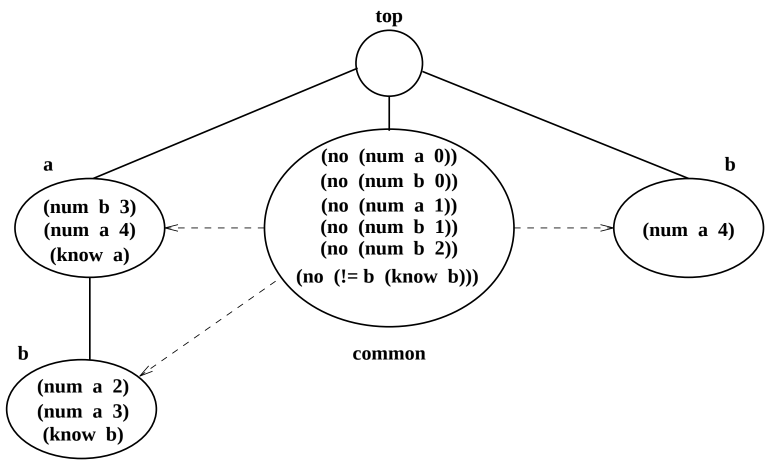

FIG. 12. A finds out that the number on his forehead is neither 2 nor 3.

forehead is $n + 1$. At the instant when it is known that the number on the forehead of A is not 0 and 1, B assumes that the number on his forehead is 1 or 2, which helps him reach an incoherence, and derive new facts. However, B's assuming that the number on his forehead is, say 8, would not help him much.

5 Conclusion

Our primary aim was to analyze PROSIT [14, 15, 5] which is based on situation theory, and to investigate applications that can be nurtured by situation theory. We chose epistemic puzzles [16, 17, 18, 20] as a test domain because these embody the knowledge

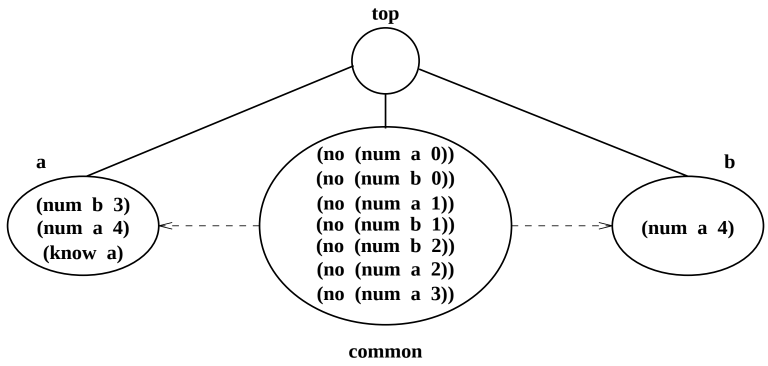

FIG. 13. *A* knows that the number on his forehead is 4.

individuals have and the way they reason about it. We believe that situation theory provides an ontologically adequate framework to represent such puzzles.

Our results show that PROSIT is especially appropriate for problems involving knowledge and belief. PROSIT provides useful handles on technical concepts such as self-referential expressions and situations as arguments of infons. Moreover, it offers additional tools such as the situation tree hierarchy and the inheritance mechanism, which facilitate the representation of individual and common knowledge.

We hope that our study may provide the necessary motivation for further investigations on computational systems based on situation theory, and their utilization in assorted problems of knowledge representation. A logical next step in this regard may be to exercise the capabilities of BABY-SIT [23, 22, 21].[1]

Acknowledgements

We are grateful to Stanley Peters for help and advice regarding PROSIT. We also thank Hideyuki Nakashima and Hiroyuki Suzuki. Two anonymous referees of the *Bulletin* suggested various improvements on our initial manuscript; many of their alterations were incorporated in this version which, we hope, is more readable.

References

[1] J. Barwise. On the model theory of common knowledge. In *The Situation in Logic*, number 17 in CSLI Lecture Notes, pages 201–220. Center for the Study of Language and Information, Stanford, CA, 1989.

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

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

[4] A. W. Black. *An Approach to Computational Situation Semantics.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh, U.K., 1993.

[1] Readers interested in obtaining the experimental pieces of PROSIT code developed during the writing of this paper are kindly asked to contact V. Akman. PROSIT is available from CSLI (Stanford University) using the URL http://csli-www.stanford.edu/.

[5] Janik Borota, Michael Frank, John Fry, Atsushi Ito, Hideyuki Nakashima, Stanley Peters, Michael Reilly, and Hinrich Schütze. The PROSIT language v1.0. Center for the Study of Language and Information, Stanford, CA, 1994.

[6] Robin Cooper. Three lectures on situation-theoretic grammar. In M. Filgueiras, L. Damas, N. Moreira, and A. P. Tomás, editors, *Natural Language Processing*, number 476 in Lecture Notes in Artificial Intelligence, pages 102–140. Springer-Verlag, Berlin, 1991.

[7] Robin Cooper. A working person's guide to situation theory. Technical Report HCRC/RP–24, Human Communication Research Centre, University of Edinburgh, U.K., 1991.

[8] Keith Devlin. *Logic and Information*. Cambridge University Press, New York, 1991.

[9] J. Hintikka. *Knowledge and Belief*. Cornell University Press, Ithaca, N.Y., 1962.

[10] D. Israel and J. Perry. What is information? In P. P. Hanson, editor, *Information, Language, and Cognition*, pages 1–28. University of British Columbia Press, Vancouver, 1990.

[11] D. J. Lehmann. Knowledge, common knowledge, and related puzzles. In *Proc. 3rd Ann. ACM Conf. on Principles of Distributed Computing*, pages 62–67, 1984.

[12] J. McCarthy. Programs with common sense. In V. Lifschitz, editor, *Formalizing Common Sense: Papers by John McCarthy*, pages 1–16. Ablex, Norwood, N.J., 1990.

[13] Yoram Moses, Danny Dolev, and Joseph Y. Halpern. Cheating husbands and other stories: A case study of knowledge, action, and communication. *Distributed Computing*, 1:167–176, 1986.

[14] Hideyuki Nakashima, Stanley Peters, and Hinrich Schütze. Communication and inference through situations. In *Proc. 12th Intl. Joint Conf. on Artificial Intelligence (IJCAI '91)*, pages 76–81, Sydney, 1991.

[15] Hideyuki Nakashima, Hiroyuki Suzuki, Per-Kristian Halvorsen, and Stanley Peters. Towards a computational interpretation of situation theory. In *Proc. Intl. Conf. on Fifth Generation Computer Systems (FGCS–88)*, pages 489–498, Tokyo, 1988.

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

[17] Raymond Smullyan. *The Lady or The Tiger? (and Other Logic Puzzles)*. Knopf, New York, 1982.

[18] Raymond Smullyan. *Alice in Puzzle-land: A Carrollian Tale for Children Under Eighty*. Penguin, New York, 1984.

[19] Raymond Smullyan. Logicians who reason about themselves. In J. Y. Halpern, editor, *Proc. 1st Conf. on Theoretical Aspects of Reasoning About Knowledge*, pages 341–351, Morgan Kaufmann, San Mateo, CA, 1986.

[20] Raymond Smullyan. *Forever Undecided: A Puzzle Guide to Gödel*. Oxford University Press, New York, 1987.

[21] E. Tın and V. Akman. Information-oriented computation with BABY-SIT. In *Proc. Conf. on Information-Oriented Approaches to Logic, Language, and Computation*, Saint Mary's College, Moraga, CA, 1994.

[22] Erkan Tın and Varol Akman. BABY-SIT: A computational medium based on situations. In P. Dekker and M. Stokhof, editors, *Proc. 9th Amsterdam Colloq.*, volume III, pages 665–681, Inst. for Logic, Language and Computation, University of Amsterdam, 1993.

[23] Erkan Tın and Varol Akman. BABY-SIT: Towards a situation-theoretic computational environment. In C. Martín-Vide, editor, *Current Issues in Mathematical Linguistics*, pages 299–308. Elsevier, Amsterdam, 1994.

[24] Erkan Tın and Varol Akman. Computational situation theory. *SIGART Bulletin*, 5(4):4–17, 1994.

Received 8 September 1994. Revised 3 January 1995