Situated Nonmonotonic Temporal Reasoning with BABY-SIT*

Erkan Tın and Varol Akman
Department of Computer Engineering and Information Science,
Bilkent University, Bilkent, Ankara 06533, Turkey
email: {tin,akman}@cs.bilkent.edu.tr

Abstract

After a review of situation theory and previous attempts at 'computational' situation theory, we present a new programming environment, BABY-SIT, which is based on situation theory. We then demonstrate how problems requiring formal temporal reasoning can be solved in this framework. Specifically, the Yale Shooting Problem, which is commonly regarded as a canonical problem for nonmonotonic temporal reasoning, is implemented in BABY-SIT using Yoav Shoham's causal theories.

1 Introduction

Serious thinking about the computational aspects of situation theory has started only a decade ago [13]. There have been recent proposals in this direction (i.e., PROSIT and ASTL), with varying degrees of divergence from the ontology of the theory. We believe that a programming environment incorporating bona fide situation-theoretic constructs will be useful in AI and describe our very recent BABY-SIT implementation. A brief account of PROSIT and ASTL is also offered in order to put BABY-SIT into perspective.

While BABY-SIT is a general-purpose programming environment, in this paper we demonstrate its operation on a well-known example from the AI literature, viz. the Yale Shooting Problem (YSP). Since our essential aim is to display the available functionalities of BABY-SIT for representational issues in AI, we simply borrow a theoretical approach—causal theories due to Yoav Shoham—to solve the YSP [18, 17].

*Please address all correspondence to the second author.

It should be noted that our goal in writing this paper is not to offer a new approach to formal temporal reasoning [19]. There are numerous works in AI to that effect. Neither do we want our work to be understood as a temporal data (world model) manager. Regarding this issue, cf. [20] for a system which is in fairly wide use, and seems to be popular due to its speed.

2 Basic Situation Theory

Situation theory is a unified mathematical theory of information content. The original proposal was due to Jon Barwise and John Perry [3]. The theory has matured over the last decade or so [10] and various versions of it have been applied to a number of problems in logic, language, and cognition [6].

In this section, we introduce the basic notions of situation theory. To this end, we follow the definitions given by Devlin [7] almost verbatim. We also use his notation.

The basic ontology of situation theory consists of entities that a finite cognitive agent individuates and/or discriminates as it makes its way in the world. These entities are known as *uniformities* and include:

- *Individuals*: objects that the agent individuates as single, essentially unitary items; denoted by a, b, c, etc.
- *Relations*: uniformities that hold of, or link together specific numbers of, certain other uniformities; denoted by P, Q, R, etc.
- *Spatial locations*: these are not necessarily like the 'points' of mathematical spaces, for they can have a spatial extension; denoted by l, l', l_0, etc.
- *Temporal locations*: as with spatial locations, temporal locations may be either points in time or regions of time; denoted by t, t', t_0, etc.
- *Situations*: structured parts (concrete or abstract) of the world individuated by the agent; denoted by s, s', s_0, etc.
- *Types*: higher order uniformities; denoted by S, T, U, etc.

A *scheme of individuation*, i.e., a way of carving the world into uniformities, is an essential aspect of situation theory. This is the agent-relative framework that 'picks out' the ontology. In other words, the basic constituents of the theory are determined by the agent's scheme of individuation.

In situation theory, information is always taken to be information about some situation and is in the form of discrete items. *Infons* are these discrete items of information and *situations* are first-class objects which describe limited portions of the real world. Infons are denoted as $\ll R, a_1, \ldots, a_n, i \gg$ where R is an n-place relation, $a_1, \ldots, a_n$ are objects appropriate for the respective argument places of R, and i is the polarity assigned to the sequence $R, a_1, \ldots, a_n$. A polarity value of 1 (respectively, 0) indicates that objects $a_1, \ldots, a_n$ do (respectively, do not) stand in the relation R.

If R is an n-place relation and $a_1, \ldots, a_m$, $m \leq n$, are objects that are appropriate for the argument places $i_1, \ldots, i_m$ of R, and if the filling of these argument places is sufficient to satisfy the minimality conditions for R, then for $i \in \{0, 1\}$, $\ll R, a_1, \ldots, a_m, i \gg$ is a well-defined infon. *Minimality conditions* for R are the collection of conditions that determine which particular groups of argument roles need to be filled in order to produce an infon. If $m < n$, the infon is said to be *unsaturated*; if $m = n$, it is *saturated*.

Given an infon σ and a situation s, we write $s \models \sigma$ to indicate that infon σ is 'made factual' by s—or that σ is an item of information that is true of s. It is also said that s *supports* σ. In case σ is not true of s, this is denoted by $s \not\models \sigma$. Situations are intensional objects. For this reason, abstract situations are proposed to be their counterparts amenable to mathematical manipulation. Given a real situation s, the set $\{\sigma \mid s \models \sigma\}$ is the corresponding abstract situation.

Situations in which a sequence is assigned both polarities are *incoherent*. For instance, s is incoherent if $s \models \ll has, alice, A\heartsuit, 0 \gg$ and $s \models \ll has, alice, A\heartsuit, 1 \gg$. This is a situation in which Alice holds the $A\heartsuit$ and she does not hold the $A\heartsuit$ in a particular card game. There cannot be a real situation s validating this. Nevertheless, the sequence $\ll has, alice, A\heartsuit \gg$ may be assigned both polarity values for spatio-temporally distinct situations (say, two subsequent card games s and s').

Situation theory provides a collection of *basic types* that can be used for individuating or discriminating uniformities in the real world. The higher types of the theory are defined by (recursively) applying *type-abstraction* procedures over the basic types [7]. There are nine basic types:

1. TIM: the type of a temporal location.
2. LOC: the type of a spatial location.
3. IND: the type of an individual.
4. REL^n: the type of an n-place relation.
5. SIT: the type of a situation.
6. INF: the type of an infon.
7. PAR: the type of a parameter.

8. POL: the type of a polarity.

9. TYP: the type of a type.

For each basic type T other than PAR, there is an infinite collection T_1, T_2, ... of *basic parameters*, denoting arbitrary objects of type T. Occasionally, $\dot{a}$, $\dot{s}$, $\dot{t}$, $\dot{l}$, etc. are used to denote parameters for objects of type IND, SIT, TIM, LOC, etc., respectively. Given an object x and a type T, we write $x : T$ to indicate that x is *of type* T.

Abstraction can be captured by allowing parameters in infons. Parameters are generalizations over classes of non-parametric objects (e.g., individuals, spatial locations). For example, $\ll sees, \dot{x}, alice, 1 \gg$ and $\ll sees, \dot{x}, \dot{y}, 1 \gg$ are *parametric infons* where $\dot{x}$ and $\dot{y}$ stand for individuals. The former says that someone sees Alice, whereas the latter says that someone sees someone else.

A situation s' is said to be a *part of* another situation s—or s' is a *subsituation* of s—just in case $\forall\sigma[s' \models \sigma \rightarrow s \models \sigma]$. The *part-of* relation (denoted by $s' \subseteq s$) is reflexive, anti-symmetric, and transitive, and consequently provides a partial ordering of the situations.

In situation theory, information flow [8] is made possible by a network of abstract links between situation types. These links are called *constraints*. They capture systematic regularities connecting situations of one kind with situations of another. One way to picture the functioning of constraints is to think of a constraint $S \Rightarrow T$ as providing a passage that leads from the class of all situations of type S to the class of all situations of type T. Given a situation $s : S$, the constraint $S \Rightarrow T$ provides the information that there is a situation $t : T$. Hence, if an agent attuned to this constraint encounters a situation s and recognizes that s is of type S, then it has the information that the world of which s is a part is such that there is a situation t of type T.

The role of constraints in information flow is best illustrated with an example. The statement *Smoke means fire* expresses the law-like relation that links situations where there is smoke to situations where there is a fire. If T_{smoke} is the type of situations where there is smoke and T_{fire} is the type of situations where there is a fire, then by being attuned to the constraint $T_{smoke} \Rightarrow T_{fire}$, an agent can pick up the information that there is a fire in a particular situation by observing that there is smoke (in that particular situation).

3 Situation-Theoretic Computational Systems

Two pioneering systems, PROSIT and ASTL, that we will review in this section incorporate only some of the original features of situation theory; the remaining features they omit for the sake of achieving particular goals. In BABY-SIT, which we review in the next section, we have tried to stick to the essentials of the theory and adopted the ontology which was first put forward by Barwise and Perry [3], and then refined by Devlin [7].

3.1 PROSIT

PROSIT (PROgramming in SItuation Theory) was developed by Nakashima et al. [15, 16] and implemented in Lisp. In PROSIT one can define situations and assert knowledge into particular situations. It is also possible to define relations between situations in the form of constraints. There is an inference engine similar to a Prolog interpreter.

One can assert facts that a situation will support. For example, if situation `s1` supports the fact that Bob is a young person, this can be defined in the current situation `s` as:

```
s: (!= s1 (young Bob))
```

PROSIT has no special polarity argument in infons. Thus, `(young Bob)` represents a positive infon whereas `(no (young Bob))` stands for the negation of that infon.

In PROSIT, there exists a tree hierarchy, with the situation `top` (the global situation) at the root of the tree. One can traverse the tree using special predicates. It is possible to make queries from a situation about any other situation, the result depending on where the query is made. If a situation `s2` is defined in the current situation, `s1`, then `s1` is said to be the *owner* of `s2`.

PROSIT has two relations defined between situations. These are the *subtype* relation and the *subsituation* relation. When the subtype relation, denoted by `(-> s1 s2)`, is asserted, it means that `s1` supports every infon valid in `s2` and that `s1` 'respects' (to be defined shortly) every constraint that is respected by `s2`, i.e., `s2` becomes a subtype of `s1`. The subsituation relation, denoted as `(s< s1 s2)`, is the same as `(-> s1 s2)` except that only infons, but no constraints, are inherited.

There is no notion of situation type in PROSIT. For this reason, one cannot represent abstractions over situations and specify relations between them without having to create situations and assert facts to them.

Constraints can be specified using either of the three relations $\Rightarrow$, $\Leftarrow$, and $\Leftrightarrow$. Constraints specified using $\Rightarrow$ (respectively, $\Leftarrow$) are forward (respectively, backward) chaining constraints; the ones using $\Leftrightarrow$ are both backward and forward chaining constraints. Backward chaining constraints are of the form ($\Leftarrow$ *head* $fact_1 \ldots fact_n$). If all the facts are supported by the situation, then *head* is supported by the same situation. Forward chaining constraints are of the form ($\Rightarrow$ *fact* $tail_1 \ldots tail_n$). If *fact* is asserted to the situation, then all the tail facts are asserted to the same situation.

For a constraint to be applicable to a situation, the situation must be declared to 'respect' the constraint. For example, to state that every man is a human being, one writes (`*X` denotes a variable):

```
s:  (resp s1 (<= (human *X) (man *X)))
```

This states that `s1` respects the stated constraint (namely, every man is a human being). Since assertions are situated, a situation may or may not respect a constraint depending on where the query is made. If we assert:

```
s:  (!= s1 (man Bob))
```

then PROSIT will affirmatively answer the query:

```
s?  (!= s1 (human Bob))
```

3.2 ASTL

Black's ASTL (A Situation Theoretic Language) is another programming language based on situation theory [4]. One can define in ASTL constraints and rules of inference over the situations. An interpreter, implemented in Lisp, processes ASTL definitions and answers queries.

ASTL allows individuals, relations, situations, parameters, and variables. These constitute the basic terms of the language. Complex terms are in the form of i-terms (which are simply infons of the form $\langle rel, arg_1, \ldots, arg_n, pol \rangle$), situation types, and situations. Sentences in ASTL are constructed from terms and can be constraints, grammar rules, or word entries.

A *situation type* is given as $[par \mid cond_1 \ldots cond_n]$, where $cond_i$ has the form $par \models$ *i-term*. For example, if situation `S1` supports the fact that Bob is a young person, this can be written as (`S` is a parameter):

$$\texttt{S1:} \quad [\texttt{S} \mid \texttt{S} \models \langle \texttt{young,bob,1} \rangle]$$

The colon indicates that `S1` supports the situation type on its right. The supports relation in ASTL is global rather than situated. Consequently, query answering is carried out independently of the situation in which the query is made.

Constraints are actually backward chaining constraints. Each constraint is of the form $sit_0 : type_0 \Leftarrow sit_1 : type_1, \ldots, sit_n : type_n$, where sit_i is a situation and $type_i$ is a situation type. If each sit_i, $1 \leq i \leq n$, supports the corresponding situation type, $type_i$, then according to the above constraint sit_0 supports $type_0$. For example, the constraint that every man is a human being can be written as follows (`*S` and `*X` are variables):

$\texttt{*S: [S | S} \models \langle\texttt{human,*X,1}\rangle\texttt{]} \Leftarrow \texttt{*S: [S | S} \models \langle\texttt{man,*X,1}\rangle\texttt{]}$

An interesting property of ASTL is that constraints are global. Thus, a new situation of the appropriate type need not have a constraint explicitly added to it. Assume that situation S1, supporting the fact that Bob is a man, is asserted:

$\texttt{S1: [S | S} \models \langle\texttt{man,bob,1}\rangle\texttt{]}$

This together with the constraint above would give:

$\texttt{S1: [S | S} \models \langle\texttt{human,bob,1}\rangle\texttt{]}$

4 Fundamental Notions of BABY-SIT

BABY-SIT has been developed in KEE (Knowledge Engineering Environment) [12]. The BABY-SIT desktop runs on a SPARCstation. The primary motivation underlying BABY-SIT is to facilitate the development and testing of programs in domains ranging from linguistics to AI within a unified framework built upon situation-theoretic constructs [23, 25].

BABY-SIT accommodates the basic features of situation theory and, compared to the existing approaches [4, 5, 16], enhances these features [24, 26]. Devlin's reformed approach to situation theory [7] has been extensively used in designing the formal skeleton of BABY-SIT.

Akin to the basic types in situation theory, there are nine basic types which are employed in BABY-SIT: ~IND (individuals), ~TIM (times), ~LOC (places), ~REL (relations), ~POL (polarities), ~INF (infons), ~PAR (parameters), ~SIT (situations), and ~TYP (types). These are special objects of BABY-SIT that can be used in the same way as ordinary objects are. Moreover, they can be used to associate a type with a new object in the system.

Suppose bob is an individual and s1 is a situation. Then, these objects can be declared as:

```
I> bob:~IND
I> s1:~SIT
```

In BABY-SIT, situations are viewed, as usual, at an abstract level. This means that situations are sets of parametric infons, but they may be non-well-founded (circular) [2]. All situations are required to cohere. Situations (and hence the infons they support) may have spatio-temporal dimensions. A situation can have information about another which is a part of the former. BABY-SIT has a minimal situation w, called the 'background situation,' which is a part of every other situation.

Relations are categorized into two: (i) infonic relations that can be used as major constituents of infons, and (ii) non-infonic relations that can only fill the argument roles of infonic relations. Each infonic relation has 'appropriateness conditions' that determine the types of its arguments. The number of arguments that an infonic relation can take defines the minimality conditions for that relation. Consider the relation *seeing*. If we would like it to be an infonic relation with at most two arguments, the former being of type individual and the latter being of type either a situation or an individual, we write:

```
I> <sees | ~IND, {~SIT, ~IND}> [1]
```

Here, the number in square brackets indicates the minimum number of arguments that can be used with *sees*. Hence, `≪sees, bob, 1≫`, for example, is a valid (unsaturated) infon in the system.

In order for the parameters to be anchored to objects of the appropriate type, parameters must be declared to be from only one of the primitive domains. It is also possible to put restrictions on a parameter. Suppose we want to have a parameter `E` denoting any individual that sees situation `s1`. This can be done by asserting:

```
I> E = IND1 ^ ≪sees, IND1, s1, 1≫
```

`IND1` is a default system parameter of type `~IND`. `E` is an object of type `~PAR` such that if it is anchored to an object, say `o1`, then `o1` must be of type `~IND` and `w` ⊨ `≪sees, o1, s1, 1≫`.

Parametric types are also allowed in BABY-SIT. They are of the form $[P|s \models I]$, where P is a parameter, s is a situation, and I is a set of infons. The type of all situations that Bob sees can be defined as follows:

```
I> ~SITALL = [SIT3 | w ⊨ ≪sees, bob, SIT3, 1≫]
```

Hence, `~SITALL` can be used as a type specifier for declaration of new objects in the environment. An object of type `~SITALL`, say `o2`, is an object of basic type `~SIT` such that `w` ⊨ `≪sees, bob, o2, 1≫`. (`o2` is of basic type `~SIT` since the abstraction parameter `SIT3` is a default system parameter of type `~SIT`.)

Variables in BABY-SIT are used in constraints and query expressions, and have scope only within the constraint or the query expression in which they appear. A variable can match any object appropriate for the argument role it appears in. For example, variables `?S` and `?X` in `?S` ⊨ `≪sees, ?X, s1, 0≫` can only match objects of type `~SIT` and `~IND`, respectively.

BABY-SIT allows the use contextual information which plays a critical role in all forms of behavior and communication [1]. Situations and constraints can be grouped to form a so-called 'perspectivity (constraint) set,' which provides a computational context. Moreover, the partial nature of situations facilitates computation with incomplete information. Constraints in BABY-SIT come in three standard flavors:

Computational Feature	PROSIT	ASTL	BABY-SIT
Unification	$\surd$	$\surd$	$\surd$
Type-theoretic	$-$	$-$	$\surd$
Coherence	$-$	$-$	$\surd$
Forward chaining	$\surd$	$-$	$\surd$
Backward chaining	$\surd$	$\surd$	$\surd$
Bidirectional	$\surd$	$-$	$\surd$
Conditional constraints	$-$	$-$	$\surd$
Partiality	$\surd$	$\surd$	$\surd$
Parameters	?	?	$\surd$
Type abstraction	?	?	$\surd$
Parameter restriction	$-$	?	$\surd$
Unsaturated infons	?	$-$	$\surd$

Legend: $\surd$: exists, $-$: doesn't exist, ?: partially/conceptually exists.

Table 1: A simple comparison of PROSIT, ASTL, and BABY-SIT.

forward chaining, backward chaining, and bidirectional.

Assertions may activate the forward chaining mechanism of BABY-SIT which lets one derive new information via the forward chaining constraints (or bidirectional chaining constraints) of a given perspectivity constraint set. A candidate forward chaining constraint is activated whenever its antecedent is satisfied. All the consequences are asserted if they do not yield a contradiction in the situation into which they are asserted. New assertions may in turn activate other candidate forward chaining constraints.

BABY-SIT provides a useful query mechanism. It is possible to make situated or unsituated queries in its query mode. Queries can be 'proved' by using the backward chaining constraints (or bidirectional chaining constraints) of a given perspectivity set. In addition to querying which situation supports what, it is also possible to ask which (particular) situation does not support an infon or a set of infons.

A comparison of BABY-SIT with PROSIT and ASTL is given in Table 1.

5 YSP

Various nonmonotonic formal systems have been proposed to facilitate commonsense reasoning. *Situation calculus* [14] has initially been used to reason about the effects of actions. Hanks and McDermott [11] describe what they call *temporal projection* in the framework of situation calculus as follows. Given a description of the current situation, descriptions of the effects of possible actions, and a sequence of actions to be performed, how do we predict the properties of the world in the resulting situation?

Hanks and McDermott [11] applied some of the existing logics to scenarios to see whether the expected results are indeed produced. YSP is one of these scenarios, a paradigm to show how the temporal projection problem arises in logical frameworks. At some point in time, a person (Fred) is alive. A loaded gun, after waiting for a while, is fired at Fred. What are the results of this action? One expects that Fred would die and the gun would be unloaded after the firing. But Hanks and McDermott [11] demonstrate that unintended models are obtained; the gun gets unloaded during the waiting stage and firing the gun does not kill Fred.

After Hanks and McDermott showed how existing logics fail to produce the expected results for YSP, researchers proposing new formalisms applied their methods to YSP and other similar scenarios to show how they succeed in avoiding the unintended models. Hanks and McDermott argue that a solution to the temporal projection problem should answer two questions [11, p. 409]:

1. Given a logical theory that admits more than one model, what are the preferred models of that theory (i.e., what is the preference criterion)?
2. Given a theory and preference criterion, how do we find the theorems that are true in all 'most preferred' models?

Shoham's causal theories[1] and preference criterion [17] provide a satisfactory answer to these questions. Moreover, Shoham gives an algorithm that computes the true sentences in the models preferred under his preference criterion, thus making causal theories computationally attractive.

5.1 YSP and Causal Theories

Causal theories contain axioms to reason about the effects of actions. Proceeding in time, knowledge about the future is obtained from what is known and what is not known about the past. We will demonstrate how causal theories can be modeled in BABY-SIT. Suppose that Mary loads a gun at time 0 and fires it at Fred at time 2. We would like to reason about the effect of firing the gun. We provide below a possible axiomatization (using a total of 8 axioms) in causal theories.[2]

$$\textbf{(1)}\ \ \Box(0, loads, mary, gun)$$

[1] Cf. Appendix A for a technical introduction to causal theories.

[2] Axiom schemes (4)–(8) are considered to be implicitly universally quantified over t.

$$(\mathbf{2})\ \ \Box(0, alive, fred)$$

$$(\mathbf{3})\ \ \Box(2, fires, mary, gun)$$

$$(\mathbf{4})\ \ \Box(t, loads, mary, gun) \supset \Box(t+1, loaded, gun)$$

$$(\mathbf{5})\ \ \Box(t, alive, fred) \wedge \Diamond(t, \neg fires, mary, gun) \wedge \Diamond(t, exists, air)$$
$$\supset \Box(t+1, alive, fred)$$

$$(\mathbf{6})\ \ \Box(t, loaded, gun) \wedge \Diamond(t, \neg emptied\_manually, gun) \wedge \Diamond(t, \neg fires, mary, gun)$$
$$\supset \Box(t+1, loaded, gun)$$

$$(\mathbf{7})\ \ \Box(t, alive, fred) \wedge \Box(t, loaded, gun) \wedge \Box(t, fires, mary, gun)$$
$$\wedge \Diamond(t, \neg marshmallow\_bullets\_in, gun) \wedge \Diamond(t, has\_firing\_pin, gun)$$
$$\supset \Box(t+1, \neg alive, fred)$$

$$(\mathbf{8})\ \ \Box(t, loaded, gun) \wedge \Box(t, fires, mary, gun)$$
$$\wedge \Diamond(t, has\_firing\_pin, gun) \wedge \Diamond(t, \neg marshmallow\_bullets\_in, gun)$$
$$\wedge \Diamond(t, exists, air) \supset \Box(t+1, hears, mary, noise)$$

Axioms (1)–(3) are the boundary conditions. (4) is an axiom scheme saying that loading a gun makes it loaded. (5) and (6) are axiom schemes needed for persistence. For instance, (5) says that Fred remains alive unless certain conditions obtain; (6) says that the gun remains loaded unless it is manually emptied or is fired by Mary. Axioms (6)–(8) are known as causal schemes. (8), for example, states that Mary's firing a loaded gun causes her to hear a noise unless certain conditions obtain, viz. the gun has no firing pin, has marshmallow bullets, or there is no air.

The axiom schemes (4)–(8) above must be replicated by instantiating the variable t to time points 0, 1, and 2. This actually results in a finite causal theory. The axioms of this causal theory should then be ordered with respect to the latest time points of the atomic base sentences on the left hand side of causal rules. The cmi model of this causal theory is computed by stepping over each axiom in ordered form and checking whether the left hand side is satisfied. The atomic base sentence on the right hand side of a causal rule is asserted to the knowledge base as soon as its left hand side is satisfied. Therefore, this causal theory produces the expected atomic base sentences: $\top(1, loaded, gun)$, $\top(1, alive, fred)$, $\top(2, loaded, gun)$, $\top(2, alive, fred)$, $\top(3, \neg alive, fred)$, $\top(3, hears, mary, noise)$.[3]

Note that the division between $\Box$- and $\Diamond$-conditions in causal rules is somewhat unclear. Consider causal rule (8). One can start with the hypothesis that firing a gun causes Mary's hearing a loud noise. To quote Shoham [18, p. 166]: "[He] might then modify that, and condition the prediction on the gun's being loaded. Then, in time, [he] might learn about the other necessary conditions, such as there being air, the gun having a firing pin, and so on and so forth. Depending on the likelihood of each of these conditions and on the gravity of making wrong default assumptions, they will become either $\Box$-conditions or $\Diamond$-conditions. After a short while the theory will become sufficiently stable so that, unless that [sic] someone were fed seriously biased data, any new modifications will require adding only $\Diamond$-conditions." Hence, we can say that the distinction between these two classes of conditions is somewhat dependent on the particular context of reasoning and that the $\Diamond$-conditions of a causal rule are those typically true in that particular context.[4]

5.2 Axiomatization of YSP in BABY-SIT

5.2.1 Representing the Objects

We should first define the objects that will be used in the axiomatization of the problem. BABY-SIT *Dialogue Mode* provides working sessions for various modes of operation: Assertion Mode, Constraint Edit Mode, Query Mode, and (Object) Deletion Mode. However, only in *Assertion Mode* it is possible to directly introduce new objects to the system (Figure 1). Assertion Mode allows the user to assert propositions which will define objects in the system, establish hierarchies between situations, or add infons to the existing situations.

We start with the introduction of relations such as `alive`, `loads`, `fires`, etc. that will hold among objects. For example, `fires` is defined as follows:

```
I> <fires | ~IND, ~IND> [2]
```

Its arguments must be filled by objects of type `~IND` since the minimality conditions of the relation is given as 2.[5] Then, we define individuals such as `fred`, `mary`, etc.

[3] Here and in Appendix A, $\top$ stands for *TRUE*.

[4] The reader may refer to [22] and [21] for a discussion on various aspects of causal reasoning in Shoham's model.

[5] We have used only basic types in our declarations in order not to diverge from the essence of

Figure 2 shows the declaration of these objects in Assertion Mode.

We consider each snapshot in time as a situation. For instance, Mary's loading the gun and Fred's being alive at time 0 form a state. Any inferred information about future will be collected in a new state. For this reason, we create a situation `s0` as the initial state and assert the facts that Mary loads the gun and that Fred is alive in `s0`. As soon as `s0` is declared as a situation, it appears in BABY-SIT *Situation Browser* (SBR) (Figure 2). SBR displays situation tree structures graphically. Each situation is shown as a node labeled by the name of that situation. A situation is always displayed at a lower level than its subsituation and a line is drawn from the situation to its subsituation to indicate the tree hierarchy. Each line ends on the supersituation with a filled box. Hence, each situation at the lower parts of SBR is a supersituation with respect to the ones it is connected to in this way. Each time a new situation is created via assertions either in Assertion Mode or during chaining, it is displayed on SBR. Any change in the current situation tree hierarchy is also reflected to SBR. SBR provides an interactive environment so that the user can manipulate the situation tree structure, view infons, add/delete infons, anchor parameters of infons, and issue queries in specific situations. By clicking on a situation node on SBR, the user can perform various menu-driven operations (Figure 3).

5.2.2 Implementing the Axiom Schemes

The next step in the axiomatization of YSP includes writing equivalent BABY-SIT constraints for axiom schemes (4)–(8). Defining objects and asserting infons into situations form a description. This description evolves as new assertions are made. On the other hand, it is possible in BABY-SIT to make inference over a given description by the help of *constraints* and hence carry the current description to new stages. There are mainly three types of constraints in BABY-SIT: *forward chaining*, *backward chaining*, and *bidirectional chaining* constraints. Forward chaining constraints enable one to infer new information from the existing information, alter the environment, and act accordingly. (For this reason, these constraints may be called 'action' constraints.) We will implement the axiom schemes of YSP in the form of forward chaining constraints. Hence, we elaborate on this constraint type in the sequel and define some terms that are employed by BABY-SIT constraints.

A *schematic infon* is an infon that contains at least one variable as either its major constituent or one of its minor constituents. A *schematic compound infon* is a collection of infons, in which there exists at least one schematic infon.

A *schematic infonic proposition* is an expression of the form s $\{\models, \not\models\}$ Γ, where s is either a situation or a variable, and Γ is either a compound infon or a schematic

YSP. Clearly, it is also possible to define complex types built out from the basic types.

compound infon. The ones having the form $s \models \Gamma$ are called *positive* schematic infonic propositions; those having the form $s \not\models \Gamma$ are called *negative* schematic infonic propositions.

A *schematic infonic condition* is of the form $s : \Gamma$ where s is either a situation or a variable, and Γ is either a compound infon or a schematic compound infon.

A forward chaining constraint has two constituent parts: the *body* and the *background conditions*. The body of a constraint has the form:

$$antecedent_1, \ldots, antecedent_n \Rightarrow consequent_1, \ldots, consequent_m$$

where each $antecedent_i$, $1 \leq i \leq n$, is a (positive or negative) schematic infonic proposition and each $consequent_j$, $1 \leq j \leq m$, is a positive schematic infonic proposition. The background conditions of a constraint has the form:

$$condition_1, \ldots, condition_k$$

where each $condition_l$, $1 \leq l \leq k$, is a schematic infonic condition. (The background conditions may be empty.)

Whenever the propositions in the antecedent and the background conditions of a forward chaining constraint succeed, all of the propositions in its consequent are asserted.

Each constraint has a unique identifier associated with it and it must belong to a group. Inferences can be drawn with respect to a given group of constraints. These groups are called *perspectivity (constraint) sets*. For example, the following is a forward chaining constraint named FALLING-BLOCK under the perspectivity set NATURAL-LAW-PERSPECTIVE, and states that blocks drop if not supported:

```
NATURAL-LAW-PERSPECTIVE:
    FALLING-BLOCK:
      ?S1 ⊨≪block, ?B, 1≫,
      ?S1 ⊭≪supports, ?C, ?B, 1≫ => ?S2 ⊨≪drops, ?B, 1≫
    BACKGROUND-CONDITIONS:
      w:≪exists, gravity, 1≫
```

Here, `?S1`, `?S2`, `?B`, and `?C` are variables. `?S1` and `?S2` can only be assigned objects of type `~SIT` while `?B` and `?C` can have values of some type appropriate for the argument roles of `block` and `drops`, and `supports`, respectively.[6] Since the supporting situation variables in the antecedent and consequent parts of the constraint are different, a new situation, in which the block drops, is created when the constraint is activated. In this way, a state change for the block of concern is obtained.

[6] Note that `?S1 ⊨≪block, ?C, 1≫` need not appear in the antecedent of the constraint, for the block may well be supported by any other object, e.g., the table.

FALLING-BLOCK is associated with a background condition. A background condition is, in fact, an assumption which is required to hold for a constraint to be eligible for activation. This constraint can become a candidate for activation only if it is the case that `w` $\not\models$≪`exists, gravity, 0`≫, i.e., if there is no information as to the absence of gravity in the background situation.

Each conditional constraint in BABY-SIT can be reduced to an unconditional one by providing a semantically equivalent negative schematic infonic proposition for each condition in the background conditions of the constraint. For example, an equivalent unconditional constraint for FALLING-BLOCK is:

```
NATURAL-LAW-PERSPECTIVE:
     FALLING-BLOCK:
       ?S1 |=<<block, ?B, 1>>,
       ?S1 |/=<<supports, ?C, ?B, 1>>,
         w |/=<<exists, gravity, 0>> => ?S2 |=<<drops, ?B, 1>>.
```

Before examining the representation of YSP axiom schemes (1)–(8) in BABY-SIT, we provide some rules governing the functionality of the forward chaining mechanism:

- The forward chaining mechanism is initiated either when the user tells the system to do so or by asserting a new proposition into the system.
- Forward chaining can be performed with respect to a given perspectivity set, i.e., by using the forward chaining constraints in a specific perspectivity set. Otherwise, the default system constraints are used.
- To find the candidates for activation, forward chaining constraints are examined according to the 'constraint ordering.' The criterion for constraint ordering used by the forward chaining mechanism is 'constraint type.' 'Non-action' constraints are considered first, and then 'action' constraints.
- Negative schematic infonic propositions in the antecedent part of a forward chaining constraint are always evaluated last, with respect to the positive schematic infonic propositions in that part.
- The consequents of an activated forward chaining constraint are asserted into the system only if they are valid propositions (i.e., if the arguments of infons are of appropriate type and the assertion of the propositions does not produce an incoherent situation).

For axiom schemes (4)–(8) in YSP, we can write equivalent forward chaining constraints in BABY-SIT. $\Box$-conditions of these axiom schemes can be represented by positive schematic infonic propositions while their $\Diamond$-conditions are assumptions and hence can be represented by background conditions in constraints.

Consider axiom scheme (8). By a direct transformation, we obtain the corresponding BABY-SIT forward chaining constraint:

```
?S1 ⊨ {≪loaded, ?G, 1≫, ≪fires, ?M, ?G, 1≫} =>
?S2 ⊨≪hears, ?M, noise, 1≫,
?S1 ⊨≪precedes, ?S1, ?S2, 1≫
-
  w: ≪exists, air, 1≫
?S1: {≪has-firing-pin, ?G, 1≫,
      ≪marshmallow-bullets-in, ?G, 0≫}
```

The "-" sign separates the body and the background conditions of the constraint. The constraint simply states the following: "if g is loaded in a situation s_1, and is fired by m in that situation, m hears a noise in a situation s_2 that temporally succeeds s_1, given that the assumptions (existence of air, g's having a firing pin, but not being loaded with marshmallow bullets) hold." Since these assumptions are required to hold in s_1, we use the same supporting situation variable in the body and the background conditions of the constraint. However, we have required existence of air in the background situation `w`. Note that any infon supported by `w` is also supported by other situations since it is, by default, a part of every other situation. Since we use a situation variable (`?S2`) that only appears in the consequent part of the forward chaining constraint, activating this constraint would create a new situation and assert the instantiated form of `≪hears, ?M, noise, 1≫` into that situation.

Now consider axiom scheme (5). The corresponding BABY-SIT forward chaining constraint can be written as:

```
?S1 ⊨≪alive, ?F, 1≫ =>
?S2 ⊨≪alive, ?F, 1≫, ?S1 ⊨≪precedes, ?S1, ?S2, 1≫
-
  w: ≪exists, air, 1≫
?S1: {≪fires, ?M, gun, 0≫,
      ≪precedes, ?S3, ?S1, 0≫}
```

This constraint says "if f is alive in a situation s_1, it will be alive in a situation s_2 that temporally succeeds s_1, given that there exists air and that another object m does not fire the gun." One point worth mentioning here is the inclusion of an assumption that there should be no situation that precedes the current reasoning situation s_1. This assumption is needed to avoid reasoning over the same situation repeatedly.

As for the axiom scheme (4), it could have been represented in a similar manner. However, we have implemented it as a forward chaining constraint with no state transition, i.e., loading the gun does not cause a change of state:

```
?S1 |=<<loads, ?M, ?G, 1>> => ?S2 |=<<loaded, ?G, 1>>
```

Axiomatization of YSP in the form of BABY-SIT constraints is shown in Figure 4, where constraints (R5)–(R1) in descending order correspond to axiom schemes (4)–(8) in ascending order.

It may be that there appear to be two different situations that immediately succeed a situation. These two situations, however, must represent a unique snapshot in time. For this reason, they must be unified (made into a single situation). We achieve this by introducing new constraints, R6 and R7, which make one of these situations a part of the other. When an infon of the form $\ll$*make-part-of*, s_1, s_2, 1$\gg$ is asserted, s_1 becomes a subsituation of s_2. The relation *part-of* is automatically asserted into s_2. s_2 cannot be the background situation since it is always a subsituation of other situations in the system. Therefore, if there are two situations s_2 and s_3 that succeed situation s_1, constraint R7 unifies s_2 and s_3 (by making s_3 a part of s_2). Similarly, if there are two situations s_1 and s_2 (where one is a part of another) and if these situations have successors s_3 and s_4, respectively, then constraint R6 unifies s_3 and s_4 (Figure 4).

5.2.3 Inferencing Over Situations

Assertion Mode Setup of BABY-SIT serves as a control unit for the evaluation of the expressions asserted in Assertion Mode. There are four possible actions that can be controlled by the user via Assertion Mode Setup (Figure 5):

1. *Anchoring the parameters.* If this option is enabled, each parameter in the asserted expression is replaced by a corresponding individual according to the anchoring defined in the anchoring situation. The anchoring situation can be specified by the user on the corresponding slot of the Assertion Mode Setup template (i.e., *anchoring situation before assertion* in Figure 5). (The anchoring situation defaults to `w`.)

2. *Making inferences with the existing information.* The proposition asserted can be used to make inferences over the existing information by employing a set of forward chaining constraints. An existing perspectivity set name should be typed by the user on the corresponding slot of the Assertion Mode Setup template (i.e., *perspectivity constraint set* in Figure 5). Otherwise, the system perspectivity set is used. In either case, the forward chaining constraints of the current perspectivity set are employed in drawing inferences.

3. *Using a different perspectivity set.* The antecedents of forward chaining constraints in the current perspectivity set are normally proved with respect to the backward chaining constraints in this set. Additionally, the backward chaining constraints in another perspectivity set can be used for this task.

4. *Verifying antecedents.* The antecedents of a candidate forward chaining constraint may be verified prior to its activation.

In our example, we only need the simple inference mechanism of the system, and hence use the second option above. We first set the current forward chaining perspectivity set to GUNFIRE. Then asserting the fact that Mary loads the gun causes constraint R5 to be activated and the gun gets loaded in `s0` (Figure 6). Forward chaining over the existing information creates, by activating constraint R3, a new situation `R3-1` where the gun remains loaded and then creates, by activating R4, another situation `R4-2` where Fred is alive (Figure 7). Constraint R6 unifies these situations in the next activation of forward chaining (Figure 8). Activation of forward chaining once more creates situations `R3-3` and `R4-4` by reasoning over `R3-1` and `R4-2`, respectively (Figure 9).

Subsequent activations of forward chaining unify new states and then create new states by forming new situations in a similar manner. Assume that we iterate forward chaining twice and assert the fact that Mary fires the gun. Then two new situations `R2-7` and `R1-8` are created (by R2 and R1), where Fred ceases to be alive and Mary hears a noise, respectively (Figure 10).

However, if we had asserted $\texttt{w} \models \ll\texttt{exists, air, 0}\gg$ prior to Mary's firing the gun, a new situation, `R2-9`, would be created, where Fred would be dead but Mary would not be able to hear a noise (Figure 11).

5.2.4 Time-dependency in Causal Computations

Shoham's algorithm computes the atomic base sentences known in all cmi models of a *finite* causal theory. Thus, his approach has a problem when the causal theories contain axiom schemes: computation is time-dependent because the size of the corresponding finite causal theory depends on the 'time span' of the theory. However, it is possible in BABY-SIT to represent both finite causal theories and causal theories having axiom schemes. In YSP, except boundary conditions, BABY-SIT contained axiom schemes.

Note that in either case one must draw inferences by beginning from some initial time t_0 and moving forward to times $t_0 + 1$, $t_0 + 2$, and so on. But one may wish to 'jump into conclusions' about the consequences of an action. For example, one may want to directly observe the effect of Mary's loading and/or firing a gun. This can also be done in BABY-SIT with a slight modification of the model presented above. We now describe this modification.

In temporal projection scenarios, there exist two types of axiom schemes. The first type takes care of the persistence of facts, permitting inferences about what remains unchanged. For example, if you load a gun, it will stay loaded unless you fire or empty it. Such schemes are called *persistence axiom schemes*. In YSP, (5) and (6) are persistence axiom schemes.

The second type represents what changes occur in the environment. Such schemes are called *causal axiom schemes*. More specifically, these allow one to infer what kind of changes actions bring about. In YSP, (4), (7), and (8) are causal axiom schemes.

In order to be able to jump into conclusions for YSP, we divide BABY-SIT constraints into two with respect to this categorization. Constraints R1, R2, and R5 form a perspectivity set called CAUSAL, and constraints R3 and R4 are collected into a perspectivity set called PERSIST. We also put R6 and R7 into a separate perspectivity set called UNIFY. Given an initial situation, forward chaining over PERSIST would produce facts that remain unchanged in the next situation in which one one would reason about the effects of an action via CAUSAL. Forward chaining over UNIFY must be needed just after forward chaining over PERSIST to unify the results of persistence (into a single state). Therefore, the order of forward chaining perspectivity sets to determine the consequences of an action should be as follows: PERSIST, UNIFY, CAUSAL.

We form the initial situation `s0` by asserting the fact that Fred is alive. We then set the forward chaining perspectivity set to CAUSAL and assert the fact that Mary loads the gun. R5 is immediately activated and the gun gets loaded in `s0`. Setting the current perspectivity set to PERSIST and enabling forward chaining causes R3 to be activated and creates a new situation `R3-1`, where the gun remains loaded. At the same time R4 is activated and another situation `R4-2` is created, where Fred is alive. Then forward chaining over UNIFY forms a unique state with situations `R3-1` and `R4-2`. Finally, setting the perspectivity set to CAUSAL and asserting the fact that Mary fires the gun in `R3-1` activate constraints R1 and R2. Hence, two new situations `R1-4` and `R2-3` are created where Mary hears a noise and Fred ceases to be alive, respectively.[7]

6 Conclusion

We have shown how AI problems that necessitate a nonmonotonic temporal approach can be handled in the situation-theoretic computational framework of BABY-SIT. Specifically, we have considered the Yale Shooting Problem and have implemented the well-known approach of Yoav Shoham. The conceptual clarity gained by the adoption of situation theory as a knowledge representation scheme makes its presence strongly felt in our handling of YSP. Our immediate goal is to apply situation theory

[7]These two new situations correspond to `R1-8` and `R2-7`, respectively, in Figure 10.

to other representation problems in AI.[8]

On a more general note, we believe that computational aspects of situation theory call for deeper investigation. Although the situation-theoretic systems reviewed in this paper are in their infancy, they are promising in terms of applicability in AI [25].

7 Acknowledgments

We owe special thanks to the Editor-in-Chief for his encouragement and invaluable advice. Comments of an anonymous referee of the journal have also been extremely useful and led to a major revision of the initial manuscript. As usual, all the remaining inadequacies are our own.

A Shoham's Causal Theories

This appendix is summarized from Shoham's doctoral dissertation [18]. It may be skipped by readers already knowledgeable about this work.

The standard monotonic logic on which Shoham's causal theories are based is called the *logic of temporal knowledge* (TK). The syntax and semantics of TK are given below.

Let P be a set of primitive propositions, TV a set of temporal variables, $TC = Z$ (integers: the structure of time), and $U = TC \cup TV$.

Well-formed formulae (wff) are defined as follows:

1. If u_1, $u_2 \in U$ and $p \in P$, then $u_1 = u_2$, $u_1 \leq u_2$, and $\top(u_1, u_2, p)$ are wff.
2. If φ is a wff, then so is $\neg\varphi$.
3. If φ_1 and φ_2 are wff, then so is $\varphi_1 \wedge \varphi_2$.
4. If φ is a wff, then so is $\Box\varphi$. $\Box\varphi$ stands for "φ is known." We define $\Diamond \equiv \neg\Box\neg\varphi$. $\Diamond\varphi$ stands for "φ is not known to be false."
5. If φ is a wff and $v \in TV$, then $\forall v \varphi$ is also a wff.

[8]For a preliminary attempt in this vein, the reader is referred to [9] which studies puzzles of knowledge/belief in PROSIT.

Some abbreviations: $\Box\top(t_1, t_2, p)$ is shortened as $\Box(t_1, t_2, p)$, $\Box\neg\top(t_1, t_2, p)$ as $\Box(t_1, t_2, \neg p)$, $\Diamond\top(t_1, t_2, p)$ as $\Diamond(t_1, t_2, p)$, and $\Diamond\neg\top(t_1, t_2, p)$ as $\Diamond(t_1, t_2, \neg p)$. Finally, $\top(t, p)$ abbreviates $\top(t, t, p)$.

A *Kripke interpretation* (κ) is a pair $< W, M >$, where W is a nonempty universe of possible worlds and M is a meaning function such that $M : P \longrightarrow 2^{W \times Z \times Z}$.

A *variable assignment* is a function $va : TV \longrightarrow Z$. A *valuation function* (val) is such that $val(u) = va(u)$ if $u \in TV$, and $val(u) = u$ if $u \in TC$.

A Kripke interpretation κ and a world $\omega \in W$ *satisfy* a formula φ under va (written $\kappa, \omega \models \varphi[va]$) according to the following definition:

1. $\kappa, \omega \models u_1 = u_2[va]$ iff $val(u_1) = val(u_2)$. $\kappa, \omega \models u_1 \leq u_2[va]$ iff $val(u_1) \leq val(u_2)$. $\kappa, \omega \models \top(u_1, u_2, p)[va]$ iff $< \omega, val(u_1), val(u_1) > \in M(p)$.
2. $\kappa, \omega \models \neg\varphi[va]$ iff $\kappa, \omega \not\models \varphi[va]$.
3. $\kappa, \omega \models \varphi_1 \wedge \varphi_2[va]$ iff $\kappa, \omega \models \varphi_1[va]$ and $\kappa, \omega \models \varphi_2[va]$.
4. $\kappa, \omega \models \Box\varphi[va]$ iff $\kappa, \omega' \models \varphi[va]$ for all $\omega' \in W$. (Therefore, we are able to write $\kappa \models \Box\varphi[va]$ without fear of ambiguity.)
5. $\kappa, \omega \models \forall v \varphi[va]$ iff $\kappa, \omega \models \varphi[va']$ for all va' that agree with va everywhere except possibly on v.

A Kripke interpretation κ and a world $\omega \in W$ are a *model* for a formula φ (written $\kappa, \omega \models \varphi$) if $\kappa, \omega \models \varphi[va]$ for any variable assignment va. A wff is *satisfiable* if it has a model, and *valid* if its negation has no model. φ_1 *entails* φ_2 (written $\varphi_1 \models \varphi_2$) iff φ_2 is satisfied by all models of φ_1.

Base formulae are those wff containing no occurrence of the modal operators. The *latest time point* (ltp) of a base formula is the (chronologically) latest time point mentioned in it.

κ_2 is *chronologically more ignorant* than κ_1 (written $\kappa_1 \subset_{ci} \kappa_2$) if there exists t_0 such that

1. For any base sentence φ (whose ltp $\leq t_0$), if $\kappa_2 \models \Box\varphi$ then also $\kappa_1 \models \Box\varphi$.
2. There exists a base sentence φ (whose ltp is t_0) such that $\kappa_1 \models \Box\varphi$ but $\kappa_2 \not\models \Box\varphi$.

κ is said to be a *chronologically maximally ignorant* (cmi) model of φ if $\kappa \models_{\subset_{ci}} \varphi$, i.e., if $\kappa \models \varphi$ and there is no other κ' such that $\kappa' \models \varphi$ and $\kappa \subset_{ci} \kappa'$. The *logic of* *chronological ignorance*, CI, is the nonmonotonic logic obtained by associating the preference relation $\sqsubset_{ci}$ with TK.

Base sentences in CI are those sentences containing no occurrence of the modal operators, i.e., sentences that refer directly to the real world and not to a knowledge of it. *Atomic base sentences* are of the form $\top(t_1, t_2, p)$ or $\top(t_1, t_2, \neg p)$.

A *causal theory* Ψ is a theory in CI, in which all sentences have the form $\Phi \wedge \Theta \supset \Box\varphi$, where (in the following $[\neg]$ means that $\neg$ may or may not appear):

1. $\varphi = \top(t_1, t_2, [\neg]p)$.
2. $\Phi = \bigwedge_{i=1}^{n} \Box\varphi_i$, where φ_i is an atomic base sentence whose ltp $t_i < t_1$.
3. $\Theta = \bigwedge_{j=1}^{m} \Diamond\varphi_j$, where φ_j is an atomic base sentence whose ltp $t_j < t_1$.
4. Φ or Θ may be empty. A sentence in which Φ is empty is called a *boundary condition.* Other sentences are called *causal rules.*
5. There is a time point t_0 such that if $\Theta \supset \Box(t_1, t_2, [\neg]p)$ is a boundary condition, then $t_0 < t_1$.
6. There do not exist two sentences in Ψ such that one contains $\Diamond(t_1, t_2, p)$ on its left hand side and the other contains $\Diamond(t_1, t_2, \neg p)$ on its left hand side.
7. If $\Phi_1 \wedge \Theta_1 \supset \Box(t_1, t_2, p)$ and $\Phi_2 \wedge \Theta_2 \supset \Box(t_1, t_2, \neg p)$ are two sentences in Ψ, then $\Phi_1 \wedge \Theta_1 \wedge \Phi_2 \wedge \Theta_2$ is inconsistent.

An essential property of a causal theory is that it has cmi models, and in all of these the same set of atomic base sentences is known [18, pp. 112–113].

References

[1] V. Akman and M. Surav. Steps toward Formalizing Context. *AI Magazine*, 17(3):55–72, 1996.

[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. 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, Edinburgh, U.K., 1993.

[5] J. Borota, M. Frank, J. Fry, A. Ito, H. Nakashima, S. Peters, M. Reilly, and H. Schütze. The PROSIT Language, Version 1.0. Manuscript, Center for the Study of Language and Information, Stanford, CA, 1994.

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

[7] K. Devlin. *Logic and Information.* Cambridge University Press, Cambridge, U.K., 1991.

[8] F. Dretske. *Knowledge and the Flow of Information.* MIT Press, Cambridge, MA, 1981.

[9] M. Ersan and V. Akman. Situated Modeling of Epistemic Puzzles. *Bulletin of the IGPL*, 3:51–76, 1995.

[10] J. E. Fenstad, P.-K. Halvorsen, T. Langholm, and J. van Benthem. *Situations, Language, and Logic.* Reidel, Dordrecht, Holland, 1987.

[11] S. Hanks and D. V. McDermott. Nonmonotonic Logic and Temporal Projection. *Artificial Intelligence*, 33:379–412, 1987.

[12] KEE^{TM}. Knowledge Engineering Environment Software Development System, Version 4.1. IntelliCorp, Inc., Mountain View, CA, 1993.

[13] Y. Lespérance. Toward a Computational Interpretation of Situation Semantics. *Computational Intelligence*, 2:9–27, 1986.

[14] J. McCarthy and P. J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, pages 463–502. Edinburgh University Press, Edinburgh, U.K., 1969.

[15] H. Nakashima, S. Peters, and H. Schütze. Communication and Inference through Situations. In *Proceedings of the Third Conference on Artificial Intelligence Applications*, pages 76–81, IEEE Computer Society Press, Washington, D.C., 1987.

[16] H. Nakashima, H. Suzuki, P.-K. Halvorsen, and S. Peters. Towards a Computational Interpretation of Situation Theory. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 489–498, Institute for New Generation Computer Technology, Tokyo, Japan, 1988.

[17] Y. Shoham. Chronological Ignorance: Experiments in Nonmonotonic Temporal Reasoning. *Artificial Intelligence*, 36:279–331, 1988.

[18] Y. Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence.* MIT Press, Cambridge, MA, 1988.

[19] Y. Shoham and D. V. McDermott. Problems in Formal Temporal Reasoning. *Artificial Intelligence*, 36:49–61, 1988.

[20] J. P. Stillman and R. Arthur. Tachyon: A Model and Environment for Temporal Reasoning. In *National Conference on Artificial Intelligence: Workshop on Implemented Temporal Reasoning*, San Jose, CA, 1992.

[21] R. Sun. *Integrating Rules and Connectionism for Robust Commonsense Reasoning.* John Wiley & Sons, New York, N.Y., 1994.

[22] E. Tın and V. Akman. Computing with Causal Theories. *International Journal of Pattern Recognition and Artificial Intelligence*, 6(4):699–730, 1992.

[23] E. Tın and V. Akman. BABY-SIT: A Computational Medium Based on Situations. In P. Dekker and M. Stokhof, editors, *Proceedings of the 9th Amsterdam Colloquium*, pages 665–681. Institute for Logic, Language, and Computation, University of Amsterdam, Amsterdam, Holland, 1993.

[24] E. Tın and V. Akman. Computational Situation Theory. *ACM Sigart Bulletin*, 5(4):4–17, 1994.

[25] E. Tın and V. Akman. Information-Oriented Computation with BABY-SIT. In J. Seligman and D. Westerståhl, editors, *Language, Logic, and Computation: Volume 1*, number 58 in CSLI Lecture Notes, pages 19–34. Center for the Study of Language and Information, Stanford, CA, 1996.

[26] E. Tın, V. Akman, and M. Ersan. Towards Situation-Oriented Programming Languages. *ACM Sigplan Notices*, 30(1):27–36, 1995.

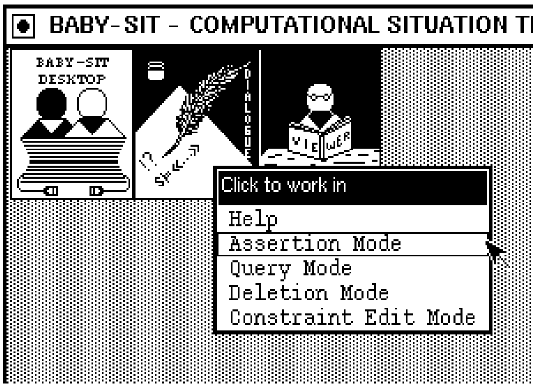

Figure 1: Available operations in Dialogue Mode.

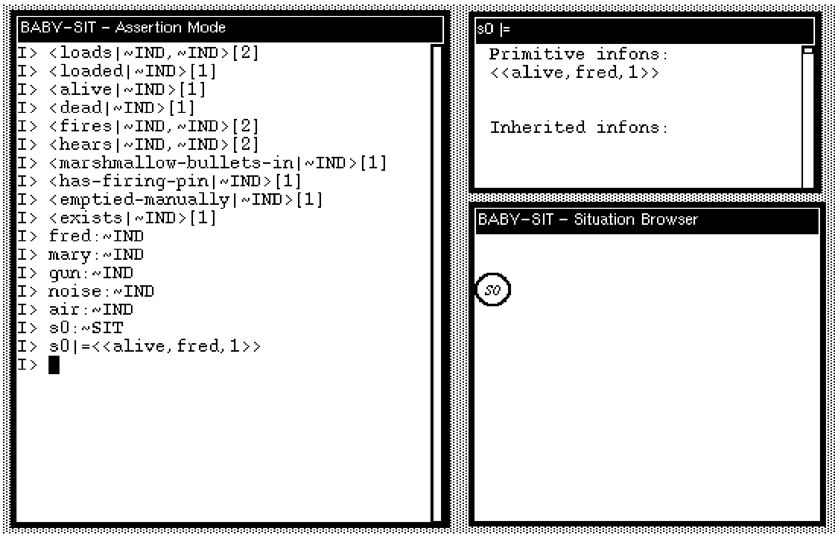

Figure 2: Object declarations for Yale Shooting Problem.

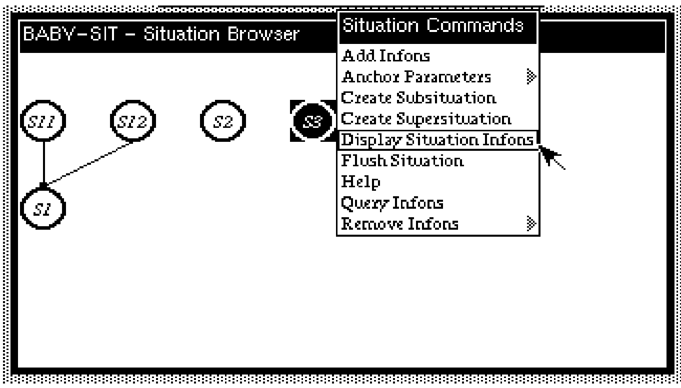

Figure 3: Situation Browser allows menu-driven operations on situations.

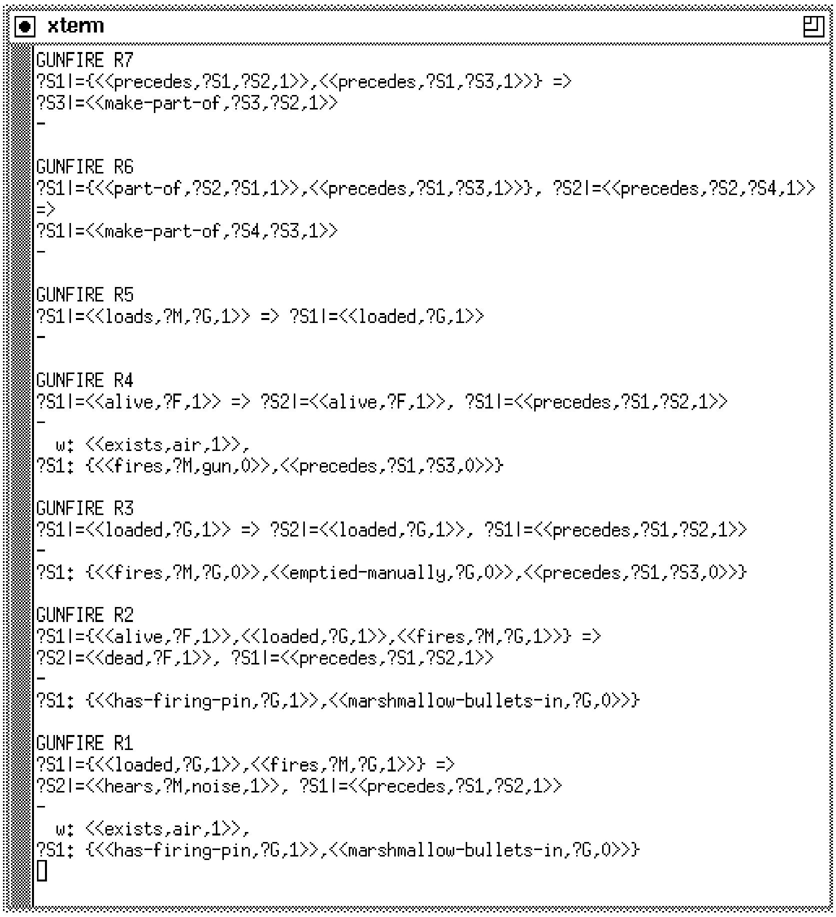

Figure 4: The constraints for Yale Shooting Problem.

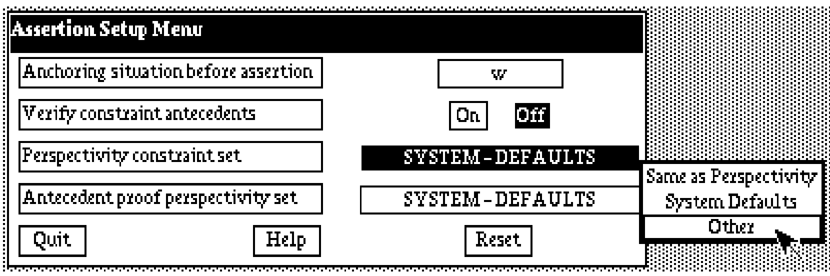

Figure 5: A view from Assertion Mode Setup.

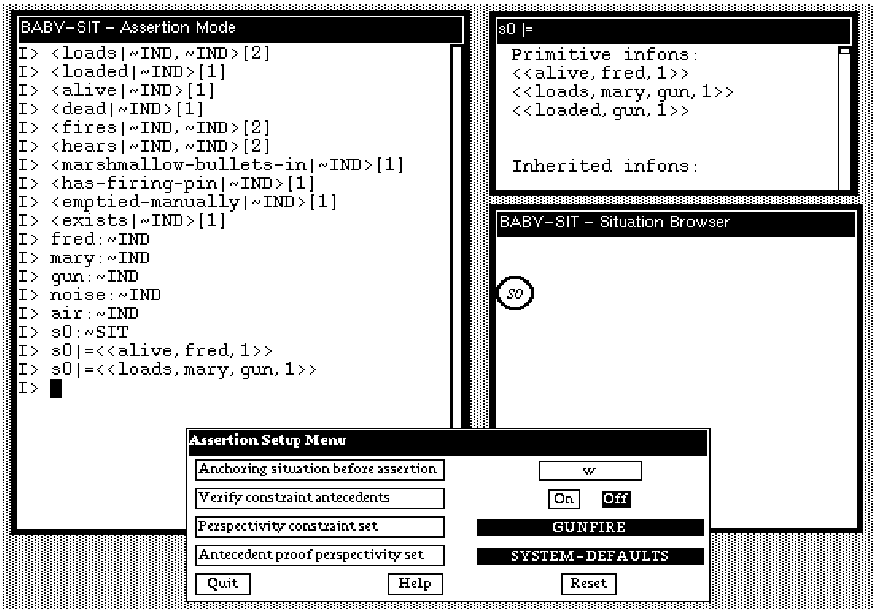

Figure 6: YSP: After Mary's loading the gun.

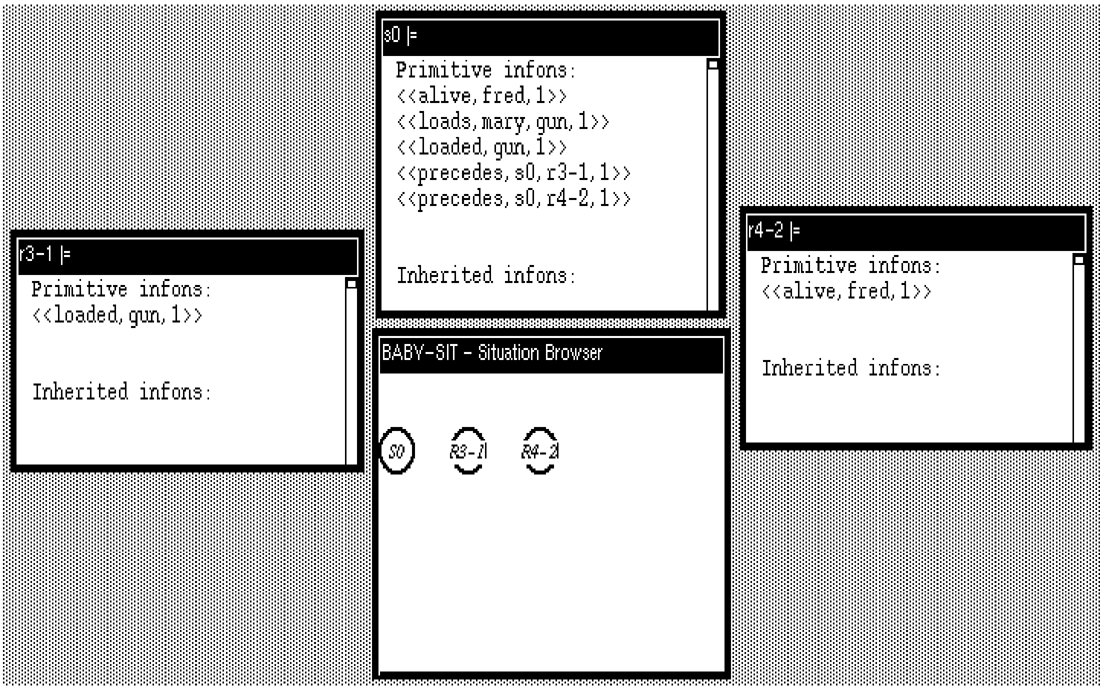

Figure 7: YSP: After the first iteration of forward chaining.

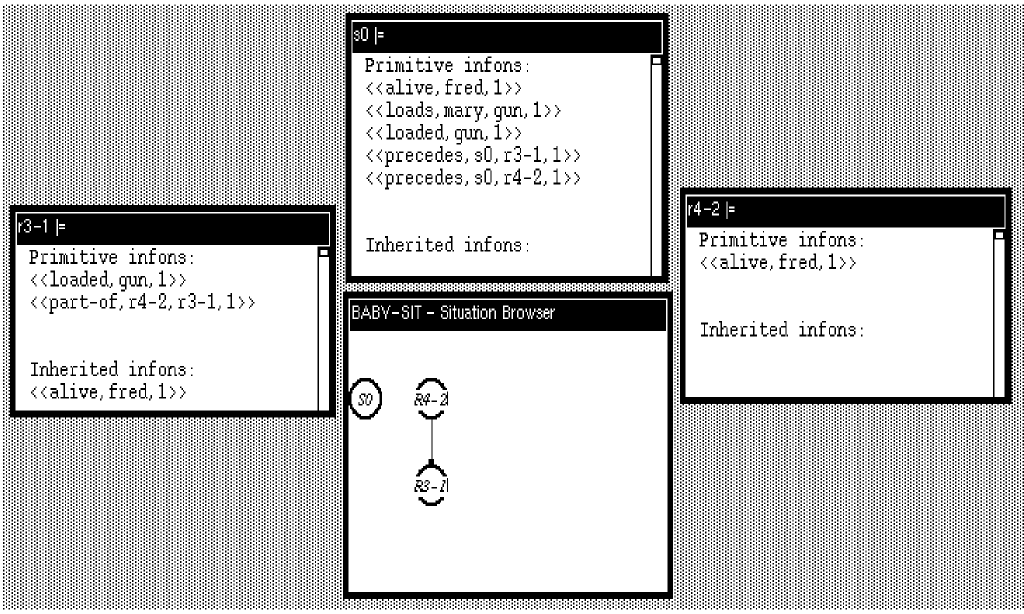

Figure 8: YSP: After the second iteration of forward chaining.

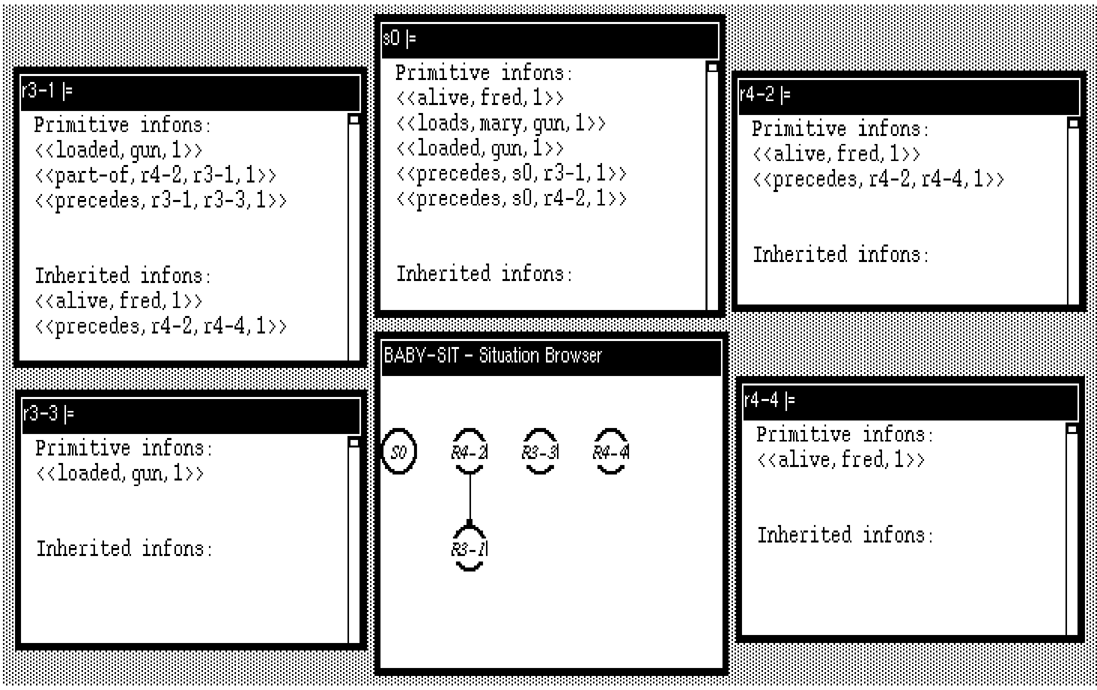

Figure 9: YSP: After the third iteration of forward chaining.

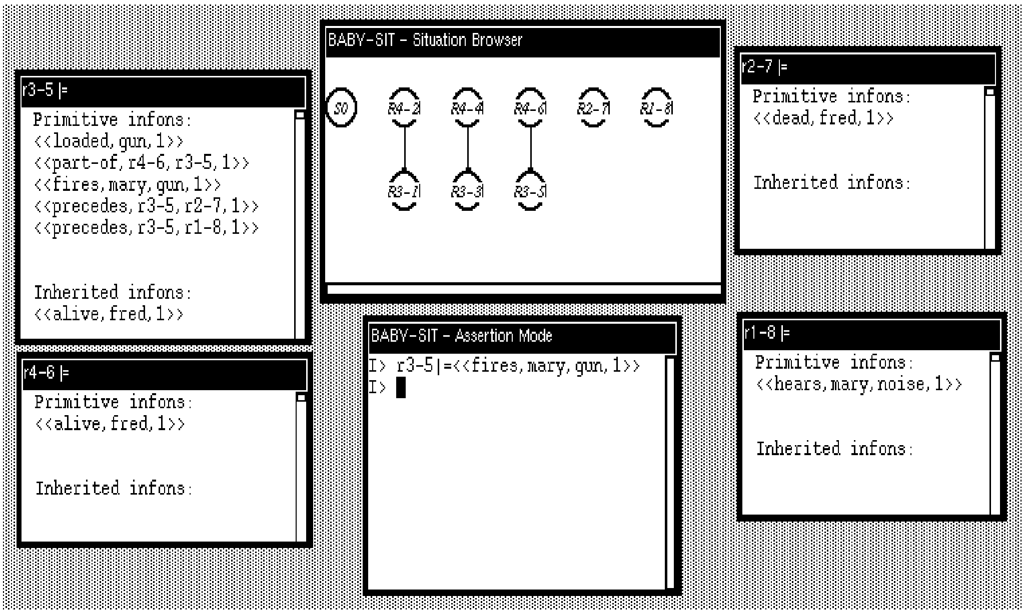

Figure 10: YSP: After Mary's firing the gun.

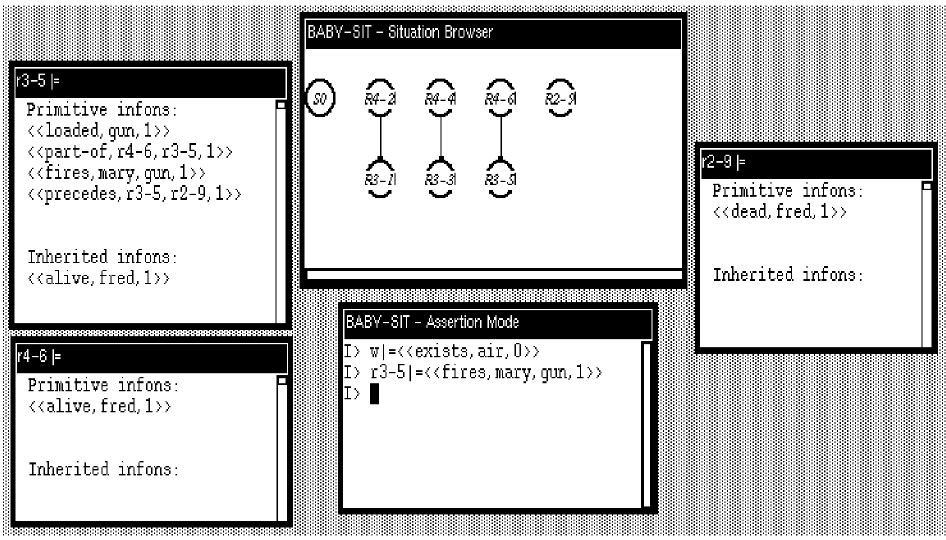

Figure 11: YSP: After Mary's firing the gun in vacuum condition.