Information-Oriented Computation with BABY-SIT

Erkan Tın and Varol Akman

1 Introduction

While situation theory and situation semantics (Barwise and Perry 1983) provide an appropriate framework for a realistic model-theoretic treatment of natural language, serious thinking on their 'computational' aspects has only recently started (Black 1993, Nakashima et al. 1988). Existing proposals mainly offer a Prolog- or Lisp-like programming environment with varying degrees of divergence from the ontology of situation theory. In this paper, we introduce a computational medium (called BABY-SIT) based on situations (Tın and Akman 1994a, Tın and Akman 1994b). The primary motivation underlying BABY-SIT is to facilitate the development and testing of programs in domains ranging from linguistics to artificial intelligence in a unified framework built upon situation-theoretic constructs.

2 Constructs for Situated Processing

Intelligent agents generally make their way in the world as follows: pick up certain information from a situation, process it, and react accordingly (Devlin 1991, Dretske 1981, Israel and Perry 1990). Being in a (mental) situation, such an agent has information about the situations it sees, believes in, hears about, etc. Awareness of some type of situation causes the agent to acquire more information about that situation as well as other situation types, and to act accordingly. Assuming the possession of prior information and knowledge of some constraints, the acquisition of an item of information by an agent can also provide the agent with an additional item of information.

Reaping information from a situation is not the only way an agent processes information. It can also act in accordance of the obtained information to change the environment. Creating new situations to arrive at new information and conveying information it already has to other agents are the primary functions of its activities.

In situation theory, abstraction can be captured in a primitive level by allowing parameters in infons. *Parameter-free infons* are the basic items of information about the world (i.e., 'facts') while parametric infons are the essential units that are utilized in a computational treatment of information flow.

To construct a computational model of situation theory, it is convenient to have available abstract analogs of objects. As noted above, by using parameters we can have parametric situations, parametric individuals, etc.

This yields a rich set of data types. Abstract situations can be viewed as models of real situations. They are set-theoretic entities that capture only some of the features of real situations, but are amenable to computation. We define abstract situations as structures consisting of a set of parametric infons.

Information can be partitioned into situations by defining a hierarchy between situations. A situation can be larger, having other situations as its subparts. For example, an utterance situation for a sentence consists of the utterance situations for each word forming the sentence. The *part-of* relation of situation theory can be used to build hierarchies among situations and the notion of nested information can be accommodated.

Being in a situation, one can derive information about other situations connected to it in some way. For example, from an utterance situation it is possible to obtain information about the situation it describes. Accessing information both via a hierarchy of situations and explicit relationships among them requires a computational mechanism. This mechanism will put information about situation types related in some way into the comfortable reach of the agent and can be made possible by a proper implementation of the *supports* relation, $\models$, of situation theory.

Constraints enable one situation to provide information about another and serve as links. When viewed as a backward-chaining rule, a constraint can provide a channel for information flow between types of situations, from the antecedent to the consequent. This means that such a constraint behaves as a 'definition' for its consequent part. Another way of viewing a constraint is as a forward-chaining rule. This enables an agent to alter its environment.

3 Computational Situation Theory

3.1 PROSIT

PROSIT (PROgramming in SItuation Theory) is a situation-theoretic programming language (Schütze 1991, Nakashima et al. 1988). PROSIT is tailored more for general knowledge representation than for natural language processing. One can define situation structures and assert knowledge in particular situations. It is also possible to define relations between situations in the form of constraints. PROSIT's computational power is due to an ability to draw inferences via rules of inference which are actually constraints of some type. PROSIT can deal with self-referential expressions (Barwise and Etchemendy 1987).

One can assert facts that a situation should support and queries can be posed about one situation from another, but the results will depend on where the query is made.

Constraints can be specified as forward-chaining constraints, backward-chaining constraints, or both. Backward-chaining constraints are activated at query-time while forward-chaining constraints are activated at assertion-time. For a constraint to be applicable to a situation, the situation must be declared to 'respect' the constraint. Constraints in PROSIT are about local facts within a situation rather than about situation types. That is, the interpretation of constraints does not allow direct specification of constraints between situations, only between infons within situations.

Situated constraints offer an elegant solution to the treatment of *conditional constraints* which only apply in situations that obey some condition. This is actually achieved in PROSIT since information is specified in the constraint itself. Situating a constraint means that it may only apply to appropriate situations and is a good strategy to enforce *background conditions*. However, it might be required that conditions are set not only within the same situation, but also between various types of situations.

Parameters, variables, and constants are used for representing entities in PROSIT. Variables match any expression in the language and parameters can be equated to any constant or parameter. That is, the concept of *appropriateness conditions* is not exploited in PROSIT. It is more useful to have parameters that range over various classes rather than to work with parameters ranging over all objects.

Given a parameter of some type (individual, situation, etc.), an anchor is a function which assigns an object of the same type to the parameter (Devlin 1991, pp. 52–63). Hence, parameters work by placing restrictions on anchors. However, there is no appropriate anchoring mechanism in PROSIT since parameters are not typed.

3.2 ASTL

Black's ASTL (A Situation Theoretic Language) is another programming language based on situation theory (Black 1993). ASTL is aimed at natural language processing. The primary motivation underlying ASTL is to figure out a framework in which semantic theories can be described and possibly compared. One can define in ASTL constraints and rules of inference over the situations.

ASTL ontology incorporates individuals, relations, situations, parameters, and variables. These form the basic terms of the language. Situations can contain facts which have those situations as arguments. Sentences in ASTL are constructed from terms in the language and can be constraints, grammar rules, or word entries. Constraints are actually backward-chaining constraints and are global. Thus, a new situation of the appropriate type need not have a constraint explicitly added to it. Grammar rules are yet another sort of constraints with similar semantics. Although one can define constraints between situations in ASTL, the notion of a background condition for constraints is not available. Similar to PROSIT, ASTL cares little about coherence within situations. This is left to the user's control.

Declaring situations to be of some type allows abstraction over situations to some degree. But, the actual means of abstraction over objects in situation theory, viz., parameters, carry little significance in ASTL.

As in PROSIT, variables in ASTL have scope only within the constraint they appear. They match any expression in the language unless they are declared to be of some specific situation type in the constraint. Hence, it is not possible to declare variables (nor parameters) to be of other types such as individuals, relations, etc. Moreover, ASTL does not permit a definition of appropriateness conditions for arguments of relations.

ASTL does not have a mechanism to relate two situations so that one will support all the facts that the other does. This might be achieved via constraints, but there is no built-in structure between situations.

3.3 Situation Schemata

Situation schemata have been introduced (Fenstad et al. 1987) as a theoretical tool for extracting and displaying information relevant for semantic interpretation from linguistic form. A situation schema is an attribute-value system which has a choice of primary attributes matching the primitives of situation semantics. The boundaries of situation schemata are flexible and, depending on the underlying theory of grammar, are susceptible to amendment.

Situation schemata can be adopted to various kinds of semantic interpretation. One could give some kind of operational interpretation in a suitable programming language, exploiting logical insights. But in its present state, situation schemata do not go further than being a complex attribute-value structure. They allow representation of situations within this structure, but do not use situation theory itself as a basis. Situations, locations, individuals, and relations constitute the basic domains of the structure. Constraints are declarative descriptions of the relationships holding between aspects of linguistic form and the semantic representation itself.

4 BABY-SIT

4.1 Computational Model and Architecture

The computational model underlying the current version of BABY-SIT consists of nine primitive domains: *individuals* (I), *times* (T), *places* (L), *relations* (R), *polarities* (O), *parameters* (P), *infons* (F), *situations* (S), and *types* (K). Each primitive domain carries its own internal structure:

- Individuals: Unique atomic entities in the model which correspond to real objects in the world.
- Times: Individuals of distinguished type, representing temporal locations.

- Places: Similar to times, places are individuals which represent spatial locations.
- Relations: Various relations hold or fail to hold between objects. A relation has argument roles which must be occupied by appropriate objects.
- Polarities: The 'truth values' 0 and 1.
- Infons: Discrete items of information of the form $\ll rel,\ arg_1,\ \ldots,\ arg_n,\ pol \gg$, where rel is a relation, arg_i, $1 \leq i \leq n$, is an object of the appropriate type for the ith argument role, and pol is the polarity.
- Parameters: 'Place holders' for objects in the model. They are used to refer to arbitrary objects of a given type.
- Situations: (Abstract) situations are set-theoretic constructs, e.g., a set of *parametric infons* (comprising relations, parameters, and polarities). A parametric infon is the basic computational unit. By defining a hierarchy between them, situations can be embedded via the special relation *part-of.* A situation can be either (spatially and/or temporally) *located* or *unlocated.* Time and place for a situation can be declared by *time-of* and *place-of* relations, respectively.
- Types: Higher-order uniformities for individuating or discriminating uniformities in the world.

The *model*, M, is a tuple $<I, T, L, R, O, P, F, S, K>$. This is shared by all components of the system. *Description* of a model, D_M, consists of a definition of M and a set of *constraints*, C. The *computational model* is then defined as a tuple $<D_M, A, A', U>$ where A is an anchor for parameters, A' is an assignment for variables, and U is an interpretation for D_M. A is provided by the anchoring situations while A' is obtained through unification. U is dynamically defined by the operational semantics of the computation. Each object in the environment must be declared to be of some type.

The architecture of BABY-SIT is composed of seven major parts: *programmer/user interface*, *environment*, *background situation*, *anchoring situations*, *constraint set*, *inference engine*, and *interpreter* (Figure 1).

The interface allows interaction of the user with the system. The environment initially consists of static situation structures and their relationships. These structures can be dynamically changed and new relationships among situation types can be defined as the computation proceeds. Information conveyance among situations is made possible by defining a *part-of* relation among them. In this way, a situation s can have information about another situation s' which is part of s. The background situation contains infons which are inherited by all situation structures in the environment. However, a situation can inherit an infon from the background situation only if it does not cause a contradiction in that situation.

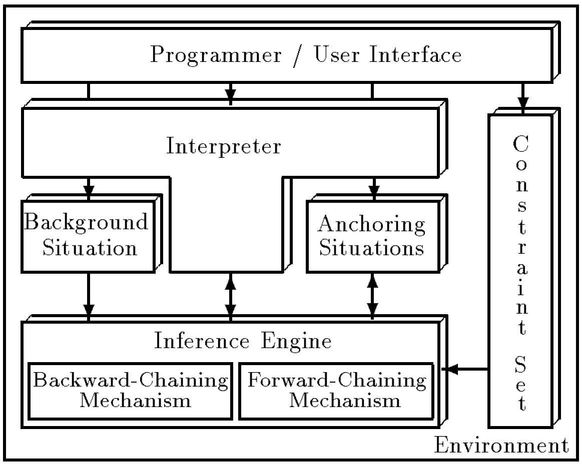

FIGURE 1 The Architecture of BABY-SIT.

A situation in the environment can be realized if its parameters are anchored to objects in the real world. This is made possible by the anchoring situations which allow parameters to be anchored to objects of appropriate types—an individual, a situation, a parameter, etc. A parameter must be anchored to a unique object by an anchoring situation. On the other hand, more than one parameter may be anchored to the same object. Restrictions on parameters assure anchoring of one parameter to an object having the same qualifications as the parameter.

In addition to the *part-of* relation among situations, constraints are potent means of information conveyance between situations. They link various types of situations. Constraints may be physical laws, linguistic rules, law-like correspondences, conventions, etc. In BABY-SIT, they are realized as forward-chaining constraints or backward-chaining constraints, or both. Assertion of a new object into BABY-SIT activates the forward-chaining mechanism. Once their antecedent parts are satisfied, consequent parts of the forward-chaining constraints are asserted into BABY-SIT, unless this yields a contradiction. In case of a contradiction, the backward-chaining mechanism is activated to resolve it. The interpreter is the central authority in BABY-SIT. Anchoring of parameters, evaluation of constraints, etc., are all controlled by this part of the system.

4.2 Modes of Computation

A prototype of BABY-SIT is currently being developed in KEE (Knowledge Engineering Environment) (KEE^{TM} 1993) on a SPARCstation^{TM}. Some of the available modes of computation in this evolving system are described below.

TABLE 1 Syntax of the Assertion Mode.

< *proposition* > ::=
<*situation-proposition*> | <*parameter-type-proposition*> |
<*situation/object-type-proposition*> | <*infon-proposition*>
<*type-of-type-proposition*> | <*relation-proposition*>

<*situation-proposition*> ::= <*constant*> “|=” <*infonic-set*>

<*parameter-type-proposition*> ::= <*parameter*> “=”
{<*basic-type*>, <*type-name*>,
<*restricted-parameter-type*>}

<*situation/object-type-proposition*> ::=
<*constant*> “:” {<*basic-type*>, <*type-name*>
<*type-abstraction*>}

<*infon-proposition*> ::= <*constant*> “=” <*infon*>

<*type-of-type-proposition*> ::=
<*type-name*> “=” {<*basic-type*>, <*type-abstraction*>}

<*relation-proposition*> ::= “<” <*relation*> [“|” <*type-specifier*>
(“,” <*type-specifier*>)*] “>”
“[” (<*digit*>)+ ”]”

<*type-specifier*> ::= <*basic-type*> | <*type-name*> |
“{” {<*basic-type*>, <*type-name*>}
(“,” {<*basic-type*>, <*type-name*>})*] “}”

<*type-abstraction*> ::=
“[” <*parameter*> “|” { <*constant*>, <*parameter*>}
“|=” <*infonic-set*> “]”

<*restricted-parameter-type*> ::= <*parameter*> “ ^ ” <*infonic-set*>

<*basic-type*> ::= “∼LOC” | “∼TIM” | “∼IND” | “∼REL” |
“∼SIT” | “∼INF” | “∼TYP” | “∼PAR” | “∼POL”

<*infonic-set*> ::= “{” <*infon*> (“,” <*infon*>)* “}” | <*infon*>

<*infon*> ::=
“≪” <*relation*> (“,” <*argument*>)* [“,” <*polarity*>] “≫”

<*relation*> ::= <*special-relation*> | <*constant*>

<*argument*> ::= <*constant*> | <*parameter*> |
<*basic-type*> | <*type-name*>

TABLE 2 Syntax of the Assertion Mode (continued).

<*polarity*> ::= "0" | "1"

<*constant*> ::= {<*digit*>, <*lower-case-letter*>}
({<*digit*>, <*lower-case-letter*>})*

<*parameter*> ::=
<*upper-case-letter*> ({<*upper-case-letter*>, <*digit*>})*

<*type-name*> ::=
"~" <*upper-case-letter*> ({<*upper-case-letter*>, <*digit*>})*

<*lower-case-letter*> ::= "a" | "b" | ... | "z" | "-"

<*upper-case-letter*> ::= "A" | "B" | ... | "Z"

<*digit*> ::= "0" | "1" | ... | "9"

4.2.1 Assertions

Assertion mode provides an interactive environment in which one can define objects and their types. There are nine basic types corresponding to nine primitive domains: ~IND (individuals), ~TIM (times), ~LOC (places), ~REL (relations), ~POL (polarities), ~INF (infons), ~PAR (parameters), ~SIT (situations), and ~TYP (types). For instance, if l is a place, then l is of type ~LOC, and the infon $\ll$*type-of*, ~LOC, l, 1$\gg$ is a fact in the background situation. Note that type of all types is ~TYP. For example, the infons $\ll$*type-of*, ~TYP, ~LOC, 1$\gg$ and $\ll$*type-of*, ~TYP ~TYP, 1$\gg$ are default facts in the background situation. The syntax of the assertion mode is the same as in (Devlin 1991) (cf. Tables 1 and 2).

Suppose *fred* is an individual, *can-think* is a relation, and *s0* is a situation. Then, these objects can be declared as:

I> *fred*: ~IND
I> *can-think*: ~REL
I> *s0*: ~SIT

The definition of relations includes the *appropriateness conditions* for their argument roles. Each argument can be declared to be from one or more of the primitive domains above. Consider *can-think* above. If we like it to have only one argument of type ~IND, we can write:

I> <*can-think* | ~IND> [1]

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 in the environment. Suppose we want to have a parameter E that denotes any thinking individual. This can be done by asserting:

I> E = IND1 ^ $\ll$*can-think*, IND1, 1$\gg$

IND1 is a default system parameter of type ~IND. E is considered as an object of type ~PAR such that if it is anchored to an object, say *fred*, then *fred* must be of type ~IND and the background situation (denoted by w) must support the infon $\ll$*can-think*, *fred*, 1$\gg$.

Parametric types are also allowed in BABY-SIT. They can be formed by obtaining a type from a parameter. This process is known as *(object-) type-abstraction*. Parametric types are of the form $[P \mid s \models I]$ where P is a parameter, s is a situation (i.e., a *grounding* situation), and I is a set of infons. The type of all thinking individuals can be defined as follows:

I> ~HUMAN = [IND1 | $w \models \ll$*can-think*, IND1, 1$\gg$]

~HUMAN is seen as an object of type ~TYP and can be used as a type specifier for declaration of new objects in the environment. For instance:

I> mary: ~HUMAN

yields an object, *mary*, which is of type ~IND such that the background situation supports the infon $\ll$*can-think*, *mary*, 1$\gg$.

Infons can be added into situations in BABY-SIT. The following sequence of assertions adds $\ll$*fires*, *mary*, *gun*, 1$\gg$ into *s0* (cf. Figure 3):

I> ~WEAPON = ~IND
I> *gun*: ~WEAPON
I> *fires*: ~REL
I> <*fires* | ~HUMAN, ~WEAPON>
I> *s0* $\models \ll$*fires*, *mary*, *gun*, 1$\gg$

4.2.2 Constraints

All possible types of constraints in situation theory can be classified as either *conditional* or *unconditional*. Conditional constraints can be applied to situations that satisfy some condition while unconditional constraints can be applied to all situations.

Variables in BABY-SIT are only used in constraints and query expressions, and have scope only within the constraint or the query expression they appear. A variable can match any object appropriate for the place or the argument role it appears in. For example, given the declaration above, variables ?X and ?Y in the proposition ?S $\models \ll$*fires*, ?X, ?Y, 1$\gg$ can only match objects of type ~HUMAN (i.e., of basic type ~IND where the background situation supports the fact that this object can think) and ~WEAPON (or simply ~IND), respectively.

A BABY-SIT constraint is of the form:

$$antecedent_1, \ldots, antecedent_n \ \{\Leftarrow, \Rightarrow, \Leftrightarrow\} \\ consequent_1, \ldots, consequent_m.$$

Each $antecedent_i$, $1 \leq i \leq n$, and each $consequent_j$, $1 \leq j \leq m$, is of the form *sit* $\{\models, \not\models\} \ll rel,\ arg_1,\ \ldots,\ arg_l,\ pol \gg$ such that *sit*, *rel*, and each arg_k, $1 \leq k \leq l$, can either be an object of appropriate type or a variable.

Each constraint has an identifier associated with it and must belong to a group of constraints. For example, the following is a forward-chaining constraint named R6 under the constraint group GUNFIRE:

GUNFIRE:
 R6:
  ?S1 $\models \ll$*loads*, ?M, ?G, 1$\gg \Rightarrow$?S1 $\models \ll$*loaded*, ?G, 1$\gg$

where ?S1, ?M and ?G are variables. ?S1 can only be assigned an object of type ∼SIT while ?M and ?G can have values of some type appropriate for the argument roles of *loads* and *loaded*. This constraint can apply in any situation. Hence, BABY-SIT constraints can be global. Constraints can also be situated. For example, R6 above can be rewritten to apply only in situation *sit1*:

GUNFIRE:
 R6:
  sit1 $\models \ll$*loads*, ?M, ?G, 1$\gg \Rightarrow$ *sit1* $\models \ll$*loaded*, ?G, 1$\gg$.

In BABY-SIT, conditional constraints come with a set of *background conditions* which must be satisfied in order for the constraint to apply. Background conditions are accepted to be true by default, unless stated otherwise. For example, to state that one hears noise upon firing a loaded gun, we can write:

GUNFIRE:
 R0:
  ?S1 $\models \{\ll$*loaded*, ?G, 1$\gg$, $\ll$*fires*, ?M, ?G, 1$\gg\} \Rightarrow$
  ?S2 $\models \ll$*hears*, ?M, *noise*, 1$\gg$
 UNDER-CONDITIONS:
  w: $\ll$*exists*, *air*, 1$\gg$.

Background conditions are, in fact, assumptions which are required to hold for constraints to be eligible for activation. R0 can become a candidate for activation only if it is the case that $w \not\models \ll$*exists*, *air*, 0$\gg$, i.e., if the absence of air is not known in the background situation. Hence, background conditions provide a contextual environment for constraints (Akman and Tın 1990).

A candidate forward-chaining constraint is activated whenever its antecedent part 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. Candidate backward-chaining constraints are activated either when a query is entered explicitly or is issued by the forward-chaining mechanism. In

```
xterm
GUNFIRE R6
?S1|=<<loads,?M,?G,1>> => ?S1|=<<loaded,?G,1>>
-

GUNFIRE R5
?S1|=<<situation-p,?S1,1>>, ?S1|/=<<successor-of,?S2,?S1,1>> =>
?S3|={<<situation-p,?S3,1>>,<<time-of,?T,?S3,1>>}, ?S1|=<<successor-of,?S3,?S1,1>>
-

GUNFIRE R4
?S1|={<<alive,?F,1>>,<<successor-of,?S2,?S1,1>>} => ?S2|=<<alive,?F,1>>
-
  w: <<exists,air,1>>,
?S1: <<fires,?M,gun,0>>

GUNFIRE R3
?S1|={<<loaded,?G,1>>,<<successor-of,?S2,?S1,1>>} => ?S2|=<<loaded,?G,1>>
-
?S1: {<<fires,?M,?G,0>>,<<emptied-manually,?G,0>>}

GUNFIRE R2
?S1|={<<alive,?F,1>>,<<loaded,?G,1>>,<<fires,?M,?G,1>>,<<successor-of,?S2,?S1,1>>} =>
?S2|=<<dead,?F,1>>
-
?S1: {<<has-firing-pin,?G,1>>,<<marshmallow-bullets-in,?G,0>>}

GUNFIRE R1
?S1|={<<loaded,?G,1>>,<<fires,?M,?G,1>>,<<successor-of,?S2,?S1,1>>} =>
?S2|=<<hears,?M,noise,1>>
-
  w: <<exists,air,1>>,
?S1: {<<has-firing-pin,?G,1>>,<<marshmallow-bullets-in,?G,0>>}
```

FIGURE 2 BABY-SIT Constraints for the YSP.

BABY-SIT, constraints between situation types as well as between infons of a situation can be easily modeled. Grouping of constraints enables one to view the world and make inferences from different perspectives. Figure 2 illustrates the axiomatization of the so-called *Yale Shooting Problem* (YSP)[1] with BABY-SIT constraints.

4.2.3 Querying

The query mode enables one to issue queries about situations. There are several possible actions which can be further controlled by the user:

- Replacing each parameter in the query expression by the corresponding individual if there is a possible anchor, either partial or full, provided by the given anchoring situation for that parameter.
- Returning solutions. (Their number is determined by the user.)
- Displaying a solution with its parameters replaced by the individuals to which they are anchored by the given anchoring situation.
- For each solution, displaying infons anchoring any parameter in the solution to an individual in the given anchoring situation.
- Displaying a trace of anchoring of parameters in each solution.

The computation upon issuing a query is done either by direct querying through situations or by the application of backward-chaining constraints. A situation, s, *supports* an infon if the infon is either explicitly asserted to hold in s, or it is supported by a situation s' which is part of s, or it can

[1] At some point in time, a person (Fred) is alive and a loaded gun, after waiting for a while, is fired at Fred. What are the results of this action? (Shoham 1988)

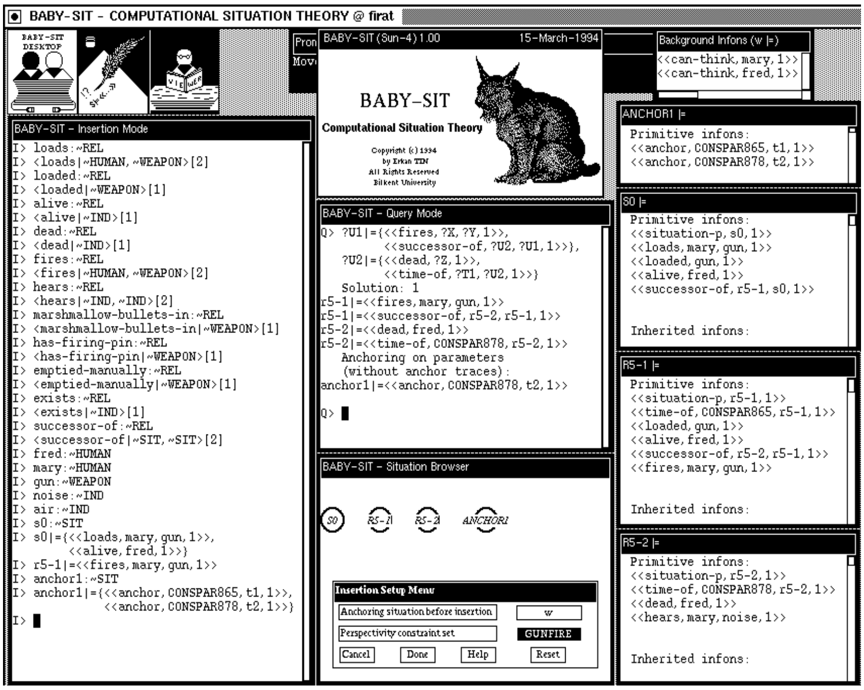

FIGURE 3 Solution of the YSP in BABY-SIT.

be proven to hold by application of backward-chaining constraints. The syntax of the query expressions is given in Table 3. Given *anchor1* as the anchoring situation, a query and the system's response to it are as follows (cf. Figure 3):

Q> $?\text{U1} \models \{\ll \textit{fires}, ?\text{X}, ?\text{Y}, 1\gg, \ll \textit{successor-of}, ?\text{U2}, ?\text{U1}, 1\gg\}$,
$?\text{U2} \models \{\ll \textit{dead}, ?\text{Z}, 1\gg, \ll \textit{time-of}, ?\text{T1}, ?\text{U2}, 1\gg\}$

Solution: 1

$\text{r5-1} \models \ll \textit{fires}, \textit{mary}, \textit{gun}, 1\gg$

$\text{r5-1} \models \ll \textit{successor-of}, \text{r5-2}, \text{r5-1}, 1\gg$

$\text{r5-2} \models \ll \textit{dead}, \text{fred}, 1\gg$

$\text{r5-2} \models \ll \textit{time-of}, \text{CONSPAR878}, \text{r5-2}, 1\gg$

Anchoring on parameters (without anchor traces):

$\text{anchor1} \models \ll \textit{anchor}, \text{CONSPAR878}, \text{t2}, 1\gg$.

In addition to query operations, a special operation, *oracle*, is allowed in the query mode. An *oracle* is defined over an object and a set of infons (*set of issues*) (Devlin 1991). The oracle of an object enables one to chronologically view the information about that object from a particular perspective provided by the given set of infons. One may consider oracles as 'histories'

TABLE 3 Syntax of the Query Mode.

< *query* > ::= <*situation-query*> | <*oracle-query*>

<*situation-query*> ::=
<*situation*> {“|=”, “|=/=”} <*query-infonic-set*>
(”,” <*situation*> {“|=”, “|/=”} <*query-infonic-set*>)*

<*oracle-query*> ::=
<*constant*> “=” “@” “(” <*constant*> “)” [<*issue-set*>]

<*situation*> ::= <*constant*> | <*query-variable*>

<*issue-set*> ::= “{” <*issue-infon*> (“,” <*issue-infon*>)* “}”

<*query-infonic-set*> ::=
“{” <*query-infon*> (“,” <*query-infon*>)* “}” | <*query-infon*>

<*query-infon*> ::=
“≪” {<*relation*>, <*query-variable*>}
(“,” {<*argument*>, <*query-variable*>})* “,” <*polarity*> “≫”

<*issue-infon*> ::=
“≪” <*relation*> (“,” <*argument*>)* “,” <*polarity*> “≫”

<*query-variable*> ::= “?” <*parameter*>

of specific objects. Given an object and a set of issues, BABY-SIT anchors all parameters in this set of issues and collects all infons supported by the situations in the system under a specific situation, thus forming a ‘minimal’ situation which supports all parameter-free infons in the set of issues.

5 Concluding Remarks

BABY-SIT accommodates the basic features of situation theory and, compared with existing approaches, enhances these features (cf. Table 4). Situations are viewed at an abstract level. This means that situations are sets of parametric infons, but they may be non-well-founded. Parameters are place holders, hence they can be anchored to unique individuals in an anchoring situation. A situation can be realized if its parameters are anchored, either partially or fully, by an anchoring situation. Each relation has ‘appropriateness conditions’ which determine the type of its arguments. Situations (and hence infons they support) may have spatio-temporal dimensions. A hierarchy of situations can be defined both statically and dynamically. A built-in structure allows one situation to have information about another which is part of the former. Grouping of situations provides a computational context. Partial nature of the situations facilitates com-

TABLE 4 Tableau Comparison of Existing Approaches.

Constraint Type	PROSIT	ASTL	BABY-SIT
Nomic	$\surd$	$\surd$	$\surd$
Necessary	$\surd$	$\surd$	$\surd$
Conventional	–	–	?
Conditional	–	–	$\surd$
Situated	$\surd$	–	–

Computation	PROSIT	ASTL	BABY-SIT
Unification	$\surd$	$\surd$	$\surd$
Type-theoretic	–	–	$\surd$
Coherence	–	–	$\surd$
Forward-chaining	$\surd$	–	$\surd$
Backward-chaining	$\surd$	$\surd$	$\surd$
Bidirectional-chaining	$\surd$	–	$\surd$

Miscellaneous Features	PROSIT	ASTL	BABY-SIT
Circularity	$\surd$	$\surd$	$\surd$
Partiality	$\surd$	$\surd$	$\surd$
Parameters	?	?	$\surd$
Abstraction	?	?	$\surd$
Anchoring	?	?	$\surd$
Appropriateness conditions	–	–	$\surd$
Information nesting	$\surd$	$\surd$	$\surd$
Set operations	$\surd$	–	–
Oracles	–	–	?

LEGEND: $\surd$: available, – : not available, and ? : partially/conceptually available.

putation with incomplete information. Constraints can be violated. This aspect is built directly into the computational mechanism: a constraint can be applied to a situation only if it does not lead to an incoherence.

With these features, BABY-SIT provides a programming environment incorporating situation-theoretic constructs for various domains of application including artificial intelligence and computational linguistics. A preliminary study towards employing BABY-SIT in the resolution of pronominal anaphora has been recently initiated (Tın and Akman 1994c).

References

Akman, V., and E. Tın. 1990. What is in a Context? In *Proceedings of the 1990 Bilkent International Conference on New Trends in Communication, Control, and Signal Processing*, ed. E. Arıkan. 1670–1676. Amsterdam, Holland: Elsevier.

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

Barwise, J., and J. Perry. 1983. *Situations and Attitudes.* Cambridge, Massachusetts: MIT Press.

Black, A. W. 1993. *An Approach to Computational Situation Semantics.* Doctoral dissertation, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, U.K.

Devlin, K. 1991. *Logic and Information.* Cambridge, U.K.: Cambridge University Press.

Dretske, F. 1981. *Knowledge and the Flow of Information.* Cambridge, Massachusetts: MIT Press.

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

Israel, D., and J. Perry. 1990. What is Information? In *Information, Language, and Cognition*, ed. P. P. Hanson. 1–28. Vancouver, Canada: The University of British Columbia Press.

KEE^{TM}. 1993. (Knowledge Engineering Environment) Software Development System, Version 4.1. IntelliCorp, Inc., Mountain View, California.

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

Schütze, H. 1991. The PROSIT Language, Version 0.4. CSLI Memo, Center for the Study of Language and Information, Stanford University, Stanford, California.

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

Tın, E., and V. Akman. 1994a. BABY-SIT: Towards a Situation-Theoretic Computational Environment. In *Current Issues in Mathematical Linguistics*, ed. C. Martín-Vide. North-Holland Linguistic Series, Vol. 56, 299–308. Amsterdam, Holland: North-Holland.

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

Tın, E., and V. Akman. 1994c. Situated Processing of Pronominal Anaphora. In *KONVENS '94, 2. Konferenz, Verarbeitung natürlicher Sprache*, ed. H. Trost. 369–378. Informatik Xpress 6. Berlin, Germany: Springer Produktions-Gesellschaft.