Telos: Features and Formalization

Manolis Koubarakis, John Mylopoulos
Martin Stanley, and Alex Borgida

FORTH/CSI/TR/1989/018

Computer Science Institute, Heraklio, Crete
Foundation of Research and Technology – Hellas
Telos: Features and Formalization

Manolis Koubarakis, John Mylopooulos
Martin Stanley, and Alex Borgida

February 1989

FORTH/CSI/TR/1989/018
Technical Report Series
Computer Science Institute, FORTH †
P.O.Box 1385, Heraklio, Crete, 711-10 Greece
Tel.: +30.81.229302, Fax: +30.81.229342, Telex: 262389 CCI GR
E-mail – UUCP: mcvax!ariadne!csi

This work was partially supported by the ESPRIT projects LOKI (No 107) and DAIDA (No 892).
† "Foundation of Research and Technology – Hellas," formerly “Research Center of Crete” (RCC).
Chapter 1

Introduction

Telos is a knowledge representation language which provides representational facilities for concepts and assertions about an intended subject matter, structuring mechanisms to facilitate the construction of knowledge bases, integrated facilities for representing and reasoning about temporal knowledge, and inference facilities for query evaluation as well as integrity checking.

The language was developed with a particular class of applications in mind, namely requirements modelling for information systems. Such requirements models are built during the early phases of development of an information system, when one is doing domain analysis. They are intended to capture, among other things, the states and activities in the environment within which the system will eventually function. A requirements model therefore can be used to find the relevant information that needs to be stored by the proposed system, as well as the legal and desired ways of manipulating it. It is also useful in the design and implementation of the information system by providing the knowledge needed to make decisions about the data structures and algorithms to be used. Finally, it serves as a means of communication between the designer and end users who wish to know what does the information stored in the system mean vis-a-vis its intended subject matter. There is considerable literature on requirements modelling in general [TSE77] [Com85] and, more specifically, the world modelling view of requirements modelling [Gres84], [DHL+86]. In [BJM+87] an information system development environment is described which adopts Telos as its requirements modelling language.

Practice indicates that a requirements model generally involves very many descriptions of entities and activities appearing in the organizational environment of an information system. For example, a requirements model may contain descriptions of who works within the organization, what activities he participates in, what effect these activities have on the state of the organization, what activities will the intended information system perform and how these are affected by its environment.

The nature of requirements models has led to a set of design goals for Telos. Firstly, the language must support structuring mechanisms, to facilitate the construction of large knowledge bases. Secondly, because time and change are such ubiquitous aspects of reality, Telos must provide primitive constructs for representing and reasoning with temporal knowledge. Since the proposed descriptions will involve complex logical assertions, we wish to provide ways to facilitate their expression; extensibility of the language – allowing for the definition of new metatypes which are instantiated during the construction of particular requirements models – is just one way in which

---

1Hence the name Telos from the Greek word τέλος which means end; the object aimed at in an effort; purpose. The word teleology has τέλος as one of its roots.
this can be supported. Finally, as with all language design, uniformity and orthogonality were
given high priority throughout, in an attempt to make the language easier to learn and apply.

1.1 Genealogy

Telos is the result of a long-term effort to apply knowledge representation techniques to the early
phases of software development, through the design of new languages, methodologies and tools.
Our first step in that direction was the language RML, described in [Gre84]. RML, like Telos, pro-
vides an object-oriented representational framework which supports generalization, classification
and aggregation. Moreover, all objects are classified as \textit{entities, activities or assertions}, and all
attributes associated with an object are classified into \textit{attribute categories}. Thus, generic entities
(entity classes) have \textit{producer}, \textit{updater} and \textit{consumer} attributes which have as values activities
that create, update and destroy instances of these entities. Likewise, activity classes have \textit{input},
\textit{output} and \textit{control} attributes, which have as values entities that are consumed, produced or need
to be present during the execution of instances of these activities. Turning to time, RML is based
on a linear, dense model of time points, which pervades all descriptions.

An effort to revise and extend RML was initiated in 1985, thus leading to a new language named
CML; this was described in [Sta86] and [Kou88]. An early version of CML was implemented in 1986
[GS86] while implementations of more recent versions of it are described in [Kou88] and [JIR88].
Telos is a less ambitious and more clearly defined dialect of CML as described in [Kou88]. The
main advantages of CML and Telos over RML are its more powerful object-oriented framework, its
treatment of time, and the more careful use of assertions for consistency checking and for making
inferences. An implementation of Telos is described in [TK].

Much of the work on both RML and Telos, née CML, has been carried out at the University
of Toronto as part of the Taxis project. The project has addressed, among other things,
the development of Taxis, which is a programming language for information systems, as well as
the development of a methodological framework based on languages such as Taxis and Telos for
building software systems. A recent overview of the project and the rationale behind it appears
in [MBG+87].

1.1.1 Related Work

Requirements modelling came of age during the seventies as an essential phase in software develop-
ment. Languages like PSL/PSA, [TH77], RSL-SREM [BBD77] and SADT\footnote{SADT is a trademark of Softech, Inc., Waltham Mass.} [RS77] provided tools
and methodologies for describing software systems. By-and-large, however, the above languages
were used to construct abstract (function- or automata-theoretic) descriptions of the behavior of a
system to be built. SADT is an exception to this rule in that it aspires "...to bind up structure and
communication units of thought expressed in any other chosen language..." [RS77]. The result is
a graphical notation that allows one to structure and annotate concepts using natural language for
the annotations. SADT influenced heavily RML, which was originally conceived as a formal ver-
sion of SADT. Perhaps the work closest in spirit to Telos is ERAE [DHL*86]). Like Telos, ERAE
adopts a semantic network representation for the declaration of concepts. Statements about the
subject matter, however, are expressed in terms of logical assertions.

The other family of languages relevant to Telos are knowledge representation languages pro-
posed over the years. Semantic network formalisms and frame-based representations, particularly
PSN [LM79], have influenced the object-oriented framework of Telos, though Telos takes a rather
conservative stand on default inferences. Knowledge representation languages like Omega [AS81], KRYPTON [BFL83], and KEE\(^3\) [FK65] offer some combination of an object-oriented framework and an assertional sublanguage and are perhaps closest to this work.

Omega, unlike Telos, treats the knowledge base units as (indefinite) descriptions which are structured in ways that are analogous to generalization, aggregation and classification. Powerful operations are provided for the construction of new descriptions, and assertions are viewed as special descriptions that refer to "true" or "false". Reasoning with respect to an Omega knowledge base is viewed as a generalized subsumption operation which attempts to match a query against each description in the knowledge base, proceeding from more general to more specialized ones. Telos does not offer a classification operation (subsumption), rather it relies more on instantiation as the fundamental knowledge base operation. Omega also comes close to offering a framework for representing time through its notion of viewpoint\(^4\).

The design of KRYPTON is based on the premise that terminological knowledge, i.e., definitions of terms relevant to a subject matter, is different from assertional knowledge, which asserts facts about that subject matter. Accordingly, different facilities are offered for the assertion and query of the two types of knowledge. The terminological component of a knowledge base has a semantic network-like structure and comes with a subsumption operation for matching a new term to existing ones. The assertional component of the knowledge base, on the other hand, comes with a powerful theorem prover for query evaluation. Telos adopts the abstract data type approach of KRYPTON in the definition of its knowledge base operations, but not KRYPTON's rigid separation of terminological and assertional knowledge.

KEE offers a number of procedural features, including demons, methods etc., which provide hooks to LISP for the KEE user; Telos avoids this uncleanliness. KRYPTON is different from Telos in its distinction of terminological and assertional knowledge, also in its neat (but limited) methodology for integrating terminological knowledge into the theorem prover which reasons with respect to the assertional knowledge base.

A final area of related research includes formal specification languages, such as [BG79], [Was84] and [Heh84]. A major point of difference between Telos and specification languages is the intended subject matter, which in the case of specification languages is programs and programming knowledge rather than (real world) organizational environments and embedded information systems. Accordingly, such languages focus on clean mathematical notations, monotonic proof theories and an appropriate programming methodology. Similarly, the Plan Calculus described in [Ric82], also focuses on programming knowledge and emphasizes different features than those stressed here.

1.1.2 Features

As indicated earlier, Telos adopts an object-oriented representational framework, based on ideas from semantic oneworks and frame-based representations. A Telos knowledge base structure consists of propositions grouped into classes and related to other propositions through attributes, which are also propositions. Thus individuals (entities, objects, concepts, nodes) and attributes (relationships, links) are both first class citizens in the adopted representational framework thus making the language fully extensible.

In addition, Telos supports a number of structuring mechanisms which have been used by knowledge representation languages as well as semantic data models\(^5\) allowing the designer of

---

\(^3\) KEE is a trademark of IntelliCorp.

\(^4\) This is more properly understood as a variant of contents and partitions proposed by early AI languages and semantic network formalisms.

\(^5\) See [HKST], [Bor83a], [AL86], [ABSS] for overviews of the field.
Chapter 2

An Amateur’s Guide to Telos

This chapter introduces Telos and gives a comprehensive example (the modelling of an IFIP working conference [Gre84]) illustrating its features and use.

2.1 The Representational Framework

By representational framework we mean the structure provided by Telos for representing knowledge. This structure is a generalization of graph-theoretic data structures used in semantic networks and structurally object-oriented representations such as those discussed in [Fin79] and [SB83] respectively. The representational frameworks that have influenced most of this work are those of FSN, a knowledge representation language that makes a valiant effort to gracefully integrate semantic nets and procedures [LM78] and Taxis, a design language for information systems that adopts many of the representation principles of FSN [MBW80].

2.1.1 Propositions

Every Telos knowledge base is a collection of propositions. These are atomic structures representing elementary statements such as:

the chairman of the conference committee in December 1986 was McCarthy.

Formally, propositions are quadruples with components source, label, destination, and duration which are themselves propositions. For example, the above statement is represented by the proposition \texttt{chpn\_is\_McCarthy} with components

\texttt{chpn\_is\_McCarthy}:= \langle \texttt{conf\_committee}, \texttt{chpn}, \texttt{McCarthy}, 1986/12 \rangle.

Here \texttt{conf\_committee}, \texttt{chpn}, \texttt{McCarthy}, and 1986/12 are also propositions, as defined below:

\texttt{\langle conf\_committee, ..., conf\_committee, ... \rangle}
\texttt{\langle chpn, chpn, chpn, ... \rangle}
\texttt{\langle McCarthy, ..., McCarthy, ... \rangle}
\texttt{\langle 1986/12, ..., 1986/12, 1986/12 \rangle}

Components of a proposition that are not of interest to us are denoted by the ellipsis (\ldots). We can assume that they are propositions generated internally by the Telos system.
The functions from, label, to, and when take as argument a proposition and return the source, label, destination, and duration of the proposition respectively.

Propositions can be divided into two disjoint categories: individuals (entities, nodes, objects, etc. in other formalisms) and attributes (relationships, links, etc.). Furthermore, we can distinguish the following classes of individuals:

- **Simple individuals** are propositions representing entities in the application domain; their distinguishing characteristic from a structural point of view is that they are self-referential in that they have themselves as source and destination e.g., McCarthy or conf.committee above. Integers and strings are also considered to be individuals. For example, the number 2 can be represented by:

  \[<2, \ldots, 2, \ldots>\]

- **Labels** are a special case of individuals defined to have themselves as labels as well. For example, chpn is represented as:

  \[<chpn, chpn, chpn, \ldots>\]

- **Time intervals**, again, constitute a special case of individuals which have themselves as duration. For example, the time interval 1986 will be represented as

  \[<1986/12, \ldots, 1986/12, 1986/12>\]

So far we have demonstrated that Telos knowledge bases are essentially collections of propositions. In the next sections, we introduce a number of facilities for structuring such knowledge bases.

### 2.1.2 The Instantiation Hierarchy

The first structuring principle we introduce is classification/instantiation. According to this, every proposition must be an instance of one or more generic propositions called *classes*. The intent is to allow the user to group similar propositions under a common generic description. This description can be thought of as serving a role analogous to that of a "frame" [Min75], though Telos downplays defaults and default reasoning. For example, we can define the class *committee* and its instance *conf.committee* as follows:

```
TELL CLASS committee
  IN S_Class
  WITH
    attribute
      charter: task_description
    necessary
      mbrs: potential_participant;
      chair: potential_participant
END
```
TELL TOKEN conf_committee
IN committee
WITH
  charter
  char: task1
  chair
  chpn: McCarthy;
  mbrs 
  : McDermott;
  : Allen;
  : Hayes
END

The above is an example of defining a new proposition using the command TELL. This command can also be used to attach new information to propositions already present in the knowledge base. Its syntax will become clearer to the reader as we present more detailed examples. For the time being, the following remarks should suffice:

- The keywords CLASS and TOKEN distinguish between defining a class and defining a token.
- There is a compulsory IN clause giving the list of classes of which the defined proposition is an instance. If this proposition is a token the clause can be omitted.
- There is an optional WITH clause attaching a number of attributes to the defined proposition.

Besides individuals, attributes can also be grouped into attribute classes. For example, the generic fact that "the chair of a committee is a potential participant" can be represented by the proposition

chpn_is_potential_part:= <committee, chair, potential_participant, ...>.

Now, the token proposition chpn_is_McCarthy denoting that McCarthy was the chairman of our committee in 1986 can be an instance of this attribute class. This instantiation relation is depicted as a semantic net in figure 2.1. Note that, ovals denote token propositions while rectangles denote classes. Attribute relationships are depicted as arrows with open heads while instantiation relationships as arrows with closed heads.

The following constraint is imposed on instantiation. It is essentially an extended form of the usual type constraints required in similar representational frameworks.

**Instantiation Constraint:** If a proposition p is an instance of a proposition q for a time period t then from(p) must be an instance of from(q) for a time period that contains t, to(p) must be an instance of to(q) for a time period that contains t and when(p) must be contained in when(q).

For individual propositions this constraint holds if and only if when(p) is contained in when(q) since from(p)=to(p)=p and from(q)=to(q)=q.

Since classes are themselves propositions they must be instances of other, more generic, classes. Therefore instantiation defines an infinite dimension along which propositions can be classified, as in FSN [LM79] and other semantic networks. Each one of the levels of the dimension contains the following:
Figure 2.1: A simple Telos semantic net

Level 0: It includes all tokens (i.e., objects having no instances) such as:

\[
\text{conf\_committee} := \langle \text{conf\_committee}, \ldots, \text{conf\_committee}, \ldots \rangle \\
\text{chpn\_is\_McCarthy} := \langle \text{conf\_committee}, \text{chpn}, \text{McCarthy, 1986} \rangle.
\]

Level 1: It includes all simple classes (i.e., objects having only tokens as instances) such as:

\[
\text{committee} := \langle \text{committee}, \ldots, \text{committee}, \ldots \rangle \\
\text{chpn\_is\_potential\_part} := \langle \text{committee, chair, potential\_participant}, \ldots \rangle
\]

Level 2: It includes all \textit{m}_1\textit{\_classes} i.e., these having only simple classes as instances.

Level 3: It includes all \textit{m}_2\textit{\_classes} i.e., these having only \textit{m}_1\textit{\_classes} as instances.

\vdots

Level \omega: It includes all \omega\textit{-classes} (to be introduced below). \omega\textit{-classes} can have instances from any level.

Classes after level 1 are called \textit{metaclasses}.

The following example defines the propositions \texttt{paper\_class}, \texttt{paper}, and \texttt{krypton} one in each of the first three levels of the dimension:

```
TELL CLASS paper\_class
  IN M1\_Class
  WITH
    attribute
      card: 0..100;
    paper\_descr: Class
END
```
TELL CLASS paper
   IN S_Class, paper_class
   WITH
      card
      : 40
      paper_dscr
      author: Author;
      title: String;
      subject: subject_area;
      date: TimeInterval
END

TELL TOKEN krypton
   IN paper
   WITH
      author
      : Brachman
      title
      : 'Krypton: A hybrid representation system'
      date
      : 1986/12
      subject
      : kr
END

In order to deal with instantiation in an orderly manner, we assume the existence of the following built-in classes; one at each level of the instantiation dimension. The extension of each one of these classes is the set of all propositions in the previous level of the dimension.

Token: It is a simple class and has all tokens as instances.
S.Class: It is an ml.class and has all simple classes, including Token, as instances.
M1.Class: It is a m2.class and has all ml.classes, including S.Class, as instances.

The following ω-classes are also offered to facilitate a most general structuring of the knowledge base.

Proposition: It has all propositions as instances including itself.
Class: It has all classes as instances including itself.
Individual: It has all individuals as instances including itself.
Attribute: It has all attributes as instances.
IndividualClass: It has all individual classes as instances including itself.
AttributeClass: It has all attribute classes as instances including itself.
OmegaClass: It has all ω-classes as instances including itself.

The propositions Proposition, Class, Individual, IndividualClass, and OmegaClass are individuals, therefore:
Proposition:= <Proposition, ..., Proposition, Alltime>
Class:= <Class, ..., Class, Alltime>
Individual:= <Individual, ..., Individual, Alltime>
IndividualClass:= <IndividualClass, ..., IndividualClass, Alltime>
MegaClass:= <MegaClass, ..., MegaClass, Alltime>

Attribute and AttributeClass are defined as follows:

Attribute:= <Proposition, attribute, Proposition, Alltime>
AttributeClass:= <Class, attribute, Class, Alltime>.

Alltime is an infinite time interval. It has the property that every other time interval occurs during it. The time component of all the propositions representing built-in objects or relationships between them is Alltime.

Now let us find what the components of the class Token should be. Token propositions are used to represent elementary facts about the domain. For example, the fact “the author of the paper on Krypton is Brachman” is represented by the token proposition:

author_is_B:= <krypton, ..., Brachman, ...>.

The components of the above token proposition are tokens but this is not the case for every token proposition\(^1\). For example, the attribute token proposition representing the elementary fact “the cardinity of the class of papers is 40” can be represented by the proposition

card_is_40:= <paper, ..., 40, ...>.

Therefore Token must be the proposition:

Token:= <Proposition, token, Proposition, Alltime>

For similar reasons, the S_Class and M1_Class are the following propositions:

S_Class:= <Class, s_Class, Class, Alltime>
M1_Class:= <Class, m1_class, Class, Alltime>

The attribute class Necessary used above has the following components:

Necessary:= <Class, necessary, Class, Alltime>

2.1.3 The Isa Hierarchy

Orthogonally to the instantiation dimension, classes can be specialized/generalized through isa hierarchies [BMW84]. For this purpose, we introduce an optional ISA clause in the TELL command that can be used to declare the superclasses of the defined class. For example, since conference papers are papers, we might define the class conf_paper as follows:

TELL CLASS conf_paper
    IN S_Class, paper_class
    ISA paper
    WITH
        necessary
            status: paper_status
END

\(^1\)However, if the components of a proposition are tokens then it is token as well.
The following constraints are imposed on specialization:

**First Specialization Constraint:** Only classes residing on the same level of the instantiation dimension can be IsA related and they do so when they concurrently exist.

**Second Specialization Constraint:** If a class \( P \) is a specialization of a class \( Q \) and there are attribute classes \( <P, 1, V_1, t_1> \) and \( <Q, 1, V_2, t_2> \), defined for \( P \) and \( Q \) respectively, where \( t_1 \) and \( t_2 \) are overlapping time intervals and \( V_1 \) is different than \( V_2 \) then \( V_1 \) should be a specialization of \( V_2 \) over a time interval that contains the overlap of \( t_1 \) and \( t_2 \).

These constraints are again extended forms of constraints imposed on specialization by related semantic data models.

By default, every class is a specialization of the built-in class (Token, S.Class, M1.Class etc.) residing on the same level of the instantiation hierarchy.

If one class is a specialization of another, then every instance of the first is an instance of the second. More formally, we have the following postulate:

**Specialization Postulate:** If a class \( P \) is a specialization of a class \( Q \) for the time period \( t_1 \), and \( p \) is an instance of \( P \) for the time period \( t_2 \) then \( p \) is an instance of \( Q \) for the time period common to both \( t_1 \) and \( t_2 \).

This postulate implies that instances of a class automatically inherit attributes from any generalizations of this class\(^2\).

Returning to the built-in classes defined earlier, all s.classes are specializations of Token, all m1.classes are specializations of M1.Class, and so on. In addition:

- AttributeClass isA Class, AttributeClass isA Attribute
- Attribute isA Proposition, Class isA Proposition
- IndividualClass isA Class, IndividualClass isA Individual
- Individual isA Proposition, OmegaClass isA IndividualClass

Figure 2.2 gives the semantic net corresponding to the \( \omega \)-classes we have presented so far. Specialization is denoted by a double arrow. Note also that the instantiation links to OmegaClass are not shown in the picture.

### 2.1.4 The Attribute Mechanism of Telos

Now we have all the machinery required to explain the attribute mechanism of Telos. The existence of an attribute class enables the creation of attribute tokens for the instances of its source component. These attribute tokens are of course instances of the attribute class. Thanks to the instantiation constraint, they are also required to have instances of the destination component of the attribute class as values. In this fashion, the built-in attribute class

\[
\text{AttributeClass} := <\text{Class}, \text{attribute}, \text{Class}, \text{Alltime}>
\]

introduced above enables us to give the following TELL commands:

---

2\(^\text{Examples of this situation are given in the sequel after the attribute mechanism has been explained.}\)
Figure 2.2: Instantiation and specialization relationships between the \( \omega \)-classes
TELL CLASS conf_entity_class
   IN M1_Class
   WITH
      attribute
         conf: conference
END

TELL CLASS paper_class
   IN M1_Class
   ISA conf_entity_class
   WITH
      attribute
         card: 0..100;
         paper_descr: Class
END

The network configuration defined by these metaclass descriptions is shown in figure 2.3. It includes the following propositions along with the instantiation and specialization relations between them:

conf_entity_class = <conf_entity_class, ..., conf_entity_class, ...>
paper_class = <paper_class, ..., paper_class, ...>

conf_entity_class in M1_Class, IndividualClass
paper_class in M1_Class, IndividualClass
paper_class isa conf_entity_class, S_Class

P1: = <paper_class, card, 0..100, ...>
P1 in AttributeClass, S_Class, P1 isa Token

P2: = <paper_class, paper_descr, Class, ...>
P2 in AttributeClass, M1_Class, P2 isa S_Class

P3: = <conf_entity_class, conf, conference, ...>
P3 in AttributeClass, M1_Class, P3 isa S_Class

P1, P2, and P3 are internal names of propositions not available to the user of the system.

In the above definitions, the label attribute of AttributeClass is an attribute category for the attributes:

<conf_entity_class, conf, conference, ...>
<paper_class, card, 0..100, ...>
<paper_class, paper_descr, Class, ...>

conf, card, and paper_descr are the attribute labels for these attributes while conference, 0..100, and Class are the attribute values respectively.
Figure 2.3: Propositions conf.entity_class and paper_class

Now that brand new attribute classes have been defined, new attributes can be created for instances of the above individual classes. For example:

**TELL CLASS** paper  
IN S.Class, paper_class  
WITH  
  paper_dscr  
    author: Author;  
    title: String;  
    subject: subject_area;  
    date: TimeInterval

**END**

In a similar vein, we can give the following **TELL** command:

**TELL TOKEN** krypton  
IN paper  
WITH  
  author  
    first_author: Brachman;  
    : Fikes;  
    : Levesque  
  title  
    : 'A hybrid representation system'  
  date  
    : 1986/12  
  subject  
    : kr

**END**
Figure 2.4: Propositions conference, entity, class, paper, class, paper, and krypton
The absence of an attribute label from some of krypton's attributes means that the label will be internally generated by the Telos system.

The network configuration corresponding to the above definitions is shown in figure 2.4. By now, the details of the attribute category mechanism should have become clear: an attribute label of any class can become an attribute category of any instance of this class therefore inducing zero or more attributes for this instance.

2.1.5 Multiple Inheritance

Telos allows generalization hierarchies to be acyclic graphs rather than trees therefore introducing multiple inheritance of attributes. The following definitions illustrate such a situation introducing the generalization hierarchy shown in figure 2.5.

```
TELL CLASS potential_participant
  IN potential_participant_class
  ISA Person
  WITH
    necessary
      interest: subject_area
END

TELL CLASS potential_referee
  IN person_class
  ISA potential_participant
  WITH
    necessary
      task: Task;
      subject: subject_area
END

TELL CLASS pc_member
  IN person_class
  ISA potential_participant
  WITH
    necessary
      task: Task
END

TELL CLASS referee_manager
  IN person_class
  ISA potential_referee, pc_member
END
```

The class referee_manager is a subclass of both pc_member and potential_referee. Thus instances of referee_manager can inherit attributes from both of these classes. However, things can become complicated.
Figure 2.5: An example of multiple inheritance
TELL TOKEN Winston
IN referee_manager
WITH
  interest
  : vision;
  : robotics;
  : kr
subject
  : vision
task
  : task1
END

The above definition is ambiguous since we cannot determine whether the attribute category task comes from the class pc_member or from potential_referee. To resolve the ambiguity, we allow attribute categories to be qualified with a FROM clause naming the ancestor class at which they have been introduced. Using this construct, the command

TELL TOKEN Winston
IN referee_manager
WITH
  task FROM pc_member
  : task1
END

eliminates the above ambiguity.

2.1.6 Instantiation and Specialization Relationships as Propositions

Instantiation and specialization relationships between propositions have been treated so far as primitive. In keeping with a design decision to make the Telos representational framework as uniform as possible, we now extend this framework by treating instantiation and specialization relationships as propositions in their own right.

Consider the instantiation relationship between conf.committee and committee. It can be represented as the proposition:

<conf.committee, instanceOf, committee, t>

where t indicates how long conf.committee is a committee.

Similarly, the proposition

<paper, instanceOf, paper_class, ...>

represents the fact that paper is an instance of the class paper_class.

If instantiation relationships are to be treated as propositions, they must be instances of some appropriately structured class. This class is the s.classInstanceOf with components:

InstanceOf = <Proposition, instanceOf, Class, Alltime>.

The reader might have already noticed that by treating instantiations as propositions and assuming that every proposition is an instance of at least one class, we have introduced an infinite
regress shown in figure 2.6. To make things simple and realistic, only the first two levels of the regress are used in the framework.

Specialization relationships can also be represented by propositions. We only need to assume the existence of the class:

\[ \text{IsA} := \langle \text{Class, IsA, Class, Alltime} \rangle \]

Now the propositions representing the specialization relationship between conf_paper and paper can be represented by the following propositions:

\[ P_1 := \langle \text{conf_paper, IsA, paper, t0} \rangle \]
\[ P_2 := \langle P_1, \text{instanceOf}, \text{IsA, t0} \rangle \]
\[ P_3 := \langle P_2, \text{instanceOf}, \text{InstanceOf, t1} \rangle \]

2.2 Temporal Knowledge in Telos

The formal framework adopted for time is based on Allen's framework as presented in a series of papers (see [All81], [All83],[All84], [AH85] for details). The primitive notion in this framework is that of an interval that is a time corresponding to an event with duration. There is one primitive binary relation \text{meets} allowing for the possibility that there is no time between two intervals and no time shared by them. Every other possible relation between two time intervals can be defined in terms of \text{meets}. Figure 2.7 shows the time relations offered in Telos.

Most of these relations are self-explanatory but a few comments are in order. The during and over relations also hold for any intervals that coextend, or are equal. The rightbefore relation is a synonym for \text{meets}. Finally, \text{t1 before t2} asserts that \text{t1} ends before \text{t2} begins.

Telos also provides a number of time interval constants. \text{Alltime} is an infinite time interval including all other time intervals. There are time constants referring to a one year interval (e.g., 1975, 1976, and so on), a month interval (e.g., 1986/12, 1985/3, and so on), a day interval (e.g., 1985/12/28, 1985/3/3, and so on) as well as hour, minute and second intervals. There are also conventional time intervals of varying duration, finite or infinite (e.g., 1986/5...1986/8, 1986...). Infinite time intervals are further divided into \text{left-infinite} (e.g., *, 1986) and \text{right-infinite} ones (e.g., 1986...*). Finally, there exists a special constant named \text{Now} that represents the current system time.

The above conventional intervals can in fact be defined in Telos [Sta86] but were included in the syntax of the language for convenience. [Sta86] also describes a rudimentary algebra intended
<table>
<thead>
<tr>
<th>Temporal Relations</th>
<th>Inverses</th>
<th>Graphical Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1 during t2</td>
<td>t2 over t1</td>
<td><img src="image1" alt="Diagram" /></td>
</tr>
<tr>
<td>t1 startsbefore t2</td>
<td>t2 startsafter t1</td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td>t1 endbefore t2</td>
<td>t2 endafter t1</td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
<tr>
<td>t1 rightbefore t2</td>
<td>t2 rightafter t1</td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td>t1 before t2</td>
<td>t2 after t1</td>
<td><img src="image5" alt="Diagram" /></td>
</tr>
<tr>
<td>t1 starts t2</td>
<td>t2 starts t1</td>
<td><img src="image6" alt="Diagram" /></td>
</tr>
<tr>
<td>t1 covers t2</td>
<td>t2 covers t1</td>
<td><img src="image7" alt="Diagram" /></td>
</tr>
<tr>
<td>t1 overlaps t2</td>
<td>t2 overlapped-by t1</td>
<td><img src="image8" alt="Diagram" /></td>
</tr>
<tr>
<td>t1 equals t2</td>
<td>t2 equals t1</td>
<td><img src="image9" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Figure 2.7: Time relations and their inverses
to simplify temporal expressions. We do not deal with this issue in this report.

As argued in [All81], the model of time used here has several important advantages. It allows
the representation of relative temporal relations such as

John married after graduating from school

where it is not known when John married or when he graduated from school. The model also
allows variable “grain size” for temporal relations e.g.,

John married in 1985

There are two other desirable properties of any proposed treatment of time suggested in [All81]:

The representation should have a strong sense of now; this should be done in a manner
which allows the present moment (now) to change without requiring major changes in
the knowledge base.

The representation should allow a sense of persistence by facilitating default inferences
of the form “If P is true now, it will remain true until I hear otherwise”.

The first of the above properties is demonstrated in the formalization of the operations on the
knowledge base presented in the next chapter while the second is facilitated by the introduction of
persistent time intervals. Persistent time intervals are assumed to be right-infinite time intervals
if this is consistent with the rest of the constraints on them. Their use is limited in Telos and will
be outlined in section 2.2.2.

2.2.1 Representing Temporal Relations as Propositions

Every temporal relation between two intervals can be represented as a proposition. For every possible
relationship between two time intervals an appropriately structured simple class is introduced.
For example:

During := <TimeInterval, during, TimeInterval, Alltime>

TimeInterval is a built-in simple class, the class of all time intervals. We also assume that there
exists a simple class TemporalRelation and

During isa TemporalRelation.

Now, the temporal relationship

\( t_1 \) occurs during \( t_2 \)

can be represented by the proposition

\(<t_1, \ldots, t_2, Alltime>\)

which is an instance of the class During.
2.2.2 History of the Domain vs. Knowledge of This History

A very useful feature of a knowledge representation system supporting time is the ability to represent the progression of states of the domain over a period of time in a way that enables it to answer any historical query with respect to its current state but also with respect to a previous state. In a nutshell, it should be able to represent not only the history of the domain but also the system's knowledge of this history.

The above feature has also been considered in the design of temporal relational databases [SA85], [Snc87]. An intuitively clear way to achieve the above by extending a relational system is to augment every tuple in a relation with two temporal attributes representing the time during which this tuple is valid as well as the time during which the transaction entering this tuple took place. These two times are called valid and transaction time respectively [SA85].

The same issue has been recently studied in the context of temporal deductive databases in [Sri88] using Kowalski's event calculus framework.

The following example illustrates the way the above feature is incorporated in Telos:

\[
\text{TELL CLASS conf_paper} \\
\quad \text{IN paper_class} \\
\quad \text{ISA paper} \\
\quad \text{WITH} \\
\quad \text{paper_dscr} \\
\quad \text{referee: Referee} \\
\text{END}
\]

The propositions corresponding to the above command are:

\[
\text{conf_paper} := <\text{conf_paper}, \ldots, \text{conf_paper}, T_0>
\]

\[
P_1 := <\text{conf_paper}, \text{instanceOf}, \text{paper_class}, T_1>
\]

\[
P_2 := <P_1, \text{instanceOf}, \text{InstanceOf}, T_2>
\]

\[
P_3 := <\text{conf_paper}, \text{isA}, \text{paper}, T_3>
\]

\[
P_4 := <P_3, \text{instanceOf}, \text{isA}, T_3>
\]

\[
P_5 := <P_4, \text{instanceOf}, \text{InstanceOf}, T_4>
\]

\[
P_6 := <\text{conf_paper}, \text{referee}, \text{Referee}, T_5>
\]

\[
P_7 := <P_6, \text{instanceOf}, \text{Q}, T_6>
\]

\[
P_8 := <P_7, \text{instanceOf}, \text{InstanceOf}, T_7>
\]

where Q is the following proposition:

\[
Q := <\text{paper_class}, \text{paper_dscr}, \text{Class}, \ldots>
\]

By convention, the time component of the proposition denoting a specialization relationship is the same with the time component of the proposition denoting its instantiation to the built-in class \text{isA}.

The propositions \text{conf_paper}, P_1, P_3-P_4, P_6 and P_7 provide us with the following temporal knowledge:

- The length of \text{conf_paper}'s lifetime is T_0.
- The class \text{conf_paper} is an instance of \text{paper_class} throughout T_1.
• The class `conf.paper` is a specialization of `paper` throughout `T3`.
• The length of attribute `P6`'s lifetime is `T5`.
• The class `conf.paper` has the attribute `P6` for the time period `T6`.

These propositions provide us with information about the history of our domain. We use the remaining propositions (i.e., the links to the class `InstanceOf` to represent the system's beliefs with respect to this history. In this way, the propositions `P2`, `P5` and `P8` represent the following information:

• The system believes `P1` throughout `T2`.
• The system believes `P3-P4` throughout `T4`.
• The system believes `P7` throughout `T6`.

In the sequel, time intervals like `T1`, `T3` and `T5` are called `history time intervals` or `history times` while time intervals like `T2`, `T4` and `T6` are called `belief time intervals` or `belief times`.

History time intervals must be provided by the user while belief time intervals must be under the control of the system. Consequently, the syntax of `TELL` is augmented so that the user is able to specify temporal information concerning the time components of the propositions entered in the knowledge base. In fact, for every proposition the user is enabled to specify at most one relationship of its temporal component with a conventional time interval. The relationships allowed are the ones presented above. Thus, the syntax for the `TELL` command becomes as follows:

```
TELL CLASS <identifier> ( <temporal relation> <time interval> )
    IN <identifier> ( <temporal relation> <time interval> )
    ISA <identifier> ( <temporal relation> <time interval> )
    WITH
    <category>
    <label> : <value> ( <temporal relation> <time interval> )
END
```

The above temporal components are optional. If anyone of them is missing then (at `systime..*`) is assumed to be in its place where `systime` is the time the system executes the above command.

Let us now clarify the extended syntax of `TELL` with the following example:

```
TELL TOKEN krypton ( at 1986/10..* )
    IN paper ( at 1986/10..* )
    WITH
    author
        first_author: Brachman ( at 1988/10..* );
    : Fikes ( at 1987/1..* );
    : Levesque ( during 1986/10 )
END
```

The purpose of this command is to inform the system that krypton came into being at the beginning of October 1986 and it is a paper since then. Brachman (the first author) started

---

We assume that a class is being defined and that the instantiation, specialization and attribute classes contain only one element. The complete syntax for the command `TELL` can be found in Appendix A.
writing this paper at the beginning of that month and Fikes joined him on the first day of January 1987. Levesque was also one of the authors during October 1986.

This command will result in the following collection of propositions being entered in the knowledge base.

\[ krypton := \langle krypton, \ldots, krypton, t0 \rangle \]

\[ p1 := \langle krypton, instanceOf, paper, t1 \rangle \]
\[ p2 := \langle p1, instanceOf, InstanceOf, t2 \rangle \]

\[ p3 := \langle krypton, first_author, Brachman, t3 \rangle \]
\[ p4 := \langle p3, instanceOf, q, t3 \rangle \]
\[ p5 := \langle p4, instanceOf, InstanceOf, t4 \rangle \]

\[ p6 := \langle krypton, \ldots, Fikes, t5 \rangle \]
\[ p7 := \langle p6, instanceOf, q, t5 \rangle \]
\[ p8 := \langle p7, instanceOf, InstanceOf, t6 \rangle \]

\[ p9 := \langle krypton, \ldots, Levesque, t7 \rangle \]
\[ p10 := \langle p9, instanceOf, q, t7 \rangle \]
\[ p11 := \langle p10, instanceOf, InstanceOf, t8 \rangle \]

Note that \( q \) is the following attribute class:

\[ q := \langle paper, author, Author, \ldots \rangle \]

The temporal relationships in this command also introduce the following constraints on the above history time intervals:

\[ t0 \text{ at } 1986/10..* \]
\[ t1 \text{ at } 1986/10..* \]
\[ t3 \text{ at } 1986/10..* \]
\[ t5 \text{ at } 1987/1..* \]
\[ t7 \text{ during } 1988/10 \]

In addition to this, the system imposes the following constraints on the above belief time intervals:

\[ t2 \text{ costarts systime} \]
\[ t4 \text{ costarts systime} \]
\[ t6 \text{ costarts systime} \]
\[ \text{persistent}(t2) \]
\[ \text{persistent}(t4) \]
\[ \text{persistent}(t3) \]

It should be clear from the above how our persistence mechanism works. The system starts believing the new knowledge from the time it is entered in the knowledge base and on. In other words, it assumes that every belief time interval is persistent and starts at the time the corresponding TELL command is executed (systime in the above example). Later on, the commands UNTELL and RETELL will be introduced as the means of overriding this default mechanism i.e., telling the system that it should stop believing a certain statement since it no longer holds. It is intuitively clear that relationships built-in in the system must be believed for ever i.e., they must have \text{alltime} as their belief time interval.

The previous discussion allows us to introduce the following persistence rule:

\[ 24 \]
Persistence Rule: A sentence in the knowledge base holds for ever if and only if its belief time is alltime. If the belief time of a sentence is a persistent time interval then this sentence holds until the system is told otherwise.

As we have already said, the commands UNTELL and RETELL presented later in this chapter are the only ways to inform the system otherwise.

This notion of persistence is in fact the same as the one considered by Allen [All81]. Allen defines a persistent interval as one that “should be assumed to extend as far as possible given the constraints”. In our framework, the only constraints that can be attached to a persistent interval is that it costarts (and later on possibly ends) with a conventional time interval. This restriction makes our mechanism simpler and more efficient.

Kowalski has also used a persistence axiom similar to ours for his event calculus [KS85], [Kow86]. However, his approach is different since the notion of an event is considered to be primitive in his framework.

The above discussion did not mention any of the propositions describing the default instantiation of krypton to the appropriate built-in classes. These propositions are the following:

\[
\begin{align*}
q_1 &= <\text{krypton}, \text{instanceOf}, \text{Proposition}, \text{alltime}> \\
q_2 &= <q_1, \text{instanceOf}, \text{InstanceOf}, \text{alltime}> \\
q_3 &= <\text{krypton}, \text{instanceOf}, \text{Individual}, t_0> \\
q_4 &= <q_2, \text{instanceOf}, \text{InstanceOf}, t> \\
q_5 &= <\text{krypton}, \text{instanceOf}, \text{Token}, t_0> \\
q_6 &= <q_2, \text{instanceOf}, \text{InstanceOf}, t>
\end{align*}
\]

persistent(t) \\
t costarts systime

This example illustrates the rules used to determine the time components of the instantiation relationships of any object to its corresponding built-in classes. These rules are the following:

1. Every proposition defined is made an instance of the \(\omega\)-class Proposition. Both history and belief time of this instantiation relationship are semi-infinite time intervals starting at the time the corresponding command was executed.

2. Every individual token (resp. individual class) defined is made an instance of the \(\omega\)-class Individual (resp. IndividualClass). The history time for this instantiation is the lifetime of the token (resp. class). By default, this is also the history time interval for the instantiation to the built-in class Token (resp. S.Class or M1.Class etc.). The belief time of these relationships are always set by the system to be persistent intervals starting at the time the corresponding command was executed.

3. Every attribute token (resp. attribute class) defined is made an instance of the \(\omega\)-class Attribute (resp. AttributeClass). The history time for this instantiation is the lifetime of the attribute token (resp. attribute class). This is also the history time interval for the instantiation to the built-in class Token (resp. S.Class or M1.Class etc.). The belief time of these relationships are always set by the system to be persistent intervals starting at the time the corresponding command was executed.

25
2.3 The Assertional Component of Telos

So far, we have introduced the means for specifying a knowledge base structure as well as rudimentary constraints on this structure. Now we augment Telos with an assertion language that can be used to state domain-specific constraints on the knowledge base, augment the knowledge in it, and also query the current or any past state of it.

The Telos assertion language is a typed first order language with equality. The set of constants in this language contains every proposition introduced in the knowledge base (e.g., John, paper etc.) while its set of types contains every class. The set of terms in the language is defined as follows:

- Variables and constants are terms
- If t is a term then from(t), label(t), to(t) and when(t) is a term.

The predicate symbols applied to terms yield the atomic formulas of the language:

- prop(p,x,y,z,t): p is a proposition with components x, y, z, and t.
- instanceof(x,y,t1,t2): x is an instance of y for the time period t1 and this is believed by the system for the time period t2.
- isa(x,y,t1,t2): x is a specialization of y for the time period t1 and this is believed by the system for the time period t2.
- att(x,y,t1,t2): y is a value of an attribute of x with attribute category att for the time period t1 and this is believed by the system for the time period t2.
- For every terms x and y in the language and every temporal predicate \( \theta \), \( x \theta y \) is an atomic formula with the obvious meaning.
- instanceof(x,y,t): x is an instance of y for the time period t.
- isa(x,y,t): x is a specialization of y for the time period t.
- att(x,y,t): y is a value of an attribute of x with attribute category att for the time period t.

The well-formed formulas (wffs) in the language are defined as follows:

- Every atomic formula is a wff.
- If a and b are wffs then a or b, a and b, a \( \rightarrow \) b and a \( \leftrightarrow \) b are also wffs.
- If a is a wff then not a is a wff.
- If a is a wff, C is a type and x is a variable then (Forall x/C) a and (Exists x/C) a are wffs.

\( ^1 \)These functions return the source, label, destination and duration of their arguments respectively.
In addition to the above, we also introduce the standard function and predicate symbols of integer arithmetic in the language.

The above language is similar to data definition and query languages used by deductive databases [GMN84]. The presence of multi-valued attributes in Teles motivates us to extend this language to include set-valued terms. The introduction of sets has also been motivated by their use in complex object databases [BS86] and it is in agreement with recent proposals for deductive database languages [Zan88].

In the following we expand the above language to include set terms. These can be introduced in any of the following ways:

- Sets can be explicitly enumerated. For example, the set of the propositions John and Mary is denoted by the term \{John, Mary\}.

- The following set valued functions are introduced:

  1. \texttt{x.n [\theta t1 - believed \xi t2]}: It is defined only for the case where \(x\) is an instance of a class for which an attribute with label \(n\) has been defined. It returns the attribute values of all attributes of \(x\), induced by the attribute category \(n\), during a time interval \(t2\) such that \(t2 \theta t1\), and this is believed during a time interval \(t3\) such that \(t3 \xi t1\). Of course, \(\theta\) and \(\xi\) are time relations.

     \[
     \texttt{x.n [\theta t1 - believed \xi t2]} := \\
     \{ v \mid \text{(Exists p,P,C,L,V/Proposition)} \\
     \text{(Exists t3,t4/TimeInterval)} \\
     \text{(prop(p, x, l, v, t3) and prop(P, C, n, v, t4) and p in P [\theta t1 - believed \xi t2])}\}
     \]

  2. \texttt{x.n [\theta t1 - believed \xi t2]}: It is identical to the previous one, but returns the propositions themselves rather than the destination components.

     \[
     \texttt{x.n [\theta t1 - believed \xi t2]} := \\
     \{ p \mid \text{(Exists v,V,P,C,L/Proposition)} \\
     \text{(Exists t3,t4/TimeInterval)} \\
     \text{(prop(p, x, l, v, t3) and prop(P, C, n, v, t4) and p in P [\theta t1 - believed \xi t2])}\}
     \]

  3. \texttt{x.n [\theta t1 - believed \xi t2]}: It returns the set of destination components of the attribute propositions having source \(x\), label \(n\), and time component \(t3\) such that \(t3 \theta t1\) and they are believed by the system during a time \(t4\) such that \(t4 \xi t2\).

     \[
     \texttt{x.n [\theta t1 - believed \xi t2]} := \\
     \{ z \mid \text{(Exists q/Proposition)(Exists t3/TimeInterval)} \\
     \text{(prop(q, x, n, z, t3) and q in Attribute [\theta t1 - believed \xi t2])}\}
     \]

  4. \texttt{x.n [\theta t1 - believed \xi t2]}: It is identical to the previous one, but returns the propositions instead of the destination components.

     \[
     \texttt{x.n [\theta t1 - believed \xi t2]} := \\
     \{ q \mid \text{(Exists l/Proposition)(Exists t3/TimeInterval)} \\
     \text{(prop(q, x, n, l, t3) and q in Attribute [\theta t1 - believed \xi t2])}\}
     \]

Figure 2.8 shows some examples illustrating the above definitions. For all the above functions the part enclosed in square brackets can be absent (e.g., \texttt{krrypton.author}). In this case no constraint is imposed on the corresponding time intervals.
Subsets of a known set are terms which can be constructed using the following syntax:

\{ Each \textit{<variable>} / \textit{<set-expression>} : \textit{<formula>} \}

For example, the set of John's elder brothers is denoted by the term:

\{ Each x/John.brother : John.brother.age > John.age \}

The extension of a class during a certain time period according to the beliefs of the system during another time period is denoted by the terms:

\{ Each \textit{<class>} [ \textit{<time relation>} \textit{<time interval>} - believed \textit{<time relation>} \textit{<time interval>} ] \}

For example, \{ Each Person [during 1986 - believed during 1987] \} denotes the set of all instances of Person during the year 1986 according to the beliefs of the system during the year 1987.

The standard set-theoretic functions \textit{Union}, \textit{Intersection} and \textit{Difference} can be applied to sets to produce a new set.

The standard set-theoretic predicate \textit{memberOf} is introduced.

The general syntax of a quantified expression becomes:

\((\textit{<quantifier>} \textit{<variable>} / \textit{<set-expression>} ) \textit{<formula>}\)

where \textit{<quantifier>} can be \textit{Forall} or \textit{Exists}.

When the set expression used as the domain of a quantifier is the extension of a class, things can be simplified by omitting the curly brackets and the keyword Each. For example:

(\textit{Forall} x/Person [during 1986 - believed during 1988])

is a legal form of universal quantification.

The temporal components in all the above set-expressions are optional and are used to constrain the times for which these expressions must be evaluated. The following sections present a number of defaults used to fill in these components depending on the use of the assertion language formula.

We now introduce an infix version of the four-argument \textit{instanceOf} predicate:

\[ x \text{ in } y [\theta \text{ at } t_1 - \text{believed} \xi t_2 ] \iff (\text{Exists } t_3,t_4 / \text{TimeInterval}) (\text{instanceOf}(x,y,t_3,t_4) \text{ and } (t_3 \theta \xi t_1) \text{ and } (t_4 \xi \xi t_2)) \]

Note that

\[ x \text{ in } y [\text{at } t_1 - \text{believed at } t_2 ] \iff \text{instanceOf}(x,y,t_1,t_2) \]

We also assume that any temporal predicate \(\theta\) can be applied to arbitrary pairs of propositions. The following equivalence gives the meaning of such an expression:

\[ P_1 \theta P_2 \iff \text{when}(P_1) \theta \text{ when}(P_2) \]
In the above network:

- krypton\_author[overlaps t0] = \{Brachman, Fikes\}
- krypton\_author[over t0] = \{Brachman\}
- krypton\_author[over t0] = \{<krypton, princ\_author, Brachman, t1>\}
- krypton\_author[during t0] = \{
- krypton\_princ\_author[over t0] = \{Brachman\}

Figure 2.8: Examples of the primitive Tekos functions
2.3.1 Integrating Assertion Language Formulas into the Framework

We integrate closed formulas of the assertion language (hereafter simply assertions) into the Telos framework by representing them by assertion propositions. These are individual propositions that are instances of the simple built-in class Assertion. Any assertion can be attached to a proposition, called its subject, via the attribute mechanism.

Drawing from work in deductive databases [GMN84], we distinguish two special classes of assertions: integrity constraints and deductive rules. The former constrain the knowledge in the knowledge base while the latter are used to derive implicit knowledge.

2.3.2 Integrity Constraints

We introduce a built-in attribute class IntegrityConstraint with components:

\[ \text{<Proposition, integrityConstraint, Assertion, Alltime>} \]

This attribute class can be used to attach integrity constraints to propositions in the knowledge base. In effect, it enables TELL statements of the following form:

TELL CLASS conf_paper
  IN S_Class
  ISA paper
  WITH
    integrityConstraint:
      \$ (\forall p/\text{conf_paper}) (\forall x/\text{Author}) (\forall t_1,t_2/\text{TimeInterval})
        (\text{author}(p,x,t_1,t_2) \implies \text{not}(x \text{ MemberOf } p.\text{referee})) \$ (at 1988..*)
  END

By convention, assertions appearing as attribute values in TELL statements must be enclosed in dollar signs. This constraint asserts that from 1988 and on, authors of a conference paper must not be referees for it.

Let us assume that the above command was processed on January 2nd, 1988. The set of propositions and temporal relations corresponding to its WITH is the following:

\[ p_1 := \text{<conf_paper, 13450, \{(\forall p/\text{conf_paper}) \ldots \}, } t_1 > \]
\[ p_2 := \text{<p_1, instanceOf, IntegrityConstraint, } t_2 > \]
\[ p_3 := \text{<p_2, instanceOf, InstanceOf, } t_3 > \]

\[ t_1 \text{ at 1988..*} \]
\[ t_2 \text{ at 1988..*} \]
\[ t_3 \text{ starts 1988/01/02} \]
\[ \text{persistent}(t_2) \]

Note that 13450 is an internal label generated by the system and the destination component of \( p_1 \) is the text of the assertion (not given completely for brevity).

Integrity constraints refer to history and belief times using the facilities offered by the assertion language. However, since integrity constraints are special attribute values they also have a history time and a belief time associated with them. For simplicity, we constrain the temporal components given for an integrity constraint to be of the form:

\[ (\text{at } <\text{time interval}> ) \]
If no time component is given the default (i.e., \(\text{at systime..*}\)) is also used in this case.

In this fashion, integrity constraints are interpreted as formulas that must be entailed by the knowledge base during their history time according to the beliefs of the system during their belief time. Therefore, a new formula in the assertion language can be deduced which captures this interpretation of an integrity constraint. This can be achieved by making the temporal constraints posed by the history and belief time explicit in the integrity constraint. In particular, these constraints are made explicit in every atomic subformula whose history or belief time component is unconstrained. The formula deduced this way is called the full form of the integrity constraint with respect to these history and belief times. This formula must always be entailed by the knowledge base.

Let us now illustrate the construction of the full form of an integrity constraint with an example. The constraint in the last TELL command asserts that for every conference paper none of its authors can be a referee for it as well. Taking into account the above history and belief times, the full form of this constraint is the following formula:

\[
(\forall t_1, t_2 / \text{TimeInterval})(t_1 \text{ during } 1988..* \text{ and } t_2 \text{ during } 1989/1/12..* \implies \\
(\forall p / \text{Proposition})(\text{instanceOf}(p, \text{conf_paper}, t_1, t_2) \implies \\
(\forall x / \text{Proposition})(\text{instanceCf}(x, \text{Author}, t_1, t_2) \implies \\
(\forall t_3, t_4 / \text{TimeInterval})(\text{author}(p, x, t_3, t_4) \implies \\
\text{not (x MemberOf p.referee [at t_1 - believed at t_2] )})))
\]

Or equivalently:

\[
(\forall t_1, t_2, t_3, t_4 / \text{TimeInterval})(\forall p, x / \text{Proposition}) \\
(t_1 \text{ during } 1988..* \text{ and } t_2 \text{ during } 1989/1/12..* \text{ and } \\
\text{instanceCf}(p, \text{conf_paper}, t_1, t_2) \text{ and } \text{instanceOf}(x, \text{Author}, t_1, t_2) \text{ and } \\
\text{author}(p, x, t_3, t_4) \implies \\
\text{not (x MemberOf p.referee [at t_1 - believed at t_2] )})
\]

Appendix B describes in detail how to compute the full form of an integrity constraint with respect to some history and belief time.

2.3.3 Deductive Rules

We now introduce the attribute class DeductiveRule which can be used to attach a deductive rule to any proposition. It is defined as follows:

\[
\text{DeductiveRule} := \langle \text{Proposition, deductiveRule, Assertion, Alltime} \rangle
\]

Let us now give an example of a deductive rule in Telos. Assume that the following TELL commands have been executed.

```
TELL CLASS person
IN S_Class
WITH
  necessary
    name: String;
    address: Address;
  attribute
    phone_num: String
END
```
TELL CLASS Employer
   IN S_Class
   WITH
      necessary
      name : String;
      emp_address : Address
END

If we want to assert that an employer's address is the work address of every author working for this employer, we can define the class Author as follows:

TELL CLASS Author
   IN S_Class
   ISA person
   WITH
      attribute
      affiliation : Employer
      deductiveRule
      :$ (Forall x/Author)(Forall y/Address)(Forall t1/TimeInterval)
         (y MemberOf x.affiliation.emp_address [at t1]
          ==> address(x,y,t1))$ 
END

This deductive rule asserts that every address of the employer of an author is effectively an address for the author himself.

If the above command was executed on January 2nd, 1988 then the set of propositions and temporal relations entered in the knowledge base because of the deductiveRule attribute is the following:

\[ p1 := \langle Author, 1789, (Forall x/Author) \ldots', t1 \rangle \]
\[ p2 := \langle p1, instanceOf, DeductiveRule, t2 \rangle \]
\[ p3 := \langle p2, instanceOf, InstanceOf, t3 \rangle \]
\[ t1 at 1988\slash 01\slash 02\ldots* \]
\[ t2 at 1988\slash 01\slash 02\ldots* \]
\[ t3 costarts 1988\slash 01\slash 02 \]
\[ persistent(t3) \]

Note that 1789 is an internal label generated by the system and the destination component of \( p1 \) is the text of the assertion (not given completely for brevity). Since no time component for the rule as a proposition was present in the above command the default (at 1988/01/02..*) was used. However, the above deductive rule is also entered in the knowledge base and can later be used by the system to derive addresses of any author. This fact forces us to impose certain restrictions on the expressive power of deductive rules. The first restriction is that they must be of the form

\[ (Forall x1/S1) \ldots (Forall xn/Sn) \ W ==> A \]

where \( W \) is an arbitrary formula in the assertion language, \( A \) is an atomic formula with predicate one of the defined 3-place predicates and the only free variables of \( W \) and \( A \) are \( x_1, x_2, \ldots, xn \) whose ranges are the sets \( S_1, \ldots, Sn \). The intuitive meaning of such a deductive rule is that
if \( W \) is true then we are able to conclude \( A \). We constrain the deductive rules to have the above form since reasoning with them is a lot easier than with arbitrary first order logic.\(^5\)

In addition to the above, the predicate of the atomic formula \( A \) must not reference belief times since these are set by the system itself. In fact, before any deductive rule is put in the knowledge base, it is translated into its full form exactly as an integrity constraint. In the full form, the \( A \) part is substituted by an atomic formula with a belief time guaranteed not to create future inconsistencies. Considering the form of the deductive rules it is not very difficult to see that there should be no inconsistencies if the belief time of any derived fact starts at the same time or after the latest of the belief times of any atomic formula in the \( W \) part of the rule and the belief time of the rule itself.

In this fashion, the full form of the above deductive rule is the following:

\[
(\text{Forall } t1,t2/\text{TimeInterval})(t1 \text{ during } 1988/01/02..* \text{ and } t2 \text{ during } 1988/01/02..* \implies (\text{Forall } x,y/\text{Proposition})(\text{Forall } t3/\text{TimeInterval})
\]
\[
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{instanceOf}(x,\text{Author},t1,t2) \text{ and } \text{instanceOf}(y,\text{Address},t1,t2) \text{ and }
\]
\[
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{y MemberOf } x.\text{affiliation}.\text{omp}.\text{address} \text{ at } t3 \implies \text{believed at } t2) \implies
\]
\[
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{address}(x,y,t3,\text{latest}([t2,1988/01/02])))
\]

This formula can be written equivalently as:

\[
(\text{Forall } t1,t2/\text{TimeInterval})(\text{Forall } x,y/\text{Proposition})(\text{Forall } t3/\text{TimeInterval})
\]
\[
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad (t1 \text{ during } 1988/01/02..* \text{ and } t2 \text{ during } 1988/01/02..* \implies
\]
\[
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{instanceOf}(x,\text{Author},t1,t2) \text{ and } \text{instanceOf}(y,\text{Address},t1,t2) \text{ and }
\]
\[
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{y MemberOf } x.\text{affiliation}.\text{omp}.\text{address} \text{ at } t3 \implies \text{believed at } t2) \implies
\]
\[
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{address}(x,y,t3,\text{latest}([t2,1988/01/02])))
\]

2.3.4 Extensibility Through User Defined Attribute Classes

So far, we have only demonstrated that the language allows the user to define simple attribute classes as attributes of individual classes. Using the full machinery of propositions, time, and assertions for the definition of attribute classes, makes the language fully extensible, without resorting to external program calls as in systems such as KEE [FK85].

For example, we can introduce author attributes for instances of the class paper in the following way:

\[
\text{TELL CLASS AuthorAttribute}
\]
\[
\text{COMPONENTS <paper, author, Author, Alltime>}
\]
\[
\text{IN AttributeClass, S_Class}
\]
\[
\text{WITH}
\]
\[
\text{integrityConstraint}
\]
\[
\text{hasAuthor}:$(\text{Forall } x/\text{Proposition})(\text{Forall } t/\text{TimeInterval})
\]
\[
\text{x in paper \text{ at } t) \implies (\text{Exists } p/\text{Proposition})
\]
\[
\text{(p in AuthorAttribute \text{ during } t) and from(p)=x)}
\]
\[
)$
\]

The clause COMPONENTS can only be used for attribute class definitions. It simply provides the components of this attribute class.

The above constraint requires that every instance of paper must have an author attribute during its "paperhood". More generic constraints can be expressed in Telos through attribute metaclasses. For instance, the built-in attribute class Necessary can be defined as:

\[^5\text{A formula of this form is equivalent to a deductive database clause as considered by Lloyd and Topor in [LT85].}\]
Figure 2.9: The direct constraint hasValue.

TELL CLASS Necessary
  COMPONENTS <Class, necessary, Class, Alltime>
  IN AttributeClass, M1_Class
  WITH
    integrityConstraint
      hasValue:$(Forall p1/AttributeClass)(Forall ti/TimeInterval)
        (Forall C/Class)(Forall x/Proposition)
        (p1 in Necessary [at ti] and from(p1)=x and
         x in C [during ti] => (Exists p2/Attribute)
         (Exists t2/TimeInterval)(p2 in p1 [over t2]
         and from(p2)=x))$
  END

The configuration asserted by the hasValue constraint is shown in figure 2.9: every instance of a class C where a necessary attribute class has been defined must have an attribute which is an instance of this attribute class.

Necessary can now be used to define necessary attributes:

TELL CLASS committee
  IN S-Class
  WITH
    attribute
      charter : task_description
      necessary
      mbrs : potential_participant_class;
      chair : potential_participant
  END

The above definition constrains the knowledge base so that for every instance of paper there must be a mbrs and a chair attribute.

34
It is possible to define other meta-attributes similar to _necessary_ which can be used to define constraints on the knowledge base. Consider for example _Single_:

```
TELL CLASS Single
    COMPONENTS <Class, single, Class, Alltime>
    IN AttributeClass, M1_Class
    WITH
        necessary
        onlyOne : $(\forall s/Sing)(\forall p/Proposition)
            $(\forall q/Proposition)(\neg (\exists t1/TimeInterval)
                (\exists t2/TimeInterval)p \in s \{at t1\}) \land
                q \in s \{at t2\} \land \text{from}(p)=\text{from}(q) \land
                (t1 \text{ overlaps } t2 \text{ or } t2 \text{ overlaps } t1)$
END
```

_Single_ requires that its attribute instances should have a single attribute value (in fact, either zero or one).

### 2.3.5 Talking about Assertions in the Assertion Language

Keeping with a design decision to make the framework as uniform as possible, we now introduce a way of talking about assertions in the assertion language. This feature is very powerful and has been exploited mainly in defining meta-attribute classes.

An easy way to achieve the above is to introduce a metalanguage which allows us to talk about assertions. However, since we also want to be able to talk about these metalanguage assertions, we will have to introduce a metametalanguage and so on. Instead of doing this, we propose to amalgamate assertion language and metalanguage following [BK32]. Formally, this can be done by introducing a set of constants in the assertion language which will serve as names for assertions, new sorts whose elements are these constants, and possibly new predicates whose arguments would range over them.

In the following, we will use double quotes to construct names for assertions. For example, to assert that the deductive rule in the previous section is an instance of the built-in class _Assertion_ we would write (in the assertion language):

```
(\exists t1/TimeInterval)(\exists t2/TimeInterval)
    instanceOf("(\forall x/Author) ...", Assertion, t1, t2)
```

The built-in class _Assertion_ serves as a sort whose elements are assertions. We also introduce a 1-place predicate _prove_ that captures the provability relation in the knowledge base. It can optionally be followed by a temporal component of the form

```
[\theta t1 - believed t2]
```

and it is true if and only if its argument is an assertion provable from the knowledge base at a time which is in relation \(\theta\) with \(t1\) according to the beliefs of the system at a time which is in relation \(\theta\) with \(t2\). In other words,

```
prove(a)[\theta t1 - believed t2]
```

is true if and only if the full form\(^6\) of the assertion \(a\) with respect to the above temporal component is provable from the knowledge base.

\(^6\)See Appendix B.
As we have already mentioned we can exploit this feature of Telos in defining meta-attributes. Consider \texttt{PreCondition} as an example:

\begin{verbatim}
TELL CLASS PreCondition  
    COMPONENTS <Class, preCondition, Assertion, Alltime>
    IN AttributeClass, Mi_Class
    WITH
        integrityConstraint
        :$(\forall p/PreCondition)(\forall x/Proposition)(\forall t/TimeInterval)
        ( x \text{ in from}(p) [\text{at } t] \implies \text{prove(to}(x)) [\text{costarts } t] )$
END
\end{verbatim}

This attribute class constrains its instances to have destination components which are assertions provable from the knowledge base at the beginning of the life-time of their source component. Consider for example the following command:

\begin{verbatim}
TELL CLASS conference  
    IN conf_entity_class
    WITH
        attribute
            budget: Money
        precondition
            :$(\forall x/conference) ( x.budget \geq 10000 )$
END
\end{verbatim}

This defines the class \texttt{conference} and asserts that every time an instance of it (an actual conference) is created, its budget must exceed $10,000.

\section{2.4 Query Answering in Telos}

We consider a Telos knowledge base (from now on simply \textit{KB}) as being partitioned into two parts: the \textit{definitional part} (denoted by \textit{KB}_D) and the \textit{assertional part} (denoted by \textit{KB}_A). \footnote{The reader should be warned that knowledge bases in Telos consider different pieces of knowledge definitional than \textsc{Krypton}; for example, a specific person's age is an assertional fact in \textsc{Krypton}, where only generic concepts are definitional. This is certainly not the case in Telos.} The assertional part of the \textit{KB} consists solely of the deductive rules given by the user. The definitional part is composed of the initial knowledge base, which represents the built-in Telos objects and their associations, as well as the propositions corresponding to any object definitions processed by the system. The reason for adopting this partition is that the \textit{KB} is simpler if we do not consider the deductive rules so queries can be answered more efficiently. The \textit{KB}_A part can be consulted when no answer was obtained from the \textit{KB}_D part, but the reasoning then will be more complicated.

\subsection{2.4.1 The RETRIEVE Command}

The \texttt{RETRIEVE} command is used when we want to query the definitional part of the \textit{KB}. The general form of \texttt{RETRIEVE} is

\begin{verbatim}
RETRIEVE t1/s1, ..., xn/sn : W
ON t1
AS OP t2
\end{verbatim}
where \( W \) is a formula of KBL, \( x_1, \ldots, x_n \) its only free variables, \( s_1, \ldots, s_n \) are set expressions denoting the ranges of these variables and \( t_1, t_2 \) are conventional dates. The clauses \( \text{ON} \) and \( \text{AS OF} \) determine the history times and belief times that are of interest to us. Therefore, \text{RETRIEVE} \) computes its answers assuming that the historical slice \( t_1 \) of the world as described by the beliefs of the system throughout \( t_2 \) is of interest to us. \text{RETRIEVE} \) returns all the possible substitutions for the variables \( x_1, \ldots, x_n \) ranging over \( s_1, \ldots, s_n \) which make \( W \) true after every missing history or belief time component in a set expression in the query has been replaced by \( \text{over} \) \( t_1 \) and \( \text{over} \) \( t_2 \) respectively. Note that the \( \text{ON} \) and \( \text{AS OF} \) clauses are optional. When they are absent \( \text{ON Now} \) and \( \text{AS OF Now} \) can be assumed by default. In other words, a query with no temporal qualifications is interpreted as questioning the current state of affairs according to the current beliefs of the system.

If we want to prove that a closed formula \( W \) is true considering the history of the world throughout \( t_1 \) according to the beliefs of the system throughout \( t_2 \), we can pose the following query:

\[
\text{RETRIEVE} : W \\
\text{ON} \ t_1 \\
\text{AS OF} \ t_2
\]

The answer in this case will be \text{yes}, \text{no}, or \text{unknown}.

Assume that the \text{TELL} statements in section 2.3.3 have been executed. Let us now execute the following statements:

\begin{verbatim}
TELL TOKEN Ron
  IN Author
  WITH
    name    : 'Ron B.'
    affiliation : 'UofS'
    address
      home_address : r_addr ( at 1988/10/01..* )
END

TELL TOKEN Hector
  IN Author
  WITH
    name    : 'Hector L.'
    affiliation : 'UCRT
    address
      home_address : h_addr ( at 1980..* )
END
\end{verbatim}
TELL TOKEN UofT
IN Employer
WITH
  emp_name
    : 'University of Toronto'
  emp_address
    : UofT_addr
END

TELL TOKEN UofS
IN Employer
WITH
  emp_name
    : 'University of Saskatchewan'
  emp_address
    : UofS_addr
END

Now, if we want to find all authors working at the University of Toronto, we should pose the query:

RETRIEVE x/Author : UofT MemberOf x.affiliation

The answer to the above query will be the set { Hector }. Note that since no GN or AS CF class is present the answer was computed with according to the current beliefs of the system about the current history.

Now assume that we want to find all current mailing addresses of Hector. Posing the query

RETRIEVE x/Address : x MemberOf Hector.address

will get us the set { h_addr }. Therefore, we cannot get a complete answer to this query since RETRIEVE does not consider the deductive rules present in the knowledge base.

The following queries are examples of asking whether something follows from the definitional part of the KB:

RETRIEVE : Ron.home_address [over 1987/11] = r_addr

RETRIEVE : Ron.home_address = r_addr
ON 1987/11

RETRIEVE : Hector.home_address = r_addr

RETRIEVE : UofT_addr MemberOf Hector.address

The first two of these are different forms of the same query and have answer yes. The second and the third have answer no.

2.4.2 The ASK Command

The syntax of ASK is similar to that of RETRIEVE:
ASK x₁/s₁, . . . , xₙ/sₙ : W
ON t₁
AS CF t₂

ASK has the same capabilities as RETRIEVE but it is less efficient since the deductive rules given by the user are also consulted to find the answer to a posed query.

Continuing the example of the previous section, let us pose two queries concerning the current mailing addresses of Hector according to the current beliefs of the system:

ASK x/Address : x MemberOf Hector.address

ASK : UofT_addr MemberOf Hector.address

As the reader might expect, the answers to these queries are { h_addr, UofT_addr } and yes respectively.

2.5 Updating the Knowledge Base

As we have already seen, the TELL statement can be used to enter a new object in the knowledge base as well as append new attributes to it. In this section, we introduce two modification statements that can be used to inform the system about any changes in its relationships to other objects. These modification statements do not delete any information from the knowledge base but rather change the beliefs of the system.

2.5.1 The UNTELL Command

The UNTELL command can be used to specify that some explicitly created\(^8\) instantiation, specialization, or attribute relationship of an object does not hold for a certain time period. As we have already noted, the propositions corresponding to these relationships are not explicitly deleted from the KB but rather the fact that the system no longer believes them is recorded.

The UNTELL command has a syntax similar to TELL:

\[
\text{UNTELL } \langle\text{class-or-token}\rangle \langle\text{identifier}\rangle \ ( \langle\text{temporal relation}\rangle \ \langle\text{time interval}\rangle \ )
\]

IN \langle\text{identifier}\rangle \ ( \langle\text{temporal relation}\rangle \ \langle\text{time interval}\rangle \ )

ISA \langle\text{identifier}\rangle \ ( \langle\text{temporal relation}\rangle \ \langle\text{time interval}\rangle \ )

WITH

\langle\text{category}\rangle

\langle\text{label}\rangle : \langle\text{value}\rangle \ ( \langle\text{temporal relation}\rangle \ \langle\text{time interval}\rangle \ )

END
\]

The IN, ISA and WITH clauses are optional. Moreover, the label and value part are optional within the WITH clause. These clauses specify that certain relationships are no longer true for a certain time period. If none of these clauses is present, the UNTELL command is interpreted as a request to terminate all the active relationships of the object UNTELLed except its being an instance of Proposition. The temporal component of any of the above clauses is also optional. If it is absent, the default (at systime..+) is understood to be in its place. The examples in the rest of this section illustrate the semantics of the UNTELL command.

Let us reconsider the TELL command defining Ron in section 2.4.1. The propositions corresponding to the attribute address are the following:

\(^{8}\text{I.e., by a command TELL and not by a deductive rule.}\)
p33 := <Ron, home_address, r_addr, t33>
p34 := <p33, instanceOf, P, t34>
p35 := <p34, instanceOf, InstanceOf, t35>

P is the inducing attribute class defined at the class Person. Now if we want to inform the system that from 1988 on the address of Ron is no longer r_addr, we can execute the following TELL command:

```
TELL TOKEN Ron
     WITH
       address
         home_address : r_addr ( at 1988 .. *)
END
```

The effect of this statement is that the system stops believing that Ron's home address is r_addr after 1988 as soon as the above statement is processed. However, the system's beliefs about the address of Ron before 1988 will remain unchanged. Note that this update can be either pro-active or post-active i.e., it can be executed either before or after the beginning of 1988.

The above can be achieved if t35 is no longer considered a persistent interval and the following propositions and temporal relationships are added in the knowledge base:

```
p68 := <p33, instanceOf, P, t68>
p69 := <p68, instanceOf, InstanceOf, t69>
t35 coends systime
t68 costarts t34
t68 coends 1987
persistent(t69)
t69 costarts systime
```

Figure 2.10 depicts the participating time intervals before and after the update. The dashed interval shown only once represents the time specified by the TELL operation.

It is interesting to note that the above TELL commands terminated the belief of the system that p33 is an attribute of Ron i.e., it is an instance of the built-in class Attribute. This can happen only by specifying it explicitly in the TELL command, for example:

```
TELL TOKEN Ron
     WITH
       address, attribute
         home_address : r_addr ( at 1988 .. *)
END
```

If no time component is present in any of the IN, ISA or WITH clauses then the clause should be interpreted as stating that the system must no longer believes that the respective relationships hold during their previously defined time periods. For example, assume that we have learned that the information about Ron's home address is incorrect. If we execute the command

```
TELL TOKEN Ron
     WITH
       address home_address
END
```
Before UNTELL:

\[
\begin{array}{c}
\text{1986/10/01} \\
\hline
\text{t34} \\
\hline
\text{t35} \\
\hline
\text{1988*} \\
\end{array}
\]

After UNTELL:

\[
\begin{array}{c}
\text{t34} \\
\hline
\text{t35} \quad \text{systime} \\
\hline
\text{t68} \\
\hline
\text{1987/12/31} \\
\hline
\text{t69} \\
\end{array}
\]

Figure 2.10: Time intervals before and after an UNTELL
The above RETELL command has the effect that t45 is no longer considered a persistent interval. Moreover, the following propositions and temporal relationships are added in the knowledge base:

\[ p78 := \langle p77, \text{instanceOf}, P, t78 \rangle \]
\[ p79 := \langle p78, \text{instanceOf}, \text{InstanceOf}, t79 \rangle \]

\[ t45 \text{ coends systime} \]

\[ t78 \text{ costarts t44} \]
\[ t78 \text{ coends 1987} \]

\[ \text{persistent}(t78) \]
\[ t79 \text{ costarts systime} \]

\[ p80 := \langle \text{ Hector, home_address, h_addr1}, t80 \rangle \]
\[ p81 := \langle p80, \text{instanceOf}, P, t81 \rangle \]
\[ p82 := \langle p81, \text{instanceOf}, \text{InstanceOf}, t82 \rangle \]

\[ t80 \text{ at 1988} \]
\[ t81 \text{ at 1988} \]

\[ t82 \text{ costarts systime} \]
\[ \text{persistent}(t82) \]

The propositions p78 and p79 together with their corresponding temporal relationships deal with the beliefs of the system that are not affected by the update while propositions p80-p82 effectively enter the new information in the knowledge base.

### 2.6 Iteration in Telos: the FOR command

We introduce an iterative construct with the following syntax:

\[
\text{FOR } \langle \text{variable} \rangle \text{ IN } \langle \text{set expression} \rangle
\]

This command provides the user with a useful iterative facility. The effect of a command

\[
\text{FOR } x \text{ IN } s
\]

is that any subsequent statements using the variable \( x \) will be executed once for each element of the set \( s \) until the variable \( x \) is redeclared by another FOR statement.

For example, the sequence of commands

\[
\text{FOR } x \text{ IN krypton.author}
\]
\[
\text{TELL TOKEN } x
\]
\[
\text{WITH}
\]
\[
\text{address}
\]
\[
\text{work_address : kr_ltd_addr}
\]
\[
\text{END}
\]

appends a common work address to each of the authors of Krypton. Similarly, the commands

```
FOR x IN Winston.task

TELL TOKEN Fox
    IN referee_manager
    WITH
        interest
        :gps;
        :kr
        task
        :x
END

UNTELL TOKEN Winston
    WITH
        task
END

introduce Fox as a new referee manager who takes over all Winston's jobs.
Chapter 3

A Knowledge-Level Analysis of the Telos System

3.1 Introduction

There are many reasons for presenting a formal account of a knowledge representation language. Firstly, and perhaps most importantly, such an account serves to concisely express the meaning of the various language constructs in a thorough and organized manner. Indeed, with Telos, as with many other languages, ambiguities and inconsistencies were discovered during the process of constructing a formal account of the language. Secondly, a formal account of a knowledge representation language explains when a knowledge base is well-formed, when it is consistent, and what exactly it says about its intended subject matter, independent of any implementation.

3.2 Methodological Considerations

In [BL86], Brachman and Levesque argue that there are at least three orthogonal ways of looking at a knowledge representation system. The first deals with what they call the systems engineering level. This level is concerned with the techniques and principles needed by a user for representing and structuring the knowledge that will be entered in the knowledge base. An appropriate knowledge representation language can offer solutions to most of the user’s problems at this level. Telos makes an attempt to tackle the problems faced at this level by a knowledge engineer.

The second viewpoint for looking at a knowledge representation system is the knowledge level (see also [New82]). At this level, according to Levesque and Brachman in [LB86], “a knowledge base is characterized functionally, in terms of what it can be asked or can be told about the domain. Essentially, a knowledge base is treated as an abstract data type that interacts with a user or a system only through a small set of operations”. This chapter looks at the Telos system from this specific viewpoint.

Finally, the last viewpoint for looking at a knowledge representation system is the symbol level. It deals with the data structures and algorithms used by the system for implementing the operations specified at the knowledge level. The formalization of Telos was of great use in the implementations described in [Kou88] and [TK] which provide an account of the Telos system at this level.

45
variable symbols: for each sort \( i \), there are variables \( v_i, v'_i, \ldots \) of sort \( i \).

quantifiers symbols: for each \( i \) there is a universal and an existential quantifier symbol \( \forall_i \) and \( \exists_i \), respectively.

predicate symbols: for each \( n > 0 \) and each \( n \)-tuple \( (i_1, \ldots, i_n) \) of sorts, there is a set (possibly empty) of \( n \)-place predicate symbols, each of which is said to be of sort \( i_1 \times \cdots \times i_n \).

constant symbols: for each sort \( i \) there is a set (possibly empty) of constant symbols each of which is said to be of sort \( i \).

function symbols: for each \( n > 0 \) and each \( (n+1) \)-tuple \( (i_1, \ldots, i_n, i_{n+1}) \) of sorts, there is a set (possibly empty) of \( n \)-place function symbols, each of which is said to be of sort \( i_1 \times \cdots \times i_n \times i_{n+1} \).

Each term is assigned a unique sort by the following recursive definition:

- Any variable or constant symbol of sort \( i \) is a term of sort \( i \).
- If \( t_1, \ldots, t_n \) are terms of sort \( i_1, \ldots, i_n \) respectively, and \( f \) is a function symbol of sort \( i_1 \times \cdots \times i_n \times i_{n+1} \), then \( f(t_1, \ldots, t_n) \) is a term of sort \( i_{n+1} \).

An atomic formula is a sequence \( P(t_1, \ldots, t_n) \) consisting of a predicate symbol of sort \( i_1 \times \cdots \times i_n \) and terms \( t_1, \ldots, t_n \) of sort \( i_1, \ldots, i_n \) respectively. The non-atomic well formed formulas (or wffs) are constructed using the connectives \( \neg, \land, \lor, \rightarrow, \leftrightarrow \) and the quantifier symbols. We use the more intuitive notation \( \forall v_n/i \) and \( \exists v_n/i \) instead of \( \forall v_n^d \) and \( \exists v_n^d \).

"Types" in \( \mathcal{L} \)

We want \( \mathcal{L} \) to exhibit a certain type hierarchy \([\text{Cop71}]\). For us the notion of type of a closed term of \( \mathcal{L} \) is the formal analogue of the notion of the level of a Telos proposition defined in section 2.1.2. We introduce a function \( \text{type} \) that takes as argument a symbol (constrained to be a ground term, as noted above), in \( \mathcal{L} \) and returns a natural number or the constant \( \omega \).

\[
\text{type} : \text{Proposition} \to \mathbb{N} \cup \{\omega\}
\]

where \( \text{type}(x) = \omega \) if \( x \in \text{OmegaClass} \) and \( \text{type}(x) \in \mathbb{N} \) otherwise. \( \text{Proposition} \) is sort of \( \mathcal{L} \).

Notational Conventions

We adopt the following conventions with regard to the symbols in \( \mathcal{L} \). We will write \( (\exists x, y) \) instead of \( (\exists x)(\exists y) \) as well as \( (\forall x, y) \) instead of \( (\forall x)(\forall y) \). Predicate and function symbols will be written with lower case letters, variables will usually be single letters (of either case and with or without sub-scripts)\(^2\) and constants will also be of mixed case (but will either be identified as such or be obvious from the context).

In addition, we make use of the infix form of certain predicate and function symbols. We will present the sort and type information in a shorthand manner rather than explicitly listing the full expression in \( \mathcal{L} \). For example, to state (meta-linguistically) that a predicate \( p \) takes an argument of sort \( \text{Proposition} \) and type 0, we will write:

\[
p : \text{Proposition}^0
\]

\(^2\) Since predicate symbols are never used as arguments to other predicates there should be no ambiguity.
instead of
\[ p(x) \iff (x \in \text{Proposition}) \land (\text{type}(x) = 0). \]

And, as another example, to state that a function \( f \) takes an argument of sort \( \text{Proposition} \) and
\( \text{type} \) 1, and has a value of sort \( \text{Proposition} \) and \( \text{type} \) 2, we will write:
\[ f : \text{Proposition}^1 \rightarrow \text{Proposition}^2 \]

The justification for this convention for functions is in [End72], where it is proven that one can restrict the arguments to, and value of, a function and then show its equivalence to standard one-sorted logic.

If the type superscript is \( n \), we mean by this that the type is any integer \( > 1 \), and if the
superscript is \( n + 1 \), we mean \( 1 \) greater than \( n \). If the type is \( 0 \), we drop the type information completely.

### 3.3.2 Functions, Predicates, Constants and Axioms

As previously stated, we consider a knowledge base as being represented by a set of sentences in
a first-order logic. In this section we will build the initial theory \( KB_0 \). In section 3.4 we will show how knowledge bases are formed from previous knowledge bases, how they are updated and how they are queried.

Since time is pervasive in Telos, in the sense that every object has a time component, we must first present the axioms dealing with time.

**Axiomatization of Time**

We axiomatize time using Allen and Hayes’ (see [AH85]) proposal as a basis. The basic premise of their proposal for time is that any model which is cast in terms of the primitives they give must (among other things) allow the possibility that two intervals meet. This is defined as the situation where there is no interval between the two intervals which meet, and that there is no time period which both intervals share\(^2\). The entire theory can be axiomatized in terms of the single two-place relation:

\[ \text{meets} : \text{TimeInterval} \times \text{TimeInterval} \]

and three constants \( +\text{Infinity}, -\text{Infinity}, \) and \( \text{Alltime} \).

In the following, we will use the notational simplification of writing the \( \text{meets} \) relation as an in-fix predicate and we will write conjunctions such as \(((i \text{ meets } j) \land (j \text{ meets } k)) \) as \( (i \text{ meets } j \text{ meets } k) \). There are five basic axioms (from Allen) which we modify slightly.

- **Uniqueness of Beginning and Ending Positions\(^2\):**
  \[
  (\forall i, j)(\exists t)(i \text{ meets } t \land j \text{ meets } t) \implies (\forall k)(i \text{ meets } k \implies j \text{ meets } k)
  
  (\forall i, j)(\exists t)(t \text{ meets } i \land t \text{ meets } j) \implies (\forall k)(k \text{ meets } i \implies k \text{ meets } j)
  
\]

\(^2\)Justification for making this a basis of a naive theory of time can be found in [AH83], [AH84] as well as in [AH85].

\(^3\)This does not imply that these positions, especially the end of a time interval, is known.
• Ordering Axioms:
  \((\forall i, j, k, l)(i \text{ meets } j \land k \text{ meets } l) \implies (i \text{ meets } l) \lor (\exists m)(i \text{ meets } m \text{ meets } l) \lor (\exists n)(k \text{ meets } n \text{ meets } j)\)

• All Time Intervals meet Two Others:
  \((\forall i, j, k)(j \text{ meets } i \text{ meets } k) \implies (j \text{ meets } i \text{ meets } +\text{Infinity}) \lor (-\text{Infinity} \text{ meets } i \text{ meets } k) \lor (i = \text{Alltime})\)

• Existence of the Sum (+) of Intervals:
  \((\forall i, j)(j \neq +\text{Infinity}) \land (i \neq -\text{Infinity})
  \implies (i \neq \text{Alltime}) \land (j \neq \text{Alltime}) \land (i \text{ meets } j)
  \implies (\exists k, l, m)(m = (i + j)) \land (k \text{ meets } i \text{ meets } j \text{ meets } l)
  \land (k \text{ meets } (i + j) \text{ meets } l)\)

We introduce the constants \textit{Alltime}, +\text{Infinity}, -\text{Infinity}, all of which are of sort \textit{TimeInterval}. The constants +\text{Infinity} and -\text{Infinity} are used to denote the fact that an interval can continue on in one or both directions. \textit{Alltime} is related to all other intervals as follows:

\((\forall t)\text{Alltime over } t^1\)

The following are the derived time predicates. These are all two-place relation of sort \textit{TimeInterval} x \textit{TimeInterval} and are defined in terms of \textit{meets}. Note that \((\forall t_1, t_2/\text{TimeInterval})\) is assumed for the following formulas.

\((t_1 \text{ equals } t_2) \iff (t_1 \text{ at } t_2) \iff
  (\exists k, l)(k \text{ meets } t_1 \text{ meets } l \land k \text{ meets } t_2 \text{ meets } l)\)

\((t_1 \text{ during } t_2) \iff (t_1 \text{ over } t_2) \iff
  (\exists i, j, k, l)(i \text{ meets } j \text{ meets } t_1 \text{ meets } k \text{ meets } l \land i \text{ meets } t_2 \text{ meets } l)\)

\((t_1 \text{ starts before } t_2) \iff (t_2 \text{ starts after } t_1) \iff
  (\exists i, j)(i \text{ meets } t_1 \land i \text{ meets } j \text{ meets } t_2)\)

\((t_1 \text{ ends before } t_2) \iff (t_2 \text{ ends after } t_1) \iff
  (\exists i, j)(t_1 \text{ meets } i \text{ meets } j \land t_2 \text{ meets } j)\)

\((t_1 \text{ right before } t_2) \iff (t_2 \text{ right after } t_1) \iff
  t_1 \text{ meets } t_2\)

\((t_1 \text{ before } t_2) \iff (t_2 \text{ after } t_1) \iff
  (\exists i)(t_1 \text{ meets } i \text{ meets } t_2)\)

---

5 The symbol \(\oplus\) used above denotes the exclusive disjunction of wffs. Also, see below for the introduction of +\text{Infinity}, -\text{Infinity}, and \textit{Alltime}.

6 It can be shown that when \((i + j)\) exists, it is unique, and that + is associative. See [AH85].

7 This is implied from the fourth time axiom above, but we have this form here for clarity.

8 Note that \((t_1 \text{ equals } t_2) = (t_1 \text{ during } t_2) \land (t_2 \text{ during } t_1)\).

9 This is essentially the meets relation with respect to one end of the interval.
\[(t_1 \text{ costs } t_2) \iff (3i, j, k)i \text{ meets } t_1 \text{ meets } j \text{ meets } k \land i \text{ meets } t_2 \text{ meets } k\]

\[(t_1 \text{ coends } t_2) \iff (3i, j, k)i \text{ meets } j \text{ meets } t_1 \text{ meets } k \land i \text{ meets } t_2 \text{ meets } k\]

\[(t_1 \text{ overlaps } t_2) \iff (t_2 \text{ overlapped } - \text{ by } t_1) \]

\[(3i, j, k, l, m)i \text{ meets } t_1 \text{ meets } l \text{ meets } m \land i \text{ meets } j \text{ meets } t_2 \text{ meets } m \land j \text{ meets } k \text{ meets } l\]

Conventional Time Intervals

The conventional time intervals offered in Telos can be formally defined using lists of integers. See, for example [Lad86]. Additionally, the obvious time relationships are asserted about them, e.g. [1986] before [1987].

Other Constant Symbols

For every built-in class of Telos a similarly named constant is included in \(L\). These include:

- \(Token, \text{SClass, M1Class, M2Class, ...}\)
- \(TimeInterval\)
- \(InstanceOf, IsA\)
- \(Integer, String\)

The \(\omega\)-classes:

- \(Proposition, Class, Individual, Attribute\)
- \(IndividualClass, AttributeClass, OmegaClass\)

Predicates and Functions

The basic unit in Telos is a proposition.

\[\text{prop} : Proposition \times Proposition \times Proposition \times Proposition \times TimeInterval\]

A given proposition is made up of four parts: from, label, to, and when. In order to get hold of of each of these components we define the following functions:

\[\text{from, label, to, when} : Proposition \rightarrow Proposition\]

\[(\forall p, z, y, z \in Proposition)(\forall t \in TimeInterval)\text{prop}(p, x, y, z, t) \implies \text{from}(p) = z \land \text{label}(p) = y \land \text{to}(p) = z \land \text{when}(p) = t\]

In order to constrain the allowable configurations of these objects and the relationships that they participate in, we introduce the following predicates, whose definition is given later:

- \(InstanceOf : Proposition^3 \times Class^3 \times TimeInterval \times TimeInterval\)
- \(isA : Class^3 \times Class^3 \times TimeInterval \times TimeInterval\)
attribute : individual × proposition^n × proposition^m × timeinterval × timeinterval

instanceOf denotes the instantiation relation, isa denotes the specialization relation, while
attribute asserts an attribute relationship between two propositions.

Basic Framework Axioms

Proposition

(∃l/proposition) prop(proposition, proposition, l, proposition, alltime)
instanceOf(proposition, individualclass, alltime, alltime)
instanceOf(proposition, omegaclass, alltime, alltime)

Class

(∃l/proposition) prop(class, class, l, class, alltime)
isA(class, proposition, alltime, alltime)
instanceOf(class, alltime, alltime)
instanceOf(class, omegaclass, alltime, alltime)

Individual

(∃l/proposition) prop(individual, individual, l, individual, alltime)
isA(individual, proposition, alltime, alltime)
instanceOf(individual, individualclass, alltime, alltime)
instanceOf(individual, omegaclass, alltime, alltime)

IndividualClass

(∃l/proposition) prop(individualclass, individual, l, individual, alltime)
isA(individualclass, class, alltime, alltime)
isA(individualclass, individual, alltime, alltime)
instanceOf(individualclass, omegaclass, alltime, alltime)

Attribute

(∃l/proposition) prop(attribute, proposition, l, proposition, alltime)
instanceOf(attribute, proposition, attribute, alltime, alltime)
instanceOf(attribute, attributeclass, alltime, alltime)
instanceOf(attribute, omegaclass, alltime, alltime)

AttributeClass

(∃l/proposition) prop(attributeclass, attributeclass, l, attributeclass, alltime)
isA(attributeclass, attribute, alltime, alltime)
isA(attributeclass, class, alltime, alltime)
instanceOf(attributeclass, attributeclass, alltime, alltime)
instanceOf(attributeclass, omegaclass, alltime, alltime)

\(^{10}\) See below for definition of instanceOf.
\(^{11}\) See below for definition of isa.
OmegaClass

\( (\exists l/\text{Proposition})\text{prop}(\text{OmegaClass}, \text{OmegaClass}, l, \text{OmegaClass}, \text{Alltime}) \)
\( \text{instanceOf}(\text{OmegaClass}, \text{OmegaClass}, \text{Alltime}, \text{Alltime}) \)

Token, S.Class, M.Class, ...

\( (\exists l/\text{Proposition})\text{prop}(\text{Token}, \text{Proposition}, l, \text{Proposition}, \text{Alltime}) \)
\( (\exists l/\text{Proposition})\text{prop}(\text{S.Class}, \text{Proposition}, l, \text{Proposition}, \text{Alltime}) \)
\( (\exists l/\text{Proposition})\text{prop}(\text{M.Class}, \text{Proposition}, l, \text{Proposition}, \text{Alltime}) \)

\( \text{instanceOf}(\text{Token}, \text{Proposition}, \text{Alltime}, \text{Alltime}) \)
\( \text{instanceOf}(\text{S.Class}, \text{Proposition}, \text{Alltime}, \text{Alltime}) \)
\( \text{instanceOf}(\text{M.Class}, \text{Proposition}, \text{Alltime}, \text{Alltime}) \)

\( \text{IntegrityConstraint}, \text{DeductiveRule}, \text{and Assertion} \)

\( \text{prop}(\text{IntegrityConstraint}, \text{Proposition}, \text{integrityConstraint}, \text{Assertion}, \text{Alltime}) \)
\( \text{instanceOf}(\text{IntegrityConstraint}, \text{AttributeClass}, \text{Alltime}, \text{Alltime}) \)
\( \text{prop}(\text{DeductiveRule}, \text{Proposition}, \text{deductiveRule}, \text{Assertion}, \text{Alltime}) \)
\( \text{instanceOf}(\text{DeductiveRule}, \text{AttributeClass}, \text{Alltime}, \text{Alltime}) \)
\( (\exists l/\text{Proposition})\text{prop}(\text{Assertion}, \text{Assertion}, l, \text{Assertion}, \text{Alltime}) \)
\( \text{instanceOf}(\text{Assertion}, \text{IndividualClass}, \text{Alltime}, \text{Alltime}) \)
\( \text{instanceOf}(\text{Assertion}, \text{S.Class}, \text{Alltime}, \text{Alltime}) \)
\( \text{instanceOf}(\text{IntegrityConstraint}, \text{S.Class}, \text{Alltime}, \text{Alltime}) \)
\( \text{instanceOf}(\text{DeductiveRule}, \text{S.Class}, \text{Alltime}, \text{Alltime}) \)

IsA

\( \text{prop}(\text{IsA}, \text{Class}, \text{isA}, \text{Class}, \text{Alltime}) \)
\( \text{instanceOf}(\text{IsA}, \text{S.Class}, \text{Alltime}, \text{Alltime}) \)

InstanceOf

\( \text{prop}(\text{InstanceOf}, \text{Proposition}, \text{instanceOf}, \text{Proposition}, \text{Alltime}) \)
\( \text{instanceOf}(\text{InstanceOf}, \text{S.Class}, \text{Alltime}, \text{Alltime}) \)

Axioms

A proposition's components are unique:

\( (\forall p, s, s', l, l', v, v'/\text{Proposition})(\forall t, t'/\text{TimeInterval}) \)
\( \text{prop}(p, s, l, v, t) \land \text{prop}(p, s', l', v', t') \implies (s = s') \land (l = l') \land (v = v') \land (t = t') \)
Temporal Maximalitiy of Attributes: No two objects have the same source and destination during overlapping history time intervals, in every belief state of the knowledge base. This ensures that attributes are asserted over the maximal time interval for which they hold.

\[(\forall p_1, p_2/\text{Proposition})(\forall t_1, t_2, t_3/\text{TimeInterval})\]
\[\text{instanceOf}(p_1, \text{Attribute}, t_1, t_2) \land \text{instanceOf}(p_2, \text{Attribute}, t_2, t_3) \land\]
\[(\text{from}(p_1) = \text{from}(p_2)) \land (\text{label}(p_1) = \text{label}(p_2)) \land\]
\[(t_1 \text{ overlaps } t_2) \lor (t_2 \text{ overlaps } t_1) \implies (p_1 = p_2)\]

\(\text{isA is transitive:}\)

\[(\forall p_1, p_2, p_3/\text{Proposition})(\forall t_1, t_2/\text{TimeInterval})\]
\[\text{isA}(p_1, p_2, t_1, t_2) \land \text{isA}(p_2, p_3, t_3, t_4) \implies\]
\[\text{isA}(p_1, p_3, (t_1 \times t_3), (t_2 \times t_4))\]

\(\text{Specialization Postulate: an instance of a class is also an instance of every superclass of this class.}\)

\[(\forall p/\text{Proposition})(\forall c_1/\text{Class})(\exists c_2/\text{Class})(\exists t_1, t_2, t_3, t_4/\text{TimeInterval})\]
\[\text{isA}(c_1, c_2, t_1, t_2) \land \text{instanceOf}(p, c_1, t_3, t_4) \implies\]
\[\text{instanceOf}(p, c_2, (t_1 \times t_3), (t_2 \times t_4))\]

\(\text{IsA and time: Classes can only be IsA related while they concurrently exist.}\)

\[(\forall p_1, p_2/\text{Proposition})(\forall t_1, t_2/\text{TimeInterval})\]
\[\text{isA}(p_1, p_2, t_1, t_2) \implies (t_1 \text{ during } \text{when}(p_1) \land \text{when}(p_2))\]

The Instantiation Constraint: if a proposition \(p_1\) is an instance of a proposition \(p_2\) then \(p_1\)'s source and destination components are constrained to be instances of \(p_2\)'s source and destination components, respectively. This constraint must hold during the times that the propositions exist and are believed.

\[(\forall p_1, p_2/\text{Proposition}, t_1, t_2/\text{TimeInterval})\]
\[\text{instanceOf}(p_1, p_2, t_1, t_2) \implies\]
\[\text{instanceOf}(\text{from}(p_1), \text{from}(p_2), t_3, t_4) \land\]
\[\text{instanceOf}(\text{to}(p_1), \text{to}(p_2), t_3, t_4) \land (t_1 \text{ during } t_3) \land\]
\[\land (t_4 \text{ during } t_2) \land (t_1 \text{ during } t_4) \land (t_2 \text{ during } t_3) \land\]
\[\text{when}(p_1) \text{ overlaps } \text{when}(p_2)\]

All propositions are instances of \(\text{Proposition}\):

\[(\forall p/\text{Proposition})(\exists t_1, t_2/\text{TimeInterval})\text{instanceOf}(p, \text{Proposition}, t_1, t_2)\]

All individual propositions are instances of \(\text{Individual}\):

\[(\forall p_i/\text{Proposition})(\exists t_i/\text{TimeInterval})\]
\[\text{prop}(p_i, p, i, t_i) \implies\]
\[\text{instanceOf}(p, \text{Individual}, t_i, t_i)\]

The type hierarchy:

\(^{12}\text{"\times" is the intersection operator which can easily be defined in terms of the primitive time operators.}\)
(∀p/Proposition)(∃t₁, t₂/TimeInterval)
instanceOf(p, Token, t₁, t₂) ⇐⇒ Type(p) = 0

(∀p/Proposition)(∃t₁, t₂/TimeInterval)
instanceOf(p, S.Class, t₁, t₂) ⇐⇒ Type(p) = 1

(∀p/Proposition)(∃t₁, t₂/TimeInterval)
instanceOf(p, M.1Class, t₁, t₂) ⇐⇒ Type(p) = 2

(∀p/Proposition)(∃t₁, t₂/TimeInterval)
instanceOf(p, M.nClass, t₁, t₂) ⇐⇒ Type(p) = n + 1 (Where n ≥ 1)

(∀p/Proposition)
instantaneous(p, OmegaClass, Alltime, Alltime) ⇐⇒ Type(p) = ω

(∀p/Proposition)(∀t₁, t₂/TimeInterval)
Type(p) > 0 ⇐⇒ instanceOf(p, Class, t₁, t₂)

The IsA hierarchy:

(∀p/Class)(∃t₁, t₂/TimeInterval)
isA(p, Token, t₁, t₂) ⇐⇒ instanceOf(p, S.Class, t₁, t₂)

(∀p/Class)(∃t₁, t₂/TimeInterval)
isA(p, S.Class, t₁, t₂) ⇐⇒ instanceOf(p, M.1Class, t₁, t₂)

(∀p/Class)(∃t₁, t₂/TimeInterval)
isA(p, M.1Class, t₁, t₂) ⇐⇒ instanceOf(p, M.2Class, t₁, t₂)

...

Any proposition that is not an individual is an attribute:

(∀p/Proposition)(¬(∃t₁, t₂/TimeInterval)
instanceOf(p, Individual, t₁, t₂) ⇐⇒
(∃t₃, t₄/TimeInterval)instanceOf(p₁, Attribute, t₃, t₄))

The definition of the instanceOf predicate:\(^{13}\):

(∀p₁/Proposition)(∀c/Class)(∀t₁, t₂/TimeInterval)
c ∉ InstanceOf ⇐⇒ instanceOf(p₁, c, t₁, t₂)

(∀p₁/Proposition)prop(p₁, p₁, instanceOf, c, t₁) ∧
(∀p₂/Proposition)prop(p₂, p₂, instanceOf, InstanceOf, t₂)

(∀p₁, p₂/Proposition)(∀t/TimeInterval)
prop(p₁, p₂, instanceOf, InstanceOf, t) ⇐⇒
instanceOf(p₁, InstanceOf, t, Alltime)

Similarly, for IsA propositions:

(∀c₁, c₂/Class)(∀t₁, t₂/TimeInterval)
isA(c₁, c₂, t₁, t₂) ⇐⇒
(∀p/Proposition)prop(p, c₁, IsA, c₂, t₁) ∧ instanceOf(p, IsA, t₁, t₂)

This completes the specification of the initial theory KB₀.

\(^{13}\)This case the infinite regression of the instance of relation.
3.3.3 Integrity Constraints in Telos

In this chapter, knowledge bases are formally considered to be sets of sentences in \( \mathcal{L} \). Integrity constraints are also viewed as logical formulas\(^{14}\) which must be satisfied in the initial knowledge base and whenever an update occurs. We now explore some alternative definitions of integrity constraint satisfaction and give the rationale for adopting one of them.

Until recently, two definitions of integrity constraint satisfaction prevailed in the literature. One has to do with the notion of logical consistency: a knowledge base \( KB \) satisfies an integrity constraint \( IC \) if and only if \( KB \cup IC \) is consistent \([SK87]\). The other definition exploits the notion of entailment: a knowledge base \( KB \) satisfies an integrity constraint \( IC \) if and only if \( KB \models IC \) \([LT85]\). Reiter has recently challenged these views by pointing to a number of cases where these definitions do not render what it is intuitively expected \([Rei88]\). He argued that integrity constraints are epistemic in nature and went on to propose a modal logic that captures more gracefully the intuitive notion of integrity constraint satisfaction.

Although we agree with Reiter's proposal, we prefer to use the second definition above because, when combined with the limited form of the Closed World Assumption \([Rei78]\) assumed by the implementation described in \([TK]\), it gives us the intuitively expected results in all but trivial cases. Our choice was also motivated by the existence of a number of efficient algorithms for integrity constraint simplification and integrity constraint checking for this case (see \([BDM88]\) for an excellent discussion).

3.3.4 Non-monotonicity in Telos: Persistence of Beliefs

The persistence of beliefs rule introduced in section 2.2.2 introduces a flavor of non-monotonicity in Telos. In effect, what we want is that the system keeps on believing things TELled until it receives information to the contrary by UNTELL or RETELL.

This non-monotonic feature can be captured formally by the introduction of the following default rule \( \delta \) (using Reiter's default logic \([Rei80a]\)) which is the formal counterpart of the persistence of beliefs rule:

\[
(\exists p,t_0)\text{InstanceOf}(p,\text{InstanceOf},t_0,t) \rightarrow M \text{ (t meets } \pm \infty) \\
(\forall t \text{ (meets } \pm \infty))
\]

This default rule can be understood as saying that if it is consistent to assume that a belief time interval is semi-infinite to the right then we can assume so.

Let us now assume that \( S \) is a set of WFS of \( \mathcal{L} \). The pair \( T = (\{\delta\}, S) \) is called a default theory in \([Rei80a]\). It is actually a normal default theory therefore it has a unique extension\(^{15}\). In what follows, we denote the unique extension of \( T \) by \( \overline{T} \). In particular, \( KB \) will denote the unique extension of the default theory \( T^* = (\{\delta\}, KB) \).

3.4 The Functional Specification of the Telos System

We present a functional specification of the Telos system using the approach of \([LB88]\). We characterize knowledge bases not in terms of what data structures they use to represent knowledge but rather in terms of what they can be told or asked about the domain of the application. A

\(^{14}\) Section 3.5 gives the function that translates any assertion in Telos to formulas in \( \mathcal{L} \).

\(^{15}\) The extension of this default theory can be intuitively defined as the set of all the logical consequence of the formulas in \( S \) taking of course, \( S \) into account. The reader who is not familiar with default logic is advised to see \([Rei80a]\) for details and proofs.
functional specification of this kind tell us what the system knows about its domain as well as what follows from what it knows. Our approach differs from [LB86] in that we do not provide a semantics for Telos but only a translation of Telos statements to collections of “equivalent” statements in C.

A knowledge base is treated as an abstract data type interacting with the user only through a small set of operations. These operations are TELL, UNTELL, RETELL, RETRIEVE, and ASK. They formally specify the commands provided by our system, therefore they can be used to verify the correctness of any proposed implementation.\footnote{We assume that these operations accept statements of Telos that are fully expanded using the default rules presented in chapter 2. Also, each FOR statement is converted into the appropriate equivalent sequence of knowledge base operations.}

In the sections below, we first present the Telos operations in a high-level manner: we explain that they operate on sets of sentences without explaining how they actually create these sets. We show below in 3.5 the exact form of these new sets of sentences and how they are created by each Telos operation.

In this section we will show how the knowledge base is updated to form subsequent knowledge bases. Recall that $KB_0$ contains the axioms and constraints presented in section 3.3.2, while $IC_0$, $DR_0$ are initially empty. New knowledge bases ($KB_n$, $IC_n$, and $DR_n$) are constructed as shown below. Note that when we speak of a knowledge base we mean the union of the above subsets, while when we speak of $KB_n$ we mean all the sentences of the knowledge base except for the deductive rules and integrity constraints.

The knowledge base can be queried by RETRIEVE and ASK. In fact, as mentioned above in section 3.3.4, it is the extension of the knowledge base under the default rule for persistence of beliefs that is queried.

3.4.1 The TELL Operation

The TELL operation can be functionally understood as:

$$\text{TELL} : KB \times IC \times DR \times O \times \text{Time} \rightarrow KB \times IC \times DR$$

TELL takes as arguments a set of sentences $KB_n$, a set of integrity constraints $IC_n$ on $KB_n$, a set of deductive rules $DR_n$, a set of object definitions $O$, and a representation of the current system time and produces a new knowledge base $KB_{n+1}$ and an enhanced set of integrity constraints and deductive rules $IC_{n+1}$ and $DR_{n+1}$, respectively. The TELL operation takes place only if all the integrity constraints in $IC_{n+1}$ are satisfied\footnote{Remember that an integrity constraint in Telos is said to be satisfied if it is entailed by the knowledge base.} and $(KB_{n+1} \cup DR_{n+1})$ is consistent.

$$\text{TELL}(KB_n, IC_n, DR_n, O, \text{sys time}) = (KB_{n+1}, IC_{n+1}, DR_{n+1})$$

3.4.2 The UNTELL Operation

The UNTELL operation can be functionally understood as:

$$\text{UNTELL} : KB \times IC \times DR \times O \times \text{Time} \rightarrow KB \times IC \times DR$$

UNTELL takes the same arguments as TELL and also produces $KB_{n+1}$, $IC_{n+1}$, and $DR_{n+1}$. (The contents of these sets are of course different since these are different operations). As with the
The \textit{TELL} operation, \textit{UNTELL} takes place only if all the integrity constraints in $IC_{n+1}$ are satisfied and $(KB_{n+1} \cup DR_{n+1})$ is consistent.

$$\textit{UNTELL}(KB_n, IC_n, DR_n, O, systime) = (KB_{n+1}, IC_{n+1}, DR_{n+1})$$

### 3.4.3 The \textit{RETELL} Operation

The \textit{RETELL} operation can be functionally understood as:

$$\textit{RETELL}: KB \times IC \times DR \times O \times Time \rightarrow KB \times IC \times DR$$

\textit{RETELL} takes the same arguments and returns the same kinds of sets as \textit{TELL} and \textit{UNTELL}. It does this using a composition of the \textit{TELL} and the \textit{UNTELL} operation on differing parts of $O$. The \textit{RETELL} operation only takes place if both the \textit{UNTELL} and \textit{TELL} operations succeed.

Let $\textit{UNTELL}(KB_n, IC_n, DR_n, O', systime) = (KB_{n+1}, IC_{n+1}, DR_{n+1})$

and let $\textit{TELL}(KB_{n+1}, IC_{n+1}, DR_{n+1}, O'', systime) = (KB_{n+2}, IC_{n+2}, DR_{n+2})$

then $\textit{RETELL}(KB_n, IC_n, DR_n, O, systime) = (KB_{n+2}, IC_{n+2}, DR_{n+2})$

where $O'$ and $O''$ are constructed simply from $O$ by using the appropriate new and old definitional parts from $O$. See section 3.5.

### 3.4.4 The \textit{RETRIEVE} Operation

\textit{RETRIEVE} is used to query the knowledge base without considering the deductive rules in it.

$$\textit{RETRIEVE}: KB \times \text{Query} \rightarrow \text{Answers}$$

It takes two arguments: a $KB_n$ and a query of the form $<x_1/\tau_1, \ldots, x_n/\tau_n \mid W>$ where $W$ is a wff of $\mathcal{L}$ and $x_1, \ldots, x_n$ are its only free variables, each constrained to be of type $\tau_1, \ldots, \tau_n$, respectively. \textit{RETRIEVE} returns all the possible substitutions for the free variables of the query that make $W$ true in $KB_n$. If we want to prove that a closed formula $W$ is true, we have to pose the query $< |W>$. The answer to such a query will be \textit{yes}, \textit{no}, or \textit{unknown}.

Formally, if query is of the form $< |W>$, then

$$\textit{RETRIEVE}(KB_n, \text{query}) = \begin{cases} 
\text{yes} & \text{if } KB_n \models W \\
\text{no} & \text{if } KB_n \models \neg W \\
\text{unknown} & \text{if } KB_n \not\models W \\
\text{and } KB_n \not\models \neg W 
\end{cases}$$

If query is of the form $<x_1/\tau_1, \ldots, x_n/\tau_n \mid W>$ then

$$\textit{RETRIEVE}(KB_n, \text{query}) = \begin{cases} 
\{ \theta \} & \text{if for every substitution } \theta \in \varnothing, KB_n \models W \theta \\
\{ \} & \text{if there is no substitution } \theta \text{ such that } KB_n \models W \theta 
\end{cases}$$
3.4.5 The ASK Operation

ASK is used to query the entire knowledge base. It can be functionally understood as follows:

\[
ASK : KB \times DR \times Query \rightarrow Answers
\]

ASK adds the deductive rules to the knowledge base it considers. As with RETRIEVE, ASK returns all the possible substitutions for the free variables of the query that make \( W \) true in \( KB_n \) and for a closed formula \( W \) the answer will be yes, no, or unknown.

Formally, if \( query \) is of the form \( \langle \|W\| \rangle \), then

\[
ASK(KB_n, DR_n, query) = \begin{cases} 
  \text{yes} & \text{if } KB_n \cup DR_n \models W \\
  \text{no} & \text{if } KB_n \cup DR_n \models \neg W \\
  \text{unknown} & \text{if } KB_n \cup DR_n \not\models W \\
  & \text{and } KB_n \cup DR_n \not\models \neg W 
\end{cases}
\]

If \( query \) is of the form \( \langle x_1/\tau_1, \ldots, x_n/\tau_n \ | \ W \rangle \) then

\[
ASK(KB_n, DR_n, query) = \begin{cases} 
  \emptyset & \text{if for every substitution } \theta_i \in \emptyset, \ KB_n \cup DR_n \models W \theta_i \\
  \emptyset & \text{if there is no substitution } \theta \text{ such that } KB_n \cup DR_n \models W \theta_i 
\end{cases}
\]

3.5 Translating Telos Definitions

Here we show in detail how, for each operation, the sets of sentences which make up our knowledge base are produced. What we need to show is how to translate any Telos declaration into sentences of our theory, i.e., sentences in \( \mathcal{L} \). Note that the declarations in \( O \) are fully expanded with respect to the default rules outlined in chapter 2. It is interesting to note that the set of sentences generated for a given declaration is relatively small due to the axioms and constraints in the initial theory.

We show how the operations TELL, UNTELL and RETELL map Telos commands into \( \mathcal{L} \). This happens directly for all parts of Telos except for expressions in the assertion language and for certain time constants. For these we need a special translation function \( \sigma \) whose complete definition is given below in 3.7. In the following, whenever a string appears that needs to be handled by \( \sigma \) we assume that it will be called on it and the result used in its place.

For RETRIEVE and ASK the query is translated into \( \mathcal{L} \) by applying \( \sigma \). The system argument above is the current system time represented as a Telos time interval.

3.5.1 How TELL changes the knowledge base

We present the general form of \( O \) for TELL. It is the complete specification for \( \langle t e l l \rangle \) from the Telos grammar (see appendix A) using Greek letters to replace strings in Telos.
TELL α β (θ τ)
COMPONENTS
   <γ₁, γ₂, γ₃>
 IN
   δ (θ₁ τ₁)
 ISA
   ε (θ₂ τ₂)
 WITH
   ζ FROM η
   i : κ (θ₃ τ₃)
   λ : μ $ (θ₄ τ₄)
END

where the above Greek letters range over strings in Telos as follows:

α : <token - or - class >       β : <identifier >
θ : <time - constant >        τ : <time - constant >
γ₁ : <proposition >          γ₂ : <proposition >
γ₃ : <proposition >          δ : <class >
θ₁ : <time - constant >         θ₂ : <time - relation >
i : <class >
θ₃ : <class >
η : <attribute - value >
θ₄ : <time - constant >         θ₅ : <time - relation >
κ : <attribute - label >
θ₆ : <time - constant >         θ₇ : <time - relation >
μ : <assertion >
θ₈ : <time - constant >         θ₉ : <time - relation >
τ : <time - constant >
θ₁₀ : <time - relation >

The symbols in angle-brackets refer to non-terminals, while those without refer to terminals in the Telos grammar. Each of the above Greek symbols (except ω) represent a set of strings which can be empty in accordance with the syntax of Telos.

As explained above, TELL returns the following sets of sentences: KBₙ, ICₙ, and DRₙ. We now show exactly how these sets are constructed. Unless otherwise specified the appropriate sentences below are put into KBₙ. In the following, we use tᵣ and pₙ (where n : 1, 2, . . .) to represent new unique constants, created internally by the system¹⁸. The symbols u, v, w, x, y, z and z' are used below to represent quantified variables in the particular sentence.

Object Definition Part:
if there is no COMPONENTS part:
    prop(β, β, l, β, t₁)
    (t₁ τ₁)
else (there is a COMPONENTS part):
    prop(β, γ₁, γ₂, γ₃, l, t₁)
    (t₁ τ₁)

¹⁸ See 3.3.1 for more on this.
if $\alpha = \text{TOKEN}$ then:

\(\text{instanceOf}(\beta, \text{Individual}, t_3, t_3)\)
\((t_3 \theta \tau)\)
\((t_3 \text{ costarts systime})\)

else ($\alpha = \text{CLASS}$)

\(\text{instanceOf}(\beta, \text{IndividualClass}, t_2, t_3)\)
\((t_2 \theta \tau)\)
\((t_3 \text{ costarts systime})\)

**In-Clause Part:**

\(\text{instanceOf}(\beta, \delta, t_5, t_5)\)
\((t_5 \theta_2 \tau_2)\)
\((t_5 \text{ costarts systime})\)

**IsA-Clause Part:**

\(\text{isA}(\beta, \epsilon, t_7, t_8)\)
\((t_7 \theta_2 \tau_2)\)
\((t_8 \text{ costarts systime})\)

**With-Clause Part - non-assortional case:**

\(\text{prop}(p_1, \beta, i, \kappa, t_9)\)
\((t_9 \theta_3 \tau_3)\)

if $\kappa$ is $\text{Integer - subrange}$ then:

let $\kappa_0$ be $\text{lower - bound}$

and let $\kappa_0$ be $\text{upper - bound}$

\((\forall z/\text{Integer})(\exists t_1, t_2/\text{TimeInterval})\)
\(\text{instanceOf}(z, \kappa, t_1, t_2) \iff z \geq \kappa_1)\land (z \leq \kappa_2)\)\(^{12}\)

if FROM $\eta$ is missing then:

\((\forall z, y, z/\text{Proposition})(\forall u/\text{TimeInterval})\)
\(\text{prop}(x, y, z, z, w) \land (\exists t_1, t_2/\text{TimeInterval})\)
\(\text{instanceOf}(\beta, y, t_1, t_2) \implies \text{instanceOf}(p_1, x, t_0, t_{10})\)
\((t_9 \theta_3 \tau_3)\)
\((t_{10} \text{ costarts systime})\)

else (if FROM $\eta$ is not missing)

\((\forall z, y/\text{Proposition})(\forall w/\text{TimeInterval})\)
\(\text{prop}(x, z, \zeta, y, w) \implies \text{instanceOf}(p_1, x, t_0, t_{10})\)
\((t_9 \theta_3 \tau_3)\)
\((t_{10} \text{ costarts systime})\)

\(^{12}\)Integer is a built in class, and as explained above, $\kappa_1$ is actually $\alpha(v_1)$. Likewise for $\kappa_2$. 
Case 1: \((\tau_1 \text{ equals } w)\)

\((\forall w.\text{Proposition})\text{instanceOf}(\beta, \delta, \tau_1, w) \implies (w \text{ coends systime})\)

Case 2: \(-((\tau_1 \text{ equals } w) \land (\tau_1 \text{ coends } w))\) UNTELL w.r.t. left end of the interval

\((\forall w.\text{Proposition})\text{instanceOf}(\beta, \delta, \tau_1, w) \land (w \text{ over systime}) \land (\tau_1 \text{ during } x) \implies (w \text{ coends systime}) \land (\tau_1 \text{ meets } t_1) \land (t_1 \text{ coends } w) \land \text{instanceOf}(\beta, \delta, t_1, t_2) \land (t_2 \text{ costsarts systime})\)

Case 3: \(-((\tau_1 \text{ equals } w) \land (\tau_1 \text{ coends } w))\) - the right end of the interval

\((\forall w.\text{Proposition})\text{instanceOf}(\beta, \delta, \tau_1, w) \land (w \text{ over systime}) \land (\tau_1 \text{ during } x) \implies (w \text{ coends systime}) \land (t_1 \text{ meets } \tau_1) \land (t_1 \text{ costsarts } w) \land \text{instanceOf}(\beta, \delta, t_1, t_2) \land (t_2 \text{ costsarts systime})\)

Case 4: \(-((\tau_1 \text{ equals } w) \land (\tau_1 \text{ costsarts } w))\) - middle of the interval

\((\forall w.\text{Proposition})\text{instanceOf}(\beta, \delta, \tau_1, w) \land (w \text{ over systime}) \land (\tau_1 \text{ during } x) \implies (w \text{ coends systime}) \land (t_1 \text{ costsarts } w) \land (t_1 \text{ meets } w) \land (\tau_1 \text{ meets } t_2) \land (t_2 \text{ coends } w) \land \text{instanceOf}(\beta, \delta, t_1, t_3) \land \text{instanceOf}(\beta, \delta, t_2, t_4) \land \text{instanceOf}(\beta, \delta, t_3, t_4) \land (t_3 \text{ costsarts systime}) \land (t_4 \text{ costsarts systime})\)

IsA-Clause Part:

The following must hold for the operation to take place.

\((\exists x, y.\text{Proposition})\text{isA}(\beta, \epsilon, x, y) \land (y \text{ over systime}) \land (\tau_2 \text{ during } x)\)

This part is nearly identical to the In-Clause part in that there are four isomorphic cases.

We show a representative case:

Case 2: \(-((\tau_1 \text{ equals } z) \land (\tau_1 \text{ costsarts } z))\) - left end of the interval

\((\forall x, y.\text{Proposition})\text{isA}(\beta, \epsilon, x, y) \land (y \text{ over systime}) \land (\tau_2 \text{ during } z) \implies (y \text{ coends systime}) \land (\tau_2 \text{ meets } t_1) \land (t_1 \text{ coends } z) \land \text{isA}(\beta, \epsilon, t_1, t_3) \land (t_2 \text{ costsarts systime})\)
With-Clause Part - non-assertional case:
Again, there are four cases isomorphic to the above. We show only the first case.

if FROM $\eta$ is missing then:
If the following holds:
($\exists u, v/Proposition)(\exists h, x, y, z/TimeInterval$

(label($u$) = $\xi$) $\land$ instanceOf($\beta$, from($u$), $x$, $w$) $\land$

prop($v$, $\beta$, $\iota$, $x$, $y$) $\land$

instanceOf($v$, $u$, $z$, $\iota'$) $\land$

($\iota'$ over systime) $\land$ ($\tau_3$ during $z$)

then the following sentence is added to $KB_n$:
($\forall u, v/Proposition)(\forall w, x, y, z/TimeInterval$

(label($u$) = $\xi$) $\land$ instanceOf($\beta$, from($u$), $x$, $w$) $\land$

prop($v$, $\beta$, $\iota$, $x$, $y$) $\land$

instanceOf($v$, $u$, $z$, $\iota'$) $\land$

($\iota'$ over systime) $\land$ ($\tau_3$ during $z$) $\implies$

($\iota'$ coends systime)

else (FROM $\eta$ is not missing)
If the following holds:
($\exists u, v/Proposition)(\exists w, x, y, z/TimeInterval$

(label($u$) = $\xi$) $\land$ instanceOf($\beta$, $\eta$, $z$, $w$) $\land$

prop($v$, $\beta$, $\iota$, $x$, $y$) $\land$

instanceOf($v$, $\eta$, $z$, $\iota'$) $\land$

($\iota'$ over systime) $\land$ ($\tau_3$ during $z$)

then the following sentence is added to $KB_n$:
($\forall u, v/Proposition)(\forall w, x, y, z/TimeInterval$

(label($u$) = $\xi$) $\land$ instanceOf($\beta$, $\eta$, $z$, $w$) $\land$

prop($v$, $\beta$, $\iota$, $x$, $y$) $\land$

instanceOf($v$, $\eta$, $z$, $\iota'$) $\land$

($\iota'$ over systime) $\land$ ($\tau_3$ during $z$) $\implies$

($\iota'$ coends systime)

Note that if an attribute class is $UNTELL$ed then all of its instances will be $UNTELL$ed at the same time in a similar manner.

With-Clause Part - assertional case:
This is equivalent to the non-assertional case without a FROM clause and therefore is omitted.
Note that there is no change to either $DR_n$ nor $IC_n$, only the belief times (which are stored in $KB_n$) are affected.

Special case for $UNTELL$:
If the form of the $UNTELL$ operation is:

\[
\text{UNTELL } \alpha \beta (\theta \tau)
\]

...
then all relationships that $\beta$ participates in are also \textit{UNTELLed} using the methods outlined above.

3.5.3 How \textit{RETEL}L changes the knowledge base

The general form of $O$ for \textit{RETEL}L is as follows:

\begin{verbatim}
RETEL $\beta$
  IN
    $\delta$ (at $\tau_1$)
  BECOMES
    $\delta$ ($\theta_1, \tau_2$)
  ISA
    $\varepsilon$ (at $\tau_3$
  BECOMES
    $\varepsilon$ ($\theta_2, \tau_3$)
  WITH
    $\zeta$ FROM $\eta$
    $\iota : \kappa$ ($\theta_3, \tau_4$)
    BECOMES
      $\kappa_1$ ($\theta_4, \tau_5$)
      $\lambda : \mu$ $\$( $\theta_5, \tau_6$)
    BECOMES
      $\lambda : \mu$ $\$( $\theta_6, \tau_7$)
END
\end{verbatim}

Recall the functional specification of \textit{RETEL}L. (See 3.4.3). Since \textit{RETEL}L is simply the composition of \textit{UNTELL} and \textit{TELL}, we omit presenting the composition of the sets $KB_n$, $IC_n$ and $DR_n$ again. Suffice it to say that the non-BECOMES parts form the $O'$ for \textit{UNTELL} and the BECOMES' parts, the $O''$ for \textit{TELL}.

3.6 Formalizing the Provability Relation in $L$

The only thing left out from the formalization so far is the predicate \textit{prove}. This predicate represents the provability relation in the assertion language so it should be formalized in $L$ accordingly. In fact, now we introduce a predicate \textit{prove} that represents the provability relation in $L$.

One way of doing this is by introducing $L'$, an extension of $L$, which possesses an extra predicate \textit{prove} which can be formalized appropriately. However, now we are not able to talk about the provability relation of $L'$ itself. Since we do not want to introduce a predicate \textit{prove} for every meta level, a better approach to solving our problem is to amalgamate language and meta-language i.e., to be able to formalize the notion of provability for $L$ in $L$ itself. We follow the approach of [BK82] who have done the same thing for a Horn clause language.

At first, we need a way to name arbitrary expressions of $L$ in $L$ itself. To be able to achieve this, every expression of $L$ is mapped to a new constant which is a \textit{standard name} for it. This can be done as follows:

- Every constant $c$ is named by a new constant $c'$.
- Every variable $x$ is named by a new constant $x'$.
• Every term \( f(c_1, \ldots, c_n) \) is named by a new term \( \text{term}(f', c') \) where \( f' \) names \( f \) and \( c' \) names the list of the arguments.

• Every atomic formula \( p(x_1, \ldots, x_n) \) is named by a new term \( \text{atomic}(p', c') \) where \( p' \) names the \( p \) and \( c' \) names the list of arguments.

• Every expression of the form \( \neg \alpha \) is named by the term \( \text{not}(\alpha) \) where \( \alpha \) names \( \alpha \).

• Every expression of the form \( \alpha \implies \beta \) is named by the term \( \text{implies}(\alpha, \beta) \) where \( \alpha \) and \( \beta \) names \( \alpha \) and \( \beta \) respectively.

• Every expression of the form \( (\forall x)\alpha \) is named by the term \( \text{all}(x', \alpha) \) where \( x' \) and \( \alpha \) names \( x \) and \( \alpha \) respectively.

All the above mappings are injective. The assignment of names to terms of \( \mathcal{L} \) must respect the equality theory of the knowledge base. An easy way to do this is to consider the set of terms in the language partitioned into equivalence classes, where two terms belong to the same equivalence class if and only if they are equal. Then you can assign the same name to every member of the class.

In addition to the above, we assume that for every naming of a constant or a variable or a term of \( \mathcal{L} \) an atomic formula \( \text{name}(x, x') \) is entered in the knowledge base.

Now, we are able to give axioms about prove. For example, the following formula says that \( \alpha \implies \alpha \) must be always provable in the theory.

\[
(\forall x)(\forall y)\text{name}(x, y) \implies \text{prove}(\text{implies}(y, y))
\]

From now on, we will freely write expressions like the above using quotations. For example:

\[
(\forall \alpha)\text{prove}(\text{"\( \alpha \implies \alpha \)"})
\]

The reader must keep in mind that the quoting and unquoting device is available whenever it is needed.

The following axioms are introduced for prove. All free variables are assumed to be universally quantified at the beginning of the formulas.

\[
\begin{align*}
(\exists \phi)\text{prove}(\phi, i) & \implies \text{prove}(\phi) \\
\text{axiom}(\phi) & \implies \text{proof}(\phi, i) \\
\text{proof}(\phi, j) \land \text{proof}(\text{pro}(\phi \implies \psi), k) & \land k > j \land i > k \implies \text{proof}(\psi, i) \\
\text{proof}(\phi, j) \land j < i & \implies \text{proof}(\text{\( (\forall x)\phi \)"}, i)
\end{align*}
\]

\[
\begin{align*}
\text{logicalAxiom}(\phi) & \implies \text{axiom}(\phi) \\
\text{nonLogicalAxiom}(\phi) & \implies \text{axiom}(\phi)
\end{align*}
\]

\[
\begin{align*}
\text{logicalAxiom}(\text{\( \phi \implies (\psi \implies \phi) \)}) \\
\text{logicalAxiom}(\text{\( ((\phi \implies (x \implies \psi)) \implies ((\phi \implies (x \implies \xi)) \implies (\phi \implies \psi)) \)}) \\
\text{logicalAxiom}(\text{\( ((\neg \phi \implies \neg \psi) \implies (\neg \phi \implies \psi) \implies \phi) \)}) \\
\text{occursIn}(x, \psi) \land \text{freeFor}(t, x, \psi) \land \text{substitute}(t, \psi, \phi) & \implies \text{logicalAxiom}(\text{\( (\forall x)\phi \implies \psi \)}) \\
\text{notFreeIn}(x, \phi) & \implies \text{logicalAxiom}(\text{\( (\forall x)\phi \implies \psi \}) \implies (\phi \implies (\forall x)\psi))
\end{align*}
\]

65
This formalization of provability follows the notion of proof of [Men79]. We do not give axioms for occursIn, freeFor, substitute, and notFreeIn since they are trivial but rather cumbersome. Also, for every formula \( \phi \) in the knowledge base an axiom

\[
\text{nonLogicalAxiom}(\phi')
\]

is added in the knowledge base where \( \phi' \) is a name for \( \phi \).

It is not difficult to show that the above predicate \( \text{prove correctly represents} \) the provability relation in our knowledge base i.e., whenever \( \phi \) is a sentence of \( \mathcal{L} \) named by a term \( \phi' \) then

\[
KB \vdash \phi \text{ iff } KB \vdash \text{prove}(\phi')
\]

This however does not require that the knowledge base implies \( \neg \text{prove}(\phi') \) whenever \( \phi \) cannot be proved from \( KB \), i.e., the provability predicate is incomplete.

### 3.7 Translation of formulas of the Assertion Language into formulas of \( \mathcal{L} \)

Formulas in the assertion language are easily translated into formulas of \( \mathcal{L} \) by means of a special function \( \sigma \). This function is specified below. We use italics for symbols in \( \mathcal{L} \), the others are symbols in Telos.

Assertion language predicates and functions are translated into \( \mathcal{L} \) as follows:

- \( \sigma(a) = \alpha \) (where \( \alpha \) is any variable or constant)
- \( \sigma(a \land b) = \sigma(a) \land \sigma(b) \)
- \( \sigma(a \lor b) = \sigma(a) \lor \sigma(b) \)
- \( \sigma(a \rightarrow b) = \sigma(a) \rightarrow \sigma(b) \)
- \( \sigma(a \leftrightarrow b) = \sigma(a) \leftrightarrow \sigma(b) \)
- \( \sigma(\neg a) = \neg \sigma(a) \)

\[
\sigma(\text{from}(p)) = \text{from}(\sigma(p)) = \text{from}(p)_{21}
\]

\[
\sigma(\text{label}(p)) = \text{label}(p)
\]

\[
\sigma(\text{to}(p)) = \text{to}(p)
\]

\[
\sigma(\text{when}(p)) = \text{when}(p)
\]

- \( \sigma(\theta) = \theta \) (where \( \theta \) is a string in the form of \( \langle \text{time} - \text{relation} \rangle \))

- \( \sigma(x \theta y) = x \theta y \) (where \( \theta \) is a string in the form of \( \langle \text{time} - \text{relation} \rangle \) and \( x \) and \( y \) are strings in the form of \( \langle \text{time} - \text{interval} \rangle \))

\[
\sigma(\text{prop}(x,y,z,t)) = \text{prop}(x,y,z,t)
\]

\[
\sigma(\text{instanceOf}(x,y,z,t_1,t_2)) = \text{instanceOf}(x,y,z_1,t_2)
\]

\[
\sigma(\text{isA}(x,y,z,t_1,t_2)) = \text{isA}(x,y,z_1,t_2)
\]

\[
\sigma(\text{attribute}(x,y,z,t_1,t_2)) = \text{attribute}(x,y,z_1,t_2)
\]

(Where \( \alpha \) is a string in the form of \( \langle \text{attribute} - \text{label} \rangle \))

\[
\sigma(\text{instanceOf}(x,y,t_1)) = \text{instanceOf}(x,y,t_1)
\]

---

\( ^{21} \) The intermediary step is omitted henceforth.
\[ \sigma(\text{isA}(x, y, t_1)) = \text{isA}(x, y, t_1) \]
\[ \sigma(\text{attribute}(x, y, t_1)) = \text{attribute}(x, y, t_1) \]

(where \( \alpha \) is a string in the form of \(<\text{attribute} - \text{label}>\))

\[ \sigma(x \in y[\theta t_1 - \text{believed} \xi t_2]) = \]
\[ (\exists x, t_4 / \text{TimeInterval}) \text{instanceOf}(x, y, t_3, t_4) \land (t_3 \theta t_1) \land (t_4 \xi t_2) \]

Now we give the translation for some formulas with set expressions:

\[ \sigma(\forall x/S \alpha) = \forall x \sigma(\alpha) \]
\[ \sigma(\exists x/S \alpha) = \exists x \sigma(\alpha) \]
\[ \sigma(\exists x \text{MemberOf} S) = \exists x \sigma(S, x) \]

In addition to the above, there are a number of other cases for the \( \sigma \) function involving set expressions. In particular, for the case of \( \sigma(\alpha) \), where \( \alpha \) is any Telos assertion sentence containing one of the special terms (we will call them \( s\text{terms} \) listed below, the translation is done in terms of \( \sigma' \). In the following \( s_1 \) and \( s_2 \) represent set-oriented expressions and \( \theta \) and \( \xi \) represent one of the time predicates. The \( s\text{term}s \) are:

\{\(x_1, x_2, ..., x_n\)\}
\{Each \( x / s_1 : \alpha \)\}
\{Each \( C[\theta t_1 - \text{believed} \xi t_2] \)\}
\( s_1 \text{Union} s_2 \)
\( s_1 \text{Intersection} s_2 \)
\( s_1 \text{Difference} s_2 \)
\( x.n[\theta t_1 - \text{believed} \xi t_2] \)
\( x.n[\theta t_1 - \text{believed} \xi t_2] \)
\( x.n[\theta t_1 - \text{believed} \xi t_2] \)
\( x.n[\theta t_1 - \text{believed} \xi t_2] \)

We can now define \( \sigma \) for \( s\text{terms} \) as follows:

\[ \sigma(\alpha) = (\forall u \sigma'(\alpha, u) \implies \alpha') \]

where \( u \) is a new variable and \( \alpha' \) is the same as \( \alpha \), but with all occurrences of \( s\text{term}s \) replaced by the variable \( u \).

We define \( \sigma' \) recursively:

\[ \sigma'(s_1, u) = \beta[s] \]

where \( s_1 \) is an \( s\text{term} \) not containing any \( s\text{term} \), \( \beta[s] \) is a particular set of sentences depending on which \( s\text{term} \) \( s_1 \) is, and has a free variable \( u \). The following list gives the set of sentences for each \( s\text{term} \):

\[ \sigma'((x_1, x_2, ..., x_n), u) = \]
\[ (u = x_1 \lor u = x_2 \lor ... \lor u = x_n) \]

\[ \sigma'(x.n[\theta t_1 - \text{believed} \xi t_2], u) = \]
\[ (\exists x, t_4 / \text{TimeInterval}) \]

\(^{22}\text{See below for the definition of } \sigma'.\)
\[ \sigma'(x[n[\theta \ t_1 \ - \ believed \ \xi \ t_2], u) = \\
(\exists u, V, P, C, t_4/\text{Proposition})(\exists t_3, t_4/\text{TimeInterval}) \\
prop(u, x, l, v, t_3) \land prop(P, C, n, V, t_4) \land \\
p \text{ in } P[\theta \ t_1 \ - \ believed \ \xi \ t_2] \]  

\[ \sigma'(x[y[\theta \ t_1 \ - \ believed \ \xi \ t_2], u) = \\
(\exists p, x, y/\text{Proposition})(\exists t_3, t_4/\text{TimeInterval}) \\
prop(p, x, y, u, t_3) \land instanceOf(p, \text{Attribute}, t_3, t_4) \land \\
t_3 \theta t_1 \land (t_4 \xi t_2) \\
\]  

\[ \sigma'(x[n[\theta \ t_1 \ - \ believed \ \xi \ t_2], u) = \\
(\exists p, x, y/\text{Proposition})(\exists t_3, t_4/\text{TimeInterval}) \\
prop(u, x, y, p, t_3) \land instanceOf(y, \text{Attribute}, t_3, t_4) \land \\
t_3 \theta t_1 \land (t_4 \xi t_2) \\
\]  

For more complex terms, i.e., where the components are themselves terms, we have the following:

\[ \sigma'(\langle \text{Each } G / \text{st : } \alpha \rangle, u) = \\
\sigma'(C, u) \land \sigma'(\alpha') \quad \text{(where } \alpha' \text{ is } \alpha, \text{ with } u \text{ substituted for all occurrences of } \alpha)\]

\[ \sigma'(\langle \text{Each } G[\theta \ t_1 \ - \ believed \ \xi \ t_2]\rangle, u) = \\
u \in C[\theta \ t_1 \ - \ believed \ \xi \ t_2] \\
\sigma'(\text{Union } st_1, u) = \\
\sigma'(st_1, u) \lor \sigma'(st_2, u) \\
\sigma'(\text{Intersection } st_1, u) = \\
\sigma'(st_1, u) \land \sigma'(st_2, u) \\
\sigma'(\text{Difference } st_1, u) = \\
-\sigma'(st_1, u) \land \sigma'(st_2, u) \\
\sigma'(st_1[\theta \ st_2 \ - \ believed \ \xi \ st_4], u) = \\
(\exists t_1, t_2, u_2, u_3, u_4)/\text{Proposition} \\
\sigma'(st_1, u_1) \land \sigma'(st_2, u_2) \land \\
\sigma'(st_3, u_3) \land \sigma'(st_4, u_4) \land \\
(\exists t_1, t_2/\text{TimeInterval}) \\
\text{attribute}(u_2, u_1, u_1, t_3, t_2) \land (t_1 \theta u_2) \land (t_2 \xi u_4) \\
\sigma'(st_1[\theta \ st_2 \ - \ believed \ \xi \ st_4], u) = \\
(\exists t_1, t_2, u_2, u_3, u_4)/\text{Proposition} \\
\sigma'(st_1, u_1) \land \sigma'(st_2, u_2) \land \\
\]  

\[ \text{The predicate } is \text{ is defined as follows:} \\
(x \text{ in } y[\theta \ t_1 \ - \ believed \ \xi \ t_2]) \iff \\
(\exists t_2, t_4/\text{TimeInterval}) \text{instanceOf}(x, y, t_3, t_4) \land (t_3 \theta t_1) \land (t_4 \xi t_2) \
\]  

68
\[\sigma'(st_3, u_3) \land \sigma'(st_4, u_4) \land \\
(\exists v, w, P, C/\text{Proposition})(\exists t_1, t_2/\text{TimeInterval}) \\
\text{prop}(u, u_1, u_3, v, t_1) \land \\
\text{prop}(P, C, w, V, t_2) \land \\
u \text{ in } P[6 u_3 = \text{believed} \xi u_4] \]

\[
\sigma'(st_1 \cdot st_2[\delta st_3 - \text{believed} \xi st_4], u) = \\
(\exists u_1, u_2, u_3, u_4)/\text{Proposition} \\
\sigma'(st_1, u_1) \land \sigma'(st_2, u_2) \land \\
\sigma'(st_3, u_3) \land \sigma'(st_4, u_4) \land \\
(\exists p/\text{Proposition})(\exists t_1, t_2/\text{TimeInterval}) \\
\text{prop}(p, u_1, u_2, u, t_1) \land \text{instanceOf}(u, \text{Attribute}, t_1, t_2) \land \\
(t_1 \theta u_3) \land (t_2 \theta u_4) \\
\]

\[
\sigma'(st_1 \cdot st_2[\delta st_3 - \text{believed} \xi st_4], u) = \\
(\exists u_1, u_2, u_3, u_4)/\text{Proposition} \\
\sigma'(st_1, u_1) \land \sigma'(st_2, u_2) \land \\
\sigma'(st_3, u_3) \land \sigma'(st_4, u_4) \land \\
(\exists p/\text{Proposition})(\exists t_1, t_2/\text{TimeInterval}) \\
\text{prop}(u, u_1, u_2, p, t_3) \land \text{instanceOf}(u, \text{Attribute}, t_1, t_2) \land \\
(t_1 \theta u_3) \land (t_2 \theta u_4) \\
\]

This fully defines \(\sigma'\) for all terms; now the translation function is fully specified.
Chapter 4

Conclusions and Directions for Further Research

We consider the treatment of attributes and the representation of temporal knowledge as key contributions of this work to the state of the art for knowledge representation languages. Treating attributes as first class "unis" in the language, on a par with individuals, leads to several interesting consequences: (a) the (weak!) typing mechanism offered by instantiation applies equally to attributes and individuals, (b) it is possible to represent relationships on relationships through attributes of attributes, without having to worry ahead of time whether this need will arise, (c) built-in features and assumptions of other knowledge representation languages, such as facets [FK85] and property categories [Gre84] can all be expressed in Telos through attribute metaclasses, thus providing a powerful extension facility. In PSN [LM79] attributes are treated as instantiated objects of classes but are not allowed to have their own attributes. [Mai84] also offers to treat concepts (our individuals) and slots (our attributes) uniformly, but proposes a solution from a narrower viewpoint that does not make use of an instantiation dimension nor does it try to address issues concerning general query evaluation and integrity checking mechanisms.

Selection of a suitable model for temporal knowledge, integration of the model into an object-oriented framework, development of suitable notations and design of a tractable inference procedure for temporal knowledge were key issues that had to be addressed in formulating the temporal component of Telos. Our approach in tackling these issues is an eclectic synthesis, based on recent work on time [AH85] [VK86] but also temporal databases [Sno87]. Of course, the nature of this component of Telos was complicated immensely by all the other features that need to coexist within one linguistic framework: from a notational viewpoint (making sure that time does not get in the way as the user ASKs and TELLS), a semantic viewpoint (deciding how much default reasoning do we want to allow, the meaning of belief time intervals), a structural viewpoint (essentially offering time as a fourth dimension of knowledge organization in Telos) and finally, from an implementation viewpoint. To our knowledge, no other knowledge representation language provides facilities for time that are as tightly integrated into the overall representational framework.

The assertional component of the language has always been a source of problems and much interesting research remains to be done there. The first problem is expressiveness of the assertional language vs. tractability. We currently offer deductive rules that are equivalent in expressive power to deductive database clauses considered in [LT85]. These are more general than Horn clauses thus harder to deal with. Lloyd and Topor give a sound method for translating these clauses into a set of Horn clauses possibly with negation in their body [LT84] [LT85]. This clausal
form seems to be more appropriate when issues like safety of queries, elimination of recursion, and query optimization have to be addressed. Thus it might turn out to be useful to restrict the assertion language to clauses of this form.

The second problem with the assertional component is that assertions are expected to be attached to objects they reference but this is not enforced by the constructs of the language but it is left to the good intention of the user. Therefore with the current form of the assertional component it should be hard to find methods to explore the topology of the network for doing the form of resource-limited reasoning suggested in [Jar].

Another interesting issue is the ability of the assertional language to express statements that relate the contents of the knowledge base to entities in the application domain. A preliminary step towards this goal has been the definition of the Universe of Discourse Language in [Sta86]. However, this approach has not been pursued further so far. A different and intuitively more appealing approach is to introduce Levesque's knowledge operator $K$ in the assertional language thus enabling statements about the knowledge base as well as about the application domain. This will also give us a chance to explore the auto-epistemic nature of integrity constraints [Rei88] for the case of Telos. These improvements in the assertional language must be attempted with the provision that a lot of expressiveness will be gained but also tractability will not be sacrificed altogether.
Appendix A

Syntax

We use a slightly modified BNF notation to show the syntax of the subset of Telos we are currently implementing. The following notational conventions are used:

1. *Terminal*: symbols in boldface, single characters are also quoted (") for clarity.
2. *Non-terminal*: symbols consisting of a string starting with a "<" and ending with a ">
3. [...] means 0 or 1 occurrences of "..."
4. { ... } means 0 or more occurrences of "...
5. The terminal *identifier* stands for every sequence of letters, numbers, or underscores with the first character in the sequence being a letter.

A.1 The Syntax of Telos

<program> ::= <command> | <command> 

<command> ::= 

<tell> | <untell> | <retell> 
<retrieve> | <ask> | <range>

<tell> ::= 

TELL < token -- or -- class > identifier [<temporal -- constraint >] 
[ <components -- clause > ] 
[ <in -- clause > ] 
[ <isA -- clause > ] 
[ <with -- clause > ]
END

<untell> ::= 

UNTELL < token -- or -- class > identifier [<temporal -- constraint >] 
[ <in -- clause > ]
[ <isA -- clause > ]
[<with-clause>]
END

<retell> ::= 
  RETELL <token-or-class> identifier  
  [<retell-in-clause>]
  [<retell-isA-clause>]
  [<retell-with-clause>]
END

<retrieve> ::= 
  RETRIEVE <query>

<ask> ::= 
  ASK <query>

<token-or-class> ::= 
  TOKEN | CLASS

<components-clause> ::= 
  COMPONENTS '{' <proposition> ',' <proposition>  
  '}'<proposition> '}'

<in-clause> ::= 
  IN <class> [ <temporal-constraint> ]
  [ '{' <class> [ <temporal-constraint> ] ]

<isA-clause> ::= 
  ISA <class> [ <temporal-constraint> ]
  [ '{' <class> [ <temporal-constraint> ] ]

<with-clause> ::= 
  WITH <attribute-declaration> 
  [ <attribute-declaration> ]

<retell-in-clause> ::= 
  IN <class> [ <temporal-constraint> ]
  BECOMES <class> [ <temporal-constraint> ]
  [ '{' <class> [ <temporal-constraint> ] ]
  BECOMES <class> [ <temporal-constraint> ]

<retell-isA-clause> ::= 
  ISA <class> [ <temporal-constraint> ]
  BECOMES <class> [ <temporal-constraint> ]
  [ '{' <class> [ <temporal-constraint> ] ]
  BECOMES <class> [ <temporal-constraint> ]

<retell-with-clause> ::= 
  WITH <retell-attribute-declaration> 
  [ <retell-attribute-declaration> ]
<query> ::=  
<variables> : ' <logical-expression>

<variables> ::=  
  identifier '/' <set-expression>  
  { ',' <variable> '/' <set-expression> } *  

<temporal-constraint> ::=  
  ' ( <time-relation> <time-constant> )'  

<time-constant> ::=  
  Now | Alltime | <conventional-interval>

<conventional-interval> ::=  
  <unit-interval> | <unit-interval>.. <unit-interval> |  
  <unit-interval>.. | .. <unit-interval>

/unit-interval> ::=  
  <year>['/' <month>['/' <day>  
  [ '-' <hours>[':' <minutes>[':' <seconds>]]]]]

/year> ::=  
  integer

/month> ::=  
  integer

/day> ::=  
  integer

/hours> ::=  
  integer

/minutes> ::=  
  integer

/seconds> ::=  
  integer

/attribute-declaration> ::=  
  <attribute-category> {' , ' <attribute-category> }  
  <attribute> {' , ' <attribute> }

/retell-attribute-declaration> ::=  
  <attribute-category> <retell-attribute>  
  {' , ' retell-attribute }

/attribute-category> ::=  
  identifier [ <from-clause> ]  
  <built-in-attribute-class> [ <from-clause> ]

/attribute> ::=  
  [ <attribute-label> : ' ] <attribute-value> [ <temporal-constraint> ]
  ' : ' attribute-value [ <temporal-constraint> ]
\[ <\text{tell}} \text{- attribute} > \text{ ::=} \\
\quad <\text{attribute} \text{- label}> [':\ <\text{attribute} \text{- value}>][ <\text{temporal} \text{- constraint}>] \\
\quad \text{BECOMES} <\text{attribute} \text{- value}>[<\text{temporal} \text{- constraints}>] \\
\quad <\text{attribute} \text{- label} > \text{ ::=} \text{ identifier} \\
\quad <\text{attribute} \text{- value} > \text{ ::=} \\
\quad \quad <\text{proposition}> | <\text{assertion}> \\
\quad <\text{from} \text{- clause} > \text{ ::=} \\
\quad \quad \text{FROM} \text{ identifier} \\
\quad <\text{integer} \text{- subrange} > \text{ ::=} <\text{lower} \text{- bound}>..<\text{upper} \text{- bound}> \\
\quad <\text{lower} \text{- bound} > \text{ ::=} \text{ integer} \\
\quad <\text{upper} \text{- bound} > \text{ ::=} \text{ integer} \\
\quad <\text{function} > \text{ ::=} \\
\quad \quad \text{from} '(' <\text{function}> ')' | \\
\quad \quad \text{label} '(' <\text{function}> ')' | \\
\quad \quad \text{to} '(' <\text{function}> ')' | \\
\quad \quad \text{when} '(' <\text{function}> ')' | \\
\quad \quad <\text{proposition}> \\
\quad <\text{class} > \text{ ::=} \\
\quad \quad \text{identifier} | <\text{built} \text{- in} \text{- class}> \\
\quad <\text{proposition} > \text{ ::=} \\
\quad \quad \text{identifier} | <\text{built} \text{- in} \text{- class}> | \\
\quad \quad <\text{built} \text{- in} \text{- attribute} \text{- class}> | <\text{time} \text{- constants}> | \\
\quad \quad \text{integer} | \text{string} | <\text{integer} \text{- subrange}> \\
\quad <\text{built} \text{- in} \text{- class}> \text{ ::=} \\
\quad \quad \text{Proposition} | \text{Class} | \text{Individual} | \\
\quad \quad \text{IndividualClass} | \text{OmegaClass} | \text{Integer} | \\
\quad \quad \text{String} | \text{TimeInterval} | \text{Token} | \\
\quad \quad \text{S\_Class} | \text{M1\_Class} | \text{M2\_Class} ... \\
\quad <\text{built} \text{- in} \text{- attribute} \text{- class}> \text{ ::=} \\
\quad \quad \text{Attribute} | \text{AttributeClass} \\
\quad <\text{time} \text{- relation} > \text{ ::=} \\
\quad \quad \text{equals} | \text{at} | \text{during} | \text{over} | \text{startsbefore} | \text{startsafter} | \text{endsbefore} | \\
\quad \quad \text{endsafter} | \text{rightbefore} | \text{rightafter} | \text{before} | \text{after} | \text{costarts} | \\
\quad \quad \text{coends} | \text{overlaps} | \text{overlapped} \text{- by} | \text{meets} \]
A.2 Assertions

<assertion> ::= 'S' <logical-expression> 'S'

<logical-expression> ::= 
    { '(' <quantifier> <variable> '/' <set-expression> [ <time-constraints> | ']' ) 
    <logical-expression> | 
    not <logical-expression> | 
    <atomic-expression> <logical-connective> <atomic-expression>

<time-constraints> ::= 
    '[' <time> ':' believed <time> ']' 

<quantifier> ::= Forall | Exists 

<variable> ::= <identifier> 

<set-expression> ::= 
    <set-expression> <set-operator> <set-expression> | 
    '(' <set-expression> ')')) | 
    <set-enumeration> | <set-construction> | 
    <set-valued-function> | <class-extension>

<set-operator> ::= 
    Union | Intersection | Difference 

<set-valued-function> ::= 
    <simple-set-valued-function> <selector> identifier | 
    <simple-set-valued-function> <selector> <built-in-attribute-class>

<simple-set-valued-function> ::= 
    <simple-set-valued-function> <selector> identifier | 
    <simple-set-valued-function> <selector> <built-in-attribute-class>
    | 
    <function>

<set-enumeration> ::= 
    '{' '[ <proposition> ] ' } | 
    '{ ' [ <proposition> ] ' ' 

<set-elements> ::= 
    <proposition> { ',' <proposition> } | ε

<set-construction> ::= 
    '{ ' ' EACH identifier ':' <assertion> ' } ' 

<class-extension> ::= '{ ' ' EACH <class> <time-constraints> ' } ' 

<logical-connective> ::= 
    and | or | '<=' | '=>' 

78
<atomic-expression>::=
  <prefix-predicate>|<infix-predicate>

<prefix-predicate>::=
  <built-in-prefix-predicate>(' <arguments>')|<identifier>(' <arguments>')

<built-in-prefix-predicate>::=instanceOf|isa|prop

<arguments>::=<argument>{','<argument>}
<argument>::=integer|string|<proposition>

<infix-predicate>::=
  <argument>in<argument><time-constraint>|<argument><time-relation><argument>|<proposition>MemberOf<set-expression>|<expression><arithmetic-predicate><expression>

<expression>::=
  <expression><arithmetic-operator><expression>|(' <expression>')|'.<expression>|<proposition>

<arithmetic-operator>::=
  '+'|'-'.|'*'.|'/'}
Appendix B

The Full Form of a Formula in the Assertion Language

In this appendix, we define the full form of any formula in the assertion language with respect to a history time component and a belief time component.

B.1 Integrity Constraints and Deductive Rules

The full form of an integrity constraint or a deductive rule $\xi$ with respect to history time $T_1$ and belief time $T_2$ is defined to be the formula:

$$(\forall t_1, t_2/\text{TimeInterval})(t_1 \text{ during } T_1 \text{ and } t_2 \text{ during } T_2 \implies \xi')$$

where $\xi'$ is a formula which can be derived from $\xi$ by applying the following transformations repeatedly:

1. Substitute any universally quantified expression $(\forall x/S_1)a$ with:
   $$(\forall x/\text{Proposition}) (x \text{ MemberOf } S_1 \implies a)$$

2. Substitute any existentially quantified expression $(\exists x/S_1)a$ with:
   $$(\exists x/\text{Proposition}) (x \text{ MemberOf } S_1 \text{ and } a)$$

3. When none of the above steps can be applied any more, substitute any missing history or belief time components in any set expression with at $t_1$ and at $t_2$ respectively.

B.2 Arguments of the prove Predicate

The full form of an argument $\delta$ of the predicate prove with respect to a temporal component

$$(\delta \text{ T}_1 = \text{ believed } \xi \text{ T}_2)$$

is defined to be a new assertion $\xi'$ which can be derived by applying the following transformations to $\xi$.
1. Substitute any universally quantified expression \((\text{Forall } x/S1)a\) with:
\[(\text{Forall } x/\text{Proposition}) (\ x \text{ MemberOf } S1 \implies a)\]
2. Substitute any existentially quantified expression \((\text{Exists } x/S1)a\) with:
\[(\text{Exists } x/\text{Proposition}) (\ x \text{ MemberOf } S1 \text{ and } a)\]
3. When none of the above steps can further be applied, substitute any missing history or belief time components in any set expression with \(\theta t1\) and \(\theta t2\) respectively.
Bibliography


