Evolving Logical Specification
in Information Systems

Stefan Conrad¹, Jaime Ramos², Gunter Saake¹, Cristina Sernadas²

1 Otto-von-Guericke-Universität Magdeburg
   Institut für Technische Informationssysteme
   Postfach 4120, D-39016 Magdeburg
   Germany
   {conrad|saake}@iti.cs.uni-magdeburg.de

2 Instituto Superior Técnico
   Departamento de Matemática
   Av. Rovisco Pais, P-1096 Lisboa Codex
   Portugal
   {jabr|css}@math.ist.utl.pt

April 1997
Abstract

Traditional logic-based specification approaches fix the structure and the dynamics of an object system at specification time. Information systems are applications with a very long life-time. Therefore, object and specification evolution to react on changing requirements is a relevant aspect of describing information systems as object societies.

We present a logical specification framework for evolving objects based on the concepts of object developed for the languages Troll and Gnome and the underlying temporal logic OSL. The syntactic notion of object descriptions is extended to explicitly manipulate temporal axioms during behaviour evolution. An extension of OSL called dyOSL establishes a logical framework where basic temporal formulae are evaluated by a second logical layer. dyOSL explicitly manipulates state-dependent sets of temporal formulae to model evolution of object axioms.

Syntax and semantics of dyOSL are given together with a translation of the specification language constructs into dyOSL. The corresponding proof theory is discussed together with guidelines for formally certifying object properties. Additionally, we show the use of the formalism for some specific information system scenarios.
Acknowledgments

We are grateful to Amilcar Sernadas for discussing with us this difficult problem. Work presented here was partially supported by PRAXIS XXI Project 2/2.1/MAT/262/94 SitCalc, ESPRIT III Working Group 8319 ModelAge, ESPRIT IV Working Group 22704 ASPIRE, and ESPRIT IV Working Group 23531 FIREworks.
Contents

1 Introduction 1

2 Motivation and Language 5

3 Syntax and Semantics of the Logic 13
   3.1 Signatures .................................................. 13
   3.2 Terms and Formulae ........................................ 14
   3.3 Pre-interpretation structures .............................. 16
   3.4 Satisfaction ................................................ 17
   3.5 Specifications and Theories ............................... 20

4 Translation of Language into Logic 21

5 Using the Logical Framework 25
   5.1 A Hilbert calculus .......................................... 25
   5.2 An invariant calculus ..................................... 27

6 Concluding Remarks 31
Chapter 1

Introduction

In the last years, logic-based approaches for specifying information systems were
developed and investigated. Especially, in combination with object-oriented principles for
the design of information systems ([Sernadas et al., 1991b]) a lot of work was done to
find adequate logic-based approaches to describe the dynamic behaviour as well as the
interaction of objects in information systems.

Mainly for describing the dynamic behaviour of objects in information systems, tem-
poral logic ([Manna and Pnueli, 1992]) is a good and widely accepted choice. This is
due to the fact that information systems (as a generalization of database systems) are
state-based systems for which the state-based approach of temporal logic is obviously
appropriate (cf. [Khosla and Maibaum, 1989]).

Therefore, it is no wonder that the (temporal) specification of informa-
tion systems became a popular research area, especially because it is based
on a clean and well-understood theory (cf. e.g. [Fiadeiro and Maibaum, 1991,
Ehrich et al., 1993, Conrad, 1996, Sernadas et al., 1996]). Furthermore, several spec-
ification languages like ALBERT ([Dubois et al., 1993]), LCM ([Wieringa, 1991,
Feenstra and Wieringa, 1993]), OBLOG ([Sernadas et al., 1987, Sernadas et al., 1991a]),
Gnome ([Sernadas and Ramos, 1994]), and TROLL ([Jungclaus et al., 1996]) have been
developed for supporting the specification of objects and their dynamic behaviour based
on temporal (or dynamic) logics.

Of course there are other formal approaches to specifying dynamic behaviour of
objects. For instance, conditional term rewriting ([Meseguer, 1993]) is another way
of describing object behaviour. The specification language Maude ([Meseguer, 1992,
Meseguer and Qian, 1993]) is based on such a conditional term rewriting semantics. In the
database area a lot of interesting work concerning the description of database dynamics
(e.g. in [Lipeck and Thalheim, 1993]) and temporal databases (cf. [Tansel et al., 1993])
can be found.

In contrast to other approach (like the evolving algebra approach by [Gurevich, 1991,
Gurevich, 1993]) in our context the notion of evolution refers to the evolution of specifi-
cations during the runtime of a system.

However, all these approaches to specifying information systems assume that the dynamic behaviour of the system and its parts is totally known at specification time. In addition, once specified the behaviour is fixed for the entire life (or run) time of the information system. This restriction seems not to be adequate for a large number of typical information system applications. Because of the long life- (or run-) time of information systems changes of the dynamic behaviour are often required within the lifetime of a system.

Typical examples for such changes of the dynamic behaviour can already be found in usual application areas for information systems like libraries and banks. For a library it may happen that the rules for borrowing and returning books change in order to avoid too much administrative work in writing reminders. Due to changes of laws (or bank rules) it could become necessary to change the computation of yearly interests or to restrict the evolution of an account in some way. These changes are modifications of axioms describing the dynamic behaviour of the system. In consequence, we have to find a framework for describing evolving specifications where we can specify the evolution of first-order dynamic behaviour specifications.

In this paper we continue work started in [Saake et al., 1995b] where a first, very restricted form of evolving temporal specification was proposed. In [Conrad and Saake, 1995] and [Conrad and Saake, 1996] a first ad hoc formalization of a temporal logic allowing evolving behaviour was presented and discussed. Different ways of handling the evolution of specifications within a specification language were investigated in [Türker et al., 1996]. After having consolidating this preparatory work we are now able to present a proposal for an adequate specification language together with a formal definition of an extended temporal logic for specifying objects and their behaviour which may be dynamically modified.

For the logical part we build on the Object Specification Logic OSL proposed in [Sernadas et al., 1995]. For our purposes we need to distinguish two logical levels: the first level allows the usual temporal specification of the objects’ behaviour by means of OSL, and the second level allows us to deal with behaviour evolution. By this separation into two levels we are able to define a clean extension of OSL. For the purposes of this paper, we focus the presentation on the treatment of behaviour evolution and ignore object-oriented concepts like inheritance. Incorporating those concepts into our logic can be done in the same way as it has been done for the original OSL in [Sernadas et al., 1995].

The remainder of this paper is organized as follows: In Chapter 2 we introduce and discuss an example specification for documents in an information system managing offers and contracts in a sales department. By means of this example we motivate the need for evolving behaviour specification and we additionally introduce basic parts of a corresponding object specification language. In Chapter 3 we define syntax and semantics of the logic, called dynamic OSL (dyOSL). Afterwards, we show how to translate the interesting
parts of specifications, namely the evolving behaviour part, into dyOSL (Chapter 4). The usage of our logical framework in proving properties of objects from their specification is briefly dealt with in Chapter 5. Finally, we conclude by summarizing our approach presented in this paper and by pointing out open questions for future work.
Chapter 2

Motivation and Language

In this chapter, we discuss the limitations of current object-oriented specification approaches using temporal logic for describing the dynamic behaviour of objects. In order to overcome these limitations we propose an extension allowing evolution of the dynamic behaviour of objects. As starting point we use TROLL which is one of the object specification languages developed so far. By means of examples we motivate the extensions of the logic that are needed.

For presenting the current status of object-oriented specification we introduce a simple example.

Example 2.1 In an information system for the sales department of an enterprise all offers made to customers and all contracts with customers are to be managed. In Figure 2.1 a specification of an object class for documents (as a unifying notion for offers and contracts) is given using a TROLL-like specification language.

Here, documents have four attributes. The document number ‘DocNo’ is unique for each document. In the identification part the attribute ‘DocNo’ is specified to be used for identifying document objects. Therefore, there must not be duplicate document numbers in our information system. The document type ‘DocType’ states whether a document is currently an offer made to a customer, a preliminary contract which has to be confirmed by the customer, or already an official (legal) contract. The attribute ‘Valid’ provides a date until an offer or a preliminary contract is valid. The document text is stored as a value of the attribute ‘Content’.

There are five kinds of events (actions) which may occur for document objects. The occurrence of a ‘create’ event denotes the birth of a document object, i.e. a new document is created. During the existence of the object ‘revise’, ‘prepare_contract’, and ‘fix_contract’ events may occur. A document is deleted when a ‘resolve’ event occurs for the corresponding object. The effect of each of these events is described in the rigid axioms part. Therefore, the occurrence of a ‘create’ event results in the specified values for the attributes. Moreover, the event ‘addDocToOffers’ is called in another object which has the name ‘DocManager’ (assuming that this object is specified elsewhere).
**object class** Documents
  **identification** DocId: (DocNo)
  **template**
  **attributes**
  DocNo: int,
  DocType: \{offer, pre_contract, contract\},
  Valid: date,
  Content: text;
  **actions**
  **birth** create(DocNo:int,Content:text),
  revise(NewContent:text),
  prepare_contract,
  fix_contract,
  **death** resolve;
  **rigid axioms**
  create(D,C)
  **changing** DocType = offer, DocNo = D, Content = C,
  Valid = now + 30
  **calling** DocManager.addDocToOffers(self);
  revise(C)
  **enabled** DocType = offer and Valid ≥ now,
  **changing** Valid = now + 30,
  Content = C;
  prepare_contract
  **enabled** DocType = offer and Valid ≥ now,
  **changing** DocType = pre_contract,
  Valid = now + 10,
  Content = C;
  fix_contract
  **enabled** DocType ≠ contract and Valid ≥ now,
  **changing** DocType = contract,
  **calling** DocManager.mvOfferToContracts(self);
  resolve
  **calling**
  [DocType≠contract] DocManager.mvOfferToArchive(self),
  [DocType=contract] DocManager.mvContractToArchive(self);
  **end object class**

Figure 2.1: A specification of documents.
In addition to their effect on attribute values and the calling of other events we impose enabling conditions for ‘revise’, ‘prepare_contract’, and ‘fix_contract’ events. Thereby, such events are only allowed to occur if the corresponding enabling condition is satisfied. The calling of other events can also be combined with additional pre-conditions. For instance, the occurrence of a ‘resolve’ event may cause the occurrence of two different events in the object called ‘DocManager’. Which of these two events is called depends on the value of the attribute ‘DocType’.

The possible evolution of a document is specified such that, when it is created, it is an offer being valid for 30 days. Offers may be revised during their validity. After each revision the offer is again valid for 30 days. In case the customer is satisfied by the offer, the document can be changed to a contract. There are two ways for an offer to become a contract. The direct way is fixing the offer and turning it immediately into a contract. For that, the ‘fix_contract’ event can be used.

Sometimes, it might be useful to have an additional phase for preparing the official contract. By means of the ‘prepare_contract’ event the offer can be turned into a pre-contract. A pre-contract is only valid for 10 days. Within that period the contract must be fixed finally. Otherwise, the pre-contract becomes invalid.

Considering the specification it becomes obvious that problems might occur in the future during the runtime of the system. The complete behaviour is fixed at specification time, i.e. the specification can only respect the possible dynamic behaviour of document objects in the information system as far as it can be foreseen at that moment. Of course, if we already expect that certain changes of requirements will occur later, we are able to specify that in the usual object specification framework.

However, if we look at the evolution of information systems in practice, we have to face the fact that there often are changes of requirements which could not be foreseen at specification time. Such changes may be due to the enacting of new laws or due to the adaptation of business rule to a unforeseen evolution of the market.

In usual specification approaches such changes are not allowed. In case the specification of an information system has to be modified the implemented system which is already running for some time does not fulfill the modified specification anymore. In consequence, the implementation must be modified according to the changes in the specification. However, considering the complete lifetime of an information system there is an obvious problem. For each instant of time the current implementation fits to the current specification.

Unfortunately, neither the original nor the modified specification describes the complete evolution of the objects in the information system. This is due to the fact that the modification of the dynamic behaviour was not described in the original specification and that the modified specification does usually not include the evolution of the system before the specification was modified.
In order to overcome this dilemma which is due to the restrictions of usual specification approaches we are developing a novel specification framework with more flexibility. The main idea of our approach is that we distinguish between rigid axioms and non-rigid axioms. Rigid axioms are axioms which cannot be changed at all. In contrast non-rigid axioms may be added, removed, or changed during runtime. For that additional actions must be introduced into the specification. We call this special kind of actions “mutators” or “mutation actions” (“mutation events” for concrete instances) because their occurrence causes a mutation of the object specification. In order to be able to introduce new axioms by occurrences of such mutation actions, a special type for axioms (spec) is provided. This type must not be used within rigid axioms. Only mutation events may have axioms as parameter.

For managing the set of currently valid non-rigid axioms we have to introduce a special kind of attribute called ‘axiom attribute’. In any case one axiom attribute is needed in which the set of currently valid non-rigid axioms is stored. In the following examples we call this axiom attribute ‘Axioms’. In fact, it might be useful for certain applications to have several such axioms attributes, e.g. for being able to switch between different sets of behaviour axioms (cf. [Türker et al., 1996]). In that case one of these axiom attributes must be marked as the one containing the currently valid axioms. For simplifying the presentation we only use one axiom attribute in the examples of this paper.

Example 2.2 Continuing our example, we now add a simple mechanism for allowing evolving behaviour. Due to the fact that the rigid part of the behaviour specification has already been given in Figure 2.1, we only present the extension of the specification in Figure 2.2.

The extension given here is a rather unrestricted one. It allows to add arbitrary behaviour axioms (and remove those axioms later on). For doing so, special actions, called mutators, are introduced. By means of an occurrence of a ‘add_new_rule’ mutation action we are able to add an arbitrary formula as a behaviour axiom for document objects. For instance, this formula may describe additional effects of an action or impose an additional enabling condition for some action. □

From a logical point of view, adding a new axiom can be considered as restricting the further behaviour of the object. Thereby, it is not possible to change that part of the behaviour which is specified in the rigid axioms section. It is only possible to specify additional effects of actions and to introduce restrictions e.g. with regard to enabling of actions. In this way it is also possible to disallow certain actions to occur at all.

Example 2.3 In the following we present examples for non-rigid axioms which could be added during runtime of our information system for managing documents. The syntax for representing those axioms is close to that we use for object specifications.

1. Due to a change of law it might become necessary to store the date when a contract was fixed. In order to bring this new requirement into our specification an occurrence of a mutation event ‘add_new_rule’ with the following parameter can be used:
object class Documents
   identification  DocId: (DocNo)
   template
      attributes
         ...
      actions
         ...
      rigid axioms
         ...
      axiom attributes
         Axioms initialized { }
   mutators
      add_new_rule(Rule:spec)
      remove_rule(Rule:spec)
   dynamic specification
      add_new_rule(Rule)
      changing  Axioms = Axioms + { Rule }
      remove_rule(Rule)
      changing  Axioms = Axioms - { Rule }
end object class

Figure 2.2: Extending the specification for documents.

fix_contract
   changing  Valid = now

Thereby, ‘fix_contract’ events will have an additional effect. They will set the attribute
‘Valid’ to the current date. Thereby, the date of fixing the contract can later be accessed.

2. Another kind of additional effects of an event can also be specified in the same way as
it would be done for rigid axioms. For instance, let us assume that there is a special
supervisor object called ‘DocManager’ which is responsible for keeping track of the
evolution of documents. This supervisor object may cause the occurrence of a mutation
event ‘add_new_rule’ with the following parameter:
   revise(C)
      calling  DocManager.inform_me(DocNo, C)

From that moment each occurrence of a ‘revise’ event causes an occurrence of an
‘inform_me’ event in the supervisor object. This is new effect of ‘revise’ events—in
addition to the effects described as rigid axioms.

3. In the same way we can restrict the enabling of events. For instance, due to a change
of laws it might become necessary to keep document objects alive forever in case they
have reached to state of an official contract. In order to adapt the system to this new requirement we could add to following axiom which allows the occurrence of 'resolve' events only in case the document is not already a contract:

\[
\text{resolve}
\]

\[
\text{enabled} \quad \text{DocType} \neq \text{contract}
\]

4. By imposing a pre-condition which cannot be fulfilled we can even disallow events to occur at all. For instance, we can forbid ‘prepare_contract’ events by adding the following axiom:

\[
\text{prepare_contract}
\]

\[
\text{enabled} \quad \text{false}
\]

Please note that we now have two enabling conditions which have to be fulfilled in case a ‘prepare_contract’ is to occur. The enabling condition ‘DocType = offer’ which is specified as a rigid axiom is still valid. In addition, the new enabling condition ‘false’ must be respected.

5. Of course, we may also add other kinds of axioms. For example, the following axiom is an integrity constraint restricting the allowed states of an object:

\[
\text{Valid} > \text{now} \rightarrow
\]

\[
(\text{DocType} = \text{offer} \; \text{or} \; \text{DocType} = \text{pre_contract})
\]

This axiom requires documents to be an offer or a pre-contract in case the validity date is set to some date in the future. □

Due to the fact that everything which is specified in the rigid axioms section cannot be changed during runtime of the information system, we have to be very careful in specifying our system. In case we expect (or it could be possible) that the effect an event has on a certain attribute might change we should not specify the effect of this event in the rigid axioms section. Instead we can specify the default effect as part of the initial value of the axiom attribute ‘Axioms’. Then, it is possible to change it later.

Of course, we should avoid to specify objects in a way that everything can be changed later. In that case, it will be rather difficult to prove interesting properties for those objects. Therefore, specifying objects requires a thorough design concerning the question which properties are rigid and which properties may be affected by evolution.

For being able to specify the evolution of dynamic behaviour as adequate as possible we can also impose enabling conditions to mutation events. Thereby, changes of the set of currently valid non-rigid behaviour axioms (the contents of the axiom attribute Axioms) can be better controlled. In certain cases it might even be useful to have integrity constraints restricting the contents of the axiom attribute to fulfill certain properties. The following example presents the restriction of a mutation event by imposing an enabling condition.
Example 2.4 The enabling of mutation actions can be restricted in the same way as it is possible for other events:

```plaintext
dynamic specification
  add_new_rule(Rule)
    enabled   DocType ≠ contract
    changing  Axioms = Axioms + { Rule }
```

Here, we require that the document type of a document must be different from 'contract' in order to allow occurrences of 'add_new_rule' mutation events.

Considering the evolution of information systems in practice clearly shows that there is a real need for allowing behaviour evolution. Although there are several possibilities for extending existing specification language in an ad hoc manner, the question how evolving behaviour can adequately be reflected on the semantical level has still to be answered. In the next section we define syntax and semantics of a logic for that purpose.
Chapter 3

Syntax and Semantics of the Logic

In this chapter, we will present syntax and semantics of the logic dyOSL. dyOSL extends the logic OSL presented in [Sernadas et al., 1995] with a second temporal level, the meta level, allowing evolving axiom attributes.

3.1 Signatures

We assume as given once and for all the underlying data signature $\Sigma_{DT} = \langle S_{DT}, OP \rangle$, with $\text{bool} \in S_{DT}$ and $OP_{\text{bool}} = \{\text{false, true}\}$. We also assume as given a set of sorts $S_{OB}$ (object identifiers).

Definition 3.1 dyOSL is based on the following sets of sorts: $S = S_{DT} \oplus S_{OB}$, $S^* = S \oplus \{\tau\}$ and $\overline{S} = S \oplus \{\mu\} \oplus \{\sigma\}$.

The sort $\tau$ is called action sort, the sort $\mu$ is called mutation sort and the sort $\sigma$ is called specification sort.

Definition 3.2 An object signature $\Sigma$ is a tuple $\langle ACT, ATT, MUT, MAT \rangle$ such that:

- $ACT$ is an $S^*$-indexed family of finite sets such that $\bot \in ACT_\varepsilon$;
- $ATT$ is an $S^* \otimes S^*$-indexed family of finite sets such that $\text{occ} \in ATT_{\varepsilon, \tau}$;
- $MUT$ is an $(S + \{\sigma\})^*$-indexed family of finite sets such that $\overline{\bot} \in MUT_\varepsilon$;
- $MAT$ is an $\overline{S}^*$-indexed family of finite sets such that:
  - $\overline{\text{occ}} \in MAT_{\varepsilon, \mu}$;
  - $Ax \in MAT_{\varepsilon, \sigma}$. 

\qed
3.2. Terms and Formulae

We assume as given once and for all an $\mathcal{S}$-indexed family $\mathbf{X}$ of infinite, pairwise disjoint sets of base variables.

**Definition 3.3** The $\mathcal{S}$-indexed family $\mathcal{T}$ of sets of base terms is inductively defined as follows:

- $y \in \mathcal{T}_s$, provided that $y \in \mathbf{X}_s$;
- $f(t_1, \ldots, t_n) \in \mathcal{T}_s$, provided that $f \in OP_{s_1 \ldots s_n,s}$ and $t_i \in \mathcal{T}_{s_i}$ for every $i = 1, \ldots, n$;
- $z(t_1, \ldots, t_n) \in \mathcal{T}_c$, provided that $z \in ACT_{s_1 \ldots s_n}$ and $t_i \in \mathcal{T}_{s_i}$ for every $i = 1, \ldots, n$;
- $a(t_1, \ldots, t_n) \in \mathcal{T}_s$, provided that $a \in ATT_{s_1 \ldots s_n,s}$ and $t_i \in \mathcal{T}_{s_i}$ for every $i = 1, \ldots, n$.

Each element of $\mathcal{T}_s$ is called a base term of sort $s$.

**Definition 3.4** The set $\mathcal{L}$ of base formulae is inductively defined as follows:

- $t \in \mathcal{L}$, provided that $t \in \mathcal{T}_{\text{bool}}$;
- $(t = t') \in \mathcal{L}$, provided that $t, t' \in \mathcal{T}_s$;
- $* \in \mathcal{L}$;
- $(\neg \varphi) \in \mathcal{L}$, provided that $\varphi \in \mathcal{L}$;
- $(\varphi \Rightarrow \varphi'), (\varphi \lor \varphi') \in \mathcal{L}$, provided that $\varphi, \varphi' \in \mathcal{L}$.
• (∀y φ) ∈ ℒ, provided that φ ∈ ℒ and y ∈ X_s. □

As usual we are free to introduce abbreviations. In the sequel we use: (φ ∨ φ') for
((¬φ) ⇒ φ'), (φ ∧ φ') for (¬((¬φ) ∨ (¬φ'))), (φ ↔ φ') for (((φ ⇒ φ') ∧ (φ' ⇒ φ)), (∃y φ)
for (¬(∀y (¬φ))), (X φ) for ((φ ∧ (¬φ)) U φ), (F φ) for ((φ ∨ (¬φ)) U φ), and (G φ) for
(¬(F(¬φ))).

Additionally, we will use the weak until operator. It is defined as follows: (φ U^t ψ) is
an abbreviation of ((G φ) ∨ (φ U ψ)). In the case of U^t* we mean φ U^t* where φ is any
formula meaning that φ must hold until the next mutation if this one occurs.

The first level of our logic (the base level) defined so far is a usual linear temporal logic.
The following definitions extend this logic to a second level (the meta level) manipulating
sets of axioms.

As for the base level, we also assume as given once and for all an S-indexed family X
of infinite, pairwise disjoint sets of meta variables, such that X_s = X_s for each s ∈ S.

A formula without temporal operators and without the attribute symbol occ is called
a state formula. We denote the set of all state formulae (at the base level) by stℒ.

Definition 3.5 The S-indexed family ℋ of sets of meta terms is inductively defined as
follows:

• y ∈ ℋ_s, provided that y ∈ X_s;
• t ∈ ℋ_s, provided that t ∈ ℒ_s;
• m(t_1, . . . , t_n) ∈ ℋ_m, provided that m ∈ MUT_{s_1...s_n} and t_i ∈ ℋ_{s_i}, for every i =
1, . . . , n;
• b(t_1, . . . , t_n) ∈ ℋ_b, provided that b ∈ MAT_{s_1...s_n,s} and t_i ∈ ℋ_{s_i}, for every i = 1, . . . , n;
• [φ] ∈ ℋ_φ, provided that φ ∈ ℒ. □

Each element of ℋ_s is called a meta term of sort s. A set {φ_0, . . . , φ_n} of base formulae
can be denoted by the term [φ_0 ∧ . . . ∧ φ_n] ∈ ℋ_φ.

Definition 3.6 The set ℒ of meta formulae is inductively defined as follows:

• t ∈ ℒ, provided that t ∈ ℋ_book;
• (t = t') ∈ ℒ, provided that t, t' ∈ ℋ_s;
• (¬ψ) ∈ ℒ, provided that ψ ∈ ℒ;
• (ψ ⇒ ψ'), (ψ U ψ') ∈ ℒ, provided that ψ, ψ' ∈ ℒ;
• \((\forall y \psi) \in \overline{\mathcal{L}}, \) provided that \(\psi \in \overline{\mathcal{L}}\) and \(y \in X_s.\)

Like for the base level, we are also free to introduce the first-order and temporal abbreviations for meta formulae.

Analogously to \(st\mathcal{L}\), a formula without temporal operators and without the meta attribute symbol \(\overline{\mathcal{O}}\) is called a state meta formula. We denote the set of all state meta formulae by \(\overline{\mathcal{L}}.\)

### 3.3 Pre-interpretation structures

We assume given an algebra \(dt\) over \(\Sigma_{DT} = \langle S_{DT}, OP \rangle\) such that \(dt_{\text{bool}} = \{false_{dt}, true_{dt}\}.\)

We also assume given an \(SOB\)-indexed family \(pop\) (\(pop\) for population) of carrier sets for \(SOB,\) and we extend the notation so that \(pop_s\) stands for \(dt_s\) when \(s \in S_{DT},\) and \(pop_{\sigma}\) stands for \(\overline{\mathcal{L}}.\)

**Definition 3.7** The set \(act\) of actions over \(\Sigma\) for \(pop\) is as follows:

\[
\bigcup_{\langle s_1...s_n \rangle \in S^*} \{z(u_1, ..., u_n) : z \in ACT_{s_1...s_n}, u_i \in pop_{s_i}, \text{ for } i = 1, ..., n\}
\]

In the sequel we write \(pop_{\tau}\) for \(act.\)

**Definition 3.8** The set \(att_s\) of attributes of sort \(s\) over \(\Sigma\) for \(pop\) is as follows:

\[
\bigcup_{\langle s_1...s_n \rangle \in \overline{\Sigma}^*} \{a(u_1, ..., u_n) : a \in ATT_{s_1...s_n, s}, u_i \in pop_{s_i}, \text{ for } i = 1, ..., n\}
\]

**Definition 3.9** The set of attributes over \(\Sigma\) for \(pop\) is as follows:

\[
\bigcup_{s \in \overline{\Sigma}} att_s
\]

**Definition 3.10** The set \(mut\) of mutations over \(\Sigma\) for \(pop\) is as follows:

\[
\bigcup_{\langle s_1...s_n \rangle \in (\overline{\Sigma} \oplus \{\sigma\})^*} \{m(u_1, ..., u_n) : m \in MUT_{s_1...s_n}, u_i \in pop_{s_i}, \text{ for } i = 1, ..., n\}
\]
In the sequel we write $\pop_\mu$ for $\textit{mut}$.

**Definition 3.11** The set $\textit{mat}_s$ of mutation attributes of sort $s$ over $\Sigma$ for $\pop$ is as follows:

$$\bigcup_{\langle s_1\ldots s_n \rangle \in \Sigma} \{b(u_1, \ldots, u_n) : b \in \textit{MAT}_{s_1\ldots s_n}, u_i \in \pop_{s_i}, \text{ for } i = 1, \ldots, n\}$$

\[\square\]

**Definition 3.12** The set of mutation attributes over $\Sigma$ for $\pop$ is as follows:

$$\bigcup_{s \in \Sigma} \textit{mat}_s$$

\[\square\]

We use the $\Sigma$-indexed family $\textit{obs}$ to denote $\textit{att} \oplus \textit{mat}$, defined as follows:

- $\textit{obs}_s = \textit{att}_s \oplus \textit{mat}_s$, for each $s \in \Sigma$;
- $\textit{obs}_s = \textit{mat}_s$, for each $s \in (\Sigma \setminus \Sigma)$.

For each element $o \in \textit{obs}_s$, called observation symbol of sort $s$, we say that $\pop_s$ is domain $D_o$ of $o$.

**Definition 3.13** A pre-interpretation structure over $\Sigma$ is triple $I = \langle \textit{dt}, \{\pop\}_{s \in S_{\textit{obs}}}, \lambda \rangle$, with $\lambda = \{\lambda_k(o)\}_{k \in \mathbb{N}, o \in \textit{obs}}$, such that:

- $\lambda_k(o) \in \pop_s$ whenever $o \in \textit{obs}_s$;
- $\lambda_k(\textit{occ}) \neq \bot$ \Rightarrow $\lambda_k(\overline{\textit{occ}}) = \overline{\bot}$;
- if $\lambda_k(\textit{occ}) = \bot$ then $\lambda_{k+1}(o) = \lambda_k(o)$ for each $o \in \textit{att}$ such that $o \neq \textit{occ}$;
- if $\lambda_k(\overline{\textit{occ}}) = \overline{\bot}$ then $\lambda_{k+1}(o) = \lambda_k(o)$ for each $o \in \textit{mat}$ such that $o \neq \overline{\textit{occ}}$.  

\[\square\]

### 3.4 Satisfaction

After having defined the syntax of the logic and the underlying semantic structures, we can now start to define the satisfaction relation between formulae and structures.

**Definition 3.14** An assignment $\rho$ for $\pop$ is an $\Sigma$-indexed family of maps $\rho_s : X_s \rightarrow \pop_s$.  

\[\square\]
**Definition 3.15** Let $y \in X_s$ and $\rho, \rho'$ be assignments. We say that $\rho$ and $\rho'$ are $y$-equivalent iff for every $s' \in S$ and $y' \in X_{s'}$, $\rho_s(y') = \rho'_s(y')$ whenever $y' \neq y$. □

**Definition 3.16** The interpretation of base terms in $I$ for $\rho$ at point $k \in \mathbb{N}$ is inductively defined as follows:

- $[y]_{i, \rho, k, s} = \rho_s(y)$;
- $[f(t_1, \ldots, t_n)]_{i, \rho, k, s} = f_{dt_{s_1}, \ldots, s_n}([t_1]_{i, \rho, k, s_1}, \ldots, [t_n]_{i, \rho, k, s_n})$;
- $[z(t_1, \ldots, t_n)]_{i, \rho, k, r} = z([t_1]_{i, \rho, k, s_1}, \ldots, [t_n]_{i, \rho, k, s_n})$;
- $[\alpha(t_1, \ldots, t_n)]_{i, \rho, k, s} = \lambda_k(\alpha([t_1]_{i, \rho, k, s_1}, \ldots, [t_n]_{i, \rho, k, s_n}))$. □

**Definition 3.17** The interpretation of meta terms in $I$ for $\rho$ at point $k \in \mathbb{N}$ is inductively defined as follows:

- $[y]_{i, \rho, k, s} = \rho_s(y)$;
- $[t]_{i, \rho, k, s} = \langle t \rangle_{i, \rho, k, s}$;
- $[m(t_1, \ldots, t_n)]_{i, \rho, k, u} = m([t_1]_{i, \rho, k, s_1}, \ldots, [t_n]_{i, \rho, k, s_n})$;
- $[b(t_1, \ldots, t_n)]_{i, \rho, k, s} = \lambda_k(b([t_1]_{i, \rho, k, s_1}, \ldots, [t_n]_{i, \rho, k, s_n}))$;
- $[[\varphi]]_{i, \rho, k, \sigma} = \varphi$. □

**Definition 3.18** The satisfaction of base formulae in $I$ for $\rho$ at point $k \in \mathbb{N}$ is inductively defined as follows:

- $I, \rho, k \vdash t$ iff $\langle t \rangle_{i, \rho, k, bool} = true$;
- $I, \rho, k \vdash (t = t')$ iff $\langle t \rangle_{i, \rho, k, s} = \langle t' \rangle_{i, \rho, k, s}$;
- $I, \rho, k \vdash *$ iff $\langle occ \rangle_{i, \rho, k, u} \neq \top$;
- $I, \rho, k \vdash \neg \neg \varphi$ iff $I, \rho, k \vdash \varphi$;
- $I, \rho, k \vdash (\varphi \Rightarrow \varphi')$ iff not $I, \rho, k \vdash \varphi$ or $I, \rho, k \vdash \varphi'$;
- $I, \rho, k \vdash (\varphi \lor \varphi')$ iff there is $k' \in \mathbb{N}$ such that $k < k'$, $I, \rho, k' \vdash \varphi'$, and for each $k'' \in \mathbb{N}$ if $k < k'' < k'$ then $I, \rho, k'' \vdash \varphi$. 
\[ I, \rho, k \models (\forall y \varphi) \text{ iff } I, \rho', k \models \varphi \text{ for every } \rho' \text{ } y\text{-equivalent to } \rho. \]

**Definition 3.19** Given a pre-interpretation structure \( I \) and a base formula \( \varphi \in \mathcal{L} \) we say that \( I \) satisfies \( \varphi \), written \( I \models \varphi \), iff \( I, \rho, 0 \models \varphi \) for every assignment \( \rho \).

It should be remarked, that we use an *anchored version of temporal logic* for dyOSL. This due to the fact that we do not want to have an implicit necessitation to all formulae dynamically added to the specification by mutations. Exactly that would be the case in a fluent version of temporal logic. Because it should be possible to again remove an axiom dynamically added before, we must avoid the implicit necessitation which states that something holds for ever and cannot be removed.

**Definition 3.20** The *satisfaction* of meta formulae in \( I \) for \( \rho \) at point \( k \in \mathbb{N} \) is inductively defined as follows:

- \( I, \rho, k \models t \) iff \( \models_{I, \rho, k, \text{bool}} = \text{true} \).
- \( I, \rho, k \models (t = t') \) iff \( \models_{I, \rho, k, s} = [t']_{I, \rho, k, s} \).
- \( I, \rho, k \models (\neg \psi) \) iff not \( I, \rho, k \models \psi \).
- \( I, \rho, k \models (\psi \Rightarrow \psi') \) iff not \( I, \rho, k \models \psi \) or \( I, \rho, k \models \psi' \).
- \( I, \rho, k \models (\psi \cup \psi') \) iff there is \( k' \in \mathbb{N} \) such that \( k < k', I, \rho, k' \models \psi' \), and for each \( k'' \in \mathbb{N} \) if \( k < k'' < k' \) then \( I, \rho, k'' \models \psi \).
- \( I, \rho, k \models (\forall y \psi) \) iff \( I, \rho', k \models \psi \) for every \( \rho' \) \( y\)-equivalent to \( \rho \).

**Definition 3.21** Given a pre-interpretation structure \( I = \langle dt, \{\text{pop}\}_{s \in \text{OB}}, \lambda \rangle \) we say that \( I \) is an *interpretation structure* iff, for every assignment \( \rho \) and \( k \in \mathbb{N}, I, \rho, k \models \lambda_k(Ax) \) whenever \( \lambda_{k-1}(\text{occ}) \neq \bot \).

**Definition 3.22** Given an interpretation structure \( I \) and a meta formula \( \psi \in \mathcal{L} \), we say that \( I \) satisfies \( \psi \), written \( I \models \psi \), iff \( I, \rho, 0 \models \psi \) for every assignment \( \rho \).

**Definition 3.23** Let \( \Psi \subseteq \mathcal{L} \) and \( \psi \in \mathcal{L} \). We say that \( \Psi \) *entails* \( \psi \), written \( \Psi \models \psi \), iff, for every interpretation structure \( I \), if \( I \models \varphi \), for every \( \varphi \in \Psi \), then \( I \models \psi \).

**Definition 3.24** Let \( \Psi \subseteq \mathcal{L} \). The *closure by entailment* of \( \Psi \) is the set \( \Psi^\models = \{ \psi : \Psi \models \psi \} \).
3.5 Specifications and Theories

Finally, we can define the notions of specification and theory for the logic dyOSL.

**Definition 3.25** An object specification is a pair \( \langle \Sigma, \Gamma \rangle \) such that \( \Sigma \) is an object signature and \( \Gamma \) is again a pair \( \langle \underline{\Gamma}, \overline{\Gamma} \rangle \) such that \( \underline{\Gamma} \subseteq \underline{\mathcal{L}} \) and \( \overline{\Gamma} \subseteq \overline{\mathcal{L}} \).

By splitting \( \Gamma \) into \( \underline{\Gamma} \) and \( \overline{\Gamma} \) we are able to distinguish between rigid axioms and non-rigid axioms, respectively.

**Definition 3.26** An object theory is an object specification \( \langle \Sigma, \Gamma \rangle \) such that \( \Gamma^{\overline{\Gamma}} = \Gamma \). ☐

From now on we concentrate on a given object specification \( \text{spec} = \langle \Sigma, \Gamma \rangle \).
Chapter 4

Translation of Language into Logic

In this chapter, we present the basic principles for translating object specifications with evolving behaviour into corresponding sets of dyOSL formulae. Due to the fact that the translation of the standard parts of object specifications, i.e. the specification of the rigid behaviour, into OSL is already described in detail in [Jungclaus, 1993], we here concentrate on the parts for specifying evolving behaviour.

We are going to show the translation of the language into the logic by example. For that we consider a modifiable timer.

Example 4.1 The specification of a modifiable timer is given in Fig. 4.1. The timer can be started by setting time and duration. The timer will then decrease the time until zero, then the alarm will be raised. The alarm will stop when the duration has been decreased to zero. For the action ‘press_button’ there is no effect specified as rigid axiom.

By means of the mutation actions ‘m1’ and ‘m2’ it is possible to change the behaviour of the timer. Each of these two mutation actions assign a certain effect to the ‘press_button’ action. ‘m1’ introduces an axiom that if ‘press_button’ occurs when the alarm is raised then the timer has to be restarted such that an alarm will be raised after 100 seconds (and this alarm will then last for 10 seconds). ‘m2’ adds an axiom describing that, if ‘press_button’ occurs when the timer has been started but the alarm has not yet be raised, then the raising of the alarm is delayed by 60 additional seconds.

We now illustrate how to translate this specification into the logic. We will only sketch the main transformation steps. The first step is the extraction of the signature.

Example 4.2 (Timer signature) For the previous example, we get the following signature $\Sigma_{\text{Timer}} = \langle ACT, ATT, MUT, MAT \rangle$:

- $ACT_ε = \{ \downarrow_{\text{tic}}, \text{press\_button} \}, \quad ACT_{\text{nat\_nat}} = \{ \text{start} \}$;
- $\quad \text{ATT}_{\text{ε, nat}} = \{ \text{time, duration} \}, \quad \text{ATT}_{\text{ε, boot}} = \{ \text{alarm} \}$;
object Timer

template

attributes
  time: nat initialized 0,
  duration: nat initialized 0,
  alarm: bool initialized false;

actions
  start(time:nat,duration:nat),
  tic,
  press_button;

rigid axioms
  start(t,d)
    changing alarm = false,
    time = t,
    duration = d;

  tic
    enabled time 0 or duration 0,
    changing if time 1 then time = time-1,
    if time = 1 then alarm = true,
    if time = 0 then duration = duration-1,
    if time = 0 and duration = 1 then alarm = false;

axiom attributes
  Axioms initialized {};

mutators
  m1,
  m2,
  reset;

dynamic specification
  m1
    changing Axioms = Axioms +
      { press_button
        calling start(100,10) if alarm=true };

  m2
    changing Axioms = Axioms +
      { press_button
        changing if alarm=false
          then time=time+60 };

  reset
    changing Axioms = {};

end object

Figure 4.1: Specification of a Timer.
• $MUT_e = \{ \underline{\text{I}}, m1, m2, \text{reset} \}$;
• $MAT_{e, \mu} = \{ \overline{\text{occ}} \}$, $MAT_{e, \sigma} = \{ Ax \}$.

The dyOSL signature can be directly derived from the specification signature of a Troll specification. The special signature elements $\underline{\text{I}}$, $\overline{\text{occ}}$ and $Ax$ have to be added for each specification.

The next step is to extract the specification (note: we have to change the definition of specification, so that it includes the part corresponding to the rigid axioms)

Example 4.3 (Timer specification) For the previous example, we get the following specification $(\Sigma, (\underline{\Gamma}, \Gamma))$:

• $\underline{\Gamma}$ includes the following axioms:
  1. $time = 0 \land duration = 0 \land alarm$;
  2. $G(oc \equiv tic \Rightarrow (time \neq 0 \lor duration \neq 0))$;
  3. $G(oc \equiv start(t, d) \land \varphi_{\text{time}, \text{duration}, \text{alarm}} \Rightarrow (X \varphi))$, for $\varphi \in st \mathcal{L}$;
  4. $G((oc \equiv tic \land time > 1 \land \varphi_{\text{time-1}}) \Rightarrow (X \varphi))$, for $\varphi \in st \mathcal{L}$;
  5. $G((oc \equiv tic \land time = 1 \land \varphi_{\text{time-1, true}}) \Rightarrow (X \varphi))$, for $\varphi \in st \mathcal{L}$;
  6. $G((oc \equiv tic \land time = 0 \land duration \neq 1 \land \varphi_{\text{duration-1}}) \Rightarrow (X \varphi))$, for $\varphi \in st \mathcal{L}$;
  7. $G((oc \equiv tic \land time = 0 \land duration = 1 \land \varphi_{\text{duration-1, false}}) \Rightarrow (X \varphi))$, for $\varphi \in st \mathcal{L}$.

Rule 1. results from the initialization of state attributes. Rule 2. is the translation of the enabling condition for the action tic. The remaining rules 3. to 7. model the effect of the actions on state formulae stating that valid state formulae are valid after the state change if we substitute in these formulae attribute names with the new attribute values (see [Sernadas et al., 1996] for more details).

• $\Gamma$ includes the following axioms:
  1. $Ax = \{ \}$;
  2. $G(oc \equiv m1 \land \varphi_{Ax + \{\psi_1\}} \Rightarrow (X \varphi))$ for $\varphi \in st \mathcal{L}$ and $\psi_1$ is $(\star \Rightarrow ((oc \equiv press\_button \Rightarrow alarm = \text{true}) U^2 \star)) \land (\star \Rightarrow ((oc \equiv press\_button \Rightarrow X(oc \equiv start(100, 10))) U^2 \star))$;
  3. $G(oc \equiv m2 \land \varphi_{Ax + \{\psi_2\}} \Rightarrow (X \varphi))$ for $\varphi \in st \mathcal{L}$ and $\psi_2$ is $(\star \Rightarrow ((oc \equiv press\_button \Rightarrow alarm = \text{false}) U^2 \star)) \land (\star \Rightarrow ((oc \equiv press\_button \land \psi_{\text{time} + 60} \Rightarrow (X\psi)) U^2 \star))$, where $\psi \in st \mathcal{L}$;
  4. $G((oc \equiv \text{reset} \land \varphi_{\{\}} \Rightarrow (X \varphi))$ for $\varphi \in st \mathcal{L}$. 
Again, rule 1. models the initialization of the meta attribute $A_x$. The remaining rules 2. to 4. model the valuation of the mutation actions in analogy to the previous rules for the base level.
Chapter 5

Using the Logical Framework

In this chapter we illustrate the use of the logic to prove some properties about dyOSL specifications (cf. Sernadas et al., 1996) where a more detailed description of proofs can be found for the basic OSL).

5.1 A Hilbert calculus

We start by introducing a Hilbert calculus for the logic. We have to provide two consequence operators, one to reason at the base level and another to reason at the meta-level. These must be defined within the scope of a specification.

**Definition 5.1** The closure by consequence of a specification $\langle \Sigma, \underline{\Gamma}, \Gamma \rangle$ is the tuple $\langle \Sigma, \Gamma^+, \Gamma^- \rangle$ such that $\Gamma^+$ and $\Gamma^-$ are inductively defined as follows:

1. $\psi \in \Gamma^+$ provided that $\psi \in \underline{\Gamma}$;
2. $\psi \in \Gamma^+$ provided that $\psi \in Ax$;
3. $\psi \in \Gamma^+$ provided that $\psi$ is an axiom of an anchored linear temporal, first order with equality calculus;
4. $\psi' \in \Gamma^+$ provided that $\psi$ is a rigid data formula such that $\rho, dt \vdash \psi$ for every assignment $\rho$;
5. $\psi' \in \Gamma^+$ provided that $\psi, (\psi \Rightarrow \psi') \in \Gamma^+$;
6. $\psi' \in \Gamma^+$ provided that $\psi, (\psi \Rightarrow \psi') \in \Gamma^+$;
7. $\varphi \in \Gamma^+$ provided that $\varphi \in \Gamma^-$, for each $\varphi \in \underline{\Gamma}$.
8. \((z(x_1, \ldots, x_n) = z(x'_1, \ldots, x'_n)) \Rightarrow ((x_1 = x'_1) \land \cdots \land (x_n = x'_n)) \in \Gamma^+\) provided that \(z \in ACT_{s_1, \ldots, s_n}\);

9. \((m(x_1, \ldots, x_n) = m(x'_1, \ldots, x'_n)) \Rightarrow ((x_1 = x'_1) \land \cdots \land (x_n = x'_n)) \in \Gamma^+\) provided that \(m \in MUT_{s_1, \ldots, s_n}\);

10. \((- (z(x_1, \ldots, x_n) = z'(x'_1, \ldots, x'_{n'}))) \in \Gamma^+\) provided that \(z \in ACT_{s_1, \ldots, s_n}, z' \in ACT_{s'_1, \ldots, s'_{n'}}, \) and \(z \neq z'\);

11. \((- (m(x_1, \ldots, x_n) = m'(x'_1, \ldots, x'_{n'}))) \in \Gamma^+\) provided that \(m \in MUT_{s_1, \ldots, s_n}, m' \in MUT_{s'_1, \ldots, s'_{n'}}, \) and \(m \neq m'\);

12. \((\forall y (\forall_{s_1, \ldots, s_n} (\exists y_1 \cdots y_n (y = z(y_1, \ldots, y_n) \cdots))) \in \Gamma^+\);

13. \((\forall y (\forall_{s_1, \ldots, s_n} (\exists y_1 \cdots y_n (y = m(y_1, \ldots, y_n) \cdots))) \in \Gamma^+\);

14. \((\text{oce} = m(t_1, \ldots, t_n) \land m(t_1, \ldots, t_n) \neq \top) \Leftrightarrow *) \in \Gamma^+\);

15. \((\text{oce} \neq \bot \Rightarrow \text{oce} = \top) \in \Gamma^+\);

16. \((\text{oce} = \bot \Rightarrow ((a(x_1, \ldots, x_n) = y) \Rightarrow (X(a(x_1, \ldots, x_n) = y)))) \in \Gamma^+\) provided that \(a \in ATT_{s_1, \ldots, s_n, s} \) and \(a \neq \text{oce}\);

17. \((\text{oce} = \top \Rightarrow ((b(x_1, \ldots, x_n) = y) \Rightarrow (X(b(x_1, \ldots, x_n) = y)))) \in \Gamma^+\) provided that \(b \in MAT_{s_1, \ldots, s_n, s} \) and \(b \neq \text{oce}\).}

We briefly explain some of these rules (note that some of these are very similar to the ones presented in [Sernadas et al., 1996]). Rule 1 states that each axiom of the specification is a theorem: \(AX\). Rule 3 captures the first-order linear temporal logic reasoning: \(LPO\). Rules 5 and 6 are \textit{modus ponens}: \(MP\). Rule 7 states that the theorems of the base level are also theorems of the meta-level. Rules 8-13 reflect that actions (resp. mutation actions) are not interpreted and are generated: \(ACT\) (resp. \(MUT\)). Rule 14 reflects a property of \(*\) and mutations: \(STAR\). Rules 15-17 reflect properties for the life-cycles: \(RUN\).

The presented logical system is sound that is if \(\varphi \in \Gamma^+\) then \(\Gamma \models \varphi\) where the latter means: an interpretation structure \(I\) satisfies \(\varphi\) whenever \(I\) satisfies all the formulae in \(\Gamma\).

**Definition 5.2** Let \(\text{spec} = \langle \Sigma, \Gamma, \overline{\Gamma} \rangle\) be a specification and \(\Psi = \langle \Psi, \overline{\Psi} \rangle\) such that \(\Psi \subseteq \Sigma\) and \(\overline{\Psi} \subseteq \overline{\Sigma}\). The \textit{consequence} of \(\Psi\) on \(\text{spec}\) is the tuple \(\text{spec}^{\Psi^+} = \langle \Sigma, \Gamma^{\Psi^+}, \overline{\Gamma}^{\Psi^+} \rangle\) where \(\Gamma^{\Psi^+}\) and \(\overline{\Gamma}^{\Psi^+}\) are inductively defined as follows:

- \(\varphi \in \Gamma^{\Psi^+}\) provided that \(\varphi \in \Gamma^+\);
\[ \varphi \in \Gamma \Phi^- \text{ provided that } \varphi \in \Psi; \]

\[ \psi \in \Gamma \Phi^- \text{ provided that } (\varphi \Rightarrow \psi), \varphi \in \Gamma \Phi^-; \]

\[ \varphi \in \overline{\Gamma} \Phi^- \text{ provided that } \varphi \in \overline{\Gamma}; \]

\[ \varphi \in \overline{\Gamma} \Phi^- \text{ provided that } \varphi \in \overline{\Psi}; \]

\[ \psi \in \overline{\Gamma} \Phi^- \text{ provided that } (\varphi \Rightarrow \psi), \varphi \in \overline{\Gamma} \Phi^- \text{.} \]

We denote \( \varphi \in \Gamma \Phi^- \) (resp. \( \varphi \in \overline{\Gamma} \Phi^- \)) by \( \Psi \vdash \varphi \) (resp. \( \overline{\Psi} \vdash \varphi \)).

### 5.2 An invariant calculus

In this section we present a calculus for proving invariants (cf. [Manna and Pnueli, 1992]).

Our aim is to give a proof-theoretic approach to certifying invariants for an evolving specification. Evolving sets of formulae are usually interpreted at runtime which is very flexible but does not allow for proving interesting global invariants, i.e. formulae that are to be satisfied all over.

We restrict ourselves to prove base invariants, i.e. invariants not containing meta formulae as subformulae. To prove invariants, we have to check the invariant against each possible set of base axioms arising during specification evolution. We call such formulae sets completions. A completion is a set of base-formulae, that completely specify the effects of actions, and which includes the set of rigid axioms. A completion is reachable if it can be constructed in the following way:

- For a given value of the meta attribute \( Ax \), a completion is the union of following sets:
  - the rigid axioms \( rigidaxioms \),
  - the values of \( Ax \), and
  - a frame rule for each action \( act \) which has no explicit valuation rule in \( rigidaxioms \) or \( Ax \):
    \[
    (\ast \Rightarrow ((occ = act \land \varphi) \Rightarrow (X \varphi)) \cup^\ast \ast))
    \]
- The value of \( Ax \) is the initial value or the result of an enabled sequence of mutation events.
The number of reachable completions is finite, if we use constant terms of type \textit{spec} and set-operations on the \textit{Ax} meta attribute only. For a proof, we have to consider all reachable completions.

Some invariants can even be proven based on the \textit{rigidaxioms} only. In general, this is only possible for invariants containing no state-dependent attribute or in the situation where all actions are fully specified — for more interesting assertions, we have to use the presented method relying on completions.

\textbf{Example 5.3} In the case of our example, from the specification of the \textit{Timer}, we can "derive" the following completions:

\begin{itemize}
  \item \( C_1 = \text{rigidaxioms} + \)
    \{ (*) \Rightarrow (((\textit{occ} = \textit{press.button} \land \varphi) \Rightarrow (X \varphi)) U^\tau *) \};
  \item \( C_2 = \text{rigidaxioms} + \)
    \{ (*) \Rightarrow (((\textit{occ} = \textit{press.button} \land \textit{alarm} = \textit{true}) \Rightarrow X(\textit{occ} = \textit{start(100, 10)}) U^\tau *)),
    (\star \Rightarrow (((\textit{occ} = \textit{press.button} \land \varphi) \Rightarrow X \varphi)) U^\tau *)) \};
  \item \( C_3 = \text{rigidaxioms} + \)
    \{ (*) \Rightarrow (((\textit{occ} = \textit{press.button} \land \textit{alarm} = \textit{false} \land \textit{time}^{\text{time+60}}) \Rightarrow (X \varphi))) U^\tau *)) \};
  \item \( C_4 = \text{rigidaxioms} + \)
    \{ (*) \Rightarrow (((\textit{occ} = \textit{press.button} \land \textit{alarm} = \textit{true}) \Rightarrow X(\textit{occ} = \textit{start(100, 10)}) U^\tau *)),
    (\star \Rightarrow (((\textit{occ} = \textit{press.button} \land \textit{alarm} = \textit{false} \land \textit{time}^{\text{time+60}}) \Rightarrow (X \varphi))) U^\tau *)) \};
\end{itemize}

The formula added in completion \( C_1 \) models the frame rule for the action \textit{press.button}: As long as nothing additional is stated in the meta part of the specification, this action has no effect on the object state. Completion \( C_2 \) includes this frame rule as well. For completion \( C_3 \) this frame rule is overruled by a rule changing all properties by modifying the time attribute. Completion \( C_4 \) includes all \textit{rigidaxioms} added my the mutation actions ‘m1’ and ‘m2’. In addition it contains the same frame rule as \( C_3 \) due to the fact that that frame rule overrules the one of \( C_2 \). \vspace{0.5cm}

\hfill \Box

In what follows, the results only hold when \( \varphi \) is a state formula (i.e., a formula without temporal operators and without the symbol \textit{occ}).

\textbf{Definition 5.4} (Global invariant rule) Let \(|\Sigma, \Gamma, \Gamma|\) be a specification, and let \( C_1, \ldots, C_n \) be its completions. Then:

\begin{itemize}
  \item \((G \varphi) \in \Gamma^+\) provided that:
    \begin{itemize}
      \item \( \exists \varphi; \)
    \end{itemize}
\end{itemize}
\[ -C_i \models ((\varphi \land \star) \Rightarrow (\varphi \cup \star)) \text{ for } i = 1, \ldots, n. \] 

Beside the global invariant rule we also need the following local invariant rule.

**Definition 5.5 (Local invariant rule)** Let \( \langle \Sigma, \Delta, \Gamma \rangle \) be a specification, and let \( C \) be one of its completions. Then:

- \( C \models ((\varphi \land \star) \Rightarrow (\varphi \cup \star)) \) provided that:
  
  \[ -C_i \models ((\varphi \land \star) \Rightarrow ((\texttt{occ} = a \land \varphi) \Rightarrow (X \varphi)) \cup \star) \text{ for every } a \in \text{ACT}_w; \]

Note that in order to prove the soundness of this rule, we may use the following result:

\[ (\star \land \varphi) \Rightarrow (X \varphi) \]

**Example 5.6** Assume that we want to prove the following property:

\[ \models (G((\text{time} \neq 0) \Rightarrow (\text{alarm} = \text{false}))) \]

From example 5.3 we now that we have four possible completions, so we have to use rule 5.5 for each one of these completions. We only consider one of these. The others are similar. Let us consider \( C_3 \). Let \( \alpha \) be \((\text{time} \neq 0) \Rightarrow (\text{alarm} = \text{false})\).

**A** \[ \models (\text{time} \neq 0) \Rightarrow (\text{alarm} = \text{false}) \]

1. \( \text{time} = 0 \land \text{duration} = 0 \land \text{alarm} = \text{true} \in \Gamma \) \hspace{2cm} \text{AX}
2. \((\text{time} \neq 0) \Rightarrow (\text{alarm} = \text{false}) \) \in \Gamma^+ \hspace{2cm} \text{LPO : 1}
3. \[ \models (\text{time} \neq 0) \Rightarrow (\text{alarm} = \text{false}) \] \hspace{2cm} \text{by definition of derivation}

**B** \[ C_3 \models (\alpha \land \star) \Rightarrow ((\texttt{occ} = \text{start}(t, d) \land \alpha) \Rightarrow (X \alpha) \cup \star) \]

1. \[ G((\texttt{occ} = \text{start}(t, d) \land (t \neq 0 \Rightarrow \text{false} = \text{false})) \Rightarrow (X \alpha)) \in \Gamma \] \hspace{2cm} \text{AX}
2. \[ G((\texttt{occ} = \text{start}(t, d)) \Rightarrow (X \alpha)) \in \Gamma^+ \] \hspace{2cm} \text{LPO : 1}
3. \[ G((\texttt{occ} = \text{start}(t, d) \land \alpha) \Rightarrow (X \alpha)) \in \Gamma^+ \] \hspace{2cm} \text{LPO : 2}
4. \[(\alpha \land \star) \Rightarrow ((\texttt{occ} = \text{start}(t, d) \land \alpha) \Rightarrow (X \alpha) \cup \star)) \in \Gamma^+ \] \hspace{2cm} \text{LPO : 4}
5. \[ C_3 \models (\alpha \land \star) \Rightarrow ((\texttt{occ} = \text{start}(t, d) \land \alpha) \Rightarrow (X \alpha) \cup \star)) \] \hspace{2cm} \text{def of derivation}

**C** \[ C_3 \models (\alpha \land \star) \Rightarrow ((\texttt{occ} = \text{press.button} \land \alpha) \Rightarrow (X \alpha) \cup \star)) \]

1. \[ (\star \Rightarrow ((\texttt{occ} = \text{press.button} \land ((\text{time} + 60 \Rightarrow \text{alarm} = \text{false}) \Rightarrow (X \alpha)) \cup \star)) \in C_3 \] \hspace{2cm} \text{def of der.}
2. \[ (\star \Rightarrow ((\texttt{occ} = \text{press.button} \Rightarrow \text{alarm} = \text{false}) \cup \star)) \in C_3 \] \hspace{2cm} \text{def of der.}
3. \[ ((\alpha \land \star) \Rightarrow ((\texttt{occ} = \text{press.button} \land \alpha) \Rightarrow (X \alpha) \cup \star)) \in C_3 \] \hspace{2cm} \text{LPO : 1, 2}

The rest of the proof is similar. The reader is referred to [Sernadas et al., 1996] for more detailed proofs of this sort.
5.2. An invariant calculus
Chapter 6

Concluding Remarks

In this paper, we presented an extension of object specification for capturing evolving behaviour. For motivating the necessary of dealing with evolving behaviour we used an extended object specification language. Furthermore, we defined the logic dyOSL. Thereby, we provided a semantics based on an extended linear temporal logic. We discussed the translation of the specification language into that logic and, finally, we demonstrated the usage of the logical framework for proving object properties from their (dynamic) specifications.

The extensions we are considering form a stable basis for future work in which we aim at integrating additional aspects like normative specification (based on deontic logic). In [Saake et al., 1995a] we sketch a vision of a future specification language incorporating several specification concepts going beyond current object-oriented specification formalisms. The work presented here is a major step into that direction. This is due to the fact that we leave the level of first-order temporal logic by allowing the manipulation of first-order specification axioms within the specification. Thereby, we introduce some kind of reasoning capabilities into the objects.

We do not strive for a real higher-order approach. Nevertheless, we think that some higher-order aspects (as they have recently been investigated in several areas, for instance for algebraic specification in [Möller, 1994, Schobbens, 1994]) could help. Another approach using a 2-level logic is presented in [Gabbay and Hunter, 1993]. There, the second level in the logic is introduced for handling inconsistencies on the first level.

The presented proof-theoretic approach seems to be very restricted because we have to be able to compute all completions. However, we can see this situation from the opposite: If we can prove for each specification in isolation, that it satifies together with the rigid axioms the invariant, all of them can be savely added (by union) to $Ax$ during object lifetime. This holds because of the construction of completions where the the number of ‘frame rules’ may only decrease if we add additional formulae to $Ax$.

Of course, there still is a large number of open questions due to the fact that the work
presented here has to be considered as a starting point for further research. Among the issues for future work developing guidelines for specifying objects having an evolutionary behaviour is very important. We have to support specifiers in making their decisions which axioms should be rigid and which should be part of the dynamic specification part. Concerning the specification language syntactic restrictions must be provided for specifying behaviour evolution in a more controlled way. Furthermore, we have not dealt with schema evolution in this paper. It may happen that changing the dynamic behaviour of objects requires to introduce new attributes or to modify the types of existing attributes. For that, appropriate extensions of $dyOSL$ are needed.
Bibliography


