A Basic Calculus for Verifying Properties of Interacting Objects

Stefan Conrad

Universität Magdeburg, Institut für Technische Informationssysteme

Postfach 4120, D–39016 Magdeburg, Germany

e–mail: conrad@iti.cs.uni-magdeburg.de
tel: ++49-391-67-18066
fax: ++49-391-67-12020

Abstract

We introduce a basic calculus for expressing and proving properties of interacting objects. The objects considered have dynamic behaviour and are organized into object communities in a hierarchical way. After a detailed and formal presentation of the calculus, essential properties of the calculus are discussed, especially the question of compositionality.

The calculus constitutes a basis for investigating issues of verifying properties of objects and object communities. In consequence, we have focused on a small number of essential concepts. This calculus can be seen as an extensible basis applicable to different object-oriented specification frameworks.

Keywords: object specification, dynamic objects, verification, compositional specification, compositional verification, basic calculus.

1 Introduction

In the last years, formal specification techniques have received more and more attention (cf. [4, 45, 3]). They are used for supporting the program and system development process. Assuming a correct specification of a system, each step within the software development process can or should be verified or validated to yield a correct result with respect to the formal specification.

Especially, the development of complex software systems requires more and more formal support in order to get a grasp of the increasing requirements on software. Here, important goals are reliability and correctness. Typical complex software systems are reactive systems or information systems.

Among the different approaches object-orientation seems to be most adequate for capturing static as well as dynamic aspects of such systems. Especially for the development of information systems, object-oriented modelling of the Universe of Discourse (UoD) allows an easy-to-understand treatment of entities which are to be represented in an information system (e.g. [43, 35, 29]. Object-oriented modelling or specification is now widely accepted and understood and is based on semantic foundations which have been worked out in [30, 38, 42, 13, e.g.].

A formal specification is an indispensable basis for proving properties of the system to be developed, e.g., the correctness of the final implementation can only be stated with regard to a formal specification. However, only considering the correctness of implementation steps cannot guarantee that the final implementation meets the requirements. Therefore, it is necessary to convince oneself of the adequacy of specifications as early as possible. There are two main ways of checking adequacy. On the one hand we can validate

the specification by generating prototype systems from the specification or by using an animation system. On the other hand we can formulate crucial properties of the information system (or of the objects constituting the system) and prove these properties on the basis of the given specification.

Here, we concentrate on the latter point. We present a basic calculus for formulating and proving object properties. It is a many-sorted, first-order sequent-based calculus which brings together the advantages of two other approaches ([17] and [39]). From [17] we especially take the structure of formulas (Gentzen-style formulas) which allows a very simple and comprehensible proof system. Thereby, we only need to consider a small number of proof rules. Here, we fill a gap with regard to that approach: we allow to reason about single objects as well as several objects within the same logic in a very intuitive way. [39] introduces object identifiers into the logic. Thereby, it is possible to refer to different objects within one formula. Furthermore, inheritance between objects can be expressed in that approach. For the purposes of this paper, we disregard the aspect of inheritance. It is obvious that inheritance can be added to our logical framework in the same way as it was done in [39]. In that approach there is a certain problem in specifying object communities: due to the chosen global time model one has always to take the behaviour of the whole object community into account when specifying the local behaviour of an object. Here, we remove this burden and allow to specify the behaviour of objects in an as independent as possible way.

We put the main emphasis of this paper on the presentation of a calculus which is to be applicable to different object-oriented specification languages like Albert [12], CMSL [42], LCM [15], OBLOG [40], Mondel [5], or TROLL [25, 28]. The calculus is intended to provide a common basis for verification purposes. Therefore, we have focused our

attention to very elementary concepts. Nevertheless, we have taken care that further concepts can easily be integrated into this basic calculus and, thus, it becomes possible to tailor the calculus to specific needs or specific languages.

The remainder of this paper is organized as follows. First, we sketch our notion of object and object community and introduce a running example. Next, we introduce the syntax of the basic calculus. Then, an appropriate model theory based on linear (Kripke-) structures is outlined which captures hierarchical composition of object communities. Afterwards, we briefly present basic inference rules for the calculus, before we discuss how compositionality of proofs corresponds to composition of object communities and their models. Finally, we conclude by pointing out some obvious and meaningful extensions and by briefly comparing our calculus with related work.

2 Basic Concepts for Object Specification

In the sequel we briefly sketch our view of objects and object communities. In order to provide a common basis for a number of object-oriented specification languages we focus on a small number of essential concepts.

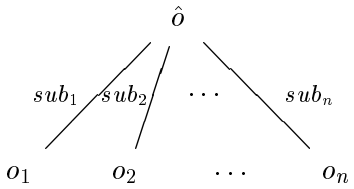
First of all we have to emphasize that we are considering dynamic objects, i.e., objects have an internal state which can be changed by the occurrence of events. Objects have attributes, and the values of these attributes determine object states. The attributes may be data-valued as well as object-valued. Furthermore, multi-valued attributes are possible. By object-valued attributes we understand attributes with references to objects as values.

The dynamics of objects is based on local events. The occurrence of an event in some object causes a state transition to that object, i.e., changes of attribute values (in general). The occurrence of events may be restricted by enabling conditions depending on the object state. Furthermore, we demand that at each moment there may be at most one event occurring in an object. Of course, in an object community there may be several events occurring simultaneously in different objects.

By sub-object relationships objects are organized into object hierarchies. Each object may have several sub-objects, but only one super-object. In consequence, the objects of an object community always form trees induced by sub-object relationships. In general, an object community may consist of several object hierarchies. In this paper we consider object communities consisting of one object hierarchy because the generalisation to

several hierarchies is straightforward. Please note that sub-object relationship is simply a relationship between object instances. There is one major consequence of sub-object relationship: a sub-object existentially depends on its super-object.

The following diagram depicts sub-object relationship between objects. An object $\hat{o}$ has objects $o_1, \ldots, o_n$ as sub-objects. For each sub-object o_i there is a corresponding sub-object name sub_i given in $\hat{o}$:



In such an object hierarchy a super-object can be regarded as a manager object for its sub-objects. In consequence, we do not need to consider the additional concept of a "class", because a super-object acts as a class as well. Moreover, we get class attributes, meta classes (i.e., classes of classes), and further concepts required occasionally for free. In addition, aggregation of objects is also captured by object hierarchies.

Between objects in an object community interaction is possible. The basic mechanism we are considering is event calling, i.e., the occurrence of an event in an object may require the occurrence of another event in another object. These events must occur simultaneously. If the called event is not enabled to occur then the calling event is also not allowed to occur. This mechanism realizes a kind of synchronous interaction.

Before we introduce the running example it is necessary to make a remark to the selection of object concepts. It might appear that we have forgotten to take such an important concept like "inheritance" into consideration. In case we restrict inheritance to be monotonic, there is no problem to express inheritance in the logic we are going to

present here. In case we allow non-monotonic inheritance, i.e. inheritance with overriding, however, we have to extend our logical framework to a non-monotonic one, e.g. to default logic [6, 36].

Example 2.1

Throughout the paper we will take a part of a library information system as our running example. The part considered consists of objects representing authors and one special object for managing these author objects (we call this special object an authorclass object because it represents a class). Typical descriptions of such objects are given in Fig. 1 and 2 using the object specification language TROLL *light* [26].

Each author object has attributes: the attributes `Name` and `Status` for the name of the author and for his/her marital status. The attribute `SoldBooks` allows to store the number of books sold in the year which is given as parameter. Of course, author objects can be created and deleted (events `create` and `destroy`). In addition it is possible to change the name of an author (by the event `changeName`) and to store a number of sold books for some year (`storeSoldBooks`).

The authorclass object (Fig. 2) has an attribute `NumberOfAuthors` which always contains the current number of authors present in the library. Of course, there may be further attributes. We have certain events for creating and deleting the authorclass object (`createClass` and `destroyClass`). Moreover, there are events for adding authors to the library and for removing authors from the library (`addAuthor` and `removeAuthor`). The authors to be managed are represented by sub-objects of this authorclass object. In order to distinguish between the author objects, there is a unique (local) name for each of them. In the example the authors are identified by names like `Authors(12)` or

```
TEMPLATE author
  DATA TYPES  string, date, nat;
  ATTRIBUTES  Name: string;
              Status: string;
              SoldBooks(Year: nat): nat;
  EVENTS      BIRTH create(Name: string, Status: string);
                    changeName(NewName: string);
                    storeSoldBooks(Year: nat, Number: nat);
              DEATH destroy;
  VALUATION   [create(N,S)] Name=N, Status=S;
              [changeName(N)] Name=N;
              [storeSoldBooks(Y,N)] SoldBooks(Y)=N;
  BEHAVIOR    PROCESS authorlife1 =
                 ( storeSoldBooks -> authorlife1 |
                   changeName -> authorlife2 |
                   destroy -> POSTMORTEM );
              PROCESS authorlife2 =
                 ( storeSoldBooks -> authorlife2 |
                   destroy -> POSTMORTEM );
              ( create -> authorlife1 );
END TEMPLATE;
```

Figure 1: TROLL *light* specification of author objects.

```
TEMPLATE authorclass
  DATA TYPES   string, date, nat;
  TEMPLATES    author;
  SUBOBJECTS   Authors(No: nat): author;
  ATTRIBUTES   NumberOfAuthors: nat;
  EVENTS       BIRTH createClass;
                     addObject(No: nat, Name: string, Status: string);
                     removeObject(No: nat);
               DEATH destroyClass;
  VALUATION    [createClass] NumberOfAuthors=0;
               [addObject] NumberOfAuthors=NumberOfAuthors+1;
               [removeObject] NumberOfAuthors=NumberOfAuthors-1;
  INTERACTION  addObject(No,N,S) >> Authors(No).create(N,S);
               removeObject(No) >> Authors(No).destroy;
  BEHAVIOR     PROCESS authorclasslife =
                 ( addObject, removeObject -> authorclasslife |
                   destroyClass -> POSTMORTEM );
               ( createClass -> authorclasslife );
END TEMPLATE;
```

Figure 2: TROLL *light* specification of authorclass objects.

`Authors(42)`.

In the specification of these objects we can additionally restrict the occurrences of events by giving conditions telling in which states of an object certain events may occur and by describing life cycles an object must go through. For instance, the behaviour of author objects is restricted in such a way that a `changeName` event may only occur once in the life of an object. A typical interaction between objects is, for example, that the occurrence of an `addAuthor` event in the authorclass object causes the simultaneous occurrence of the `create` event of a sub-object.

In Fig. 3 an object community state of the considered part of the library information system is shown. ⋈

For specifying objects we have developed the specification language TROLL *light* [11, 22, 26] in which we realized the concepts sketched above. This language is a dialect of TROLL [37, 28] and is designed for investigating validation and verification of object specifications. TROLL *light* (as well as TROLL) is based on a general semantic framework for specifying object communities (see for instance [13]).

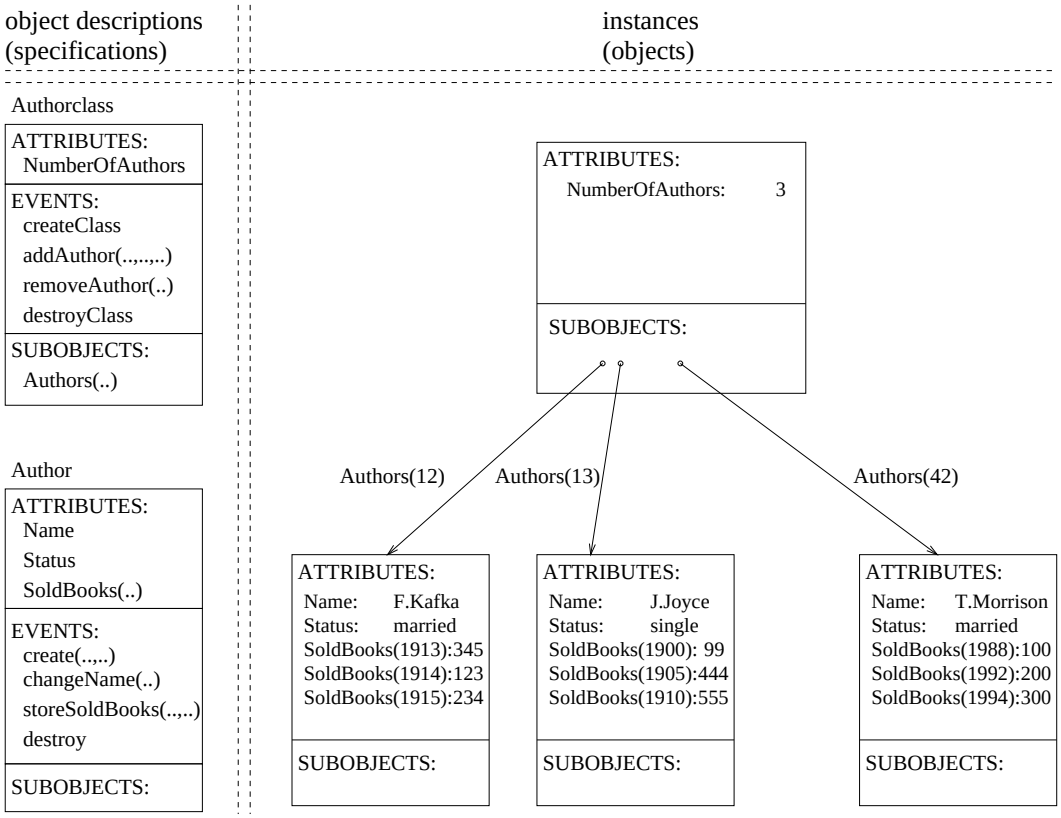


Figure 3: An object community state.

3 Syntax of the Calculus

In this section we define the syntactical structure of the basic calculus. First, we introduce data signatures and community signatures. Then, we deal with the construction of terms, propositions, and formulas.

Definition 3.1 (Data Signature)

A data signature $\Sigma_D = (\mathrm{S}_D, \Omega_D, \Pi_D)$ consists of

- a set S_D of data sorts,
- a family of sets of operation symbols $\Omega_D = \langle \Omega_{D,wd} \rangle_{wd \in (\mathrm{S}_D^* \times \mathrm{S}_D)}$, and
- a family of sets of predicate symbols $\Pi_D = \langle \Pi_{D,w} \rangle_{w \in \mathrm{S}_D^*}$.

$\triangleleft$

Example 3.1

In our running example (see Ex. 2.1) we use the following (fragment of a) data signature $\Sigma_D = (\mathrm{S}_D, \Omega_D, \Pi_D)$:

$$
\begin{array}{lcl}
\mathrm{S}_D & = & \{\texttt{string}, \texttt{nat}, \ldots\} \\
\vdots & & \\
\Omega_{D,\texttt{string}} & = & \{\texttt{bottom}_{\texttt{string}}, \ldots\} \\
\Omega_{D,\texttt{nat}} & = & \{\texttt{bottom}_{\texttt{nat}}, \ldots\} \\
\vdots & & \\
\Omega_{D,\texttt{string string string}} & = & \{\texttt{concat}, \ldots\} \\
\Omega_{D,\texttt{nat nat nat}} & = & \{+, -, *, \ldots\} \\
\vdots & & \\
\Pi_{D,\texttt{string string}} & = & \{\texttt{substr}, \ldots\} \\
\Pi_{D,\texttt{nat nat}} & = & \{>, <, >=, <=, \ldots\} \\
\vdots & &
\end{array}
$$

It should be noted that we assume a constant $\texttt{bottom}_s$ for each data sort $s \in \mathrm{S}_D$. This constant represents the undefined value. ⋈

Obviously, there are several ways of handling undefinedness. One possibility is to assume all operations to be strict, i.e., an undefined value as argument of an operation causes an undefined value as result of the operation.

Definition 3.2 (Community Signature)

A community signature $\Sigma_C = (\Sigma_D,\ \mathcal{SB}_C,\ \mathtt{OBG}_C,\ \mathtt{ATT}_C,\ \mathtt{EVT}_C)$ consists of

- a data signature $\Sigma_D = (\mathrm{S}_D, \Omega_D, \Pi_D)$,
- a tuple $\mathcal{SB}_C = (\mathrm{S_O}, \mathrm{S_E}, \mathtt{evt})$ with a set $\mathrm{S_O}$ of object sorts, a set $\mathrm{S_E}$ of (local) event sorts, and a (global) event sort `evt`,

  (in the sequel we use the following abbreviation: $\mathrm{S} := \mathrm{S}_D \cup \mathrm{S_O}$)
- a family of sets of object name generators $\mathtt{OBG}_C = \langle \mathtt{OBG}_{C,\,w,o'} \rangle_{w,o' \in (\mathrm{S}^* \to \mathrm{S_O})}$,
- a family of sets of attribute symbols $\mathtt{ATT}_C = \langle \mathtt{ATT}_{C,ows} \rangle_{ows \in (\mathrm{S_O} \times \mathrm{S}^* \times \mathrm{S})}$, and
- a family of sets of event symbols $\mathtt{EVT}_C = \langle \mathtt{EVT}_{C,\,w,o_\mathtt{evt}} \rangle_{w,o_\mathtt{evt} \in (\mathrm{S}^* \to \mathrm{S_E})}$.

The sets $\mathrm{S_O}$, $\mathrm{S_E}$, and S_D are pairwise disjoint. Furthermore, there is a bijective mapping between $\mathrm{S_O}$ and $\mathrm{S_E}$, such that there is a unique sort $o_\mathtt{evt} \in \mathrm{S_E}$ for each $o \in \mathrm{S_O}$.

Finally, we define the set $\hat{\mathrm{S}}$ of all sorts: $\hat{\mathrm{S}} := \mathrm{S} \cup \{\mathtt{evt}\} \cup \mathrm{S_E}$. ◁

Here, we have defined the notion of community signature such that it is independent of any concrete object specification language. In [7] a special version of this calculus is presented which is tailored for the specification language TROLL *light*.

By means of object name generators we can construct terms for identifying objects. We will later see that a certain restriction is necessary for unique identification.

The differentiation between local event sorts ($o_\mathtt{evt} \in \mathrm{S_E}$) and a global event sort `evt` is necessary and reasonable because it permits more flexibility in constructing terms and formulating axioms as we will see below.

Example 3.2

In explanation of this definition we consider the community signature for the running example. In Ex. 2.1 we have considered objects of kind "authorclass" having objects of kind "author" as sub-objects. The corresponding community signature is $\Sigma_{\texttt{Authorclass}}$ (which we sometimes abbreviate to $\Sigma_{\texttt{AC}}$). We assume a data signature Σ_D to be given (see Example 3.1).

$\Sigma_{\texttt{Authorclass}}$ must include two object sorts, i.e., `author` and `authorclass`, and for each of these object sorts a corresponding local event sort (`author_evt` and `authorclass_evt`). Of course, there is also the global event sort `evt`. The attribute symbols and event symbols belonging to these two kinds of objects are listed in Fig. 4. For attribute symbols as well as object name generators the first parameter is used to denote the object of which we are interested to know the corresponding attribute value or its subobjects. Please note that the example is given for the language TROLL *light*. For other object specification languages the identification of object may work in different way. $\bowtie$

From now on we omit the index C in $\texttt{OBG}_C$, $\texttt{ATT}_C$, and $\texttt{EVT}_C$ and the index D in Ω_D and Π_D provided that no confusion can arise.

After having introduced data signatures and community signatures we can define terms, propositions, and formulas over community signatures.

$$
\begin{aligned}
\mathcal{SB}_{\texttt{AC}} &= (\ \{\texttt{authorclass}, \texttt{author}\},\\
&\quad\ \{\texttt{authorclass_evt}, \texttt{author_evt}\},\\
&\quad\ \texttt{evt}\ )\\
\\
\texttt{OBG}_{\texttt{AC, authorclass nat, author}} &= \{\ \mathit{Authors}\ \}\\
\\
\texttt{ATT}_{\texttt{AC, authorclass nat}} &= \{\ \mathit{NumberOfAuthors}\ \}\\
\texttt{ATT}_{\texttt{AC, author string}} &= \{\ \mathit{Name},\ \mathit{Status}\ \}\\
\texttt{ATT}_{\texttt{AC, author nat nat}} &= \{\ \mathit{SoldBooks}\ \}\\
\\
\texttt{EVT}_{\texttt{AC, authorclass_evt}} &= \{\ \mathit{createClass},\ \mathit{destroyClass}\ \}\\
\texttt{EVT}_{\texttt{AC, nat, authorclass_evt}} &= \{\ \mathit{removeAuthor}\ \}\\
\texttt{EVT}_{\texttt{AC, nat string string, authorclass_evt}} &= \{\ \mathit{addAuthor}\ \}\\
\texttt{EVT}_{\texttt{AC, string string, author_evt}} &= \{\ \mathit{create}\ \}\\
\texttt{EVT}_{\texttt{AC, string, author_evt}} &= \{\ \mathit{changeName}\ \}\\
\texttt{EVT}_{\texttt{AC, nat nat, author_evt}} &= \{\ \mathit{storeSoldBooks}\ \}\\
\texttt{EVT}_{\texttt{AC, author_evt}} &= \{\ \mathit{destroy}\ \}
\end{aligned}
$$

Figure 4: Components of the community signature $\Sigma_{\texttt{Authorclass}}$.

Definition 3.3 (Term)

Given a community signature $\Sigma_C = (\Sigma_D, \mathcal{SB}, \mathtt{OBG}, \mathtt{ATT}, \mathtt{EVT})$ and an $\hat{\mathrm{S}}$-indexed family of sets of variables, $X = \langle X_s \rangle_{s \in \hat{\mathrm{S}}}$. The minimal $\hat{\mathrm{S}}$-indexed family $\mathtt{TERM}(X) = \langle \mathtt{TERM}_s(X) \rangle_{s \in \hat{\mathrm{S}}}$ of sets of terms over Σ_C and X is inductively defined by:

- if $x \in X_s$ $(s \in \hat{\mathrm{S}})$, then $x \in \mathtt{TERM}_s(X)$,
- if $\omega \in \Omega_{d_1 \ldots d_n, d}$ and $t_i \in \mathtt{TERM}_{d_i}(X)$, then $\omega(t_1, \ldots, t_n) \in \mathtt{TERM}_d(X)$,
- if $\mathrm{og} \in \mathtt{OBG}_{s_1 \ldots s_n, o}$ and $t_i \in \mathtt{TERM}_{s_i}(X)$, then $\mathrm{og}(t_1, \ldots, t_n) \in \mathtt{TERM}_o(X)$,
- if $\mathrm{e} \in \mathtt{EVT}_{s_1 \ldots s_n, o_\mathtt{evt}}$ and $t_i \in \mathtt{TERM}_{s_i}(X)$, then $\mathrm{e}(t_1, \ldots, t_n) \in \mathtt{TERM}_{o_\mathtt{evt}}(X)$, and
- if $t_o \in \mathtt{TERM}_o(X)$ and $t_{o_\mathtt{evt}} \in \mathtt{TERM}_{o_\mathtt{evt}}(X)$, then $t_o.t_{o_\mathtt{evt}} \in \mathtt{TERM}_{\mathtt{evt}}(X)$.

$\lhd$

For building global event terms we have introduced the constructor "." (which we call "event constructor"). Thereby, we can put together a term of an object sort $o \in \mathrm{S_O}$ and a term of the corresponding local event sort $o_\mathtt{evt} \in \mathrm{S_E}$ to form a term of the global event sort $\mathtt{evt}$.

Example 3.3

Taking the data signature Σ_D given in Example 3.1 and the community signature $\Sigma_{\mathtt{Authorclass}}$ given in Example 3.2 we can construct the following terms. For this we assume a family X of sets of variables with $X_{\mathtt{string}} = \{S, S'\}$, $X_{\mathtt{nat}} = \{N\}$, $X_{\mathtt{authorclass}} = \{AC\}$, $X_{\mathtt{author}} = \{A\}$, and $X_{\mathtt{author_evt}} = \{EV\}$:

AC $\in \mathtt{TERM}_{\mathtt{authorclass}}(X)$

$A, \quad Authors(AC, 42)$ $\in \mathtt{TERM}_{\mathtt{author}}(X)$

$createClass, \quad removeAuthor(N)$ $\in \mathtt{TERM}_{\mathtt{authorclass_evt}}(X)$

$create(S, \text{"}\mathtt{single}\text{"}), \quad changeName(\text{"}\mathtt{E.Kishon}\text{"})$ $\in \mathtt{TERM}_{\mathtt{author_evt}}(X)$

$AC.removeAuthor(17),$

$Authors(AC, 42).create(S, S'), \quad A.EV$ $\in \mathtt{TERM}_{\mathtt{evt}}(X)$

These terms already show why we have explicitly distinguished between local and global event sorts. Without local event sorts we could not build terms like $A.EV$ and $AC.removeAuthor(17)$. But these terms are necessary to talk about the effects of concrete local events (e.g., $removeAuthor(17)$) to arbitrary objects of the corresponding object sort. Furthermore, a term like $myauthor.EV$ (where $myauthor$ identifies a concrete object) could be useful for expressing that an arbitrary event can happen to the specific object $myauthor$. ⋈

Definition 3.4 (Proposition)

Given a community signature Σ_C and a $\hat{\mathrm{S}}$-indexed family of sets of variables $X = \langle X_s \rangle_{s \in \hat{\mathrm{S}}}$. The set of propositions $\mathtt{PROP}(X)$ over Σ_C and X is the minimal set satisfying:

- if $e \in \mathtt{TERM}_{\mathtt{evt}}(X)$, then $\mathsf{enable}(e) \in \mathtt{PROP}(X)$ and $\mathsf{occur}(e) \in \mathtt{PROP}(X)$,
- if $\pi \in \Pi_{d_1 \ldots d_n}$ and $t_i \in \mathtt{TERM}_{d_i}(X)$, then $\pi(t_1, \ldots, t_n) \in \mathtt{PROP}(X)$,
- if $\mathrm{a} \in \mathtt{ATT}_{os_1 \ldots s_n s}$, $t_o \in \mathtt{TERM}_o(X)$, $t_i \in \mathtt{TERM}_{s_i}(X)$, and $t \in \mathtt{TERM}_s(X)$, then $\mathrm{a}(t_o, t_1, \ldots, t_n, t) \in \mathtt{PROP}(X)$,
- if $t_1, t_2 \in \mathtt{TERM}_s(X)$ $(s \in \hat{\mathrm{S}})$, then $(t_1 = t_2) \in \mathtt{PROP}(X)$,

- if $P \in \mathtt{PROP}(X)$, then $\neg P \in \mathtt{PROP}(X)$, and
- if $e \in \mathtt{TERM}_{\mathtt{evt}}(X)$ and $P \in \mathtt{PROP}(X)$, then $[e]P \in \mathtt{PROP}(X)$.

$\lhd$

For events there are state-dependent predicates (enable and occur) for expressing whether an event may occur (i.e., is enabled) and whether an event is really occurring. Further predicate symbols for state-dependent predicates are attribute symbols. This is due to the fact that the value of an attribute may change in the course of time. Propositions of the form $[e]P$ are called positional propositions: $[e]P$ is used for expressing that "P holds after the occurrence of e", or, more precisely, that "if e occurs next then P holds in the moment after the occurrence of e".

Example 3.4

Using the terms given in Example 3.3 we can build the following propositions as propositions over the community signature $\Sigma_{\mathtt{Authorclass}}$ (cf. Example 3.2):

$$\mathsf{enable}(AC.removeAuthor(17)),$$

$$\mathsf{occur}(Authors(AC, 42).create(S, S')),$$

$$Name(Authors(AC, 42), \text{"}\mathtt{E.Kishon}\text{"}),$$

$$[A.create(S, S')]\neg\,\mathsf{enable}(A.create(S, S')),$$

$$[AC.removeAuthor(17)]NumberOfAuthors(AC, 6).$$

$\bowtie$

Definition 3.5 (Formula)

Given propositions $P_1, \ldots, P_n, Q_1, \ldots, Q_m \in \texttt{PROP}(X)$ over a community signature Σ_C. Then

$$\{P_1, \ldots, P_n\} \to \{Q_1, \ldots, Q_m\}$$

is a formula over Σ_C. The set of all formulas which can be constructed over a community signature Σ_C and an $\hat{\text{S}}$-indexed family of sets of variables $X = \langle X_s \rangle_{s \in \hat{\text{S}}}$ is denoted by $\texttt{FORM}(X)$.

In general we omit braces in formulas and write instead $P_1, \ldots, P_n \to Q_1, \ldots, Q_m$. $\lhd$

Example 3.5

Continuing Examples 3.1–3.4 we present two formulas over the community signature $\Sigma_{\texttt{Authorclass}}$:

$$\to [A.changeName(S)]Name(A, S)$$

$$NumberOfAuthors(AC, 7) \to [AC.removeAuthor(17)]NumberOfAuthors(AC, 6)$$

$\bowtie$

Thus, we have provided the syntactical basis of the calculus. The next section presents a model-theoretic semantics.

4 Model-Theoretic Semantics

The model-theoretic semantics of the basic calculus is founded on a variation of Kripke-structures. Before we define these interpretation structures we introduce state-independent interpretation of sorts and operations. Then we proceed with term evaluation. Afterwards we can give the definition of interpretation structures. After that we continue by dealing with satisfaction of propositions and truth of formulas. Finally, we introduce the notions "semantic entailment" and "semantic closure".

Definition 4.1 (State-independent Interpretation)

Interpretations of sorts $s \in \hat{\mathrm{S}}$, of operation symbols $\omega \in \Omega_{d_1 \ldots d_n, d}$, of predicate symbols $p \in \Pi_{d_1 \ldots d_n}$, of object name generators $\mathrm{og} \in \mathtt{OBG}_{s_1 \ldots s_n, o}$, and of event symbols $\mathrm{e} \in \mathtt{EVT}_{s_1 \ldots s_n, o_\mathtt{evt}}$ are denoted by $I(s)$, $I(\omega)$, $I(p)$, $I(\mathrm{og})$, and $I(\mathrm{e})$, respectively:

- $I(s)$ denotes a carrier set for $s \in \hat{\mathrm{S}}$,
- $I(\omega) : I(d_1) \times \ldots \times I(d_n) \to I(d)$ for $\omega \in \Omega_{d_1 \ldots d_n, d}$,
- $I(p) \subseteq I(d_1) \times \ldots \times I(d_n)$ for $p \in \Pi_{d_1 \ldots d_n}$,
- $I(\mathrm{og}) : I(s_1) \times \ldots \times I(s_n) \to I(o)$ for $\mathrm{og} \in \mathtt{OBG}_{s_1 \ldots s_n, o}$, and
- $I(\mathrm{e}) : I(s_1) \times \ldots \times I(s_n) \to I(o_\mathtt{evt})$ for $\mathrm{e} \in \mathtt{EVT}_{s_1 \ldots s_n, o_\mathtt{evt}}$.

Object sorts $o \in \mathrm{S_O}$ and local event sorts $o_\mathtt{evt} \in \mathrm{S_E}$ are interpreted by the term algebras $\mathtt{TERM}_o$ and $\mathtt{TERM}_{o_\mathtt{evt}}$, respectively — modulo equivalence of terms induced by $A(\Sigma_D)$.

The carrier set $I(\texttt{evt})$ (the interpretation of sort `evt`) can be constructed by the disjoint union of all Cartesian products which are obtained from the carrier sets of object sorts and their corresponding local event sorts:

$$I(\texttt{evt}) = \biguplus_{o \in \mathrm{S_O}} (I(o) \times I(o_\texttt{evt}))$$

The event constructor $_._ : \biguplus_{o \in \mathrm{S_O}} (o \times o_\texttt{evt}) \to \texttt{evt}$ is injective and surjective. $\lhd$

For the data signature we may assume an initial algebra $A(\Sigma_D) = (A(\mathrm{S}_D), A(\Omega_D), A(\Pi_D)) \in \mathrm{ALG}_{\Sigma_D}$ as given. Thereby, $I(s)$ for each $s \in \mathrm{S}_D$, $I(\omega)$, for each $\omega \in \Omega_D$, and $I(p)$ for each $p \in \Pi_D$ is fixed. For the interpretation of undefined values we have several possibilities. As already mentioned w.r.t. to Example 3.1 we could for instance assume all operations to be strict, i.e., undefined values are propagated by all operations.

For the interpretation of sort `evt`, the disjoint union of all Cartesian products which are obtained from the carrier sets of object sorts and their corresponding local event sorts seems to be the most general case. In several approaches events of different objects are assumed to be one event (e.g., an interaction or communication event). Our approach allows to explicitly state that these events must always occur simultaneously. Thereby, we can at least express the same facts.

In the following we use the notation $\underline{s_i}$ to denote an element of $I(s_i)$.

Definition 4.2 (Assignment)

An assignment $\mathcal{A}$ for a given family X of sets of variables assigns a value $\mathcal{A}(x) \in I(s)$ to each variable $x \in X_s$. $\lhd$

Such an assignment is necessary for interpreting terms containing variables.

Definition 4.3 (Term Evaluation)

The value $[\![t]\!]^{\mathcal{A}}$ of a term t by employing an assignment $\mathcal{A}$ is inductively defined as follows:

- for $x \in X_s$ we have $[\![x]\!]^{\mathcal{A}} = \mathcal{A}(x)$,
- for $\omega \in \Omega_{d_1 \ldots d_n, d}$ and $t_i \in \mathtt{TERM}_{d_i}(X)$ we have
  $[\![\omega(t_1, \ldots, t_n)]\!]^{\mathcal{A}} = I(\omega)([\![t_1]\!]^{\mathcal{A}}, \ldots, [\![t_n]\!]^{\mathcal{A}})$,
- for $\mathrm{og} \in \mathtt{OBG}_{s_1 \ldots s_n, o}$ and $t_i \in \mathtt{TERM}_{s_i}(X)$ we have
  $[\![\mathrm{og}(t_1, \ldots, t_n)]\!]^{\mathcal{A}} = I(\mathrm{og})([\![t_1]\!]^{\mathcal{A}}, \ldots, [\![t_n]\!]^{\mathcal{A}})$,
- for $\mathrm{e} \in \mathtt{EVT}_{s_1 \ldots s_n, o_\mathtt{evt}}$ and $t_i \in \mathtt{TERM}_{s_i}(X)$ we have
  $[\![\mathrm{e}(t_1, \ldots, t_n)]\!]^{\mathcal{A}} = I(\mathrm{e})([\![t_1]\!]^{\mathcal{A}}, \ldots, [\![t_n]\!]^{\mathcal{A}})$, and
- for $t_o \in \mathtt{TERM}_o(X)$ and $t_{o_\mathtt{evt}} \in \mathtt{TERM}_{o_\mathtt{evt}}(X)$ we have
  $[\![t_o.t_{o_\mathtt{evt}}]\!]^{\mathcal{A}} = I(.)([\![t_o]\!]^{\mathcal{A}}, [\![t_{o_\mathtt{evt}}]\!]^{\mathcal{A}})$.

$\lhd$

The term evaluation is defined in such a way that all terms are interpreted independent of time and, hence, independent of the state of the object community. This is in contrast to some other approaches like [17, 19] where the proof system is much more complex due to the additional manipulation of time-varying terms.

Definition 4.4 (State Proposition)

The set of state propositions SPROP is defined by:

$$\begin{aligned}
&\texttt{SPROP} := \\
&\quad \{\mathrm{a}(\underline{o}, \underline{s_1}, \ldots, \underline{s_n}, \underline{s}) \mid \mathrm{a} \in \texttt{ATT}_{os_1 \ldots s_n s}, \underline{o} \in I(o), \underline{s_1} \in I(s_1), \ldots, \underline{s_n} \in I(s_n), \underline{s} \in I(s)\} \\
&\cup \{\mathsf{enable}(\underline{o.e}) \mid \underline{o.e} \in I(\texttt{evt})\} \\
&\cup \{\mathsf{occur}(\underline{o.e}) \mid \underline{o.e} \in I(\texttt{evt})\}
\end{aligned}$$

◁

Here, a remark concerning the difference between PROP and SPROP is necessary. The set PROP provides all propositions which may be constructed *syntactically* (i.e., these propositions are uninterpreted and may contain variables). In contrast the set of state propositions SPROP includes all *interpreted* propositions which are predicate expressions built with state-dependent predicate symbols and interpreted value. In the following definition we use the set SPROP as domain for assigning each state of an object community a subset of SPROP as interpretation.

Definition 4.5 (Interpretation Structure)

Given a community signature Σ_C and for Σ_C a state-independent interpretation, an interpretation structure $\mathcal{IS} = (\mathcal{W}, \rho)$ for Σ_C consists of

- an infinite, totally ordered set $\mathcal{W} = \{w_0, w_1, \ldots\}$ of (community) states where the indices corresponds to the chronological order and
- a function $\rho : \mathcal{W} \to 2^{\texttt{SPROP}}$ assigning a set of state propositions to each community state.

The function ρ has to fulfill the following condition:

$$\mathsf{occur}(\underline{o}.\underline{e}) \in \rho(w) \ \wedge \ \mathsf{occur}(\underline{o}.\underline{e'}) \in \rho(w) \quad \Longrightarrow \quad \underline{e} = \underline{e'}$$

As notational simplification we define a function $\mathcal{O}cc : \mathcal{W} \to 2^{I(\mathtt{evt})}$ with $\mathcal{O}cc(w) := \{ \underline{o.e} \mid \mathsf{occur}(\underline{o.e}) \in \rho(w) \}$ for each $w \in \mathcal{W}$. $\lhd$

By this definition we restrict our considerations to linear interpretation structures by requiring a total order on $\mathcal{W}$. This is a reasonable restriction because we strive for a basic calculus being as simple as possible and, therefore, we do not include operators requiring a branching time interpretation. Although we require $\mathcal{W}$ to be an infinite set, finite life cycles are not excluded for objects. This is due to the fact that there may be states in which no event occurs. We also allow "trivial" interpretation structures without any event occurrence. The condition imposed on ρ reflects our assumption about objects that there must not happen more than one event at a moment (or in a state). Beside this property we do not have fixed other properties being inherent in our object model. For instance, we could have added the requirement that an occurring event must be enabled at that moment. However, this can easily be expressed by an axiom which has to be added to each specification (cf. Example 5.1). In case we bring this property directly into the definition of interpretation structures for our logic we would lose flexibility. Because our logical framework is intended to be adaptable to different object specification approaches we need this kind of flexibility.

In consequence, we have not fixed a relation between enabling and occurrence of events in interpretation structures which are linear time structures. Moreover, the property that an occurring event must be enabled can even be expressed without using an explicit

enabling predicate. However, if we would like to use branching time as underlying model we have to modify the definition of interpretation structures and we need an explicit notion of "enabling".

For illustrating interpretation structures we use diagrams in which the states are connected by edges representing the successor relationship (as known, e.g., from state transition diagrams). We label each edge starting at state w_i by its set of event occurrences $\mathcal{O}cc(w_i)$:

$$\mathcal{IS}: \quad w_0 \xrightarrow{\mathcal{O}cc(w_0)} w_1 \xrightarrow{\mathcal{O}cc(w_1)} w_2 \xrightarrow{\mathcal{O}cc(w_2)} w_3 \xrightarrow{\mathcal{O}cc(w_3)} w_4 \xrightarrow{\mathcal{O}cc(w_4)} w_5 \xrightarrow{\mathcal{O}cc(w_5)} \cdots$$

Example 4.1

As an example for an interpretation structure we consider an interpretation structure $\mathcal{IS}_{\texttt{Authorclass}} = (\mathcal{W}, \rho)$ for the community signature $\Sigma_{\texttt{Authorclass}}$ given in Example 3.2. In Fig. 5 we partially describe $\rho : \mathcal{W} \to 2^{\texttt{SPROP}}$ assuming an arbitrary but fixed element $\underline{o} \in I(\texttt{authorclass})$. Furthermore, we use the following abbreviations:

$$\underline{a12} := I(Authors)(\underline{o}, I(12))$$

$$\underline{a13} := I(Authors)(\underline{o}, I(13))$$

$\bowtie$

After this introduction of interpretation structures we can define satisfaction of propositions.

$$
\begin{array}{lcl}
\rho(w_0) & \supseteq & \{\ \mathtt{occur}(\underline{o}.I(createClass))\ \} \\
\rho(w_1) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(0))), \\
 & & \quad \mathtt{occur}(\underline{o}.I(addAuthor)(I(12), I(\text{"}\mathtt{F.Kafka}\text{"}), I(\text{"}\mathtt{married}\text{"}))), \\
 & & \quad \mathtt{occur}(\underline{a12}.I(create)(I(\text{"}\mathtt{F.Kafka}\text{"}), I(\text{"}\mathtt{married}\text{"})))\ \} \\
\rho(w_2) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(1))), \\
 & & \quad Name(\underline{a12}, I(\text{"}\mathtt{F.Kafka}\text{"})),\ Status(\underline{a12}, I(\text{"}\mathtt{married}\text{"})), \\
 & & \quad \mathtt{occur}(\underline{o}.I(addAuthor)(I(13), I(\text{"}\mathtt{J.Joyce}\text{"}), I(\text{"}\mathtt{single}\text{"}))), \\
 & & \quad \mathtt{occur}(\underline{a13}.I(create)(I(\text{"}\mathtt{J.Joyce}\text{"}), I(\text{"}\mathtt{single}\text{"})))\ \} \\
\rho(w_3) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(2))), \\
 & & \quad Name(\underline{a12}, I(\text{"}\mathtt{F.Kafka}\text{"})),\ Status(\underline{a12}, I(\text{"}\mathtt{married}\text{"})), \\
 & & \quad Name(\underline{a13}, I(\text{"}\mathtt{J.Joyce}\text{"})),\ Status(\underline{a13}, I(\text{"}\mathtt{single}\text{"})), \\
 & & \quad \mathtt{occur}(\underline{a12}.I(storeSoldBooks)(I(1993), I(42)))\ \} \\
\rho(w_4) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(2))), \\
 & & \quad Name(\underline{a12}, I(\text{"}\mathtt{F.Kafka}\text{"})),\ Status(\underline{a12}, I(\text{"}\mathtt{married}\text{"})), \\
 & & \quad Name(\underline{a13}, I(\text{"}\mathtt{J.Joyce}\text{"})),\ Status(\underline{a13}, I(\text{"}\mathtt{single}\text{"})), \\
 & & \quad SoldBooks(\underline{a12}, I(1993), I(42)), \\
 & & \quad \mathtt{occur}(\underline{a13}.I(storeSoldBooks)(I(1993), I(555))), \\
 & & \quad \mathtt{occur}(\underline{a12}.I(changeName)(I(\text{"}\mathtt{E.Kishon}\text{"})))\ \} \\
\rho(w_5) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(2))), \\
 & & \quad Name(\underline{a12}, I(\text{"}\mathtt{E.Kishon}\text{"})),\ Status(\underline{a12}, I(\text{"}\mathtt{married}\text{"})), \\
 & & \quad Name(\underline{a13}, I(\text{"}\mathtt{J.Joyce}\text{"})),\ Status(\underline{a13}, I(\text{"}\mathtt{single}\text{"})), \\
 & & \quad \mathtt{occur}(\underline{a13}.I(storeSoldBooks)(I(1993), I(555))), \\
 & & \quad \mathtt{occur}(\underline{a12}.I(changeName)(I(\text{"}\mathtt{E.Kishon}\text{"})))\ \} \\
\vdots & & \vdots \\
\rho(w_{1342}) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(123))), \\
 & & \quad Name(\underline{a12}, I(\text{"}\mathtt{E.Kishon}\text{"})),\ Status(\underline{a12}, I(\text{"}\mathtt{married}\text{"})), \\
 & & \quad Name(\underline{a13}, I(\text{"}\mathtt{J.Joyce}\text{"})),\ Status(\underline{a13}, I(\text{"}\mathtt{single}\text{"})), \\
 & & \quad \mathtt{occur}(\underline{o}.I(removeAuthor)(I(13))), \\
 & & \quad \mathtt{occur}(\underline{a13}.I(destroy))\ \} \\
\rho(w_{1343}) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(122))), \\
 & & \quad Name(\underline{a12}, I(\text{"}\mathtt{E.Kishon}\text{"})),\ Status(\underline{a12}, I(\text{"}\mathtt{married}\text{"})), \\
 & & \quad \mathtt{occur}(\underline{a13}.I(storeSoldBooks)(I(1994), I(666)))\ \} \\
\vdots & & \vdots
\end{array}
$$

Figure 5: Fragment of a concrete interpretation structure $\mathcal{IS}_{\mathtt{Authorclass}}$.

Definition 4.6 (Satisfaction of Propositions)

The satisfaction of propositions $P \in \mathtt{PROP}(X)$ over a community signature Σ_C by an interpretation structure $\mathcal{IS}$ for Σ_C and an assignment $\mathcal{A}$ in a state $w_i \in \mathcal{W}$ (written $\langle \mathcal{IS}, \mathcal{A}, w_i \rangle \models P$) is inductively defined as follows:

- for $p \in \{\mathsf{occur}, \mathsf{enable}\} \cup \bigcup_{ows \in \mathrm{S_O} \times \mathrm{S}^* \times \mathrm{S}} \mathtt{ATT}_{o\,s_1 \ldots s_n\,s}$:

  $\langle \mathcal{IS}, \mathcal{A}, w_i \rangle \models p(t_1, \ldots, t_n)$ iff $p([\![t_1]\!]^{\mathcal{A}}, \ldots, [\![t_n]\!]^{\mathcal{A}}) \in \rho(w_i)$,

- for $p \in \Pi_{d_1 \ldots d_n}$: $\langle \mathcal{IS}, \mathcal{A}, w_i \rangle \models p(t_1, \ldots, t_n)$ iff $([\![t_1]\!]^{\mathcal{A}}, \ldots, [\![t_n]\!]^{\mathcal{A}}) \subseteq I(p)$,

- $\langle \mathcal{IS}, \mathcal{A}, w_i \rangle \models \neg P$ iff not $\langle \mathcal{IS}, \mathcal{A}, w_i \rangle \models P$,

- $\langle \mathcal{IS}, \mathcal{A}, w_i \rangle \models [o.e]P$ iff

  - $[\![o]\!]^{\mathcal{A}}.[\![e]\!]^{\mathcal{A}} \in \mathcal{O}cc(w_i) \wedge \langle \mathcal{IS}, \mathcal{A}, w_{i+1} \rangle \models P$, or
  - $[\![o]\!]^{\mathcal{A}}.\underline{e'} \in \mathcal{O}cc(w_i) \wedge \underline{e'} \neq [\![e]\!]^{\mathcal{A}}$, or
  - $(\neg \exists \underline{o'}.\underline{e'} \in \mathcal{O}cc(w_i) : \underline{o'} = [\![o]\!]^{\mathcal{A}}) \wedge \langle \mathcal{IS}, \mathcal{A}, w_{i+1} \rangle \models [o.e]P$.

$\lhd$

The last item of the definition deals with the satisfaction of propositions with positional operator. Given an assignment $\mathcal{A}$ a proposition $[o.e]P$ is satisfied in a state $w_i \in \mathcal{W}$ iff we can conclude from the fact, that the event $[\![e]\!]^{\mathcal{A}}$ occurs next in the object denoted by $[\![o]\!]^{\mathcal{A}}$, that P holds in the subsequent state (after the occurrence of $[\![e]\!]^{\mathcal{A}}$). For clarification we consider the three possible cases in detail:

- The first case is that the event $[\![e]\!]^{\mathcal{A}}$ occurs directly in state w in the object denoted by $[\![o]\!]^{\mathcal{A}}$:

$$\mathcal{IS}: \quad \cdots \xrightarrow{\cdots} w \xrightarrow{\ldots, [\![o]\!]^{\mathcal{A}}.[\![e]\!]^{\mathcal{A}}, \ldots} w' \xrightarrow{\cdots} \cdots$$

Then, for $[o.e]P$ to be satisfied in w, P must be satisfied in w'.

- The second case is given by the situation that in state w an event $\underline{e'}$ different from $[\![e]\!]^{\mathcal{A}}$ occurs in object $[\![o]\!]^{\mathcal{A}}$:

$$\mathcal{IS}: \quad \cdots \xrightarrow{\cdots} w \xrightarrow{\ldots, [\![o]\!]^{\mathcal{A}}.\underline{e'}, \ldots} w' \xrightarrow{\cdots} \cdots$$

  In that case we say that $[o.e]P$ holds in w without any further condition. The reason for this is that we can consider $[o.e]P$ as an implicit implication: if $o.e$ occurs then P holds afterwards.

- The remaining case is that in state w no event is occurring in object $[\![o]\!]^{\mathcal{A}}$ at all:

$$\mathcal{IS}: \quad \cdots \xrightarrow{\cdots} w \xrightarrow{\mathcal{O}cc(w)} w' \xrightarrow{\cdots} \cdots$$

  where $\neg\exists \underline{e} : [\![o]\!]^{\mathcal{A}}.\underline{e} \in \mathcal{O}cc(w)$.

  Here, we have an idle-transition for object $[\![o]\!]^{\mathcal{A}}$ from w to w'. This may be due to the occurrence of events in other objects which do not effect the object $[\![o]\!]^{\mathcal{A}}$. From the local point of view of the object $[\![o]\!]^{\mathcal{A}}$ the state w' cannot be distinguished from w. Therefore, we say that $[o.e]P$ is satisfied in state w if and only if $[o.e]P$ is also satisfied in w'.

Because there may happen at most one event in an object at a moment, there is no further case and these three cases are disjoint.

Example 4.2

For the interpretation structure $\mathcal{IS}_{\texttt{Authorclass}}$ given in Example 4.1 we regard some propositions and check whether they are satisfied in certain community states under the assumption of some assignment. As assignments we use $\mathcal{A}_1$ and $\mathcal{A}_2$ where $\mathcal{A}_1(AC) = \underline{o}$, $\mathcal{A}_1(A) = \underline{a12}$ $(= I(\texttt{Authors}(\underline{o}, I(12)))$), $\mathcal{A}_1(S) = I(\text{"}\texttt{F.Kafka}\text{"})$ and $\mathcal{A}_2(AC) = \underline{o}$, $\mathcal{A}_2(A) = \underline{a12}$ $(= I(\texttt{Authors}(\underline{o}, I(12)))$), $\mathcal{A}_2(S) = I(\text{"}\texttt{E.Kishon}\text{"})$:

$$\begin{aligned}
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_1, w_2 \rangle &\models Name(A, S) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_2 \rangle &\models \neg Name(A, S) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_4 \rangle &\models \neg SoldBooks(A, 17) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_1, w_1 \rangle &\models \texttt{occur}(A.create(\text{"}\texttt{F.Kafka}\text{"}, \text{"}\texttt{married}\text{"})) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_4 \rangle &\models [A.changeName(S)]Name(A, S) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_1 \rangle &\models \\
&\quad \neg[AC.addAuthor(12, \text{"}\texttt{F.Kafka}\text{"}, \text{"}\texttt{married}\text{"})]NumberOfAuthors(AC, 0)
\end{aligned}$$

$\bowtie$

Before we proceed to define truth of formulas we have to point out that the propositions $[o.e]\neg P$ and $\neg[o.e]P$ are in general not equal with regard to their satisfaction. This is due to the implicit implication mentioned before and can easily be seen by comparing $A \to \neg B$ and $\neg(A \to B)$ which are not equivalent in classical propositional or predicate logic as well.

Definition 4.7 (Truth of Formulas)

A formula $\phi \equiv P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ is called true in an interpretation structure $\mathcal{IS}$ for Σ_C (written $\mathcal{IS} \models \phi$) if and only if the following condition holds:

$$\text{for each } w \in \mathcal{W} \text{ and each assignment } \mathcal{A}:$$

$$\langle \mathcal{IS}, \mathcal{A}, w \rangle \models P_1 \text{ and } \ldots \text{ and } \langle \mathcal{IS}, \mathcal{A}, w \rangle \models P_n$$

$$\text{implies} \quad \langle \mathcal{IS}, \mathcal{A}, w \rangle \models Q_1 \text{ or } \ldots \text{ or } \langle \mathcal{IS}, \mathcal{A}, w \rangle \models Q_m$$

$\triangleleft$

Example 4.3

For checking whether a formula is true we have to inspect all states of an interpretation structure. Because we have presented the interpretation structure $\mathcal{IS}_{\texttt{Authorclass}}$ incompletely in Example 4.1 we can only show for some formulas that are not true in $\mathcal{IS}_{\texttt{Authorclass}}$. To verify this it is sufficient to find a community state and an assignment for which the considered formula does not hold. As assignments we reuse the assignments $\mathcal{A}_1$ and $\mathcal{A}_2$ of Example 4.2:

- $[A.changeName(S)]Name(A, S) \;\rightarrow\; Name(A, S)$

  This formula cannot be true in $\Sigma_{\texttt{Authorclass}}$ because $\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_4 \rangle \models [A.changeName(S)]Name(A, S)$ holds, but $\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_4 \rangle \models Name(A, S)$ does not hold.

- $NumberOfAuthors(AC, 0)$

  $\rightarrow\; [AC.addAuthor(12, \text{"}\texttt{F.Kafka}\text{"}, \text{"}\texttt{married}\text{"})]NumberOfAuthors(AC, 2)$

Here, $\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_1 \rangle \models NumberOfAuthors(AC, 0)$ holds. Moreover, the event $AC.addAuthor(12, \text{"}\texttt{F.Kafka}\text{"}, \text{"}\texttt{married}\text{"})$ occurs in state w_1. However, $NumberOfAuthors(AC, 2)$ is not satisfied in the following state w_2, i.e., we have the situation that $\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_1 \rangle \models [AC.addAuthor(12, \text{"}\texttt{F.Kafka}\text{"}, \text{"}\texttt{married}\text{"})] NumberOfAuthors(AC, 2)$ does not hold.

Of course, there are also formulas which are true in $\Sigma_{\texttt{Authorclass}}$, provided that the fragment of this interpretation structure is completed adequately. Such formulas are for instance,

$$\rightarrow\ [A.create(S, S')]Name(A,\ S)$$

$$\mathsf{occur}(AC.addAuthor(Nr, S, S')) \ \rightarrow\ \mathsf{occur}(Authors(AC, Nr).create(S, S'))$$

The first one describes that after the occurrence of a `create` event for an author object the attribute `Name` of this author object has exactly the value given as parameter to the `create` event. The second one deals with interaction between an authorclass object and one of its sub-objects. An occurrence of an `addAuthor` event in an authorclass object always causes an occurrence of a `create` event in the corresponding sub-object. The parameter of the `create` event are given by the parameter of the `addAuthor` event. $\bowtie$

Definition 4.8 (Semantic Entailment)

Given a community signature Σ_C and a set Φ of formulas over Σ_C. That a formula ϕ is semantically entailed by Φ (written $\Phi \models_{\Sigma_C} \phi$) is defined by:

$$\Phi \models_{\Sigma_C} \phi \qquad :\Longleftrightarrow \qquad \left(\begin{array}{ll} \text{for each interpretation structure } \mathcal{IS} \text{ for } \Sigma_C\text{:} & \\ & \mathcal{IS} \models \phi' \text{ for each } \phi' \in \Phi \\ \text{implies} & \mathcal{IS} \models \phi \end{array} \right)$$

$\lhd$

Definition 4.9 (Semantic Closure)

Given a community signature Σ_C and a set Φ of formulas over Σ_C. The semantic closure $\Phi^{\models_{\Sigma_C}}$ of Φ is the set of all formulas over Σ_C which are semantically entailed by Φ:

$$\phi \in \Phi^{\models_{\Sigma_C}} \qquad :\Longleftrightarrow \qquad \Phi \models_{\Sigma_C} \phi$$

$\lhd$

Finally, we can define the notion "model".

Definition 4.10 (Model)

An interpretation structure $\mathcal{IS}$ over a community signature Σ_C is a model for a set Φ of formulas over Σ_C, if and only if all formulas in Φ are true in $\mathcal{IS}$. $\lhd$

Because the logical description of an object community is given as a set of formulas we call a model for this set of formulas also a model of the considered object community.

Assuming a set of formulas being a logical description of our running example, we could complete the fragment of the interpretation structure $\Sigma_{\texttt{Authorclass}}$ given in Example 4.1 in order to get a model for this object community.

In proving object properties by means of a theorem proving system, however, concrete models are not to the fore. Instead the syntactical derivation of true formulas is more important. Therefore, the next section deals with an axiomatic semantics for our calculus.

5 Axiomatic Semantics

In the sequel we define the syntactical derivation of formulas. First we introduce the notion of substitution. Then we present basic inference rules for our calculus. Afterwards we define syntactical derivation (or syntactical entailment) as application of inference rules and introduce the notion of "generated theory". We conclude by discussing soundness and completeness issues of the basic calculus.

Definition 5.1 (Substitution)

A substitution s is a function which substitutes a finite number of variables by terms. These terms must not contain any variable which is replaced by this substitution. A concrete substitution is written as $\{X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n\}$. The application of a substitution s on a proposition P is denoted by $s(P)$. $\lhd$

Substitutions are used in context with inference rules. Applying a substitution to propositions in a formula results in a new formula which is an instantiation of the former one.

Definition 5.2 (Inference Rules)

The basic calculus includes the following inference rules:

$$\frac{}{P \to P}\ \textbf{(axiom)} \qquad \frac{R, P_1, \ldots, P_n \to Q_1, \ldots, Q_m}{P_1, \ldots, P_n \to Q_1, \ldots, Q_m, \neg R}\ \textbf{(negR)}$$

$$\frac{P_1, \ldots, P_n \to Q_1, \ldots, Q_m}{R, P_1, \ldots, P_n \to Q_1, \ldots, Q_m, R'}\ \textbf{(ext)} \qquad \frac{P_1, \ldots, P_n \to Q_1, \ldots, Q_m, R}{\neg R, P_1, \ldots, P_n \to Q_1, \ldots, Q_m}\ \textbf{(negL)}$$

$$\frac{P_1, \ldots, P_n \to Q_1, \ldots, Q_m, R \qquad R, P'_1, \ldots, P'_k \to Q'_1, \ldots, Q'_l}{P'_1, \ldots, P'_k, P_1, \ldots, P_n \to Q_1, \ldots, Q_m, Q'_1, \ldots, Q'_l}\ \textbf{(cut)}$$

$$\frac{P_1, \ldots, P_n \to Q_1, \ldots, Q_m}{s(P_1), \ldots, s(P_n) \to s(Q_1), \ldots, s(Q_m)}\ \textbf{(sub)} \qquad \frac{\mathsf{occur}(o_1.e_1) \to \mathsf{occur}(o_2.e_2)}{[o_2.e_2]P \to [o_1.e_1]P}\ \textbf{(occ)}$$

$$\frac{P_1, \ldots, P_n \to Q_1, \ldots, Q_m}{[o.e]P_1, \ldots, [o.e]P_n \to [o.e]Q_1, \ldots, [o.e]Q_m}\ \textbf{(nex)}$$

◁

Disregarding the last three rules we have a typical proof system for a propositional calculus with Gentzen-clauses as formulas (cf. for instance [21, 20]). The rule **(sub)** is necessary for renaming and replacing variables in order to get more concrete instances of a formula. Occurrence dependencies between events can be transferred into positional operators by rule **(occ)**. The last rule makes it possible to introduce positional operators. Such a rule is also used in [16, 17] for similar calculi.

Essentially, the presented inference rules are dealing with the propositional part of our calculus. Here, we have refrained from adding further rules, e.g., for dealing with equality. This is due to the fact that our main goal is the development of a basic calculus which can later be extended.

Before we proceed to the definition of syntactical entailment we have to mention that properties of objects which are inherent in our object model must be given as axioms (or axiom schemes). Here, we give some examples:

Example 5.1

The following axioms (or axiom schemes) express properties being inherent in our object model:

- $\mathsf{occur}(O.E) \rightarrow \mathsf{enable}(O.E)$

  An occurring event must always be enabled.

- $\mathsf{occur}(O.E_1), \mathsf{occur}(O.E_2) \rightarrow O.E_1 = O.E_2$

  There cannot be two (or more) different events occurring in the same object at the same moment.

- $\mathrm{a}(O, P_1, \ldots, P_n, V_1),\ \mathrm{a}(O, P_1, \ldots, P_n, V_2) \rightarrow V_1 = V_2$

  This axiom scheme requires for each attribute a (with $\mathrm{a} \in \mathtt{ATT}_{O\,s_1 \ldots s_n, s}$) that the attribute value is unique for each instant of time.

$\bowtie$

These are only a few examples of properties inherent to the object model. These axioms have to be included into each specification of an object community. We have chosen this explicit representation of such properties because our logical framework is to be flexible in order to be adaptable to several different object models. Hence, we must only know the inherent properties of an object model and express them by additional axioms. In this way we can easily use our logic for several different object specification approaches.

Of course, we could have coded such properties into the definition of interpretation structures. But then, we lose the intended flexibility.

In the next definition we use the terms "formula schema" and "instantiation of a formula schema". We call $\psi = P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ a formula schema if $P_1, \ldots, P_n, Q_1, \ldots, Q_m$ are variables standing for arbitrary propositions (where n and m may be unfixed). An instantiation of such a formula schema is a formula of the basic calculus such that each occurrence of a variable in the formula schema is replaced by the same proposition.

Definition 5.3 (Syntactical Entailment)

A formula ϕ can be (syntactically) entailed from a set Φ of formulas (written $\Phi \vdash_{\Sigma_C} \phi$) iff

- $\phi \in \Phi$ (the trivial case), or
- $\Phi \vdash_{\Sigma_C} \phi_1, \ \ldots, \ \Phi \vdash_{\Sigma_C} \phi_n$, and there is an inference rule

$$\frac{\psi_1 \quad \ldots \quad \psi_n}{\psi} \quad ,$$

  where $\psi_1, \ldots, \psi_n, \psi$ are formula schemas and

  - $\phi_1, \ldots, \phi_n$ are instantiations of the formula schemas $\psi_1, \ldots, \psi_n$ and
  - ϕ is an instantiation of ψ,

  where all instantiations replace the same variable by the same proposition.

$\triangleleft$

Please note that we distinguish between "semantic entailment" (Def. 4.8) and "syntactical entailment". In order to avoid confusion we use the adjectives "semantic" and "syntactical" whenever necessary.

Definition 5.4 (Generated Theory)

The theory generated by a set Φ of formulas over a community signature Σ_C is denoted by $\Phi^{\vdash_{\Sigma_C}}$ and is obtained by:

$$\phi \in \Phi^{\vdash_{\Sigma_C}} \quad :\Longleftrightarrow \quad \Phi \vdash_{\Sigma_C} \phi$$

$\lhd$

Thus, given a logical description Φ_C of an object community with signature Σ_C, the generated theory $\Phi^{\vdash_{\Sigma_C}}$ is the set of all formulas over Σ_C which can be syntactically entailed (by means of the inference rules given in Def. 5.2).

The following theorem states that the calculus is sound, i.e., that we cannot syntactically entail formulas which are not semantically entailed.

Theorem 5.1 (Soundness)

The basic calculus is sound, i.e., $\Phi^{\vdash_{\Sigma_C}} \subseteq \Phi^{\models_{\Sigma_C}}$, or equally:

$$\Phi \vdash_{\Sigma_C} \phi \quad \Longrightarrow \quad \Phi \models_{\Sigma_C} \phi$$

$\bigcirc$

Sketch of proof:

The main point in proving this theorem is the soundness of the rules **(nex)** and **(occ)**. For the other rules we can easily adopt soundness proofs of other sequent calculi. This

is due to the fact that these rules do not involve positional operators and, therefore, we can reduce soundness of these rules to the soundness of the corresponding rules for first order predicate calculi.

For the soundness of the rule **(nex)** we may argue that the truth of $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ is sufficient for the truth of $[E]P_1, \ldots, [E]P_n \rightarrow [E]Q_1, \ldots, [E]Q_m$. This results from the fact that the introduction of the positional operator can be seen as a restriction on the set of states which have to satisfy $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$.

In [7] we give a more detailed and formal proof for the soundness of these inference rules accept for the **(occ)** rule. The soundness of **(occ)** can easily be seen by considering the following cases for two events $o_1.e_1$ and $o_2.e_2$ for which we assume that $\mathsf{occur}(o_1.e_1) \rightarrow \mathsf{occur}(o_2.e_2)$ is true:

Neither $o_1.e_1$ nor $o_2.e_2$ occurs (in a certain state):

> Then $[o_1.e_1]P$ and $[o_2.e_2]P$ are satisfied in that state for each P. Thus, $[o_2.e_2]P \rightarrow [o_1.e_1]P$ holds.

o_2, e_2 occurs, but $o_1.e_1$ does not:

> Here, $[o_1.e_1]P$ is satisfied and thereby $[o_2.e_2]P \rightarrow [o_1.e_1]P$ holds, too.

$o_1.e_1$ occurs:

> It follows that $o_2.e_2$ also occurs (due to the premise). Thus, if P holds after the occurrence of $o_2.e_2$ then, of course, it also holds after the occurrence of $o_1.e_1$.

After having stated the soundness of the proof system with regard to the underlying model theory, the question of completeness arises. We do not present a completeness theorem due to two main reasons:

- On the one hand we have not aimed at completeness in constructing the proof system. Instead we have only intended to provide a simple and clear collection of sound inference rules as a basis for later extensions. Therefore, the given proof system only copes with the logical kernel common to all intended extensions and modifications of our calculus.

- On the other hand, regarding other calculi for which the question of completeness has already been answered yields a meaningful assessment for our calculus. It is well-known that there are complete calculi for first-order predicate logic. By means of skolemization first-order predicate logic can obviously be embedded into our calculus. Thereby, our calculus is at least as expressive as first-order predicate calculus. If our calculus is more expressive, then it cannot be complete at all (assuming a strong notion of completeness). For a large class of temporal logics Abadi has already shown that these logics cannot be strongly complete due to the fact that they are at least as expressive as Peano arithmetics which is known to be incomplete [1].

In consequence, we have a justified assumption that our calculus cannot be completed w.r.t. the strong notion of completeness.

We conclude this section by giving an example of a simple proof. Especially, this proof shows the effect of interaction between objects.

Example 5.2

Continuing the running example the property we want to prove is as follows: after the occurrence of an `addAuthor` event in the authorclass object the name of the new author

object is exactly the name given as parameter to the `addAuthor` event:

$$\rightarrow [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\ S)$$

The proof uses two formulas (φ_1, φ_2) which were already given in Ex. 4.3:

$$\dfrac{\dfrac{\varphi_1}{\varphi_3}\ \textbf{(sub)} \qquad \dfrac{\varphi_2}{\varphi_4}\ \textbf{(occ)}}{\varphi_5}\ \textbf{(cut)}$$

where

φ_1 : $\rightarrow\ [A.create(S, S')]Name(A,\ S)$

φ_2 : $\mathsf{occur}(AC.addAuthor(Nr, S, S'))\ \rightarrow\ \mathsf{occur}(Authors(AC, Nr).create(S, S'))$

φ_3 : $\rightarrow\ [Authors(AC, Nr).create(S, S')]Name(Authors(AC, Nr),\ S)$

φ_4 : $[Authors(AC, Nr).create(S, S')]Name(Authors(AC, Nr),\ S)$

$\qquad\qquad \rightarrow\ [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\ S)$

φ_5 : $\rightarrow\ [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\ S)$

Starting with φ_1 the substitution of A by $Authors(AC, Nr)$ yields φ_3. Formula φ_4 is obtained by applying the rule **(occ)** to φ_2. Finally, we can combine φ_3 and φ_4 using the rule **(cut)**. The result is φ_5, i.e., the formula to be proved. $\bowtie$

6 On Models and Proofs for Composed Specifications

In the sequel we discuss the relationship between models of a composed specification and models of its components. Furthermore, we consider the corresponding theories and, thus, proofs. Beforehand, we take a look at object hierarchies in a general and abstract way.

First, we regard a universal scheme for object hierarchies. Therefore, we need a super-object $\hat{o}$ having sub-objects $o_1, \ldots, o_n$ as depicted in Fig. 6.

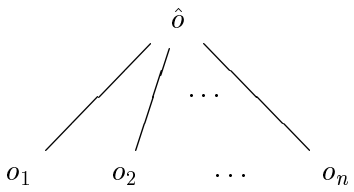


Figure 6: Object hierarchy (scheme).

Each of these objects is specified by a TROLL *light* template: let $\hat{o}$ be described by a specification $\hat{S}$, o_1 by S_1, o_2 by S_2, ..., and o_n by S_n. In general, there is no need for these specifications to be pairwise distinct. In fact, a super-object often has a large number of similar sub-objects described by only one specification. For instance, in Example 2.1 we have presented such an object community, where the super-object described by the object specification `Authorclass` has sub-objects described by the object specification `Author`.

In correspondence to the object hierarchy, we can state relationships between the spec-

ifications involved (see Fig. 7 where the directed edges mean that, e.g., specification S_1 describes a component of $\hat{S}$).

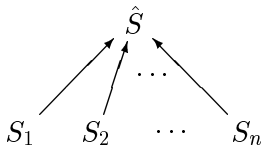


Figure 7: Relationships among the corresponding specifications.

Whereas the objects of an object community are organized in a hierarchy (or, more generally, in a directed acyclic graph), the relationship graph for the corresponding specifications is directed, but in general not acyclic.

Although we often talk of $\hat{o}$ as a single object, we sometimes mean the whole object hierarchy with $\hat{o}$ as root when using $\hat{o}$. In consequence, $o_1, \ldots, o_n$ do not need to be single objects, they may stand for object hierarchies as well. In this way several object hierarchies may be composed to form a new larger hierarchy.

Next, we consider models. Each model describes a possible life cycle of an object community. Restricting the object hierarchy given above to the super-object $\hat{o}$ and two sub-objects $o_1,\ o_2$, the following diagram depicts a possible life cycle for $\hat{o}$ (i.e., it illustrates an interpretation structure $\mathcal{IS} = (\mathcal{W}, \rho)$ for $\Sigma_{\hat{S}}$):

$$\mathcal{IS}: \quad w_0 \xrightarrow{\hat{o}.e_1} w_1 \xrightarrow{\hat{o}.e_2,\ o_1.e_1,\ o_2.e_1} w_2 \xrightarrow{\hat{o}.e_3,\ o_1.e_2} w_3 \xrightarrow{o_1.e_3, o_2.e_2} w_4 \xrightarrow{\hat{o}.e_4,\ o_2.e_3} w_5 \xrightarrow{o_1.e_4} \cdots$$

State transitions are caused by sets of simultaneously occurring events. This life cycle for $\hat{o}$ includes life cycles for the sub-objects o_1 and o_2. The latter life cycles can be extracted by only taking the events for o_1 and for o_2, respectively, into account. Thereby, we obtain

two life cycles (and, thus, two corresponding interpretation structures $\mathcal{IS}^{o_1} = (\mathcal{W}^{o_1}, \rho^{o_1})$ and $\mathcal{IS}^{o_2} = (\mathcal{W}^{o_2}, \rho^{o_2})$):

$$\mathcal{IS}^{o_1} : w_0^{o_1} \longrightarrow w_1^{o_1} \xrightarrow{o_1.e_1} w_2^{o_1} \xrightarrow{o_1.e_2} w_3^{o_1} \xrightarrow{o_1.e_3} w_4^{o_1} \longrightarrow w_5^{o_1} \xrightarrow{o_1.e_4} \cdots$$

$$\mathcal{IS}^{o_2} : w_0^{o_2} \longrightarrow w_1^{o_2} \xrightarrow{o_2.e_1} w_2^{o_2} \longrightarrow w_3^{o_2} \xrightarrow{o_2.e_2} w_4^{o_2} \xrightarrow{o_2.e_3} w_5^{o_2} \longrightarrow \cdots$$

With regard to the single objects, state transitions are caused by at most one event. However, there may be state transitions without an event occurrence, so-called *idle-transitions*. An idle-transition represents the possibility for the environment of the object to change without concerning the considered object itself. As we have already pointed out in Sect. 4, life cycles with idle-transitions can be considered as logically equivalent to life cycles without idle-transitions (from the local point of view of the considered object). We have indexed the states in the life cycle diagrams for o_1 and o_2 by the corresponding object names in order to clarify that these states do not equal those in the life cycle of $\hat{o}$. Especially, the functions ρ^{o_1} and ρ^{o_2} differ from ρ due to the fact that the signatures Σ_{S_1} and Σ_{S_2} are only signatures of components, i.e., they are only restrictions of $\Sigma_{\hat{S}}$. Hence, $\texttt{SPROP}^{o_1}$ and $\texttt{SPROP}^{o_2}$ are only subsets of SPROP.

Thereby, we can easily construct models for sub-objects (or sub-hierarchies) from a given model of their super-object. In principle, we only have to restrict ρ to $\texttt{SPROP}^{o_1}$ and $\texttt{SPROP}^{o_2}$, respectively. Thus, a rather simple projection is sufficient for extracting models for sub-objects (Fig. 8).

In contrast, the other direction usually does not work. We cannot compose arbitrary models for components in order to obtain a model for the composed specification. On the one hand, arbitrary models of sub-objects cannot always be put together because

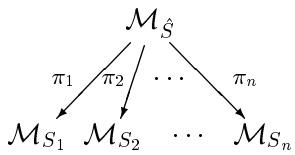


Figure 8: Relationships among the corresponding models.

of properties specified for the super-object, i.e., if the specification of super-objects requires an interaction between the sub-objects which is not fulfilled by the combination of the chosen models for the sub-objects. On the other hand, the super-object may have additional attributes and events for which there is no information in models of sub-objects.

After having seen that models may be decomposed according to the object hierarchy we go over to the corresponding theories. The following considerations are valid for generated theories (Def. 5.4) as well as for semantic closures (Def. 4.9). Therefore, we just talk of theories, i.e., of sets of formulas closed under a given deduction principle. Such a set of formulas is a logical description of an object community, e.g., we have a theory $\mathcal{T}h_S$ of formulas over Σ_S.

In contrast to the projection of models in Fig. 8, the theories $\mathcal{T}h_{S_1}, \ldots, \mathcal{T}h_{S_n}$ for the sub-objects cannot be obtained by restricting $\mathcal{T}h_{\hat{S}}$ to the signatures of S_1, ..., S_n, respectively. This property is however not necessary for proving object properties. Instead the opposite direction from a theory $\mathcal{T}h_{S_i}$ of a sub-object to the theory $\mathcal{T}h_{\hat{S}}$ of the super-objects has to be considered. Here, we can easily see that $\mathcal{T}h_{S_i}$ is always a (closed) sub-theory of $\mathcal{T}h_{\hat{S}}$, i.e., $\mathcal{T}h_{S_i} \subseteq \mathcal{T}h_{\hat{S}}$ for each i. In order to prove this, we can show that each formula $\phi \in \mathcal{T}h_{S_i}$ must also be included in $\mathcal{T}h_{\hat{S}}$. This is due to the

fact that ϕ is fulfilled by each model $\mathcal{M}_{S_i}$ and, thereby, ϕ holds in each model $\mathcal{M}_{\hat{S}}$ as well. Otherwise, if ϕ does not hold in some model $\mathcal{M}_{\hat{S}}$, it cannot hold in the model $\mathcal{M}_{S_i}$ yielded by the model projection.

The relationships between the theories corresponding to the community signatures from Fig. 7 are depicted in Fig. 9.

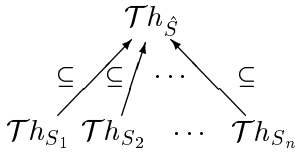


Figure 9: Inclusion hierarchy of the corresponding theories.

The property that theories for sub-objects are included in the theory of the super-object is important for proving object properties. This leads us to the following principle for proving object properties: Having proved a property of a component locally with regard to its (sub-) object community described by S_i, this property may also be used in the context of super-object, i.e., the (monotonic) extension of community signature preserves the validity of proofs. Thereby, we have a locality principle for verifying object specification, because we may carry out proofs as local as possible. In this way not only locality of objects is reflected, but also compositionality of object specification and compositionality of object verification go together very nicely.

The properties of models and theories, as we have sketched them above, are strong indications of having an institution (cf. [23]). Indeed, we can prove that our calculus fits into an institution [7].

7 Related Work and Conclusion

We have presented a basic calculus for verifying synchronously interacting objects. This calculus was originally designed for the object specification language TROLL *light* [11, 22] which is a dialect of TROLL [27, 28] restricted to a small number of essential concepts. However, the calculus may be used for other object specification languages as well, provided they have a similar view of objects and, especially, they assume synchronous interaction.

For TROLL *light* we have developed a transformation yielding a set of formulas as a logical description from a given TROLL *light* specification [8, 7]. Furthermore, in [41] a first prototype of this calculus was realized on top of the generic theorem prover Isabelle [34]. By means of this prototype we have already been able to carry out several proofs. At the time being, however, the degree of automation is quite limited. For improving this situation we would have to find out whether there is a cut-free version of our calculus which would allow a much more automated reasoning.

Our calculus essentially originated in the following approaches: the object calculi presented in [16, 17] and the object specification logic OSL [39]. We have combined these two approaches. From the object calculi we have mainly borrowed the syntactical structure of formulas. In contrast to e.g. [17] we do not allow terms which have to be interpreted in dependence of time. This would complicate the proof system, especially the substitution of terms would have to be handled more carefully. Instead we have adopted the view from OSL that only predicates should be time-varying. Furthermore,

the composition of object specifications on the logical level is related to the composition realized for OSL.

We have extended the local reasoning about object properties as it is possible in the object calculi to a global reasoning by allowing global event term in positional operators. This was mainly affected by the OSL in which referring to different objects is allowed. We have pursued further this idea and, thus, we have obtained a calculus which allows a more uniform treatment of objects with regard to global reasoning about an object community. Especially, we do not need to distinguish different proof systems for local and global reasoning. With regard to the intended kind of compositionality which we have presented and discussed in great detail this is a very important point. Of course, there are several other approaches to compositional temporal logic or compositional temporal specification around (e.g., [14, 18, 39, 2, 33]). A lot of them distinguish between a local and a global language and, thus, between local and global reasoning. Several authors promise compositionality without showing at all how to compose specifications. Moreover, there are some approaches presenting really compositional temporal logics several of which are not intuitive because of the way next operators are interpreted on the global level. A more detailed discussion on this topic can be found in [9].

Here, we have presented a basic calculus for which several extensions might be useful. We restricted the treatment of temporal properties to the use of positional operators which can be seen as an element of dynamic logic [24]. Introducing the temporal operators F and G would be necessary for being able to talk about further temporal properties.

Another interesting direction is the integration of deontic logic concepts (cf. [31, 44]). Although there is already some work on combining temporal and deontic logic this

mainly results in interpreting temporal operators in a deontic style, cf. [17]. Over and above that, deontic logic seems to be a promising way of dealing with temporarily non-normative behaviour in information systems, e.g. the violation of integrity constraints during a transaction. A first approach preparing the integration of deontic concepts into our calculus is given in [10].

With regard to the intended application area, the object-oriented modelling of information systems, integration of other object-oriented concepts must be considered. Although, we have not explicitly talked about inheritance when discussing the calculus, there is no special problem in expressing inheritance within our calculus — as long as we are considering monotonic kinds of inheritance. For non-monotonic inheritance, e.g., inheritance allowing overriding, a non-monotonic logic is necessary. We decided to have a monotonic logic due to the fact that most of the specification languages in the area of object-oriented specification only allow monotonic inheritance. Nevertheless, the treatment of non-monotonic inheritance becomes very important when we want to use default assumptions in specifying objects (cf. [6]).

Although we have focused on hierarchical composition of object communities throughout this paper, the calculus allows more liberal forms. This is due to the way object name generators are introduced in Def. 3.2. Thereby, the identification of objects can be organized in many different ways.

Apart from all technical extensions of the basic calculus another main aspect of future work must be the investigation of reasoning about objects and object communities in practice. Up to now, there seems to exist a general lack of knowledge about practical needs as well as a non-understanding of how to adequately reflect the interaction of

communicating objects in proofs and proof systems.

Acknowledgements:

Work reported here has been partially supported by the CEC projects ModelAge and ISCORE II (Esprit Basic Research Working Groups No. 8319 and 6071) and by the German National Ministry for Research and Technology (BMFT) under Grant No. 01 IS 203 D (project KorSo). I am especially grateful to G. Denker, H.-D. Ehrich, M. Gogolla, R. Herzig, and N. Vlachantonis for many fruitful discussions on our common work in the KorSo project. Furthermore, Th. Hartmann, J. Kusch, G. Saake, I. Schmitt, and C. Türker discussed with me on certain aspects of this work. Last but not least I am grateful to the anonymous referees which considerably helped to improve this paper.

References

[1] M. Abadi, The Power of Temporal Proofs, *Theoretical Computer Science* 65 (1989) 35–83.

[2] M. Abadi and L. Lamport, Composing Specifications, *ACM Trans. on Programming Languages and Systems* 15 (1993) 73–132.

[3] E. Astesiano and G. Reggio, Algebraic Specification of Concurrency, in M. Bidoit and C. Choppy, eds., *Recent Trends in Data Type Specification WADT'91* (Springer LNCS 655, Berlin, 1993) 1–39.

[4] D. Bjorner, C.A.R. Hoare, and H. Langmaack, eds., *VDM'90: VDM and Z — Formal Methods in Software Development* (Springer LNCS 428, Berlin, 1990).

[5] G. v. Bochmann, M. Barbeau, M. Erradi, L. Lecomte, P. Mondain-Monval, and N. Williams, Mondel: An Object-Oriented Specification Language, Département d'Informatique et de Recherche Opérationnelle, Publication 748, Université de Montréal, 1990.

[6] S. Brass, U.W. Lipeck, and P. Resende, Specification of Object Behaviour with Defaults, in U.W. Lipeck and G. Koschorreck, eds., *Proc. of the Int. Workshop on Information Systems — Correctness and Reusability IS-CORE'93*, Informatik–Bericht 01/93 (Universität Hannover, 1993) 155–177.

[7] S. Conrad, *Ein Basiskalkül für die Verifikation von Eigenschaften synchron interagierender Objekte*, Fortschritt-Berichte Reihe 10, Nr. 295 (VDI-Verlag, Düsseldorf, 1994).

[8] S. Conrad, On Certification of Specifications for TROLL *light* Objects, In H. Ehrig and F. Orejas, eds., *Recent Trends in Data Type Specification — 9th Workshop on Abstract Data Types/4th Compass Workshop* (Springer LNCS 785, Berlin, 1994) 158–172.

[9] S. Conrad, Temporal Logic Specification of Objects: An Approach to Compositionality and Reusability Allowing `next` Operators, in R. Wieringa and R. Feenstra, eds., *Working papers of the International Workshop on Information Systems - Correctness and Reusability IS-CORE'94*, Report No. IR-357 (Vrije Universiteit Amsterdam, 1994) 228–241.

[10] S. Conrad and H.-D. Ehrich, An Elementary Logic for Object Specification and Verification, in U. Lipeck and G. Vossen, eds., *Workshop Formale Grundlagen für den Entwurf von Informationssystemen, Tutzing*, Informatik-Bericht 03/94 (Universität Hannover, 1994) 197–206.

[11] S. Conrad, M. Gogolla, and R. Herzig, TROLL *light*: A Core Language for Specifying Objects, Informatik-Bericht 92–02, TU Braunschweig, 1992.

[12] E. Dubois, P. Du Bois, and M. Petit, O-O Requirements Analysis: An Agent Perspective, in O.M. Nierstrasz, ed., *Proc. European Conf. on Object-Oriented Programming ECOOP'93* (Springer LNCS 707, Berlin, 1993) 458–481.

[13] H.-D. Ehrich, G. Denker, and A. Sernadas, Constructing Systems as Object Communities, in M.-C. Gaudel and J.-P. Jouannaud, eds., *Proc. TAPSOFT'93: Theory and Practice of Software Development* (Springer LNCS 668, Berlin, 1993) 453–467.

[14] P. Enjalbert and M. Michel, Many-Sorted Temporal Logic for Multi-Processes Systems, in M.P. Chytil and V. Koubek, eds., *Mathematical Foundations of Computer Science* (Springer LNCS 176, Berlin, 1984) 273–281.

[15] R.B. Feenstra and R.J. Wieringa, LCM 3.0: A Language for Describing Conceptual Models, Technical Report IR-344, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1993.

[16] J. Fiadeiro, *Cálculo de Objectos e Eventos*, PhD thesis, Instituto Superior Técnico (Technical University of Lisbon, 1988).

[17] J. Fiadeiro and T. Maibaum, Temporal Reasoning over Deontic Specifications, *Journal of Logic and Computation* 1 (1991) 357–395.

[18] J. Fiadeiro and T. Maibaum, Temporal Theories as Modularisation Units for Concurrent System Specification, *Formal Aspects of Computing* 4 (1992) 239–272.

[19] J. Fiadeiro and A. Sernadas, Logics of Modal Terms for System Specification, *Journal of Logic and Computation* 1 (1990) 187–227.

[20] J. Gallier, Constructive Logics — Part I: A Tutorial on Proof Systems and Typed λ-Calculi, *Theoretical Computer Science* 110 (1993) 249–339.

[21] G. Gentzen, Untersuchungen über das logische Schließen, *Mathematische Zeitschrift* 39 (1935) 176–210.

[22] M. Gogolla, S. Conrad, and R. Herzig, Sketching Concepts and Computational Model of TROLL *light*, in A. Miola, ed., *Proc. 3rd Int. Conf. Design and Implementation of Symbolic Computation Systems DISCO'93* (Springer LNCS 722, Berlin, 1993) 17–32.

[23] J.A. Goguen and R.M. Burstall, Institutions: Abstract Model Theory for Specification and Programming, *Journal of the ACM* 39 (1992) 95–146.

[24] D. Harel, Dynamic Logic, in D. Gabbay and F. Guenthner, eds., *Handbook of Philosophical Logic II: Extensions of Classical Logic* (D. Reidel, Boston, 1984) 497–604.

[25] T. Hartmann, G. Saake, R. Jungclaus, P. Hartel, and J. Kusch, Revised Version of the Modelling Language TROLL (Version 2.0), Informatik-Bericht 94–03, Technische Universität Braunschweig, 1994.

[26] R. Herzig, S. Conrad, and M. Gogolla, Compositional Description of Object Communities with TROLL *light*, in C. Chrisment, eds., *Proc. Basque Int. Workshop on Information Technology BIWIT'94*, (Cépaduès-Éditions, Toulouse, 1994) 183–194.

[27] R. Jungclaus, *Modeling of Dynamic Object Systems—A Logic-Based Approach*, Advanced Studies in Computer Science (Vieweg, Braunschweig/Wiesbaden, 1993).

[28] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas, TROLL – A Language for Object-Oriented Specification of Information Systems, *ACM Transactions on Information Systems* (1996), *to appear*.

[29] G. Kappel and M. Schrefl, Using an Object-Oriented Diagram Technique for the Design of Information Systems, in H.G. Sol and K.M. Van Hee, eds., *Dynamic Modelling of Information Systems* (North-Holland, Amsterdam, 1991) 121–164.

[30] J. Meseguer, A Logical Theory of Concurrent Objects, Proc. OOPSLA, *ACM SIGPLAN Notices* 25 (1990) 101–115.

[31] J.-J.Ch. Meyer and R.J. Wieringa, Deontic Logic: A Concise Overview, in J.-J.Ch. Meyer and R.J. Wieringa, eds., *Deontic Logic in Computer Science — Normative System Specification* (Wiley Professional Computing, 1993) 3–16.

[32] J.-J.Ch. Meyer and R.J. Wieringa, eds., *Deontic Logic in Computer Science — Normative System Specification* (Wiley Professional Computing, 1993).

[33] A. Mokkedem and D. Méry, A Stuttering Closed Temporal Logic for Modular Reasoning about Concurrent Programs, in D.M. Gabbay and H.J. Ohlbach, eds., *Proc. of the First Int. Conf. on Temporal Logic ICTL'94* (Springer LNAI 827, Berlin, 1994) 382–397.

[34] L.C. Paulson, Isabelle: The Next 700 Theorem Provers, in P. Odifreddi, ed., *Logic and Computer Science* (Academic Press, 1990) 361–385.

[35] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design* (Prentice-Hall, Englewood Cliffs, 1991).

[36] M. Ryan, Defaults in Specifications, in A. Finkelstein, ed., *Proc. IEEE Conf. on Requirements Engineering* (IEEE Computer Society Press, 1993).

[37] G. Saake and T. Hartmann, Modelling Information Systems as Object Societies, in K. von Luck and H. Marburger, eds., *Management and Processing of Complex*

Data Structures — Proc. 3rd Workshop on Information Systems and Artificial Intelligence (Springer LNCS 777, Berlin, 1994) 157–180.

[38] A. Sernadas and H.-D. Ehrich, What Is an Object, After All? in R. Meersman, W. Kent, and S. Khosla, eds., *Object-Oriented Databases: Analysis, Design and Construction — Proc. IFIP WG 2.6 Working Conference DS-4* (North-Holland, Amsterdam, 1991) 39–70.

[39] A. Sernadas, C. Sernadas, and J.F. Costa, Object Specification Logic, *Journal of Logic and Computation* 5 (1995).

[40] A. Sernadas, C. Sernadas, P. Gouveia, P. Resende, and J. Gouveia, OBLOG — Object-Oriented Logic: An Informal Introduction, Technical Report, INESC, Lisbon, 1991.

[41] Jens Siemer, Realisierung des TROLL *light*–Verifikationskalküls mit Hilfe des generischen Theorembeweisers ISABELLE, Master Thesis, TU Braunschweig, 1994.

[42] R. Wieringa, Equational Specification of Dynamic Objects, in R.A. Meersman, W. Kent, and S. Khosla, eds., *Object-Oriented Databases: Analysis, Design and Construction — Proc. IFIP WG 2.6 Working Conference DS-4* (North-Holland, Amsterdam, 1991) 415–438.

[43] R.J. Wieringa, Steps towards a Method for the Formal Modeling of Dynamic Objects, *Data & Knowledge Engineering* 6 (1991) 509–540.

[44] R.J. Wieringa and J.-J.Ch. Meyer, Applications of Deontic Logic in Computer Science: A Concise Overview, in J.-J.Ch. Meyer and R.J. Wieringa, *Deontic Logic in*

Computer Science — Normative System Specification (Wiley Professional Computing, 1993) 17–40.

[45] M. Wirsing, Algebraic Specification, in J. Van Leeuwen, ed., *Handbook of Theoretical Computer Science Vol. B* (North-Holland, Amsterdam, 1990) 677–788.