Feasible Object Certification‡

Paula Gouveia*
Ralf Jungclaus†
Gunter Saake§
Cristina Sernadas*

Abstract

In this paper we deal with the problem of proving properties of object behaviour based on temporal logic specifications. In general proving temporal properties for arbitrary specifications is not feasible. We show that using a restricted specification language and translating it to temporal logic enables an efficient certification of relevant classes of properties namely invariants and liveness properties. The proposed certification method uses tableau systems and techniques for arithmetic and equational reasoning leading to small and structured proofs that are easy to manage.

1 Introduction

Object-orientation is helpful for conceptual modeling and development of software systems [RBP+91, Jac92, CAB+94, Boo94, SM92]. Only few approaches, however, are based on a formal framework, among them [SSE87, SE90, FSMS90, Wie90, SF91, KS91, Jun93, JSHS95]. In this paper, we investigate the *certification* aspects of object-oriented conceptual model specifications by proving assertions about the specifications. We identify classes of assertions that can be feasibly proven and illustrate the proof technique.

The advantages of using object-based or object-oriented approaches to conceptual modeling can be summarized as follows:

- Objects are likened to things in the Universe of Discourse (UoD).
- Object modeling in many cases results in more intuitive models.

*Departamento de Matemática, Instituto Superior Técnico, Av. Rovisco Pais, 1096 Lisboa Codex, Portugal. Email: {mpg|css}@raf.ist.utl.pt

†Deutsche Telekom AG, Generaldirektion, P.O.Box 2000, D–53105 Bonn, Germany. Email: jungclau@u9000mst.nez.telekom.de

§Institut für Technische Informationssysteme, Otto-von-Guericke-Universität Magdeburg, D-39106 Magdeburg, Germany. Email: saake@iti.cs.uni-magdeburg.de

‡This work was partly supported by CEC under ESPRIT-III BRA WG 6071 IS-CORE (Information Systems – COrrectness and REusability) and WG 8319 ModelAge.

- The locality of object properties supports the structuring of conceptual models.
- A higher degree of extensibility and flexibility of specifications is achieved.
- Reuse is supported by encapsulation of objects.

The Troll approach to object-oriented conceptual modeling [JSS91, JSHS91, HJS92, Jun93, Saa93, JSHS95] is used in this paper. In Troll, we may specify the structure and behaviour as well as abstraction mechanisms like specializations, roles, composite objects and interactions between objects.

By using precisely defined notations as early as possible in the development process of software systems, requirements and concepts can be better analyzed. A precise representation based on formal semantics supports quality improvements and validation/requirements checking. Generally, it is more productive to avoid inconsistencies than to correct them later in the development process. Therefore, inconsistencies have to be *detected* as early as possible. One way of detecting inconsistencies is to *verify assertions* against the conceptual model specification.

Assertions to be checked can also be regarded as questions about a specification which have to be answered, which is one way to validate properties of the specification.

Verification in the context of temporal logic was addressed by several approaches namely model based ones [MP92] and proof based ones [Eme90, FM92, SSC92]. We adopt a proof approach translating Troll specifications into OSL [SSC92] because with proofs we can concentrate on relevant parts of the specification and avoid the consideration of unmanageable models. For doing so we do not take an axiomatic approach as in [MP92] but rather a tableau based method. This method provides a more direct way of implementing the theorem prover. In our implementation [GA93] we extended the temporal reasoning with a module coping with arithmetic and equational reasoning

A central problem in specification verification is the feasibility of proofs. Following [MP92] we will identify classes of properties that can be feasibly proven using a theorem prover for OSL based on the tableau method.

The main classes of assertions are the following:

- We may verify arbitrary temporal properties for single object specifications. Examples for such properties are invariants, state transition properties and possible evolutions of objects.
- For static specializations and aggregations we may prove e.g. properties that are only valid for specialized objects or objects being components of others, but not for those objects in isolation.

The paper is structured as follows: In the next section, we briefly present the Troll language as well as the language for stating assertions to be proven. Section 3 describes the translation of Troll specifications into theory presentations of the underlying object logic OSL. In Section 4, we present the tableau technique used for proving assertions. Finally, we give some conclusions. All the figures are presented in the Appendix.

2 Troll-Specifications and Assertions over them

To show the evidence of our approach we start by giving some examples around a bank application. Moreover we illustrate some assertions that we would like to prove.

2.1 Specifying with Troll

A template specification defines the structural and behavioural properties of admissible instances. A *class specification* defines possible collections of instances which subscribe to the same template. Thus, it can be regarded as being the *schema* for a class. An instance is specified by a template identified by an identifier. Thus, a class type is defined by a *template* for instances along with a set of possible identifiers (the *identification space*) [JSHS95]. We ignore the aspect of specifying an identification mechanism in the following. A class can be seen as a *container* for instances of the corresponding class type.

As an example consider the following specification of a class of objects representing accounts.

Example 2.1 Specification of very simple accounts

```
object class Account
  template
    attributes
            Balance:integer;
            Limit:natural;
    events
       birth open(Limit:natural);
             deposit(Amount:natural);
             withdraw(Amount:natural);
    valuation
       [open(m)]Limit=m; Balance=0;
       [deposit(m)]Balance=Balance+m;
       [withdraw(m)]Balance=Balance-m;
    permissions
       {(Balance-m)>= - Limit}withdraw(m);
       {m<=2000}withdraw(m);
end object class Account
```

■

In the above specification we have the attributes `Balance` and `Limit` and the events `open`, `deposit` and `withdraw`. Event `open` is the birth event which means that it occurs at most once and it is always the first event to occur. Valuations state the effect of events upon attributes: `[deposit(m)]Balance=Balance+m`, for instance, states that the occurrence of `deposit(m)` increases by m the value of `Balance`. Permissions impose conditions on the occurrence of the events. The permission `{m<=2000}withdraw(m)`, for instance, states that `withdraw(m)` can only occur provided that the value of `m` is less or equal than 2000.

A *specialization* represents a more detailed aspect of an object. Hence from a class we can extract a subclass whose instances can have more attributes and events and even a more specialized behaviour. When specializing an object class we are assuming that no new birth events are introduced.

Example 2.2 In this example we specify a class of savings accounts. A savings account is an account with additional properties (interest is paid for savings accounts but not for accounts in general) and with restricted admissible behavior (we may only open it provided the limit is set to zero).

```
object class SavingsAccount
  specializing Account
  template
    attributes
        InterestRate:integer;
    events
        accumulateInterest;
    permissions
        {m=0}open(m);
    obligations
        accumulateInterest;
    interaction
        accumulateInterest>>deposit(InterestRate*Balance/100);
end object class SavingsAccount
```

■

Due to the locality principle in TROLL we can not directly state effects of additional events on attributes of the base template—i.e. we may not directly specify how the occurrence of the event `accumulateInterest` affects the value of the (inherited) attribute `Balance`. We must use *delegation* here and specify an interaction that causes the `deposit` event (initialized in an appropriate way) to occur whenever the `accumulateInterest` event occurs for a *SavingsAccount*. The locality principle in TROLL prohibits any violation of properties of a base instance in a specialization.

The behaviour of the object class *SavingsAccount* is more restricted than the behaviour of *Account* since we have introduced a permission involving the event `open` and the obligation `accumulateInterest` which imposes the occurrence of this event.

Example 2.3 In this example we present (a fragment of) the specification of the object `Bank`. In [JSHS95] several language faetures for specifying composed objects are described. For our purpose, we use the definition of `component` objects specifying instances to be embedded in the aggregated object.

```
object Bank
  template
    components
         acc_1: Account;
         acc_2: Account;
           ⋮
         acc_n: Account;
    events
         transfer(Acc1:Account,Acc2:Account,Amount:natural);
    interaction
         transfer(a,b,m)>>a.withdraw(m);
         transfer(a,b,m)>>b.deposit(m);
    permissions
         {a<>b}transfer(a,b,m);
           ⋮
 end object Bank
```

Component objects are still encapsulated; the modification of their attributes has to be done via communication inside the complex object. ■

2.2 Assertions

Assertions to be verified over TROLL specifications are stated in an assertion language similar to TROLL. They have the following general form:

```
assertion <assertion name> context <list of template/class names>
   <assertion class> : <statement>;
```

We consider three classes of assertions: *invariant*, *transition invariant* and *liveness*. Those assertions reflect properties we want to be verified during the object's life, i.e., after its birth. Class invariant includes (non-temporal) properties that should be verified in every state, e.g., properties related to attribute values. Class transition invariant includes (non-temporal) properties that should be preserved from state to state (either always or only under certain conditions). Finally, class liveness includes (non-temporal) properties that should be verified in some state.

Over the `Account` specification in example 2.1 we might want to verify assertions like the following.

Example 2.4 Invariant: The `Balance` never falls below $-$`Limit`.

```
assertion INV1 context Account
invariant : Balance>= - Limit;
```

■

Example 2.5 Transition Invariant: The `Balance` is never decreased by more than 2000.

```
assertion INV2 context Account
transition invariant :(Balance=n implies next Balance>=n-2000));
```

■

Example 2.6 Specialization: Since `accumulateInterest` must occur and this triggers `deposit`, we can assert that if an account is specialized, the `deposit` event does occur sometime. The proof of this assertion will illustrate how inheritance affects proofs.

```
assertion LIV context SavingsAccount
   liveness: (exists m:integer)occurs(deposit(m));
```

■

Example 2.7 Transition Invariant: Transferring an amount of money between two accounts does not change their global balance

```
assertion INV3 context Bank
   transition invariant : occurs transfer(a,b,m) implies
                          (a.balance+b.balance=n implies
                          next (a.balance+b.balance=n));
```

■

3 Translation into an Object Specification Logic

We start by making a short introduction to OSL. For more details see [SSC92]. Then we provide the translation of Troll specifications into OSL. We do it in three steps: signature, specification and assertions.

3.1 Object Specification Logic OSL

In the following we assume a pre-defined data specification $spec_{DT} = (\Sigma, Eq)$ with $\Sigma = (DS, OP)$. We assume that sort *bool* is in DS. This data specification is called the *data environment*. We start by defining signatures and specifications of primitive object classes.

Definition 3.1 (Object Template Signature)
A *object template signature* $\Phi = (AS, \alpha, ATT, EVT)$ consists of

- a set of *object sorts* AS and a *local sort* $\alpha \in AS$;
- an $S^* \times S$-indexed family of finite sets of *attribute symbols* ATT and an S^*-indexed family of finite sets of *event symbols* EVT where $S = DS \cup AS$. ∎

Definition 3.2 (Terms)
Let $\Phi = (AS, \alpha, ATT, EVT)$ be an object template signature and X an S-indexed family of sets of variables. The S-indexed family $T_\Phi(X)$ of terms is defined as

- $o(t_1, ..., t_n) \in T_{\Phi,s}(X)$ for $o \in OP_{s_1 \ldots s_n, s}$ and $t_i \in T_{\Phi,s_i}(X)$, $1 \leq i \leq n$, $n \geq 0$
- $a(t_1, ..., t_n) \in T_{\Phi,s}(X)$ for $a \in ATT_{s_1 \ldots s_n, s}$ and $t_i \in T_{\Phi,s_i}(X)$, $1 \leq i \leq n$, $n \geq 0$

A term that only involves symbols in OP or X is a *data term.* ∎

Definition 3.3 (Atomic Formulae)
Let $\Phi = (AS, \alpha, ATT, EVT)$ be an object template signature and X an S-indexed family of sets of variables. The set of *object template atomic formulae* $\mathcal{AL}_\Phi(X)$ is defined as

- $born \in \mathcal{AL}_\Phi(X)$
- $t_1 = t_2 \in \mathcal{AL}_\Phi(X)$ where t_1, t_2 are terms of the same sort;
- $t \in \mathcal{AL}_\Phi(X)$ where t is a term of sort bool;
- $\diamond evt(t_1, ..., t_n), \triangledown evt(t_1, ..., t_n) \in \mathcal{AL}_\Phi(X)$ where $evt \in EVT_{s_1 \ldots s_n}$ and $t_i \in T_{\Phi,s_i}(X)$, $1 \leq i \leq n$, $n \geq 0$. ∎

Intuitively, $\diamond evt(t_1, ..., t_n)$ means that the event is enabled in the current state and $\triangledown evt(t_1, ..., t_n)$ means that the event occurred in the current state.

Definition 3.4 (Object Template Formulae)
Let $\Phi = (AS, \alpha, ATT, EVT)$ be an object template signature and X an S-indexed family of sets of variables. The set of *object template formulae* $\mathcal{TL}_\Phi(X)$ is defined as

- $\phi \in \mathcal{TL}_\Phi(X)$ if $\phi \in \mathcal{AL}_\Phi(X)$;
- $(\neg \varphi), (\varphi \Rightarrow \psi), (\forall x \varphi, \in \mathcal{TL}_\Phi(X)$ if $\varphi, \psi \in \mathcal{TL}_\Phi(X)$ and $x \in X$;
- $(\mathbf{X}\, \varphi), (\varphi \;\mathbf{U}\; \psi) \in \mathcal{TL}_\Phi(X)$ if $\varphi, \psi \in \mathcal{TL}_\Phi(X)$. ∎

We consider the usual connectives and universal quantifier and the temporal operators $\mathbf{X}$ and $\mathbf{U}$. Intuitively, $\mathbf{X}\,\varphi$ holds in a state when φ holds in the next state; $\varphi\,\mathbf{U}\,\psi$ holds in a state when there is a future state where ψ holds and, in every state from the former until the later, φ holds. An object template formula φ is a *state formula* iff φ does not involve any temporal operator.

For convenience, let us immediately introduce derived connectives and operators that are used in OSL specifications:

- $(\varphi \vee \psi) \equiv ((\neg\varphi) \Rightarrow \psi)$
- $(\varphi \wedge \psi) \equiv (\neg(\varphi \Rightarrow (\neg\psi)))$
- $(\varphi \Leftrightarrow \psi) \equiv ((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi))$
- $(\mathbf{F}\,\varphi) \equiv (true\,\mathbf{U}\,\varphi)$
- $(\mathbf{G}\,\varphi) \equiv (\neg(\mathbf{F}(\neg\varphi)))$
- $\exists x \phi \equiv (\neg(\forall x{:}\, s)\neg\phi)$

Besides the usual connectives we consider the temporal operators $\mathbf{F}$ (sometime in the future) and $\mathbf{G}$ (always in the future). Intuitively $\mathbf{F}\,\varphi$ ($\mathbf{G}\,\varphi$) holds in a state whenever φ holds in some (every) future state.

Definition 3.5 (Object Template Specification)
An *object template specification* is a pair $spec = (\Phi, Ax)$ where Φ is a template signature and $Ax \subseteq \mathcal{TL}_\Phi(X)$ is a set of object template formulae (the axioms of the specification). ■

For the interpretation structures of OSL specifications we consider linear-time models (see [MP92, SSC92]). Thus, an object life cycle is a linear sequence of states where each state is a set of currently valid facts. We consider the anchored version of temporal logic [MP92, SSC92]: a linear sequence of states satisfies a formula φ iff φ holds at its initial state. As a consequence, specification axioms will be, mainly, formulae of the kind $\mathbf{G}\,\varphi$.

OSL has the following universal axioms (besides the usual ones for first order logic and temporal logic):

- The occurrence of an action implies that the action must be enabled. For instance, in the completion of the specification of *Account* we have the axiom $\mathbf{G}(\triangledown deposit(m) \Rightarrow \diamond deposit(m))$.

- An attribute can only change its value providing that an action occurred. For instance, in the completion of the specification of *Account* we have the axiom $\mathbf{G}((\exists t(balance = t \wedge \mathbf{X}(balance \neq t)) \Rightarrow (\exists m \triangledown open(m) \vee (\exists m \triangledown deposit(m) \vee (\exists m \triangledown withdraw(m))))$. Note that this is the well known frame rule.

- Two actions of the same kind cannot occur at the same time. For instance, in the completion of the specification of *Account* we have the axiom $\mathbf{G}(((\triangledown deposit(m) \wedge \triangledown deposit(m')) \Rightarrow m = m'))$.

We can construct object template specifications from other object template specifications (e.g. specialization and aggregation) using *template morphisms*. A template morphism basically translates the symbols from a template signature into symbols of another template signature (where a translation may also be forgetting about certain symbols in the target signature).

Definition 3.6 (Template Signature Morphism)
Let $\Phi_i = (AS_i, \alpha_i, ATT_i, EVT_i), i = 1, 2$, be template signatures. A *template signature morphism* $\sigma : \Phi_1 \rightarrow \Phi_2$ consists of:

- a map $\sigma_s : AS_1 \rightarrow AS_2$;
- an $S^* \times S$-indexed family of maps
  $\sigma_{ATT_{s_1 \ldots s_n, s}} : ATT1_{s_1 \ldots s_n, s} \rightarrow ATT2_{\sigma_s(s_1) \ldots \sigma_s(s_n), \sigma_s(s)}$;
- an S^*-indexed family of maps $\sigma_{EVT_{s_1 \ldots s_n}} : EVT1_{s_1 \ldots s_n} \rightarrow EVT2_{\sigma_s(s_1) \ldots \sigma_s(s_n)}$. ■

In the case of specialization, for instance, σ_s, $\sigma_{ATT_{s_1 \ldots s_n, s}}$, $\sigma_{EVT_{s_1 \ldots s_n}}$ are inclusions. A signature morphism is used to translate formulas. Let φ be a formula over Φ_1; then we write $\sigma(\varphi)$ for the translation along the template signature morphism.

Definition 3.7 (Template Specification Morphism)
A *template specification morphism* σ: $(\Phi_1, Ax_1) \rightarrow (\Phi_2, Ax_2)$ is a template signature morphism such that $\sigma(ax) \in Ax_2$ for every $ax \in Ax_1$. ■

3.2 Induced OSL Signatures

Given a primitive class specification in TROLL, the induced object template signature in OSL is obtained from the attribute and event declarations in a straightforward way. For instance, consider the class specification in Example 2.1. It induces the object template signature $\Phi_{Account}$ where:

- $AS = \{Account\}$;
- $\alpha = Account$;
- $ATT_{integer} = \{balance\}$, $ATT_{natural} = \{limit\}$;
- $EVT_{natural} = \{open, deposit, withdraw\}$.

Given a specialization class specification in TROLL, the induced object template signature in OSL includes the signature induced by the superclass specification plus the induced signature of the subclass specification. For instance consider the class specification in Example 2.2. It induces the object template signature $\Phi_{SavingsAccount}$ where:

- $AS = \{Account, SavingsAccount\}$;
- $\alpha = SavingsAccount$;
- $ATT_{integer} = \{balance\}$,$ATT_{natural} = \{limit, interestRate\}$;
- $EVT_{\epsilon} = \{accumulateInterest\}$,
  $EVT_{natural} = \{open, deposit, withdraw\}$.

Note that in the template signature morphism σ :$\Phi_{Account} \rightarrow \Phi_{SavingsAccount}$ the maps σ_s,$\sigma_{ATT_{integer}}$,$\sigma_{EVT_{\epsilon}}$, $\sigma_{EVT_{integer}}$ and $\sigma_{EVT_{natural}}$ are inclusions.

The class specification in Example 2.3 induces the object template signature Φ_{Bank} where:

- $AS = \{Acc_1, \ldots, Acc_n, Bank\}$;
- $\alpha = Bank$;
- $ATT_{integer} = \{balance_i : 1 \leq i \leq n\}$
  $ATT_{natural} = \{limit_i : 1 \leq i \leq n\}$;
- $EVT_{natural} = \{open_i, deposit_i, withdraw_i : 1 \leq i \leq n\}$,
  $EVT_{Acc_i,Acc_j,natural} = \{tranfer\}$ $(1 \leq i, j \leq n, i \neq j)$.

In this case there are n template signature morphisms σ^i: $\Phi_{Account} \rightarrow \Phi_{Bank}$ $(1 \leq i \leq n)$ where:

- $\sigma^i_s(Acount) = Acc_i$;
- $\sigma^i_{ATT_{natural}}(limit) = limit_i$,
  $\sigma^i_{ATT_{integer}}(balance) = balance_i$;
- $\sigma^i_{EVT_{natural}}(open) = open_i$,
  $\sigma^i_{EVT_{natural}}(deposit) = deposit_i$,
  $\sigma^i_{EVT_{natural}}(withdraw) = withdraw_i$.

3.3 Induced OSL Specifications

Let $\Phi = (AS, \alpha, ATT, EVT)$ be the object template signature in OSL induced by a class specification in Troll. The axioms of the induced OSL specification are translations into OSL of Troll assertions, universal axioms of OSL (see section 3.2) and implicitly generated axioms which reflect the basic assumptions made for Troll. We have to translate four kinds of Troll assertions, namely, valuations, permissions, obligations and interactions:

- The valuation rules for an event `e(t`$_1$`,...,t`$_n$`)`, `[e(t`$_1$`,...,t`$_n$`)]a`$_1$`=t´`$_1$,..., `[e(t`$_1$`,...,t`$_n$`)]a`$_m$`=t´`$_m$, are translated into one OSL formula schema, designated in the sequel by *valuation schema*,

$$\mathbf{G}(\triangledown e(t_1, ..., t_n) \wedge \mathbf{f}^{a_1,...,a_m}_{t'_1,...,t'_m} \Rightarrow \mathbf{X}\,\mathbf{f})$$

  where $\mathbf{f}$ is a formula variable taking values in a set which is said the *scope of the schema*. Given a formula φ in the scope of the schema, the φ-instance of the schema is $\triangledown e(t_1, ..., t_n) \wedge \varphi^{a_1,...,a_m}_{t'_1,...,t'_m} \Rightarrow \mathbf{X}\,\varphi$ where $\varphi^{a_1,...,a_m}_{t'_1,...,t'_m}$ is the formula obtained from φ by replacing all occurrences of a_i by t'_i.

- Permissions of the form `{c}e(t`$_1$`,...,t`$_n$`)` are translated into OSL formulae of the form

$$\mathbf{G}(\diamond e(t_1, ..., t_n) \Rightarrow c)$$

- Obligations of the form `e(t`$_1$`,...,t`$_n$`)` are translated into OSL formulae of the form

$$\mathbf{F}\,\triangledown e(t_1, ..., t_n)$$

- Interactions like `e(t`$_1$`,...,t`$_n$`)>>e´(t´`$_1$`,...,t´`$_m$`)` are translated into OSL formulae of the form

$$\mathbf{G}(\triangledown e(t_1, ..., t_n) \Rightarrow \triangledown e'(t'_1, ..., t'_m))$$

  and interactions like `e(t`$_1$`,...,t`$_n$`)<>e´(t´`$_1$`,...,t´`$_n$`)` are translated into OSL formulae of the form

$$\mathbf{G}(\triangledown e(t_1, ..., t_n) \Leftrightarrow \triangledown e'(t'_1, ..., t'_m))$$

The implicit axioms are related with the declaration of birth events and with conflicting event effects. Let B be the set of event symbols corresponding to the events declared as birth events in the TROLL specification.

The declaration of birth events imposes the following axioms, where *born* is an implicit boolean attribute, $b \in B$, $e \notin B$,$t_i, t'_j \in T_\Phi(X)$, $1 \leq i \leq n$, $1 \leq j \leq n$ and $n, m \geq 0$:

$$\begin{array}{ll} \mathbf{G}(\triangledown b(t_1, ..., t_n) \Leftrightarrow \mathbf{X}(\mathbf{G}\,born)) & (1) \\ \mathbf{G}(\diamond b(t_1, ..., t_n)\;\mathbf{U}\;born) & (2) \\ \mathbf{G}(born \Rightarrow (\neg\diamond b(t_1, ..., t_n))) & (3) \\ \mathbf{G}(\neg\diamond e(t'_1, ..., t'_m)\;\mathbf{U}\;born) & (4) \end{array}$$

Two events have conflicting effects whenever they change some attribute in different ways. The simultaneous occurrence of several events is possible, but attribute values are deterministic. Formulae as the following prevent the simultaneous occurrence of events having conflicting effects:

$$\mathbf{G}(\neg(\triangledown e_1 \wedge \triangledown e_2))$$

The next three examples present the OSL specifications induced by the TROLL specification introduced above.

Example 3.8 The specification `Account` (Example 2.1) is translated into the template specification $spec_{Account} = (\Phi_{Account}, Ax_{Account})$ where $\Phi_{Account}$ is the object template signature described above and $Ax_{Account}$ is the set

$$\begin{array}{l}
\{\mathbf{G}(\triangledown open(m) \wedge \mathbf{f}^{limit,balance}_{m,0} \Rightarrow \mathbf{X}\,\mathbf{f}),\\
\mathbf{G}(\triangledown deposit(m) \wedge \mathbf{f}^{balance}_{balance+m} \Rightarrow \mathbf{X}\,\mathbf{f}),\\
\mathbf{G}(\triangledown withdraw(m) \wedge \mathbf{f}^{balance}_{balance-m} \Rightarrow \mathbf{X}\,\mathbf{f}),\\
\mathbf{G}(\diamond withdraw(m) \Rightarrow (balance - m \geq -limit)),\\
\mathbf{G}(\diamond withdraw(m) \Rightarrow m \leq 2000)\}
\end{array}$$

where the scope of the schemata is the set of state formulae in $\mathcal{TL}_{\Phi_{Account}}(X)$.

Example 3.9 The specification `SavingsAccount` (Example 2.2) is translated into the template specification $spec_{SavingsAccount} = (\Phi_{SavingsAccount}, Ax_{SavingsAccount})$ where $\Phi_{SavingsAccount}$ is the object template signature described above and $Ax_{SavingsAccount}$ is the set

$$\begin{array}{l}
Ax_{Account} \bigcup\\
\{\mathbf{G}(\diamond open(m) \Rightarrow m = 0),\\
\mathbf{F}\,\triangledown accumulateInterest,\\
\mathbf{G}(\triangledown accumulateInterest \Rightarrow \triangledown deposit(InterestRate * balance/100))\}
\end{array}$$

Example 3.10 The specification `Bank` (Example 2.3) is translated into the template specification $spec_{Bank} = (\Phi_{Bank}, Ax_{Bank})$ where Φ_{Bank} is the object template signature described above and Ax_{Bank} is the set*

$$\begin{array}{l}
\{\sigma^i(ax) : 1 \leq i \leq n, ax \in Ax_{Account}\} \bigcup\\
\{\mathbf{G}(\triangledown transfer(acc_i, acc_j, m) \Rightarrow \triangledown withdraw_i(m)),\\
\mathbf{G}(\triangledown transfer(acc_i, acc_j, m) \Rightarrow \triangledown deposit_j(m)) : 1 \geq i, j \geq n, i \neq j\}.
\end{array}$$

where if ax is a valuation schema in $Ax_{Account}$ then the scope of $\sigma^i(ax)$ is the set of state formulae in $\sigma^i(\mathcal{TL}_{\Phi_{Account}}(X))$.

*considering $\{acc_i\}$ as carrier set of sort Acc_i, $1 \leq i \leq n$

3.4 Induced Assertions

In section 2.2 we discussed how to state assertions about a TROLL object class specification. Those assertions express properties that should hold along the object's life. In order to verify those assertions over the specification we have to translate them into OSL. Each assertion induces, in a straightforward way, an object template formula over the object template signature induced by the TROLL object class specification.

Example 3.11 The assertion `INV1` presented in Example 2.4 induces the following object template formula over $\Phi_{Account}$:

$$\mathbf{G}(born \Rightarrow (balance \geq -limit))$$

Example 3.12 The assertion `INV2` presented in Example 2.5 induces the following object template formula over $\Phi_{Account}$:

$$\mathbf{G}(born \Rightarrow (balance = n \Rightarrow \mathbf{X}(balance \geq n - 2000)))$$

Example 3.13 The assertion `LIV` presented in Example 2.6 induces the following object template formula over $\Phi_{SavingsAccount}$:

$$\mathbf{F}(\exists m \triangledown deposit(m))$$

Example 3.14 The assertion `INV3` presented in Example 2.7 induces the following object template formula over Φ_{Bank}, where $\varphi \equiv balance_i + balance_j = n$:

$$\mathbf{G}(\triangledown transfer(acc_i, acc_j, m) \Rightarrow (\varphi \Rightarrow \mathbf{X}\,\varphi))$$

In general we allow the following kinds of assertions:

- assertions of class invariant induce formulae of the kind $\mathbf{G}\,\varphi$ where φ is a state formula;
- assertions of class transition invariant induce formulae of the kind $\mathbf{G}(\psi \Rightarrow (\varphi_1 \Rightarrow \mathbf{X}\,\varphi_2))$ where φ_1, φ_2, ψ are state formulae;
- assertions of class liveness induce formulae of the kind $\mathbf{F}\,\varphi$ where φ are state formulae.

4 Proving Assertions

4.1 Tableau systems

In order to prove assertions about an object specification we use tableau systems [BM77]. Tableau systems are refutation systems where each tableau is a labeled tree whose labels are finite sets of formulae. Given a formula, a tableau is built in order to find a counterexample for it. The root is labeled with the negation of the formula and the tableau is built following construction rules related to connectives and operators.

The construction rules of a tableau system for first order logic are illustrated in Figure 1 where π is a finite set of first order formulae, $\varphi, \varphi_1, \varphi_2$ are first order formulae, $\varphi(x \mid t)$ is obtained from $\forall x \varphi$ by substituting t for x [†], t is a term and y is a variable [‡]. Note that in rule $\forall$, t can be any term since the variable x is universally quantified whereas in rule $\neg\forall$ we can only assure the existence of some term t' such that $\neg\varphi(x \mid t')$ holds and so in this rule we have to substitute a variable y for x. Node labels are sets but it is usual to depict them without braces and to use a comma for representing set union. The rules presented are related to the primitive connectives and quantifiers considered. From them we can easily derive rules for the connectives and quantifiers introduced as abbreviations.

If the tableau fails to find a counterexample, i.e., if all the attempts (tree branches) lead to an absurd, this means that the given formula is valid. In the context of first order logic a tableau branch is considered an *absurd* if the union of the labels of its nodes contains an *absurd set*, i.e., a set $\{\varphi, \neg\varphi\}$ where φ is some first order formula. However, due to an universal OSL axiom, in this context we can consider also as absurd any branch containing $\{\triangledown e, \neg\diamond e\}$ for some event e. A tableau is said to be *closed* iff all its branches are absurd.

Figure 2 presents a closed tableau for $\varphi \Rightarrow (\psi \Rightarrow \varphi)$. The attempt to find a counter example for $\varphi \Rightarrow (\psi \Rightarrow \varphi)$ failed because we reach the absurd $\{\varphi, \neg\varphi\}$ within the (only) tableau branch. As a consequence, we proved that $\varphi \Rightarrow (\psi \Rightarrow \varphi)$ is a valid formula. In this example (and in the sequel) we use the symbols X and $\gg$ to mark, respectively, an absurd branch and formulae of an absurd set.

In particular, it is possible to prove that some formula ψ is consequence of the set of formulae $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ by proving the validity of $(\varphi_1 \wedge \ldots \wedge \varphi_n) \Rightarrow \psi$, i.e., building a closed tableau with root label $\{\neg((\varphi_1 \wedge \ldots \wedge \varphi_n) \Rightarrow \psi)\}$, or equivalently, with root label $\{\varphi_1, \ldots, \varphi_n, \neg\psi\}$.

4.2 Proving invariants

This section deals with verification of assertions of class invariant and transition invariant. As mentioned above, they are translated, in the first case, into formulae of

[†] with the usual restrictions

[‡] with the proviso that y must not be free in any formula of the branch which is being extended

the kind $\mathbf{G}\,\varphi$ and, in the second case, into formulae of the kind $\mathbf{G}(\psi \Rightarrow (\varphi_1 \Rightarrow \mathbf{X}\,\varphi_2))$ ($\varphi_1, \varphi_2, \psi$ state formulae).

The translation $\mathbf{G}\,\varphi$ of an invariant assertion is satisfied by a life cycle of an object (linear sequence of states) iff each state satisfies φ. We are mainly interested in the case where $\varphi \equiv born \Rightarrow \varphi'$. It is straightforward to see that, taking into account the implicit axioms introduced in section 3.3, if φ' is established by each birth event and preserved by each update event then every state satisfies φ.

In what concerns transition invariants, it is also straightforward to see that $\mathbf{G}(\psi \Rightarrow (\varphi_1 \Rightarrow \mathbf{X}\,\varphi_2))$ is satisfied by a life cycle of an object if, whenever ψ and φ_1 are true, the occurrence of each event establishes φ_2. Therefore, the proof of these kinds of assertions involves several tableaux as, for each event, we are going to build a tableau that will prove or disprove that the event establishes/preserves the state formula.

Arithmetic or equational reasoning can also lead to absurd sets. For instance, $\{\neg(n > n)\}$ and $\{x \geq y, x + n < y\}$, when $n \in I\!N$, are absurd sets. We detect these absurd sets through a suitable equational and numeric reasoning module following [NO79, NO80]. In the sequel, when depicting a tableau, we use the symbol X_a to mark absurd branches resulting from arithmetic or equational absurd sets and X_l in the other case.

An event e establishes φ if, in every state, if e occurs then in the next state φ holds. As consequence, to prove that e establishes φ, we have to prove that $\triangledown e \Rightarrow \mathbf{X}\,\varphi$ follows from the object template specification axioms, on every state. We build a closed tableau with $\{\varphi_1, \ldots, \varphi_n, \neg(\triangledown e \Rightarrow \mathbf{X}\,\varphi)\}$ as root label, where $\varphi_1, \ldots, \varphi_n$ come from axioms of the object template specification. For simplicity reasons, we usually omit the axioms that are not relevant for the case at hand. In a similar way, to prove that an event e establishes φ from ψ we build a closed tableau with $\{\varphi_1, \ldots, \varphi_n, \neg(\triangledown e \wedge \psi \Rightarrow \mathbf{X}\,\varphi)\}$ as root label.

An event e preserves φ if, in every state, if e occurs and φ holds, then, on the next state, φ still hold. So, to prove that e preserves φ, we have to prove that $(\triangledown e \wedge \varphi) \Rightarrow \mathbf{X}\,\varphi$ follows, on every state, from the object template specification axioms. The proof is similar to the previous case.

Lets illustrate how it works through some examples.

Example 4.1 Consider the object template formula $\mathbf{G}(born \Rightarrow (balance \geq -limit))$ over $\Phi_{Account}$ (Example 3.11). In order to prove that $born \Rightarrow (balance \geq -limit)$ is true in every state of a life cycle we build three tableaux.

- To prove that $\varphi \equiv balance \geq -limit$ is established by the birth event, we build the tableau (Figure 3) with root label

$$\{\triangledown open(m) \wedge \varphi_{m,0}^{limit,balance} \Rightarrow \mathbf{X}\,\varphi,$$

$$\neg(\triangledown open(m) \Rightarrow \mathbf{X}\,\varphi)\}.$$

In the construction of this tableau we have applied the non-branching rule $\neg \Rightarrow$ and the branching rules $\Rightarrow$ and $\neg\wedge$. The tableau has three branches. The first one is an absurd because it includes the formulae $\triangledown open$ and $\neg\triangledown open$ (marked with $>>_1$). Using arithmetical reasoning we can conclude that the second branch is also an absurd since it contains the formula $\neg(0 \geq -m)$ (marked with $>>_3$) and m is natural number. Finally, the third branch is again an absurd because it includes the formulae $\neg\,\mathbf{X}(balance \geq -limit)$ and $\mathbf{X}(balance \geq -limit)$ (marked with $>>_3$).

- To prove that $\varphi \equiv balance \geq -limit$ is preserved by the event $withdraw(m)$, we build the tableau (Figure 4) with root label

$$\{\triangledown withdraw(m) \wedge \varphi^{balance}_{balance-m} \Rightarrow \mathbf{X}\,\varphi,$$

$$\triangledown withdraw(m) \Rightarrow balance - m \geq limit,$$

$$\neg(\triangledown withdraw(m) \wedge \varphi \Rightarrow \mathbf{X}\,\varphi)\}$$

- To prove that $\varphi \equiv balance \geq -limit$ is preserved by the event $deposit(m)$, we build the tableau (Figure 5) with root label

$$\{\triangledown deposit(m) \wedge \varphi^{balance}_{balance+m} \Rightarrow \mathbf{X}\,\varphi,$$

$$\neg(\triangledown deposit(m) \wedge \varphi \Rightarrow \mathbf{X}\,\varphi)\}.$$

We do not need more than those tableaux because due to locality no more events can occur. As the three tableaux are closed we can conclude that the birth event establishes $balance \geq -limit$ and the two update events preserve $balance \geq -limit$ and, therefore, $born \Rightarrow (balance \geq -limit)$ is true in every state of the life cycle of the object.

Example 4.2 Consider the object template formula $\mathbf{G}(born \Rightarrow (balance = n \Rightarrow \mathbf{X}(balance \geq n - 2000)))$ over $\Phi_{Account}$ (Example 3.12). In order to prove that $birth \Rightarrow (balance = n \Rightarrow \mathbf{X}(balance \geq n - 2000))$ is true in every state of a life cycle we look at two cases:

- to prove that $\varphi_2 \equiv balance \geq n - 2000$ is established by the event $withdraw(m)$ whenever $\varphi_1 \equiv balance = n$ holds, we consider the tableau (Figure 6) with root label

$$\{\triangledown withdraw(m) \wedge \varphi_2\,{}^{balance}_{balance-m} \Rightarrow \mathbf{X}\,\varphi_2,$$

$$\diamond withdraw(m) \Rightarrow m \geq 2000,$$

$$\neg(\triangledown withdraw(m) \wedge \varphi_1 \Rightarrow \mathbf{X}\,\varphi_2)\}$$

- to prove that $\varphi_2 \equiv balance \geq n - 2000$ is established by the event $deposit(m)$ whenever $\varphi_1 \equiv balance = n$ holds, we consider the tableau (Figure 7) with root label

$$\{\triangledown deposit(m) \wedge \varphi_2\, {}^{balance}_{balance+m} \Rightarrow \mathbf{X}\, \varphi_2,$$
$$\neg(\triangledown deposit(m) \wedge \varphi_1 \Rightarrow \mathbf{X}\, \varphi_2)\}$$

As the above tableaux are closed we can conclude that after the object's birth if in some state $balance = n$ then in the following state $balance \geq n - 2000$. Note that the birth event is not relevant to this proof because the value of $balance$ is not defined until the occurrence of $open$ and so it cannot be equal to n when the birth event occurs.

Example 4.3 Consider the object template formula over Φ_{Bank} (Example 2.7)

$$\mathbf{G}(\triangledown transfer(acc_i, acc_j, m) \Rightarrow (\varphi \Rightarrow \mathbf{X}\, \varphi))$$

where $\varphi \equiv balance_i + balance_j = n$. In order to prove this assertion we take advantage of a meta-rule similar to the one presented in [SS94]. The meta-rule is the following:

$$\mathbf{G}((\bigwedge_{i=1}^{n}(\triangledown e_i \wedge f^{\sigma^1(a^1_1)\ldots\sigma^1(a^1_{p_1})\ldots\sigma^n(a^n_1)\ldots\sigma^n(a^n_{pn})}_{\sigma^1(t^1_1)\ldots\sigma^1(t^1_{p_1})\ldots\sigma^n(t^n_1)\ldots\sigma^n(t^n_{pn})})) \Rightarrow \mathbf{X}\, f)$$

provided that

$$\mathbf{G}((\triangledown e_i \wedge f_i\, {}^{a^i_1\ldots a^i_{p_i}}_{t^i_1\ldots t^i_{p_i}}) \Rightarrow \mathbf{X}\, f_i)$$

where $1 \leq i \leq n$, $\sigma^i : \Phi_i \to \Phi$ is a template signature morphism, the scope of the first valuation schema is the set of state formulae in $\bigcup_{i=1}^{n} \sigma^i(\mathcal{TL}_{\Phi_i}(X))$ and the scope of the second schema is the set of state formulae in $\sigma^i\mathcal{TL}_{\Phi_i}(X)$.Therefore, in this case, we have

$$\triangledown withdraw_i(m) \wedge \triangledown deposit_j(m) \wedge f^{balance_i\ \ balance_j}_{balance_i - m\ \ balance_j + m} \Rightarrow \mathbf{X}\, f)$$

and since $\mathbf{G}(\triangledown transfer(acc_i, acc_j, m) \Rightarrow (\triangledown withdraw_i(m) \wedge \triangledown deposit_j(m))$, to prove that $\mathbf{G}(\triangledown transfer(acc_i, acc_j, m) \Rightarrow (\varphi \Rightarrow \mathbf{X}\, \varphi))$ we build the tableau (see Figure 8) with root label

$$\{\triangledown withdraw_i(m) \wedge \triangledown deposit_j(m) \wedge \varphi^{balance_i\ \ balance_j}_{balance_i - m\ \ balance_j + m} \Rightarrow \mathbf{X}\, \varphi),$$

$$\triangledown withdraw_i(m) \wedge \triangledown deposit_j(m), \neg(\varphi \Rightarrow \mathbf{X}\, \varphi)\}$$

As the tableau is closed, we conclude that $balance_i + balance_j = n$ is preserved whenever $transfer(acc_i, acc_j, m)$ occurs.

4.3 Proving liveness

In order to prove liveness properties the tableau system for first order logic is not enough since we have the need to deal explicitly with temporal operators. We will use *prefixed tableaux systems* where tableaux are similar to ones presented above but node labels are now sets of prefixed formulae. A prefixed formula has the general form $p : \varphi$, where p (the *prefix*) is an element of a suitable set (of prefixes) and φ is an OSL formula. These prefixes are a way to represent, syntactically, the states where we want formulae to be true. These tableaux systems are inspired on the prefixed tableau systems for propositional modal logic [Fit83].

The set of prefixes is $W = \{w_0, w_1, ...\}$. Although we use the subscripts 0, 1,..., we do not assume that they imply any particular order on the states. Actually, the syntactical representation of the linear order on the states relies on a new kind of formulae. These formulae are sequences in W^+ or $W(W \bigcup \{\bullet w : w \in W\})^+$. A formula $w_{i_1} \cdots w_{i_n} \in W^+$ will be satisfied when each w_{i_k} represents a state that is in the future w.r.t. to the one represented by $w_{i_{k-1}}$. A formula $w_{i_1} \cdots w_{i_{k-1}} \bullet w_{i_k} \cdots w_{i_n} \in W(W \bigcup \{\bullet w : w \in W\})^+$ will be satisfied if the same condition holds and, additionally, the state represented by w_{i_k} is the state immediately after the one represented by $w_{i_{k-1}}$.

Construction rules related with the connectives $\neg$, $\Rightarrow$ and the quantifier $\forall$ are similar to the ones introduced in the previous section: when applying one of these rules to some prefixed formula $p : \varphi$, the formulae introduced are as above and we prefix them with the prefix p. The new construction rules, related to the temporal operators are depicted in (Figure 9).

In this context, the notions of absurd set and absurd branch are similar to the presented above. In Figure 10 we present a very simple tableau that illustrates the use of the rules. We can conclude from this closed tableau that in every linear sequence of states $\mathbf{F}\,\psi$ follows from $\mathbf{G}(\varphi \Rightarrow \psi)$ and $\mathbf{F}\,\varphi$.

In this case we only use one tableau to prove an assertion. The label of the root of this tableau will include all the axioms of the class description or class descriptions involved (as above, we can also omit the ones that are not relevant for the proof at hand). These tableaux tend to be more complex than the others presented above. The next example illustrates these concepts in the context of a specialization.

Example 4.4 Consider the object template formula $\mathbf{F}\,\exists m \triangledown deposit(m)$ over the signature $\Phi_{SavingsAccount}$ (Example 2.7). In order to prove that, in every life cycle, $\exists m \triangledown deposit(m)$ holds in some state we build a closed tableau (Figure 11) having as root label

$$\{\mathbf{G} \triangledown accumulate\_interest \Rightarrow \triangledown deposit(\frac{interestRate * Balance}{100}),$$

$$\mathbf{F} \triangledown accumulate\_interest, \neg(\mathbf{F} \exists m\ \triangledown deposit(m))\}.$$

5 Conclusions

Starting with the basic language features of object specification languages, we have shown how these language constructs can be mapped onto temporal logic formulae. This establishes an axiomatic semantics of object-oriented languages like TROLL being the starting point of formally proving object properties. The used temporal logic dialect OSL supports directly object-oriented concepts like encapsulation and inheritance. Therefore, these concepts can be used to make proofs feasible for example using the locality principle enabling smaller proofs. The use of tableau techniques has been shown to lead to small und understandable proofs.

The presented techniques were implemented in a project at IST Lisbon. Our theorem prover was implemented in PROLOG in order to take advantage of the facilities offered by the language [GA93]. The program consists basically of five modules: (1) the data entry module, (2) the logical module (where the tableaux are built and the logic absurds are detected), (3) the arithmetic module (based on the simplex algorithm), (4) the equational module and (5) the cooperative module (establishing a link between the modules (2),(3) and (4)).

Acknowledgments

We would like to thank A. Sernadas for many fruitful discussions as well as other colleagues from the Esprit BRA Working Group IS-CORE.

References

[BM77] J. Bell and M. Machover. *A Course in Mathematical Logic.* North-Holland, 1977.

[Boo94] G. Booch. *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings, Redwood City, CA, 2nd edition, 1994.

[CAB+94] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremes. *Object-Oriented Development – The Fusion Method.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[Eme90] E. Emerson. Temporal and Modal Logic. J. van Leeuwen, editor, *Handbook of Theoretical Computer Science B: Formal Models and Semantics*, pages 995–1072, Elsevier, 1990.

[Fit83] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, Reidel, 1983.

[FSMS90] J. Fiadeiro, C. Sernadas, T. Maibaum, and G. Saake. Proof-Theoretic Semantics of Object-Oriented Specification Constructs. In R. Meersman,

W. Kent, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design and Construction (Proc. IFIP WG 2.6 Working Conference DS-4)*, pages 243–284, Windermere (UK), 1990. North-Holland, Amsterdam, 1991.

[FM92] J. Fiadeiro and T. Maibaum. Temporal theories as modularisation Units for Concurrent System Specification. In *Formal Aspects of Computing 4(3)*, pages 239–272, 1992.

[GA93] J.Gomes and M.-J. Apolinário. *Demonstração Automática.* Diploma Thesis (in Portuguese). Instituto Superior Técnico, 1993.

[HJS92] T. Hartmann, R. Jungclaus, and G. Saake. Aggregation in a Behavior Oriented Object Model. In O. Lehrmann Madsen, editor, *ECOOP'92 European Conference on Object-Oriented Programming (Proceedings)*, pages 57–77, Utrecht (NL), 1992. LNCS 615, Springer-Verlag, Berlin, 1992.

[Jac92] I. Jacobson. *Object-Oriented Software Engineering.* Addison-Wesley, Reading, MA, 1992.

[JSHS91] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Object-Oriented Specification of Information Systems: The TROLL Language. Informatik-Bericht 91-04, TU Braunschweig, 1991.

[JSHS95] R. Jungclaus; G. Saake; T. Hartmann; C. Sernadas: TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 1995. *To appear.*

[JSS91] R. Jungclaus, G. Saake, and C. Sernadas. Formal Specification of Object Systems. In S. Abramsky and T. Maibaum, editors, *Proc. TAPSOFT'91 Vol. 2*, pages 60–82, Brighton, 1991. LNCS 494, Springer-Verlag, Berlin, 1991.

[Jun93] R. Jungclaus. *Modeling of Dynamic Object Systems.* Vieweg Verlag, Wiesbaden/Braunschweig, 1993.

[KS91] G. Kappel and M. Schrefl. Object/Behavior Diagrams. In *Proc. Int. Conf. on Data Engineering*, pages 530–539, Kobe, 1991. IEEE Computer Society Press, 1991.

[MP89] Z. Manna and A. Pnueli. The Anchored Version of the Temporal Framework. In W.-P. deRoever and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models of Concurrency*,pages 201–284. LNCS 354, Springer,1989.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems —Vol. 1: Specification.* Springer-Verlag, New York, 1992.

[NO79] G. Nelson and D. Oppen. Simplification by Cooperating Decision Procedures. In *ACM Transactions on Programming Languages and Systems, 1 (2)*, pages 245–257, 1979.

[NO80] G. Nelson and D. Oppen. Fast Decision Procedures based on Congruence Closure. In *Journal of the Association for Computing Machinery, 27 (2),*, pages 356–364, 1980.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs, NJ, 1991.

[Saa93] G. Saake. *Objektorientierte Spezifikation von Informationssystemen.* Teubner Verlag, Leipzig, 1993. *In German.*

[SE90] A. Sernadas and H.-D. Ehrich. What Is an Object, After All? In R. Meersman, W. Kent, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design and Construction (Proc. IFIP WG 2.6 Working Conference DS-4)*, pages 39–70, Windermere (UK), 1990. North-Holland, Amsterdam, 1991.

[SF91] C. Sernadas and J. Fiadeiro. Towards Object-Oriented Conceptual Modelling. *Data & Knowledge Engineering*, 6:479–508, 1991.

[SM92] S. Shlaer and S. J. Mellor. *Object Lifecycles: Modeling the World in States.* Yourdon Press/Prentice Hall, Englewood Cliffs, NJ, 1992.

[SSC92] A. Sernadas, C. Sernadas, and J. F. Costa. Object Specification Logic. Research Report, Depart. of Mathematics, Instituto Superior Técnico, Av. Rovisco Pais, 1096 Lisboa Codex, Portugal, 1992. *Journal of Logic and Computation*(in print).

[SS94] A. Sernadas and C. Sernadas. Object Certification. Research Report, Depart. of Mathematics, Instituto Superior Técnico, Av. Rovisco Pais, 1096 Lisboa Codex, Portugal, 1992.

[SSE87] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-Oriented Specification of Databases: An Algebraic Approach. In P. Hammerslay, editor, *Proc. 13th Int. Conf. on Very Large Databases VLDB'87*, pages 107–116, Brighton (GB), 1987. Morgan-Kaufmann, Palo Alto, 1987.

[Wie90] R. J. Wieringa. *Algebraic Foundations for Dynamic Conceptual Models.* PhD thesis, Vrije Universiteit, Amsterdam, 1990.

Appendix

$$\begin{array}{c} \pi, \neg\neg\varphi \\ \Big|\, \neg\neg \\ \varphi \end{array} \qquad\qquad \begin{array}{ccc} & \pi, \varphi_1 \Rightarrow \varphi_2 & \\ \swarrow & \Rightarrow & \searrow \\ \neg\varphi_1 & & \varphi_2 \end{array}$$

$$\begin{array}{c} \pi, \neg(\varphi_1 \Rightarrow \varphi_2) \\ \Big|\, \neg \Rightarrow \\ \varphi_1 \\ \neg\varphi_2 \end{array} \qquad \begin{array}{c} \pi, \forall x\, \varphi \\ \Big|\, \forall \\ \varphi(x \mid t) \end{array} \qquad \begin{array}{c} \pi, \neg\forall x\, \varphi \\ \Big|\, \neg\forall \\ \neg\varphi(x \mid y) \end{array}$$

Figure 1: Tableau Construction Rules

$$\begin{array}{rc} & \neg(\varphi \Rightarrow (\psi \Rightarrow \varphi)) \\ & \Big|\, \neg \Rightarrow \\ \gg & \varphi \\ & \neg(\psi \Rightarrow \varphi) \\ & \Big|\, \neg \Rightarrow \\ & \psi \\ \gg & \neg\varphi \\ & \mathsf{X} \end{array}$$

Figure 2: A closed tableau for $\varphi \Rightarrow (\psi \Rightarrow \varphi)$

$$\triangledown open(m) \wedge 0 \geq -m \Rightarrow \mathbf{X}(balance \geq -limit)$$
$$\neg(\triangledown open(m) \Rightarrow \mathbf{X}(balance \geq -limit))$$

$$\neg \Rightarrow$$

$$\gg_1 \quad \triangledown open(m)$$
$$\gg_2 \quad \neg\mathbf{X}(balance \geq -limit)$$

$$\Rightarrow$$

$$\neg(\triangledown open(m) \wedge 0 \geq -m) \qquad\qquad \gg_2 \ \mathbf{X}(balance \geq -limit)$$
$$\mathsf{X}_l$$

$$\neg\wedge$$

$$\gg_1 \ \neg\triangledown open(m) \qquad \gg_3 \ \neg(0 \geq -m)$$
$$\mathsf{X}_l \qquad\qquad \mathsf{X}_a$$

Figure 3: Tableau for Proving Assertion with Birth Event

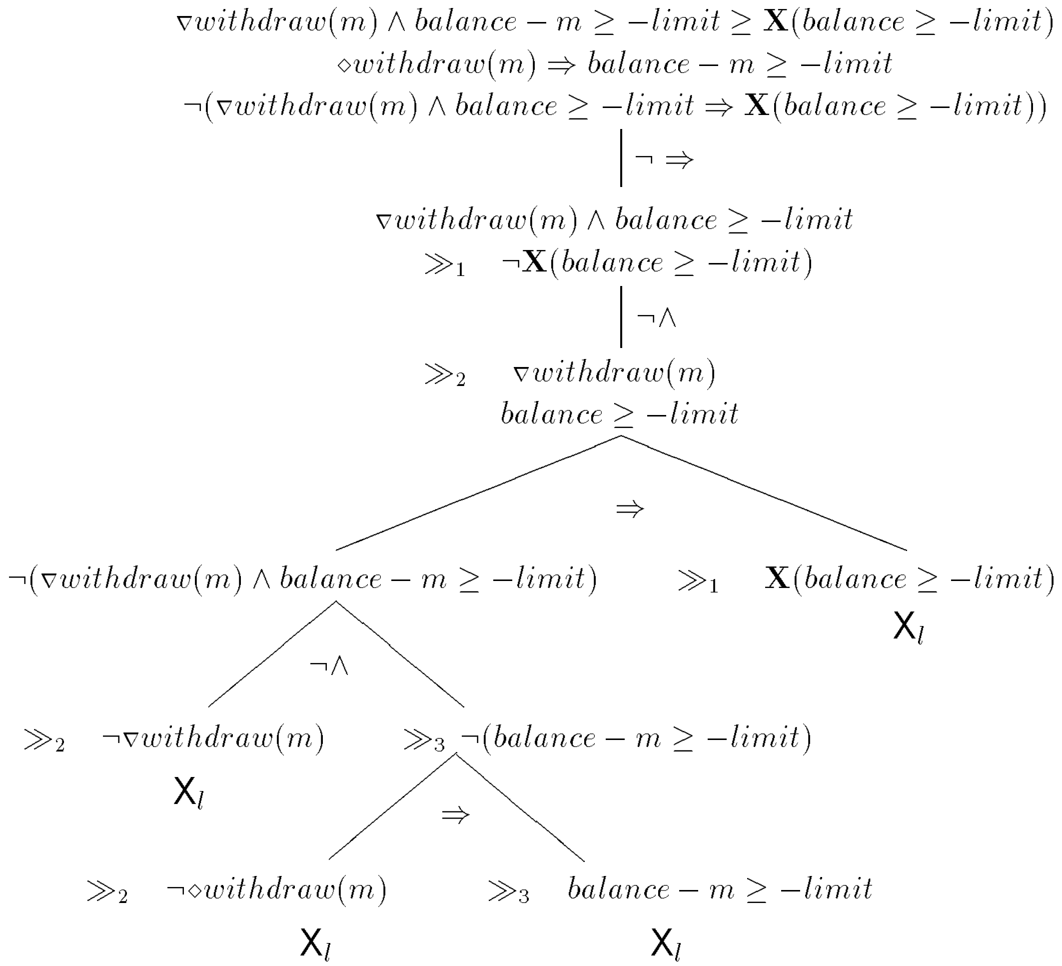

Figure 4: Tableau for Proving Assertion `INV1` with $withdraw(m)$

$$\triangledown deposit(m) \wedge balance + m \geq -limit \Rightarrow \mathbf{X}(balance \geq -limit)$$
$$\neg(\triangledown deposit(m) \wedge balance \geq -limit \Rightarrow \mathbf{X}(balance \geq -limit))$$

$$\neg \Rightarrow$$

$$\triangledown deposit(m) \wedge balance \geq -limit$$
$$\gg_1 \quad \neg\mathbf{X}(balance \geq -limit)$$

$$\wedge$$

$$\gg_2 \quad \triangledown deposit(m)$$
$$\gg_3 \quad balance \geq -limit$$

$$\Rightarrow$$

$$\neg(\triangledown deposit(m) \wedge balance + m \geq -limit) \qquad \gg_1 \quad \mathbf{X}(balance \geq -limit)$$
$$\mathsf{X}_l$$

$$\neg\wedge$$

$$\gg_2 \quad \neg\triangledown deposit(m) \qquad \gg_3 \quad \neg(balance + m \geq -limit)$$
$$\mathsf{X}_l \qquad \mathsf{X}_a$$

Figure 5: Tableau for Proving Assertion `INV1` with *deposit*(m)

$$\triangledown withdraw(m) \wedge balance - m \geq n - 2000 \geq \mathbf{X}(balance \geq n - 2000)$$
$$\diamond withdraw(m) \Rightarrow m \leq 2000$$
$$\neg(\triangledown withdraw(m) \wedge balance = n \Rightarrow \mathbf{X}(balance \geq n - 2000))$$

$\neg \Rightarrow$

$$\triangledown withdraw(m) \wedge balance = n$$
$$\gg_1 \quad \neg\mathbf{X}(balance \geq n - 2000)$$

$\wedge$

$$\gg_2 \quad \triangledown withdraw(m)$$
$$balance = n$$

$\Rightarrow$

$$\neg(\triangledown withdraw(m) \wedge balance - m \geq n - 2000) \qquad \gg_1 \quad \mathbf{X}(balance \geq n - 2000)$$
$$\mathsf{X}_l$$

$\neg\wedge$

$$\gg_2 \quad \neg\triangledown withdraw(m) \qquad \gg_3 \quad \neg(balance - m \geq n - 2000)$$
$$\mathsf{X}_l$$

$\Rightarrow$

$$\gg_2 \quad \neg\diamond withdraw(m) \qquad \gg_3 \quad m \leq 2000$$
$$\mathsf{X}_l \qquad \mathsf{X}_l$$

Figure 6: Tableau for Proving Assertion `INV2` with *withdraw*(m)

$$\triangledown deposit(m) \wedge balance + m \geq n - 2000 \Rightarrow \mathbf{X}(balance \geq n - 2000)$$
$$\neg(\triangledown deposit(m) \wedge balance = n \Rightarrow \mathbf{X}(balance \geq n - 2000))$$

$$\neg \Rightarrow$$

$$\triangledown deposit(m) \wedge balance = n$$
$$\gg_1 \quad \neg\mathbf{X}(balance \geq n - 2000)$$

$$\wedge$$

$$\gg_2 \quad \triangledown deposit(m)$$
$$\gg_3 \quad balance = n$$

$$\Rightarrow$$

$$\neg(\triangledown deposit(m) \wedge balance + m \geq n - 2000) \qquad \gg_1 \quad \mathbf{X}(balance \geq n - 2000)$$
$$\mathsf{X}_l$$

$$\neg\wedge$$

$$\gg_2 \quad \neg\triangledown deposit(m) \qquad \gg_3 \quad \neg(balance + m \geq n - 2000)$$
$$\mathsf{X}_l \qquad \mathsf{X}_a$$

Figure 7: Tableau for Proving Assertion `INV2` with $deposit(m)$

$$\gg_1 \quad \triangledown withdraw_i(m) \wedge \triangledown deposit_j(m)$$

$$(\triangledown withdraw_i(m) \wedge \triangledown deposit_j(m)) \wedge (balance_i - m + balance_j + m = n)$$
$$\Rightarrow \mathbf{X}(balance_i + balance_j = n)$$

$$\neg(balance_i + balance_j = n \Rightarrow \mathbf{X}(balance_i + balance_j = n))$$

$$\neg \Rightarrow$$

$$\gg_2 \quad balance_i + balance_j = n$$

$$\gg_3 \quad \neg\mathbf{X}(balance_i + balance_j = n)$$

$$\Rightarrow$$

$$\gg_1 \quad \neg(\triangledown withdraw_i(m) \wedge \triangledown deposit_j(m)) \qquad balance_i - m + balance_j + m = n$$
$$\Rightarrow \mathbf{X}(balance_i + balance_j = n)$$

$$\mathsf{X}_l$$

$$\Rightarrow$$

$$\gg_2 \quad \neg(balance_i - m + balance_j + m = n) \qquad \gg_3 \quad \mathbf{X}(balance_i + balance_j = n)$$

$$\mathsf{X}_a \qquad \mathsf{X}_l$$

Figure 8: Tableau for Proving Assertion `INV3`

$$\begin{array}{c} \pi, \omega_i : \mathbf{F} f \\ \Big| \; \mathbf{F} \\ \omega_i \, \omega_j \\ \omega_j : f \end{array} \qquad\qquad \begin{array}{c} \pi, \omega_i : \mathbf{X} f \\ \Big| \; \mathbf{X} \\ \omega_i \bullet \omega_j \\ \omega_j : f \end{array}$$

$$\begin{array}{c} \pi, \omega_{i_1}, \ldots, \omega_i, \ldots, \omega_j, \ldots, \omega_{i_n} \\ \omega_i : \neg \mathbf{F} f \\ \Big| \; \neg \mathbf{F} \\ \omega_j : \neg f \end{array} \qquad\qquad \begin{array}{c} \pi, \omega_{i_1}, \ldots, \omega_i \bullet \omega_j, \ldots, \omega_{i_n} \\ \omega_i : \neg \mathbf{X} f \\ \Big| \; \neg \mathbf{X} \\ \omega_j : \neg f \end{array}$$

Figure 9: Construction Rules Related to **F** and **X**

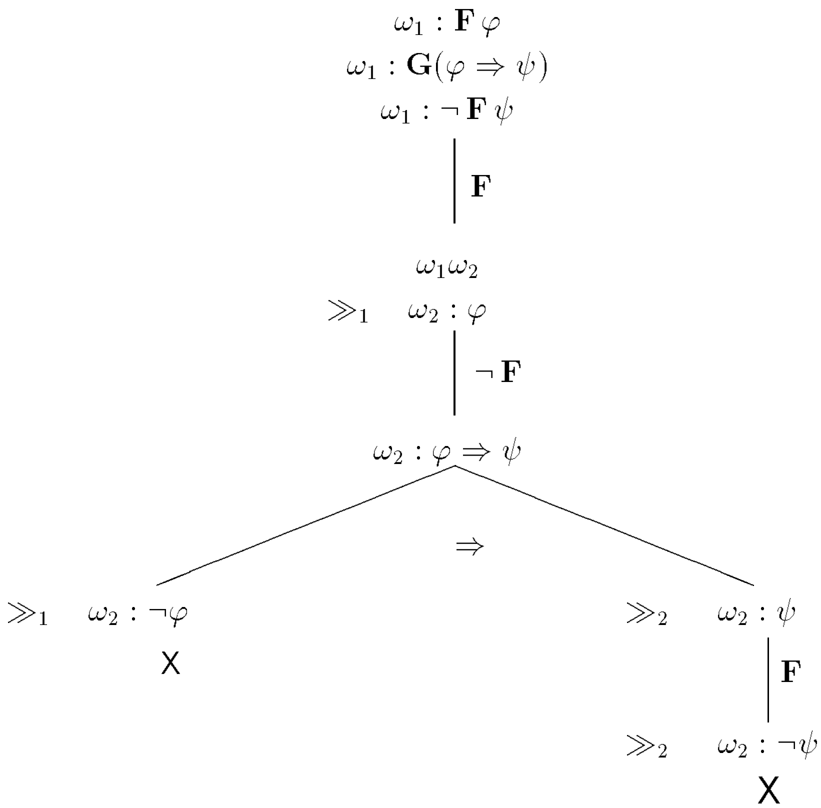

Figure 10: Closed Prefixed Tableau

$$\omega_0 : \mathbf{F} \triangledown accumulate\_interest$$

$$\omega_0 : \mathbf{G} \left(\triangledown accumulate\_interest \Rightarrow \triangledown deposit(\tfrac{interestRate*Balance}{100}) \right)$$

$$\omega_0 : \neg(\mathbf{F} \exists m \; \triangledown deposit(m))$$

$\mathbf{F}$

$\omega_0 \, \omega_1$

$\gg_1$ $\quad \omega_1 : \triangledown accumulate\_interest$

$\mathbf{G}$

$$\omega_1 : \triangledown accumulate\_interest \Rightarrow \triangledown deposit(\tfrac{interestRate*Balance}{100})$$

$\Rightarrow$

$\gg_1$ $\quad \omega_1 : \neg \triangledown accumulate\_interest$

X_l

$\gg_2$ $\quad \omega_1 : \triangledown deposit(\tfrac{interestRate*Balance}{100})$

$\neg\mathbf{F}$

$\omega_1 : \neg \exists m \; \triangledown deposit(m)$

$\neg\exists$

$\gg_2$ $\quad \omega_1 : \neg \triangledown deposit(\tfrac{interestRate*Balance}{100})$

X_l

Figure 11: Tableau for Proving Assertion `LIV`