Specification, Abstraction and Verification in a Concurrent Object-Oriented Language[1]

Ulrike Lechner and Christian Lengauer
Fakultät für Mathematik und Informatik
Universität Passau
D–94030 Passau, Germany
email: {lechner,lengauer}@fmi.uni-passau.de

We use Maude as our specification language and the modal μ-calculus as our logic. We apply to specifications in Maude a framework of abstraction and verification based on property-preserving mappings between transition systems. Firstly, we demonstrate how to employ abstraction in verification of object-oriented specifications of distributed systems. Secondly, we use this framework to find classes of properties preserved by Maude's inheritance relation.

1 Introduction

Subject to verification of object-oriented sequential programs are the data structures, the states of single objects and the global states. The more concurrency a language admits, the more important become the properties of the behavior of objects in verification. Among these properties are, e.g., which messages are accepted or refused by objects, the sequences of messages an object accepts, or whether deadlock is possible. The verification of both sequential and concurrent programs quickly becomes so complex that there is a strong need for abstraction.

The first level of abstraction is provided by Maude [4], an object-oriented specification language for concurrent systems. Particular to Maude is a very abstract way of specifying the synchronization and communication between objects. Moreover, Maude's unconventional object model enables the harmonic integration of the two concepts of concurrency and inheritance.

Abstraction is also the key concept of our approach to verification. We employ a framework of abstraction and verification of concurrent programs [1, 3]. Our mechanism for relating the states of the concrete specification and the states of the abstract specification, which is more easily verified, is the Galois connection. A *Galois connection* consists of two monotonic functions: an abstraction and a concretion function. A simulation relation between a concrete and an abstract transition system, whose states are related via a Galois connection, preserves certain properties formulated in the μ-calculus. We infer the validity of properties in the concrete specification from their validity in the abstract specification.

We verify a mutual exclusion property of a specification in Maude, and we identify a class of properties that is inheritable by Maude's inheritance relation.

[1] [2] is a longer version of this paper.

2 Maude

This section briefly introduces Maude. For a more detailed overview see [4]. Maude has two parts: one which is functional (the specification language OBJ3) and another which specifies states (so-called configurations) and state changes. The state-modeling part is specified by an operational (state transition) semantics given by *transition rules*. The state of a Maude specification is a multiset (or bag) of objects and messages and it is of sort configuration (abbreviated also by Cf). The union of two configurations C_1 and C_2 (a single object or message is also a configuration) is written $C_1\ C_2$. The sort of object identifiers is `OId`.

Let us give the specification `AIRPORT` as an example of a specification in Maude and explain it subsequently. For brevity we omit the variable declarations and the message declarations.

```
omod AIRPORT is
 uses CONFIGURATION TIME .
  class Plane | tower:OId, dest:OId, arrival:Time,
          runway:OId, flno:Nat .
   subclasses onGroundPl TakeoffPl FlyingPl < Plane .

  class Tower | free: OIdSet, nfree: OIdSet, schedule: OId .
  class DB | conf: Configuration .
  class FlightData | plane: OId, dest: OId, arrival: Time .

[T] rl (P req new flight)
       < P:onGroundPl | tower:T >
       < T:Tower | schedule:S >
       < S:DB | conf: <Y:FlightData|plane:P,dest:X,arrival:A> C >
      => < P:onGroundPl | tower:T, dest:X, arrival:A >
         < T:Tower >
         < S:DB | conf: C >
         (to T : P ready for take-off) .
endo
```

Class `Plane` has several attributes. The attributes `tower`, `dest` and `arrival` are used to store identifiers of objects to which a plane sends messages. The attributes `runway` and `flno` (flight number), contain the data of the current flight of the plane. The declaration "`subclasses` ... `< Plane`" states that classes `onGroundPl`, `TakeoffPl` and `FlyingPl` are subclasses of `Plane` and inherit thus all attributes and rules from their superclass.

The only transition rule of this specification has label `[T]` and specifies an atomic state transition involving three objects: a plane, the tower whose object identifier the plane has stored, and a database containing the data in an object of class `FlightData` of the flights. The access to the database is modeled by a pattern match: the object identifier of the plane must match the value of attribute `plane` one of the data objects in the database; the object with the matching entry is removed. Note also that we model states of objects not only by attributes but also by classes: in rule `[T]` a plane does not change its class. For class changes see the rules in Fig. 1.

Generally speaking, transition rules specify *explicit, asynchronous communication* via message passing: if a message is part of a configuration, a state transition may happen and new messages waiting to be processed in subsequent state transitions may be created as part of the resulting configuration. But, furthermore, transition rules specify synchronization between all objects at the left-hand side of a rule. To compute the resulting state of a transition, all values (object identifiers, values of attributes) of all objects and messages involved may be "exchanged" between the objects. We call this *implicit, synchronous communication.*

A *rewriting calculus* applies transition rules to the states in a system. A rule is applicable if the left-hand side of the rule can be matched in a configuration. In contrast to [4] we use a rewriting calculus that applies one rule a time.

Throughout we work in the initial term-generated model of our specifications. Let $Sp = (\Sigma, E, T)$ be a specification which contains all the basic sorts, e.g. Cf, Msg, OId. We choose the structure (A, R) as our model of Sp such that A is the initial term-generated model of (Σ, E). To interpret a formula ϕ with respect to a valuation v in (A, R) we have a standard interpretation function I that extends v to terms. The interpretation of the transition relation, specified by the transition rules is the relation R which relates elements of A_{Cf}, the carrier set of sort Cf of A. We define R_m as the subset of R in which the transitions are labeled by m, i.e., are "caused" by a rule processing a message m.

3 The modal μ-calculus

In this section, we introduce briefly the modal μ-calculus [3] and the propositions of states we are going to use to reason about Maude specifications. Our language of propositions $P(\Sigma)$ over a signature Σ has the following syntactic structure:

$$p ::= \texttt{true} \mid \texttt{false} \mid \text{“}o\text{”} \mid \text{“}m\text{”} \mid \neg p$$

Here, o denotes an object and m a message. The double quotes around an object or message represent the proposition "this object exists", or "this message exists", respectively. Let $p \in P(\Sigma)$ be a proposition. We define the formulas $\mathcal{L}(P(\Sigma))$ of the *modal μ-calculus* over a set of basic propositions as follows:

$$\begin{array}{rcl} \phi & ::= & p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \langle L\rangle\phi \mid [L]\phi \\ & & \mid (\nu X.\phi) \mid (\mu X.\phi) \mid (\forall\, v\!:\!V.\phi(v)) \mid (\exists\, v\!:\!V.\phi(v)) \end{array}$$

L is a set of labels. $[L]\phi$ and $\langle L\rangle\phi$ are the labeled modal connectives. Intuitively, $[L]\phi$ means that ϕ holds immediately after all transitions with labels in L. Dually, $\langle L\rangle\phi$ holds if there is a transition with a label in L such that ϕ holds immediately afterwards. ν is the greatest fixed point operator used, typically, for invariant (safety, "always") properties. μ is the least fixed point operator used, typically, for variant (liveness, "sometime") properties. $\forall$ and $\exists$ are used for quantification over finite sets of primitive values, i.e., not over values of sort Cf.

We give special names to the following fragments of the modal μ-calculus: $\langle\,\rangle\mathcal{L}_\mu(P(\Sigma))$ denotes $[\,]$-free formulas with atomic propositions $P(\Sigma)$, $[\,]\mathcal{L}_\mu(P(\Sigma))$ denotes $\langle\,\rangle$-free formulas. We give negation-free fragments the superscript "+".

We define the truth of the propositions of our language in (A,R) with respect to a configuration C and a valuation v. We extend I_v to range over sets of terms. $\overleftrightarrow{\texttt{a:v}}$ denotes a set of attributes with their values, $\overleftrightarrow{p}$ the parameters of m.

$$
\begin{array}{llll}
(A,R),C,v & \models & \text{“}\texttt{<O:X|}\overleftrightarrow{\texttt{a:v}}\texttt{>}\text{”} & \text{iff for some } \texttt{D},\texttt{D} \leq \texttt{X}, \text{ and some } \texttt{atts} \\
 & & & I_v(\texttt{<O:D|}\overleftrightarrow{\texttt{a:v}}\texttt{,atts>}) \in C \\
(A,R),C,v & \models & \text{“}\texttt{m(}\overleftrightarrow{p}\texttt{)}\text{”} & \text{iff, } I_v(\texttt{m(}\overleftrightarrow{p}\texttt{)}) \in C \\
(A,R),C,v & \models & \langle L\rangle\phi & \text{iff for some } l \in I_v(L), C' \in A_{\mathrm{Cf}}, \\
 & & & (C,C') \in R_l \text{ and } (A,R),C',v \models \phi \\
(A,R),C,v & \models & [L]\phi & \text{iff for all } l \in I_v(L), C' \in A_{\mathrm{Cf}}, \\
 & & & (C,C') \in R_l \text{ implies } (A,R),C',v \models \phi
\end{array}
$$

In the following, let $|\,\phi\,|_{(A,R),I}\ (v)$ denote the set of all configurations that satisfy the formula ϕ under I_v, in model (A,R), i.e., $C \in |\,\phi\,|_{(A,R),I}\ (v)$ if and only if $(A,R),C,v \models \phi$.

4 Connections and simulations

When relating two Maude specifications, we have to take into account different signatures, different representations by objects and also different implementations with respect to synchronization and message passing. Following [3], we use Galois connections to relate the states of two specifications and simulation relations to relate transition systems.

Let us introduce some abbreviations and notation. $\overline{X}$ is the the complement of X in the domain of X. Id^Q is the identity function on a set Q. The dual of a function α is $\widetilde{\alpha}$, defined by $\widetilde{\alpha}(X) =_{\mathrm{def}} \overline{\alpha(\overline{X})}$. Let Q be a set of states, $X \subseteq Q$; $\mathit{pre}(R)(L)(X)$ represents the set of predecessors in Q in a transition relation R by transitions with a label in the label set L.

Definition 1 (Galois connection) *Let Q_1 and Q_2 be two sets of states. A Galois connection (α,γ), from $\wp(Q_1)$ to $\wp(Q_2)$ is a pair of monotonic functions, α, γ where $\alpha : \wp(Q_1) \to \wp(Q_2)$ and $\gamma : \wp(Q_2) \to \wp(Q_1)$, such that $Id^{Q_1} \subseteq \gamma \circ \alpha$ and $\alpha \circ \gamma \subseteq Id^{Q_2}$.*

Note that α and $\widetilde{\gamma}$ distribute over union, i.e., $\alpha(S_1 \cup S_2) = \alpha(S_1) \cup \alpha(S_2)$, for $\widetilde{\gamma}$ analogously. We give the mappings only for single states, i.e., configurations, not for sets of configurations. We allow to omit braces around single configurations, e.g., we write $\alpha(C)$ instead of $\alpha(\{C\})$ and $\alpha(S) = C$ instead of $\alpha(S) = \{C\}$.

Based on Galois connections as a formal framework for relating sets of states, we define simulation relations between transition systems whose states are in a Galois connection.

Definition 2 ($\sqsubseteq_{(\alpha,\gamma)}$ and $\simeq_{(\alpha,\gamma)}$) *Let (Q_1, R_1) and (Q_2, R_2) be two transition systems, L_1 a set of labels and (α, γ) a connection from $\wp(Q_1)$ to $\wp(Q_2)$. (Q_2, R_2) is an (α,γ)-simulation of (Q_1, R_1) written $(Q_1, R_1) \sqsubseteq_{(\alpha,\gamma)} (Q_2, R_2)$, if and only if for any $L \subseteq L_1$, $\alpha \circ pre(R_1)(L) \circ \gamma \subseteq pre(R_2)(\alpha(L))$.*

(Q_1, R_1) and (Q_2, R_2) are (α,γ)-bisimilar, written $(Q_1, R_1) \simeq_{(\alpha,\gamma)} (Q_2, R_2)$, if and only if $(Q_1, R_1) \sqsubseteq_{(\alpha,\gamma)} (Q_2, R_2)$ and $(Q_2, R_2) \sqsubseteq_{(\widetilde{\gamma},\widetilde{\alpha})} (Q_1, R_1)$.

Preservation of a formula by a function f means that, if a formula holds for a set of states, then it holds as well for the image of this set under f, i.e., f *preserves* ϕ, if $c \in \mid \phi \mid_{S_1,I}$ implies $f(c) \subseteq \mid \phi \mid_{S_2,\alpha\circ I}$. f preserves a set Φ of propositions if it preserves all $\phi \in \Phi$. A function f is *consistent* with an interpretation function I if $f(\overline{I(\phi)}) \cap f(I(\phi)) = \{\}$ for all formulas ϕ.

Theorem 3 (Preservation of properties) *Let (Σ_1, E_1, T_1) and (Σ_2, E_2, T_2) be two specifications, and (A_1, R_1), (A_2, R_2) their state transition systems. Let $P_1 \subseteq P(\Sigma_1)$ and $P_2 \subseteq P(\Sigma_2)$ be two languages of basic propositions.*

If $(A_1, R_1) \sqsubseteq_{(\alpha,\gamma)} (A_2, R_2)$ and if α preserves P_1^+ then α preserves $\langle\,\rangle\mathcal{L}_\mu^+(P_1)$ for I_1 and, if α is consistent with I_1, then α preserves $\langle\,\rangle\mathcal{L}_\mu(P_1)$.

If $(A_1, R_1) \sqsubseteq_{(\alpha,\gamma)} (A_2, R_2)$ and if $\widetilde{\gamma}$ preserves P_2^+ for I_2, then $\widetilde{\gamma}$ preserves $[\,]\mathcal{L}_\mu^+(P_2)$ for I_2, if $\widetilde{\gamma}$ is consistent with I_2, then $\widetilde{\gamma}$ preserves $[\,]\mathcal{L}_\mu(P_2)$ for I_2.

If $(A_1, R_1) \simeq_{(\alpha,\gamma)} (A_2, R_2)$ and α preserves P_1^+ then α preserves $\mathcal{L}_\mu^+(P_1)$ for I_1 and, if α is consistent with I_1, then α preserves $\mathcal{L}_\mu(P_1)$ for I_1.

We must take a closer look at which basic properties are preserved by a simulation relation. While α and $\widetilde{\gamma}$ preserve `true` and `false`, we should not add the formulas "o" and "m" for all messages and all classes to our set of basic propositions. The reason is that we would like the interpretation functions to be consistent with α and $\widetilde{\gamma}$ in order to be allowed to use implication. Therefore, in Sect. 5, e.g., "`<P:Plane>`" does not belong to the set of basic propositions.

5 Abstraction and verification

We verify that a mutual exclusion property of runways holds for the transition system that is the model of the Maude specification `AIRPORT`. From an abstract point of view, a runway can be modeled by a binary semaphore, an airport with several runways by a set of binary semaphores. In this section, we use the subscripts Sema and Airp, for specifications, interpretation functions and transition systems. The transition rules can be found in Fig. 1. Let us briefly explain the two specifications and their rules.

A plane sends a request for a runway, and the data necessary for the selection of a runway from the set of free runways are exchanged in a synchronous communication. Then the plane taxis to the runway and takes off subsequently.

A binary semaphore reacts to three messages. The usual two are `P` for entering and `V` for leaving the critical section governed by the semaphore. For technical reasons, we need to introduce a third action, `T`, which serves as the abstraction of all messages which have to do with a runway but do not change its state. P and V are only accepted by a semaphore if it is in the appropriate state: if the semaphore is free it accepts a P, if it is occupied it accepts a V. The mutual exclusion property is given by a formula which describes which sequences of actions are allowed: after a P (entering the critical section governed by a semaphore) there has to be a V (leaving the critical section) before the next P action is allowed. All actions apart from P(R) and V(R), e.g., also T(R), do not change the state of semaphore R.

$$\begin{aligned}&ME_{\text{Sema}} =_{\text{def}} (\forall \texttt{R} : T_{\text{OId}}(\Sigma).\text{“}\texttt{<R:Sema|status:0>}\text{”} \Rightarrow \\ &(\nu X_1.\ [(\texttt{V(R)})]\ \text{false}\ \wedge [T_{\text{Msg}}(\Sigma_{\text{Sema}})\backslash\{(\texttt{V(R)}),(\texttt{P(R)})\}]\ X_1\ \wedge [(\texttt{P(R)})]\ (\nu X_2. \\ &\qquad [(\texttt{P(R)})]\ \text{false}\ \wedge [T_{\text{Msg}}(\Sigma_{\text{Sema}})\backslash\{(\texttt{V(R)}),(\texttt{P(R)})\}]\ X_2\ \wedge [(\texttt{V(R)})]\ X_1)))\end{aligned}$$

This formula belongs to $[\ ]\mathcal{L}_\mu$ and since we would like to infer from its validity in the abstract specification SEMA the validity in the concrete specification AIRPORT, we proceed as follows. We give a Galois connection between the states of the transition systems of AIRPORT and SEMA. We use α to abstract from the transition rules of AIRPORT and obtain transition rules of SEMA such that SEMA (α, γ)-simulates AIRPORT. Then, we are able to apply the preservation result from Thm. 3 to the formula $ME_{\texttt{Sema}}$ to prove mutual exclusion for runways.

As an example for $\alpha : \wp(A_{\text{Airp}}) \to \wp(A_{\text{Sema}})$, we give the mapping of a configuration which consists of a tower, two runways and planes, to a configuration consisting of two semaphores. If a runway is contained in the list in which the free runways are stored, then the semaphore is in state 0, otherwise in state 1.

$$\begin{array}{lcl} \alpha(\texttt{<R1:Runway> <R2:Runway> <T:Tower|free:F,nfree:N>}) & & \\ \quad = \texttt{<R1:Sema|status:x1> <R1:Sema|status:x2>} & & \\ \qquad \text{if } \texttt{Ri} \in \texttt{F} \text{ then } \texttt{xi} = \texttt{0} \text{ otherwise } \texttt{xi} = \texttt{1}. & & \\ \alpha(\texttt{<P:Plane>}) & = & \epsilon \\ \alpha(\texttt{(P taxis to R)}) & = & \texttt{(P(R))} \\ \alpha(\texttt{(P takeoff at R)}) & = & \texttt{(V(R))} \\ \alpha(\texttt{(P airborne at R)}) & = & \texttt{(T(R))} \\ \alpha(\texttt{(req runway from T for P to take off)}) & = & \texttt{(T(R))} \end{array}$$

The function γ maps two semaphores to one airport. Since α abstracts from part of the structure of the airports (planes, the tower and its data), γ yields a high number of concrete states.

We use the function α to find for each transition rule of AIRPORT a transition rule of SEMA. The corresponding transition rules are given in Fig. 1. Note that we map rule [A1], which is responsible for mobility, to [S1], which generates P messages for an arbitrary semaphore (the identifier R is free). Thus, the selection of a runway is modeled by the possibility of entering any semaphore.

Lemma 4 *(α, γ) is a Galois connection, $(A_{Airp}, R_{Airp}) \sqsubseteq_{(\alpha,\gamma)} (A_{Sema}, R_{Sema})$, $(A_{Sema}, R_{Sema}) \models ME_{Sema}$, $\widetilde{\gamma}$ preserves ME_{Sema}.*

We relate specifications with quite different states: the state of a semaphore is determined by its attribute. The state of a runway depends on the state of

<table>
<tr><th>Rules of <code>AIRPORT</code></th><th>Rules of <code>SEMA</code></th></tr>
<tr><td><pre>[A1]
rl (req runway from T for P to take off)
   < P:onGroundPl >
   < T:Tower | free:F, nfree:N >
 => < P:StartingPl | runway:select(F) >
    < T:Tower | free:F, nfree:N >
    (P taxis to R) .</pre></td><td><pre>[S1]
rl (T(R))
   < R:Sema >
 => < R:Sema >
    (P(R)) .</pre></td></tr>
<tr><td><pre>[A2]
rl (P taxis to R)
   < P:StartingPl | runway:R >
   < R:Runway >
   < T:Tower | free:F, nfree:N >
 => < P:TakeoffPl >
    < R:Runway >
    < T:Tower | free: remove R from F,
                nfree: insert R in N >
    (P takeoff at R) .</pre></td><td><pre>[S2]
rl (P(R))
   < R:Sema |
         status:0 >
 => < R:Sema |
          status:1 >
    (V(R)) .</pre></td></tr>
</table>

Figure 1: Corresponding transition rules

the tower and of a plane. While, in `SEMA`, an object of class `Sema` controls via attribute `state` which messages it accepts, a runway is passive in the sense that it cannot influence which planes take off or land on it. We have only proven the mutual exclusion property for take-offs, not for landings. The landing is specified analogously, thus the specification of semaphores is an abstraction of both take-off and landing. This decreases the complexity of the proof considerably.

6 Inheritance of properties

We are interested in the "classes of formulas" which can be inherited via Maude's inheritance relation, i.e., which are preserved by a simulation relation parameterized by a Galois connection modeling Maude's inheritance relation.

Objects are able to change class (see Sect. 2). This makes the class hierarchy comparably fine-grained and inheritability of properties even more important. Assume we would like to distinguish and modularize our specification of planes by two means: the kind (cargo or passenger planes) and the state (`FlyingPl`). Flying cargo planes inherit all rules from flying planes and from cargo planes (and there might even be additional rules for them). Thus, the final specification simulates its ancestor specification and α maps the ancestor specification to the heir specification and γ vice versa.

We give the mapping for three attributes: (1) `flno` represents all attributes inherited from `Plane`, and thus common to `FlyingPl` and `CargoPl`, (2) `speed` is inherited from `FlyingPl` and, (3) `weight` represents the attributes of `CargoPl`.

$$\begin{aligned}&\alpha(\texttt{<P:FlyingPl|flno:F,speed:S>},\texttt{<P:CargoPl|flno:F,weight:W>})\\ &\quad = \ \texttt{<P:FlyingCargoPl|flno:F,speed:S,weight:W>}\end{aligned}$$

$$\begin{aligned}&\gamma(\texttt{<P:FlyingCargoPl|flno:F,speed:S,weight:W>})\\ &\quad = \ (\texttt{<P:FlyingPl|flno:F,speed:S>},\texttt{<P:CargoPl|flno:F,weight:W>})\end{aligned}$$

α preserves formulas in $\langle\,\rangle\mathcal{L}_\mu$, i.e., a configuration C in A_{Heir} "inherits" properties as, e.g., the liveness property, specifying that all planes request a runway, taxi to the runway and take off (See also Fig. 1):

$$\begin{aligned}&(\forall \texttt{P} : \texttt{OId}.(\exists \texttt{R},\texttt{T} : \texttt{OId}.\text{“}\texttt{<P:onGroundPl>}\text{”} \ \Rightarrow\\ &\quad \langle(\texttt{req runway from T for P to take off})\rangle\\ &\quad \langle(\texttt{P taxis to R})\rangle\langle(\texttt{P takeoff at R})\rangle))\end{aligned}$$

7 Concluding remarks

Since our approach of abstraction is based on transition systems, it is language independent. It provides us with a formal semantical basis for reuse constructs which are more flexible than Maude's structure- and syntax-oriented way of manipulating objects and configurations. Subject to further work will be to apply this approach to a subclass relation as a reuse construct at the object level as well as to reuse concepts for entire specifications.

Acknowledgements: We would like to thank Glenn Bruns, Steve Hodges, Cliff Jones, Narciso Martí-Oliet, Stephan Merz and especially Martin Wirsing for carefully reading versions of this paper. This work is part of the DFG project OSIDRIS and received travel support from the ARC program of the DAAD.

References

[1] G. Bruns. A practical technique for process abstraction. In E. Best, editor, *4th Int. Conf. on Concurrency Theory (CONCUR'93)*, Lecture Notes in Computer Science 715, pages 37–49. Springer-Verlag, 1993.

[2] U. Lechner and C. Lengauer. Modal–μ–Maude — properties and specification of concurrent objects. In B. Freitag, C.B. Jones, C. Lengauer, and H.-J. Schek, editors, *Object Orientation with Parallelism and Persistence*. Kluwer, 1996. To appear.

[3] C. Loiseaux, A. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstraction for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–45, 1995.

[4] J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT Press, 1993.