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

(Extended Abstract)

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

Abstract: 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 the 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 of a collection of objects. The more concurrency a language admits, the more important in verification become the properties of the behavior of objects. Such properties of classes are, e.g., which (sequences of) messages are accepted or refused. Liveness, absence of deadlock and mutual exclusion are properties relevant in verification of a global state. Important for modular verification of structured object-oriented specifications or programs is the inheritability of properties.

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 [10, 11, 12], 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 [7].

Abstraction is also the key concept of our approach to verification. We employ a framework of abstraction and verification of concurrent programs [1, 5, 6, 9]. 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 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 focus on properties and behavior of the global system, and we use this framework for the verification of properties and for identifying a class of properties that is inheritable by Maude's inheritance relation.

A longer version of this paper is [5] (and [6]). More details will appear in [3]. [4] deals with properties and specification of single classes.

2 Maude, the μ-calculus and simulations

Maude [10] 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.

Let us give as an example of a transition rule a rule modeling a state transition of an airport and explain it briefly. Class `Plane` has (here) two attributes. The only transition rule specifies an atomic state transition involving three objects: a plane and two objects modeling a runway and a tower whose object identifier the plane has stored. The plane changes its class –and thus its state– from `StartingPl` to `TakeoffPl` [8].

```
rl (P taxis to R)
   < P:StartingPl | runway:R, tower: T >   < R:Runway >
   < T:Tower | free:F, nfree:N >
 => < P:TakeoffPl | runway:R, tower: T >   < R:Runway >
    < T:Tower | free: remove R from F, nfree: insert R in N >
    (P takeoff at R) .
```

A *rewriting calculus* applies transition rules to the states in a system [10, 12, 11]. 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 state. Hereby, all values (object identifiers, values of attributes) of all objects and messages involved may be "exchanged".

This is abstract, but operational. To reason about properties, we use the μ-calculus [2, 9] with propositions of states. Formulas of the μ-calculus consist of propositional variables, $\neg$, $\vee$ and $\wedge$, greatest fixpoint and least fixpoint. $[L]\phi$ and $\langle L\rangle\phi$ are the labeled modal connectives (L is a set of labels). 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. We introduce propositions on states, "o" and "m", representing "this object exists", or "this message exists", respectively. L is a set of labels.

Following [9], we use a Galois connection (α, γ) to relate the states of two specifications and (α, γ)-simulation or bisimulation relations to relate transition systems. The semantics of a Maude specification is a transition system, namely the initial term-generated model. An interpretation of a μ-formula hold for (some) states of the transition system. α preserves $[\,]$-free formulas while the dual of γ preserves $\langle\,\rangle$-free formulas. The formal definitions can be found in [5, 6].

3 Abstraction, verification and inheritance

This framework relates two transition systems such that certain classes of μ-formulas are preserved. The formula and in particular, its modal connectives determines the relation between the concrete and the abstract specification.

To obtain the abstract specification for the proof of a concrete specification, we can apply the functions α, resp., γ to the transition rules of the concrete specification and obtain so the transition rules of the abstract specification.

In [5, 6] this framework is applied to verification of a mutual exclusion property for runways in a specification of an airport. Abstractly, a runway can be modeled by a binary semaphore, an airport with several runways by a set of binary semaphores. The mutual exclusion property is a $\langle\,\rangle$-free formula. We use α to abstract from the transition rules of the aiport and obtain transition rules of semaphores such that we get an (α, γ)-simulation relation. The (abstract) rule corresponding to the (concrete) transition rule of Sect. **??** is: `rl (P(R)) <R:Sema|status:0> => <R:Sema|status:1> (V(R)).` We prove that mutual exclusion holds for semaphores and we apply the preservation result in inferring mutual exclusion for the airport from the semaphore specifications. Hereby, the complexity of the proof is considerably reduced, since all actions changing the state of a runway (not only take-offs but also landings) are mapped to the few actions of semaphores.

Next we apply this framework to identify the properties that are preserved by Maude's inheritance relations, i.e., that are preserved by a Galois connection and a simulation relation modeling Maude's inheritance relation. A heir inherits from its ancestor all attributes and all transition rules. Thus, a heir is able to react to all messages the ancestor reacts to. The ancestor is simulated by an heir. In [5, 6] we give the Galois connection and the simulation relation between the two transition systems of the ancestor and the heir.

Hereby, α maps hereby states of the ancestor to states of the heir. α preserves $[\,]$-free formulas, which model typically liveness properties of single objects [4] and collections [5]. This result also justifies the use of transition rules and a rewriting calculus that allows only to express that certain transitions *may* happen, since only those properties are inheritable.

4 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 G. Bruns, S. Hodges, C. Jones, N. Martí-Oliet, S. Merz helpful comments and especially M. Wirsing for many fruitful discussions. This work is part of the DFG project OSIDRIS and received travel support from the ARC program of the DAAD.

Bibliography

[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] M. Dam. Model checking mobile processes. In E. Best, editor, *4th Int. Conf. on Concurrency Theory (CONCUR'93)*, Lecture Notes in Computer Science 715, pages 22–36. Springer-Verlag, 1993.

[3] U. Lechner. Object-oriented specification of distributed systems. To appear.

[4] U. Lechner. Object-oriented specifications of distributed systems in the μ-calculus and Maude. *ENTCS*, 5, 1996. Proceedings of the First International Workshop on Rewriting Logics and its applications. To appear.

[5] 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.

[6] U. Lechner and C. Lengauer. Specification, abstraction and verification in a concurrent object-oriented language (extended abstract). In J.-P. Bahsoun, J. Fiadero, D. Galmiche, and A. Yonezawa, editors, *Proceedings of the ECOOP96 workshop on Proof Theory of Concurrent Object-Oriented Programming*, 1996.

[7] U. Lechner, C. Lengauer, F. Nickl, and M. Wirsing. (Objects + Concurrency) & Reusability – A Proposal to Circumvent the Inheritance Anomaly. In *ECOOP'96*, Lecture Notes in Computer Science 1098, pages 232–248. Springer-Verlag, 1996.

[8] U. Lechner, C. Lengauer, and M. Wirsing. An Object-Oriented Airport: Specification and Refinement in Maude. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *10th Workshop on Specification of Abstract Data Types Joint with the 5th COMPASS Workshop, Selected papers*, Lecture Notes in Computer Science 906, pages 351–367. Springer-Verlag, 1995.

[9] 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.

[10] 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.

[11] J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Concur 96*, Lecture Notes in Computer Science. Springer-Verlag, 1996. To appear.

[12] J. Meseguer and T. Winkler. Parallel programming in Maude. In J.-P. Banâtre and D. Le Métayer, editors, *Research Directions in High-Level Parallel Programming Languages*, Lecture Notes in Computer Science 574, pages 253–293. Springer-Verlag, 1992.