A Formalization of Abstraction in LAMBDA *

Anthony McIsaac

Laboratory for the Foundations of Computer Science
The King's Buildings, University of Edinburgh
Edinburgh EH9 3JZ, Scotland

Abstract. In a mixed approach to system verification using theorem provers with an interface to specialized model-checking tools, it may be necessary to simplify models by considering abstract versions of them. We report on work in progress that aims to develop support within LAMBDA for a systematic approach to abstraction. We give a formalization in LAMBDA of a notion of abstraction for transition systems; the abstract systems have two sorts of transition, and are related to specifications in modal process logic. We prove that formulae in the modal mu-calculus are satisfied in an abstract version of a model only if they are satisfied in the model itself. We illustrate how the proof of an inductive step in the verification of a satisfaction relation for an infinite model can be reduced to the verification of a satisfaction relation for a very small finite model.

1 Introduction

General-purpose theorem provers such as those based on higher-order logic have a place in supporting system design and verification, not least because of the wide range of properties that can be expressed in them. But, even with modern sophisticated theorem provers, formal verification is a cumbersome business. On the other hand, for properties expressed in particular formalisms, efficient automated model-checking methods have recently been implemented in tools such as the Concurrency Workbench [5] or those based on BDDs [3]. There is therefore strong reason to investigate approaches that combine the best of both worlds, the flexibility and interactive capabilities of theorem provers with the efficiency of automated specialized packages. Seger and Joyce [12, 8] have constructed a sound interface between the HOL theorem prover and the Voss symbolic trajectory evaluation system, and shown that the hybrid system can be used to good effect.

There is some middle ground between the use of a higher-order logic theorem prover to show that some verification problem expressed in a familiar language can be reduced to a model-checking task expressed in a special formalism, and the passing of this task to a specialized tool. The model may be infinite, or too enormous for the tool. In this case it will be necessary to reduce the model-checking task to questions about simpler models. One technique that can be

* This research was supported by the SERC grant GR/F 31199 'Formal System Design Tools'
used to this end is *abstraction*. This is a matter of replacing a model $M$ with
an abstract version $A$ that contains only some of the information in $M$, but no
information that is not in $M$, so that, roughly speaking, any statement that is
true for $A$ is true for $M$. The use of one such notion of abstraction has been
explored by Clarke, Grumberg and Long [6]; we propose a somewhat more gen-
eral notion. In Section 2, we investigate how this notion can be formalized in
higher-order logic, and establish some theorems that can be used in applications
of the technique.

We would like to explore an approach to model checking in which information
about a model could be used in only two ways: to prove that one model is an
abstraction of another (this is to be done within a theorem prover, making use
of one's understanding of the structure of the model, and should be as straight-
forward as possible), and direct checking that a formula holds in a model (this
is to be done by an automated tool with an interface to the theorem prover, and
may involve combinatorial complexities, but the model must be of manageable
size, and certainly finite). An alternative approach would be to use the theorem
prover to simulate reasoning in some logic for the model, as is done for example
in [11], but we feel that our approach makes a clearer division between different
aspects of the task. In section 3, we test this approach on a very simple example.

This paper describes work in progress that aims to develop a systematic,
formalized approach to abstraction, as part of a mixed approach to system design
and verification, following ideas suggested by Mike Fourman. All the work has
been carried out using the higher-order logic theorem prover LAMBDA\footnote{LAMBDA is a product of Abstract Hardware Limited} [7]. We
report on how much we have formally proved in LAMBDA, but the simulated
typescript in the text is not an exact transcript of our sessions.

2 Abstract Transition Systems

2.1 Formalization of Transition Systems

Our aim is to reduce the problem of checking whether a formula $\phi$ holds in some
model $M$ to that of checking whether some formula $\psi$ holds in some simpler
model $A$; the latter task must be fully automatable, and $A$ must be finite.

The models $M$ here will be taken to be labelled transition systems. A labelled
transition system consists of a set $S$ of states, a set $K$ of labels, and a subset $T$ of
$S \times S \times K$, where $(x, y, k) \in T$ means that there is a transition from $x$ to $y$ with
label $k$. The language we choose for the formulae $\phi$ is the modal mu-calculus
[9], which can express a wide range of properties using its fixed-point operators.
Formulations in other, more immediately comprehensible, logics such as CTL or
CTL* [4], can be translated into the modal mu-calculus. Also, systems specified,
for example, as Petri nets or in CCS have transition systems associated with
them, so we are remaining reasonably general.

In our formalization of the satisfaction relation in LAMBDA, a model has
the polymorphic type $(\forall x \rightarrow \text{om}) \ast ((\forall x \ast (\forall y \ast \text{b}) \rightarrow \text{om})$. Here the states