Specification and Verification of Invariant Properties of Transitional Systems

Daniel Găină

Institute of Mathematics for Industry, Kyushu University
daniel@imi.kyushu-u.ac.jp

Abstract. The paper advances a verification method for transition systems whose reachable states are explicitly described by membership axioms. The proof technique is implemented in Constructor-based Inductive Theorem Prover (CITP), a proof management tool built on top of a variation of conditional equational logic enhanced with many modern features. This approach complements the so-called OTS/CafeOBJ method, a verification procedure for observational transitional systems that is already implemented in CITP.

1 Introduction

We propose a formal method for analyzing transition systems whose reachable states are described explicitly by membership axioms [8]. The specification and verification technique is supported by Constructor-based Inductive Theorem Prover (CITP) [7], which is implemented in Maude [1]. The formal language of CITP is based on a variation of conditional equational logic [3] with sub-sorting relations [4], membership axioms, transitions [2] and constructor operators [5], which makes Maude notation suitable to write specifications in the CITP language. The methodology described in the present contribution complements the one already supported by CITP [6] for analyzing invariant properties of Observational Transition Systems (OTS). In the latter technology, the system states are "black-boxes" that are distinguished only by observational functions.

Inductive theorem provers are interactive, in general. They require a trained user to direct the theorem prover towards discharging the goals which cannot be proved by automatic techniques. In many cases, the tool needs the user's help to perform trivial proofs. One major direction of research in inductive theorem proving is improving and reducing the user interaction. This issue is tackled in the present contribution from the following angles: (1) better strategies for generating induction schemes, (2) improved decision procedures to perform automated reasoning, and (3) improvements of the proof assistant interface to help the user understand the current state of the proof and interact with the tool in a more natural way.

The CITP's source code has been refactored to be more readable and improved to make it a better platform for future extensions. The new features consist of (a) an induction scheme based on constructors given as membership axioms, (b) a tactic for computing and joining critical pairs, and (c) a new interface designed to improve the user's interaction with the tool and then implemented from scratch in Core Maude. Note that in the previous version the interface was implemented in Full Maude. The command parsing component of the interface was upgraded to generate better error messages. CITP can be downloaded from `http://imi.kyushu-u.ac.jp/~daniel/citp.html`.

2 Preliminaries

Note that a specification consists of a signature, a set of sentences and a class of models. The set of sentences gives the operational semantics since they form a term rewriting system which makes the specification executable by rewriting. The class of models gives the denotational semantics of the underlying specification. Our proof technique is intimately linked to the structure of the sentences and takes advantage of both denotational and operational semantics in order to improve the user's interaction with the tool: (a) it is equipped with specialized proof rules for the initial data types that are often used in practice such as sequences or Booleans, and (b) it uses term rewriting for verification.

A goal is a pair $\langle \mathtt{SP}, E\rangle$, where SP is a specification and E is a set of formulas to prove. A proof rule is a mapping from a goal $\langle \mathtt{SP}, E\rangle$ to a list of specifications $\langle \mathtt{SP}_1, E_1\rangle \dots \langle \mathtt{SP}_n, E_n\rangle$. The tactics are obtained by canonically extending the proof rules to mappings from lists of goals to lists of goals. In the current version of CITP, the user can give commands which may consist of lists of tactics rather than a single tactic. It follows that the proofs consist of sequences of lists of tactics instead of trees (whose nodes are goals and edges are lists of tactics). This has the advantage of simplifying the design of the tool interface and increase the automation level of the proof process by making all goals to be discharged available to the user simultaneously. However, if the user wants to apply a tactic list `tctList` to the current goal then the command `(. tctList)` can be fed to the tool. In addition, `(select n)` moves the goal `n` to the top of the goal list making it the current goal.

Assume that the initial goal is a pair $\langle \mathtt{SP}, \gamma\rangle$ consisting of a specification SP and a sentence $\gamma = (\forall X) \bigwedge H \Rightarrow C$, where X is a set of variables and $H \cup \{C\}$ is a set of atomic formulas given as equations, membership axioms or transitions. The idea is to develop tactics that decompose the sentence γ into simpler basic constituents, e.g. equational formulas, which can be discharged using term rewriting. There are two tactics designed to eliminate universal quantification: induction and theorem of constants. While the latter is applicable to all variables in X, the former can eliminate only variables of *constrained* sort [1]. It is the user's responsibility to separate variables of constrained sorts, which will be dealt by induction, from the rest of the variables, which will be handled by theorem of constants. Induction is applied first and then theorem of constants comes into play. The goals $\langle \mathtt{SP}_1, \gamma_1\rangle \dots \langle \mathtt{SP}_n, \gamma_n\rangle$ obtained by applying induction

[1] The sorts that have constructors are called constrained. The sorts with no constructors are called loose.

and theorem of constants have only quantifier free formulas, i.e. γ_i is of the form $\bigwedge H_i \Rightarrow C_i$ for all $i \in \{1, \ldots, n\}$. The tactic implication is designed to eliminate the logical implication. For example, by applying implication to a goal $\langle \mathtt{SP}_i, \gamma_i \rangle$ the result is $\langle \mathtt{SP}_i[H_i], C_i \rangle$, where $\mathtt{SP}_i[H_i]$ is the specification obtained from $\mathtt{SP}_i$ by adding the axioms in H_i. The goals $\langle \mathtt{SP}_i[H_i], C_i \rangle$ are discharged using term rewriting.

Consider the following specification of natural numbers with addition:

```
fmod PNAT is
 sorts PNat PNzNat .
 subsorts PNzNat < PNat .
 op 0 : -> PNat [ctor] .
 op s_ : PNat -> PNzNat [ctor].
 op _+_ : PNat PNat -> PNat [prec 33].
 vars M N : PNat .
 eq 0 + N = N          [metadata "1"].
 eq s M + N = s(M + N)[metadata "2"].
endfm
```

Suppose one wants to prove the associativity of addition. The goal is introduced by the command: `(goal PNAT |- eq (X:PNat + Y:PNat)+ Z:PNat = X:PNat + (Y:PNat + Z:PNat);)`. Variable `X` is chosen for induction and the tactic `(ind on X:PNat)` is applied. The tool will generate two subgoals, one for each constructor, zero `0` and successor `s`. By theorem of constants `(tc)`, the variables `Y` and `Z` are introduced as constants to the body of the specifications of the subgoals generated by induction. Since the logical implication does not occur in the formulas to prove, the goals can be discharged by the reduction tactic `(red)`. The proof of associativity consists of the tactic list `(ind on X:PNat tc red)`. The command `(show proof)` displays the current proof and can be used any time during the proof process. The command `(show goals)` display the remaining goals, and the command `(rollback)` returns the proof process to the previous state. In case of large specifications, it is undesirable to display all its sentences during the proof process. Therefore, only sentences with the attribute `metadata "n"` will be displayed, where `n` is a natural number. The axioms introduced in the proof process will get automatically a natural number as attribute.

The verification method described above is developed further in two directions to prove invariant properties of transitional systems: (1) one technique is designed for observational transitional systems, where the structure of the states is not accessible to the users and the states are distinguished only by some "observational" functions, and (2) the other approach is built for transitional systems where the states are described explicitly by membership equations. Therefore, the methods share common tactics such as theorem of constants, implication or reduction. The implementation of the tool is modular allowing to reuse the code for the shared tactics. Note that the present contribution focuses on the latter methodology.

References

1. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007.
2. R. Diaconescu and K. Futatsugi. *Cafeobj Report - The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.
3. J. A. Goguen and J. Meseguer. Completeness of many-sorted equational logic. *SIGPLAN Notices*, 17(1):9–17, 1982.
4. J. A. Goguen and J. Meseguer. Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theor. Comput. Sci.*, 105(2):217–273, 1992.
5. D. Găină, K. Futatsugi, and K. Ogata. Constructor-based Logics. *J. UCS*, 18(16):2204–2233, 2012.
6. D. Găină, D. Lucanu, K. Ogata, and K. Futatsugi. On Automation of OTS/CafeOBJ Method. In S. Iida, J. Meseguer, and K. Ogata, editors, *Specification, Algebra, and Software*, volume 8373 of *LNCS*, pages 578–602. Springer, 2014.
7. D. Găină, M. Zhang, Y. Chiba, and Y. Arimoto. Constructor-based Inductive Theorem Prover. In R. Heckel and S. Milius, editors, *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013*, volume 8089 of *LNCS*, pages 328–333. Springer, 2013.
8. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97*, volume 1376 of *LNCS*, pages 18–61. Springer, 1997.