Local and Symbolic Bisimulation
Using Tabled Constraint Logic Programming

Samik Basu, Madhavan Mukund, C.R. Ramakrishnan, I.V. Ramakrishnan, and Rakesh Verma

1 Department of Computer Science, State University of New York at Stony Brook
Stony Brook, New York, U.S.A.
{bsamik,cram,ram}@cs.sunysb.edu
2 Chennai Mathematical Institute, Chennai, India.
madhavan@cmi.ac.in
3 Department of Computer Science, University of Houston, Texas.
rmverma@cs.uh.edu

Abstract. Bisimulation is a fundamental notion that characterizes behavioral equivalence of concurrent systems. In this paper, we study the problem of encoding efficient bisimulation checkers for finite- as well as infinite-state systems as logic programs. We begin with a straightforward and short (less than 10 lines) encoding of finite-state bisimulation checker as a tabled logic program. In a goal-directed system like XSB, this encoding yields a local bisimulation checker: one where state space exploration is done only until a dissimilarity is revealed. More importantly, the logic programming formulation of local bisimulation can be extended to do symbolic bisimulation for checking the equivalence of infinite-state concurrent systems represented by symbolic transition systems. We show how the two variants of symbolic bisimulation (late and early equivalences) can be formulated as a tabled constraint logic program in a way that precisely brings out their differences. Finally, we show that our symbolic bisimulation checker actually outperforms non-symbolic checkers even for relatively small finite-state systems.

1 Introduction

A tabled logic programming system offers an attractive platform for encoding computational problems in the specification and verification of systems. The XMC system [12] casts the problem of model checking—verifying whether a given concurrent system is in the model of a temporal logic formula—as query evaluation over an “equivalent” logic program [11]. This formulation is based on the connection between models of logic programs and models of temporal logics. In this paper, we consider the related problem of bisimulation checking which checks for equivalence of two system descriptions.

* This research was partially supported by NSF Grants EIA-9805735, EIA-9705998, CCR-9876242, CCR9732186 and IIS-0072927.

© Springer-Verlag Berlin Heidelberg 2001
Bisimulation checking is a problem of fundamental importance in verification. Many verification systems such as the Concurrency Workbench of the New Century (CWB-NC) and CADP incorporate bisimulation checkers in their tool sets. Informally, a pair of automata $M, M'$ are said to be bisimilar if for every transition in $M$ there exists a corresponding transition in $M'$ and vice versa. There has been a lot of research on efficient algorithms for bisimulation checking. But the focus of this vast body of work has been on finite-state systems, i.e., one assumes that $M$ and $M'$ are both finite state. Hennessy and Lin were the first to consider the problem of bisimilarity checking of infinite-state systems in the setting of value-passing languages. This initial work has been recently expanded. Nevertheless research on this problem remains in a state of infancy.

In this paper, we explore the use of logic programming for the above problem. We begin with a direct formulation of strong- and weak-bisimulation checking for finite-state systems (see Section 2). We show that, using query evaluation with a tabled logic programming system, this encoding yields a local bisimulation checker: one where the state space of the concurrent systems is explored only until the first evidence of non-bisimilarity is found. Note that when the systems are indeed bisimilar, the local checker explores the entire (reachable) state space. Even in this case, our bisimulation checker encoded in XSB logic programming system shows performance comparable to the global bisimulation checker in CWB-NC. For systems that are non-bisimilar, the local checker outperforms the global checker by several orders of magnitude.

We introduce symbolic transition systems (STSs) which can finitely represent infinite-state systems (see Section 3). STSs are more general than Symbolic Transition Graphs (STGs) and STGs with Assignments (STGAs) used in and respectively. We formulate symbolic bisimulation algorithms for checking two kinds of equivalences widely studied in the literature—late- and early-equivalences— as tabled constraint logic programs (see Section 4). Similar to the finite-state case, our formulation is a direct encoding of the definition of the bisimulation relations themselves. We describe how the programs can be evaluated using a constraint meta-interpreter implemented in XSB. Our experimental results show that symbolic bisimulation is practical for realistic systems. Surprisingly, our results show how even for relatively small finite-state systems, it may be better to perform symbolic bisimulation on its infinite-state counterparts. We conclude in Section 5 with a short discussion of the implications of this work.

2 Bisimilarity of Finite-State Systems

Labeled transition systems (LTSs) are widely used to capture the operational behavior of concurrent systems. An LTS is denoted by $L = (S, Act, \rightarrow)$, where $S$ is a finite set of states, $Act$ is a finite set of actions (transition labels), and $\rightarrow \subseteq S \times Act \times S$ is a transition relation. Transition from state $s$ to $t$ on an action $a$ is represented by $s \xrightarrow{a} t$. Example LTSs are given in Figure 1.