A BDD-Like Implementation of an Automata Package

Jean-Michel Couvreur

LSV, ENS Cachan, Cachan, France
couvreur@lsv.ens-cachan.fr

Finite Automata are basic structures that appear in many areas of Computer Science and other disciplines. The emergence of new tools based on automata for the manipulation of infinite structures [4, 5, 1] makes a crucial challenge of improving the efficiency of automata packages. The present work is motivated by model-checking problems. where most of the algorithms are based on fixed point computations that share many identical subcomputations. A promising track is the realization of a BDD-like package because BDDs have proved these capability to take advantage of this aspect when using cache technique. Since Bryant's original publication of BDD algorithms [2], there has been a great deal of research in the area. One of the most powerful applications of BDDs has been symbolic model checking, used to formally verify digital circuits and other finite state systems [3]. A BDD package is based on an efficient implementation of the if-then-else (ITE) operator. It uses essentially two principles:

- (1) a hash table, called unique table, maintains a strong canonical form in BDDs and stores a forest of BDDs sharing common substructures,
- (2) a hash table, called computed cache, keeps subresults when evaluating a recursive ITE operation.

Applying the BDD principle to automata is not that easy. Thus a solution to our problem has to design new principles to overcome the following difficulties: define a strong canonical form for automata, handle a forest of automata sharing common substructures, design a constant time procedure to check automata equality and an efficient hash function. Notice that classic notion of minimal automata are far from solving these problems. One needs to design a new structure, well-adapted to substructures sharing and a new algorithm transforming an automaton into this new structure, guaranteeing a strong canonical form.

In this paper we propose a data structure, called shared automata, for representing deterministic finite automata. Informally, a shared automaton codes a strongly connected component of an automaton and its exit states. Thus, an automaton may be considered as an acyclic graph of shared automata. This representation is well-adapted to substructure sharing between automata. We have designed an incremental algorithm based on this decomposition producing shared automata where states respect some canonical order. During the canonisation of an automaton, produced shared automata are stored in a unique table, guaranteeing a strong canonical form like for BDDs. In our system, automata operations, as set operations, are obtained when computing on-the-fly a non canonical representation of the result while applying the canonical algorithm.

M. Domaratzki et al. (Eds.): CIAA 2004, LNCS 3317, pp. 310–311, 2005.

Table 1. Experimentation results for some Petri nets

Model	| {Place} |	| {Transition} |	LASH	PresTaf
LEA	30	35	6min 36s	1min 13s
Manufacturing System	14	13	9min 37s	1min 4s
CSM	13	8	14min 38s	1min 2s
PNCSA	31	36	66min	3min22
ConsProd	18	14	1316min	3min55

During this evaluation, subresults are stored in a computer cache, avoiding unnecessary re-evaluation of subexpressions. We experimentally compare PresTaf, a direct implementation of the Presburger arithmetic built on the shared automata package, and the Presburger package LASH [5] based on standard automata algorithms. The goal of this experimentation is to evaluate the benefits that shared automata techniques can bring to systems using standard automata algorithms. Comparison with other kind of Presburger package is out of the scope of our experimentation. We chose a classic problem verification: the backward symbolic state space exploration for Petri nets. Experimental results (see Table 1) show the great benefit of the new canonical structure applied to this kind of problems. As BDD [6], the main factor of this benefit is the computed cache. Indeed, the iterations of a state space exploration share many subproblems.

References

1. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast Acceleration of Symbolic Transition systems. In *CAV '03*, volume 2725 of *LNCS*, pages 118–121, 2003.
2. R. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.
3. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10e20 states and beyond. *Information and Computation*, 98(2):97–109, 1998.
4. J. Elgaard, N. Klarlund, and A. Moller. Mona 1.x: new techniques for WS1S and WS2S. In *CAV '98*, volume 1427 of *LNCS*, 1998.
5. P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *TACAS'00*, volume 1785 of *LNCS*, pages 1–19, 2000.
6. B. Yang, R. E. Bryant, D. R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R. K. Ranjan, and F. Somenzi. A performance study of BDD-based model checking. In *FMCAD'98*, pages 255–289, 1998.