Compact representation of large performability models based on extended BDDs

Markus Siegle
Universität Erlangen-Nürnberg, IMMD VII
Martensstraße 3, 91058 Erlangen, Germany
siegle@informatik.uni-erlangen.de

Abstract: We discuss compact symbolic representations of large state spaces, based on binary decision diagrams (BDD). Extensions of BDDs are considered, in order to represent *stochastic* transition systems for performability analysis. Parallel composition of components can be performed in this context, without leading to state space explosion. Furthermore, we discuss state space reduction by Markovian bisimulation, also based on symbolic techniques.

1 Introduction

During system design and analysis, there often arises the problem of generating, manipulating and analysing large labelled transition systems (LTS). Such transition systems can be very difficult to handle in practice, due to memory limitations. In this paper, the focus is on *stochastic* LTSs (SLTS) for performability analysis, where transitions are associated with exponential delays. For example, SLTSs are generated from stochastic process algebra (SPA) models. Under certain conditions, such SLTSs can be interpreted as Markov chains.

We investigate a novel approach to SLTS representation and manipulation which is based on symbolic techniques. This is motivated by the fact that, in recent years, the problem of large LTS analysis has been very successfully approached by using compact symbolic representations, in particular binary decision diagrams (BDD). Most of this work took place in the context of formal verification and model checking, i.e. it deals exclusively with functional behaviour, see e.g. [2, 3, 5]. The success of symbolic techniques for functional analysis induced us to experiment with BDD-based representations of performability models.

2 Symbolic representation of transition systems

A Binary Decision Diagram (BDD) [1] is a symbolic representation of a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$. Its graphical interpretation is a rooted directed acyclic graph. As a simple example, Fig. 1 (left) shows the BDD for the function $\overline{a}\ t + a\ s\ \overline{t}$. The function value for a given truth assignment can be determined by following the corresponding edges (one-edges drawn solid, zero-edges dashed) from the root until a terminal node is reached. In the graphical representation of a BDD, for reasons of simplicity, the terminal false-node and its adjacent edges are usually omitted, see Fig. 1 (right).

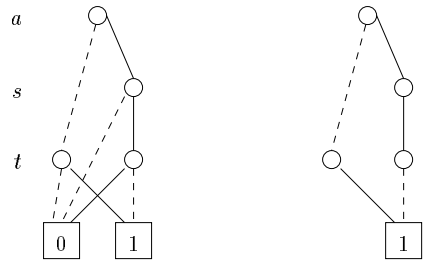

Figure 1: BDD for $\overline{a}\ t + a\ s\ \overline{t}$, simplified graphical representation (right)

BDDs provide a canonical representation for Boolean functions, once the ordering of Boolean variables is fixed. Given a Boolean function, the size of the resulting BDD (the number of nodes) is highly dependent on the chosen variable ordering.

A LTS consists of states and transitions between states. In addition, each transition carries a label, associating the transition with a particular action, see Fig. 2 (left). For the symbolic representation of a LTS, states and action names are encoded as binary vectors. We use Boolean variables act_i to encode the action, and s_i and t_i to encode the source and target state of a transition (the sets of actions and states are assumed to be finite, which is a prerequisite for the symbolic encoding). Fig. 2 (middle) shows how transitions are encoded by binary vectors. The Boolean function corresponding to the whole LTS is simply the disjunction of all the encodings of the individual transitions.

Experience has shown that the resulting BDD is small if the ordering of Boolean variables is chosen such that the variables encoding the action come first, followed by the variables for source and target state interleaved [5]. In particular, this ordering is advantageous in view of the parallel composition operator discussed below. Fig. 2 (right) depicts the resulting BDD under this variable ordering.

3 Stochastic transition systems: Decision Node BDD

In *stochastic* transition systems (SLTS), each transition has as a second attribute a real number, the *rate* of the transition (we assume that actions have exponential delays). Clearly, pure BDDs do not of-

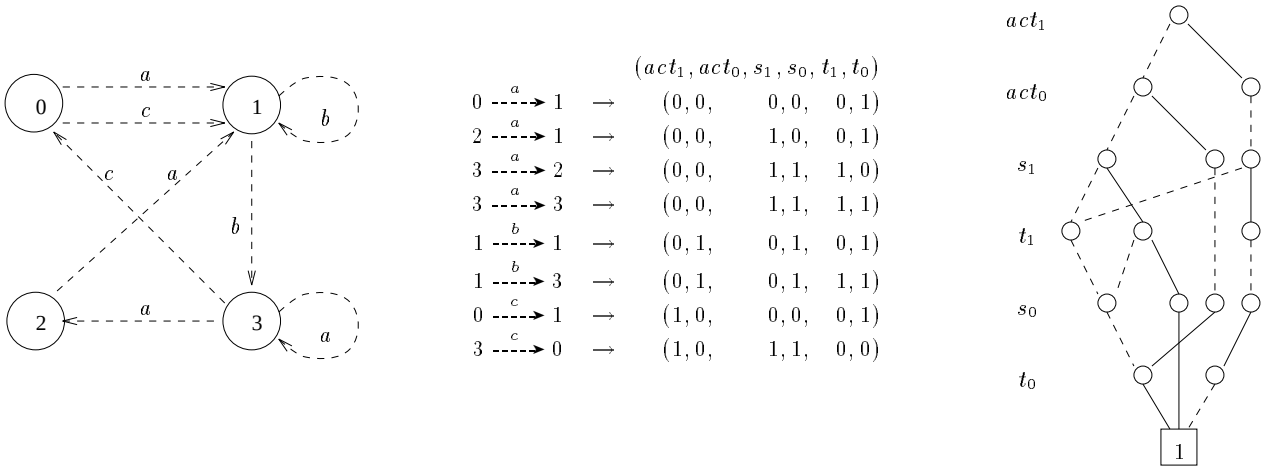

Figure 2: LTS, transition encoding and corresponding BDD

fer any mechanism for representing the (numerical) rate information. Multi-terminal BDDs [4] and Edge-valued BDDs [11] have been suggested for representing functions of the type $f : \{0,1\}^n \rightarrow I\!R$. However, in both of these the efficiency of the data structure, due to the sharing of isomorphic subtrees, is diminished. Based on this observation, we decided to develop a different approach which we call decision-node BDD (DNBDD) [13]. The distinguishing feature of DNBDDs is that the basic BDD structure remains completely untouched when moving from an LTS encoding to an SLTS encoding. The additional rate information is attached to specific edges of the BDD in an orthogonal fashion.

A non-terminal BDD-node is called *decision node* if both its successor nodes are different from the terminal false-node. A *decision node BDD (DNBDD)* is a BDD enhanced by a function

$$rates : Paths \rightarrow (I\!R)^+$$

(where $Paths$ is the set of paths from the root node to the terminal true-node) such that for any such path p, $rates(p) \in (I\!R)^{2^k}$ if k is the number of "don't cares" on path p (a "don't care" is a variable not explicitly checked on a path). The rate list $rates(p)$ associated with a path p is attached to the outgoing edge of the last decision node on that path.

This concept is illustrated in Fig. 3 (in the figure, decision nodes are drawn black). The SLTS of this example contains four transitions. Therefore there are four Boolean assignments evaluating to *true*, each of which is mapped onto a rate as shown in the middle part of the figure. The first two assignments share the same path through the DNBDD, a path which has a "don't care" in the Boolean variable s. Therefore, the corresponding rate list (λ, μ) has length two.

The practical realisation of the DNBDD concept introduced so far induces the following problem: There are situations, where several paths share their last decision node. In such a case, several rate lists would be assigned to the same edge. We introduce a pointer structure, the *rate tree* of a DNBDD, such that the mapping from paths to rate lists is one-to-one [13].

4 Symbolic Parallel Composition and Minimisation

When building structured performability models, an important operation is parallel composition of components. The parallel composition operator of a SPA, for instance, can be realised directly and efficiently on the BDD-based representation of the two operand processes. The resulting DNBDD describes all transitions which are possible in the product space of the two processes. Given a pair of initial states, only part of the product space may be reachable due to synchronisation conditions. Reachability analysis can be performed directly on the resulting DNBDD, restricting it to those transitions which originate in reachable states.

Consider the parallel composition of two processes, which can be written in process algebraic notation as $P = P_1 \,|[A]|\, P_2$, and assume that the DNBDDs which correspond to processes P_1 and P_2 have already been generated and are denoted $\mathcal{P}_1$ and $\mathcal{P}_2$. The set of synchronising actions A (i.e. the set of actions which

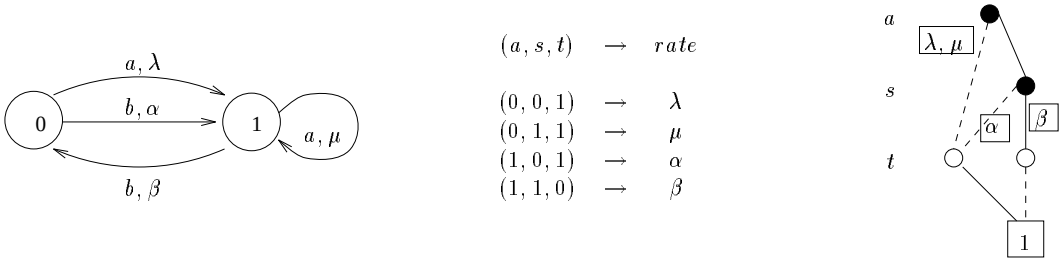

Figure 3: SLTS, mapping of Boolean assignments to rates and corresponding DNBDD

processes P_1 and P_2 must perform together) can also be coded as a BDD, namely $\mathcal{A}$. The DNBDD $\mathcal{P}$ which corresponds to the resulting process P can then be written as a Boolean expression:

$$\begin{aligned}\mathcal{P} \quad = \quad & (\mathcal{P}_1 \wedge \mathcal{A}) \wedge (\mathcal{P}_2 \wedge \mathcal{A}) \\ \vee \quad & (\mathcal{P}_1 \wedge \overline{\mathcal{A}} \wedge Stab_{P_2}) \\ \vee \quad & (\mathcal{P}_2 \wedge \overline{\mathcal{A}} \wedge Stab_{P_1})\end{aligned}$$

Note that, during the evaluation of this expression, the rate tree of DNBDD $\mathcal{P}$ is computed from the rate trees of $\mathcal{P}_1$ and $\mathcal{P}_2$. The term on the first line of the above expression is for the synchronising actions in which both P_1 and P_2 participate. The term on the second (third) line is for those actions which P_1 (P_2) performs independently of P_2 (P_1) — these actions are all from the complement of the set A, which in Boolean terms is expressed as $\overline{\mathcal{A}}$. The meaning of $Stab_{P_2}$ ($Stab_{P_1}$) is a BDD which expresses stability of the non-moving partner of the parallel composition, i.e. the fact that the source state of process P_2 (P_1) equals its target state.

Under the variable ordering described above, the growth of the BDD-based representation is only *linear* in the number of parallel components (since parallelism is implicit), as opposed to the *exponential* growth when working with transition systems and the usual interleaving of actions. This is a major advantage of the symbolic approach over conventional methods for state space representation.

A SLTS can be minimised based on an equivalence relation defined on the set of states. The idea is to reduce the state space by representing all equivalent states by a single macro state. Such a minimisation technique can be applied to the DNBDD representation of the SLTS, i.e. the minimisation is entirely based on DNBDD operations.

The equivalence relation on which we focus is known as Markovian bisimulation [7], which has a strong connection to classical Markov chain lumpability [9]. Informally, two states are Markov-bisimilar (members of the same equivalence class) if from both states all equivalence classes can be reached (in one step) by the same actions and with the same cumulative rate, where the *cumulative rate* of action a from state x to class C_i is defined as

$$\Lambda(x, a, C_i) = \sum_{x \xrightarrow{a,\lambda} y,\ y \in C_i} \lambda$$

The cumulative rate can be easily computed in the DNBDD context.

Fig. 4 illustrates how the state space S is partitioned into disjoint classes C_1, C_2, ... In the figure, the cumulative rate of action a from state x_1 to class C_2 is λ, whereas the cumulative rate of action a from state x_2 to class C_2 is $\mu + \delta$. Thus, for x_1 and x_2 to be Markov-bisimilar the condition $\lambda = \mu + \delta$ has to hold.

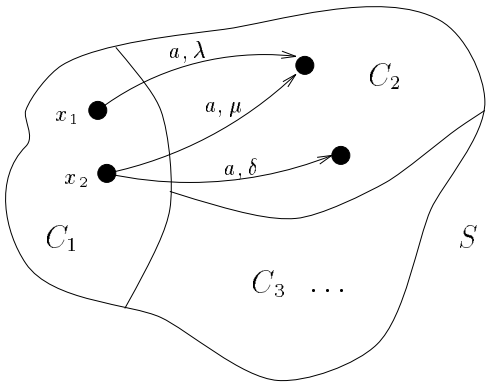

Figure 4: Partitioning of state space S

Algorithms for Markovian bisimulation traditionally follow an iterative refinement scheme [12, 6, 10]. This means that starting from an initial partition

which consists of a single class (containing all states), classes are refined until the obtained partition corresponds to a Markovian bisimulation. The result thus obtained is the largest existing Markovian bisimulation, in a sense the "best" such bisimulation, since it has a minimal number of equivalence classes.

For the refinement of a partition, the notion of a "splitter" is very important. A splitter is a pair (a, C_{spl}), consisting of an action a and a class C_{spl} (whose members play the role of target states). During refinement, classes of states (which in this context play the role of source states) are split into subclasses in an iterative fashion until no further splitting is needed. Splitting of a class C_i with respect to a splitter (a, C_{spl}) results in the computation of subclasses $C_{i_1}, C_{i_2}, \ldots, C_{i_k}$ $(k \geq 1)$, such that the cumulative rate $\Lambda(x, a, C_{spl})$ is the same for all the states x belonging to the same subclass C_{i_j}.

The key to efficient partition refinement algorithms lies in an intelligent management of the dynamic set of splitters together with a memory-efficient implementation based on BDDs. For details on symbolic Markovian bisimulation see [8].

5 Conclusion

The advantages of symbolic representations for performability analysis are obvious: The state space of a complex system can be built from small components by applying the (DN)BDD-based parallel composition operator step by step. After every parallel composition step, the intermediate LTS can be minimised without leaving the symbolic world. Thus, the use of BDD-based representations quite ideally supports the concept of compositional reduction, avoiding large state spaces at every step of model construction.

Certainly, a lot of work remains to be done in this field, but from our experience we can already say that the symbolic representation of large state spaces is a very promising approach!

References

[1] R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE ToCS*, C-35(8):677–691, August 1986.

[2] R.E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[3] J.R. Burch, E.M. Clarke, and K.L. McMillan. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, (98):142–170, 1992.

[4] E.M. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal Binary Decision Diagrams: An efficient data structure for matrix representation. In *IWLS: Int. Workshop on Logic Synthesis*, Tahoe City, May 1993.

[5] R. Enders, T. Filkorn, and D. Taubner. Generating BDDs for symbolic model checking in CCS. *Distributed Computing*, (6):155–164, 1993.

[6] J.C. Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13:219–236, 1989.

[7] H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, p. 71–88, July 1994.

[8] H. Hermanns and M. Siegle. Computing Bisimulations for Stochastic Process Algebras using Symbolic Techniques. submitted to *Sixth Int. Workshop on Process Algebra and Performance Modelling*, 1998.

[9] J. Hillston. *A Compositional Approach to Performance Modelling.* PhD thesis, Univ. of Edinburgh, 1994.

[10] P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86:43–68, 1990.

[11] Y.-T. Lai and S. Sastry. Edge-Valued Binary Decision Diagrams for Multi-Level Hierarchical Verification. In *29th Design Automation Conference*, p. 608–613. ACM/IEEE, 1992.

[12] R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM JoC*, 16(6):973–989, 1987.

[13] M. Siegle. Technique and tool for symbolic representation and manipulation of stochastic transition systems. TR IMMD 7 2/98, Universität Erlangen-Nürnberg, 1998. http://www7.informatik.uni-erlangen.de/~siegle/own.html.