BDD extensions for stochastic transition systems

Markus Siegle
Universität Erlangen-Nürnberg, IMMD 7,
Martensstraße 3, 91058 Erlangen, Germany
siegle@informatik.uni-erlangen.de

Abstract: A BDD (Binary Decision Diagram) is a compact canonical representation of a Boolean function. While BDDs are well-established in the area of functional system verification, their use for the purpose of performance analysis is a new idea.

We use BDDs to represent labelled transition systems which arise from higher-level model specifications such as stochastic process algebras or structured stochastic Petri nets. BDDs offer a compact representation of transition systems with very large state space. They are therefore promising candidates for alleviating the problem of state space explosion. However, as a survey of the relevant literature shows, the question of how to code stochastic information in a BDD context had not yet been answered satisfactorily.

We offer a new solution to this problem, concentrating on the Markovian case. A new data structure, Decision-node BDD (DNBDD), is introduced and used to represent stochastic transition systems. It is shown that DNBDD have important advantages compared to earlier approaches. A DNBDD is structurally identical with the corresponding ordinary BDD, but some of its nodes — which we call decision nodes — carry additional information.

Generation and manipulation algorithms for DNBDDs are then discussed. In particular, we show how a DNBDD can be constructed in a stepwise fashion from a stochastic LTS, and how the parallel composition of two stochastic transition systems can be performed in the DNBDD context. DNBDD-based bisimulation minimisation (which corresponds to Markov lumpability) is also discussed.

1. Introduction

A BDD (Binary Decision Diagram) [1] is a compact canonical representation of a Boolean function. While BDDs are well-established in the area of functional system verification, their use for the purpose of performance analysis is a new idea.

A simple BDD-example is shown in fig. 1. It corresponds to the Boolean function $\overline{a}\,t \vee a\,s\,\overline{t}$. In a BDD, the function value for a particular set of input values is determined by following either the one- or zero-edge, starting from the root node and progressing from node to node. Every node correponds to a certain input variable. Skipping a node level means that there is a "don't care" in that input variabe. We use the following drawing conventions: One-edges are drawn solid, zero-edges dashed, and edges leading to the terminal false-node as well as the false-node itself are omitted.

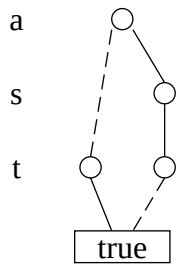

Figure 1:
Simple BDD example

We use BDDs to represent labelled transition systems (LTS) which arise from higher-level system specifications such as stochastic process algebras (SPA) [2] or structured stochastic Petri nets (SPN). It is well-known that BDDs offer a compact representation of transition systems with very large state space. They are therefore promising candidates for alleviating the problem of state space explosion which is encountered so frequently in analytical performance analysis. However, as a survey of the relevant literature shows, the question of how to code

to appear in:
Proc. of UKPEW 97, Ilkley, UK, July 1997

stochastic information in a BDD context had not yet been answered satisfactorily.

We offer a new solution to this problem, concentrating on the Markovian case where simply a rate has to be stored for every transition of the LTS. A new data structure, Decision-node BDD (DNBDD), is introduced and used to represent stochastic transition systems. It is shown that DNBDD have important advantages compared to earlier approaches. A DNBDD is structurally identical with the corresponding ordinary BDD, but some of its nodes — which we call decision nodes — carry additional information.

The rest of the paper is organised as follows: Sec. 2 contains a survey of previous approaches to the problem of storing numerical (stochastic) information in a BDD-context. Sec. 3 explains our new DNBDD data structure. Sec. 4 discusses important algorithms for DNBDDs, and Sec. 5 contains some concluding remarks.

2. BDD extensions for performance evaluation

We present several possibilities for representing rate information in a BDD through a very simple common example. Fig. 2, on top, shows a LTS in which each transition carries information about the action type (a, b) and the rate at which the transition takes place (λ, μ, δ). Common to all BDD alternatives shown in the figure, states 0 and 1 are coded in the Boolean variables s (source state) and t (target state) as 0 and 1, respectively. The action type is coded in the Boolean variable act, where a maps to 0 and b maps to 1. The characteristic Boolean function for this LTS is thus given by the expression $\overline{act}\,\overline{s}\,t \vee act\,s\,\overline{t} \vee \overline{act}\,s\,t$, where every minterm codes one transition.

The first diagram in fig. 2 shows the **purely functional** BDD, disregading all rate information.

Using **symbolic encoding of rates** [3], the rate is coded as a Boolean vector $(r_n, r_{n-1}, \ldots, r_0)$. In the presence of k distinct rate values, the vector has to have at least length $\lceil \log_2 k \rceil$. In the example, rates λ, μ and δ are coded by 00, 01 and 10. The advantage of this approach is its simplicity. But there are the following disadvantages: Extra Boolean variables are needed for coding the rates. The approach is somewhat inflexible, since more bits may be needed when new rates are introduced, and it is not clear how rate arithmetic could be done. The size of the BDD grows, since fewer subgraphs can be shared than in the corresponding functional BDD.

Edge-valued BDDs (EVBDDs) have been introduced by Lai and Sastry [4] for multi-level hierarchical circuit verification. EVBDDs offer a canonical representation of functions with Boolean domain and numeric range. In an EVBDD, each one-edge carries a numeric value. In addition, there is an incoming edge to the root which also carries a numeric value. The function value for a particular Boolean input vector is given by the sum of the edge values on the corresponding path through the EVBDD. A LTS with rate information can be represented by an EVBDD by assigning the correct rate values to the legal transition encodings and 0 to all illegal transition encodings. The advantages of EVBDD are their canonicity, and the fact that generation and manipulation algorithms are known. However, since every truth assignment to the input variables is explicitely represented on a path from the root to the terminal node, the BDD size increases quite dramatically.

Hachtel et al. use **Multi-terminal BDD**s (MTBDDs) for representing Markov chains [5]. MTBDDs have multiple terminal nodes which carry numeric values. Algorithms for MTBDDs are available. This straight forward approach has the disadvantage that the size of the BDD increases because there is less subgraph sharing than in the corresponding BDD with just a single terminal node.

The last alternative, DNBDD, is discussed in greater detail in the Sec. 3.

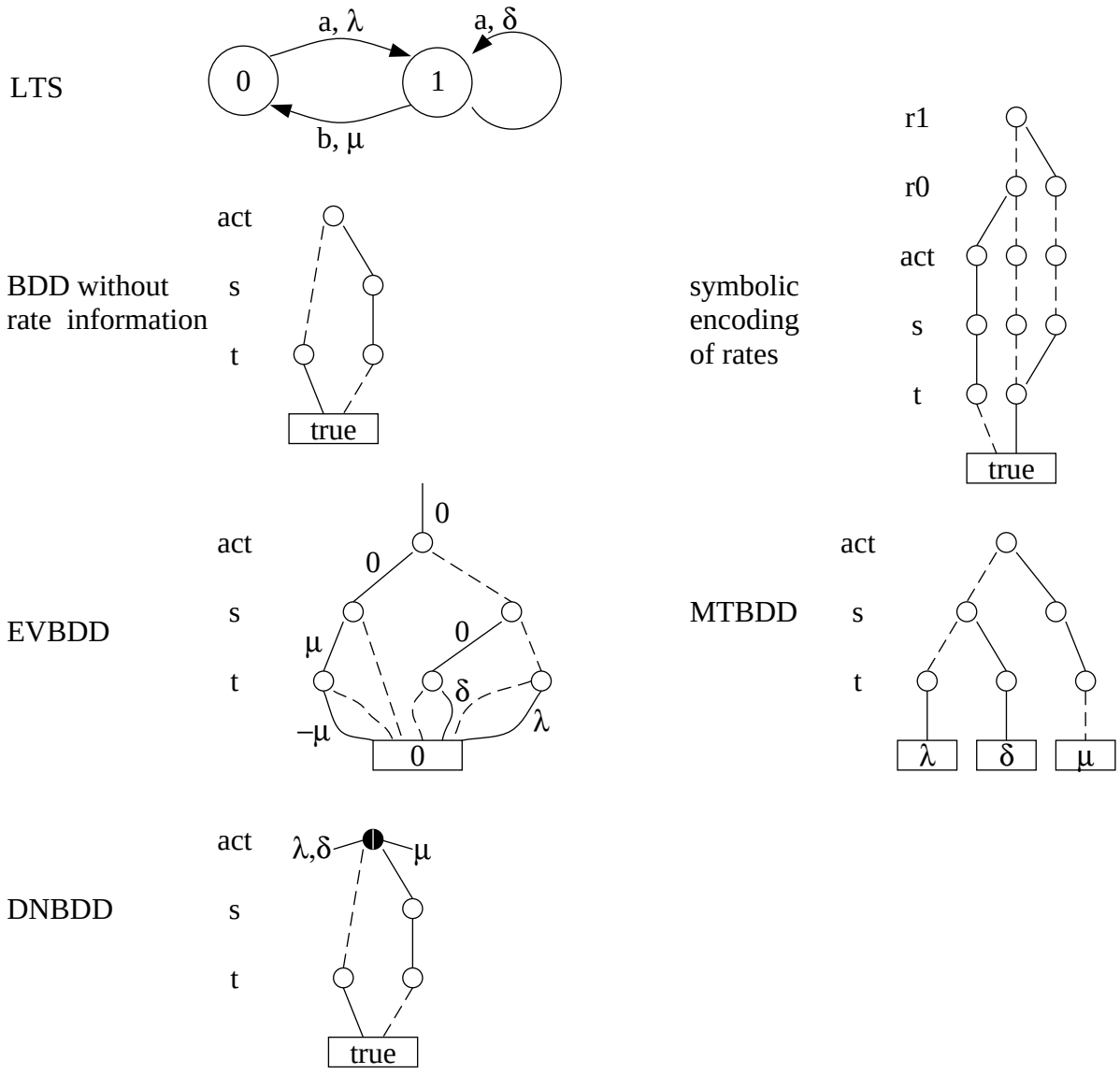

Figure 2: Possible BDD extensions for storing rate information

3. Decision-node BDDs

In this section, we introduce a new data structure, called **Decision-node BDD** (DNBDD), which is capable of representing a LTS and storing rate information for every transition of the LTS, while preserving the basic structure of purely functional BDDs. The last diagram in fig. 2 shows the DNBDD for the simple common example.

We first define the notion of a BDD node's zero-subgraph (one-subgraph).

Def: The subgraph which can be reached from a BDD node through its outgoing zero-edge (one-edge) is called the **zero-subgraph (one-subgraph)** of that node.

Starting from the BDD without rate information, our task is to assign a rate, a real number, to every path of the BDD. The idea of DNBDDs is to characterise the path by a set of decision nodes.

Def: A BDD node is called **decision node** iff the terminal true-node is reachable through both outgoing edges. Furthermore, by definition, the root node is always a decision node.

Our new DNBDDs have extensions with respect to ordinary BDDs. In addition to the ordinary BDD edges, a DNBDD contains extra edges (decision edges) linking its decision nodes.

Def: A **decision-edge** is a bidirectional edge between two decision nodes of a DNBDD. As with ordinary BDD edges, there are zero-decision-edges and one-decision-deges.

A decision node d contains the following additional information:

1. In the presence of more decision nodes in the zero-subgraph of d, let d_0 be the first such decision node. In this case, there is a zero-decision-edge from d to d_0.
2. If there is no decision node in the zero-subgraph of d, the zero-subgraph is simply a path from d to the terminal true-node. We then use the expression "zero-path" instead of "zero-subgraph". In this case, d contains (a pointer to) a rate list for the zero-path. The k-th entry of the rate list defines the rate of the k-th LTS-transition sharing this zero-path, where transitions are ordered lexicographically according to their Boolean encoding.
3. 1. and 2. also apply if "zero" is replaced by "one" and d_0 is replaced by d_1.

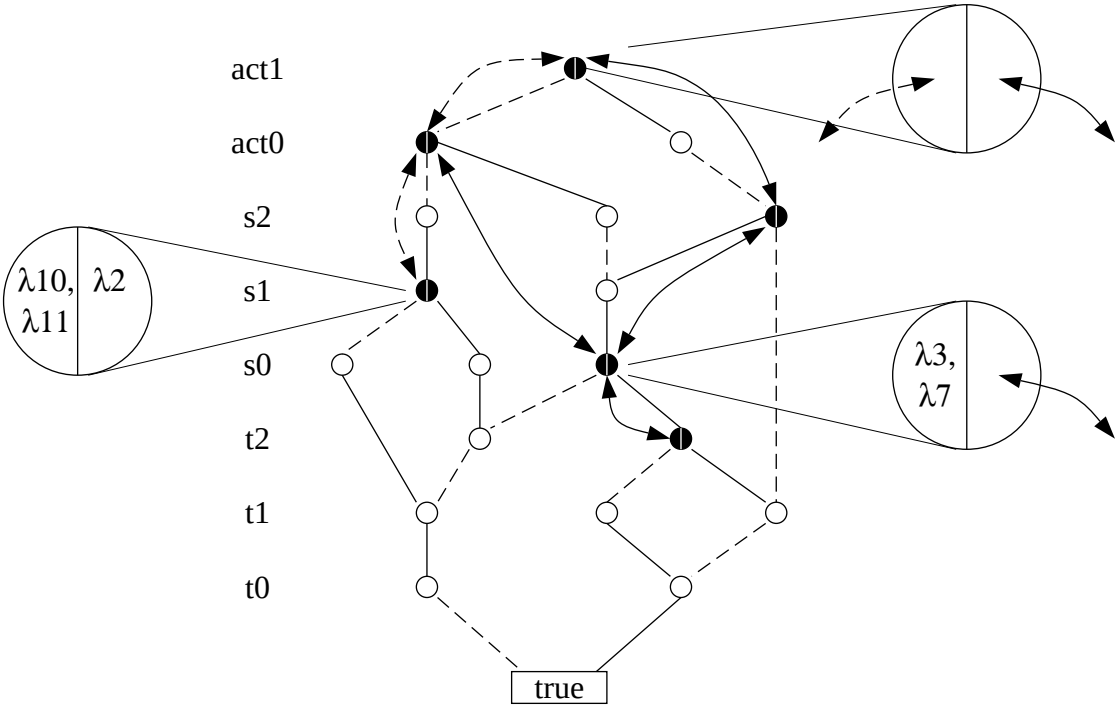

Figure 3: DNBDD with some decision nodes zoomed

Fig. 3 shows a DNBDD which is the result of encoding a LTS with rate information. This LTS has 17 transitions, some of which are listed in table 1. In the figure, decision nodes are drawn in black. Three of the decision nodes are shown zoomed. The first of them, the root, is linked by two decision edges to the first decision nodes in its zero- and one-subgraph. Regarding the decision node on the level of Boolean variable s_1, we observe that neither its zero-subgraph nor its one-subgraph contain any decision nodes. Therefore this decision node contains a rate list for the transitions corresponding to its zero-path ($\lambda_{10}, \lambda_{11}$) and a rate-list for the (single) transition corresponding to its one-path (λ_2). The third of the zoomed decision nodes is a combination of the two cases: While there exists a decision node in its one-subgraph (therefore the one-decision-edge) there does not exist a decision node in its zero-subgraph (therefore the rate list (λ_3, λ_7)).

The advantages of DNBDDs are quite obvious: DNBDDs are a canonical representation for labelled transition systems enhanced by stochastic rate information. The basic structure of the corresponding BDD is preserved. As a consequence, known algorithms for BDD generation and manipulation can be easily extended to the DNBDD case.

trans. number	action		source state		target state		rate
	type	act_1, act_0	decimal	s_2, s_1, s_0	decimal	t_2, t_1, t_0	
1	a	00	5	101	2	010	λ_{10}
2	a	00	5	101	6	110	λ_{11}
3	a	00	7	111	2	010	λ_2
4	b	01	2	010	2	010	λ_3
5	c	10	6	110	2	010	λ_7
6	c	10	7	111	3	011	λ_8
⋮							⋮

Table 1: Some transitions of the LTS coded by the DNBDD of fig. 3

As an example for such an algorithm, fig. 4 illustrates one step during generation of the DNBDD of fig. 3. The first 5 transitions (see table 1) are already coded in the DNBDD on the left. Note that every node of a DNBDD has a unique identity, given by an integer number. In this step, the 6th transition, coded in the DNBDD in the middle, is added, which requires a Boolean "or"-operation. The resulting DNBDD is shown on the right. The call tree for this "or"-operation is shown in fig. 5. The arguments to the procedure "or" are node numbers. On the top level, "or" is called for the root nodes of the operand DNBDDs (nodes 39 and 47). The procedure "makenode" generates a node for the Boolean variable which is given as its first argument. The numbers of the new nodes generated by "makenode" are given at the right of fig. 5. The resulting call tree is exactly the same as for constructing the corresponding BDD without rate information, i.e. the algorithm for an "or"-operation is basically the same for BDDs and DNBDDs (this is also true for other operations). The only

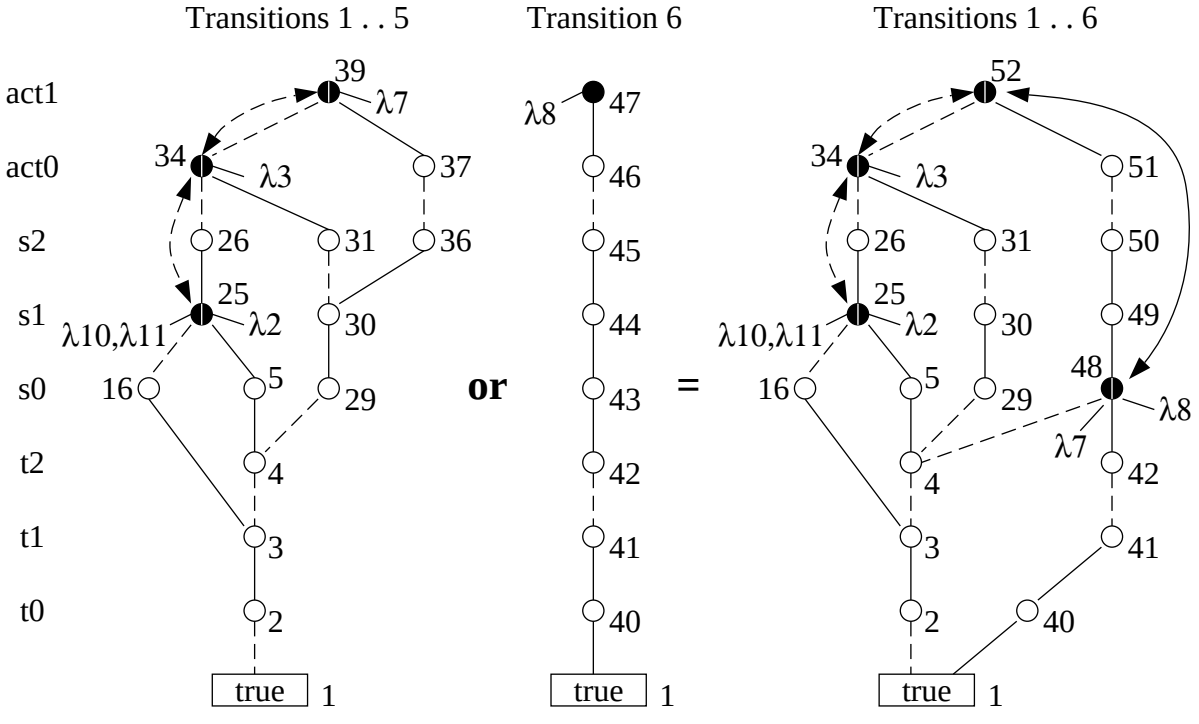

Figure 4: Example for DNBDD generation

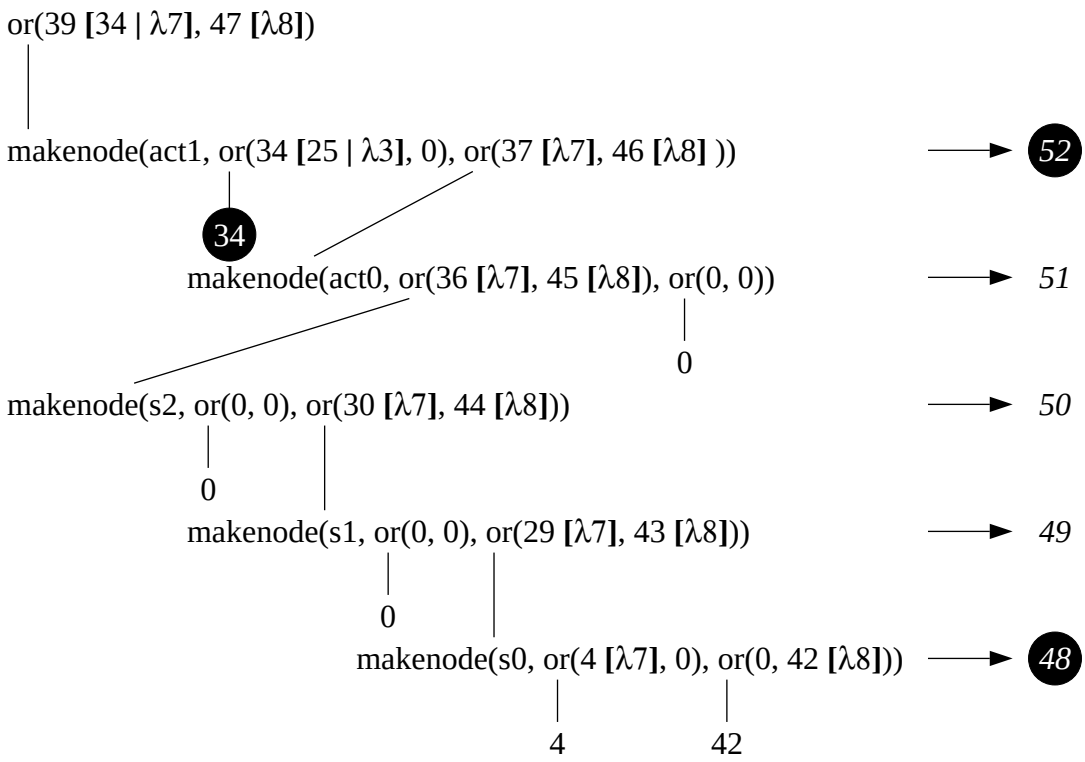

Figure 5: Call tree for the "or"-operation illustrated in fig. 4

modification necessary is to add the information in brackets []. This is information about decision nodes and rate lists. When a new node is created (procedure "makenode") it can be immediately decided, whether or not the new node is a decision node (the new node is a decision node iff both its successor nodes are non-0). If this is the case, the information for that decision node (decision edge(s), rate list(s)) is known.

4. Operations on DNBDDs

4.1. Parallel composition of DNBDDs

For BDDs representing labelled transition systems which originate from process algebras, an important operation is parallel composition. Let BDD_A and BDD_B be two BDDs which correspond to two processes A and B. Let S be the subset of actions on which the two processes synchronise. This set can also be coded as a BDD, namely BDD_S. The BDD corresponding to the parallel composition of A and B, $BDD_{\|}$, can be written as a Boolean expression:

$$\begin{aligned} BDD_{\|} = &(BDD_A \wedge BDD_S) \wedge (BDD_B \wedge BDD_S) \\ &\vee \left(BDD_A \wedge \overline{BDD_S} \wedge Stab_B\right) \\ &\vee \left(BDD_B \wedge \overline{BDD_S} \wedge Stab_A\right) \end{aligned} \tag{1}$$

The term on the first line is for the synchronizing actions in which both A and B participate. The term on the second (third) line is for those actions which A (B) performs independently of B (A) — these actions are all from the complement of S. The meaning of $Stab_A$ ($Stab_B$) is a BDD which expresses stability, i.e. the fact that the source state of process A (B) equals the target state.

The result, $BDD_{\|}$, describes all transitions which are possible in the product space of the two processes. Given a pair of initial states for A and B, only part of the state space may be reachable due to synchronisation conditions. However, reachability analysis can be performed

on the BDD representation, restricting $BDD_{||}$ to those transitions which originate in reachable states.

Parallel composition of DNBDDs can be carried out in much the same manner as described by eq. (1). One question to be answered is about the result rate of the synchronizing actions. Depending on the application, different definitions for the result rate may apply. Typical examples are the maximum, minimum, sum or product of the two partner rates. With DNBDD, the result rate will be calculated from the two partner rates during an "and"-operation (first line of eq. (1)), which has a similar call tree as the one shown in fig. 5. The structure of the call tree does not depend on the chosen alternative (maximum, minimum, . . .) — any of the listed cases can be incorporated easily at that point, i.e. DNBDDs cover any of those cases.

4.2. Bisimulation minimisation with DNBDDs

Bisimulation [6] is needed to reduce the state space of large transition systems. The idea is to partition the state space such that every subset contains equivalent states. Informally, two states are equivalent if they represent the same beaviour, i.e. if they enable the same actions which must lead to successor states which again are equivalent. The standard algorithm to compute the bisimulation relation uses iterative refinement.

A bisimulation algorithm which works on BDDs is described in [7] for the non-stochastic case. Its output is a BDD characterizing pairs of equivalent states. Such an algorithm can be extended to the stochastic case, i.e. to the case of Markovian bisimulation, working on DNBDDs. To do this, it is necessary in iteration j to compute the relation $E_{j,a,\lambda}(x, y)$ for every action a. A pair of states (x, y) is contained in that relation iff action a can take place in state x, leading to a state z which lies in the same equivalence class as y. The rate λ associated with a pair (x, y) is the sum of all the rates of a-actions from state x to states equivalent to y.

5. Conclusion

The new data structure introduced in this paper, DNBDD, has decisive advantages when representing stochastic transition systems. Most important, the structure of purely functional BDDs is preserved, and basic algorithms can be easily adapted. As a next step in our work, experiments with DNBDDs on real-world examples have to be carried out.

References

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

[2] N. Götz, H. Hermanns, U. Herzog, V. Mertsiotakis, and M. Rettelbach. *Quantitative Methods in Parallel Systems*, chapter Constructive Specification Techniques – Integrating Functional, Performance and Dependability Aspects. Springer, 1995.

[3] O. Kluge. Binäre Entscheidungsdiagramme für stochastische Prozeßalgebren. Studienarbeit, Universität Erlangen–Nürnberg, IMMD 7, Dezember 1996.

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

[5] G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian Analysis of Large Finite State Machines. *IEEE Trans. on CAD*, 15(12):1479–1493, Dec. 1996.

[6] R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.

[7] A. Bouali and R. de Simone. Symbolic Bisimulation Minimisation. In *Computer Aided Verification*, pages 96–108, 1992. LNCS 663.