Compositional Representation and Reduction of Stochastic Labelled Transition Systems based on Decision Node BDDs

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

Abstract

Compact symbolic representations of large labelled transition systems, based on binary decision diagrams (BDD), are discussed. Extensions of BDDs are considered, in order to represent *stochastic* transition systems for performability analysis. We introduce Decision Node BDDs, a novel stochastic extension of BDDs which preserves the structure and properties of purely functional BDDs. It is shown how 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 entirely based on symbolic techniques. Together, parallel composition and state space reduction enable a compositional approach to the stochastic modelling of concurrent systems.

1 Introduction

In many areas of system design and analysis, there is the problem of generating, manipulating and analysing large state spaces, usually represented in the form of labelled transition systems (LTS). Such transition systems are often very difficult to handle in practice, due to memory limitations. The use of structured models – consisting of a number of interacting components – makes it easier to specify complex systems. However, the size of the state space of structured models is usually exponential in the number of components. In this paper, we present a novel approach to an efficient representation of such structured models which avoids the exponential blow-up.

We focus on *stochastic* LTSs, where each transition is associated with a stochastic delay. Such stochastic LTSs (SLTS) occur during performance evaluation and performability analysis of distributed systems. For example, stochastic LTSs are generated during the analysis of Markovian stochastic process algebra (SPA) models. Abstracting from their functional information, SLTSs can be interpreted as Markov chains and analysed by numerical methods.

We propose an approach to SLTS representation and manipulation which is based on symbolic techniques. Our work has been motivated by the fact that, in recent years, the problem of large LTS analysis has been very successfully approached by using 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. [4, 5, 8, 10]. Experience showed that symbolic representations make it possible to handle much larger state spaces than traditional methods. The success of symbolic techniques for functional analysis induced us to experiment with BDD-based representations of *stochastic* LTS. Inclusion of non-functional information into BDDs is possible [22, 9, 6, 13], but representation of stochastic LTS has not got much consideration in the past. Among the few publications in this line are [13, 12, 16].

In this paper, the data structure DNBDD [24, 25] is formally introduced. This data structure can capture not only functional, but also *temporal (stochastic)* information. It is tailored for SLTS and allows a very compact representation. It is shown that known algorithms for BDDs can be adapted and enhanced for the new data structure. We discuss a DNBDD-based procedure for the parallel composition of submodels which can be used for efficiently building complex models from small components, without inducing the problem of state space explosion. Furthermore, we describe a minimisation algorithm for stochastic LTS which is based on the concept of Markovian bisimulation and works entirely on the new DNBDD data structure.

The paper is organised as follows: Sec. 2 provides an introduction to labelled transition systems and BDDs and explains how LTSs can be represented by BDDs. Sec. 3 explains how *stochastic* LTSs can be represented by the extended data structure DNBDD. In Sec. 4, we explain how DNBDD-based parallel composition of components and DNBDD-based state space minimisation work. After a brief description of our prototype tool (Sec. 5) the paper concludes with Sec. 6.

2 Symbolic representation of labelled transition systems

2.1 Labelled Transition Systems

Informally, a transition system consists of states and transitions between states. The transitions are labelled with symbols from a set L which may correspond, for example, to the set of actions Act of a stochastic process algebra. A LTS can be graphically interpreted as a directed graph (with a distinguished initial node) whose edges are labelled with labels from L. Fig. 1 shows an example LTS.

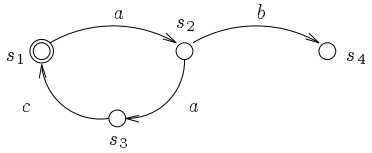


Figure 1: Example of a labelled transition system (LTS)

Def: *Labelled Transition System (LTS)*
Let $S = \{s_1, s_2, \ldots\}$ be a finite set of states, s_1 being the initial state. Let $L = \{l_1, l_2, \ldots\}$ be a finite set of labels. Let $\rightarrow$ be a relation

$$\rightarrow \subseteq S \times L \times S$$

We call $\mathcal{T} = (S, L, \rightarrow, s_1)$ a *Labelled Transition System*. If $(x, a, y) \in \rightarrow$, we write $x \stackrel{a}{\rightarrow} y$. ■

Note that in our definition the set of states S is assumed to be finite. Finiteness of the state space is a prerequisite for the symbolic encoding of states and transitions which is described below.

2.2 Binary Decision Diagrams

A Binary Decision Diagram (BDD) [4, 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 with one or two terminal nodes. Each non-terminal node is associated with a Boolean variable. The graph is essentially a collapsed binary decision tree in which isomorphic subtrees are merged and don't care nodes are skipped. As a simple example, Fig. 2 (left) shows the BDD for the function $(\overline{a} \wedge t) \vee (a \wedge s \wedge \overline{t})$. 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 node 0 and its adjacent edges are usually omitted, see Fig. 2 (right).

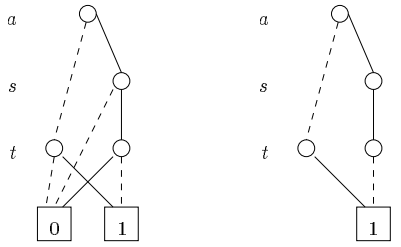


Figure 2: BDD for $(\overline{a} \wedge t) \vee (a \wedge s \wedge \overline{t})$, simplified graphical representation (right)

Def: *Binary Decision Diagram (BDD)*
A (reduced, ordered) *Binary Decision Diagram* $\mathcal{B} = (Nodes, var, low, high)$ is defined by

- a finite set of nodes $Nodes = T \cup NT$, where T (NT) is the set of terminal (non-terminal) nodes, $|\,Nodes\,| \geq 1$, $T \subseteq \{0,1\}$,
- a function $var : NT \rightarrow Vars$, where $Vars$ is a set of Boolean variables $Vars = \{v_1, \ldots, v_n\}$ with a fixed ordering relation "$<$", such that $v_1 < \ldots < v_n$,
- a function $low : NT \rightarrow Nodes$ and a function $high : NT \rightarrow Nodes$,

with the following constraints:

1. $\forall x \in NT : low(x) \in T \vee var(low(x)) > var(x)$
   $\forall x \in NT : high(x) \in T \vee var(high(x)) > var(x)$
   (respect ordering relation),
2. $\forall x \in NT : low(x) \neq high(x)$
   (no redundant (don't care) nodes),

3. $\forall x, y \in NT: \quad var(x) \neq var(y)$
$\vee\ low(x) \neq low(y)$
$\vee\ high(x) \neq high(y)$
(no isomorphic nodes) ■

Each BDD node unambiguously defines a Boolean function. The definition is based on the so-called Shannon expansion which states that

$$f(v_1, \ldots, v_n) =$$
$$(\overline{v_1} \wedge f(0, v_2, \ldots, v_n)) \vee (v_1 \wedge f(1, v_2, \ldots, v_n))$$

Def: *Boolean function Bool(x)*
The *Boolean function Bool(x) represented by a BDD-node* $x \in Nodes$ is recursively defined as follows:

- if $x \in T$ then $Bool(x) = x$, i.e. either 0 or 1,
- else (if $x \in NT$)
$$Bool(x) = \left(\overline{var(x)} \wedge Bool(low(x))\right) \vee \left(var(x) \wedge Bool(high(x))\right)$$ ■

For convenience, we define the following notation: Let $\{b_1, \ldots, b_k\} \in \{0,1\}^k$ be a fixed assignment for a subset of the Boolean variables $\{v_{i_1}, \ldots, v_{i_k}\} \subseteq Vars$. The Boolean function represented by node $x \in Nodes$ under this assignment is $Bool(x \mid v_{i_1} = b_1, \ldots, v_{i_k} = b_k) = Bool(x)|_{v_{i_1}=b_1,\ldots,v_{i_k}=b_k}$. Most times one is interested in the case where x corresponds to the BDD root.

It is known, that BDDs provide a canonical representation for Boolean functions, i.e. a given Boolean function has a unique BDD representation (assuming a fixed ordering of the Boolean variables) [4]. For this reason, some computationally hard problems (e.g. satisfyability, test-for-tautology, equivalence) can be solved in constant or linear time, once the BDD representation of the Boolean functions involved is known [1]. Algorithms for BDD construction from a Boolean expression and algorithms for Boolean operations on BDD arguments follow a recursive descent scheme according to the above Shannon expansion. It should be noted that, given a Boolean function, the size of the resulting BDD is highly dependent on the chosen variable ordering.

2.3 Symbolic representation of LTS

We first define, how elements from finite sets (e.g. actions, states) are encoded as Boolean vectors.

Def: *Encoding*
An *encoding* of a finite set $S = \{s_1, \ldots, s_n\}$ is a mapping $S \to \{0,1\}^{\lceil \log_2 n \rceil}$. For $x \in S$, we write $enc(x) = (b_1, \ldots, b_{\lceil \log_2 n \rceil})$, i.e. $enc(x)$ is a Boolean vector of length $\lceil \log_2 n \rceil$. Function Enc denotes the encoding of the whole of a transition of an LTS, i.e. $Enc(s \xrightarrow{a} t) = (enc(a), enc(s), enc(t))$. ■

The next definition states in which way an LTS is represented by a BDD.

Def: *Symbolic Representation of a LTS by a BDD*
Let $\mathcal{T} = (S, L, \to, s_1)$ be a LTS. Let $\mathcal{B} = (Nodes, var, low, high)$ be a BDD with $Vars = \{a_1, \ldots, a_{n_a}, s_1, \ldots, s_{n_s}, t_1, \ldots, t_{n_s}\}$ and root node r. We say that $\mathcal{B}$ is the symbolic representation of $\mathcal{T}$ iff

$$s \xrightarrow{a} t \Leftrightarrow \left(Bool\left(r \mid (a_1,\ldots,a_{n_a},s_1,\ldots,s_{n_s},t_1,\ldots,t_{n_s}) = Enc(s \xrightarrow{a} t)\right) = 1\right)$$ ■

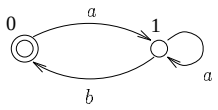
Naturally, we wish to consider "good" variable orderings to achieve "small" BDDs. Experience has shown that the resulting BDD is small if the ordering of Boolean variables is chosen according to the following heuristics [10]:

$$a_1 < \ldots < a_{n_a} < s_1 < t_1 < s_2 < t_2 < \ldots < s_{n_s} < t_{n_s}$$

i.e. the variables encoding the action come first, followed by the variables for source and target state interleaved. In particular, this ordering is advantageous in view of the parallel composition operator discussed below (see Sec. 4). Fig. 3 shows a simple LTS, the way transitions are encoded and the corresponding BDD.

The algorithm for constructing the BDD representation from a given LTS works as follows: Transitions from the LTS are processed one by one, each transition being encoded in a simple BDD which is subsequently combined by a Boolean "or" operation with the BDD representing all the previously processed transitions. The algorithm can be sketched like this:

(1) $\mathcal{B} := 0$
(2) for each transition $x \xrightarrow{a} y$ of the LTS
(3) $\quad Newtrans :=$
$\quad \left((a_1,\ldots,a_{n_a},s_1,\ldots,s_{n_s},t_1,\ldots,t_{n_s}) = Enc(x \xrightarrow{a} y)\right)$
(4) $\quad \mathcal{B} := \mathcal{B} \vee Newtrans$



	(a,s,t)
$0 \xrightarrow{a} 1$	$(0,0,1)$
$1 \xrightarrow{a} 1$	$(0,1,1)$
$1 \xrightarrow{b} 0$	$(1,1,0)$

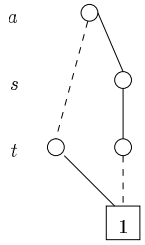


Figure 3: LTS, transition encoding and corresponding BDD

On line (1), the BDD to be constructed, $\mathcal{B}$, is initialised as 0. i.e. it does not represent any transition. On line (3), one transition of the SLTS is encoded in BDD *Newtrans* (which consists of a single path from the root to the terminal node 1, encoding action label a and source and target states x and y). On the last line, the "or" between the previous result and the new transition is computed.

3 Representing LTSs with additional rate information

3.1 Stochastic LTSs

In case of *stochastic* transition systems, each transition has as a second attribute a positive real number, the *rate* of the transition, i.e. edges are labelled with tuples from $L \times \mathbb{R}$.

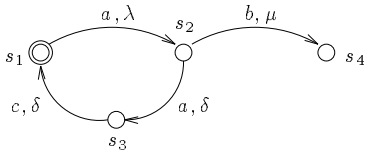


Figure 4: Example of a stochastic labelled transition system (SLTS)

Def: *Stochastic Labelled Transition System (SLTS)* Let S, s_1, and L be defined as for LTSs. Let $\rightarrow$ be defined as follows:

$$\rightarrow \subseteq S \times L \times \mathbb{R}^{>0} \times S$$

We call $\mathcal{T} = (S, L, \rightarrow, s_1)$ a *stochastic Labelled Transition System*. If $(x, a, \lambda, y) \in \rightarrow$, we say that there is an a-transition from state x to state y with rate λ and write $x \stackrel{a,\lambda}{\rightarrow} y$. ■

For practical reasons and in view of the following symbolic representation, we merge multiple a-transitions between a given pair of states into a single transition. For instance, two separate tansitions $x \stackrel{a,\lambda}{\rightarrow} y$ and $x \stackrel{a,\mu}{\rightarrow} y$ will be merged into $x \stackrel{a,\lambda+\mu}{\rightarrow} y$.

The real-valued rates specify the time spent in a particular state, which is a random value drawn from an exponential distribution. The mean of this distribution is given by the inverse of the sum of all rates of transitions leaving that state. For example, in Fig. 4, the mean time spent in state s_1 is $1/\lambda$, and the mean time spent in state s_2 is $1/(\mu + \delta)$. The Continuous Time Markov Chain (CTMC) corresponding to a SLTS is obtained by abstracting from the action labels.

3.2 Decision Node BDDs

In the previous section, we explained the basic idea of representing LTSs symbolically with the help of BDDs. Clearly, pure BDDs do not offer any mechanism for representing the information about transition rates. In the literature, Multi-terminal BDDs (MTBDD) [9] (also called Algebraic DDs (ADD) [13]), Edge-valued BDDs (EVBDD) [22] and Binary Moment Diagrams (BMD) [6] have been proposed as data structures for representing functions of the type $\{0,1\}^n \rightarrow \mathbb{R}$. However, with all of these approaches there is less sharing of subgraphs and therefore the efficiency of the representation is diminished compared to the original BDD. Therefore, in the present work, it was our aim to use the unmodified BDD (which represents the functional information of the LTS) and decorate it with the additional rate information.

The basic question for which we have to find an answer is: If we map each Boolean assignment $(b_1, \ldots, b_n)$ to a real number, how can this in-

formation be incorporated into the BDD without changing the basic BDD-structure? In other words, BDDs should be extended in order to represent functions of the type $f : \{0,1\}^n \rightarrow \{0,1\} \times I\!R$. This section explains our concept to achieve this goal. We start with the definition of path:

Def: *Path*
A *path* through a BDD (over n Boolean variables) is a vector of nodes $(x_1, \ldots, x_k)$, $1 \leq k \leq n+1$, where $x_i \in Nodes$, x_1 is the BDD root node and $x_k \in T$ (x_k is a terminal node) and $\forall i : x_{i+1} = low(x_i) \vee x_{i+1} = high(x_i)$. A path is called a *true-path* iff $x_k = 1$, otherwise it is called a *false-path*. We denote the set of all (true-) paths through a BDD by $Paths$ ($True\text{-}Paths$). For a given Boolean assignment $(b_1, \ldots, b_n) \in \{0,1\}^n$, the function $path(b_1, \ldots, b_n) = (x_1, \ldots, x_k)$ returns the corresponding path through the BDD. We define the *length* of a path by $length(x_1, \ldots, x_k) = k$. ■

If a given path $(x_1, \ldots, x_k)$ has length $k = n+1$, which is the maximal possible length for a path, that path contains a node for every Boolean variable, formally $\forall\ 1 \leq i \leq n : var(x_i) = v_i$. This means that the path corresponds to exactly one Boolean assignment $(b_1, \ldots, b_n)$. In this case, we say that the path does not contain any don't cares. If a path is of length $k < n+1$, it contains $n+1-k = d$ don't cares. Such a path corresponds to 2^d different Boolean assignments (because for every don't care two Boolean values are possible). Every Boolean assignment is mapped onto exactly one path. Several Boolean assignments (always a power of 2) may be mapped onto the same path, in which case the path has one or more don't cares. Therefore we assign to each true-path a vector of real numbers (rates), also called a rate list, whose length (a power of 2) is determined by the number of don't cares of the path. Formally, we introduce the function $rates(x_1, \ldots, x_k) = (r_1, \ldots, r_{2^{n+1-k}})$. Thus, the correspondence of Boolean assignments to rates is one to one, uniquely defined by the lexical ordering of the Boolean assigments. We illustrate this concept in Fig. 5.

So far, we decided that every true-path is mapped onto a real-valued vector whose dimension is given by the number of Boolean assignments corresponding to the path. Next we must find a practical method for attaching that information to the BDD. What are the characteristics of a path? A subset of the BDD nodes, the so-called *Decision Nodes* play a key role in this consideration.

Def: *Decision Node*
A non-terminal BDD-node $x \in NT$ is called *decision node* iff $low(x) \neq 0 \wedge high(x) \neq 0$, i.e. iff the terminal node 1 can be reached via both outgoing edges of node x. The set of decision nodes is denoted DN. ■

Let $(x_1, \ldots, x_k) \in True\text{-}Paths$. Let x_j be the "last" decision node on that path, i.e. $x_j \in DN \wedge \forall\ j < l \leq k : x_l \notin DN$. We then attach the rate list $rates(x_1, \ldots, x_k) = (r_1, \ldots, r_{2^{n+1-k}})$ to the edge (x_j, x_{j+1}). We can now give the central definition for our new data structure:

Def: *Decision Node BDD (DNBDD)*
A *Decision Node BDD* (DNBDD) is a BDD extended by a function

$$rates : True\text{-}Paths \rightarrow (I\!R)^+$$

(where $(I\!R)^+$ is the set of finite lists over $I\!R$), such that for any true-path p, $rates(p) \in I\!R^{2^d}$ if d is the number of don't cares on path p. The list $rates(p)$ is attached to the outgoing edge of the last decision node on path p, i.e. the decision node nearest to the terminal node 1. ■

Note that in this definition, the length of the vector $rates(p)$ is not a global constant but depends on the number of don't cares d (i.e. on the length k) of the individual true-path p. It can be shown that DNBDDs are a canonical representation for SLTSs. The concept of DNBDDs is illustrated in Fig. 6 (in the figure, decision nodes are drawn black). In this example, the SLTS has four transitions, i.e. there are four Boolean assignments evaluating to 1, each of which is mapped onto a rate (as shown in the middle part of the figure). The first two assignments are mapped onto the same path, 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 true-paths share their last decision node. This is the case if and only if there exists a decision node which can be reached from the root by more than one path. As an example, see Fig. 7 (left), where a decision node has two incoming edges. In such a case, several rate lists will be attached to the same edge. From the point

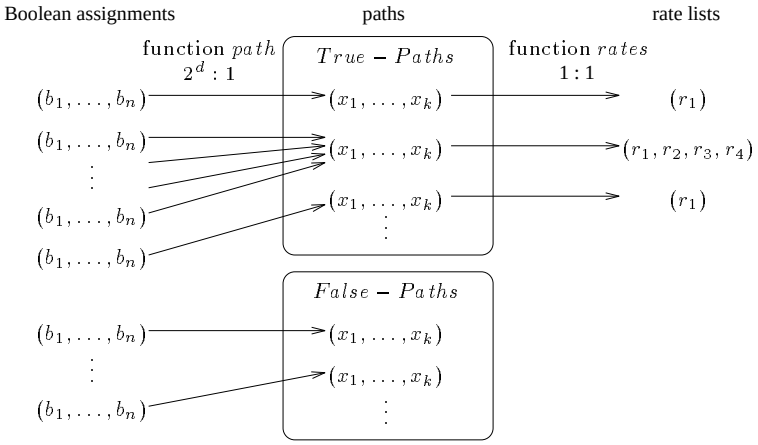


Figure 5: Correspondence between Boolean assignments, paths and rate lists

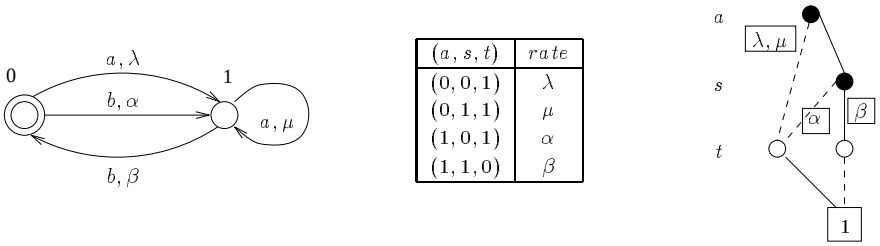


Figure 6: SLTS, mapping of Boolean assignments to rates and corresponding DNBDD

of view of canonicity this would not be a problem, since the one-to-one correspondence between true-paths and rate lists could be preserved based on lexicographical ordering. However, in the algorithms for manipulating the data structure this would result in a confusion, since during recursive descent it would not be clear any more which rate list corresponds to which path. In order to overcome this problem, we introduce a pointer structure, the so-called *rate tree*, as illustrated in Fig. 7 (right). The rate tree is a binary tree whose terminal nodes contain the rate lists. Its nodes are associated with the decision nodes of the BDD as indicated in the figure (right). In our current implementation, the rate tree is built as a separate data structure from the BDD, but it is manipulated simultaneously with the BDD, i.e. the rate tree is traversed (and possibly modified) in a recursive fashion by an appropriate extension of the procedures which manipulate the BDD data structure.

In addition to the function $Bool$, which can remain unchanged as defined before, we now define a function Num, which, given a DNBDD and a Boolean

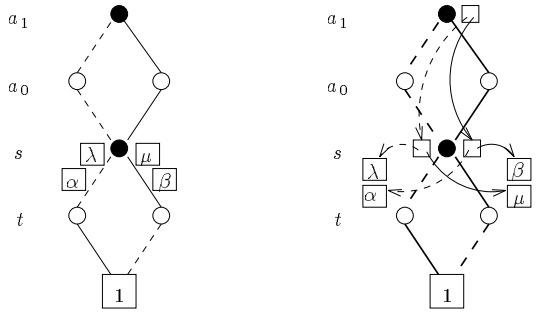


Figure 7: Two true-paths sharing their last decision node, DNBDD with rate tree

assignment, computes the numerical result.

Def: *Numeric result value* $Num(r \mid b_1, \ldots, b_n)$
Let r be the DNBDD root node and let $(b_1, \ldots, b_n)$ be a fixed assignment to the Boolean variables $v_1, \ldots, v_n$. If $Bool(r \mid v_1 = b_1, \ldots, v_n = b_n) = 0$ then the function $Num(r \mid b_1, \ldots, b_n)$ is undefined. Else let $(x_1, \ldots, x_k) = path(b_1, \ldots, b_n)$ and $rates(x_1, \ldots, x_k) = (r_1, \ldots, r_{2^{n+1-k}})$. Then $Num(r \mid b_1, \ldots, b_n) = r_i$, where i is determined unambiguously by those positions of $(b_1, \ldots, b_n)$ which

correspond to don't cares. In other words, each of the 2^{n+1-k} Boolean assignments sharing path $(x_1, \ldots, x_k)$ corresponds to exactly one element of the rate list $(r_1, \ldots, r_{2^{n+1-k}})$, and this correspondence is according to the lexicographical ordering of the Boolean assignments. ■

We are now able to define how to represent a SLTS by a DNBDD:

Def: *Symbolic Representation of a SLTS by a DNBDD*
Let $\mathcal{T} = (S, L, \rightarrow, s_1)$ be a SLTS. Let $\mathcal{B} = (Nodes, var, low, high, rates)$ be a DNBDD with $Vars = \{a_1, \ldots, a_{n_a}, s_1, \ldots, s_{n_s}, t_1, \ldots, t_{n_s}\}$ and root node r. We say that $\mathcal{B}$ is a symbolic representation of $\mathcal{T}$ iff

$$x \stackrel{a,\lambda}{\rightarrow} y$$
$$\Leftrightarrow$$
$$Bool\Big(r \,|\, (a_1,..,a_{n_a},s_1,..,s_{n_s},t_1,..,t_{n_s}) = Enc(s \stackrel{a}{\rightarrow} t)\Big) = 1$$
$$\wedge$$
$$Num\Big(r \,|\, (a_1,..,a_{n_a},s_1,..,s_{n_s},t_1,..,t_{n_s}) = Enc(s \stackrel{a}{\rightarrow} t)\Big) = \lambda$$

■

Fig. 8 shows two example SLTSs and their DNBDD representation. The first example is the same as the one given in Fig. 3, augmented by the rate information. In the second example, there are four different actions which are encoded in two bits (a_1 and a_0). In this example, the BDD contains five true-paths, two of which have a don't care in the Boolean variable a_0. Therefore, two rate lists have length two. The other three true-paths do not contain any don't cares, therefore the remaining three rate lists have length one.

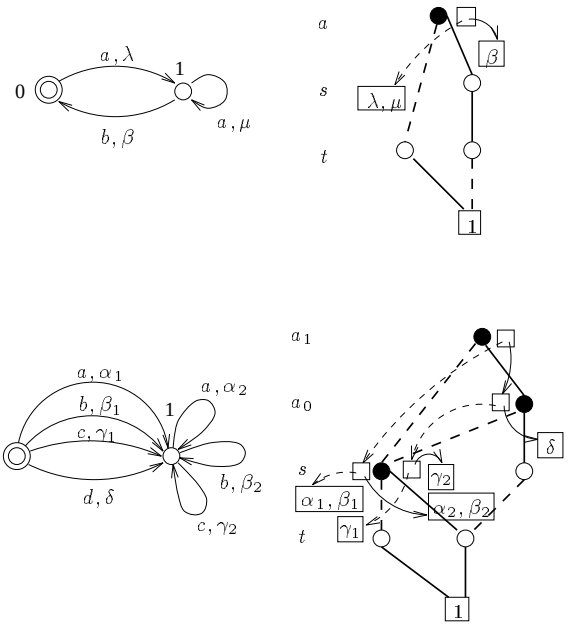


Figure 8: SLTSs and corresponding DNBDDs

4 Operations on DNBDD

4.1 Generation and parallel composition

The method of generation of a DNBDD from a given STLS is basically the same as explained earlier for the purely functional case (see Sec. 2.3), i.e. transitions are processed one by one. Each transition is first translated into a DNBDD which is then combined by an or-operation with the previously obtained intermediate result (the or-operation now also takes care of manipulating the rate-tree).

For DNBDDs representing LTSs which originate from stochastic process algebras or other modular specification techniques, an important operation is parallel composition. The parallel composition operator can be realised directly on the DNBDD representation of the two operand processes. Suppose we wish to perform the parallel composition of two processes, $P = A \parallel_S B$, where S denotes the set of synchronising actions, i.e. those actions which both partners perform simultaneously together (actions not in S are performed by one of the two partners alone, independently of the other partner). We assume that the DNBDDs which correspond to processes A and B have already been generated and are denoted $\mathcal{A}$ and $\mathcal{B}$. The set S can also be coded as a BDD, namely $\mathcal{S}$ (note that $\mathcal{S}$ is a BDD and not a DNBDD, since it does not contain any rate information). The DNBDD $\mathcal{P}$ which corresponds to the resulting process P can then be written as a Boolean expression:

$$\begin{aligned} \mathcal{P} \;=\; & (\mathcal{A} \wedge \mathcal{S}) \wedge (\mathcal{B} \wedge \mathcal{S}) \\ \vee\; & (\mathcal{A} \wedge \overline{\mathcal{S}} \wedge Stab_B) \\ \vee\; & (\mathcal{B} \wedge \overline{\mathcal{S}} \wedge Stab_A) \end{aligned}$$

The term on the first line is for the synchronising 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 sta-

bility, i.e. the fact that the source state of process A (B) equals its target state.

An important question is about the result rate of synchronising actions. Depending on the application, different expressions for the result rate may apply. Typical examples are the maximum, minimum, sum or product of the two partner rates (in [19] the concept "apparent rate" is introduced for this purpose). Choosing the product of the two partner rates has the advantage that important congruence properties can be established [17]. Using DNBDDs, the result rate is calculated from the two partner rates during the and-operation at the center of the first line of the above equation. This and-operation is flexible enough to realise any of the above alternatives (maximum, minimum, ...), i.e. DNBDDs cover any of those cases.

The result, $\mathcal{P}$, describes all transitions which are possible in the product space of the two processes, i.e. originating in any pair of states of A and B. However, given a pair of initial states for A and B, only part of the product space may be reachable due to synchronisation conditions. In this situation, reachability analysis is an important tool for reducing the size of the underlying SLTS. Symbolic reachability analysis can be performed on the DNBDD representation of the resulting process. The reachability algorithm computes a BDD which represents all states reachable from the initial state (which is the combination of initial states of A and B). At every step of the algorithm, states reachable by a single transition from states previously found are added to this BDD. This is repeated until a fixed point is reached. Finally, the DNBDD $\mathcal{P}$ representing the overall SLTS is restricted to those transitions which originate in reachable states.

4.2 Symbolic minimisation of SLTS, working on DNBDD

This subsection describes how 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. It is shown how such a minimisation technique can be applied to the DNBDD representation of the SLTS, i.e. the minimisation is entirely based on DNBDD operations. Symbolic minimisation based on BDDs for the purely functional case has been described before, see e.g. [2]. Here we describe BDD-based minimisation of *stochastic* LTSs, i.e. BDD-based minimisation which takes into account the stochastic rate information (for more details see [18]). The advantages for performance analysis are obvious: The SLTS of a complex system can be built from small components by applying the DNBDD-based parallel composition operator step by step. After every parallel composition step, the intermediate result can be minimised without leaving the DNBDD world. Thus, the use of DNBDDs quite ideally supports the concept of compositional reduction.

The equivalence relation on which we focus is known as Markovian bisimulation [17]. Informally, two states are Markovian 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 (defined below). There is a strong connection between Markovian bisimulation and classical Markov chain *lumpability* [21]. Informally, Markovian bisimulation is a refinement of lumpability, by distinguishing between different action names. Fig. 9 illustrates how a state space S is partitioned into three disjoint subsets, $C_1 \ldots C_3$, also called classes.

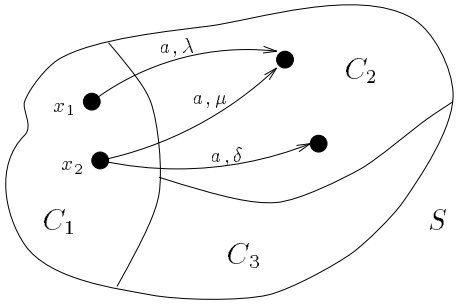


Figure 9: Partitioning of state space S

Def: *Cumulative Rate*
Let $C_1, \ldots, C_n$ be a partition of the state space S of a SLTS. Let $x \in S$. The *cumulative rate* of action a from state x to class C_i is defined as

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

■

For example, in Fig. 9, $\Lambda(x_1, a, C_2) = \lambda$ and $\Lambda(x_2, a, C_2) = \mu + \delta$.

When using DNBDDs, the cumulative rate of action a_k from state x to class C_i can be easily computed in the following way. Let $T(a, s, t)$ be the DNBDD

representing the SLTS (a, s and t are vectors of Boolean variables). For convenience, T is usually broken up into individual DNBDDs $T_{a_k}(s,t)$, one for every action a_k:

$$T_{a_k}(s,t) = (T(a,s,t) \wedge (a = enc(a_k)))$$

In order to obtain a DNBDD which represents all transitions from state x to states from class C_i we restrict $T_{a_k}(s,t)$ to the single source state x and to target states from class C_i (class C_i is represented by a BDD $C_i(t)$):

$$T_{x \xrightarrow{a_k} C_i}(s,t) = (T_{a_k}(s,t) \wedge (s = enc(x)) \wedge C_i(t))$$

The cumulative rate is then computed by applying the function *soar* (sum of all rates) to $T_{x \xrightarrow{a_k} C_i}(s,t)$. This function simply sums up all the entries in the rate tree of a DNBDD (for example, applied to the DNBDD shown in Fig 8 (bottom) the function *soar* would yield the value $\alpha_1+\alpha_2+\beta_1+\beta_2+\gamma_1+\gamma_2+\delta$).

We can now give the formal definition of Markovian bisimulation.

Def: *Markovian Bisimulation*
Let $C_1, \ldots, C_n$ be a partition of the state space S of a SLTS. Let $\overset{M}{\sim}$ be the equivalence relation corresponding to this partition. $\overset{M}{\sim}$ is called a *Markovian Bisimulation* iff

$$\forall x_1, x_2 \in S :$$
$$x_1 \overset{M}{\sim} x_2 \Rightarrow \forall a : \forall C_i : \Lambda(x_1, a, C_i) = \Lambda(x_2, a, C_i)$$ ■

Algorithms for Markovian bisimulation traditionally follow an iterative refinement scheme [23, 11, 20] (the TIPPtool [15], for instance, contains such an implementation). 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}. During refinement, a class C_i is split with respect to a splitter, which means that subclasses $C_{i1}, C_{i2}, \ldots, C_{ik}$ are computed ($k \geq 1$), such that the cumulative rate $\Lambda(x, a, C_{spl})$ is the same for all the states x belonging to the same subclass.

In the following, a DNBDD-based bisimulation algorithm is presented, in which the transition system is represented by DNBDDs $T_a(s,t)$, one for each action a, and in which the current partition is stored as a set of BDDs, one for each class. The algorithm uses a dynamic set of splitters, denoted *Splitters*, which in our implementation is realised as a pointer structure. Note that here we only present a basic version of the algorithm which can be optimised in many ways [7, 14, 18].

1. Initialisation
   $Partition := \{C_1\} = \{S\}$
   /* the initial partition consists of only one class which contains *all* states */
   $Splitters := Act \times Partition$
   /* all pairs of actions and classes have to be considered as splitters*/
2. Main loop
   while ($Splitters \neq \emptyset$)
   choose splitter $(a, C_{spl}) \in Splitters$
   forall $C_i \in Partition$ $split(C_i, a, C_{spl})$
   /* all classes (including C_{spl} itself) are split */
   $Splitters := Splitters - (a, C_{spl})$
   /* the processed splitter is removed from the splitter set */

It remains to specify the procedure *split*. Its task is to split a class C_i, using the combination (a, C_{spl}) as a splitter. Procedure *split* uses a data structure *split_tree* (shown in Fig. 10). The input class C_i is split into subclasses $C_{i1}, \ldots, C_{ik}$ according to the cumulative rate from a state in C_i to class C_{spl} (regarding transitions labelled with action a). The subclasses $C_{i1}, \ldots, C_{ik}$ are represented by BDDs.

```
procedure split(C_i, a, C_spl)
  forall s_x ∈ C_i
    Λ_{s_x} = soar(T_a(s,t) ∧ (s = enc(s_x)) ∧ C_spl(t))
    /* the cumulative rate from state s_x to class
       C_spl is computed */
    insert(split_tree, s_x, Λ_{s_x})
    /* state s_x is inserted into the split_tree which,
       finally, has k branches */
  if (k > 1)   /* only continue if C_i has been split
                  into k > 1 subclasses */
    Partition := Partition ∪ {C_i1, C_i2, ..., C_ik} − {C_i}
    Splitters := Splitters ∪ (Act × {C_i1, C_i2, ..., C_ik})
                           − (Act × {C_i})
    /* the partition and the splitter set
       are updated */
```

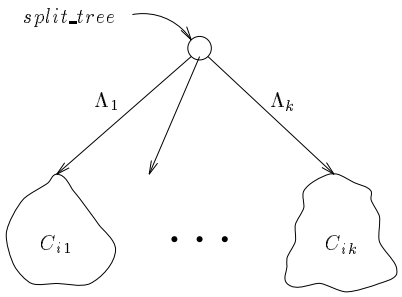


Figure 10: The *split_tree* used by procedure *split*

In the forall loop of procedure *split*, the cumulative rate Λ_{s_x} is computed for every state s_x in class C_i, and state s_x is inserted (by procedure *insert*) into the *split_tree* such that states with the same cumulative rate fall into the same branch. The *split_tree* finally has k leaves, i.e. k different values of Λ_{s_x} have appeared. If splitting has taken place (i.e. if $k > 1$), the partition must be refined and the set of splitters must be updated.

5 Tool and experimental results

The concept of DNBDDs which we described in this paper has been fully realised in an experimental tool in order to demonstrate the feasibility of our approach. The tool is still a prototype which is not optimised in terms of memory requirements and efficiency. Therefore, so far, the tool is not capable of handling very large state spaces. The tool is written in C and up to now only offers a rudimentary textual user interface. Its main capabilities are:

- generation of a DNBDD from a given SLTS. Currently, the tool reads SLTS files generated by the TIPPtool [15], a tool for the specification and analysis of Stochastic Process Algebra models.
- parallel composition of two processes whose behaviour had been encoded into DNBDD form in previous steps. Afterwards, if the user desires, reachability analysis can be performed in order to restrict the potential behaviour of the resulting process to the states which are actually reachable from a given initial state.

  As an example, Fig. 11 (left) shows the block diagram of a queueing model, consisting of an arrival process, a scheduler (which assigns incoming jobs to queues) and two queue-server pairs. The model was specified as the parallel composition of six sequential processes, using the stochastic process algebra TIPP. For this model, the operational semantics of the process algebra generates an overall SLTS with 1568 reachable states and 4760 transitions. In general, when performing parallel composition of SLTSs, the size of the resulting SLTS is exponential in the number of parallel components.

  The right hand side of Fig. 11 illustrates the use of DNBDD-based parallel composition. The SLTSs for the six sequential processes were generated by the SPA semantics and then translated into their corresponding DNBDDs. The DNBDD for the overall model was then generated by applying the DNBDD-based parallel composition algorithm in a stepwise fashion. The figure gives the number of DNBDD nodes at every composition step. The main advantage of symbolic parallel composition consists of the fact that the DNBDD size is roughly linear (!) in the number of parallel components. In this example, the DNBDD for the overall model has only 183 DNBDD nodes (which is the same number of nodes which the BDD for the purely functional system would have if it were constructed in a similar compositional way).
- minimisation on the basis of Markovian bisimulation. We implemented a version of the bisimulation algorithm which uses the splitter set administration technique explained earlier (see Sec. 4.2). As a result, the tool outputs the final partition of the state space and generates the DNBDD representation of the reduced transition system in which each class of equivalent states is represented by a single state.

  Applying symbolic minimisation, we observed the following interesting phenomenon: Minimisation of a SLTS may have the effect that, although the number of states is reduced, the size of the symbolic representation actually grows! This negative effect has also be studied theoretically in the context of Multi Terminal BDDs [16]. The reason for this rather counter-intuitive phenomenon is that minimisation destroys the regularity of the symbolic representation.
- all results computed by the tool in DNBDD form can be converted back into their SLTS representation in order to be further processed by the TIPPtool.

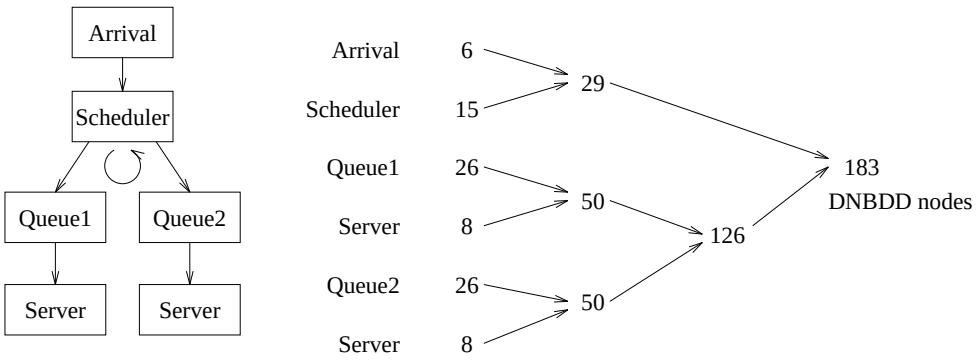


Figure 11: Example process model and corresponding number of DNBDD nodes

It is still too early to publish empirical data about the performance of our prototype tool since our implementation still needs to be improved a lot. For instance, we need to incorporate sophisticated hashing techniques (see [3]). Furthermore, a straightforward implementation of the rate tree as described in this paper means that one rate is explicitly stored for each encoded transition of the SLTS. Basically, this goes against the grain of BDDs, and experiments have shown that this may cause substantial overhead. However, it is possible to implement the binary rate tree itself as a decision diagram. We are currently working on this problem.

6 Conclusion

The problem of state space explosion remains the most prominent problem of analytical performance and dependability modelling. A lot of research has been done on how to best avoid or tolerate large state spaces, some of which led to very valuable results. Nevertheless it remains important to look out for new ideas which may improve the tractability of complex models. In this sense, symbolic techniques, in particular BDDs, are very promising, since they have been successfully used for state-space-based techniques in the area of functional analysis. In the past, the symbolic approach had not received much consideration from the performance community, which seems to make it all the more important to be looked into now.

We developed DNBDDs, a data structure which is an extension of basic BDDs, tailored to represent *stochastic* transition systems in a compact way. We were able to show that all the algorithms which are generally needed to build, manipulate and analyse SLTSs have a corresponding algorithm which works on the compact DNBDD representation. Using DNBDDs, the advantages of the symbolic approach can be enjoyed not only while working on purely functional considerations, but also during analyses which make use of the numerical information of a SLTS. Therefore the use of DNBDDs has the potential to handle more complex stochastic performance models than before.

Among future work we plan to develop DNBDD-based algorithms for the numerical analysis of the CTMC underlying a SLTS (so far, before applying numerical analysis algorithms we switch back to a sparse matrix representation). Iterative methods for the numerical analysis of CTMCs are based on vector-matrix multiplication. It is known how this operation can be performed efficiently using MTBDDs [9, 16]. In principle, vector matrix multiplication is also feasible on the basis of DNBDDs, but the details are still a topic for further research.

Furthermore, the relation of BDD-based representations to the Kronecker approach (e.g. [7]) deserves further studies. There is an obvious similarity between the two approaches: The representations of submodels are combined by an operator (Kronecker product/sum or a Boolean operation on BDDs) such that the size of the resulting representation is only linear in the number of submodels. In both approaches, this compactness is achieved by avoiding the explicit enumeration of all possible interleavings of actions in the participating submodels.

References

[1] H.R. Andersen. An Introduction to Binary Decision Diagrams. Technical report, Department

of Computer Science, Technical University of Denmark, December 1994.

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

[3] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient Implementation of a BDD Package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

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

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

[6] R.E. Bryant and Y. Chen. Verification of Arithmetic Functions with Binary Moment Diagrams. In *32nd ACM/IEEE Design Automation Conference*, pages 535–541, 1995.

[7] P. Buchholz. A Framework for the Hierarchical Analysis of Discrete Event Dynamic Systems. Habilitation thesis, Universität Dortmund, 1996.

[8] 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.

[9] 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.

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

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

[12] M. Fujita, P. McGeer, and J.C.-Y. Yang. Multi-terminal Binary Decision Diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, April/May 1997.

[13] 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.

[14] H. Hermanns. *Interactive Markov Chains*. PhD thesis, Universität Erlangen-Nürnberg, 1998.

[15] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional Performance Modelling with the TIPPtool. In R. Puigjaner, N. Savino, and B. Serra, editors, *10th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS '98)*, pages 51–62, Palma de Mallorca, September 1998. Springer LNCS 1469.

[16] H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains. Accepted for publication in 3rd International Meeting on the *Numerical Solution of Markov Chains* 1999.

[17] H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 71–88, Erlangen-Regensberg, July 1994. IMMD, Universität Erlangen-Nürnberg.

[18] H. Hermanns and M. Siegle. Bisimulation Algorithms for Stochastic Process Algebras and their BDD-based Implementation. In J.-P. Katoen, editor, *5th Int. AMAST Workshop on Real-Time and Probabilistic Systems*, pages 244–264, Springer LNCS 1601, 1999.

[19] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

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

[21] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer, 1976.

[22] 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.

[23] R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[24] M. Siegle. BDD extensions for stochastic transition systems. In D. Kouvatsos, editor, *Proc. of 13th UK Performance Evaluation Workshop*, pages 9/1 – 9/7, Ilkley/West Yorkshire, July 1997.

[25] M. Siegle. Compact representation of large performability models based on extended BDDs. In *Fourth Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS4)*, pages 77–80, Williamsburg, VA, September 1998.