Correctness Issues of Symbolic Bisimulation Computation for Markov Chains*

Ralf Wimmer and Bernd Becker

Chair of Computer Architecture
Albert-Ludwigs-University Freiburg im Breisgau, Germany
{wimmer,becker}@informatik.uni-freiburg.de

Abstract. Bisimulation reduction is a classical means to fight the infamous state space explosion problem, which limits the applicability of automated methods for verification like model checking. A signature-based method, originally developed by Blom and Orzan for labeled transition systems and adapted for Markov chains by Derisavi, has proved to be very efficient. It is possible to implement it symbolically using binary decision diagrams such that it is able to handle very large state spaces efficiently. We will show, however, that for Markov chains this algorithm suffers from numerical instabilities, which often result in too large quotient systems. We will present and experimentally evaluate two different approaches to avoid these problems: first the usage of rational arithmetic, and second an approach not only to represent the system structure but also the transition rates symbolically. In addition, this allows us to modify their actual values *after* the quotient computation.

1 Introduction

The state space explosion problem denotes the observation that the state space of a system grows exponentially in the number of components the system consists of. The size of realistic systems limits the applicability of formal verification techniques to large designs. To alleviate this effect, numerous techniques have been developed like the usage of symbolic methods (e. g. decision diagrams in various flavors, see [1]) and abstraction techniques (e. g. partial-order reduction and symmetry reduction). Bisimulation minimization can be considered as a kind of abstraction technique which can be performed fully automatically. The idea behind bisimulation minimization is to group the states into equivalence classes such that states are considered equivalent if and only if they exhibit the same step-wise behavior. A system with a minimal number of states which has the same behavior as the original system—this means that it satisfies the same

* This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See www.avacs.org for more information.

formulas of a temporal logic like CTL or CSL—can be obtained by replacing each equivalence class of the bisimulation with a single state.

While Fisler and Vardi [2] observed that bisimulation minimization does not speed up checking invariant properties of labeled transitions systems, the contrary is often the case for stochastic systems like discrete- and continuous-time Markov chains. There, model checking involves the solution of a linear equation system and is much more expensive than in the purely digital case. Hence, model checking can benefit a lot from minimizing the model prior to checking properties [3, 4].

This has led to a revival of bisimulation techniques for stochastic systems in the last few years (cf. for example [5–9]). One of the most efficient approaches is based on signature-based partition refinement, originally developed by Blom and Orzan [10] for labeled transition systems (LTSs). Wimmer et al. [11] extended this approach such that a large number of different kinds of bisimulations for LTSs can be computed symbolically using OBDDs. Derisavi [7] applied it successfully to continuous-time Markov chains. The problem with this signature-based approach for Markov chains is, as we will show, that it is very sensitive to numerical problems. In many cases they lead to quotient systems with too many states, and sometimes they can even prevent termination. In this paper we will address these problems: by using rational arithmetic to avoid numerical problems. Although considered to be computationally expensive typically, we will show that this is not the case for our bisimulation algorithm. Another possibility, which we will present, is to handle the transition rates not as numbers but as pure symbols. Besides avoiding numerical problems this has the advantage that the rates can be adjusted *after* the quotient computation without redoing the minimization. We will also present experimental results for this technique.

We have structured this paper as follows: in the next section we review the foundations which consist of continuous-time Markov chains, bisimulations, and the principle of signature-based bisimulation computation. In Section 3 we present methods which yield bisimulation relations in a reliable and/or parametric manner. Section 4 provides an experimental evaluation of these methods. Finally, in Section 5, we conclude and point out directions for future research.

2 Foundations

In this section we will briefly review the basics of continuous-time Markov chains (CTMCs), bisimulations and the signature-based algorithm for computing bisimulations on CTMCs symbolically.

Definition 1. *Let AP be a finite set of atomic propositions. A* continuous-time Markov chain *(CTMC) is a tuple $M = (S, s_0, R, L)$ such that S is a finite, non-empty set of states; $s_0 \in S$, the initial state; $R : S \times S \to \mathbb{R}^{\geq 0}$, the matrix of transition rates; and $L : S \to 2^{AP}$, a labeling function which assigns each state a set of atomic propositions from AP.*

For a set $S' \subseteq S$ of states, we use the notation $R(s, S) = \sum_{s' \in S'} R(s, s')$. The transitions of a CTMC are governed by a negative exponential distribution, i. e. the probability to take the transition from s to s' within time t is given by

$$p(s, s', t) = \frac{R(s, s')}{R(s, S)} \cdot \left(1 - e^{-R(s,S) \cdot t}\right). \tag{1}$$

A *partition* of a set S is a set $P \subseteq 2^S \setminus \{\emptyset\}$ such that the union of all elements (called blocks) of P equals S and all blocks of P are pairwise disjoint. If P is a partition of S, we write for states $s, t \in S$: $s \equiv_P t$ iff there is a block $B \in P$ such that $\{s, t\} \subseteq B$. A partition P is a *refinement* of a partition P' (denoted $P \sqsubseteq P'$) if $\forall B \in P\ \exists B' \in P' : B \subseteq B'$.

Definition 2. *Let $M = (S, s_0, R, L)$ be a continuous-time Markov-Chain. A partition P of S is a* bisimulation *on M if for all $s, t \in S$ with $s \equiv_P t$ and for all blocks $B \in P$ the following conditions hold:*

$$L(s) = L(t) \qquad \textit{and} \qquad R(s, B) = R(t, B).$$

States $s, t \in S$ are bisimilar (written $s \approx t$) if there is a bisimulation P on M such that $s \equiv_P t$.

The practically most important property of bisimilarity is that states are bisimilar if and only if they satisfy the same formulas of the temporal logic CSL [12], which is widely used for specification of requirements. This enables us to use the quotient system for checking the validity of these formulas instead of the larger original system.

The idea behind signature-based bisimulation computation is to compute for each state a kind of fingerprint, such that states can only be bisimilar if their fingerprints are identical. To obtain a refined partition, the blocks are grouped according to the signatures of their states. This is formally captured in the following definition:

Definition 3. *Let $M = (S, s_0, R, L)$ be a CTMC, $P^{(0)}$ an initial partition of S, and P a partition of S with $P \sqsubseteq P^{(0)}$. The* signature *of a state s with respect to P is then given by*

$$\mathrm{sig}(s, P) = \left\{(B, r) \in P \times \mathbb{R}^{\geq 0} \,\middle|\, r = R(s, B)\right\}.$$

The refinement $\mathrm{sigref}(P)$ *of P is defined as*

$$\mathrm{sigref}(P) = \left\{\left\{t \in S \,\middle|\, \mathrm{sig}(s, P) = \mathrm{sig}(t, P) \wedge s \equiv_{P^{(0)}} t\right\} \,\middle|\, s \in S\right\}.$$

The initial partition is needed if one wants the bisimulation quotient to preserve the validity of a certain logic like CSL or if additional information like state rewards has to be taken into account. If we have to preserve CSL properties, we set $P^{(0)} = \left\{\left\{s \in S \,|\, L(s) = L(t)\right\} \,\middle|\, t \in S\right\}$; if state rewards $r : S \to \mathbb{R}^{\geq 0}$ have to be considered, we use $P^{(0)} = \left\{\left\{s \in S \,|\, r(s) = r(t)\right\} \,\middle|\, t \in S\right\}$; otherwise,

Algorithm 2.1: SigRefine(CTMC M, partition $P^{(0)}$)
begin
    $i \leftarrow 0$ (1)
    repeat (2)
        $P^{(i+1)} \leftarrow \text{sigref}(P^{(i)})$ (3)
        $i \leftarrow i + 1$ (4)
    until $P^{(i)} = P^{(i-1)}$ (5)
    return P (6)
end

when there are no such requirements, we can use the trivial partition $P^{(0)} = \{S\}$, which consists of only one block containing all states.

We iteratively apply the sigref-operator until a fixed point is reached. The pseudo-code of this procedure is given in Algorithm 2.1.

Derisavi has shown in [7] that this algorithm terminates after at most $|S| - |P^{(0)}| + 1$ iterations and yields the coarsest bisimulation on M that refines $P^{(0)}$.

Now we will show how this algorithm can be implemented using (MT)BDDs such that their ability to represent large state spaces in a compact way is exploited.

Symbolic Implementation Ordered binary decision diagrams (OBDDs) [13] are a data structure which represents boolean functions $f : \{0,1\}^n \to \{0,1\}$ as a rooted acyclic digraph. They can be considered as a compressed form of the truth table of the represented function. Each assignment of the input variables corresponds to a path in the OBDD which ends at a leaf that is labeled with the value of the function under that assignment.

Multi-terminal BDDs are an extension of OBDDs for pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$. The only difference to OBDDs is that leaves may be labeled with arbitrary numbers (instead of being restricted to $\{0,1\}$). Both, OBDDs and MTBDDs, are called reduced if the sub-function represented at each node, is unique.

For a BDD $\mathcal{B}(\boldsymbol{x}, \boldsymbol{y})$ over the vectors $\boldsymbol{x} = (x_{n-1}, \ldots, x_0)$ and $\boldsymbol{y} = (y_{m-1}, \ldots, y_0)$ of boolean variables and a bitvector $v \in \{0,1\}^n$, the expression $\mathcal{B}(\boldsymbol{x} \leftarrow v, \boldsymbol{y})$ denotes the cofactor of $\mathcal{B}(\boldsymbol{x}, \boldsymbol{y})$ which results from fixing the variables x_i to the values given by v.

We assume that the reader is familiar with OBDDs and MTBDDs. For more information we refer the reader to Wegener's monograph [1].

OBDDs can be used to represent sets $S \subseteq \{0,1\}^n$ of bit vectors via their characteristic function χ_S such that $s \in S \Leftrightarrow \chi_S(s) = 1$. This will be used to represent the state space of the CTMC under consideration and for partitions of its state space. Several possibilities for partition representation have been proposed in the literature (cf. [9]). For the implementation of signature refinement the following technique is suited best, since it allows to execute the refinement step in linear time in the size of the signature-MTBDD [9]: each block of the partition is assigned

a unique number, which is encoded using a set of $m \geq \lceil \log_2 |P| \rceil$ new OBDD variables $\boldsymbol{k} = (k_{m-1}, \ldots, k_0)$. The partition P of S is then represented by an OBDD $\mathcal{P}(\boldsymbol{s}, \boldsymbol{k})$ such that, for a block B_i of P, $s \in B_i$ iff $\mathcal{P}(\boldsymbol{s} \leftarrow \langle s \rangle, \boldsymbol{k} \leftarrow \langle i \rangle) = 1$. Here, $\langle i \rangle$ denotes the binary encoding of i.

We will use MTBDDs to represent the transition rate matrix of the CTMC under consideration and signatures of its states. For the signatures we use an MTBDD $\sigma(\boldsymbol{s}, \boldsymbol{k})$ such that $\sigma(\boldsymbol{s} \leftarrow \langle s \rangle, \boldsymbol{k} \leftarrow \langle i \rangle) = r$ iff $(r, B_i) \in \mathrm{sig}(s, P)$. Given the transition rate matrix $\mathcal{R}(\boldsymbol{s}, \boldsymbol{t})$ and the representation $\mathcal{P}(\boldsymbol{s}, \boldsymbol{k})$ of the current partition, the MTBDD representing the signatures of all states can be computed by

$$\sigma(\boldsymbol{s}, \boldsymbol{k}) = \mathcal{Q}^{+}_{\boldsymbol{t}}.\big(\mathcal{R}(\boldsymbol{s}, \boldsymbol{t}) \cdot \mathcal{P}(\boldsymbol{t}, \boldsymbol{k})\big). \tag{2}$$

Thereby $\mathcal{Q}^{\circ}_{\boldsymbol{a}} : \big(\mathcal{B}(\boldsymbol{x}, \boldsymbol{a})\big) = \bigcirc_{i=0}^{2^l-1} \mathcal{B}(\boldsymbol{x}, \boldsymbol{a} \leftarrow \langle i \rangle)$ is the quantification operator for an associative and commutative binary operator $\circ$. It is a standard operation for MTBDDs.

In order to ensure that the initial partition is taken into account, we modify the signatures in Equation (2) such that two states can only have identical signatures if they are contained in the same block of the initial partition. Let p be a new BDD variable. We modify the signatures as follows:

$$\sigma'(\boldsymbol{s}, \boldsymbol{k}, p) = \sigma(s, k) + p \cdot \mathcal{P}^{(0)}(s, k). \tag{3}$$

Once we have computed the signatures, we have to get the refined partition. For this, we can exploit the following observation: if we use a variable order for the BDDs such that the state variables precede the block number variables (and the auxiliary variable p), the encoding of a state s corresponds to a path in $\sigma'(\boldsymbol{s}, \boldsymbol{k}, p)$ which ends in the node that represents the signature of s. Furthermore, since we only use reduced BDDs, the encodings of all states with the same signature as s lead to the same node. To obtain the refined partition, we have to replace all nodes that represent signatures by new block numbers. This can be done in linear time by traversing $\sigma'(\boldsymbol{s}, \boldsymbol{k}, p)$ recursively. More details on the symbolic implementation can be found in [11, 7].

3 Reliable Bisimulation Computation

Almost all of today's personal computers use floating-point arithmetic according to the IEEE standard 754 [14] for numerical computations. The problem of all fixed-length representations is that the result of arithmetical operations is often not representable but has to be rounded to the nearest representable number.

Example 1. We compute $0.5 = 3 \cdot 0.1 + 4 \cdot 0.05$ by adding three times 0.1 and four times 0.05 on a IEEE 754 compatible processor (left-associative). Depending on the order of the summands we obtain the following results (using the 64-bit representation):

- $0.05 + 0.05 + 0.05 + 0.05 + 0.1 + 0.1 + 0.1$
  Result:[1] $\texttt{0.01111111110.0}^{52}$
- $0.05 + 0.1 + 0.05 + 0.1 + 0.05 + 0.1 + 0.05$
  Result: $\texttt{0.01111111110.0}^{51}\texttt{1}$
- $0.1 + 0.1 + 0.05 + 0.05 + 0.05 + 0.1 + 0.05$
  Result: $\texttt{0.01111111101.1}^{52}$

We can observe that the result depends on the order of the operands.

For more detailed information about floating-point arithmetic and its problems we refer the reader to [15].

These rounding problems also affect arithmetical operations on MTBDDs. In our application, this is mainly the signature computation. Their effect is, in general, that too many leaves with only slightly different values are created, unnecessarily blowing up the size of MTBDD. For most applications this has no impact on the correctness, but only on the speed and memory consumption of the algorithm. To weaken this effect, most implementations apply the following strategy: if a leaf with value v is requested from the MTBDD manager, it is not only checked if a leaf with exactly the same value v already exists, but also if there is a leaf whose value is close enough to v. That means, a new leaf with value v is created if there exists no leaf with value v' such that $|v - v'| \leq \varepsilon$ for a predefined constant $\varepsilon > 0$. Otherwise the already existing leaf is returned. In general, there is no value of ε which ensures that the effect of inexact computations is compensated correctly.

The signature-based refinement algorithm, however, is very sensitive to rounding errors. If the signatures of two equivalent states differ slightly due to rounding, they are represented at two different nodes in the MTBDD of signatures. Therefore they are placed in different blocks of the refined partition.

This leads to the following effects: since equivalent states may be placed in different blocks, the resulting bisimulation is often unnecessarily fine. Rounding effects can also cause signatures of states with slightly different rates to be mapped onto the same value. Then these states are erroneously placed in the same block. The resulting partition may therefore be no correct bisimulation. As we will see in the experimental section, rounding effects can even prevent termination.

We therefore consider it important to develop techniques which avoid the problems caused by inexact computations.

3.1 Rational Arithmetic

To avoid the problems we have observed for our floating-point implementation of the bisimulation algorithm, we implemented a version which makes use of rational arithmetic. Rational numbers are represented as the quotient of two

[1] The dots separate the sign bit from the exponent and the exponent from the mantissa. $\texttt{0}^n$ means that the digit 0 is repeated n times.

arbitrarily long integer numbers. Then all operations we need (mainly the addition of numbers) can be performed precisely.

The MTBDDs used by our implementation have been modified such that rational numbers are stored in the leaves instead of floating-point values. The operations on the MTBDDs have also been adapted to cope with rational numbers. Besides this, we have set the value of ε to 0 such that the BDD package itself does not introduce errors.

3.2 Parametric Bisimulation Computation

Another idea how numerical problems can be avoided is not to work with concrete numbers for the transition rates but with pure symbols. Since there can be several transitions with the same rate from a state into a block of the current partition, we cannot use sets to collect the transition symbols leading into a given block. Instead, we have to use *multisets*.

Multisets allow an arbitrary (finite) number of copies of each element. For a multiset $\overline{S}$, we let $\mathbf{1}_{\overline{S}}(x)$ denote the multiplicity of element x in the multiset $\overline{S}$. The element-wise union $\overline{S} \uplus \overline{T}$ is the multiset such that for all elements the equation $\mathbf{1}_{\overline{S}\uplus\overline{T}}(x) = \mathbf{1}_{\overline{S}}(x) + \mathbf{1}_{\overline{T}}(x)$ holds. The set of multisets with elements from a set S is denoted by $\mathfrak{M}(S)$.

We can now define the symbolic counterpart of a CTMC:

Definition 4. *A* symbolic CTMC *is a tuple $\overline{M} = (S, s_0, V, \overline{R}, L)$ such that S is a non-empty finite set of states, $s_0 \in S$ the initial state, and $L : S \to 2^{AP}$ a labeling function. $V = \{\mu_{n-1}, \ldots, \mu_0\}$ is a finite set of symbols. $\overline{R} : S \times S \to \mathfrak{M}(V)$ is a labeling function which labels each transition with a multiset of symbols.*

If V is a set of symbols, we call a function $I : V \to \mathbb{R}^{\geq 0}$ an *interpretation* of V. For a multiset $\overline{S} \in \mathfrak{M}(V)$, we set $I(\overline{S}) = \sum_{\mu \in \overline{S}} \big(\mathbf{1}_{\overline{S}}(\mu) \cdot I(\mu)\big)$. For a symbolic CTMC $\overline{M} = (S, s_0, V, \overline{R}, L)$ an interpretation I induces an ordinary CTMC $M_I = (S, s_0, R, L)$ by setting $R(s, s') = I(\overline{R}(s, t))$.

We now modify the definition of the signature of a state as follows:

$$\overline{\mathrm{sig}}(s, P) = \big\{(\overline{r}, B) \in \mathfrak{M}(V) \times P \,|\, \overline{r} = \biguplus_{t \in B} \overline{R}(s, t)\big\}$$

The refinement operator remains unchanged with the exception that it now uses the modified signature:

$$\overline{\mathrm{sigref}}(P) = \big\{\{s \in S \,|\, \overline{\mathrm{sig}}(s, P) = \overline{\mathrm{sig}}(t, P) \wedge s \equiv_{P^{(0)}} t\} \,\big|\, t \in S\big\}.$$

If we use this operator for partition refinement, we obtain a partition which is a bisimulation for all possible interpretations of the symbols in V. We call such a bisimulation *interpretation-independent*. Our algorithm yields the coarsest interpretation-independent bisimulation which refines the initial partition $P^{(0)}$. The reason why not the coarsest bisimulation is computed is that under a certain interpretation I different multisets of symbols may lead to the same value, i. e. that there are $\overline{S}$ and $\overline{T}$ with $\overline{S} \neq \overline{T}$ and $I(\overline{S}) = I(\overline{T})$.

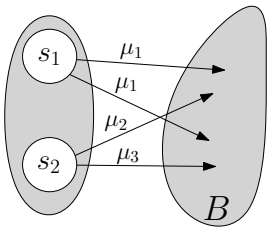

Fig. 1: Signature computation for symbolic Markov chains

If the coarsest bisimulation for a fixed interpretation is needed, it can be obtained by applying the signature refinement algorithm to the DTMC which results from applying the interpretation to the interpretation-independent quotient.

Example 2. We now consider the part of a symbolic CMTC M depicted in Figure 1 and compute the signatures of s_1 and s_2:

$$\overline{\mathrm{sig}}(s_1, P) = \big\{(\{\!|\mu_1, \mu_1|\!\}, B)\big\}$$
$$\overline{\mathrm{sig}}(s_2, P) = \big\{(\{\!|\mu_2|\!\}, B), (\{\!|\mu_3|\!\}, B)\big\}$$

Since the states do not exhibit the same signatures, the refinement operator $\overline{\mathrm{sigref}}$ puts them in different blocks.

What are the advantages of this method? We can choose the actual transition rates, i. e., the interpretation, *after* the minimization without recomputation of the quotient. This can be beneficial if during the design phase the system structure has already been fixed, but the exact rates have to be determined experimentally. Another scenario is that the analysis shows that the error rates of some components are too high to yield the required dependability. So some components have to be replaced by more robust ones. This changes the rates of the transitions. If an interpretation-independent bisimulation quotient has been obtained, only the interpretation of its symbols has to be changed. So we can save the time for computing the quotient from scratch.

The drawback of this approach is that not always the coarsest bisimulation quotient for the current interpretation of the rates can be computed. This has the effect that the quotient sometimes consists of more states than the one which would be returned by the version with rational arithmetic. Our experimental results (see Section 4), however, indicate that this effect does not occur for most of the models, and for the others the increase in size compared to the optimal quotient is not dramatic.

Example 3. Let us again consider the Markov chain M in Figure 1. We assume that the current interpretation is given by $I(\mu_1) = 3$, $I(\mu_2) = 1$, and $I(\mu_3) = 5$. If we compute the signatures of s_1 and s_2 for the ordinary Markov chain M_I, we obtain

$$\mathrm{sig}(s_1, P) = \mathrm{sig}(s_2, P) = \big\{(6, B)\big\}.$$

Since both states have the same signatures, sigref places them in the same block.

Symbolic Implementation We now show how to integrate the symbolic handling of transition rates into our BDD-based algorithm. There are mainly two possibilities: we could represent multisets as vectors in which the multiplicities of the symbols in V are stored. The leaves of the MTBDDs for the signatures would then be labeled with the representation of $\overline{R}(s, B)$ instead of with real numbers. The drawback of this option is that we have to make considerable changes in the BDD-library, since all operations and in particular the caching strategies for BDD nodes are optimized for storing integer or floating-point values.

A much simpler approach is to represent the multisets as MTBDDs in the same way as we have done for representing partitions. We pre-suppose an arbitrary, but fixed order on the symbols in V, i. e. $V = (\mu_0, \ldots, \mu_{n-1})$. We introduce $l = \lceil \log_2 n \rceil$ new BDD variables $\boldsymbol{a} = (a_{l-1}, \ldots, a_0)$ which encode the index of the symbols. A multiset $\overline{S} \in \mathfrak{M}(V)$ is then encoded by an MTBDD $\overline{\mathcal{S}}(\boldsymbol{a})$ such that $\overline{\mathcal{S}}(\boldsymbol{a} \leftarrow \langle i \rangle) = \mathbf{1}_{\overline{S}}(\mu_i)$ (here $\langle i \rangle$ again denotes the binary encoding of i). Instead of leaves which carries the transition rates, we have MTBDDs for the multiset encoding, resulting in an MTBDD $\mathcal{R}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{a})$ for the transition rate matrix.

This representation integrates seamlessly into our BDD-based framework: if we use a variable order such that the variables for block numbers and multiset encoding are placed after the state variables, we do not need to modify any operations—not even the signature computation and partition refinement. Since all leaves are now labeled with multiplicities of symbolic rates, no floating-point numbers are necessary anymore. The algorithm works with integer values solely. A further advantage is that this technique allows sharing of parts of the multiset encoding.

Obtaining an ordinary CTMC from a symbolic one, given an interpretation of the symbols is efficiently possible:

If the interpretation $I : V \to \mathbb{R}^{\geq 0}$ is given by an MTBDD $\mathcal{I}(\boldsymbol{a})$ such that $\mathcal{I}(\boldsymbol{a} \leftarrow \langle i \rangle) = I(\mu_i)$, we can obtain the transition rate matrix $\mathcal{R}(\boldsymbol{s}, \boldsymbol{t})$ from $\overline{\mathcal{R}}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{a})$ by

$$\mathcal{R}(\boldsymbol{s}, \boldsymbol{t}) = \mathcal{Q}^{+}_{\boldsymbol{a}} : \big(\mathcal{I}(\boldsymbol{a}) \cdot \overline{\mathcal{R}}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{a})\big). \tag{4}$$

4 Experiments

We have implemented the three variants of the SigRefine algorithm described in this paper using C++ as the programming language and the g++ compiler version 4.4.1. For the construction and manipulation of OBDDs and MTBDDs we used the Cudd library [16]. For the rational arithmetic we took the GNU Multiprecision Library (GMP) [17]. The three variants of the refinement algorithms are denoted by SigRef_R for the rational arithmetic; SigRef_F, for the floating-point arithmetic; and SigRef_S, for the version with symbolic representation of the transition rates. We used the default value 10^{-12} for the parameter ε, which controls the creation of new leaves in the floating-point version.

The experiments were conducted on a Dual Core AMD Opteron™ 2.4 GHz CPU with 4 GB of main memory running Linux in 64-bit mode. We have stopped

Table 1: Size of the example models

Model	States	Transitions
cycling-2	4666	18342
cycling-3	57667	305502
cycling-4	431101	2742012
fgf	80616	562536
polling-12	73728	503808
polling-13	159744	1171456
polling-14	344064	2695168
polling-15	737280	6144000
polling-16	1572864	13893632
ftwc-2-2	15769	91232
ftwc-2-3	256932	1697760
ftwc-2-4	3803193	27771984
ftwc-3-1	23040	153600
ftwc-3-2	1889947	15302784
kanban-3	58400	446400

Model	States	Transitions
robot-25	61200	325917
robot-26	68900	367397
robot-27	77220	412253
robot-28	86184	460617
robot-29	95816	512621
robot-30	106140	568397
p2p-4-4	65536	524289
p2p-4-5	1048576	10485761
p2p-4-6	16777216	201326593
p2p-5-4	1048576	10485761
p2p-5-5	33554432	419430401
p2p-5-6	1073741824	16106127361
p2p-6-5	1073741824	16106127361
p2p-7-5	34359738368	601295421441
kanban-4	454475	3979850

any experiment that took more than 7200 seconds or required more than 3 GB of main memory.

We consider seven different example models from the literature to evaluate the performance of the algorithms: a fault-tolerant worstation cluster system (`FTWC`) [18], a peer-to-peer (`P2P`) protocol based on BitTorrent (studied in [19]), a cyclic server polling system [20], a robot moving through an $n \times n$ grid [21] (`robot`), a Kanban production system [22], and two biological models: the first one describes the Fibroblast growth factor signaling (`FGF`) within cells [23], and the second one is a probabilistic model of cell cycle control in eukaryotes (`cycling`) [24].

For the FTWC model, we converted the SAN (Stochastic Activity Network) specification to the PRISM input language. We obtained the PRISM specifications of the other six models from `http://www.prismmodelchecker.org/casestudies/index.php`.

All but the FGF model are parametrized. The first two models have two parameters. For FTWC, they denote the number of computers in the system and the number of memory modules in each computer, respectively. For `P2P`, they represent the number of clients and the number of blocks of the file to be transmitted, respectively. The remaining models have only one parameter: for the polling benchmark, the parameter denotes the number of servers; for the robot benchmark, the size of the grid; in the Kanban benchmark, the parameter denotes the number of tokens in the system, and for the cell cycle control it denotes the initial number of molecules.

For the sake of simplicity, we start with the trivial initial partition $P^{(0)} = \{S\}$. With the exception of the Kanban model, all of these Markov chains can be minimized, i. e., the quotient system is smaller than the original one. The quotient model of the Kanban system, however, has the same size as the input model. Table 1 contains the number of states and transitions of all model instances before minimization.

The results we obtained with the three program versions are shown in Table 2. For each benchmark, the table contains three lines of information; the first is for SIGREF$_R$; the second, for SIGREF$_F$; and the third line contains the results obtained using SIGREF$_S$.

If we compare the rational and the floating-point variant, the first observation is that there is a benchmark, namely `fgf`, for which the floating-point variant did not terminate. A more detailed look at the program output for this case shows that after eight iterations, the algorithm starts alternating between a partition consisting of 38829 blocks and one with 38833 blocks, thereby never reaching a fixed point. Numerical errors are the reason why an unnecessarily fine partition P is computed. When the signatures are computed w. r. t. P, the rounding error which had made the signatures of two blocks B, B' different, does not occur. Therefore the signatures of the states in B and B' become identical again. Therefore, B and B' are merged again. The refinement of this finer partition results in P, closing the cycle.

The second observation is that there are some benchmarks, e. g. `cycling-3`, `cycling-4`, and all `ftwc`-benchmarks (with the exception of `ftwc-2-2`) for which SIGREF$_F$ yields a finer result due to rounding errors than SIGREF$_R$. It cannot be guaranteed that this partition is a correct bisimulation at all, since rounding errors can make the signatures of states with slightly different rate equal. Furthermore, in these cases SIGREF$_F$ needs more iterations to reach the fixed point. This increases the runtime considerably.

On benchmarks for which both the variant with rational and with floating-point arithmetic yield the same result, the runtime and memory consumption of both tools are almost identical (with rational arithmetic a few hundred kilobytes more memory are required). That the runtimes are almost identical may be surprising: typically, rational arithmetic is considered much slower than floating-point arithmetic which is directly supported by the CPU. We performed a detailed profiling of SIGREF$_R$ and measured the fraction of the runtime consumed by the rational arithmetic. In no case was it more than a few percent, because the only affected function is the computation of the signatures. The runtime of this operation—like most BDD operations—is clearly dominated by cache look-ups. They are needed to keep the BDD reduced (so-called UniqueTable) and to avoid unnecessary re-computations of intermediate results (ComputedTable).

Next we compare the version with rational arithmetic, SIGREF$_R$, and the version with the symbolic representation of transition rates, SIGREF$_S$. SIGREF$_R$ always returns the coarsest bisimulation for the current interpretation of the rate symbols, whereas SIGREF$_S$ yields the coarsest interpretation-independent bisimulation. The latter may be finer, but—as we can see in Table 2—for most of the models, the sizes of the quotient systems are identical. The only exceptions are the three `cycling` benchmarks, for which SIGREF$_S$ creates more blocks.

On the benchmarks for which both tools return the same result, SIGREF$_S$ is in most cases slightly slower than SIGREF$_R$. This is due to the additional BDD variables for encoding the rates, which make the MTBDDs larger. This is also the reason why SIGREF$_S$ requires a few Megabytes more memory. An exception

Table 2: Experimental results (first line: SIGREF$_R$, second line: SIGREF$_F$, third line: SIGREF$_S$)

Model	Iter.	Blocks	Time [s]	Mem. [MB]
cycling-2	6	3511	0.87	69.09
	6	3511	0.84	66.91
	6	3997	1.14	57.17
cycling-3	6	40659	19.55	146.18
	13	43742	48.29	149.07
	6	48138	23.81	176.58
cycling-4	6	282943	213.69	972.87
	13	321416	573.04	972.64
	6	339367	311.98	1304.48
fgf	9	38639	70.63	226.49
		— No termination —		
	9	38639	73.52	254.45
polling-12	23	6144	17.20	94.79
	23	6144	17.11	92.55
	23	6144	20.56	85.52
polling-13	25	12288	52.60	121.39
	25	12288	52.44	119.15
	25	12288	58.27	125.39
polling-14	27	24576	139.42	198.44
	27	24576	139.44	196.19
	27	24576	166.96	200.02
polling-15	29	49152	342.98	417.63
	29	49152	342.37	415.39
	29	49152	429.88	377.95
polling-16	31	98304	870.61	806.26
	31	98304	869.71	804.01
	31	98304	1101.67	749.89
ftwc-2-2	3	703	0.31	49.20
	13	703	1.25	55.79
	3	703	0.27	45.43
ftwc-2-3	3	2145	1.32	75.31
	16	10557	56.35	139.35
	3	2145	1.24	70.59
ftwc-2-4	3	5151	6.18	90.96
	20	93866	919.41	854.40
	3	5151	5.08	94.14
ftwc-3-1	3	969	0.71	69.04
	13	2126	7.03	70.64
	3	969	0.71	66.18
ftwc-3-2	3	9139	14.73	185.31
	21	24249	273.34	387.83
	3	9139	15.61	185.93
kanban-3	7	58400	49.40	267.68
	7	58400	52.31	248.10
	7	58400	52.59	284.11

Model	Iter.	Blocks	Time [s]	Mem. [MB]
robot-25	49	60600	72.18	132.50
	49	60600	72.09	130.24
	49	60600	108.91	123.61
robot-26	51	68250	87.48	134.40
	51	68250	87.23	132.15
	51	68250	127.78	137.68
robot-27	53	76518	99.56	135.78
	53	76518	99.51	133.53
	53	76518	146.66	149.06
robot-28	55	85428	120.03	139.88
	55	85428	119.81	137.63
	55	85428	162.63	163.19
robot-29	57	95004	140.98	145.19
	57	95004	140.88	142.92
	57	95004	197.68	168.32
robot-30	59	105270	201.14	162.48
	59	105270	164.18	160.21
	59	105270	228.92	181.07
p2p-4-4	3	70	0.07	39.72
	3	70	0.06	37.46
	2	70	0.04	37.50
p2p-4-5	3	126	0.72	65.90
	3	126	0.69	63.61
	2	126	0.48	63.65
p2p-4-6	3	210	12.33	122.77
	3	210	12.27	120.64
	2	210	7.41	120.56
p2p-5-4	3	105	0.36	45.83
	3	105	0.36	43.62
	2	105	0.23	43.62
p2p-5-5	3	196	8.00	83.20
	3	196	7.93	80.96
	2	196	8.77	81.11
p2p-5-6	3	336	887.49	263.14
	3	336	885.55	261.18
	2	336	563.10	260.90
p2p-6-5	3	266	267.64	88.79
	3	266	266.60	86.43
	2	266	137.60	85.89
p2p-7-5	3	336	2844.46	110.22
	3	336	3780.50	107.53
	2	336	1580.29	106.73
kanban-4	8	454475	741.55	2582.63
	8	454475	749.98	2483.12
	8	454475	816.14	2604.74

to this trend are the p2p benchmarks. On these, SIGREF$_S$ requires one iteration less than SIGREF$_R$ to reach the fixed point. This also demonstrates the effect that $\overline{R}(s, S') \neq \overline{R}(t, S')$, but for the current Interpretation I, $I(\overline{R}(s, S')) = I(\overline{R}(t, S'))$ for some set $S' \subseteq S$. Therefore, after one refinement step, SIGREF$_R$ yields a coarser partition than SIGREF$_S$, although in the end they return the same result.

5 Conclusion

In this paper we have presented two different approaches which can be used for reliable bisimulation computation: the first one is the usage of rational arithmetic for all numerical computations. This yields an algorithm which is clearly superior to the standard variant that is based on floating-point arithmetic: using rational arithmetic, we can always obtain the coarsest bisimulation, the runtime and memory overhead is negligible, and in those cases where floating-point arithmetic creates a partition which is unnecessarily fine, rational arithmetic is even faster than floating-point arithmetic. Furthermore, termination is guaranteed if rational arithmetic is used, which is not the case for floating-point arithmetic, as one of our example benchmarks has shown.

The second approach relies on a symbolic representation of the transition rates. Using this technique we can compute the coarsest interpretation-independent bisimulation, i. e. the coarsest bisimulation which does not depend on the actual values of the transition rates. Our benchmarks have shown that we nevertheless obtain in many cases the same result as with rational arithmetic. Only for a few exceptions the algorithm returns a finer partition. This symbolic handling of the transition rates causes a little overhead due to additional BDD-variables. But its advantage is that the actual values of the rates can be chosen after quotient computation.

In summary, we can conclude that there is no reason to use floating-point arithmetic for signature-based bisimulation computation. Rational arithmetic produces reliably the coarsest bisimulation without any noticeable overhead. If the option to modify the transition rates after minimization is required, it is advantageous to use the algorithm with symbolic transition rates instead of re-computing the quotient after changing the rates.

The techniques presented here are not restricted to (strong) bisimulation for CMTCs. They can also be applied to other types of bisimulation, for instance weak and backward bisimulation for CTMCs, to strong, weak, and branching bisimulation on interactive Markov chains, and to (strong) bisimulation on discrete-time Markov chains.

Acknowledgement. We thank Holger Hermanns from the Saarland University for his helpful comments.

References

1. Wegener, I.: Branching Programs and Binary Decision Diagrams – Theory and Applications. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA (2000)
2. Fisler, K., Vardi, M.Y.: Bisimulation minimization and symbolic model checking. Formal Methods in System Design **21**(1) (2002) 39–78
3. Katoen, J.P., Kemna, T., Zapreev, I., Jansen, D.N.: Bisimulation minimization mostly speeds up probabilistic model checking. In Grumberg, O., Huth, M., eds.: 13$^{\text{th}}$ Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Number 4424 in LNCS, Braga, Portugal, Springer-Verlag (2007) 87–101
4. Böde, E., Herbstritt, M., Hermanns, H., Johr, S., Peikenkamp, T., Pulungan, R., Rakow, J., Wimmer, R., Becker, B.: Compositional dependability evaluation for STATEMATE. IEEE Transactions on Software Engineering **35**(2) (2009) 274–292
5. Blom, S., Haverkort, B.R., Kuntz, M., van de Pol, J.: Distributed Markovian bisimulation reduction aimed at CSL model checking. In Černá, I., Lüttgen, G., eds.: 7$^{\text{th}}$ Int'l Workshop on Parallel and Distributed Methods in Verification (PDMC). Vol. 220(2) of Electronic Notes in Theoretical Computer Science. (2008) 35–50
6. Derisavi, S.: A symbolic algorithm for optimal markov chain lumping. In Grumberg, O., Huth, M., eds.: 13$^{\text{th}}$ Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Number 4424 in LNCS, Braga, Portugal, Springer-Verlag (2007) 139–154
7. Derisavi, S.: Signature-based symbolic algorithm for optimal Markov chain lumping. In: 4$^{\text{th}}$ Int'l Conf. on Quantitative Evaluation of Systems (QEST), Edinburgh, Scotland, IEEE Computer Society Press (2007) 141–150
8. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. Information Processing Letters **87**(6) (2003) 309–315
9. Wimmer, R., Derisavi, S., Hermanns, H.: Symbolic partition refinement with dynamic balancing of time and space. In Rubino, G., ed.: 5$^{\text{th}}$ Int'l Conf. on Quantitative Evaluation of Systems (QEST), Saint-Malo, France, IEEE Computer Society Press (2008) 65–74
10. Blom, S., Orzan, S.: Distributed state space minimization. Software Tools for Technology Transfer (STTT) **7**(3) (2005) 280–291
11. Wimmer, R., Herbstritt, M., Hermanns, H., Strampp, K., Becker, B.: Sigref – A symbolic bisimulation tool box. In: 4$^{\text{th}}$ Int'l Symposium on Automated Technology for Verification and Analysis (ATVA). Vol. 4218 of LNCS., Springer-Verlag (2006) 477–492
12. Baier, C., Katoen, J.P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. Information and Computation **200**(2) (2005) 149–214
13. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Computers **35**(8) (1986) 677–691
14. IEEE Computer Society Standards Committee. Working group of the Microprocessor Standards Subcommittee, American National Standards Institute: IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Standard 754-1985. IEEE Computer Society Press, Silver Spring, MD 20910, USA (1985)
15. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys **23**(1) (1991) 5–48
16. Somenzi, F.: CUDD: Cu decision diagram package, release 2.4.2 (2009)
17. GNU: GNU multiple precision arithmetic library (GMP), version 4.3.1 (2009) `http://gmplib.org`.

18. Sanders, W.H., Malhis, L.M.: Dependability evaluation using composed SAN-based reward models. Journal Parallel and Distributed Computing **15**(3) (1992) 238–254
19. Kwiatkowska, M., Norman, G., Parker, D.: Symmetry reduction for probabilistic model checking. In: Int'l Conf. on Computer Aided Verification (CAV). Vol. 4114 of LNCS. (2006) 234–248
20. Ibe, O., Trivedi, K.: Stochastic Petri net models of polling systems. IEEE Journal on Selected Areas in Communications **8**(9) (1990) 1649–1657
21. Younes, H., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. Software Tools for Technology Transfer (STTT) **8**(3) (2006) 216–228
22. Ciardo, G., Tilgner, M.: On the use of Kronecker operators for the solution of generalized stochastic Petri nets. ICASE Report 96–35, Institute for Computer Applications in Science and Engineering (ICASE) (1996)
23. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theoretical Computer Science **319**(3) (2008) 239–257
24. Lecca, P., Priami, C.: Cell cycle control in eukaryotes: A BioSpi model. Electronic Notes in Theoretical Computer Science **180**(3) (2007) 51–63