Modular State Level Analysis of Distributed Systems Techniques and Tool Support

Peter Buchholz, Peter Kemper

Informatik IV, Universität Dortmund
D-44221 Dortmund, Germany

Abstract. Techniques from different areas are combined to analyze parallel and distributed software within a common framework. They include bisimulation equivalences known from process algebras, Kronecker representations of labelled (stochastic) transitions systems known from performance analysis using Markov chains, and ordered natural decision diagrams (ONDDs) as a generalization of ordered binary decision diagrams famous in hardware verification as well as for the analysis of Boolean functions in general. The employed analysis tools are all part of a toolbox built on the abstract Petri net notation (APNN), a model interchange format using Petri nets. In this context we experience a cross fertilization of different research fields within a Petri net context. We exercise Lamport's mutual exclusion algorithm to discuss the strengths and limitations of the presented approach.

Keywords: Software analysis, Petri nets, Kronecker algebra, bisimulation, ordered natural decision diagrams, model checking

1 Introduction

Development of correct and efficient parallel and distributed software is by no means a trivial task. Many formalisms have been developed to obtain a clear distinction between sequential and parallel elements of a task. A popular point of view is to consider such systems as a set of communicating sequential processes, e.g. in modeling formalisms like CCS [20], CSP [14], automata networks [1], and superposed generalised stochastic Petri nets [12], but also for programming interfaces like MPI, PVM, or distributed C (EPOCA). Communication between processes is either asynchronous (by message passing) or synchronous (rendezvous), where the latter is more general, since every asynchronous communication operation can be easily described using synchronous communication primitives (see [20]), whereas the representation of synchronous communication using asynchronous primitives is much harder to realize. Thus synchronous communication is usually used in low level formalisms supporting functional analysis, whereas asynchronous communication is often part of high level paradigms for parallel or distributed programming. Since this paper focuses on model based analysis, we consider synchronous communication.

Concurrent programs have a potential for speed up due to parallel execution but carry the crux of potential deadlock and other unexpected, undesired behavior. Consequently functional analysis of parallel programs is an important

topic. A classical brute force approach is the enumeration of all possible cases, which results in a set of reachable states. Its general drawback is the state space explosion problem: even trivial examples can cause excessive dimensions in state spaces. Various techniques have been developed in different research areas to handle this problem at least for certain special, but still relevant cases and up to large dimensions of state spaces. Examples of such techniques are: ordered binary decision diagrams (OBDDs) for verification, Kronecker representations for performance analysis, also useful for model checking, reduction of components according to equivalences in Process algebras to mention only a few examples. We consider combinations of such concepts and demonstrate how they fit together for the analysis of concurrent software. We exercise Lamport's mutual exclusion algorithm as a small example to indicate the strengths and limitations of our approach. The software tools we present belong to a much larger set of tools around a model interchange format named "abstract Petri net notation" (APNN) [4] and its toolbox [3].

The paper is organized as follows: Section 2 describes the combination of different methods to represent and analyze models of synchronously communicating components. These methods include directed acyclic graphs for state space representation, Kronecker operations to describe transition relations and equivalences to reduce components before composition. Afterwards, in Sect. 3, we introduce a toolbox which supports modular analysis approaches. In Sect. 4, the modular analysis of a non-trivial example is presented. The paper ends with the conclusion including an outline of further work.

2 Representation and Analysis of Synchronized Processes

A set of processes can be but need not be sequential, hence we allow for atomic operations, sequences of such operations, and fork and join inside a process. Communication takes place via synchronous interaction. This scenario can be formalized in various manners, e.g. as process algebras like CCS [20] (if elementary processes are sequential) or as Petri nets with superposition of transitions or as automata with synchronization. We initially consider Petri nets for modeling and visualization and mainly automata for subsequent analysis.

The well known dining philosopher problem serves as a running example. A philosopher shows four activities: he or she thinks, gets two forks, eats, and puts two forks back. N philosophers sit around a table and share a total of N forks, i.e. philosopher i shares a fork to his left with $i - 1$ and to his right with $i + 1$ - the index is applied modulo N to match the cyclic setting of philosophers. We model this by a set of N processes, where each process relates to one philosopher and the fork to his left. The i -th process is synchronized with the $i + 1$ -th and $i - 1$ -th via access of forks. This example can be nicely visualized and formalized as a Petri net.

Definition 1. Place/transition net

A P/T -net is a 5 tuple (P, T, I^-, I^+, M_0) where P and T are non-empty, finite, and disjoint sets, $P \cap T = \emptyset$, $I^- , I^+ : P \times T \rightarrow \mathbb{N}_0$ are the incidence functions and $M_0 : P \rightarrow \mathbb{N}_0$ is the initial marking, as a special case of a marking $M : P \rightarrow \mathbb{N}_0$.

Fig. 1. A dining philosopher with one of his forks

An advantage of P/T-nets is their visual representation, where places are circles, transitions are boxes and incidence functions are directed, weighted arcs where arc weights of 1 are frequently omitted for readability. The initial marking is represented either as a number of dots (tokens) or numbers at the corresponding places. Let $\bullet t = \{p \in P \mid I^-(p, t) > 0\}$ denote the preset of a transition t , and $t\bullet = \{p \in P \mid I^+(p, t) > 0\}$ denote the postset. The left part of Fig. 1 indicates a philosopher process i of our example with his fork to his right and the right part of Fig. 1 indicates the neighboring process $i + 1$. The dynamic behavior of a P/T-net results from the enabling and firing of its transitions: a transition t is enabled at a marking M if $M(p) \geq I^-(p, t)$ for all $p \in \bullet t$, e.g. the initial marking in Fig. 1 enables transitions $think_i$ and $think_{i+1}$. An enabled transition t at marking M fires and yields successor marking $M'(p) = M(p) + I^+(p, t) - I^-(p, t)$. Starting from the initial marking M_0 , successive application of the firing rule for all enabled transitions yields the set of reachable markings and a reachability graph. The latter is a directed graph, where nodes are given by the set of reachable markings and arcs result from the firing of transitions. If arcs are labeled with the corresponding transition identifier, we obtain a labeled state transition system. The set of reachable markings of an isolated P/T-net of process i is given in the table to the right of Fig. 2, index i is omitted for readability and only places with a marking greater than 0 are denoted.

We consider P/T-nets with synchronization: two P/T-nets are synchronized by fusion of those transitions which are selected for synchronization. This allows to describe processes with synchronization of rendez-vous type in a natural manner, e.g. Fig. 1 shows two philosopher i and $i + 1$ which are synchronized by merging transition get_i and put_{i+1} .

A general concept of synchronization uses labels for transitions, i.e., each transition is labeled with some label from a finite set of labels. In a composition identically labeled transitions are fused. A similar form of synchronization is used

Fig. 2. Reachability graph of philosopher with one fork and its reachability set in process algebras [20] and different classes of Petri-Nets [5,6]. If the reachability set is finite, then the reachability graph of a P/T-net can be interpreted as a finite automaton. Transition labels in the automaton result from the corresponding transition labels in the P/T-net. We will not further consider P/T-nets, because all analysis steps we present in the sequel make use of the description of a system as a set of synchronized automata. Whether the automata result from the reachability graph of a Petri-net or the derivation graph of a process algebra specification is not relevant for the analysis. We have used a Petri-net formalism for specification since it has a nice visual representation. However, for other purposes process algebras might be more adequate to specify components.

Definition 2. An automaton is a 4 tuple $A = (S, \delta, s_0, L)$ where $S = \{0, 1, \dots, n-1\}^1$ is the set of states with cardinality n and initial state $s_0 \in S$. $\delta \subseteq S \times S \times L$ is the state transition relation for a finite set of labels L , where a state transition from a state s_x to a state s_y carries a label $l \in L$, such that $(s_x, s_y, l) \in \delta$.

We consider non-deterministic automata, such that δ is a relation and not necessarily a function. An automaton can be represented as an node and arc labeled graph, e.g. by the reachability graph of Petri net, or represented as a sum of boolean adjacency matrices $\sum_{l \in L} Q_l$ where $Q_l(x, y) = 1$ if $(s_x, s_y, l) \in \delta$ and 0 otherwise. The reachability graph of a dining philosopher is shown on the left side of Fig. 2 and can be represented by 6 matrices including all together 9 non-zero elements. It is often not necessary to distinguish all labels at the automata level. Thus we adopt the hiding mechanism of process algebras and use the convention that transitions which need not be distinguished result in unlabeled arcs in the automaton. If we consider in the example only those transitions that are required for synchronization, then we obtain the graph shown at the right side of Fig. 2. This graph can be represented by 5 matrices with 9 non-zero elements.

For synchronization between a set of N automata $A_1, A_2, \dots, A_N$ we use the index to characterize the different automata and define the synchronized automaton $A = A_1 | A_2 | \dots | A_N = (\times_{i=1}^N S_i, \delta, (s_1, s_2, \dots, s_n), \cup_{i=1}^N L_i)$ and the set of synchronization labels $LS = \{l \in L | \exists i \neq j : l \in L_i \wedge l \in L_j\}$. The number of states in S_i is denoted as n_i , which implies that $\times_{i=1}^N S_i$ includes $\prod_{i=1}^N n_i$ states.

¹ The state space of an automaton is isomorphic to a finite set of integers, depending on the context we use the notation x and s_x for the x -th state in the set.

Synchronization is of the rendez-vous type and refers to equal labels in A_i and A_j , i.e. neither A_i nor A_j is able to perform $l \in LS$ independently of the other after synchronization.

If one represents δ as $\sum_{l \in L} Q_l$, the synchronization of automata yields a matrix description of δ in the dimension of the cross-product of automata. δ has a space efficient compositional representation as a sum of Kronecker products, see [11,22].

Definition 3. *Kronecker product, Kronecker sum*

Let $Q^1, \dots, Q^N$ be square matrices of dimension $(k^i \times k^i)$ then their Kronecker product $Q = \bigotimes_{i=1}^N Q^i$ is defined by $Q(x, y) = \prod_{i=1}^N Q^i(x^i, y^i)$ where $x = \sum_{i=1}^N x^i g_i$ and $y = \sum_{i=1}^N y^i g_i$ with weights $g_1 = 1, g_i = k^{i-1} g_{i-1}$ (for $i > 1$). The Kronecker sum $B = \bigoplus_{i=1}^N Q^i$ is then given by $\bigoplus_{i=1}^N Q^i = \sum_{i=1}^N I_{l^i} \otimes Q^i \otimes I_{r^i}$ where I_{l^i}, I_{r^i} are identity matrices of dimension $l^i \times l^i$, resp. $r^i \times r^i$ where $r^i = \prod_{j=1}^{i-1} k^j$, $l^i = \prod_{j=i+1}^N k^j$ and $I(a, b) = 1$ iff $a = b$ and 0 otherwise.

Kronecker operations do not only apply for real valued matrices but also for Boolean matrices if addition is defined as Boolean *or* and multiplication as Boolean *and*. We focus on Boolean matrices. A Kronecker product formalizes the operation of multiplying every matrix element of one matrix with all matrix elements of the other matrices; these products of matrix elements are arranged in lexicographical order in the resulting matrix, for more details see, e.g., [11]. The key observation is that the fact “synchronization successfully takes place if all processes agree (join) to the rendez-vous” can be formally expressed by the nonzero result of a product, i.e., if all terms are nonzero (= all processes agree), the synchronization can take place. If an automaton A_i is not involved in a synchronization for label l , i.e. if $l \notin L_i$, then we define $Q_l^i = I^i$, where the identity matrix I^i is a matrix of dimension $n_i \times n_i$. For $l \in L_i$, Q_l^i is a $n_i \times n_i$ matrix representing all l -labeled transitions in automaton i . Hence one can represent δ by $\sum_{l \in L} \bigotimes_{i=1}^N Q_l^i$. This is an extremely space efficient representation of δ since for a $\prod_{i=1}^N n_i \times \prod_{i=1}^N n_i$ matrix we use only $|L|$ times N matrices of dimension $n_i \times n_i$ (where in practice a lot of matrices will be identity matrices I , that need not be stored at all).

Let R denote the set of reachable states which results from the reflexive, transitive closure of δ including the initial state s_0 . Obviously the set of reachable states R of $A = A_1 | \dots | A_N$ is a subset of $\times_{i=1}^N S_i$ due to synchronization. The set $\times_{i=1}^N S_i$ can be represented as a tree structure of N levels, where nodes at a level i have n_i sons such that a path in this tree corresponds to a state $(s_1, \dots, s_n)$. If one extracts all paths from this structure which refer to unreachable states, one trivially obtains a representation of R , where nodes at a level i provide the reachable fraction of S_i . Let $R(s_1, \dots, s_i) = \{x \in R | x_1 = s_1 \wedge \dots \wedge x_i = s_i\}$ denote the subset of states in R which match the given states of automata $1, \dots, i$. Then $R(s_1, \dots, s_i)$ refers to a subtree with a root node at level $i+1$. Let the root node of a subtree be denoted as $R_i(s_1, \dots, s_{i-1})$, which is the subset of reachable states in S_i for the context of states $s_1, \dots, s_{i-1}$. This notion reflects a sort of conditional reachability set. Consequently $R() = R$ refers to the whole tree and $R_1()$ refers to the root node.

Fig. 3. Tree a) and 2 DAG representations b),c) of the reachability set for $N=3$

Fig. 3 a) shows a tree representation of the philosopher model with 3 processes. This model contains $|R| = 32$ reachable states of $6^3 = 216$ potential states. The tree consists of 23 nodes and 22 arcs. It contains 32 paths to encode the triples which describe a state within $\times_{i=1}^3 S_i$. Employing binary search at each node, one needs $O(\log n_i)$ to search a certain path (state). Nodes at a level i belong to a single automaton A_i , but these nodes can vary in their cardinality according to the conditional reachability. Several nodes and subtrees are equal, which indicates redundancy. As for ordered binary decision diagrams (OBDDs) isomorphic subtrees do not require more than a single representation. Two trees $R(s_1, \dots, s_i)$ and $R(s'_1, \dots, s'_i)$ are equal iff $R_i(s_1, \dots, s_{i-1}) = R_i(s'_1, \dots, s'_{i-1})$ and all pairs of subtrees $R(s_1, \dots, s_i, s_{i+1})$, $R(s'_1, \dots, s'_i, s'_{i+1})$ with $s_{i+1} = s'_{i+1}$ are equal. Clearly at the bottom level, this simplifies to equality of sets $R_N(s_1, \dots, s_{N-1}) = R_N(s'_1, \dots, s'_{N-1})$. If we apply a folding operation similar to OBDDs in a bottom up manner we obtain a unique directed acyclic graph (DAG), whose set of paths is equal to the set of paths in the tree. A DAG representation of R save space, but the effort to find a state/path remains the same as in the tree representation. The corresponding DAG for the tree in Fig. 3 b) uses only 7 nodes, 14 arcs to represent 32 paths (triples, states).

For some analysis algorithms, e.g. in performance analysis based on CTMC analysis [16,9], it is important to be able to assign specific information to a single state during analysis. For such algorithms a unique, bijective mapping $m : R \rightarrow \{0, 1, \dots, |R| - 1\}$ has to be known. If one applies a lexicographical order on states in R , m simply assigns an index according to this total order on the elements of R . The mapping for lexicographical order can be integrated into the DAG structure if one recognizes that one basically has to count the number of leaves to the left of the path of a state s in the tree. The cardinality of leaves is obviously equal among isomorphic subtrees such that it remains invariant under the folding operation, e.g. as indicated by arc labels in Fig. 3 b) for such cardinalities. Consequently, by assigning corresponding weights on arcs of the DAG, one is able to evaluate m for a path $s_1, \dots, s_N$ in the DAG by summation of arc weights at each node which leave from the left positions of s_i in $R_i(s_1, \dots, s_{i-1})$ plus the position of the s_N in $R_N(s_1, \dots, s_{N-1})$. Clearly an

implementation will precompute such weights to avoid the local summation at a node s_i . Fig. 3 c) provides these values as arc labels for our example. Note that the position of state $(3,2,5)$ results from the summation of arc labels 20 and 2 and the position of $s_3 = 5$ inside the leave node, which is 1, such that we obtain $m(2, 3, 5) = 23$, which is correct since $m(0, 0, 0) = 0$.

So far we considered ways for space efficient representations of reachability graphs by Kronecker algebra and representations of reachability sets by DAGs. With these representations it is possible to perform analysis of models with very large state spaces. From the DAG representation of R , reachability of a specific state can be decided in $O(\sum \log n_i)$. From the Kronecker representation of δ , all successors of a state can be computed in a time proportional to the number of successor states. Within reachability analysis, the Kronecker representation can be further exploited using two observations: 1) transitions with labels $l \notin LS$ can occur locally and independently in the components. Thus it is possible to define some canonical ordering among those transitions instead of considering all interleavings when computing successor states of a state during reachability analysis. 2) the state ordering defined due to the Kronecker representation implies a perfect hash function such that a bit vector of length $\prod_{i=1}^N n_i$ is sufficient to decide in $O(1)$ whether a state has been reached or not. For details about the algorithm and its performance see [17].

A well known, but orthogonal approach to reduce complexity is state aggregation based on equivalence relations. The goal is to reduce the number of states and transitions but to retain the possibility to compute the required results. A successful application requires an equivalence relation has to exist which preserves the required results and an aggregation algorithm which performs faster than the analysis of the original system. The latter usually requires a congruence relation with respect to composition via synchronous transitions, such that aggregation can be applied for each subsystem at the automata level and the combination of aggregated subsystems gives an aggregated but still equivalent overall system. In this case, the Kronecker representation is convenient to combine aggregation and composition, because aggregates are computed at the level of automata matrices yielding a matrix description of the aggregated automaton which can be used instead of the original matrices in the Kronecker representations. We briefly outline the steps of equivalence computation in the context of Kronecker based analysis.

A large number of equivalence relations has been proposed in the literature (see e.g., [24] for equivalences in the context of Petri nets), especially equivalences of the bisimulation type are popular. An equivalence relation $\mathcal{R}_i \subseteq S_i \times S_i$ is a bisimulation if for all $(s_x, s_y) \in \mathcal{R}_i$ and all $l \in L_i$:

1. $Q_l^i(x, z) = 1$ implies $Q_l^i(y, z') = 1$ for some z' with $(s_z, s_{z'}) \in \mathcal{R}_i$ and
2. $Q_l^i(y, z) = 1$ implies $Q_l^i(x, z') = 1$ for some z' with $(s_z, s_{z'}) \in \mathcal{R}_i$.

The bisimulation relation with the least number of equivalence classes can be computed for finite state systems as the fixed point of the following partition refinement

$$\mathcal{R}_i^{k+1} = \{(s_x, s_y) | (s_x, s_y) \in \mathcal{R}_i^k \text{ and} \\ \begin{aligned} &Q_l^i(x, z) = 1 \text{ implies } \exists z' \text{ with } (s_z, s_{z'}) \in \mathcal{R}_i^k \text{ such that } Q_l^i(y, z') = 1 \text{ and} \\ &Q_l^i(y, z) = 1 \text{ implies } \exists z' \text{ with } (s_z, s_{z'}) \in \mathcal{R}_i^k \text{ such that } Q_l^i(x, z') = 1 \end{aligned} \},$$

where $\mathcal{R}_i^0$ is some initial relation. For a bisimulation, initial relation $\mathcal{R}_i^0 = S_i \times S_i$ is sufficient. Efficient algorithms to compute the fixed point are known [21,15] and implemented in analysis tools [10]. According to a bisimulation relation $\mathcal{R}_i$, an aggregated automaton can be built by substituting every equivalence class by a single state. Let $\tilde{n}_i$ be the number of equivalence classes and $\mathcal{R}_i[\tilde{s}_x]$ be the x -equivalence class. Then the aggregated automata is defined by $\tilde{n}_i \times \tilde{n}_i$ matrices $\tilde{Q}_l^i$ with $\tilde{Q}_l^i(\tilde{x}, \tilde{y}) = 1$ if $s_x \in \mathcal{R}_i[\tilde{s}_x]$ and $s_y \in \mathcal{R}_i[\tilde{s}_y]$ with $Q_l^i(x, y) = 1$ exist. All remaining elements are 0. Matrices $\tilde{Q}_l^i$ can then be used instead of Q_l^i in the Kronecker representation reducing the state space of the composed model by a factor of $\tilde{n}_i/n_i$. Since bisimulation is a congruence, the composed models using the original and aggregated model are bisimulation equivalent.

Bisimulation relations can also be defined for modified forms of automata. A popular modification yielding a less discriminating equivalence results in hiding of some labels. We define a label l to be local for automaton i if $l \in L_i$ and $l \notin LS$. Let L_i^{loc} be the set of local labels for automaton i . Observe that $Q_l^j = I^j$ for $l \in L_i^{loc}$ and $i \neq j$. Let $Q_{li}^i = \sum_{l \in L_i^{loc}} Q_l^i$. It is easy to show that in the Kronecker representation of δ the terms involving Q_l^i ($l \in L_i^{loc}$) can be substituted by a single term involving Q_{li}^i and matrices $Q_{li}^j = I^j$ for $j \neq i$. In contrast to the original system, different labels from L_i^{loc} can no longer be distinguished. Define $(Q_{li}^i)^* = \sum_{k=0}^{\infty} (Q_{li}^i)^k$. A transitive and reflexive closure algorithm computes $(Q_{li}^i)^*$ within $O(n_i^3)$. A bisimulation equivalence for the automaton results from $(Q_l^i)^* = (Q_{li}^i)^* Q_l^i (Q_{li}^i)^*$ for all $l \in L_i \setminus L_i^{loc}$. This equivalence relation hides the details of transitions from L_i^{loc} and still preserves the external behavior visible via labels $l \in L_i \setminus L_i^{loc}$. Corresponding bisimulations are usually denoted as weak bisimulations. Labels which should not be observed in functional analysis and which are not required for synchronization can be collected in L_i^{loc} and hidden in the aggregated automaton.

So far, bisimulation equivalences do not preserve the reachability of states, i.e., if in the aggregated automaton state $\tilde{s}_x$ is reachable, at least one state $s_y \in \mathcal{R}_i[\tilde{s}_x]$ is reachable, but it is not clear whether all states or more than one state are reachable in the original automaton. The reason is that the equivalence relations consider only the future and not the past behavior. If results with respect to specific states should be computed, the equivalences have to be extended. To formalize the approach, states are characterized by labels and an equivalence relation is computed by refining $\mathcal{R}_i^0$ where $(s_x, s_y) \in \mathcal{R}_i^0$ if s_x and s_y are identically labeled (see [10]). This approach works if the number of state labels is relatively small. However, in cases with a large number of state labels, the resulting number of equivalence classes will also be large. In particular, if the set of reachable states should be computed, the aggregation presented so far does not help because in the initial relation $\mathcal{R}_i^0$ each equivalence class contains a single state.

For reachability analysis we have to consider the past instead of the future behavior, because reachability of a state implies the existence of a path from the initial state. An equivalence relation preserving reachability has been introduced in [7]. Let $\mathcal{R}_i \subseteq S_i \times S_i$ be an equivalence relation such that $(s_0, s_x) \in \mathcal{R}_i$ implies $(Q_{li}^i)^*(0, x) = 1$ (i.e., all states which are in the same equivalence class as the initial state are reachable by internal transitions from the initial state) and for

Fig. 4. Reachability graph of an aggregated philosopher a) and DAG representation of the aggregated reachability set b).

all $(s_x, s_y) \in \mathcal{R}_i$ and all $l \in L_i$:

1. $(Q_l^i)^*(z, x) = 1$ implies $(Q_l^i)^*(z', y) = 1$ for some z' with $(s_z, s_{z'}) \in \mathcal{R}_i$ and
2. $(Q_l^i)^*(z, y) = 1$ implies $(Q_l^i)^*(z', x) = 1$ for some z' with $(s_z, s_{z'}) \in \mathcal{R}_i$.

The relation can be denoted as a weak inverse bisimulation since it considers incoming instead of outgoing transitions. The largest weak inverse bisimulation can be computed using a partition refinement algorithm with the transposed instead of the original matrices. As shown in [7] $(s_x, s_y) \in \mathcal{R}_i$ for some weak inverse bisimulation $\mathcal{R}_i$ implies that if automaton i can be in state s_x after a sequence of synchronized transitions, then it can as well be in state s_y and vice versa. This behavior is exploited for an efficient reachability analysis. In a first step, aggregated automata with respect to weak inverse bisimulation are computed, then reachability analysis for the complete system is performed using the aggregated instead of the original automata. If in the aggregated system state $(\tilde{s}_1, \dots, \tilde{s}_N)$ is reachable, then all states from $\{\mathcal{R}_1[\tilde{s}_1] \times \mathcal{R}_2[\tilde{s}_2] \times \dots \times \mathcal{R}_N[\tilde{s}_N]\}$ are reachable in the original system. If $(\tilde{s}_1, \dots, \tilde{s}_N)$ is not reachable, then all states from $\{\mathcal{R}_1[\tilde{s}_1] \times \mathcal{R}_2[\tilde{s}_2] \times \dots \times \mathcal{R}_N[\tilde{s}_N]\}$ are unreachable too. Exploitation of this results allows a very time efficient generation and space efficient representation of huge reachability sets, for details see [8].

If we compute equivalence relations for the philosophers in the example and declare all transitions which are not needed for synchronization as local, then the relation $\mathcal{R}$ with equivalence classes $\mathcal{R}[\tilde{0}] = (0, 2)$, $\mathcal{R}[\tilde{1}] = (1, 3)$ and $\mathcal{R}[\tilde{2}] = (4, 5)$ is a weak bisimulation in both directions, forward and backward. Thus the aggregated automaton (Fig 4a) with 3 states and 4 transitions can be used instead of the original one in subsequent composition and analysis. Because $\mathcal{R}$ is a weak forward bisimulation, model checking using the aggregated instead of the original system yields identical results and since $\mathcal{R}$ is a weak backward bisimulation, reachability analysis can be performed using the aggregated system. Thus, it is indeed possible to perform all analysis steps on the Kronecker and DAG representation of the aggregated system. Fig. 4b) shows the DAG of the aggregated reachability set which is more compact than the original DAG but contains the same information, e.g., reachability of aggregated state $(\tilde{0}, \tilde{2}, \tilde{1})$ implies that all 8 states from $\{(0, 2) \times (4, 5) \times (1, 3)\}$ are reachable and also the other way, since state $(\tilde{0}, \tilde{1}, \tilde{2})$ is not reachable in the aggregated system, all 8 states from $\{(0, 2) \times (1, 3) \times (4, 5)\}$ are unreachable in the original system.

Fig. 5. Structural overview of the toolbox

3 Tool support

Many software tools for analysis of finite state systems exist. They differ in modeling paradigms, analysis techniques and the kind of results they compute. Often they are standalone developments such that models cannot be interchanged. This implies that a fair comparison of different techniques is often hard, the combination of techniques from different tools is impossible and it also results in a lot of redundancy, because basic modules like state-space generators, graphical interfaces etc. are often reimplemented for a specific tool. Recent efforts aim for standardized interfaces to support the interchange of models and algorithms, e.g. the ISO standardization approach for Petri nets [13], the Petri net kernel of the Humboldt university [18] and the Electronic Tool Integration platform [25]. The presented analysis techniques are integrated in a toolbox, the APNN toolbox [3], which was developed with similar ideas in mind. The toolbox is based on two standardized file formats. First, the so-called abstract Petri net notation (APNN), an extendable file format (formal grammar) for a rather general class of Petri nets, including colored, hierarchical and stochastic nets [4]. The second format is for synchronized (stochastic) automata which matches the formalism introduced in Section 2. Figure 5 gives an overview of the APNN toolbox.

Due to lack of space, we name only those parts which are relevant for the functional analysis of systems. We neglect components that deal with quantitative system analysis using techniques for Markov chain analysis or discrete event simulation. Currently two graphical interfaces are available to specify Petri Net models structured in synchronized components. Both interfaces generate an APNN description of the model. The APNN description of a model is the input

for analysis modules at the net level. Invariant analysis is often useful to obtain first results – especially upper bounds for token populations on the places of a Petri net, which are in turn helpful to limit the size of automata state spaces in composed nets [16].

The APNN description is read by the module for state space generation. It is possible to generate the state space and transition system for the complete net or for components only. The latter results in an automata network description of the model which contains all information necessary to generate the Kronecker representation of the complete system. Reachability graphs are stored as sparse matrices, one for each transition label. Automata description can also be obtained from other modeling formalism, e.g. from a process algebra specification consisting of the parallel composition of components at the highest level, the description as an automata network can be easily generated by computing the transition systems of the components.

N	$ R $	$ \tilde{R} $	DAG nodes	DAG Size in KByte	non-zeros	Time in sec.
8	12,032	47	27	0.8	72	2
12	1,318,912	322	54	1.6	108	3
16	1.44638e+8	2,207	102	3.1	144	5
20	1.58618e+10	15,127	158	4.8	180	36

Table 1. Sizes and efforts to generate and represent the philosophers example.

The automata description is the interface for different state based analysis modules, e.g. a module for equivalence computation and the generation of reduced automata, a module to generate the reachability set using DAGs, and a module to perform model checking for computational tree logic (CTL) formulas using DAGs and Kronecker representations.

To present first results, we consider the analysis of the philosophers example using the modules of the toolbox. Tab. 1 includes results for configurations with up to 20 philosophers. The first column contains the number of philosophers, $|R|$ and $|\tilde{R}|$ are the number of markings in the reachability set and the aggregated reachability set, respectively. The size of the reachability set grows very rapidly with an increasing number of philosophers, whereas the aggregated reachability set is relatively small even for a larger number of philosophers. The following two columns include information about the DAG to represent $|\tilde{R}|$. As shown above, knowledge of $\tilde{R}$ and the equivalence classes allows us to characterize R completely. For all configurations of the example, the number of nodes in the DAG is very small compared to the size $\tilde{R}$, let alone compared to the size of R . Memory requirements to store the DAG are shown in the fifth column. Apart from the DAG, the equivalence classes and the matrices have to be stored to represent the reachability set and graph. The number of non-zero elements in all matrices which are needed to represent the reachability graph is shown in column six. The last column includes the total time required to generate the compact representations of reachability set and graph starting with the APNN-description of the model. The time is measured as “wallclock time in seconds” on a Sun UltraSparc workstation with 167MHz CPU and 128 MByte of main memory. Since the different analysis steps are performed by single modules communicating

via a file interface, time includes the effort to load programs and read and write files. However, even for the largest configuration with more than 15 billions reachable markings, the compact representation is generated in about half a minute and requires less than 10 KByte memory. The compact representation can afterwards be used in further analysis steps including model-checking or performance analysis.

Observe that we did not exploit symmetries in the model or identities of different components. This would be an additional step to improve analysis. However, we obtained similar result for non-symmetric configurations where some of the philosophers pick up forks one after the other. With the approach even configurations with more than 20 philosophers can be handled. In this case it is preferable to group two or three philosophers to a single automaton which can be aggregated to a small automaton. The philosophers example includes some features, which are common in parallel or distributed systems and support our analysis approach. Components have some internal transitions such that corresponding automata can be substantially aggregated. Additionally, synchronization takes place between adjacent components and not globally. The following example is less favorable and demonstrates limitations of our approach.

4 Example

Lamport's mutual exclusion algorithm [19] for shared-memory systems without test-and-set instructions, but with atomic read and write operations is, of course, an academic example, but it is complex and not easy to analyze. We analyze the algorithm for a system consisting of N processes cycling between local computing and access to shared memory. Figure 6 gives the pseudo code for process i . The basic idea is that in systems where contention to a shared resource is rare, it is not efficient to inspect the state of all other processes before accessing the shared resource. By a sophisticated use of variables x and y it is possible to assure exclusive access without first scanning all other processes. However, it is not straightforward to describe the meaning of x and y . It is worth mentioning that the algorithm in its original setting is not symmetric for processes. Since in the *for*-loop processes are scanned starting with process 1 and ending with N in the code of each process, processes are treated in a different way depending on their number. The difference in process behavior destroys symmetries such that methods reducing state spaces due to symmetries cannot be used for the example. Lamport's algorithm is considered in [2], where colored stochastic Petri nets are applied for its analysis and in [23], where it is modeled as a network of stochastic automata. Our model is in some sense in between these approaches since we use superposed Petri nets which are mapped on automata.

The algorithm is too complex to be described as a flat P/T net. We used a colored net with hierarchies as in [2]. The major difference is that we explicitly model the *for loop*. In contrast to [2] our net describes system with N components interacting via synchronized transitions. The net for component i contains the description of process i plus places for the variables $b[i]$, $x = i$ with complement place $x \neq i$, $y = i$ with complement place $y \neq i$. The first automaton considers additionally the situation $y = 0$. Values of the variables are described in a

```

1.  compute locally ;
2.  start:  $\langle b[i] := true \rangle$  ; /* start access to shared memory */
3.   $\langle x := i \rangle$  ;
4.  if  $\langle y \neq 0 \rangle$  then
5.     $\langle b[i] := false \rangle$  ;
6.    await  $\langle y = 0 \rangle$  ;
7.    goto start fi ;
8.   $\langle y := i \rangle$  ;
9.  if  $\langle x \neq i \rangle$  then
10.    $\langle b[i] := false \rangle$  ;
11.   for  $j := 1$  to  $N$  do await  $\langle not\ b[j] \rangle$  od ;
12.   if  $\langle y \neq i \rangle$  then
13.     await  $\langle y = 0 \rangle$  ;
14.     goto start fi fi ;
15.  critical section ;
16.   $\langle y := 0 \rangle$  ;
17.   $\langle b[i] := false \rangle$  ; /* end access to shared memory */

```

Fig. 6. Lamport's algorithm code for process i .

distributed way. Such a description is not obvious, however, the alternative, where components for the variables x and y are introduced, as in [23], results in a model where the state of the automata for the variables determine the state of the remaining automata and the modular analysis behaves similar to a conventional analysis of the complete net.

For a model with N processes, the state space of the first automaton contains $40 + 4N$ states, the remaining state spaces for the automata $2, \dots, N$ include $20 + 2N$ states each. The size of the automata state spaces depends on the number of processes due to the *for-loop*, where each other process is considered. Aggregation with respect to weak backwards bisimulation reduces the state space of the first automaton to $32 + 4N$ states and the remaining state spaces to $16 + 2N$. For this example we obtain only a small reduction by a constant value with the aggregation approach. This shows the complexity of the processes. Nevertheless, the aggregated reachability set is significantly smaller than the original one, although the sizes of both reachability sets grow rapidly with an increasing number of processes. Results for the example with 3, 4 and 5 processes are shown in Tab. 2. The effort to generate reachability sets for this example is significantly higher than for the philosophers example. However, for the largest configuration ($N = 5$) with nearly 8 millions of states, it takes less than an hour to generate the reachability set and represent it in a very compact form requiring about 26 KByte of memory. For $N = 6$, $|\tilde{R}| \geq 1.1 \cdot 10^8$ states, but could still be represented in a compact form. Nevertheless generation times become extremely large.

N	$ R $	$ \tilde{R} $	DAG nodes	DAG Size in KByte	non-zeros	Time in sec.
3	16,683	11,337	40	4.7	11,568	18
4	347,875	222,046	84	12.9	16,042	129
5	7,856,309	4,760,531	145	25.8	21,056	2,948

Table 2. Sizes and efforts to generate and represent the Lamport's algorithm.

The example shows also the limits of the approach, if processes are highly synchronized such that abstraction cannot be applied to reduce intermediate reachability sets. However, even for this example the approach outperforms conventional state space generation which fails completely for the model with $N = 5$ on the same hardware due to memory limitations.

5 Conclusion

We propose a methodology to analyze distributed software systems at the state level in a modular way. Starting from a specification of the software system as a P/T-net consisting of components interacting via synchronized transitions, we generate a network of communicating automata. For the analysis of this automata network, we apply three different concepts to manage the inherent complexity of state level analysis. First, we avoid the explicit generation of the reachability graph by representing it as a sum of Kronecker products of small automata matrices. Second, we avoid the explicit enumeration of the state space (reachability set) by representing it as a directed acyclic graph. As a third step, we integrate state level aggregation due to equivalence relations to reduce automata state spaces a priori. Aggregation can be naturally integrated into the Kronecker description of the reachability graph and can therefore be performed in a very efficient way. All concepts together reduce drastically the memory requirements to represent large reachability sets and graphs. At least from the memory perspective, the state space explosion problem can be managed for most models which are described by synchronously communicating components. The situation is a little bit different if one considers the time requirements to build the data structures for really large systems with several hundred millions or some billions of states. As shown by the dining philosophers example it is sometimes possible to generate data structures for such large models in a few seconds. However, if interactions between components become more complex such that a priori aggregation has only small effects on the size of the reachability set, then the handling of huge state spaces is still a very time consuming task, as the results for the second example indicate.

The usability of any analysis technique relies on the availability of appropriate software tools incorporating the technique. The modular state level analysis is part of a general toolbox based on two standardized file formats to describe general classes of Petri-nets at a higher level and synchronized automata at a lower level. The toolbox includes, apart from modules for functional system analysis, also modules for performance or reliability analysis based on Markov chain technique. These techniques follow similar ideas as proposed here for the functional case.

Future work will consider the exploitation of symmetries during the reachability set generation and the compositional computation of equivalence relations. We plan to implement analysis steps in a distributed way on a workstation cluster and to further enhance the toolbox interconnection with other tools.

References

1. A. Arnold. *Finite Transition systems*. Prentice Hall, 1994.
2. G. Balbo, S. C. Bruell, P. Chen, and G. Chiola. An example of modelling and evaluation of a concurrent program using coloured stochastic Petri nets: Lamport's fast mutual exclusion algorithm. *IEEE Trans. on Parallel and Distributed Systems*, 3(2):221–240, 1992.

3. F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEDS. In R. Pujanger, N. N. Savino, and B. Serra, editors, *Quant. Eval. of Comp. and Comm. Sys.*, pp. 356–359. Springer LNCS 1469, 1998.
4. F. Bause, P. Kemper, and P. Kritzinger. Abstract Petri nets notation. *Petri Net Newsletter*, (49):9–27, 1995.
5. E. Best, W. Fraczak, R. P. Hopkins, H. Klaudel, and E. Pelz. M-nets: An algebra of high-level Petri nets, with an application to the semantics of concurrent programming languages. *Acta Informatica*, 35(10):813–857, 1998.
6. P. Buchholz. A notion of equivalence for stochastic Petri nets. In G. De Michelis and M. Diaz, editors, *ATPN'95*, pp. 161–180. Springer LNCS 935, 1995.
7. P. Buchholz. Hierarchical structuring of superposed GSPNs. In *Proc. PNPM'97* pp. 81–90. IEEE CS-Press, 1997.
8. P. Buchholz and P. Kemper. Efficient computation and representation of large reachability sets for composed automata. Technical report, in preparation.
9. G. Ciardo and A. S. Miner. Storage alternatives for large structured state spaces. In R. Marie and B. Plateau, editors, *Proc. 9th Int. Conf. on Mod. Techn. and Tools for Comp. Perf. Eval.*, pp. 44–57. Springer LNCS 1245, 1997.
10. R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: a semantics based tool for the verification of concurrent systems. *ACM Trans. on Programming Languages and Systems*, 15(1):36–72, 1993.
11. M. Davio. Kronecker products and shuffle algebra. *IEEE Trans. on Computer*, 30:116–125, 1981.
12. S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In R. Valette, editor, *ATPN'94*, pp. 258–277. Springer LNCS 815, 1994.
13. J. Billington (ed.). Petri net standard. Committee draft, ISO/IEC JTC1/SC7/WG11, 1992.
14. C. Hoare. *Communicating sequential processes*. Prentice Hall, 1985.
15. P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.
16. P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Software Engineering*, 22(9):615–628, 1996.
17. P. Kemper. Reachability analysis based on structured representations. In J. Billington and W. Reisig, editors, *ATPN'96*, pp. 269–288. Springer LNCS 1091, 1996.
18. E. Kindler and J. Desel. Der Traum von einem universellen Petrinetz-Werkzeug – der Petrinetz-Kern (in German). In *Proc. 3rd Workshop on Algorithmen und Werkzeuge für Petrinetze*, 1996.
19. L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. on Computer Systems*, 5(1):1–11, 1987.
20. R. Milner. *Communication and concurrency*. Prentice Hall, 1989.
21. R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journ. on Computing*, 16(6):973–989, 1987.
22. B. Plateau. On the stochastic structure of parallelism and synchronisation models for distributed algorithms. *Performance Evaluation Review*, 13:142–154, 1985.
23. B. Plateau and K. Atif. Stochastic automata networks for modeling parallel systems. *IEEE Trans. on Software Engineering*, 17(10):1093–1108, 1991.
24. L. Pomello, G. Rozenberg, and C. Simone. A survey of equivalence relations for net based systems. In G. Rozenberg, editor, *Adv. in Petri Nets 1992*, pages 410–472. Springer LNCS 609, 1992.
25. B. Steffen, T. Margaria, and V. Braun. The electronic tool integration platform: concepts and design. *Int. Journ. on Softw. Tools for Technology Transfer*, 1(1/2):9–30, 1997.