ACTA UNIVERSITATIS UPSALIENSIS

Uppsala Dissertations from
the Faculty of Science and Technology

50
JULIEN D’ORSO

New Directions in Symbolic Model Checking

UPPSALA
UNIVERSITET

*Uppsala dissertations from the Faculty of Science and Technology* 50.  
*Distribution: Uppsala University Library, Box 510, SE-75120 Uppsala, Sweden.*

Julien d’Orso  

**New Directions in Symbolic Model Checking**

**ABSTRACT**


In today’s computer engineering, requirements for generally high reliability have pushed the notion of testing to its limits. Many disciplines are moving, or have already moved, to more formal methods to ensure correctness. This is done by comparing the behavior of the system as it is implemented against a set of requirements. The ultimate goal is to create methods and tools that are able to perform this kind of verification *automatically*. This is called *Model Checking*.

Although the notion of model checking has existed for two decades, adoption by the industry has been hampered by its poor applicability to complex systems. During the 90’s, researchers have introduced an approach to cope with large (even infinite) state spaces: *Symbolic Model Checking*. The key notion is to represent large (possibly infinite) sets of states by a small formula (as opposed to enumerating all members).

In this thesis, we investigate applying symbolic methods to different types of systems:  

**Parameterized systems.** We work within the framework of *Regular Model Checking*. In regular model checking, we represent a global state as a word over a finite alphabet. A transition relation is represented by a regular length-preserving transducer. An important operation is the so-called transitive closure, which characterizes composing a transition relation with itself an arbitrary number of times. Since completeness cannot be achieved, we propose methods of computing closures that work as often as possible.

**Games on infinite structures.** Infinite-state systems for which the transition relation is *monotonic* with respect to a well quasi-ordering on states can be analyzed. We lift the framework of well quasi-ordered domains toward *games*. We show that monotonic games are in general undecidable. We identify a subclass of monotonic games: *downward-closed games*. We propose an algorithm to analyze such games with a winning condition expressed as a safety property.

**Probabilistic systems.** We present a framework for the quantitative analysis of probabilistic systems with an infinite state-space: given an initial state $s_{init}$, a set $F$ of final states, and a rational $\theta > 0$, compute a rational $\rho$ such that the probability of reaching $F$ from $s_{init}$ is between $\rho$ and $\rho + \theta$. We present a generic algorithm and sufficient conditions for termination.

Julien d’Orso, Department of Information Technology, Uppsala University, Box 337, SE-75105 Uppsala, Sweden. Email: juldor@it.uu.se
Acknowledgements

First of all, I would like to thank my supervisor, Prof. Parosh Aziz Abdulla. He got me interested in the world of research, and helped me become part of it. Throughout my 4 PhD years in Uppsala, he lent me a considerable amount of support, especially during the numerous periods before a tough deadline. Thank you for your incredible patience!

Many “arigato’s” go to my friend of long date Alexandre David. You treaded alone the dangerous path that lead to Sweden and Uppsala, pioneering the exchange programme between Uppsala university and our school back in Brest, France. Although we have had a few bad times, I guess we survived, didn’t we?

The IT department here in Uppsala has evolved a lot since the day I started. I would like to thank my fellow PhD students at the Algorithmic Verification Group: Aletta Nylén, Pritha Mahata, Johann Demeur, Marcus Nilsson, Lisa Katti, Noémene Ben Henda, Rezine Ahmed and Mayank Saksena. You have made our working place both lively and fun. Keep up the high mood, guys!

Last, but not least, I want to express my deep gratitude to my family back in France for their unwavering support and love. It’s been 5 long years since I left, but my heart as always stayed with you. And you know what? It was worth it!

This work was supported in part by ARTES, the Swedish network for real-time research (http://www.arteres.uu.se/), as well as the European Commission, under the FET project IST-1999-29082 “ADVANCE”.
This thesis includes, summarizes and discusses results reported in 5 research papers, written between 2001 and 2003. The papers are listed below:


**Comments on my participation:**

**Paper A** I participated in developing and writing the technical part if this paper together with Parosh Abdulla.

**Paper B** The technical framework was developed and written jointly with Marcus Nilsson.

**Paper C** The technical framework was developed and written jointly with Marcus Nilsson.

**Paper D** All proofs were written jointly with Parosh Abdulla.

**Paper E** All proofs were written jointly with Parosh Abdulla.
Other work by the author:


Contents

1 Introduction .............................................. 1
   1.1 Background ........................................ 1
   1.2 Parameterized Systems .............................. 4
      1.2.1 Running Example .............................. 5
      1.2.2 Transitive Closure and Column Transducer ... 6
      1.2.3 Methods of Construction for the Transitive Closure ... 8
   1.3 Infinite Games ..................................... 9
   1.4 Infinite Probabilistic Systems ...................... 12
   1.5 Papers and Contributions .......................... 14
   1.6 Conclusions and Future Work ...................... 14

2 Paper A ................................................. 19
   2.1 Introduction ....................................... 19
   2.2 Words ................................................ 22
   2.3 Trees ................................................. 22
   2.4 Tree Automata ....................................... 24
   2.5 Symbolic Transducers .............................. 27
   2.6 Saturation .......................................... 32
   2.7 Termination ......................................... 37
   2.8 Experimental Results .............................. 41
   2.9 Conclusions and Future Work ...................... 44

3 Paper B .................................................. 49
   3.1 Introduction ....................................... 49
   3.2 An Example ......................................... 52
   3.3 Algorithm .......................................... 54
   3.4 Correctness ......................................... 57
   3.5 Implementation ..................................... 62
   3.6 Conclusions and Future Research .................. 64
4 Paper C

4.1 Introduction .................................................. 69
4.2 An Example .................................................... 72
4.3 Soundness ...................................................... 76
4.4 A coarse equivalence ......................................... 79
4.5 Implementation of the equivalence relation ................. 81
4.6 Implementation ................................................ 83

5 Paper D

5.1 Introduction ................................................... 87
5.2 Preliminaries .................................................. 89
5.3 Ordered Games ............................................... 90
5.4 B-Downward Closed Games ................................. 91
5.5 B-LCS ......................................................... 95
5.6 A-Downward Closed Games ................................. 100
5.7 Undecidability of Monotonic Games ....................... 103
5.8 Parity Games ............................................... 106

6 Paper E

6.1 Introduction .................................................. 113
6.2 Transition Systems ........................................... 115
6.3 Markov Chains ............................................... 116
6.4 Algorithm .................................................... 117
6.5 Probabilistic VASS .......................................... 119
6.6 Almost Coarse Markov Chains .............................. 121
6.7 Probabilistic Lossy Channel Systems ..................... 123
6.8 Almost Coarseness of PLCS ................................. 124
6.9 Conclusion, Discussion, and Future Work .................. 128
Chapter 1

Introduction

1.1 Background

The software and hardware communities have come to realize in the preceding decades that as systems tend to grow more complex, and cooperation of many individuals are required to complete a given project, there is a growing issue of potential mistakes, so-called bugs. Even though computer users have become quite tolerant to defects in systems that they use in their everyday life, there are instances where such obvious defects would result in the device or system to be rejected. Without even mentioning safety-critical applications, it is useful to recall that even everyday, apparently simple appliances like television sets or Hi-Fi sound components require quite sophisticated control software (see e.g., [HSLL97]). It would definitely be a disgrace for a user to “reset” a TV because the control software crashed while pressing some button on the remote control...

While testing is an industry-wide practice, experience shows that this is often not enough: some phenomena cannot be simulated in practice, and the amount of time reserved for testing a given product does not come close to allowing a coverage of all possible executions. There is a need for more formal methods and tools that could analyze an implemented system and compare it to its specification. Such methods are called Verification. The ultimate goal of Model Checking is to entirely automate these methods, so that no manual intervention is ever needed during the verification process. As a general rule, in model checking, the user provides two descriptions:

- The model, which is a description of the system to analyze, using some formalism. Mostly, models are written using languages which are variants or extensions of transition systems. Transition systems
consist of a set of states, and a binary relation over states called the transition relation.

- The specification, which is a set of requirements that the model should satisfy. Historically, there have been two main “families” of logics to reason with a system’s behaviour, namely linear-time logics (LTL) and branching-time logics (CTL). We will often be interested in a small subset of these logics, in particular so-called safety properties. Such properties express that “nothing bad will ever happen”.

Then, the user asks the question “Does the model satisfy the requirement?”. Ideally, both the model and specification can be fed into a program that gives automatically a (provably correct) answer.

Since the introduction of model checking (see e.g. [CES83]), many tools that can accomplish the task of verification for finite-state systems have appeared, e.g. VIS ([Gro96]). A limiting factor to their application is the so-called state-space explosion problem: each time a bit of data or memory is added to a system, the size of its state-space is multiplied by 2. Researchers have devised several methods for coping with this problem:

- Approximation methods. Sometimes, it is impossible to accommodate the whole state-space of a system in memory during verification. Then, one uses partial search techniques that under- or over-approximate the system’s reachable states. For example, the model checker SPIN uses an under-approximation called supertrace (see [Hol91]).

- Partial order techniques. One makes the observation that not all interleavings of independent actions need to be explored during verification (see e.g. [GW93, ERV96]).

- Symbolic methods. A set of states is not represented explicitly, but by a formula from which all members of the set can be reconstructed. A very natural symbolic representation is that of boolean formulas. For example, the use of binary decision diagrams (BDD’s, see [Bry86]), a compact representation of boolean functions, has allowed model checking to become practical for the hardware design community (see e.g. [BCL+94]).

An even more challenging problem stems from the fact that many systems that arise naturally have a state-space that can be arbitrarily large, or even infinite, and thus are beyond the capabilities of finite methods and associated tools. Infiniteness can come from distinct aspects of such systems:
1.1. BACKGROUND

- Control structures. It is common to define protocols with an arbitrary number of entities. We call such systems parameterized, since the number of participants can be seen as a parameter of the protocol. One would like to be able to reason about such a protocol independently of the actual value of the parameter.

- Data structures. Algorithms are routinely defined to operate on variables whose values range over an infinite domain, such as counters, stacks, channels, etc. Similarly, when describing timing aspects of a system, one naturally uses real-timed clocks.

To handle infinite systems, two methodologies have been introduced:

- Abstraction. In abstraction, one is given an abstraction function which maps concrete states to a smaller set of abstract states. The aim is to construct an abstract image of the concrete system, and perform verification on this hopefully tractable image (see e.g. [LBB001, CGL94]). But for the verification to be accurate, one needs to find a compromise between keeping and throwing away information. If too much information is kept, then the verification will take too much time and space. If too much information is discarded, then the abstract system will have a behavior inconsistent with the concrete system. In model checking, one has to make the choice of the abstraction function automatic. This task is very difficult.

- Extending the symbolic approach. One tries to deal with infinite sets of states rather than finite sets. The main challenge is to invent an appropriate (finite) representation for each new class of systems that are to be analyzed.

Our contributions range over the following topics:

- Parameterized systems. With standard model checking methods, one can only verify a protocol for a particular configuration of the protocol entities. Therefore, for every instance of the protocol, one needs to repeat the verification process. Using the framework of regular languages and transducers, we model and reason about parameterized versions of protocols. Given the fact that the transitive closure of a regular transducer is not in general regular, any algorithm that purports to compute closures of transducers will necessarily be incomplete. The challenge is to develop methods that will work in as many cases as possible.
• Infinite games. Very close to the area of verification is the problem of \textit{controller synthesis}. In that area, one tries to model interactions of a controlling system with its environment. The question asked is, given a number of properties that the whole system should satisfy, \textit{synthesize} a controller that will preserve these properties, irrespective of what happens in the environment. A very useful paradigm for this problem is that of \textit{games}. The question of synthesis is expressed in terms of finding a \textit{winning strategy} for an appropriate \textit{winning condition}.

• Infinite probabilistic systems. The semantics of languages based on transition systems usually includes \textit{non-determinism}, i.e. the notion that several executions of the system are possible. One can express for example the fact that a message in a communication protocol may be lost. However, non-deterministic behavior doesn’t allow differentiating between possible executions. In the example of message loss, one may wish to make a difference between losing 1 message, and losing 1000 messages in a row. A natural framework is that of \textit{probabilistic systems}, in which each transition is assigned a probability. We represent such systems with \textit{Markov chains}.

In the following sections, we present an overview of our contributions.

1.2 Parameterized Systems

We often encounter systems in which a relatively simple component (a sort of “building block”) is repeated an arbitrary number of times to form a complex system. Such systems are called \textit{parameterized}, in the sense that their definition uses the number of entities as a parameter.

Protocols for the control of a shared resource are typical examples of parameterized systems: for instance \textit{mutual exclusion} protocols. In these protocols, several identical participants compete for access to a shared resource (e.g. the right to transmit on a broadcast medium).

For such protocols, it is important that the system can be \textit{proven} correct regardless of the configuration. This kind of reasoning is called \textit{parametric}. We underline the fact that because the system can have any size, it is not possible to do an enumerative exhaustive search of the state-space of a parameterized system: it is infinite. We need a symbolic approach to deal with such systems.

The framework of \textit{Regular Model Checking} provides for both the modelling of parameterized systems and parametric reasoning. Within this
1.2. PARAMETERIZED SYSTEMS

framework, systems are modelled using regular sets to represent sets of configurations, and length-preserving regular transducers to represent transition relations. The main idea is to encode the state of a system with \( n \) cells or processes by a word of length \( n \): each position inside the word indicates the state of a distinct process. Note that in spite of the length-preserving restriction on transducers, systems using a dynamic topology (processes can die or be born at run-time) can be simulated by using an additional symbol to represent an empty position.

One important operation in regular model checking is the so-called transitive closure, which characterizes the set of all configurations that are related through one or more iterations of the transition relation. Once we have the closure, we can solve any reachability question simply by computing the image, under the closure, of a given set of initial states (safety questions are reduced to reachability). The transitive closure also makes it possible to solve repeated reachability questions.

Since the transitive closure of a regular relation may in general not even be regular, any algorithm for computing closures will necessarily be incomplete. We try instead to propose methods that work as often as possible.

The approach we take is that of acceleration, i.e. try to compute directly the effect of an arbitrary number of applications of a set or a sequence of transitions of a given system.

While it is straightforward to mathematically characterize an infinite automaton which represents exactly the transitive closure of a regular relation (we call it the column transducer, more details follow below), such an automaton is of no use for algorithmic purposes. The challenge is therefore to “compress” this automaton into a finite one. To this end, we devise an equivalence relation on the states of the column transducer. All states that are found equivalent are subsequently merged into one. The difficulty lies in finding an equivalence relation which is as coarse as possible, while guaranteeing that the quotient automaton will still represent the same relation as the hypothetical column transducer.

1.2.1 Running Example

As a support for the ideas expressed here, we use a simple toy example: the simple token passing protocol.

As illustrated in Figure 1.1, the protocol has an arbitrary number of identical processes placed in a row. Each of these processes can be either in idle mode, or in critical mode, depending on whether or not they own the (unique) token (represented as a grey bullet in Figure 1.1). Two neighboring
processes can communicate with each other as follows: a process owning the token can give it to its right-hand side neighbor.

In the framework of regular model checking, we represent a global state of this system using a word over the alphabet \(\Sigma = \{N, T\}\). An instance of the protocol with \(n\) processes will have global states represented as words of length \(n\). The symbol at position \(i\) in the word represents the state of process number \(i\). A process in the idle mode will be written \(N\), while a process in critical mode, i.e., owning the token, will be written \(T\).

The transducer for this example is shown in Figure 1.2. The action of moving the token one position to the right is represented by the two neighbor transitions \((T, N)\) and \((N, T)\), that is the token disappears from the left-hand side process, and reappears on the right-hand side process. The self-loops labelled with \((N, N)\) allow for the presence of an arbitrary number of unaffected idle processes on either side of the two active processes.

**1.2.2 Transitive Closure and Column Transducer**

The transitive closure captures arbitrary compositions of the transducer. Applied to the simple token passing example, this means moving the token to the right any number of times. The corresponding transducer is shown in Figure 1.3. Its effect is to shift the token one or more positions to the right, as expected.

**Column Transducer**

Given a transducer \(T\), the so-called *column transducer* is a characterization of the transitive closure \(T^+\) of \(T\) obtained by building the infinite union of
1.2. PARAMETERIZED SYSTEMS

\begin{figure}[ht]
\centering
\includegraphics[width=0.5\textwidth]{figure13.png}
\caption{The Closure for the Simple Token Passing example}
\end{figure}

\begin{figure}[ht]
\centering
\includegraphics[width=0.5\textwidth]{figure14.png}
\caption{Simple Token Passing – Partial Column Transducer}
\end{figure}

all the finite compositions of $T$:

$$T^+ = \bigcup_{i \geq 1} T^i$$

Intuitively, each $T^i$ is a separate part of the column transducer. $T^i$ is built as follows: if $T$ has states in the set $Q$, then the states of $T^i$ are words in $Q^+$, of length $i$. Two states $q_1q_2 \cdots q_i$ and $q'_1q'_2 \cdots q'_i$ are related by a transition $\frac{(a,b)}{a}$ if and only if there exist symbols $a_0,a_1,\ldots,a_i$ with $a_0 = a$ and $a_i = b$ such that for each $p : 0 \leq p < i$, states $q_p$ and $q'_p$ are related (in $T$) by the transition $\frac{(a_p,a_{p+1})}{a_p}$. Initial states (respectively final states) of $T^+$ are non-empty words over the set of initial (resp. final) states of $T$.

Figure 1.4 shows part ($T \cup T^2 \cup T^3$) of the column transducer obtained for the token-passing example.
1.2.3 Methods of Construction for the Transitive Closure

Each of the following methods are based on manipulations on the column transducer.

The subset construction

We determinize the column transducer, using the classical subset construction of finite automata theory.

Although each state in the deterministic transducer may merge an infinite number of states in the column transducer, this method alone fails to recognize that distinct states constructed may potentially have the same set of suffixes (they correspond to unrolling the transitive closure: \( T^+ = T^+ \cup T \circ T^+ \)), and should thus be merged.

A way to avoid this unrolling is presented in [BJNT00]: saturation. The saturation method merges more states of the column transducer based on the observation that parts of transducer \( T \) are "inactive" (the relation these parts represent is the identity), and thus these parts can be applied any number of times without changing the overall relation.

Our contribution is to extend the method from the context of words to that of trees. This allows for our framework to handle systems with not only linear but tree topologies.

The main difficulty in this task is that working with tree automata breaks the symmetry that could be found in many concepts related to word automata, in particular prefixes and suffixes. This means that even though such concepts naturally have their tree counterparts, definitions and proofs become more involved.

The matching method

The "subset construction" above is elegant, especially since all operations can be performed symbolically. However, it turns out that even for small examples, the method is computationally costly. We tackle the issue by avoiding determinizing the column transducer, but construct it in small, elementary steps, and keep it non-deterministic. To combat infiniteness and increase efficiency, we try to make the automaton under construction small, through a merging strategy.

In the transducer \( T \), we identify some states that denote "inactive" parts (in the same sense as above). We notice that such states can be repeated in a column without changing the forward or backward language of such a column. This gives rise to a simple equivalence relation on columns that can be checked syntactically and on-the-fly on the label of the column. We
present a very simple closure construction technique using this equivalence.

A coarser equivalence for the matching method
While the “matching method” above allows for a very simple and readable algorithm, in practice the implementation suffers from explosions in the size of the produced automaton. Indeed, the equivalence relation is too fine-grained, and the automaton under construction includes a lot of redundant states. Therefore, there is a need for coarser equivalences if we want the method to perform reasonably.

We introduce a general framework, based on simulations, to find adequate equivalences on columns. We try to construct a relation as large as possible. The equivalence of the “matching method” above is just a particular case (in fact, it is the finest-grained equivalence in our framework).

1.3 Infinite Games

Model checking is very closely related to the area of control and so-called reactive systems. In this domain, behaviors can naturally be described as games, played between the environment on one hand, and the reacting control system on the other hand. Control problems are then reduced to finding winning strategies for corresponding game problems.

We consider game structures in which 2 players, called $A$ and $B$ respectively, make moves in turn. This is illustrated in Figure 1.5, where player $A$ has square states, and player $B$ circular states. A sequence of moves of a game is called a play.

A play is defined by giving:

- a starting state;

- a strategy for player $A$;
• a strategy for player $B$.

In this context, a strategy is a mapping that maps each state to a transition outgoing from that state. Notice that such a strategy only needs to consider what the current state of the game is, and not the history of moves that lead to it (it is called memoryless, or positional).

We study two kinds of winning conditions for games: namely safety, and parity:

• In the safety problem, we are given an initial state, as well as a set of final ("bad") states. Player $A$ tries to avoid the set of final states, while player $B$ tries to force the game towards the final states.

• In the parity problem, each state in the game is equipped with a natural number called rank. Player $A$ wins if the smallest rank ever encountered during the play is even, and $B$ wins otherwise.

Having chosen a winning condition, the question is whether there exists a strategy for player $A$ which ensures victory, irrespective of $B$'s strategy.

While games on finite structures are well studied, the nature of reactive systems often induces structures that are infinite-state. Therefore, there is a need to study games over infinite structures.

We recall that for classical model checking, verification of safety properties is decidable whenever the set of bad states is upward closed with respect to a well quasi-ordering, and the system to analyze has a monotonic transition relation with respect to this same ordering. In [ACJYK00], the authors present a generic backward reachability algorithm that uses minimal (w.r.t. the well quasi-ordering) elements as a finite and canonical representation for infinite sets of states (upward-closed sets). Given a set of final states which is upward-closed, the authors show that monotonicity of the transition relation means that successive pre-images will also be upward-closed. Since well quasi-ordering can be extended to upward-closed sets, the authors conclude that termination of the backward reachability algorithm is ensured.

We investigate lifting this concept of monotonicity to games. It turns out that monotonicity with respect to a well quasi-ordering is not enough to ensure decidability for even the simplest kind of game problem: safety.

To prove this, we use a particular instance of games that are monotonic (with respect to the usual ordering on vectors): VASS (Vector Addition System with States — equivalent to Petri nets) games. With a VASS game with 2 counters, it is possible to simulate a two-counter machine. This is illustrated in Figure 1.6 and Figure 1.7. Since a counter machine has more operations than a VASS, we simulate these operations as follows:
1.3. INFINITE GAMES

- Increment and decrement operations of counter machines have a direct counterpart in VASSs. One player chooses the move of the counter machine, and the other player (the “opponent”) follows passively.

- The test for zero has no corresponding operation in VASSs, which can only test for a value different from zero. This means that we can let the opponent detect that the other player “cheated”, and let him win directly in such a case.

Undecidability of reachability for 2-counter machines implies undecidability of safety for monotonic games.

We further investigate decidability for a subclass of monotonic games: downward-closed games. Downward-closedness makes the safety problem decidable. However, since the two players are given an asymmetric role, we distinguish 3 cases, depending on the behavior of the players:

- When player $B$ only has a downward-closed behavior, we analyze the game with a form of backward reachability algorithm: we characterize all states from which player $B$ can force the game into the “bad” states. We show that the set of all such states is upward-closed. As in the case of backward reachability for model checking, well quasi-ordering ensures termination.

- When player $A$ only has a downward-closed behavior, we analyze the system with a reachability tree construction which is a generalization of
the “eventuality property” construction of [ČJYK00]. Termination follows from well quasi-ordering and König’s lemma.

- If both players have a downward-closed behavior, then either of the methods above will work.

We also study the problem of parity. We show that decidability is only ensured in case both players have a downward-closed behavior.

1.4 Infinite Probabilistic Systems

When modelling a system, one often needs to express the fact that the system may sometimes evolve in different ways, depending on some interaction with the environment that cannot be predicted by the system itself. The simplest solution in such a case is to model the system’s behaviour as being non-deterministic.

While non-determinism can be used to reason about all possible behaviours, it fails to capture the fact that all possibilities may not be equal in terms of probability. An illustration can be found in communication protocols over an unreliable medium: a single message may be lost with a high probability, but the probability of losing a large number of messages in a row is smaller.

A useful framework is that of Probabilistic Systems, which are transition systems augmented as follows: with each transition of the system, we associate a probability. The mathematical model we use for this is that of Markov chains. We can introduce probabilistic behavior in non-probabilistic models as follows: with each transition in the system, we associate a weight (a positive natural number). The relative weight of a transition with respect to all other possible transitions (leaving the current state) gives the probability to take this transition. Figure 1.8 shows an example of Markov Chain and the associated non-probabilistic transition system.

Much work has been done in the context of finite-state probabilistic systems: on-the-fly algorithms exist to verify properties expressed in logics such as CTL (see e.g., [CY95, Var99]). On the other hand, verification of infinite-state probabilistic systems is more delicate to deal with. Existing approaches to tackle infinite-state probabilistic systems are often dedicated to restricted classes of systems, in particular Probabilistic Lossy Channel Systems (see e.g., [AR03, IN97]). These methods mostly rely on the existence of a so-called finite attractor (for example states with empty channels) to prove decidability. An attractor, in this context, is a set of states that is
always eventually reached from any state. Finiteness of the attractor then allows a finite analysis.

Instead, we propose a general method which does not use the concept of attractor, but is based on a simple search algorithm. We are interested in reachability properties. While the computation of exact probabilities is not possible, we propose a scheme that gives an approximation within a certain error range, and we identify sufficient conditions under which such a quantitative reachability analysis can always be performed.

The scheme is a modified version of the algorithm presented in [IN97]. This is essentially a breadth-first search algorithm. Starting from some initial state, we look for reachable states for which we can say whether they belong to a set of final states (denoted $F$), or whether that set is not reachable at all (denoted $\bar{F}$). Let us call these “good” states. Whenever such a state is encountered, we update the value of two variables which give an under-approximation of the probability of reaching a final state ($\text{Yes}$) versus the probability of not reaching a final state ($\text{No}$). The idea is that the deeper we go in the search tree, the more precision we get.

When analyzing finite-state systems, we are guaranteed to eventually either reach $F$ or $\bar{F}$ during the reachability analysis. In our case, the difficulty comes from the non-finiteness of the set of “open” states, i.e. states that are neither in $F$ nor in $\bar{F}$. It may be the case that some paths remain forever in the open states during analysis. The measure of such paths need not be zero. Because of that, there is in general no guarantee that we will reach the required precision by searching a finite part of the reachability tree. Therefore, we need a property ensuring that we almost always (in the probabilistic sense: with probability 1) reach either $F$ or $\bar{F}$. When that property is satisfied, we can be sure the the sum $\text{Yes} + \text{No}$ gets close to 1.

\begin{figure}[h]
\centering
\begin{subfigure}{0.45\textwidth}
\centering
\begin{tikzpicture}
\node[state] (A) at (0,0) {1};
\node[state] (B) at (1,1) {0.6};
\node[state] (C) at (0,-1) {0.4};
\node[state] (D) at (1,-0.5) {1};

\path[->] (A) edge (B);
\path[->] (A) edge (C);
\path[->] (B) edge (D);
\path[->] (C) edge (D);
\end{tikzpicture}
\caption{A Simple Markov Chain}
\end{subfigure}
\begin{subfigure}{0.45\textwidth}
\centering
\begin{tikzpicture}
\node[state] (E) at (0,0) {1};
\node[state] (F) at (1,1) {0.6};
\node[state] (G) at (0,-1) {0.4};
\node[state] (H) at (1,-0.5) {1};

\path[->] (E) edge (F);
\path[->] (E) edge (G);
\path[->] (F) edge (H);
\path[->] (G) edge (H);
\end{tikzpicture}
\caption{The Corresponding Transition System}
\end{subfigure}
\caption{Markov Chains and Transition Systems}
\end{figure}
when we go deep in the reachability tree. This ensures that we can meet the precision requirement.

Examples of systems for which the scheme is guaranteed to work are probabilistic VASS (Vector Addition Systems with States), and probabilistic LCS (Lossy Channel Systems).

1.5 Papers and Contributions

**Paper A:** Our contribution is to extend the “subset construction” to structure-preserving regular tree transducers.

**Paper B:** Based on the remark that the image operation of the “subset construction” is expensive, and that the sets of columns obtained are quite simple, we propose a simple equivalence relation that can be used to collapse the column transducer on-the-fly.

**Paper C:** While the “matching method” above is elegant and simple, the equivalence used gives rise to a lot of redundant states in the computed transducer, because it is too fine-grained. We propose a framework based on simulations to create coarser equivalences with which we can safely collapse the column transducer.

**Paper D:** We extend the notion of monotonicity with respect to a well quasi-ordering from transition systems to infinite game structures. Our contribution is to give decidability results for such infinite games.

**Paper E:** We propose a method to perform quantitative reachability analysis of infinite-state Markov chains. Our contribution is to identify sufficient conditions under which the proposed algorithm is guaranteed to terminate.

1.6 Conclusions and Future Work

In the domain of Regular Model Checking, we have made progress in how we compute the transitive closure of a given transducer, with the aim to perform reachability analysis, and thus check safety properties. However, very little has been done when it comes to more advanced properties such as liveness. In [Nil00], the proposed approach is to analyze reachability, or repeated reachability on an automaton obtained by taking the product between the original transducer and a Büchi automaton representing the
property to be checked. This has never been tried beyond hand-coded toy examples, and should be studied in future work. Also, while we have been able to extend the “subset construction” of regular model checking from the context of linear structures to tree structures, the resulting algorithm suffers from the same kind of complexity as the word counterpart. To reduce complexity, we would like to extend the framework of Paper C to trees as well. Finally, many protocols work on topologies that are even more general than trees. A logical step would be to consider general graphs instead of trees.

In the domain of games, we have shown that monotonic games are in general undecidable when subject to safety winning conditions. We introduced a subclass, namely downward-closed games, for which safety becomes decidable. As for parity winning conditions, we find that downward-closed games are in general undecidable. While we have shown that VASS (or Petri nets) games were undecidable, it was shown in [BEM97] that pushdown games are in fact decidable. Thus, it is of interest for further work to investigate subclasses of Petri nets such as Basic Parallel Processes (BPPs), or even combinations of these systems. It would give an insight into systems exhibiting notions like recursion and multithreading.

In the domain of Probabilistic Systems, we have shown how to perform Quantitative Analysis of safety properties. While our approach is more general than that of [Rab03], in the sense that it can deal with classes of systems other than PLCSs, Rabinovich also studies liveness properties. It would be interesting to investigate whether our method can also be extended to deal with such properties.

References


REFERENCES


REFERENCES


