Selected Problems from the Area of Formal Verification

Author: Martin Kot

Supervisor: prof. Petr Jančar

PhD Thesis

Faculty of Electrical Engineering and Computer Science

Technical University of Ostrava

2009

Acknowledgments

I would like to thank my supervisor Petr Jančar for guidance, comments, fruitful discussions, collaboration on results and papers and his patience with me.

I would also like to thank Zdeněk Sawa, who was co-author of several of my papers and a bit like second supervisor for me.

I would also like to thank Jindřich Černohorský that came with the idea of using verification tool on real-time database systems and Václav Król and Jan Pokorný for acquainting me with database system V4DB.

Last but not least I would like to thank my family for their support.

Since 2005, I was supported by the Czech Ministry of Education, Grant No. 1M0567.

Declaration

I declare that this thesis was composed by myself, and all presented results are my own, unless otherwise stated.

Some of the material has been previously published in [17], [26], [22], [23], [19], [24], [25] and [18]. Publications with co-authors were really joint work, we were discussing and writing all parts together and each of authors has done approximately the same amount of work.

Martin Kot

Annotation

The thesis presents results obtained by the author in the area of verification of systems. One part of the thesis concentrates on questions of complexity of equivalence checking, , i.e., of deciding behavioral equivalences on transition systems. The other part concentrates on practical use of model checking on real time database systems. Model checking means deciding validity of temporal logic formulae which express properties of a system.

In the first part, several results on deciding bisimulation equivalence are shown. All these results concern with so called Basic Parallel Processes (BPP). The first of presented algorithms decides bisimulation equivalence between a BPP and a finite-state system. There is also presented time complexity analysis of this algorithm which shows up an upper bound $O(n^4)$. The second algorithm decides bisimulation equivalence between two normed BPPs in $O(n^3)$. The third algorithm decides for a given BPP whether there exists some equivalent finite-state system with respect to bisimulation equivalence. This problem is called Regularity of BPP. Presented algorithm works in polynomial space which, together with previously known PSPACE-hardness of regularity of BPP, gives PSPACE-completeness of this problem. The last presented equivalence checking algorithm decides bisimulation equivalence between a normed BPP and a normed BPA system. This algorithm is polynomial, its detailed analysis leads to an upper bound $O(n^7)$.

The second part of the thesis shows some possibilities how verification tool Uppaal can be used on modeling and verification of real-time database systems. Presented models are focused on concurrency control used in databases to avoid inconsistency when several transactions can be executed in parallel. There are models of several well known variants of pessimistic and optimistic protocols presented and some simple demanded properties of those protocols expressed as temporal logic formulae are checked on the models.

Anotace

Tato disertační práce prezentuje výsledky dosažené autorem v oblasti verifikace systémů. Jedna část práce se zaměřuje na otázky složitosti problémů ověřování ekvivalencí (equivalence checking), tzn. rozhodování behaviorálních ekvivalencí na přechodových systémech. Druhá část práce se zabývá praktickým využitím ověřování modelů (model checking) na real-time databázových systémech. Ověřování modelů znamená rozhodování platnosti formulí v nějaké temporální logice popisujících nějakou vlastnost systému.

V první části je uvedeno několik výsledků týkajících se rozhodování bisimulační ekvivalence. Všechny tyto výsledky se týkají takzvaných základních paralelních procesů (Basic Parallel Processes – BPP). První prezentovaný algoritmus pracující v čase $O(n^4)$ rozhoduje bisimulační ekvivalenci mezi BPP a konečně stavovým systémem. Druhý algoritmus rozhoduje bisimulační ekvivalenci mezi dvěma normovanými BPP v čase $O(n^3)$. Třetí algoritmus rozhoduje v polynomiálním prostoru pro BPP, jestli existuje nějaký bisimulačně ekvivalentní konečně stavový systém. Tento problém se nazývá regularita BPP a spolu s dříve známou PSPACE-obtížností dostáváme jeho PSPACE-úplnost. Poslední prezentovaný algoritmus z oblasti ověřování ekvivalencí rozhoduje bisimulační ekvivalenci mezi normovaným BPP a normovaným BPA systémem. Tento algoritmus je polynomiální, jeho podrobnější analýza vede k odhadu $O(n^7)$.

Druhá část práce ukazuje nějaké možnosti, jak může být použit verifikační nástroj Uppaal pro modelování a verifikaci real-time databázových systémů. Prezentované modely se zaměřují na řízení souběžného přístupu k datům, které je v databázích používáno pro zabránění nekonzistence v případě paralelního zpracovávání více transakcí současně. Jsou uvedeny modely několika (známých) variant pesimistických a optimistických protokolů a následně je na těchto modelech ověřeno několik jednoduchých, po protokolech vyžadovaných, vlastností vyjádřených ve formě formulí temporální logiky.

Contents

Chapter 1

Introduction

Recently, software and hardware systems are ubiquitous, complicated, and extensive and every bug can have serious and expensive consequences. In traffic control systems, nuclear power plant control system, etc. an error can cost many lives. Other examples of systems where we would like to avoid errors are operating systems, network communication protocols, microprocessors and other chips, automotive systems and many others. Much effort is for that reason devoted to ensure correctness of such systems. Correctness usually means that the *implementation* of some system behaves exactly as it is described in some *specification* of a desired behavior. A process of checking whether the given implementation satisfies the given specification is called *verification*.

Widely used techniques for verification are testing and simulation. Testing means that a system runs with selected inputs and a behavior of it is observed. There are many possibilities how inputs for testing can be chosen, e.g. random values, all possible values, some boundary values etc. When simulation is provided, some model of system is tested instead of implementation itself. Inputs for simulation can be chosen similarly as for testing. A merit of a simulation is that running model of a system can be easier, cheaper and often in the same time interval more test values can be processed than by testing. A drawback of the simulation is that a real system (implementation) can contain more errors than its model and this errors can not be detected in this way.

Both, testing and simulation, can be used in all stages of system development and are able to discover many possible errors and bugs in the system. They are very effective in the early stages of development because the design is usually infested with multiple bugs. But their effectiveness drops quickly as the design becomes cleaner. Then they need very large amount of time to uncover the more subtle bugs. Their common important drawback is that they usually can not ensure correctness under all possible situations. The number of inputs, possible interactions with environment etc. is usually so big (and often even infinite) that we are not able to try them all during tests or simulations. This problem become even worse for systems composed of several components running in parallel which is nowadays, with the support of multicore CPUs, fast nets between computers and other common means, very frequent. Interaction between those concurrently running components causes that the behavior of a system as a whole can be non-deterministic. The number of possible behaviors grows very fast with every added component and it can be difficult even to reproduce bugs in these systems since they can occur only under some rare circumstances.

The alternative to testing and simulation are *formal methods* or *formal verification* that conducts an exhaustive exploration of *all* possible behaviors to ensure correctness.

Formal methods provide some theoretical means for a construction of a rigorous mathematical proof of the correctness of the system. This can be done by hand, which is very laborious and error prone, or done with help of some software tools. The latter approach is called *computer aided verification* and is usually more effective. A problem is that the process can not be fully automated in general because many problems concerning behavior of computer programs are undecidable. For example even such simple question whether a program will eventually stop or not is well known undecidable problem (called Halting problem).

In general, there are three main approaches to verification that allow to ensure correctness for all possible behaviors of the system:

- Theorem proving
- Equivalence checking
- Model checking

Theorem proving is based on a construction of formal proof of correctness of a system. This can be assisted by software tools called *theorem provers.* This tools demand a guidance of their user to do the crucial steps of the proofs. A theorem prover helps with some simple, but laborious, steps, but the main responsibility to lead the proof is on its user. This requires a lot of knowledge, skills and practice from the user.

Model checking and equivalence checking can be fully automated and do not require much interaction from the user. But this methods can not be applied to arbitrary systems due to the undecidability of checked properties. Therefore, usually not computer programs are checked but instead models in some formalism that does not have the full expressive power of Turing machines are verified. This techniques are based on automata and formal languages theory because this theory offers means for finite description of infinite languages and many properties of languages are decidable.

In the case of *equivalence checking* the question is whether two (descriptions of) systems are equivalent in some sense. Usually, it is compared whether a specification and an implementation have the same (or equivalent) behavior.

In the case of *model checking* we have only one (description of) a system and some desired property of it expressed as a formula of some (temporal) logic. The goal of model checking tools and algorithms is to check whether the system satisfies the given property. Equivalence checking is often used in practise for verification of hardware circuits and chips and model checking is more common in software verification. See [9, 11, 40, 4] for more information about model checking and temporal logic.

Systems for model checking and equivalence checking can be expressed in many different ways and formalisms. Models with a great expressive power cannot be verified automatically, and models that are too restrictive do not allow to model many aspects of real systems. As well properties for model checking can be expressed in the variety of logics and there are many possible equivalences suitable for equivalence checking. Combinations of this possibilities form a big amount of verification problems. The research concentrates on decidability and complexity of those problems. One active area of research concentrates on the question which model checking and equivalence checking problems are decidable, and where exactly lies the dividing line between decidable and undecidable problems. Another important question is what is the exact computational complexity of decidable verification problems. Some verification problems can be solved theoretically but (known or even all) algorithms can be used in practice only for small instances due to its computational complexity. One well known phenomenon which makes design of verification algorithms harder is so called ‘state space explosion’. This problem appears when several components with reasonably small state spaces compose a system which as whole can have the state space exponentially larger then its components. This problem shows up unavoidable in some cases but there are also lots of techniques that avoid it in other cases or at least deal with it to some degree.

In this thesis we concentrate on two, quite different, types of problems. In the first part we will concentrate on complexity of some equivalence checking problems, and some results obtained and published by the author in this area will be presented there. In the second part we will concern on model checking. There is a much research effort devoted to the area of real-time database systems but almost none to verification of designed systems and suggested algorithms and protocols. We will describe some possibilities of using existing verification (model checking) tool Uppaal on such type of system.

Some of the results presented here are joint work with other authors – Petr Jančar and Zdeněk Sawa.

It is assumed that the reader is familiar with formal languages, basics of mathematical logic and of complexity theory.

1.1 Goals of the Thesis

There are two main goals of this thesis. The first is to contribute with some new results in the area of equivalence checking. Questions of algorithmic decidability and computational complexity of deciding different equivalences on different types of systems are the point of interest of quite many researchers. But there are still many open problems, many upper bounds on complexity are immoderate. The goal of this thesis is to contribute with some new results in this area. We mainly focused on computational complexity of selected equivalence-checking problems involving so called Basic Parallel Processes.

The second goal is a bit like case study of using verification tool Uppaal on concurrency control protocols used in real-time database systems. There are only rare attempts to use formal verification methods on real-time databases. Existing verification tools does not have direct support for database systems. Regardless of it, the goal is to show that verification tools can be used and verify some demanded properties of database systems. The part of those system controlling concurrent access to data is one of the most important. Hence, concurrency control protocols were chosen for illustration. Uppaal was selected from all available verification tool because of its support for real-time systems.

Most people dealing with real-time databases (researchers and practical users) do not have experience with verification tool and do not know possibilities of this tools. Therefore, they were not able to identify properties that were interesting for them and that are manageable to verify using a verification tool (questions as "Which protocol is better" or "How much transactions will be restarted" are in the author's opinion not suitable for verification). Hence the goal is to examine possibilities of modeling of different versions of protocol for concurrency control and identify possible abstractions and simplifications so that the models were manageable by verification algorithms of the tool. This can help authors of new protocols to verify them. And mainly, our models and simple verified properties can later serve for illustration of possibilities of verification tools to database system users and researchers in order to identify really interesting questions for verification.

1.2 Overview of the Results

The following subsections describe shortly the main results presented in this thesis.

1.2.1 Selected Equivalence Checking Problems

Systems for equivalence checking can be expressed in many different ways. Some possible formalisms used for description of verified systems were organized into so called (α, β)-PRS hierarchy ([34], see Subsection 2.2.4 for more informations). We are especially interested in three simplest classes from the hierarchy - finite state systems, basic parallel processes and basic process algebra.

We can understand those classes intuitively as follows (see Section 2.2 for exact definitions). Finite state systems (FS), as the name suggests, are systems with (explicitly or implicitly) given finite number of states. The state of the system can be changed using one of defined transitions. Basic process algebra (BPA) can model simple sequential systems with procedure calls. The state of BPA system is given as a content of a stack and the behavior is given by a set of rules which describe how the symbol on top of the stack can be changed. Basic parallel processes (BPP) can model simple parallelism without communication between parallel components. A state of BPP is a multiset of symbols and a behavior is given by a set of rules which describe how an element from the multiset can be changed for some other elements. In this thesis we will deal with such systems only, where every change of a state is accompanied by an externally observable action from some finite set of possible actions.

Elements of all three mentioned classes can be normed. This means that from every state there is a special 'empty' state reachable where no further action is possible.

There are also many possible equivalences suitable for model checking. We concern with one of most well known and important equivalence - bisimilarity or bisimulation equivalence. It can be understand as follows – whenever one of two equivalent systems can change a state using a rule with some assigned action the other system must be able to change its state using a rule with the same action and resulting states of both systems have to be equivalent again.

In chapter 2 we show some results concerning BPP, BPA and FS and bisimulation equivalence. In the Section 2.3, an algorithm which decides bisimilarity between BPP and FS systems is described. This algorithm works in time $O(n^4)$. This result was published in [26]. The next Section 2.4 contains an algorithm deciding bisimilarity between two normed BPP systems. A working time of this algorithm is $O(n^3)$. This was published in [17]. Both mentioned algorithms are based on the technique of DD-functions introduced in [16] and are variations on the algorithm presented there for deciding bisimilarity between two BPP systems.

Third result in this area (in Section 2.5, published in [22]) is a polynomial space algorithm deciding for a BPP system whether there is some bisimilar finite-state system. This problem, called regularity of BPP, was known to be PSPACE-hard hence we obtained PSPACE-completeness.

The last result in the Section 2.6 is an algorithm deciding bisimilarity between a normed BPA system and a normed BPP system. An exponential algorithm was known for this problem and we have shown in [19, 18] a polynomial time algorithm. Exact analysis of the complexity of this algorithm leads to an upper bound $O(n^7)$. There are also several side results: an algorithm transforming normed BPP into a special form where bisimilarity coincides with identity (in time $O(n^3)$); an algorithm deciding for a normed BPP whether there exists some bisimilar BPA (in time $O(n^3)$); an algorithm deciding bisimilarity between a normed BPA system and a finite-state system (in time $O(n^4)$).

1.2.2 Model Checking of Real-Time Database Systems

Real-time database systems are based on well known techniques and algorithms from conventional database systems. In addition to efficient data storage, database query maintenance etc. they are able to provide some bounds on response time which is very important in real-time systems. To this aim they use priorities, deadlines and other mechanisms.

Concurrency control is one of the most important parts of all database systems that allow concurrent access of several transactions to data. In real-time databases, modifications of concurrency control protocols known from conventional databases are used. The modifications are needed to deal with priorities and deadlines.

In chapter 3 we try to use model checking to verify some important properties of real-time concurrency control protocols. We use a software verification tool to this aim. A protocol is modeled as a net of timed automata (a modeling language of Uppaal) and demanded properties expressed as temporal logic formulae are then automatically checked on those models. The main goal was to show possibilities of using an existing verification tool on type of systems (i.e. databases) that it was not mentioned to be used on. Therefore, the main accent is laid on models of protocols and verified properties are more or less for illustration.

We suggest models of concurrency control from two views. In the Section 3.5 it is as a part of a whole database system and in the Section 3.6 some selected protocols itself are modeled. Presented models are not the only possible or the best in any way. But they can give some ideas and possible abstractions to help with modeling of parts of real-time databases when some newly suggested protocol or algorithm will need to be verified.

Chapter 2

Problems Related to Bisimilarity on Basic Parallel Processes

2.1 Introduction

In this chapter we will talk about problems related to bisimilarity on Basic Parallel Processes. As we stated previously, BPP are one class from (α, β)-PRS hierarchy. The bisimilarity on classes from this hierarchy is one of often studied problems in the area of equivalence checking. Some nice techniques and algorithms for equivalence checking are described in [6], recently updated overview of already known bisimilarity results is in [39].

Our research in this area was motivated by [16] where Jančar presented an algorithm deciding bisimilarity of Basic Parallel Processes in polynomial space. With previously published PSPACE-hardness result ([37]) it gives PSPACE-completeness of this problem.

This problem can be formulated as follows (for definitions of used notions see next section):

Problem: BPP-BISIM

INSTANCE: BPP processes (M_1, Δ_1) and (M_2, Δ_2).

QUESTION: Is $M_1 \sim M_2$ (in the disjoint union of Δ_1 and Δ_2)?

Jančar in [16] introduced a notion of so called DD-functions and this idea was suitable for solution of some subproblems of bisimilarity on Basic Parallel process (bisimilarity of BPP process with finite state process described in Section 2.3 and bisimilarity of two normed BPP processes described in Section 2.4) and for some related problems (regularity of BPP in Section 2.5 and bisimilarity of a normed BPP and a normed Basic Process Algebra in Section 2.6).

2.2 Basic Notions and Definitions

This section contains some basic definitions that are used in the remaining sections of this chapter.

2.2.1 Notation Conventions

At first we need to make clear some mathematical notation. $\mathbb{N}$ denotes the set of natural numbers $\{0, 1, 2, \dots\}$. ω is used as a symbol for infinity. We stipulate that for each $x \in \mathbb{N}$, $x < \omega$, $\omega + x = x + \omega = \omega + \omega = \omega - \omega = -\omega + \omega = \omega$, $\omega \cdot 0 = 0 \cdot \omega = 0$, and for each $x \geq 1$, $\omega \cdot x = x \cdot \omega = \omega$.

Given a set X, by $\mathcal{P}(X)$ we denote the set of all subsets of X.

We use $|X|$ to denote the cardinality of a set X. We stipulate $\min \emptyset = \omega$, using ω as a symbol for infinite amount;

X^+ denotes the set of nonempty sequences of elements of X, and $X^* = X^+ \cup \{\varepsilon\}$ where ε is the empty sequence. The length of a sequence $\alpha \in X^*$ is denoted by $|\alpha|$ ($|\varepsilon| = 0$). We use x^k (where $x \in X^*$, $k \in \mathbb{N}$) to denote the sequence $xx \cdots x$ where x is repeated k times (in particular $x^0 = \varepsilon$). Let $\alpha \in X^*$ be a sequence and $x \in X$. Then $|\alpha|_x$ denotes the number of occurrences of element x in α.

2.2.2 Labelled Transition Systems

There are many possibilities how to model or describe systems for verification purposes, for example different types of automata, grammars, Petri nets, process rewriting systems, process algebras etc. Labelled transition systems (LTS) are a concept which is underlying all of those formalisms.

An LTS is formally defined as a triple $(S, \mathcal{A}, \longrightarrow)$ where:

- S is a (possibly infinite) set of states
- $\mathcal{A}$ is a finite set of actions (or action labels)
- $\longrightarrow \subseteq S \times \mathcal{A} \times S$ is a transition relation.

.

Less formally, S is the set of all possible states of the system. $\mathcal{A}$ is a set of externally observable actions (or their names) which can be performed by the system. Transition relation represents behavior of the system.

Instead of $(s, a, s') \in \longrightarrow$ we will use $s \xrightarrow{a} s'$, this can be read as: "the system in the state s can perform the action a and go to the state s'". The notation $s \xrightarrow{a} s'$ can be extended in a natural way to sequences of actions. Let $w \in \mathcal{A}^*$, $w = a_1, a_2, \ldots, a_n$. We write $s \xrightarrow{w} s'$ iff there is a sequence of states $s_0, s_1, \ldots, s_n$ such that $s_0 = s$, $s_n = s'$ and $s_{i-1} \xrightarrow{a_i} s_i$ for each i such that $1 \leq i \leq n$.

A state s' is reachable from a state s (written $s \longrightarrow^* s'$) iff there is some $w \in \mathcal{A}^*$ such that $s \xrightarrow{w} s'$.

Given an LTS $(S, \mathcal{A}, \longrightarrow)$, we define the *distance from state s to state t* by

$$dist(s,t) = \min \left\{ |w| \mid w \in \mathcal{A}^* \text{ and } s \xrightarrow{w} t \right\}.$$

2.2.3 Bisimulation Equivalence

The *equivalence-checking* approach to the formal verification of systems is based on the following scheme: the specification S (i.e., the intended behavior) and the actual implementation I of a system are defined as states in transition systems, and then it is shown that S and I are *equivalent.*

There are many possible ways how equivalence of processes can be defined. The most prominent of equivalences defined in the literature were organized by van Glabbeek into the hierarchy called linear time – branching time spectrum [42]. We will discuss just the finest equivalence from this spectrum - *bisimulation equivalence* (also known as *bisimilarity*). It is of special importance, as its accompanying theory has found its way into many practical applications.

Let $(S, \mathcal{A}, \longrightarrow)$ be a labelled transition system. A binary relation $\mathcal{R} \subseteq S \times S$ is a *bisimulation* iff for every pair of states $(s,t) \in \mathcal{R}$ and every action $a \in \mathcal{A}$ the following conditions hold:

- If there is some $s' \in S$ such that $s \xrightarrow{a} s'$, then there is some $t' \in S$ such that $t \xrightarrow{a} t'$ and $(s', t') \in \mathcal{R}$.
- If there is some $t' \in S$ such that $t \xrightarrow{a} t'$, then there is some $s' \in S$ such that $s \xrightarrow{a} s'$ and $(s', t') \in \mathcal{R}$.

(It is said that $s \xrightarrow{a} s'$ is *matched* by $t \xrightarrow{a} t'$, resp. $t \xrightarrow{a} t'$ is matched by $s \xrightarrow{a} s'$.)

States s, t are *bisimilar*, written $s \sim t$, iff there exists some bisimulation $\mathcal{R}$ such that $(s, t) \in \mathcal{R}$. The relation $\sim$ is called *bisimulation equivalence* or *bisimilarity*.

It is not difficult to show that $\sim$ is reflexive, symmetric and transitive. Notice that a union of a several bisimulation relations is also a bisimulation relation. This implies that $\sim$ which is the union of all bisimulations is the maximal bisimulation.

Bisimulation equivalence can also relate states of *different* transition systems, because we can consider two transition systems to be a single one by taking the disjoint union of them.

Let Δ_1, Δ_2 be (descriptions of) labelled transition systems with distinguished initial states s and t. $\Delta_1 \sim \Delta_2$ iff $s \sim t$. Two processes are related by $\sim$ iff their underlying LTSs are related by $\sim$.

The bisimulation equivalence on LTS $(S, \mathcal{A}, \longrightarrow)$ can be alternatively described in terms of a *bisimulation game* played by two players called Spoiler and Duplicator. The positions in the game are the pairs $(s, t) \in S \times S$. The game proceeds in rounds. In a position (s, t), Spoiler chooses one of states s and t (say s) and some transition from the chosen state (say $s \xrightarrow{a} s'$). Duplicator then chooses some transition from the other state with the same label as the transition chosen by Spoiler (say $t \xrightarrow{a} t'$). The game then continues by the next round in the position (s', t'). If one of players is stuck (he has no possible move) in some position, the other player wins. The case when the game is infinite is considered as a win of Duplicator.

Fact 2.1
$s \sim t$ iff Duplicator has a winning strategy in the bisimulation game starting in the position (s, t).

2.2.4 Process Rewrite Systems

In this thesis we will discuss bisimilarity on several formalisms. They can be all viewed as so called *Process Rewrite Systems* (PRS). This notion was defined by Mayr in [34].

Process rewrite systems are defined as follows. Let $\mathcal{A} = \{a, b, c, \ldots\}$ be a countably infinite set of atomic actions and $\mathit{Var} = \{X, Y, Z, \ldots\}$ be a countably infinite set of process variables. Process terms are defined by the following abstract grammar

$$P ::= \varepsilon \mid X \mid P_1.P_2 \mid P_1 \parallel P_2$$

where:

- ε is the empty term,
- X is a process variable,
- '.' denotes the sequential composition
- '$\parallel$' denotes the parallel composition.

The sequential composition is associative and the parallel composition is associative and commutative. We always work with equivalence classes of terms modulo associativity of the sequential composition and modulo associativity and commutativity of the parallel composition. We further define that $\varepsilon.P = P.\varepsilon = P$ and $P \parallel \varepsilon = P$.

Process rewrite system is a finite set of rules Δ containing rules of the form $t_1 \xrightarrow{a} t_2$ where t_1 and t_2 are process terms and $a \in \mathcal{A}$ is an atomic action. By $\mathit{Var}(\Delta)$ we will denote the set of process variables occurring in Δ and by $\mathcal{A}(\Delta)$ the set of atomic actions occurring in Δ.

Process rewrite system Δ produces a corresponding labelled transition system $(S, \mathcal{A}, \longrightarrow)$ where S is the set of process terms that contain only variables from $\mathit{Var}(\Delta)$, $\mathcal{A}' = \mathcal{A}(\Delta)$, and the transition relation is the smallest relation satisfying the following inference rules where t_1, t_2, t_1', t_2' are process terms:

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2} \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1.t_2 \xrightarrow{a} t_1'.t_2}$$

$$\frac{t_1 \xrightarrow{a} t_1'}{t_1 \parallel t_2 \xrightarrow{a} t_1' \parallel t_2} \qquad \frac{t_2 \xrightarrow{a} t_2'}{t_1 \parallel t_2 \xrightarrow{a} t_1 \parallel t_2'}$$

Note that $Var(\Delta)$ and $\mathcal{A}(\Delta)$ are finite. Since Δ is finite, the generated labelled transition system is finitely branching, which means that the number of outgoing transitions is finite for every state.

Note that there is no operator for non-deterministic choice ('+'), because nondeterminism can be encoded in the set of rules Δ which can contain more rules with the same term on the left side.

There can be defined different types of subclasses of process rewrite systems. We can define four classes of process terms:

- 1 – terms consisting of a single process variable (e.g., X),
- $\mathcal{S}$ – terms consisting of ε, a single variable, or a sequential composition of process variables (e.g., $X.Y.Z$),
- $\mathcal{P}$ – terms consisting of ε, a single variable, or a parallel composition of process variables (e.g., $X \parallel Y \parallel Z$),
- $\mathcal{G}$ – all process terms without any restriction (e.g., $X.(Y \parallel Z)$).

From description of those classes follows $1 \subsetneq \mathcal{S}$, $1 \subsetneq \mathcal{P}$, $\mathcal{S} \subsetneq \mathcal{G}$, and $\mathcal{P} \subsetneq \mathcal{G}$. Classes $\mathcal{S}$ and $\mathcal{P}$ are incomparable and $\mathcal{S} \cap \mathcal{P} = 1 \cup \{\varepsilon\}$.

Let $\alpha, \beta \in \{1, \mathcal{S}, \mathcal{P}, \mathcal{G}\}$ be classes of process terms such that $\alpha \subseteq \beta$. We define (α, β)-PRS as a finite set of rules Δ where in every rewrite rule $(l \xrightarrow{a} r) \in \Delta$ the term l is from class α and $l \neq \varepsilon$ and the term r is from class β (and r can be ε).

The hierarchy of (α, β)-PRS models is depicted in Figure 2.1. Each model in the hierarchy has a name shown also in the figure and many of these (α, β)-PRS correspond to well-known classes of infinite state systems studied in the literature. A line from a higher model to a lower model means that the higher model is more general than the lower one. It is known that the hierarchy is strict with respect to bisimilarity [34].

The classes of process rewrite systems correspond to the following following formalisms:

- FS – finite-state systems,
- BPA – Basic Process Algebra [5],
- BPP – Basic Parallel Processes [8],

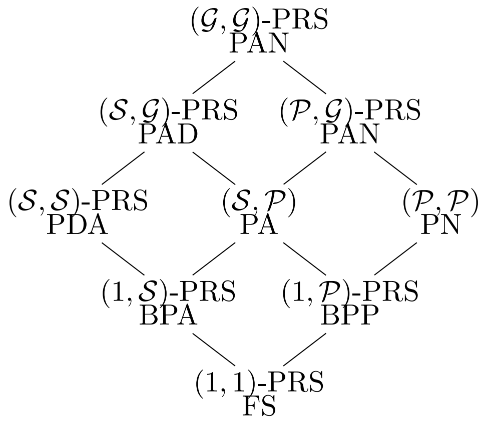

$(\mathcal{G},\mathcal{G})$-PRS
PAN
$(\mathcal{S},\mathcal{G})$-PRS
PAD
$(\mathcal{P},\mathcal{G})$-PRS
PAN
$(\mathcal{S},\mathcal{S})$-PRS
PDA
$(\mathcal{S},\mathcal{P})$
PA
$(\mathcal{P},\mathcal{P})$
PN
$(1,\mathcal{S})$-PRS
BPA
$(1,\mathcal{P})$-PRS
BPP
$(1,1)$-PRS
FS

Figure 2.1: Hierarchy of process rewrite systems

- PDA – Pushdown Automata,
- PA – Process Algebra [2],
- PN – Petri nets
- PRS – Process Rewrite Systems

Classes PAD and PAN were introduced in [33] to naturally complete the hierarchy. Class PAD is the 'smallest' common generalization of classes PDA and PA and PAN is the 'smallest' common generalization of classes PA and PN.

2.2.5 BPA and BPP

As was said in the subsection 2.2.4, many classes from (α, β)-PRS hierarchy were known and studied before definition of process rewrite systems. In this thesis we will discus mainly BPA and BPP. Both classes may be defined in several different ways, besides the PRS definition, for example as subclasses of Process Algebra (introduced in [2]). The following definitions are most suitable for our purposes.

A *BPA system*, or *BPA* for short, can be viewed as a context-free grammar in Greibach normal form. Formally, it is a triple $\Sigma = (V_\Sigma, \mathcal{A}_\Sigma, \Gamma_\Sigma)$, where:

- V_Σ is a finite set of *variables* (nonterminals),

$\cdots$

$$\begin{tikzcd}
A \arrow[r, "b"] \arrow[d, "b"] & AB \arrow[r, "b"] \arrow[d, "b"] & ABB \arrow[r, "b"] \arrow[d, "b"] & ABBB \arrow[r, "b"] \arrow[d, "b"] & \cdots \\
\varepsilon & B \arrow[l, "a"] & BB \arrow[l, "a"] & BBB \arrow[l, "a"] & \cdots \arrow[l, "a"]
\end{tikzcd}$$

Figure 2.2: Part of LTS corresponding to the BPA from example

- $\mathcal{A}_\Sigma$ is a finite set of *actions* (terminals),
- $\Gamma_\Sigma \subseteq V_\Sigma \times \mathcal{A}_\Sigma \times V_\Sigma^*$ is a finite set of *rewrite rules.*

We often use $V, \mathcal{A}, \Gamma$ without subscripts when the underlying BPA is clear from context. We also write $X \xrightarrow{a} \alpha$ instead of $(X, a, \alpha) \in \Gamma$.

A *BPA process* is a pair (α, Σ) where Σ is a BPA system and $\alpha \in V^*$; we write just α when Σ is clear from context. A BPA Σ gives rise to the LTS $\mathcal{S}_\Sigma = (V^*, \mathcal{A}, \longrightarrow)$ where $\longrightarrow$ is induced from the rewrite rules by the following (deduction) rule: if $X \xrightarrow{a} \alpha$ then $X\beta \xrightarrow{a} \alpha\beta$ for every $\beta \in V^*$.

Example. Suppose the following BPA system:

$$\begin{array}{ll}
V = \{A, B\} & A \xrightarrow{b} \varepsilon \\
\mathcal{A} = \{a, b\} & A \xrightarrow{b} AB \\
 & B \xrightarrow{a} \varepsilon
\end{array}$$

The LTS corresponding to this BPA is infinite. A part of it containing states reachable from A is diagrammatically shown on Figure 2.2.

A *BPP system*, or *BPP* for short, can be defined in a similar way, as a triple $\Delta = (V_\Delta, \mathcal{A}_\Delta, \Gamma_\Delta)$. The only difference is the deduction rule for the associated LTS $\mathcal{S}_\Delta$: if $X \xrightarrow{a} \alpha$ then $\gamma X \delta \xrightarrow{a} \gamma\alpha\delta$ for any $\gamma, \delta \in V^*$ (thus *any* occurrence of a variable can be rewritten, not just the first one). From the definition follows that BPP processes α, β with the same Parikh image (i.e., containing the same number of occurrences of each variable) are bisimilar. Hence BPP processes can be read modulo commutativity of concatenation and interpreted as multisets of variables; in the rest of the thesis we interpret BPP processes in this way whenever convenient. This also suggests to identify a BPP system Δ with a *BPP net*, a labelled Petri net in which each transition has exactly one input place (also known as communication-free Petri net).

Formally, a *BPP net* is a tuple $\Delta = (P_\Delta, Tr_\Delta, \textsc{pre}_\Delta, F_\Delta, \mathcal{A}_\Delta, l_\Delta)$ where:

- P_Δ is a finite set of *places*,
- Tr_Δ is a finite set of *transitions*,
- $\textsc{pre}_\Delta : Tr_\Delta \to P_\Delta$ is a function assigning an input place to each transition,
- $F_\Delta : (Tr_\Delta \times P_\Delta) \to \mathbb{N}$ is a *flow function*,
- $\mathcal{A}_\Delta$ is a finite set of *actions*,
- $l_\Delta : Tr_\Delta \to \mathcal{A}_\Delta$ is a *labelling function.*

We will use P, $Tr, \textsc{pre}, F, \mathcal{A}, l$ if the underlying BPP net is clear from context. We note that a transition $t \in Tr$ can be viewed as the rewrite rule $p \xrightarrow{a} \alpha$ where $\textsc{pre}(t) = p$ and $F(t, p')$ is the number of occurrences of p' in α, for each $p' \in P$.

For each grammar representation $(V, \mathcal{A}, \Gamma)$ of a BPP Δ there is an equivalent BPP net representation $(V, Tr, \textsc{pre}, F, \mathcal{A}, l)$ where for each rule $A \xrightarrow{a} \alpha$ $(A \in V,\ \alpha \in V^*,\ a \in \mathcal{A})$ there is a transition $t \in Tr$ such that $\textsc{pre}(t) = A$, $F(t, X) = |\alpha|_X$ and $l(t) = a$.

A BPP process is thus, in fact, a *marking*, i.e. a function $M : P \to \mathbb{N}$ which associates a finite number of *tokens* to each place. We will use p^k to denote the marking M where all k tokens are in one place p ($M(p) = k$ and $M(p') = 0$ for each $p' \neq p$); $p^0 = \varepsilon$ represents the *zero marking* ($M(p) = 0$ for all $p \in P$).

A transition t is *enabled* at marking M if $M(\textsc{pre}(t)) \geq 1$. An enabled transition t may fire from M, producing a marking M' defined by

$$M'(p) = \begin{cases} M(p) - 1 + F(t,p) & \text{if } p = \textsc{pre}(t) \\ M(p) + F(t,p) & \text{otherwise} \end{cases} .$$

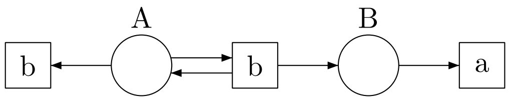
This is denoted by $M \xrightarrow{t} M'$; the notation is extended to $M \xrightarrow{\sigma} M'$ for sequences $\sigma \in T^*$. We write $M \xrightarrow{\sigma}$ if $M \xrightarrow{\sigma} M'$ for some M'.

In the above sense, a BPP Δ gives rise to the LTS $\mathcal{S}_\Delta = (\mathcal{M}_\Delta, \mathcal{A}, \longrightarrow)$ where $\mathcal{M}_\Delta = \mathbb{N}^P$ is the set of all markings (of the respective BPP net), and $M \xrightarrow{a} M'$ iff there is some $t \in Tr$ such that $l(t) = a$ and $M \xrightarrow{t} M'$.

Example. A simple BPP net with two places (A, B) and three transitions can be described using the following graphical representation:

$$\begin{tikzcd}
\vdots \arrow[d, "b"] & \vdots \arrow[d, "b"] & \vdots \arrow[d, "b"] & \vdots \arrow[d, "b"] & \\
(2,0) \arrow[r, shift left, "b"] \arrow[d, "b"] & (2,1) \arrow[l, shift left, "a"] \arrow[r, shift left, "b"] \arrow[d, "b"] & (2,2) \arrow[l, shift left, "a"] \arrow[r, shift left, "b"] \arrow[d, "b"] & (2,3) \arrow[l, shift left, "a"] \arrow[r, shift left, "b"] \arrow[d, "b"] & \cdots \arrow[l, shift left, "a"] \\
(1,0) \arrow[r, shift left, "b"] \arrow[d, "b"] & (1,1) \arrow[l, shift left, "a"] \arrow[r, shift left, "b"] \arrow[d, "b"] & (1,2) \arrow[l, shift left, "a"] \arrow[r, shift left, "b"] \arrow[d, "b"] & (1,3) \arrow[l, shift left, "a"] \arrow[r, shift left, "b"] \arrow[d, "b"] & \cdots \arrow[l, shift left, "a"] \\
(0,0) & (0,1) \arrow[l, "a"'] & (0,2) \arrow[l, "a"'] & (0,3) \arrow[l, "a"'] & \cdots \arrow[l, "a"']
\end{tikzcd}$$

Figure 2.3: LTS corresponding to the BPP from example



The LTS corresponding to this BPP is shown on Figure 2.3.

A corresponding grammar representation of the same BPP is:

$V = \{A, B\}$ $\quad A \xrightarrow{b} \varepsilon$
$\mathcal{A} = \{a, b\}$ $\quad A \xrightarrow{b} AB$
$\quad B \xrightarrow{a} \varepsilon$

A place $p \in P$ is *unbounded* in (M_0, Δ) iff for each $c \in \mathbb{N}$ there is a marking M' such that $M_0 \longrightarrow^* M'$ and $M'(p) > c$.

We define $\mathit{Tok}(M) = \sum_{p \in P} M(p)$ and $\mathit{Car}(M) = \{p \in P \mid M(p) \geq 1\}$.

In the rest of the thesis we use symbols $\alpha, \beta, \ldots$ for both BPA processes and BPP processes, and $M_1, M_2, \ldots$ only for the latter.

We say that a BPA system Σ (a BPP net Δ) is *normed* iff $\alpha \longrightarrow^* \varepsilon$ for each state α of $\mathcal{S}_\Sigma$ ($\mathcal{S}_\Delta$). We use nBPA (nBPP) for normed BPA (normed BPP).

Let α be a state (of $\mathcal{S}_\Sigma$ or $\mathcal{S}_\Delta$). The *norm* of α, denoted $\textsc{norm}(\alpha)$, is the length of the shortest $w \in \mathcal{A}^*$ such that $\alpha \xrightarrow{w} \varepsilon$. Note that this also defines norm $\textsc{norm}(X)$ for each variable (place) X.

We now note some obvious properties of norms.

- If $\alpha \neq \varepsilon$ then $\textsc{norm}(\alpha) > 0$ for any state α.

- In each nBPA (or nBPP), there is at least one variable (place) with norm 1.

- If a rule $X \xrightarrow{a} \alpha$ is used in a transition $\beta \xrightarrow{a} \beta'$ then $\textsc{norm}(\beta') - \textsc{norm}(\beta) = \textsc{norm}(\alpha) - \textsc{norm}(X)$.

- $\textsc{norm}(\alpha\beta) = \textsc{norm}(\alpha) + \textsc{norm}(\beta)$ (for the BPP-net representation it means $\textsc{norm}(M_1 + M_2) = \textsc{norm}(M_1) + \textsc{norm}(M_2)$ where the sum $M_1 + M_2$ is defined componentwise).

- If $\alpha \sim \beta$ then $\textsc{norm}(\alpha) = \textsc{norm}(\beta)$.

Note also that if $\alpha_1 \sim \alpha_2$, $w \in \mathcal{A}^*$ and $\alpha_1 \xrightarrow{w} \alpha_1'$ then there must be a *matching sequence* $\alpha_2 \xrightarrow{w} \alpha_2'$ such that $\alpha_1' \sim \alpha_2'$ (and thus also $\textsc{norm}(\alpha_1') = \textsc{norm}(\alpha_2')$).

For two states α_1, α_2 we write $\alpha_1 \longrightarrow_R \alpha_2$ if $\alpha_1 \longrightarrow \alpha_2$ and $\textsc{norm}(\alpha_2) = \textsc{norm}(\alpha_1) - 1$. Such a step is called a *norm-reducing step* and the respective rule (transition) is also called *norm reducing.* We write $\alpha_1 \longrightarrow_R^* \alpha_2$ if there is a sequence (called *norm reducing sequence*) of norm reducing steps leading from α_1 to α_2. For each variable (place) X there is at least one norm-reducing rule (transition) $X \longrightarrow_R \alpha$.

For a marking M and a set $Q \subseteq P$ we define $\textsc{norm}_Q(M)$, the *norm of M wrt Q*, as the length of the shortest $w \in \mathcal{A}^*$ such that $M \xrightarrow{w} M'$ where $M'(p) = 0$ for all $p \in Q$. In fact, $\textsc{norm}_Q(M) = \sum_{p \in Q} c_p \cdot M(p)$ where $c_p = \textsc{norm}_Q(p)$.

A set of places $R \subseteq P$ of a Petri net is a *trap* iff

$$\forall t : \textsc{pre}(t) \in R \Rightarrow (R \cap \textsc{succ}(t) \neq \emptyset)$$

Intuitively this means that every t removing tokens from a trap also adds some tokens to it, so ‘marked’ trap, i.e., a trap with at least one token, can not get unmarked.

2.3 Bisimilarity between BPP and Finite-State System

2.3.1 Introduction

In this section we present an algorithm for the problem of deciding bisimilarity on special case of BPP-BISIM where one of (unnormed) BPP processes is a finite-state process. This was published previously in [26].

The running time of the algorithm is $O(n^4)$ where n is the size of the instance. The result implies that it is possible to verify in polynomial time whether a system implemented as a finite-state automaton is equivalent to a 'specification' given as a BPP. The algorithm also generates for each state of the finite-state system a 'symbolic' semilinear representation of bisimilar BPP states.

A *finite-state system* (FS) is traditionally defined as an LTS $(S, \mathcal{A}, \longrightarrow)$ where S is finite, but for technical convenience we define it as a BPP where for each $t \in Tr$ there is exactly one $p \in P$ such that $F(t,p) = 1$ and $F(t,p') = 0$ if $p' \neq p$. For $p \in P$ we define a marking M_p such that $M_p(p) = 1$ and $M_p(p') = 0$ for $p' \neq p$. We call such marking an *FS marking*.

Our problem can be defined as follows:

Problem: BPP-FS-BISIM

INSTANCE: A BPP systems Δ_1 and Δ_2, markings $M_1 \in \mathcal{M}_{\Delta_1}, M_2 \in \mathcal{M}_{\Delta_2}$ such that Δ_1 is a finite-state system and M_1 is an FS marking..

QUESTION: Is $M_1 \sim M_2$?

We assume that BPPs in the instance are encoded as lists of places and transitions, where the encoding of each transition t contains a list of all $p \in \textsc{succ}(t)$ together with values $F(t,p)$. We assume that numbers are encoded in binary.

In the rest of this section $\Delta = (P, Tr, \textsc{pre}, F, \lambda)$ is the disjoint union of the BPP and the FS from the instance of BPP-FS-BISIM, $\mathcal{M}$ denotes its set of markings, P_{FS} and Tr_{FS} are the sets of places and transitions of the FS from this instance ($P_{FS} \subseteq P$, $Tr_{FS} \subseteq Tr$), and M_p where $p \in P_{FS}$ denotes the marking such that $M_p \in \mathcal{M}$, $M_p(p) = 1$ and $M_p(p') = 0$ for $p' \neq p$. We define $\mathcal{M}_{FS} = \{M_p \mid p \in P_{FS}\}$.

2.3.2 Notions and Claims from Jančar's Paper on BPP Bisimilarity

Let $(S, \mathcal{A}, \longrightarrow)$ be an LTS, and let $\mathcal{C} : S \to \mathcal{D}$ be a mapping assigning to each state a value from some domain $\mathcal{D}$. We say the mapping $\mathcal{C}$ is *bis-necessary* if for each $s, s' \in S$, $s \sim s'$ implies $\mathcal{C}(s) = \mathcal{C}(s')$. If we have a set of functions $\{\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_l\}$ where $\mathcal{C}_i : S \to \mathcal{D}_i$, we say the set is *bis-necessary* iff every $\mathcal{C}_i$ is bis-necessary. A predicate $\mathcal{P}$ on S can be viewed as a mapping $\mathcal{P} : S \to \{0, 1\}$, and so we can also talk about *bis-necessary predicate.* Note that if $\mathcal{P}$ is bis-necessary, then $\neg\mathcal{P}$ is also bis-necessary.

Let $\mathcal{P}$ be a predicate on S. We define the mapping $\mathit{dist}(\mathcal{P}) : S \to \mathbb{N}_\omega$ where $\mathit{dist}(\mathcal{P})(s)$ is the length of the shortest w such that $s \xrightarrow{w} s'$ and $\mathcal{P}(s')$, and if there is no such w, $\mathit{dist}(\mathcal{P})(s) = \omega$. Intuitively, $\mathit{dist}(\mathcal{P})$ represents 'distance' to $\mathcal{P}$.

Claim 2.2
If $\mathcal{P}$ is bis-necessary then $\mathit{dist}(\mathcal{P})$ is bis-necessary.

Proof: Let us assume without loss of generality that there are states s_1, s_2 such that $s_1 \sim s_2$ and $\mathit{dist}(\mathcal{P})(s_1) < \mathit{dist}(\mathcal{P})(s_2)$. Then there is some shortest $w \in \mathcal{A}^*$ such that $s_1 \xrightarrow{w} s_1'$ and $\mathcal{P}(s_1')$. Because $s_1 \sim s_2$, there must be some s_2' such that $s_2 \xrightarrow{w} s_2'$ and $s_1' \sim s_2'$. But $|w| < \mathit{dist}(\mathcal{P})(s_2')$, and so $\neg\mathcal{P}(s_2')$, which means that $\mathcal{P}$ is not bis-necessary. $\square$

Let us now consider the BPP $\Delta = (P, \mathit{Tr}, \textsc{pre}, F, \mathcal{A}, l)$ from the instance of BPP-BISIM. Let $T \subseteq \mathit{Tr}$. We say T is *disabled* in M if every $t \in T$ is disabled in M. Notice that if T is the set of all transitions t such that $\lambda(t) = a$ for some $a \in \mathcal{A}$, then 'T is disabled' is a bis-necessary predicate. Notice also that T is disabled iff each place in $\textsc{pre}(T)$ is empty. These leads to the following formal definitions. Let $Q \subseteq P$ be a set of places. We define the predicate $\textsc{zero}(Q)$ on $\mathcal{M}$ such that $\textsc{zero}(Q)(M)$ iff $\forall p \in Q : M(p) = 0$. We define *norm* of Q as the function $\textsc{norm}_Q = \mathit{dist}(\textsc{zero}(Q))$.

Every norm can be expressed as a *linear function* $L : \mathcal{M} \to \mathbb{N}_\omega$ of the form

$$L(x_1, x_2, \ldots, x_k) = c_1x_1 + c_2x_2 + \cdots + c_kx_k$$

where $c_i \in \mathbb{N}_\omega$ and k is the number of places, see [16] for details. Coefficients $c_1, c_2, \ldots, c_k$ of L for the given Q can be computed by the algorithm in Figure 2.4. Intuitively, c_i is the minimal number of transitions that remove one token in p_i from Q. In the algorithm, Q' is the set of unprocessed places and T is the set of unprocessed transitions. We write c_p instead of c_i

```
for each $p \in P$ do
    if $p \in Q$ then $c_p := \omega$ else $c_p := 0$
$Q' := Q$
$T := \{t \in Tr \mid \textsc{pre}(t) \in Q'\}$
while $Q' \neq \emptyset$ do
    let $p_{min}$ refer to some $p \in Q'$ with minimal $c_p$
    for each $t \in T$ such that $\textsc{succ}(t) \cap Q' = \emptyset$ do
        remove $t$ from $T$
        $p := \textsc{pre}(t)$; $R := \textsc{succ}(t)$
        $d_t := 1 + \sum_{q \in R} c_q \cdot F(t, q)$
        if $d_t < c_p$ then $c_p := d_t$
        if $c_p < c_{p_{min}}$ then $p_{min} := p$
    end for
    if $c_{p_{min}} = \omega$ then break;
    $Q' := Q' - \{p_{min}\}$
    remove from $T$ every $t$ such that $\textsc{pre}(t) = p_{min}$
end while
```

Figure 2.4: Computing coefficients of $\textsc{norm}_Q$ function

where $p = p_i$. Places that are not in Q' are places for which c_p was already determined. The algorithm computes for each unprocessed transition t that stores tokens only to places out of Q' the value d_t, a possible candidate for c_p where $p = \textsc{pre}(t)$, and chooses between these candidates the one with the minimal value.

We define $\Omega\text{-}\textsc{carr}(L) = \{p_i \in P \mid c_i = \omega\}$. Note that $L(M) = \omega$ iff $M(p) > 0$ for some $p \in \Omega\text{-}\textsc{carr}(L)$. It is not hard to show that $\Omega\text{-}\textsc{carr}(L)$ is a trap. From the properties of traps follows the following claim:

Claim 2.3
If $L = \textsc{norm}_Q$ for some $Q \subseteq P$ and $L(M) = \omega$, then $L(M') = \omega$ for every M' such that $\exists w \in \mathcal{A}^* : M \xrightarrow{w} M'$.

For a linear function L we can compute for each $t \in Tr$ the value

$$\delta_t^L = -c_i + \sum_{1 \leq j \leq k} c_j \cdot F(t, p_j) \tag{2.1}$$

where $\textsc{pre}(t) = p_i$. The value δ_t^L represents the 'change' on the value of L when the transition t is performed.

Claim 2.4
If $M \xrightarrow{t} M'$ then $L(M) + \delta_t^L = L(M')$. If $L(M) < \omega$ and $\delta \neq \delta_t^L$ then $L(M) + \delta \neq L(M')$.

2.3.3 The Algorithm

The basic idea is to construct a series of norm functions that are used for approximation of the bisimulation equivalence. The construction stops when no other function can be added, and at this point the approximation is exact.

The algorithm for BPP-FS-BISIM works similarly as algorithm for BPP-BISIM described in [16]. It constructs a set of linear functions $\mathcal{L} = \{L_1, L_2, \ldots\}$ such that each L_i represents norm of some set of places and where each L_i is bis-necessary. The algorithm starts with $\mathcal{L} = \emptyset$, successively adds linear functions to $\mathcal{L}$ and stops when no new linear function can be added. For $\mathcal{L}$ we define the equivalence $\equiv_{\mathcal{L}}$ on $\mathcal{M}$ such that $M \equiv_{\mathcal{L}} M'$ iff $\forall L \in \mathcal{L} : L(M) = L(M')$. Since each $L \in \mathcal{L}$ is bis-necessary, $\mathcal{L}$ is also bis-necessary, and $M \not\equiv_{\mathcal{L}} M'$ implies $M \not\sim M'$. On the other hand, we show that if $M \in \mathcal{M}_{FS}$ and $M' \in \mathcal{M}$ then $M \equiv_{\mathcal{L}} M'$ implies $M \sim M'$.

The main algorithm looks as follows:

1. Set $\mathcal{L} = \emptyset$.
2. For each $p \in P_{FS}$ perform Step described below.
3. If $\mathcal{L}$ has changed in the previous step, go to 2.

The Step looks as follows: For the given p we define the set $\mathcal{F} \subseteq \mathcal{L}$ such that $L \in \mathcal{F}$ iff $L(M_p) < \omega$. For $\mathcal{F}$ we define the equivalence $\cong_{\mathcal{F}}$ on Tr such that $t \cong_{\mathcal{F}} t'$ iff $\lambda(t) = \lambda(t')$ and $\forall L \in \mathcal{F} : \delta_t^L = \delta_{t'}^L$. Let $[t]$ denote the equivalence class of $\cong_{\mathcal{F}}$ containing t. Let $\mathcal{T}_1 = \{[t] \mid t \in \textsc{succ}(p)\}$, and let $T_0 = Tr - \bigcup_{T \in \mathcal{T}_1} T$. We define the set $\mathcal{T}$ as $\mathcal{T} = \mathcal{T}_1 \cup \{T_0\}$, respectively as $\mathcal{T} = \mathcal{T}_1$ when T_0 is empty. Note $\mathcal{T}$ is a partition of Tr. We extend the definition of Ω-carr to sets of linear functions and define

$$\Omega\text{-}\textsc{carr}(\mathcal{F}) = \bigcup_{L \in \mathcal{F}} \Omega\text{-}\textsc{carr}(L)$$

The algorithm now computes the function $L = \textsc{norm}_{\textsc{pre}(T) \cup \Omega\text{-}\textsc{carr}(\mathcal{F})}$ for each $T \in \mathcal{T}$ and adds it to $\mathcal{L}$.

We now show that the algorithm is correct.

Lemma 2.5
Every L added to $\mathcal{L}$ by the algorithm is bis-necessary.

Proof: We proceed by induction on the number of steps. The proposition is trivially true at the start. Assume now the algorithm performs STEP for some $p \in P_{FS}$ and adds $\textsc{norm}_Q$ to $\mathcal{L}$ for some $T \in \mathcal{T}$ where $Q = \textsc{pre}(T) \cup \Omega\text{-}\textsc{carr}(\mathcal{F})$. Due to Claim 2.2 it is sufficient to show that $\textsc{zero}(Q)$ is bis-necessary. Let us assume without loss of generality that $M_1 \sim M_2$, $\neg\textsc{zero}(Q)(M_1)$, and $\textsc{zero}(Q)(M_2)$. By induction hypothesis, $\forall L \in \mathcal{L} : L(M_1) = L(M_2)$. Let $R = \Omega\text{-}\textsc{carr}(\mathcal{F})$. Since $\textsc{zero}(R)(M_2)$, we have $\forall L \in \mathcal{F} : L(M_2) < \omega$, and $\textsc{zero}(R)(M_1)$, since otherwise there is some $L \in \mathcal{F}$ such that $L(M_1) = \omega \neq L(M_2)$. From this and $\neg\textsc{zero}(Q)(M_1)$ we have $\neg\textsc{zero}(\textsc{pre}(T))(M_1)$. This means there is some transition $t \in T$ such that $M_1 \xrightarrow{t} M_1'$. Because $M_1 \sim M_2$ there is some t' such that $M_2 \xrightarrow{t'} M_2'$ where $M_2 \sim M_2'$ and $\lambda(t) = \lambda(t')$, but necessarily $t' \notin T$. This means there is some $L \in \mathcal{F}$ such that $\delta_t^L \neq \delta_{t'}^L$. By Claim 2.4 this implies $L(M_1') \neq L(M_2')$, a contradiction. □

Since every L added to $\mathcal{L}$ is $\textsc{norm}_Q$ for some $Q \subseteq P$, and P is finite, it is obvious that the algorithm stops after some finite number of steps. The following lemma shows that $\equiv_{\mathcal{L}}$ corresponding to $\mathcal{L}$ computed by the algorithm coincides with $\sim$ on pairs of markings where one of markings is from $\mathcal{M}_{FS}$.

Lemma 2.6
Let $\mathcal{L}$ be the set computed by the algorithm. Then for every $M_1 \in \mathcal{M}_{FS}$ and $M_2 \in \mathcal{M}$, $M_1 \equiv_{\mathcal{L}} M_2$ implies $M_1 \sim M_2$.

Proof: We show that $\equiv_{\mathcal{L}} \cap (\mathcal{M}_{FS} \times \mathcal{M})$ is a bisimulation. Let us consider $M_1 \in \mathcal{M}_{FS}$ and $M_2 \in \mathcal{M}$ such that $M_1 \equiv_{\mathcal{L}} M_2$. Let $\mathcal{F} = \{L \in \mathcal{L} \mid L(M_1) < \omega\}$ and let $R = \Omega\text{-}\textsc{carr}(\mathcal{F})$. Note that $M_1 = M_p$ for some $p \in P_{FS}$ and the same $\mathcal{F}$ would be produced when the algorithm would perform STEP for p. Notice that $\textsc{zero}(R)(M_1)$ since otherwise there is some $L \in \mathcal{F}$ such that $L(M_1) = \omega$. Also $\textsc{zero}(R)(M_2)$ is true, because otherwise there is some $L \in \mathcal{F}$ such that $L(M_2) = \omega$ which means $L(M_1) \neq L(M_2)$. Let $\mathcal{T}$ be defined for $\mathcal{F}$ correspondingly as in STEP.

At first consider a transition $M_1 \xrightarrow{t} M_1'$. Let T be the class from $\mathcal{T}$ such that $t \in T$. Obviously $T \in \mathcal{T}_1$. Consider now the function $L_1 = \textsc{norm}_{R \cup \textsc{pre}(T)}$. It must be the case that $L_1 \in \mathcal{L}$, otherwise L_1 could be added to $\mathcal{L}$ and the algorithm has not finished yet. So $L_1(M_1) =$

$L_1(M_2)$. From this, from $\neg\textsc{zero}(\textsc{pre}(T))(M_1)$, and from $\textsc{zero}(R)(M_2)$ we have $\neg\textsc{zero}(\textsc{pre}(T))(M_2)$ and there is some $t' \in T$ such that $M_2 \xrightarrow{t'} M_2'$, $\lambda(t) = \lambda(t')$, and $\forall L \in \mathcal{F} : \delta_t^L = \delta_{t'}^L$. From this and Claim 2.4 we obtain $\forall L \in \mathcal{F} : L(M_1') = L(M_2')$. For each $L \in \mathcal{L} - \mathcal{F}$ is $L(M_1) = L(M_2) = \omega$, and, by Claim 2.3, $L(M_1') = L(M_2') = \omega$. This means $M_1' \equiv_{\mathcal{L}} M_2'$.

Now consider a transition $M_2 \xrightarrow{t'} M_2'$. This case similar to the previous case, but we must also consider the possibility $t' \in T_0$. Let $L_0 = \textsc{norm}_{R \cup \textsc{pre}(T_0)}$. Since $L_0 \in \mathcal{L}$ (otherwise the algorithm has not finished yet), $L_0(M_1) = L_0(M_2)$. Because $L_0(M_1) = 0$, we obtain $L_0(M_2) = 0$, and $\textsc{zero}(\textsc{pre}(T_0))(M_2)$, so t' is not enabled in M_2, a contradiction. $\square$

2.3.4 Time Complexity of the Algorithm

In this section we show that the running time of the algorithm is $O(n^4)$. In the rest of the Section 2.3 n denotes the size of the input instance.

The running time of the algorithm depends on implementation details of Step. In Subsection 2.3.3 we described how to, for the given $p \in P_{FS}$, compute in Step sets $\mathcal{F}$, $\Omega\text{-}\textsc{carr}(\mathcal{F})$, and $\mathcal{T}$. It is more efficient not to recompute these sets every time, but instead to store their values and perform necessary changes on them when new L is added to $\mathcal{L}$. So the algorithm maintains for each $p \in P_{FS}$ values of the corresponding $\Omega\text{-}\textsc{carr}(\mathcal{F})$ and $\mathcal{T}$. Note that $\mathcal{T}$ always contains at most $|\textsc{succ}(p)| + 1$ equivalence classes. The algorithm also maintains for each $T \in \mathcal{T}$ and for $\Omega\text{-}\textsc{carr}(\mathcal{F})$ a boolean flag indicating whether it has changed since the last invocation of Step and adds a new function $L = \textsc{norm}_{\Omega\text{-}\textsc{carr}(\mathcal{F}) \cup T}$ to $\mathcal{L}$ only when $\Omega\text{-}\textsc{carr}(\mathcal{F})$ or T is new or has actually changed.

Addition of L to $\mathcal{L}$ includes the following steps:

1. Compute coefficients $c_1, c_2, \ldots, c_k$ of L.

2. Compute δ_t^L for each $t \in \mathit{Tr}$.

3. Partition Tr according to values of δ_t^L and $\lambda(t)$.

4. For each $p \in P_{FS}$ such that $L(M_p) < \omega$:
   - Add $\Omega\text{-}\textsc{carr}(L)$ to the corresponding $\Omega\text{-}\textsc{carr}(\mathcal{F})$.
   - Modify the corresponding $\mathcal{T}$ using the partition computed in step 3.

In the proof we need the following well-known fact:

Fact 2.7
Let U be a non-empty finite set, and let $\mathcal{U}_1, \mathcal{U}_2, \ldots$ be a sequence of partitions of U such that each $\mathcal{U}_{i+1}$ is a refinement of $\mathcal{U}_i$. Then the total number of different classes in all these partitions is less then $2 \cdot |U|$.

Proof: It is simple induction on $|U|$. □

Lemma 2.8
The number of functions added to $\mathcal{L}$ is in $O(n^2)$.

Proof: Let us consider all invocations of STEP for one $p \in P_{FS}$. In invocations where Ω-CARR$(\mathcal{F})$ has changed, the algorithm adds a new function to $\mathcal{L}$ for each $T \in \mathcal{T}$. If Ω-CARR$(\mathcal{F})$ has not changed, a new function is added for each $T \in \mathcal{T}$ that has changed.

Ω-CARR$(\mathcal{F})$ can only grow, so the number of invocations of STEP when Ω-CARR$(\mathcal{F})$ has changed is $O(|P|)$. Because $|\mathcal{T}|$ is at most $h+1$ where $h = \text{SUCC}(p)$, the number of functions added in such invocations is at most $(h+1) \cdot O(|P|)$.

Consider now the possible changes of $\mathcal{T}$. Either some t was added to T_0, or some $T \in \mathcal{T}_1$ was split, or some combination of these possibilities has occurred. Since T_0 can only grow, the first possibility can occur only $O(|\mathit{Tr}|)$ times. It remains to estimate the total number of classes from $\mathcal{T}_1$. It is in $O(|\mathit{Tr}|)$ as follows from Fact 2.7, since sequence of values of $\mathcal{T}_1$ in subsequent invocations of STEP can be extended to a sequence of refined partitions by adding some classes to each $\mathcal{T}_1$.

Let us sum now the numbers of functions added to $\mathcal{L}$ for all $p \in P_{FS}$. In invocations where Ω-CARR$(\mathcal{F})$ has changed it is at most

$$\sum_{p \in P_{FS}} (|\text{SUCC}(p)| + 1) \cdot O(|P|) = O(|P| \cdot |\mathit{Tr}_{FS}|)$$

The number of function added in the remaining invocations is in $O(|P_{FS}| \cdot |\mathit{Tr}|)$, so we obtain that the total number of functions is in $O(|P| \cdot |\mathit{Tr}|)$. □

Now we consider the complexity of computation of coefficients of $L = \text{NORM}_Q$ for some $Q \subseteq P$. For $x \in \mathbb{N}_\omega$, $\mathit{size}(x)$ denotes the number of bits of x when encoded in binary. We suppose that $\mathit{size}(x+y) = 1 + \max\{\mathit{size}(x), \mathit{size}(y)\}$, $\mathit{size}(x \cdot y) = \mathit{size}(x) + \mathit{size}(y)$, and $\mathit{size}(\omega) = O(1)$.

Proposition 2.9
For each $p \in P$, size(c_p) is in $O(n)$.

Proof: Let $p_1, p_2, \ldots, p_l$ be the sequence of places from Q ordered by the order in which the algorithm determines their coefficients, let c_i be the coefficient of p_i, and let t_i be the transition used for computation of c_i, i.e., the transition such that $\textsc{pre}(t_i) = p_i$ and $c_i = d_i$, where we write d_i instead of d_{t_i}. Let *size*(t) be the number of bits of representation of $t \in T$, i.e.,

$$\mathit{size}(t) = O((1 + |R|) \cdot \mathit{size}(|P|)) + \sum_{p \in R} \mathit{size}(F(t,p))$$

where $R = \textsc{succ}(t)$.

By induction on i we prove the following proposition from which the result directly follows: For each i, $1 \leq i \leq l$, $\mathit{size}(c_i) \leq \sum_{1 \leq j \leq i} \mathit{size}(t_j)$. This holds trivially for $i = 1$ because c_1 is always 1 or ω, so suppose $i > 1$. Let $R = \textsc{succ}(t_i)$. Note that

$$d_i = 1 + \sum_{q \in R} c_q \cdot F(t_i, q) \leq 1 + \sum_{j=1}^{i-1} c_j \cdot F(t_i, p_j)$$

because when d_i is computed, each c_q is known and finite, and so it is either 0 or one from c_1 to c_{i-1}.

$\mathit{size}(c_j \cdot F(t_i, p_j)) = \mathit{size}(c_j) + \mathit{size}(F(t_i, p_j))$. The sum of all such products can be written in the size of maximal of them plus some number less then their count (overflow caused by addition). This size is less then $\mathit{size}(\max\{c_j \mid 1 \leq j < i\}) + \sum_{j=1}^{i-1} \mathit{size}(F(t_i, p_j))$. The second summand (the sum) is less then *size*(t_i). By induction hypothesis maximal c_j can be written in the count of bits needed for first $i - 1$ transitions. Therefore d_i (and hence c_i too) can be written in the space needed for representations of transitions $t_1, \ldots, t_i$. □

Proposition 2.10
All coefficients of $L = \textsc{norm}_Q$ are computed in $O(n^2)$.

Proof: The most time-consuming step is computation of all d_i. In computation of this, multiplications are more time-consuming than additions. Hence it suffices to show that aggregated complexity of all multiplications is in $O(n^2)$.

In our algorithm, each d_i is computed only once. During computation of d_i we need to determine all products $F(t_i, p_j) \cdot c_j$ where $p_j \in \textsc{succ}(t)$. From

Proposition 2.9 we know that $\mathit{size}(c_j)$ is in $O(n)$ for every c_j. Hence one product is computed in $O(n \cdot \mathit{size}(F(t_i, p_j)))$. If we sum complexities of such products for all transitions and places to which transitions give tokens, we get the aggregated complexity of all multiplications

$$O(\sum_{i,j}(n \cdot \mathit{size}(F(t_i, p_j)))) = O(n \cdot \sum_{i,j} \mathit{size}(F(t_i, p_j))) = O(n^2)$$

. □

Proposition 2.11
For given $L = \text{NORM}_Q$, changes δ_t^L caused by all transitions can be computed in time $O(n^2)$.

Proof: For each transition t, the value δ_t^L is computed using expression 2.1 from Subsection 2.3.3. If some c_j is infinite then δ_t is infinite too. Hence we do computation of the sum only for finite values. The complexity of additions is dominated by the complexity of multiplications $c_j \cdot F(t, p_j)$. Each such product is computed only once. From Proposition 2.9 we know that each c_j is in $O(n)$. Each $F(t, p_j)$ is used only once and is part of our representation of BPP. Hence we can similarly as in Proposition 2.10 for coefficients deduce that aggregated complexity of all multiplications is in $O(n^2)$, from which the result follows. □

Lemma 2.12
The algorithm adds one L to $\mathcal{L}$ in time $O(n^2)$.

Proof: As follows from Propositions 2.10 and 2.11, the running time of steps 1 and 2 is $O(n^2)$. The running time of step 3 is also $O(n^2)$ using one of standard algorithms for lexicographic sorting of strings (see [1, 36]), because values of δ_t can be represented as binary numbers, i.e., as strings of 0 and 1. Also the running time of step 4 is $O(n^2)$ if it is implemented carefully. □

Theorem 2.13
There is an algorithm solving BPP-FS-BISIM *with running time $O(n^4)$.*

Proof: We have described the algorithm. Lemmas 2.5 and 2.6 ensure the correctness of the algorithm. Since the addition of new L to $\mathcal{L}$ is the most time consuming operation of the algorithm, it follows from Lemmas 2.8 and 2.12 that the running time of the algorithm is $O(n^4)$. □

2.4 Bisimilarity of Normed BPP

It was mentioned in the Section 2.1 that BPP-BISIM is PSPACE-complete and in the Section 2.3 that a subproblem BPP-FS-BISIM of this problem where one of two systems is finite-state can be decided in $O(n^4)$. Now we will discuss another subproblem of BPP-BISIM, the nBPP-BISIM problem. There are two (possibly infinite state) BPP systems given at the instance but both of them must be normed. If we take, as usual, disjoint union of those two systems, our problem can be formulated as follows:

Problem: nBPP-BISIM

INSTANCE: A normed BPP system Δ, markings $M_1, M_2 \in \mathcal{M}_\Delta$.

QUESTION: Is $M_1 \sim M_2$?

Hirshfeld, Jerrum and Moller [15] developed a specific algorithm for nBPP-BISIM which works in polynomial time; they did not analyze the degree of the polynomial. The straightforward analysis leads to something like $O(n^5)$, maybe $O(n^4)$ could be possible with some deep insight.

The algorithm presented in this subsection was published in [17]. It is a more detailed version of Jančar's algorithm from [16] specialized only for the case of nBPP which enables worst case estimation time $O(n^3)$. The result holds (even) for the case when the numbers in specification of F and in initial marking (or equivalently the numbers of occurrences of variables in the right-hand sides of rules and in the initial sequence) are given in binary.

The result presented in Section 2.3 was published later than algorithm for deciding bisimilarity of normed BPP and it was using some complexity estimations from it. But algorithm for bisimilarity between BPP and FS uses several subprocedures in (just slightly) more general version and this generalization does not affect its time complexity. Therefore the section with more general results was presented first in the previous section and in this section we will make references to it and point on possible simplifications caused by using it only on normed systems.

Our main algorithm again performs a *stepwise decomposition* of the set *Tr* of transitions, i.e., it constructs a sequence of decompositions of *Tr*, where each new decomposition refines the old one.

For each subset (a decomposition class) $T' \subseteq Tr$, notation $\mathrm{Pre}(T')$ is used for the set $\{p \mid p = \textsc{pre}(t), t \in T'\}$.

The process starts with the (initial) decomposition according to action labels: transitions t_1, t_2 are in the same class iff $l(t_1) = l(t_2)$.

The iterated step of the main algorithm refines a current decomposition of Tr according to the changes which the rules cause on the functions $\text{NORM}_{\text{Pre}(T')}$, for all current decomposition classes T'. (For the initial decomposition we can observe that $\text{NORM}_{\text{Pre}(T')}$ is dd_a for the respective action a.)

This surely finishes, with a decomposition denoted *decomp*(Tr). Results of [16] guarantee that M_1 and M_2 are bisimilar iff $\text{NORM}_{\text{Pre}(T')}(M_1) = \text{NORM}_{\text{Pre}(T')}(M_2)$ for each class T' in *decomp*(T).

It is useful to realize that the *decomposition problem* is the crucial problem for us; the bisimilarity problem can be easily reduced to it:

Having a BPP system and two states M_1, M_2 we can add two fresh places p_1, p_2 and transition t_1, t_2 such that $\text{PRE}(t_1) = p_1$, $\text{PRE}(t_2) = p_2$, $F(t_1, p) = M_1(p)$, $F(t_2, p) = M_2(p)$ for all $p \in P$ and $l(t_1) = l(t_2) = a$ for any chosen action a. It is clear that $M_1 \sim M_2$ in the original system iff $M_{p_1} \sim M_{p_2}$ in the new system (recall that M_p means marking where $M(p) = 1$ and $M(p') = 0$ for all $p' \in p$, $p' \neq p$). Moreover, it can be readily verified that $M_{p_1} \sim M_{p_2}$ iff t_1, t_2 are in the same class of *decomp*(Tr).

For complexity analysis, we have to make precise the way of presenting (normed) BPP-systems and determining their size w.r.t. which the complexity is measured. We will generally assume that a (normed) BPP-system is encoded in the same way as in the previous section, i.e., as a list of places and transitions, where the encoding of each transition t contains a list of all $p \in P$ for which $F(t, p) > 0$ together with values $F(t, p)$. We assume that numbers are encoded in binary.

For computation of coefficients of the (linear) function NORM_Q, the algorithm from Figure 2.4 (in Section 2.3) can be used. The only difference is that for normed BPP systems we do not need a row

```
**if** $c_{p_{min}} = \omega$ **then break**;
```

The algorithm will work with it, but the condition will never be satisfied, hence the break will never be performed.

Propositions 2.9 and 2.10 presented in Section 2.3 hold in the case of normed BPPs as well. It means that the size of each coefficient of norm function is in $O(n)$ and all coefficients of a norm function are computed in time $O(n^2)$.

```
Compute $\mathcal{T} = \{T_1, T_2, \dots T_p\}$ as the decomposition of the set $Tr$
    according to the action labels.
Let $UIS$ contain all (different) sets Pre($T_1$), Pre($T_2$), ... , Pre($T_p$)
$PIS := \emptyset$
while $UIS \neq \emptyset$ do
    for each $Q \in UIS$ do
        compute all coefficients $c_Y$ of $\textsc{norm}_Q$
        for each transition $t$ appearing in a non-singleton class of $\mathcal{T}$ do
            $\delta(t) := -c_{\textsc{pre}(t)} + \sum_{p \in P} c_p F(t,p)$
        end for
        decompose each (non-singleton) class in $\mathcal{T}$
            according to the computed values $\delta(t)$
        let $\mathcal{T}$ refer to the newly arisen decomposition
    end for
    $PIS := PIS \cup UIS$
    $UIS := \emptyset$
    for each (newly arisen) class $T'$ of $\mathcal{T}$ do
        if Pre($T'$) $\notin PIS$ then $UIS := UIS \cup \{\text{Pre}(T')\}$
    end for
end while
```

Figure 2.5: The decomposition algorithm

We already sketched at the beginning of this section the ideas of the main algorithm decomposing the set Tr of transitions. The algorithm is show on Figure 2.5 and uses the following data structures, with the intended meanings

$\mathcal{T}$ ... a decomposition of the set Tr of all transitions

UIS ... a set of unprocessed (important) sets of places (the norms of such sets correspond to DD-functions)

PIS ... a set of processed (important) sets of places (for each Q here, the current decomposition $\mathcal{T}$ already separates each two transitions which cause different changes on $\textsc{norm}_Q$)

Theorem 2.14
The decomposition algorithm for normed BBPs runs in $O(n^3)$. (Hence bisimilarity for nBPP is decidable in $O(n^3)$.)

Proof: The Fact 2.7 implies that at most $2 \cdot |Tr|$ subsets of Tr can appear in the stepwise decomposition performed by the algorithm. This means that only $O(n)$ subsets Q of variables are processed (and put in PIS).

By Proposition 2.10, coefficients of each $\textsc{norm}_Q$ can be computed in $O(n^2)$. For each $\textsc{norm}_Q$, the aggregated complexity of computing the changes δ_r and the respective refinement of decomposition $\mathcal{T}$ can be also shown to be in $O(n^2)$ (similarly as in Section 2.3.4).

We can thus readily derive that the decomposition algorithm runs in time $O(n^3)$. □

2.5 Regularity of BPP

In the section 2.3 we have discussed problem of deciding bisimilarity between a given BPP system and a given finite-state system. It can be also interesting to know whether there exists some (not specified) finite state system bisimilar with given BPP system. In this section we will concern with this problem, called *regularity of BPP*:

Problem: BPP-REG

INSTANCE: BPP process (M_0, Δ).

QUESTION: Is (M_0, Δ) regular ?

This problem was known to be PSPACE-hard ([37]). In [22] was published an algorithm for this problem working in PSPACE which concludes PSPACE-completeness. In the following, the mentioned polynomial space algorithm will be described.

In this subsection we assume some fixed BPP system $\Delta = (P, Tr, \textsc{pre}, F, l)$ from the instance of BPP-REG.

According to [16], there is an algorithm working in polynomial space generating a set $\mathcal{D} \subseteq \mathcal{P}(P)$ such that for every markings M, M' is $M \sim M'$ iff $\textsc{norm}_Q(M) = \textsc{norm}_Q(M')$ for all sets $Q \in \mathcal{D}$. In the rest of the section we denote this set $\mathcal{D}_\Delta$.

We define

$$reach_\Delta(M) = \{M' \mid M \longrightarrow^* M' \text{ in } \Delta\}\,.$$

(We often write *reach* instead of $reach_\Delta$ when Δ is clear from context.)

We say a function $d : \mathcal{M}_\Delta \to \mathbb{N}_\omega$ is *bounded on* (M_0, Δ) if there is some $k \in \mathbb{N}$ such that $d(M) \leq k$ or $d(M) = \omega$ for each $M \in \mathit{reach}(M_0)$ (i.e., d takes only finitely many different values on markings from $\mathit{reach}(M_0)$).

Proposition 2.15
A BPP process (M_0, Δ) is regular iff d_Q is bounded on (M_0, Δ) for each $Q \in \mathcal{D}_\Delta$) .

Proof: ($\Rightarrow$) If (M_0, Δ) is regular then there is some FS process (q_0, Δ_{fs}) such that $M_0 \sim q_0$. The system Δ_{fs} has k states, so the distance from any state of Δ_{fs} to any other state is either less than k or ω. All markings from $\mathit{reach}(M_0)$ can be mapped to bisimilar states of Δ_{fs} (and markings from $\mathcal{M}(\Delta) - \mathit{reach}(M_0)$ are not reachable from markings in $\mathit{reach}(M_0)$) so the values of $d_Q(M)$ must be also less then k or ω for any $M \in \mathit{reach}(M_0)$ and $Q \in \mathcal{D}_\Delta$.

($\Leftarrow$) Since $\mathcal{D}_\Delta$ is finite it is obvious that if d_Q for $Q \in \mathcal{D}_\Delta$ takes only finitely many values on the elements of $\mathit{reach}(M_0)$ then the number of non-bisimilar markings in $\mathit{reach}(M_0)$ is finite. $\square$

Since all sets from $\mathcal{D}_\Delta$ can be enumerated (one by one) using only polynomial space, all we need to obtain a polynomial space algorithm is a procedure that in polynomial space tests for given $Q \in \mathcal{D}_\Delta$ whether d_Q is bounded on (M_0, Δ) or not. In fact, this test can be done in polynomial time.

Proposition 2.16
Given a set $Q \subseteq P$, it can be tested in polynomial time (wrt the size of (M_0, Δ)) if d_Q is bounded on (M_0, Δ).

Proof: Let $d_Q(M) = c_1x_1 + c_2x_2 + \cdots + c_kx_k$.

Obviously, we can ignore markings M where $d_Q(M) = \omega$ and consider only markings where $d_Q(M)$ is finite. Let

$$RF = \{M \in \mathit{reach}(M_0) \mid d_Q(M) < \omega\}$$

be the set of such reachable markings.

Note that d_Q is unbounded on (M_0, Δ) iff there is some $p \in P$ such that:

1. $0 < c_p < \omega$, and
2. the set $\{M(p) \mid M \in RF\}$ is infinite.

(Otherwise on each place p such that $0 < c_p < \omega$ there is some bound and the maximal value of d_Q on markings from RF can be bounded by some constant.)

If $d_Q(M_0) = \omega$ then $RF = \emptyset$ and the part (2) of the condition does not hold. So assume that $d_Q(M_0) < \omega$. In this case, the part (2) of the condition holds iff p_i is unbounded in (M_0, Δ') where Δ' is the BPP system obtained from Δ by removing each transition $t \in Tr$ such that $\delta_Q(t) = \omega$. (Just note that $reach(M_0, \Delta') = RF$.)

The test if p is unbounded in (M_0, Δ') can be easily done in polynomial time and we can perform this test for each p such that $0 < c_p < \omega$. □

Theorem 2.17
The problem BPP-REG *is* PSPACE-*complete.*

Proof: The polynomial space algorithm for BPP-REG works as follows: it generates all $Q \in \mathcal{D}_\Delta$ one by one (in polynomial space) and tests each of those sets whether d_Q is bounded on (M_0, Δ).

PSPACE-hardness of BPP-REG was shown by Srba [37] □

2.6 Bisimilarity between Normed BPP and Normed BPA

In this section we present a polynomial time algorithm for deciding bisimilarity between normed BPA and normed BPP processes. This result was published at conference [19] and in extended and improved version in journal [18].

2.6.1 Introduction

One long-standing open problem is the decidability question for the class PA (process algebra), which comprises "context-free" rewrite systems using both sequential and parallel composition. For the subcase of *normed* PA, a procedure working in doubly-exponential nondeterministic time was shown by Hirshfeld and Jerrum [13].

More is known about the "sequential" subclass called BPA (Basic Process Algebra) and the "parallel" subclass called BPP (Basic Parallel Processes) (which we discussed in previous sections). In the case of BPA, the best known algorithm for deciding bisimilarity seems to have doubly-exponential upper bound [7, 6]; the problem is known to be PSPACE-hard [38]. A polynomial-time algorithm for *normed* BPA was shown in [14] (with an upper bound $O(n^{13})$); more recently, an algorithm with running time in $O(n^8 \mathit{polylog}\ n)$ was shown in [32].

The most difficult part of the above mentioned algorithm for normed PA [13] deals with the case when (a process expressed as) sequential composition is bisimilar to (a process expressed as) parallel composition. A basic subproblem is to analyze when a BPA process is bisimilar to a BPP process. Černá, Křetínský, Kučera [43] have shown that this subproblem is decidable in the *normed* case; their suggested algorithm is exponential. Decidability in the general (unnormed) case was shown in [20], without providing any complexity bound.

In [19] and [19] we revisited the normed case, and we presented a *polynomial* algorithm deciding whether a given normed BPA process α is bisimilar to a given normed BPP process M. This result will be described in what follows.

One part of our algorithm (see subsection 2.6.3) is a new algorithm,based on *dd*-functions, which transforms the normed BPP process M into a special form, which we call "prime form", where bisimilarity coincides with equality. Time complexity of this transformation is $O(n^3)$. Such a transformation could be based on the prime decompositions from [15] but with worse complexity (which was, in fact, not analyzed in [15]).

A main idea of our algorithm is to derive a polynomial bound on a "finite-state core" of the transition system generated by the (transformed) BPP process M (see Subsection 2.6.4). If the size of the constructed finite-state core exceeds the derived bound, our decision algorithm answers negatively; otherwise it constructs a BPA process α' which is bisimilar to M, and the final step is to decide if the BPA processes α and α' are bisimilar.

To derive polynomiality, the mentioned final step can be handled by referring to [14] or [32]. To get a better complexity upper bound, namely $O(n^7)$, we suggested a simple self-contained algorithm, which exploits the fact that α' is "almost" a finite-state process (Subsection 2.6.6, it is not published in [19], only in [18]).

It can be interesting to know, if for a given BPP process there exists a bisimilar BPA process. A polynomial time algorithm testing this was shown in [43] with no bound on the polynomial degree. Our algorithm tests it for a given BPP as well and in time $O(n^3)$.

In the Section 2.3 we discussed bisimilarity between a BPP and a finite state process. For a similar problem of bisimilarity between a normed BPA and a FS we get an algorithm with running time $O(n^4)$ as a side result of our algorithm for bisimilarity between normed BPA and normed BPP. Polynomiality of this problem was already shown by Kučera, Mayr [29]. In fact, they provided an $O(n^{12})$ algorithm for the more general case of *weak* bisimilarity; the complexity for the special case of (strong) bisimilarity was not analyzed.

2.6.2 Definitions and Simple Observations

Our central problem is defined as follows:

Problem: nBPA-nBPP-BISIM

INSTANCE: A normed BPA-process (α_0, Σ), a normed BPP-process (M_0, Δ).

QUESTION: Is $\alpha_0 \sim M_0$ (in the disjoint union of $\mathcal{S}_\Sigma$ and $\mathcal{S}_\Delta$) ?

As the size n of an instance of nBPA-nBPP-BISIM we understand the number of bits needed for its (natural) presentation; in particular we consider the numbers $F(t,p)$ in Δ and the numbers in M_0 to be written in binary.

In the rest of this section we assume a fixed nBPA Σ and a fixed nBPP Δ.

We will later use the following straightforward propositions.

Proposition 2.18
The norms $\textsc{norm}(X)$, $\textsc{norm}(p)$ *for* $X \in V_\Sigma$, $p \in P_\Delta$ *can be written in* $O(n)$ *bits, thus all of them together in* $O(n^2)$ *bits. All these norms can be computed in time* $O(n^3)$.

Proposition 2.19
For every $Q \subseteq P$ *and* $t \in Tr$ *there is* $\delta \in \mathbb{N}_{-1}$ *such that* $M \xrightarrow{t} M'$ *implies* $\textsc{norm}_Q(M') = \textsc{norm}_Q(M) + \delta$ *(for all* M, M'*).*

A place p is called a *single final place*, an SF-place, if all transitions that take a token from p are of the form $p \xrightarrow{a} p^k$, $k \geq 0$ (they can only put tokens back to p). It is easy to see that $\textsc{norm}(p) = 1$ for every SF-place p (since Δ is normed). We say that p is a non-SF-place if it is not an SF-place.

2.6.3 Normed BPP Systems in Prime Form

We say that a *BPP net* Δ is *in the prime form* if bisimilarity coincides with identity on the generated LTS, i.e., $M \sim M'$ iff $M = M'$. (In this case, each place p is a "prime" since it is not equivalent to a composition of other places.) The prime form is technically convenient for developing our main algorithm; this subsection shows a relevant transformation (Theorem 2.26).

It follows from the unique decomposition results in [15] that for each normed BPP system Δ there is an equivalent normed BPP system Δ' in the prime form, and that Δ' can be constructed from Δ in a polynomial time using the algorithm, described in [15], which computes certain prime decompositions of BPP-variables (i.e., BPP-net places); it is a polynomial time algorithm but its precise complexity has not been analyzed. We proceed in another way, based on the *dd*-functions, which yields a transformation with time complexity $O(n^3)$.

The main idea can be sketched as follows. Given a normed BPP system $\Delta = (P, Tr, \textsc{pre}, F, \mathcal{A}, l)$, let $T_a \subseteq Tr$ be the set of transitions with label $a \in \mathcal{A}$. It is clear that $M \sim M'$ implies that the distance to disabling T_a is the same in both M and M'; by this distance in M we mean the length of the shortest w such that $M \xrightarrow{w} M_1$ and all $t \in T_a$ are disabled in M_1. In other words, we must have $\textsc{norm}_{\textsc{pre}(T_a)}(M) = \textsc{norm}_{\textsc{pre}(T_a)}(M')$ when $M \sim M'$. ($\textsc{pre}(T) = \{\textsc{pre}(t) \mid t \in T\}$.) Now suppose, e.g., that $T \subseteq T_a$ consists of all transitions with label a such that performing any $t \in T$ changes the norm wrt $\textsc{pre}(T_a)$ by $+3$ and the norm wrt $\textsc{pre}(T_b)$ by -1, for some $b \in \mathcal{A}$ ($M_1 \xrightarrow{t} M_2$ implies $\textsc{norm}_{\textsc{pre}(T_a)}(M_2) = \textsc{norm}_{\textsc{pre}(T_a)}(M_1) + 3$ and $\textsc{norm}_{\textsc{pre}(T_b)}(M_2) = \textsc{norm}_{\textsc{pre}(T_b)}(M_1) - 1$). Then $M \sim M'$ necessarily implies $\textsc{norm}_Q(M) = \textsc{norm}_Q(M')$ for $Q = \textsc{pre}(T)$. These observations have been refined in [16] to devise an algorithm for general BPP, which was then instantiated to normed BPP in [17].

Given a normed BPP system $\Delta = (P, Tr, \textsc{pre}, F, \mathcal{A}, l)$, of size n, the algorithm from [17] finishes in time $O(n^3)$ and constructs a partition

$$\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$$

of the set Tr of transitions; denoting $d_i(M) = \textsc{norm}_{\textsc{pre}(T_i)}(M)$, it holds that

$$M \sim M' \text{ iff } d_i(M) = d_i(M') \text{ for all } i = 1, 2, \ldots, m\,.$$

Moreover, each class T_i is characterized by its unique pair (a_i, δ_i) where a_i

is the label of all $t \in T_i$ and

$$\delta_i = (\delta_{i1}, \delta_{i2}, \ldots, \delta_{im})$$

is the vector in $(\mathbb{N}_{-1})^m$ capturing the following change, for any M, M':

$$\text{if } M \xrightarrow{t} M' \text{ for } t \in T_i \text{ then } d(M') = d(M) + \delta_i$$

where $d(M)$ denotes the vector $(d_1(M), d_2(M), \ldots, d_m(M))$. For convenience, we say *transition (of the type)* t_i when meaning any transition $t \in T_i$.

Using similar thoughts as those for deriving Proposition 2.18, we can obtain the following fact (proven in detail also in [17]).

Proposition 2.20
Each δ_{ij} can be written in space $O(n)$, and thus all pairs (a_i, δ_i) together in space $O(n^3)$.

Due to the normedness, for every class T_i ($i \in \{1, 2, \ldots, m\}$) there is at least one transition t_j ($j \in \{1, 2, \ldots, m\}$) which decreases d_i (when t_j is enabled in M, which also entails $d_i(M) > 0$); this is concisely captured by the next proposition.

Proposition 2.21
$\forall i \exists j : \delta_{ji} = -1$.

We say that t_i is a *key transition* if it decreases some component of d, i.e. some d_j. Formally we define

$$\text{KEY} = \{i \mid \delta_{ij} = -1 \text{ for some } j\}.$$

Proposition 2.22
$\forall i \in \text{KEY} : \delta_{ii} = -1$.

Proof: If t_i (an element of T_i) decreases some d_j then for each M there is the greatest ℓ such that $M \xrightarrow{(t_i)^\ell}$. The last firing of t_i necessarily decreases d_i. Hence $\delta_{ii} = -1$. □

Thus for each $i \in \text{KEY}$, $d_i(M)$ is the greatest ℓ such that $M \xrightarrow{(t_i)^\ell}$. (A shortest way to disable transitions in T_i is to fire them as long as possible.)

We say that t_i *reduces* t_j iff $\delta_{ij} = -1$. Formally we define the following relation RED on KEY:

$$\text{for } i, j \in \text{KEY we put } i \text{ RED } j \text{ iff } \delta_{ij} = -1\,.$$

Proposition 2.23
RED *is an equivalence relation.*

Proof: Reflexivity follows from Proposition 2.22.

To show symmetry, assume $i, j \in \text{KEY}$ (so $\delta_{ii} = \delta_{jj} = -1$) such that $\delta_{ij} = -1$ but $\delta_{ji} \geq 0$ (for the sake of contradiction). Then firing t_j from M with $d_i(M) > 0$ as long as possible results in M' with $d_j(M') = 0$ and $d_i(M') > 0$. Thus $M' \xrightarrow{t_i}$, which is a contradiction since d_j can not be decreased.

Transitivity follows similarly: Assume $i, j, k \in \text{KEY}$ and suppose i RED j and j RED k but $\neg(i \text{ RED } k)$. So all $\delta_{ii}, \delta_{jj}, \delta_{kk}, \delta_{ij}, \delta_{ji}, \delta_{jk}, \delta_{kj}$ are -1 but $\delta_{ik} \geq 0$. Starting from M with $d_k(M) > 0$, we fire t_i as long as possible and thus get M' with $d_i(M') = d_j(M') = 0$ and $d_k(M') > 0$. Thus $M' \xrightarrow{t_k}$, which is a contradiction since d_j can not be decreased. □

The following two propositions will help us later to show the size of the constructed BPP in the prime form equivalent to a given one. To simplify the notation, we put $Q_i = \text{PRE}(T_i)$ and note that $d_i(M) = \text{NORM}_{Q_i}(M)$.

Proposition 2.24
There are at most $|P|$ classes of equivalence RED.

Proof: Let $T_N \subseteq Tr$ be some set of norm reducing transitions such that for each $p \in P$ there is exactly one $t \in T_N$ with $\text{PRE}(t) = p$ (i.e. $|T_N| = |P|$). It is thus sufficient to show that for each class C of RED there exists $i \in C$ and $t \in T_N \cap T_i$. Since the net can be emptied by using only the transitions from T_N, for each $i \in \text{KEY}$ there is $t \in T_N$ which decreases the norm wrt Q_i; thus $t = t_j$ for some $j \in \text{KEY}$. Hence j RED i, and therefore j belongs to the class of i. □

Proposition 2.25
Let T_z be a class of the partition $\mathcal{T}$ containing non-key transitions. The number of classes C of equivalence RED *such that t_i decreases d_z for some $i \in C$ is at most $|T_z|$.*

Proof: Let $T_{k_1}, T_{k_2}, \ldots, T_{k_x}$ be all classes of the partition $\mathcal{T}$ such that t_{k_i} decreases d_z. Let $T_K = T_{k_1} \cup \ldots \cup T_{k_x}$ and $Q_K = Q_{k_1} \cup \ldots \cup Q_{k_x}$. Since the transitions from T_K have to be able to decrease d_z to 0 (to empty the set Q_z), it holds $Q_z \subseteq Q_K$. Each transition from T_{k_i} reduces d_z, and so its input place is from Q_z. It follows that $Q_{k_i} \subseteq Q_z$ for each k_i, and so $Q_K \subseteq Q_z$. Therefore $Q_K = Q_z$ and thus $|Q_K| \leq |T_z|$.

To complete the proof, we need to show that the number of classes of RED containing some k_i is at most $|Q_K|$. The idea is similar as in the proof of Proposition 2.24. We can take some set $T_N \subseteq T_K$ such that for each $p \in Q_K$ there is exactly one transition $t \in T_N$ for which $\text{PRE}(t) = p$. Note that each $t \in T_N$ reduces d_z and $|T_N| = |Q_K|$. Using only the transitions from T_N, the set Q_K can be emptied and all $d_{k_1}, d_{k_2}, \ldots, d_{k_x}$ set to 0. For each $i \in \{k_1, k_2, \ldots, k_x\}$ there is $t \in T_N$ which decreases the norm wrt Q_i. It follows from the definition of T_N that $t = t_j$ for some $j \in \{k_1, k_2, \ldots, k_x\}$. Hence j RED i, and therefore j belongs to the class of i. □

Theorem 2.26
There is an algorithm, with time complexity $O(n^3)$, which transforms a given normed BPP system $\Delta = (P, Tr, \text{PRE}, F, \mathcal{A}, l)$ into $\Delta' = (P', Tr', \text{PRE}', F', \mathcal{A}, l')$ in the prime form, and any given state (marking) M of Δ into M' of Δ' such that $M \sim M'$. Moreover, $|Tr'| \leq |Tr|, |P'| \leq |P|$, and Δ' is represented in space $O(n^3)$.

Proof: In the first phase we compute the partition $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ as discussed above. We easily verify that $Q_i = Q_j$ for $i, j \in \text{KEY}$ iff i RED j (and so j RED i).

The crucial idea is that Δ' will have a place p_C for each class C of the equivalence RED. For any M of Δ, the number $M'(p_C)$ will be equal to $\text{NORM}_{Q_i}(M)$ for each $i \in C$. Proposition 2.24 implies $|P'| \leq |P|$.

For every $i \in \text{KEY}$, we add a transition t'_i in Δ' such that $\text{PRE}(t'_i) = p_C$ where $i \in C$; t'_i is labelled with a_i and it realizes the (nonnegative) change on the other places $p_{C'}$ according to δ_i (restricted to KEY). The number of transitions of Δ' added in this step is at most equal to the number of key transitions of Δ.

A non-key transition t_i (with $\delta_i \geq (0, 0, \ldots, 0)$) is enabled precisely when a (key) transition decreasing d_i is enabled (recall Proposition 2.21). Thus for each p_C where C contains j with $\delta_{ji} = -1$ we add a transition t with label a_i and $\text{PRE}(t) = p_C$ which (gives a token back to p_C and) realizes the change δ_i (restricted to KEY). Proposition 2.25 implies that at most $|T_i|$ transitions are added to Δ' for every class T_i of non-key transitions.

A transition t can possibly increase all d_i. Therefore, an equivalent transition

t' can have $|P'|$ output edges. The multiplicity of each output edge can be written in space $O(n)$ (recall Proposition 2.20).

Summing up, $\Delta' = (P', Tr', \textsc{pre}', F', \mathcal{A}, l')$ can be constructed in time $O(n^3)$ and represented in space $O(n^3)$. The correctness of the construction is obvious. □

In the following text we only consider BPP systems in the prime form, if not stated otherwise.

2.6.4 A Bound on the Number of "not-all-in-one-SF" Markings

In this section we prove the following theorem.

Theorem 2.27
Assume a normed BPA system Σ, with the set V of variables, and a normed BPP system Δ in the prime form, with the set P of places. The number of markings M of Δ such that $\alpha \sim M$ for some $\alpha \in V^+$ and M does not have all tokens in one SF-place is at most $4y^2$, where $y = \max\{|V|, |P|\}$.

We start with a simple observation and then we bound the total number of tokens in the markings mentioned in the theorem.

Proposition 2.28
If $A\alpha \sim M$ where $\alpha \in V^*$ and $|Car(M)| \geq 2$ then $\textsc{norm}(A) \geq 2$.

Proof: From M with $|Car(M)| \geq 2$ we can obviously perform two different norm-reducing steps resulting in two different, and thus nonbisimilar, markings. On the other hand, any $A\alpha$ with $\textsc{norm}(A) = 1$ has a single outcome (namely α) of any norm-reducing step. □

Proposition 2.29
If $|Car(M)| \geq 2$ and $\alpha \sim M$ for $\alpha \in V^+$ then $Tok(M) \leq |V|$.

Proof: In fact, we prove a stronger proposition. To this aim, we order the variables from V into a sequence $A_1, A_2, \ldots, A_{|V|}$ so that $\textsc{norm}(A_i) \leq \textsc{norm}(A_j)$ for $i \leq j$. We now show the following claim: if $A_i\alpha \sim M$, where $|Car(M)| \geq 2$ (and $\alpha \in V^*$), then $Tok(M) \leq i$.

For the sake of contradiction, suppose a counterexample $A_i\alpha \sim M$, $Tok(M) \geq i+1$, for minimal i. Proposition 2.28 shows that $\textsc{norm}(A_i) \geq 2$, hence also

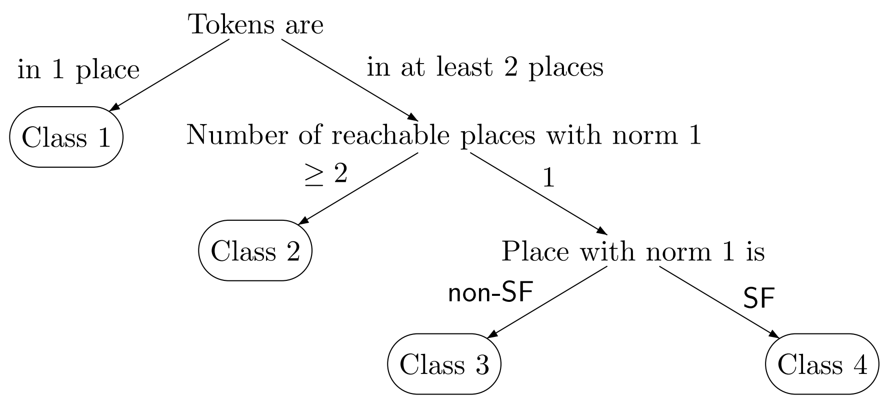


Figure 2.6: Partitioning of markings into 4 classes

$i \geq 2$ (since necessarily $\textsc{norm}(A_1) = 1$); therefore $\mathit{Tok}(M) \geq i{+}1 \geq 3$. There are two possible cases — $\mathit{Car}(M) = 2$ or $\mathit{Car}(M) \geq 3$. In the first case, at least one of the two marked places contains at least two tokens and so it can not be emptied in one step by a norm-reducing transition taking a token from this place, and it is obvious that the other marked place also remains marked after this step. In the second case, a norm-reducing step from an arbitrary marked place leads to a marking where at least two originally marked places remain marked. Hence there is at least one possible norm-reducing step $M \longrightarrow_R M'$ such that $|\mathit{Car}(M')| \geq 2$, $\mathit{Tok}(M') \geq i$. This step is matched by $A_i\alpha \longrightarrow_R A_j\beta\alpha$, $A_j\beta\alpha \sim M'$, where necessarily $\textsc{norm}(A_j) < \textsc{norm}(A_i)$ and thus $j < i$. This contradicts the minimality of our counterexample. $\square$

From the definition of a non-SF-place follows that a token from any such place may be moved (not necessarily by a norm-reducing step) to another place in such a way that the total number of tokens is not decreased by this step. From this fact and from the previous proposition, we get the following corollary.

Corollary 2.30
If $\alpha \sim M$ then $M(p) \leq |V|$ for every non-SF-place p.

We now partition the markings in the theorem into four classes (possibly tree-like description on Figure 2.6 can be more comprehensible):

Class 1. Markings M with all tokens in one (non-SF) place ($|Car(M)| = 1$).

Class 2. Markings M with $|Car(M)| \geq 2$ where at least two different places with norm 1 are reachable; this necessarily means $M \longrightarrow^* M'$ for some M' satisfying $M'(p_1) \geq 1$, $M'(p_2) \geq 1$ for some $p_1 \neq p_2$ and $\textsc{norm}(p_1) = \textsc{norm}(p_2) = 1$.

Class 3. Markings M with $|Car(M)| \geq 2$ and with exactly one reachable ("sink") place p with norm 1, where p is a non-SF-place.

Class 4. Markings M with $|Car(M)| \geq 2$ and with exactly one reachable ("sink") place p with norm 1, where p is an SF-place.

We will show that each class contains at most y^2 markings by which we prove the theorem. (In fact, our bound is a bit generous, allowing to avoid some technicalities.)

Proposition 2.31
The number of markings in Class 1 is bounded by $|V| \cdot |P| \leq y^2$.

Proof: According to Corollary 2.30 there can be at most $|V|$ tokens in any non-SF-place and there are at most $|P|$ non-SF-places. It follows that Class 1 contains at most $|V| \cdot |P| \leq y^2$ markings. $\square$

Proposition 2.32
If $\alpha \sim M$ for M from Class 2 then $\alpha = A$ for some $A \in V$. Thus the number of markings in Class 2 is at most $|V| \leq y$.

Proof: For the sake of contradiction, suppose $A\alpha \sim M$ where $\alpha \in V^+$ and M is from Class 2. We take a counterexample with the minimal length ℓ of a sequence v such that $M \xrightarrow{v} M'$ where $M'(p_1) \geq 1$, $M'(p_2) \geq 1$ for two different p_1, p_2 with norm 1. We note that $\textsc{norm}(A) \geq 2$ by Proposition 2.28, and first suppose $\ell > 0$. It is easy to verify that there is a move $M \longrightarrow M''$, matched by $A\alpha \longrightarrow B\beta\alpha$, $B\beta\alpha \sim M''$, where $|Car(M'')| \geq 2$ and the respective length ℓ decreased; this would be a contradiction with the assumed minimality. Thus $\ell = 0$, which means $M(p_1) \geq 1$, $M(p_2) \geq 1$. But then M certainly allows $M \longrightarrow_R^* M_1$, $M \longrightarrow_R^* M_2$ where $\textsc{norm}(M_1) = \textsc{norm}(M_2) = \textsc{norm}(\alpha) \geq 1$ and $M_1 \neq M_2$, and thus $M_1 \not\sim M_2$. On the other hand, $A\alpha$ can offer only α as the result of matching such sequences; hence $A\alpha \not\sim M$. $\square$

Proposition 2.33
If $A\alpha \sim M$ for $\alpha \in V^+$ and M from Class 3 or 4 then $M \longrightarrow_R^* p^{\textsc{norm}(\alpha)}$ where p is the sink place. Thus $\alpha \sim p^{\textsc{norm}(\alpha)}$.

Proof: We prove the claim by induction on the norm $\textsc{norm}(A)$. Suppose $A\alpha \sim M$ as in the statement. Proposition 2.28 implies $\textsc{norm}(A) \geq 2$. M necessarily has a token in a place $p' \neq p$ with the least norm greater than 1. Performing a norm-reducing transition with this token corresponds to some $M \longrightarrow_R M'$, and this must be matched by $A\alpha \longrightarrow_R B\beta\alpha$, $B\beta\alpha \sim M'$, where $\textsc{norm}(B) < \textsc{norm}(A)$. Either $|Car(M')| = 1$, in which case necessarily $M' = p^{\textsc{norm}(B\beta\alpha)}$, or $|Car(M')| \geq 2$, and then $M' \longrightarrow_R^* p^{\textsc{norm}(\beta\alpha)}$ due to the induction hypothesis. Since obviously $p^{\textsc{norm}(B\beta\alpha)} \longrightarrow_R^* p^{\textsc{norm}(\beta\alpha)} \longrightarrow_R^* p^{\textsc{norm}(\alpha)}$, we are done. □

Proposition 2.34
If $A\alpha \sim M$ where $\textsc{norm}(\alpha) \geq 2$ then M is not from Class 3.

Proof: For the sake of contradiction, suppose $A\alpha \sim M$ with $\textsc{norm}(\alpha) \geq 2$, M from Class 3, i.e. M has exactly one reachable sink place p which is a non-SF-place. Further assume $\textsc{norm}(A)$ minimal possible; $\textsc{norm}(A) \geq 2$ by Proposition 2.28.

If there was a step $M \longrightarrow_R M'$ with $|Car(M')| \geq 2$, the matching $A\alpha \longrightarrow_R B\beta\alpha$ would lead to a contradiction with minimality of $\textsc{norm}(A)$. Since $|Car(M)| \geq 2$, the only remaining possibility is the following: $Tok(M) = 2$, $M(p) = 1$ and $M(p') = 1$ where $p' \longrightarrow_R p^k$ for $k = \textsc{norm}(A) + \textsc{norm}(\alpha) - 2 \geq 2$.

Since the sink place p is a non-SF-place, it must be in a cycle C with at least two places. Moving a token along C cannot generate new tokens, due to Corollary 2.30, so p' is not in C. On the other hand, C contains some p'' with $\textsc{norm}(p'') = 2$. Starting in M, we can move the token from p to p'', the norm being greater than $\textsc{norm}(M) = \textsc{norm}(A\alpha)$ along the way. For the resulting M' we obviously have $M' \longrightarrow_R^* M''$ for M'' satisfying $M''(p'') = 1$ and $\textsc{norm}(M'') = \textsc{norm}(\alpha)$. $A\alpha$ can match this only by reaching α but $\alpha \sim p^{\textsc{norm}(\alpha)}$ according to Proposition 2.33 and thus $\alpha \not\sim M''$. □

We can thus have $A\alpha \sim M$ for M from Class 3 only when $\textsc{norm}(\alpha) \leq 1$, and it is thus easy to derive the following corollary.

Corollary 2.35
The number of markings in Class 3 is at most $|V|^2 \leq y^2$.

Proposition 2.36
The number of markings in Class 4 is at most $|V| \cdot |P| \leq y^2$.

Proof: Let $A\alpha \sim M$ for M from Class 4, p being the respective SF-sink place. Using Proposition 2.33, we derive $\alpha \sim I^k$ where $k = \textsc{norm}(\alpha)$ and $I \in V$, $I \sim p$ (such I must exist since $M \longrightarrow^* p$). Thus $AI^k \sim M$ but $AI^k \not\sim I^m$ for any m since $I^m \sim p^m$ and $p^m \not\sim M$ (note that $p^m \neq M$ and Δ is in the prime form).

Since $M \longrightarrow^*_R p^m$ for some m, there must be a (shortest) norm-reducing sequence $A \xrightarrow{w} B\beta$ where $\beta \sim I^{\textsc{norm}(\beta)}$, $B \not\sim I^{\textsc{norm}(B)}$ but all norm-reducing transitions $B \xrightarrow{a} \gamma$ satisfy $\gamma \sim I^{\textsc{norm}(\gamma)}$. The sequence $A\alpha \xrightarrow{w} B\beta\alpha$ (where $B\beta\alpha \sim BI^{\textsc{norm}(\beta\alpha)}$) must be matched by some $M \xrightarrow{v} M'$ where M' does not have all tokens in p but every norm-reducing transition from M' results in M'' with all tokens in p; it follows that M' has a single token (so we have at most $|P|$ possibilities for M').

This easily implies that there are at most $|V| \cdot |P| \leq y^2$ markings in Class 4. □

2.6.5 Problem nBPA-nBPP-BISIM is in PTIME

In this section we describe a polynomial time algorithm for nBPA-nBPP-BISIM.

In Subsection 2.6.5 we specify conditions, which a normed BPP process (M_0, Δ) satisfies iff there exists some normed BPA process (α_0, Σ) such that $\alpha_0 \sim M_0$. The conditions can be easily checked in a time polynomial with respect to the size of (M_0, Δ). If (M_0, Δ) satisfies them, such (α_0, Σ) can be constructed but its size can be exponential with respect to the size of (M_0, Δ).

A basic idea of an algorithm for nBPA-nBPP-BISIM is to construct an nBPA process bisimilar to a given nBPP process (if it exists) and then to use some (polynomial time) algorithm for deciding if this constructed nBPA process is bisimilar to the nBPA process from the instance of nBPA-nBPP-BISIM. The complexity of such algorithm would be exponential in general, but in Subsection 2.6.5 we show how results from Section 2.6.4 can be applied to obtain a polynomial time algorithm.

Deciding if there exists nBPA process bisimilar to a given nBPP process

We start with some technical notions concerning unbounded places that will be useful for the characterization of an nBPP process, for which a bisimilar nBPA process exists.

We first note that if moving a token along a cycle C in a BPP system Δ generates new tokens in a place p and C is reachable (markable) from M_0 then p is *primarily unbounded* (in M_0). Any place which is unbounded is either primarily unbounded, or *secondarily unbounded*, which means reachable from a primarily unbounded place. Thus any unbounded place has at least one corresponding *pumping cycle*.

We say that an SF-place p is *growing* if there is a transition $p \xrightarrow{a} p^k$ for $k \geq 2$.

Lemma 2.37
For (M_0, Δ), Δ being a normed BPP in the prime form, there exists a normed BPA process (α_0, Σ) such that $\alpha_0 \sim M_0$, iff the following conditions hold:

1. *each non-SF-place is bounded,*
2. *there is no M such that $M_0 \longrightarrow^* M$, $|Car(M)| \geq 2$ and $M(p) \geq 1$ for some growing SF-place p,*
3. *each non-growing SF-place p is bounded.*

Proof: ($\Rightarrow$) If 1. is violated then we cannot have $\alpha_0 \sim M_0$ (for any Σ with a finite variable set V) due to Corollary 2.30. If 2. or 3. is violated then, for any $c \in \mathbb{N}$, $M_0 \longrightarrow^* M$ with $|Car(M)| \geq 2$ and $Tok(M) > c$. (Any pumping cycle for p in 3. contains $p' \neq p$.) Hence we cannot have $\alpha_0 \sim M_0$ due to Proposition 2.29.

($\Leftarrow$) Suppose we have an nBPP process (M_0, Δ) where the conditions 1.,2.,3. are satisfied. We show how an appropriate (α_0, Σ) can be constructed. Since all three conditions hold, the only unbounded places in (M_0, Δ) are growing SF-places. Moreover, if some growing SF-place p is reachable from M_0 then $Tok(M_0) = 1$ and each transition sequence reaching p just moves the token into p without creating new tokens on the way.

We can construct the usual reachability graph for M_0, with the exception that the "all-in-one-SF" markings p^k are taken as "frozen" – we construct no successors for them. The thus arising *basic LTS* is necessarily finite, and we can view its states as BPA-variables; each non-frozen marking M is viewed as a variable A_M, with the obvious rewriting rules.

To finish the construction, we introduce a variable I_p for each SF-place p together with appropriate rewriting rules.

More formally, for (M_0, Δ) we could construct nBPA system $\Sigma = (\mathcal{F} \cup \mathcal{I}, \mathcal{A}, \Gamma)$ where $\mathcal{F} = \{A_M \mid M \in \mathcal{M}_{uf}\}$ (where $\mathcal{M}_{uf} = \{M_1, M_2, \ldots, M_m\}$ is the set of non-frozen markings reachable from M_0), $\mathcal{I} = \{I_p \mid p \in P_{SF}\}$ (where $P_{SF} = \{p_1, p_2, \ldots, p_\ell\}$ is the set of SF-places of Δ), and Γ contains the corresponding rewriting rules.

Note that each rule in Γ is of one of the following three forms: $A_M \xrightarrow{a} A_{M'}$, $A_M \xrightarrow{a} (I_p)^k$, or $I_p \xrightarrow{a} (I_p)^k$, where $A_M, A_{M'} \in \mathcal{F}$, $I_p \in \mathcal{I}$, and $k \in \mathbb{N}$ (this includes also rules of the form $A_M \xrightarrow{a} \varepsilon$ and $I_p \xrightarrow{a} \varepsilon$). Configuration α_0 corresponding to M_0 will be A_{M_0} (or $(I_{p_0})^k$ when all k tokens in M_0 are in one SF-place p_0). Note that each configuration α reachable from α_0 is either of the form A_M or $(I_p)^k$, and we have $(\alpha_0, \Sigma) \sim (M_0, \Delta)$. □

We note that the conditions in Lemma 2.37 can be checked by straightforward standard algorithms, linear in the size of Δ in the prime form (which means $O(n^3)$ if Δ is not in the prime form). We thus have the following corollary.

Corollary 2.38
The problem to decide if a given normed BPP process (not necessarily in the prime form) is bisimilar to some (unspecified) normed BPA process can be solved in time $O(n^3)$.

Polynomial Algorithm for nBPA-nBPP-BISIM

Assume an instance of nBPA-nBPP-BISIM, i.e., nBPA process (α_0, Σ) and nBPP process (M_0, Δ). The polynomial algorithm for nBPA-nBPP-BISIM works as follows.

It first transforms (M_0, Δ) to bisimilar (M_0', Δ') where Δ' is in the prime form; recall Theorem 2.26. Note that nothing special is assumed about (α_0, Σ) and it is not transformed to any special form. The algorithm then starts to build nBPA Σ' for (M_0', Δ') as described in the proof of Lemma 2.37 by building the set $\mathcal{M}_{uf}$ of non-frozen states. If it discovers that the number of elements of $\mathcal{M}_{uf}$ exceeds $4y^2$, where y is the maximum of $\{|V_\Sigma|, |P_{\Delta'}|\}$, then the algorithm stops with the answer $\alpha_0 \not\sim M_0$; this is correct due to Theorem 2.27. Note that it is not necessary to test the conditions of

Lemma 2.37 explicitly in the algorithm because if any of these conditions is violated, the number of non-frozen markings is infinite, which means that the number of constructed elements of $\mathcal{M}_{uf}$ necessarily exceeds $4y^2$ and the algorithm stops with the correct answer.

Remark 2.39
Generally the size of Δ' is $O(n^3)$ in the size n of the nBPA-nBPP-BISIM-*instance. But since $|P_{\Delta'}| \leq |P_\Delta|$ (recall Theorem 2.26), the bound $4y^2$ is in $O(n^2)$.*

If the number of elements of $\mathcal{M}_{uf}$ does not exceed $4y^2$, the algorithm finishes the construction of Σ'. However, it does not construct Σ' explicitly but rather a succinct representation of it where the right hand sides of rules of the form $(I_p)^k$ are represented as pairs (I_p, k) where k is written in binary (note that $O(n)$ bits are sufficient for k).

Our aim is to apply the polynomial time algorithm from [14] or [32] to decide if $\alpha_0 \sim \alpha_0'$. However, there is a small technical difficulty since this algorithm expects "usual" nBPA, not nBPA in the succinct form described above. This can be handled by adding special variables I_p^1, I_p^2, I_p^4, I_p^8, $\ldots$ $I_p^{2^m}$ for each $I_p \in \mathcal{I}$ and sufficiently large m (in $O(n)$); the rules are adjusted in a straightforward way (note that there will be at most $O(m)$ variables on the right hand side of each rewriting rule after this transformation).

The size of the constructed nBPA is clearly polynomial with respect to the size of the original instance of the problem and the algorithm from [14] or [32] can be applied.

So we obtained our main theorem:

Theorem 2.40
There is a polynomial-time algorithm deciding whether $(\alpha_0, \Sigma) \sim (M_0, \Delta)$ where Σ is a normed BPA and Δ a normed BPP.

Since (α_0', Σ') is in a very special form (it is a finite state system (FS) extended with "SF-tails"), it is in fact not necessary to use the above mentioned general algorithm. Instead we can use a specialized and more efficient algorithm described in the next section.

2.6.6 An Algorithm Deciding nBPA-nBPP-BISIM in $O(n^7)$

The aim of this section is to provide a self-contained algorithm for nBPA-nBPP-BISIM. It is inspired by the ideas used, e.g., in the proofs in [29, 30, 32]; being tailored to our specific setting, the algorithm allows to derive the upper bound $O(n^7)$. In Subsection 2.6.6 we fix some notation and in Subsection 2.6.6 we deal with the simple subcase of the "single final" configurations. Section 2.6.6 can be seen as an adaptation of the bisimulation base construction from, e.g., [29, 30]. Section 2.6.7 recalls a useful fact on boolean equation systems, which was also used in [32]; the respective application to our case is described in Subsection 2.6.8. Subsection 2.6.9 then presents the overall algorithm.

Notation

Assume we have an nBPA process (α_0, Σ) and an nBPP process (M_0, Δ) (not necessarily in the prime form) from the instance of nBPA-nBPP-BISIM, and the nBPA process (α'_0, Σ') obtained from (M_0, Δ) as described in the previous section (with $V_{\Sigma'} = \mathcal{F} \cup \mathcal{I}$) stored using the succinct representation described above (the right hand sides of the form $(I_p)^i$ are stored as pairs (I_p, i) with i represented in binary).

In the rest of the section, we assume the following:

- n is the size (in bits) of the original instance of nBPA-nBPP-BISIM,
- m is the size of Σ (note that $m < n$, $|V_\Sigma| < m$, and m is greater than the sum of lengths of the right hand sides of the rules of Σ),
- $k = |V_{\Sigma'}| = |\mathcal{F}| + |\mathcal{I}|$,
- ℓ is the total number of the rules of Σ',

It is clear from the previous discussion that $|\mathcal{F}| \in O(n^2)$, $|\mathcal{I}| < n$, and $k \in O(n^2)$. Since each reachable configuration α of (α'_0, Σ') is bisimilar to some marking of Δ, the number of transitions enabled in α is bounded by the number of transitions of Δ, and so it is less than n. This means that $\ell \in O(n^3)$.

Recall that all reachable configurations of (α'_0, Σ') are either of the form A_M or $(I_p)^i$ ($A_M \in \mathcal{F}$, $I_p \in \mathcal{I}$). We denote the set of all such configurations by $\mathit{Conf}(\Sigma')$, i.e.,

$$\mathit{Conf}(\Sigma') = \mathcal{F} \cup \{(I_p)^i \mid I_p \in \mathcal{I},\, i \geq 0\}\,.$$

Without loss of generality we assume $\mathcal{I} \neq \emptyset$, which ensures that $\varepsilon \in \mathit{Conf}(\Sigma')$.

Let $V_{all} = V_\Sigma \cup V_{\Sigma'}$. We easily note that the values $\text{NORM}(X)$, $\text{NORM}(\alpha)$ for each $X \in V_{all}$, and each α such that $X \longrightarrow \alpha$ can be written in $O(n)$ bits.

Characterization of Configurations Bisimilar to $(I_p)^i$

The following proposition allows us to characterize the set of configurations from V_{all}^* bisimilar to $(I_p)^i$ where $I_p \in \mathcal{I}$ and $i \geq 0$.

Proposition 2.41
For each $I_p \in \mathcal{I}$ there is a set $Class(I_p) \subseteq V_{all}$ such that for each $\alpha \in V_{all}^*$ we have $\alpha \sim (I_p)^i$ iff $\alpha \in Class(I_p)^*$ and $\text{NORM}(\alpha) = i$.

Proof: We construct a set $Class(I_p)$ as the maximal subset of V_{all} such that each $X \in Class(I_p)$ can perform exactly the same actions with the same changes on norm as I_p, and can be rewritten only to variables from $Class(I_p)$ (i.e., $X \xrightarrow{a} \beta$ implies $\beta \in (Class(I_p))^*$, and $I_p \xrightarrow{a} (I_p)^i$ iff $X \xrightarrow{a} \beta$ for some $\beta \in (Class(I_p))^*$ such that $\text{NORM}(\beta) - \text{NORM}(X) = i - 1$). $\Box$

Note that the classes $Class(I_p)$ for $I_p \in \mathcal{I}$ can be easily computed in polynomial time and can be precomputed at the beginning. This gives us a fast (polynomial) test for checking if $\alpha \sim (I_p)^i$.

Bisimulation Base

We start with some observations leading to the technical notions defined below. Suppose we want to check if $\alpha \sim A_M$ for some $\alpha \in V_{all}^*$ and $A_M \in \mathcal{F}$ where $\alpha = X\alpha'$ for some $X \in V_{all}$. If $X\alpha' \sim A_M$ then any norm reducing sequence $X\alpha' \longrightarrow_R^* \alpha'$ must be matched by some norm reducing sequence $A_M \longrightarrow_R^* \beta$ such that $\alpha' \sim \beta$. Obviously, β is either of the form $A_{M'}$ (for some $A_{M'} \in \mathcal{F}$) or $(I_p)^i$ (for some $I_p \in \mathcal{I}$). Since $\alpha' \sim \beta$ and $\sim$ is a congruence, we have $X\beta \sim A_M$. On the other hand, from $X\beta \sim A_M$ and $\alpha' \sim \beta$ follows $X\alpha' \sim A_M$. So we see that $X\alpha' \sim A_M$ iff there is some $\beta \in Conf(\Sigma')$ such that $X\beta \sim A_M$ and $\alpha' \sim \beta$.

This allows us to construct a bisimulation base, i.e. a succinct representation of $\sim$ on pairs of (reachable) configurations of (α_0, Σ) and (α_0', Σ'). The base is a finite set (of polynomial size) containing some bisimilar pairs from which all other bisimilar pairs can be generated.

We start by defining (an overapproximation)

$$\begin{aligned}\mathcal{B}_0 &= \{(X\alpha, A) \mid X \in V_\Sigma,\ \alpha \in \mathit{Conf}(\Sigma'),\ A \in \mathcal{F},\ \textsc{norm}(X\alpha) = \textsc{norm}(A)\} \\ &\cup\ \{(\alpha, A) \mid \alpha \in \mathit{Conf}(\Sigma'),\ A \in \mathcal{F},\ \textsc{norm}(\alpha) = \textsc{norm}(A)\}\,.\end{aligned}$$

Note that $\mathcal{B}_0$ is finite since i in $(XI^i, A) \in \mathcal{B}_0$ is determined by X, I, A and the requirement $\textsc{norm}(XI^i) = \textsc{norm}(A)$ and i can be computed as $i = \textsc{norm}(A) - \textsc{norm}(X)$ (and similarly i in $(I^i, A) \in \mathcal{B}_0$). So $\mathcal{B}_0$ contains at most $(|V_\Sigma| + 1) \cdot (|\mathcal{F}| + |\mathcal{I}|) \cdot |\mathcal{F}| = O(mk^2)$ elements.

For each $\mathcal{B} \subseteq \mathcal{B}_0$ we define the set $\mathit{Closure}(\mathcal{B})$ as the least subset of $\{(\gamma\alpha, \alpha') \mid \gamma \in V_\Sigma,\ \alpha, \alpha' \in \mathit{Conf}(\Sigma')\}$ satisfying the following properties:

(1) $\mathcal{B} \subseteq \mathit{Closure}(\mathcal{B})$.

(2) Let $X \in V_\Sigma$, $\gamma \in V_\Sigma^+$, $\alpha \in \mathit{Conf}(\Sigma')$, and $A \in \mathcal{F}$. Then $(X\gamma\alpha, A) \in \mathit{Closure}(\mathcal{B})$ iff $\exists \alpha' \in \mathit{Conf}(\Sigma') : (X\alpha', A) \in \mathcal{B} \ \wedge\ (\gamma\alpha, \alpha') \in \mathit{Closure}(\mathcal{B})$.

(3) Let $\gamma \in V_\Sigma^*$, $\alpha \in \mathit{Conf}(\Sigma')$, $I \in \mathcal{I}$, and $i \geq 0$. Then $(\gamma\alpha, I^i) \in \mathit{Closure}(\mathcal{B})$ iff $\gamma\alpha \sim I^i$.

The aim of the algorithm is to find the *bisimulation base*

$$\mathcal{B}_\sim = \{(\alpha, A) \mid (\alpha, A) \in \mathcal{B}_0,\ \alpha \sim A\}$$

which can be used as a finite representation of bisimilar pairs in the sense of the following proposition.

Proposition 2.42
$\mathit{Closure}(\mathcal{B}_\sim)$ coincides with the set $\{(\gamma\alpha, \beta) \mid \gamma \in V_\Sigma^*,\ \alpha, \beta \in \mathit{Conf}(\Sigma'),\ \gamma\alpha \sim \beta\ \}$.

Proof (idea): Follows directly from the definition of $\mathit{Closure}(\mathcal{B}_\sim)$ using induction on $|\gamma|$. □ □

Remark 2.43
Note that for each $\gamma \in V_\Sigma^*$ and $\beta \in \mathit{Conf}(\Sigma')$ we have $(\gamma, \beta) \in \mathit{Closure}(\mathcal{B}_\sim)$ iff $\gamma \sim \beta$.

Given a set $\mathcal{B} \subseteq \mathcal{B}_0$ and a pair $(\alpha, \alpha') \in \mathcal{B}$, we say (α, α') *satisfies expansion in $\mathcal{B}$* if the two following conditions are satisfied for each $a \in \mathcal{A}$:

- $\forall \beta : \alpha \xrightarrow{a} \beta \ \Rightarrow (\exists \beta' : \alpha' \xrightarrow{a} \beta' \wedge (\alpha', \beta') \in \mathit{Closure}(\mathcal{B}))$, and

- $\forall \beta' : \alpha' \xrightarrow{a} \beta' \Rightarrow (\exists \beta : \alpha \xrightarrow{a} \beta \ \wedge (\alpha', \beta') \in \mathit{Closure}(\mathcal{B}))$.

By $\mathcal{E}(\mathcal{B})$ we denote the set of those pairs in $\mathcal{B}$ that satisfy expansion in $\mathcal{B}$. Notice that the mapping $\mathcal{E}$ is monotonic, i.e., $\mathcal{B} \subseteq \mathcal{B}'$ implies $\mathcal{E}(\mathcal{B}) \subseteq \mathcal{E}(\mathcal{B}')$. Note also that $\mathcal{B}_\sim = \mathcal{E}(\mathcal{B}_\sim)$. Consider now the sequence

$$\mathcal{B}_0 \supseteq \mathcal{B}_1 \supseteq \mathcal{B}_2 \supseteq \cdots$$

where $\mathcal{B}_{i+1} = \mathcal{E}(\mathcal{B}_i)$ for $i \geq 0$. Since $\mathcal{B}_\sim \subseteq \mathcal{B}_0$ and due to monotonicity of $\mathcal{E}$ we obtain $\mathcal{B}_\sim \subseteq \mathcal{B}_i$ for each $i \geq 0$.

Since $\mathcal{B}_0$ is finite, there must be a fixpoint $\mathcal{B}_i = \mathcal{E}(\mathcal{B}_i)$ for some $i \geq 0$. As follows from the following proposition (which can be easily checked), this fixpoint coincides with $\mathcal{B}_\sim$:

Proposition 2.44
If $\mathcal{B} = \mathcal{E}(\mathcal{B})$ then $\mathit{Closure}(\mathcal{B})$ is a bisimulation.

In fact, it is not necessary to compute the sequence $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \ldots$ as it was done in [29, 30]. Instead, we can use the idea from [32] of a reduction to the problem of finding a (unique) maximal solution of a certain set of boolean equations, which was used there in the algorithm for deciding bisimilarity on normed BPA. The idea considerably simplifies the complexity analysis and gives a better complexity bound than would be obtained by a straightforward analysis of the algorithm based on the computation of the fixpoint.

2.6.7 Boolean Equation Systems

Let $\mathit{Var} = \{x_1, x_2, \ldots, x_r\}$ be a (finite) set of boolean variables. A *boolean equation system* is a set of equations of the form

$$x_i = \varphi_i(x_1, x_2, \ldots, x_r)$$

where each φ_i is a monotonic boolean formula over Var, i.e., a boolean formula constructed using variables from Var, and symbols $\wedge$, $\vee$, $\top$, and $\bot$ (symbols $\top$ and $\bot$ denote the formulas that are always true or always false, respectively). In particular, the negation $\neg$ can not be used in φ_i. A valuation ν is a mapping $\nu : \mathit{Var} \to \{\text{TRUE}, \text{FALSE}\}$; it can be extended to formulas in the obvious manner. A valuation ν is a solution of a given boolean equation system if $\nu(x_i) = \nu(\varphi_i)$ for each i.

On valuations we can define the partial order $\sqsubseteq$ such that $\nu \sqsubseteq \nu'$ iff $\nu(x) = \text{TRUE}$ implies $\nu'(x) = \text{TRUE}$ (for each $x \in \mathit{Var}$). A valuation ν is the maximal solution of a boolean equation system if it is the solution of the equation system and it is maximal wrt $\sqsubseteq$. It follows from the well-known Knaster-Tarski fixpoint theorem [41] that every boolean equation system has a unique maximal solution.

The following simple fact, also used in [32], is crucial for obtaining an efficient algorithm for the computation of $\mathcal{B}_\sim$:

Proposition 2.45
Given a boolean equation system, its maximal solution can be found in time linear wrt the size of the system.

Proof (idea): One possibility, how to get a linear time algorithm for finding the maximal solution of a boolean equation system, is to construct a boolean circuit whose inputs correspond to variables in Var and outputs to values of φ_i for each i, to assign TRUE to all gates except those that correspond to $\bot$, and then propagate values FALSE through the circuit. In particular, when the output corresponding to some φ_i is set to FALSE, the input gate corresponding to x_i is set to FALSE. □ □

2.6.8 Construction of the Boolean Equation System for Finding $\mathcal{B}_\sim$

We describe how to construct a boolean equation system BES such that the maximal solution ν_{max} of BES represents $\mathcal{B}_\sim$. Variables of BES correspond to pairs (α, β) of configurations; the variable corresponding to (α, β) is denoted $x_{(\alpha,\beta)}$. The system BES is constructed so that for each variable $x_{(\alpha,\beta)}$ of BES, $\nu_{max}(x_{(\alpha,\beta)}) = \text{TRUE}$ iff $\alpha \sim \beta$.

There are variables of two types in BES:

Type 1: For each $(\alpha, \beta) \in \mathcal{B}_0$ there is a boolean variable $x_{(\alpha,\beta)}$.

Type 2: For each $\gamma \in V_\Sigma^+$, $\alpha \in Conf(\Sigma')$ and $A \in \mathcal{F}$ such that $\text{NORM}(\gamma\alpha) = \text{NORM}(A)$ and γ is a suffix of the right hand side of some rule of Σ (i.e., $(X \xrightarrow{a} \delta\gamma) \in \Gamma_\Sigma$ for some X and δ) such that $|\gamma| > 1$, there is a boolean variable $x_{(\gamma\alpha,A)}$

Note that there are $|\mathcal{B}_0| = O(mk^2)$ variables of type 1, and since the number of suffixes of the right-hand sides of rules of Σ' is less than m, there can be at most mk^2 variables of type 2.

Before defining formulas for all variables in BES, we define auxiliary formulas $g(\alpha,\beta)$ for each α,β where $\textsc{norm}(\alpha) = \textsc{norm}(\beta)$ (formulas $g(\alpha,\beta)$ are used as subformulas in formulas in BES):

- If β is of the form I^i for some $I \in \mathcal{I}$: if $\alpha \sim \beta$ then $g(\alpha,\beta) = \top$ else $g(\alpha,\beta) = \bot$. (Recall $\alpha \sim I^i$ iff $\alpha \in \mathit{Class}(I)^*$ and $\textsc{norm}(\alpha) = i$.)
- If $\beta \in \mathcal{F}$ then $g(\alpha,\beta) = x_{(\alpha,\beta)}$. (Assuming that the variable $x_{(\alpha,\beta)}$ exists in BES, which will be ensured in the following constructions.)

The system BES contains the following equation for each variable $x_{(\alpha,\beta)}$ of type 1:

$$x_{(\alpha,\beta)} = \bigwedge_{\alpha \xrightarrow{a} \alpha'} \left(\bigvee_{\substack{\beta \xrightarrow{b} \beta' \\ \text{where } a=b \text{ and} \\ \textsc{norm}(\alpha')=\textsc{norm}(\beta')}} g(\alpha',\beta') \right) \wedge \bigwedge_{\beta \xrightarrow{b} \beta'} \left(\bigvee_{\substack{\alpha \xrightarrow{a} \alpha' \\ \text{where } a=b \text{ and} \\ \textsc{norm}(\alpha')=\textsc{norm}(\beta')}} g(\alpha',\beta') \right)$$

The equation expresses that every transition $\alpha \xrightarrow{a} \alpha'$ enabled in α must be matched by some transition $\beta \xrightarrow{a} \beta'$ enabled in β and vice versa, recall the definition of $\mathcal{E}$. (Note that all subformulas $g(\alpha,\beta)$ are defined correctly in the above formula.)

For each variable $x_{(X\alpha,A)}$ of type 2 (where necessarily $X \in V_\Sigma$ and α starts with a symbol from V_Σ), the system BES contains the equation

$$x_{(X\alpha,A)} = \bigvee_{\substack{B \in \mathcal{F} \\ \text{s.t.} \\ \textsc{norm}(B)= \\ \textsc{norm}(\alpha)}} (g(XB,A) \wedge g(\alpha,B)) \vee \bigvee_{I \in \mathcal{I}} \left(g(XI^{\textsc{norm}(\alpha)},A) \wedge g(\alpha, I^{\textsc{norm}(\alpha)}) \right) .$$

This formula directly corresponds to point (2) of the definition of $\mathit{Closure}(\mathcal{B})$.

To estimate the sizes of the formulas in BES, it is obviously sufficient to estimate the number of occurrences of subformulas $g(\alpha,\beta)$ in these formulas. (Note that the size of each $g(\alpha,\beta)$ is $O(1)$.)

Let us consider formulas for variables of type 1 of the form $x_{(X\alpha,A)}$ where $X \in \mathit{Var}_\Sigma$. The rules that can be used for possible transitions in $X\alpha$ depend only on X. If we count the total number of pairs of rules $X \xrightarrow{a} \gamma$ and $A \xrightarrow{a} \beta$ for all $X \in V_\Sigma$, $A \in \mathcal{F}$, we can see that there is at most $m\ell$

such pairs of rules. Each such pair is used in at most k formulas (there are at most k possible values of α), and it is used at most twice in each formula. So the total size of formulas for the variables of type 1 of the above mentioned form is at most $O(m\ell k)$. Similarly, the total size of formulas for the variables of type 1 of the form $x_{(\alpha,A)}$ where $\alpha \in \mathit{Conf}(\Sigma')$ is at most $O(\ell^2)$.

It is clear that the size of each formula for a variable of type 2 is $O(|\mathcal{F}|+|\mathcal{I}|) = O(k)$. Since there are at most mk^2 variables of type 2, the total size of their formulas is $O(mk^3)$.

Summing the sizes of the formulas in BES we obtain:

Proposition 2.46
The size of BES is $O(mk^3 + m\ell k + \ell^2) = O(n^7)$.

2.6.9 The Overall Algorithm

Theorem 2.47
There is an algorithm solving nBPA-nBPP-BISIM *in time* $O(n^7)$.

Proof: The algorithm works as described above. It transforms the given nBPP (M_0, Δ) into the prime form and generates (a succinct representation of) nBPA (α'_0, Σ') from it. If the construction of (α'_0, Σ') is finished (i.e., the algorithm does not stop with the negative answer), the corresponding boolean equation system BES of size $O(n^7)$ (recall Proposition 2.46) is constructed and the algorithm finds its maximal solution ν_{max} in time $O(n^7)$ (recall Proposition 2.45). The algorithm then checks if $\nu_{max}(x_{(\alpha_0,\alpha'_0)}) = \text{TRUE}$ (without loss of generality we can assume that $\alpha_0 \in V_\Sigma$, $\alpha'_0 \in \mathcal{F}$ and so BES contains the variable $x_{(\alpha_0,\alpha'_0)}$) which gives the answer for the original instance of nBPA-nBPP-BISIM.

Before the construction of BES, the rules of Σ and Σ' can be partitioned according to their labels and the changes on norms they cause. Norms for all $X \in V_{all}$ and for all suffixes of right hand sides of rules of Σ can be precomputed. Note that there are at most $O(n^5)$ different subformulas of the form $g(\alpha, \beta)$ that occur in formulas for variables of type 2 and that for every such pair the subformula $g(\alpha, \beta)$ can be precomputed in time $O(n^2)$. Using all this precomputed information, the system BES can be constructed in time $O(n^7)$.

All other steps of the algorithm (the transformation to the prime form, the generation of Σ', the precomputation of the sets $\mathit{Class}(I)$ for all $I \in \mathcal{I}$ and the precomputation of all other necessary information described above) can be obviously done in time $O(n^7)$. $\square$

After ν_{max} has been computed, it can be used for fast deciding if $\gamma \sim A$ for all $\gamma \in V_\Sigma$, $A \in \mathcal{F}$. Just note that for each suffix γ' of γ we can quickly find all $\beta \in \mathit{Conf}(\Sigma')$ such that $\gamma' \sim \beta$ (in fact there is always at most one such β due to the fact that all configurations in $\mathit{Conf}(\Sigma')$ are pairwise non-bisimilar) assuming this information was already computed for all its proper suffixes.

Remark 2.48
The above algorithm can be used for deciding bisimilarity between a given nBPA (of size m) and a finite state system (with k states and ℓ transitions) and the running time of the algorithm is $O(mk^3 + m\ell k + \ell^2) = O(n^4)$ in this case (where n is the size of the whole instance). In fact, the algorithm can be easily adapted for the case when the BPA and the FS in the instance are not required to be normed (as in [29, 30]) without affecting its complexity. The more general problem of deciding weak bisimilarity on a given BPA and FS process was considered in [29] and the algorithm presented there has running time $O(m^5(k + \ell)^7) = O(n^{12})$. The special case of the strong bisimilarity was not analyzed there and we are not aware of any tighter results concerning its complexity.

Chapter 3

Modeling and Verification of Real Time Database Management Systems Using Uppaal

In this chapter we will concern with a practical use of the model checking tool Uppaal on modeling and verification of real-time database system and its parts. The models are designed from two different perspectives:

- Concurrency control of existing real-time database management system V4DB – This is approach is presented in Section 3.5 and was published in [23]

- Individual concurrency control protocols – It is presented in Section 3.6 and was published in [24] and [25].

Before the sections with models, there is some introduction and motivation in Section 3.1, a short description of real time database systems in Section 3.2, a mention of existing real time database system V4DB and some of its properties in Section 3.3 and finally the Section 3.4 is devoted to a verification tool Uppaal.

3.1 Introduction

A number of used real-time systems and their importance grows every year. Many of them work with large amount of data. For examples in the following areas are timing requirements stringent stock exchange systems, telephone switching, radar tracking etc. Traditionally, each real time software manages data in its own, application dependent, structure. A drawback of this is that during a creation of such systems similar problems are solved again and again. For traditional (non-real-time) systems, analogous problem was solved using database management systems (DBMS). In recent years, there has been an interest in using, during years well-developed, database technology in real-time systems.

Traditional DBMS provide efficient storage of data and means for manipulation with stored information. Real-time systems are usually associated with some time constraints and DBMS can not guarantee any bound on response time. This is the reason why so-called real-time database management systems (RTDBMS) emerged ([31, 21]). They should merge some algorithms from traditional DBMS which proved to be effective in the course of time and add some real-time possibilities.

Typical use of RTDBMS is in systems where some values from various sensors are stored and using queries, an operator or a controlling part of the system can search in those values and control the system using the results. It means that RTDBMS are not the main controlling or functional part of real-time system, they only provide the support as data store for other parts of system.

Systems where RTDBMS could be useful can be basically divided into 2 groups:

- Real-time control systems – they work with large amount of data and have strict claims on time. Examples are production lines with computer control, telecommunication systems, defense systems, air and ground traffic control etc.
- Classical information systems where at least some operations have some critical deadline of execution, for example stock-exchange or bank systems.

The mentioned two categories have different demands on system. Applications from the first category have really strict demands on response to queries (e.g. late emergency stop in case of problems can by costly), validity and information value of stored data decrease in the course of time (current values from sensors have bigger influence on control of system than values several minutes or hours old), insertions and queries are often periodic. Applications from the second category have usually less strict demands on response but operations are often aperiodic and more complex.

Research in the area of RTDBMS began in the 1990's and focused on evolution of transaction processing algorithms, priority assignment strategies, concurrency control techniques, etc. Algorithms from traditional DBMS are adapted to use in real-time environment or some completely new algorithms were suggested. The research was based especially on simulation studies. Hence at Technical university of Ostrava, Václav Król, Jindřich Černohorský and Jan Pokorný designed and implemented an experimental real-time database system called V4DB [27], which is suitable for study of real-time transaction processing. The system is still in further development but some important results were obtained already.

3.2 An Overview of Real-Time Database Systems

In this section we look more in detail on RTDBMS, their parts, used algorithms and protocols etc. This section is mainly based on informations from [21] and [27].

RTDBMS use transactional model usually. A transaction is unit of work performed against a database and treated in a coherent and reliable way independent of other transactions. Transactions help in two main ways:

- To recover from failures – Before and after transaction a database should be consistent with reality and inconsistence is allowed during transaction processing only. If a transaction ended successfully, the database is surely consistent. If a transaction processing is interrupted a consistent state from the time before the start of the transaction is restored.
- To deal with concurrent access – Several transactions may be executed in parallel. It should be guaranteed that the state of the database after a successful end of all parallel transactions is the same as if those transactions were executed sequentially in some order – so called *serializability*.

The transaction processing usually works using a pattern similar to the following:

1. Begin the transaction
2. Execute operations of the transactions
3. If an error occurs rollback all changes made by the transaction so far and end the transaction
4. If all operations are successfully ended commit the transaction and end it

The described characteristics of transactions are the same in the case of traditional DMBS as well as in the case of real-time DBMS. To deal with time constraints in RTDBMS some more information may be available and used by scheduler or concurrency control unit:

1. Timing constraints
2. Criticalness
3. Value function
4. Resource requirements
5. Expected execution time
6. Data requirements
7. Periodicity
8. Time of occurrence of events
9. Other informations

The first three properties are related. A deadline is the time when the transaction should end. But for different transaction, a criticality of missing the deadline may be different. A value function measures how valuable it is to complete the transaction at some point of time. This value usually falls significantly after the deadline. Resource requirements (number of input/output operations, expected CPU time, etc.), an expected execution time and data requirements may help to choose for execution those transactions which have the biggest chance to meat their deadline. A periodicity means a period of transactions which occur repeatedly in regular intervals.

A big part of RTDBMS research concerns with scheduling of jobs in a multiprogramming environment. Transaction scheduling is not based only on CPU scheduling as it is common in other environments. Conventional scheduling algorithms make an effort to balance the number of CPU-bound and I/O-bound jobs to maximize system utilization and throughput. They try to ensure that each process gets its fair share of system resources. But transactions should be scheduled according to their criticalness and the tightness of their deadlines. Real-time database scheduling algorithms give more time and resources to transactions which are very critical and with stringent timing constraints. A popular method is to assign a priority (usually a natural number) to each transaction – a transaction with higher priority is given more CPU time and resources.

Many attributes of transaction may affect its priority. The most relevant to a RTDBMS are:

- Criticalness – more critical transactions have higher priority
- Deadline – transactions with earlier deadline have higher priority
- Amount of unfinished work – transaction with small amount of unfinished work can have higher priority. Especially a transaction in validation phase should have high priority to end as soon as possible because it has small amount of work to be done and after the finish can potentially release some resources.
- Amount of computation already invested – transactions that already have a large amount of computation done may be given a higher priority.
- Age – transactions could get priorities according to their arrival to the system, older transaction gets higher priority
- Slackness – a measure, how long an execution of transaction can be delayed while still making it possible to meet its deadline.

It this worth to combine more of the mentioned parameters to determine the real priority of a particular transaction.

Other part of RTDBMS which is given much attention in the research is concurrency control. It is the control of interaction among concurrent transactions in such a way that consistency of a database is not destroyed. The interaction between transactions is mainly realized by reads and writes of data items. Concurrency control is quite well solved in conventional DBMS as a result of an intensive research in last decades.

The main correctness criterion in concurrency control is so called serializability. A sequence of database operations is considered serializable if its effect is equivalent to a serial transaction schedule. In RTDBMS, it can be in some situations useful to improve performance of database system sacrificing serializability. But more often, the serializability remains the choice for ensuring database consistency in real time case too.

Protocols used for concurrency control in RTDBMS are either modifications of protocols used in conventional DBMS or newly created especially for RTDBMS. Mostly used are the following types of protocols:

- Pessimistic protocols
- Optimistic protocols

and from conventional databases as well known but in real-time environment less frequently used:

- Speculative protocols
- Multiversion protocols
- Protocols based on dynamic adaptation of serial control

In the following we will consider only pessimistic and optimistic protocols.

The most common pessimistic protocol used in conventional databases is two phase locking (2PL). The execution of a transaction consists of two phases. During the first phase, locks are acquired and can not be released. In the second phase, locks are only released and can not be acquired. Transaction have to wait if it is requesting a lock that is currently being held by another transaction. Conventional version of 2PL protocol is not suitable for real time database systems. Two main problems are deadlocks and priority inversion.

Priority inversion is the situation when transaction with higher priority needs a lock currently held by transaction with lower priority. It means that the transaction with higher priority waits till the transaction with lower priority ends. As transaction with lower priority get less CPU time and other resources, the delay for transaction with higher priority could be really big and possibly cause a miss of its deadline. There are several possible solutions to this problem, i.e. variants of 2PL protocol, which prevent priority inversion or minimize effect of priority inversion:

- Wait Promote – Priority of the lock holder transaction is raised to the priority of the requester. It means that transaction with (originally) higher priority will still wait, but the time needed to finish the transaction holding a lock can be significantly reduced.
- High Priority – A lock holder transaction with lower priority is aborted and the lock is granted to a requester with higher priority. It is suitable in the case when abort of lock holder is not too expensive. Priority inversion is not problem any more but a problem can be with some priority assignment strategies – aborted transaction is restarted with newly set priority which could be higher (deadline of the transaction is nearer etc.) and restarted transaction can possibly cause abort of the transaction which originally killed it. This leads to a problem of cyclic restart.
- High Priority without cyclic restart – before a lock holder transaction is aborted due to a request by a transaction with higher priority, the systems checks whether the next incarnation of the aborted transaction will have also a lower priority then the requesting transaction. If it is not the case there is no abort and transaction with higher priority waits.
- Conditional Restart – To avoid too many restarts, an abort of a transaction can be conditioned by some other criteria, not just the priority. For example, if there is high probability that requesting transaction will meet its deadline even if it is waiting for a lock held by a transaction with lower priority, an abort need not to be necessary.

The second main problem of protocols based on locks is the possibility of deadlock. Deadlock occurs when a set of transactions is involved in a circular wait (first transaction is waiting for a lock held by second transaction, second waits for third, ..., n-th transaction waits for first). There are several possible solutions. Mainly they are based on abort of some transaction (or transactions) to enable successful finish of the other involved transactions. The victim may be chosen in several ways:

- Abort a transaction that already passed its deadline
- Abort a transaction with longest deadline
- Abort the least critical transaction
- Abort the transaction with lowest priority

Pessimistic protocols based on data locks are widely used in commercial DBMS. For real-time DBMS can optimistic protocols be suitable because they are non-blocking and deadlock free.

Execution of a transaction with optimistic concurrency control can be basically divided into three phases:

1. read – data are read into memory, new values are computed
2. validation – check for conflicts
3. write – changes and computed values are stored into database

During validation phase two rules are checked against all other concurrent transactions:

1. R/W rule
2. W/W rule

Suppose that a transaction $T1$ is serialized before a transaction $T2$. The first rule means that data items written by $T1$ have not already been read by $T2$. The second rule says that each write of $T1$ should not overwrite any write of $T2$. If some of this rules is violated conflict must be resolved. This usually involves abort of one or more transactions. There were several possibilities suggested how to choose transactions for restart if a conflict occurs:

- Broadcast commit – A validating transaction is always committed and all conflicting transactions are restarted.

- Sacrifice – If a validating transaction has lower priority than some of conflicting transactions, the validating transaction is restarted.

- Wait – If a validating transaction T has lower priority than some of conflicting transactions, T waits for some time. It gives a chance for conflicting transactions with higher priority to end before their deadline. If all conflicting transactions with higher priority exceed their deadlines, T is committed.

Some other problems have to be solved in RTDBMS as well. This involves for example memory management and data buffering, I/O scheduling, support of operating system etc. As this problems were not considered in our modeling and verification, we will skip their description.

3.3 Real-Time Database System V4DB

The main goal of the V4DB project ([28, 27]) was to design and implement real-time database system suitable for study of real-time transaction processing. The system is still in development. I will slightly describe how it looked in the time when I started with its modeling and when a PhD thesis [27] appeared. Later changes in V4DB are not present in our models for Uppaal yet.

An experimental RTDBMS called V4DB is implemented as an integrated set of the most important functional parts of a veritable real-time database system. It enables testing and performance analysis of different algorithms for particular functional parts to understand the effect on system performance.

V4DB works under a real-time operating system VxWorks. The database is stored in memory to eliminate the influence of accesses to hard disk on the results of tests and analysis. The system consists of several parts working in parallel. The behavior of those parts could be changed easily by setting before start. Main executive parts are transaction generator, predispatcher, dispatcher and transaction execution including concurrency control. Those parts use data dictionary and database.

To study the database transaction processing, transactions should have known properties that can be set in advance. Hence they are generated by an internal generator. There are two different generators – periodic and random. Periodic generator creates transaction with predefined period, random generator has defined minimal and maximal interval between transactions. Generated transactions contain the following operations:

- Select – selection and read of a record
- Update – modification of a record
- Insert – insertion of new record into a table
- Delete – removal of a record

From the generator, the transaction is passed to predispatcher. This process avoids system overloading and creates the structure fully describing the transaction. This structure is used by all other functional blocks. Dispatcher then extracts the transaction parameters and dispatches transactions for execution according the priority assignment policy.

The transaction scheduled for an execution is parsed into particular commands and then the commands are processed by the command executor. To obtain a reasonable performance, multiple transactions must be able to access data concurrently. Hence there is concurrency control component that synchronizes access to the stored data. The modularity of V4DB allows to use many different concurrency control protocols. Basically, all optimistic and pessimistic protocols described in Section 3.2 are implemented and can be used. The selection of one of them is made at the start of V4DB system.

The database of V4DB is very simple. There are several tables. Each table has given number of records and each record is one value of a given size. Records that are object of some operation (select, update, etc.) are specified just by a name of a table and numerical order of the record in the table.

3.4 Verification Tool Uppaal

To our best knowledge, there is not any verification tool intended directly for real-time database systems. We have chosen the tool Uppaal because it has at least a support for real-time, it is freely available and models specified graphically as finite automata are quite easy to understand for people occupied by real-time databases and not experienced in verification.

Uppaal ([3, 10]) is a verification tool for real-time systems. It is jointly developed by Uppsala University and Aalborg University. It is designed to verify systems that can be modeled as networks of timed automata extended with some further features such as integer variables, structured data types, user defined functions, channel synchronization and so on.

A timed automaton is a finite-state automaton extended with clock variables. A dense-time model, where clock variables have real number values and all clocks progress synchronously, is used. In Uppaal, several such automata working in parallel form a network of timed automata.

An automaton has locations (known as states in the automata theory) and edges (transitions in the automata theory). Each location has an optional name and invariant. An invariant is a conjunction of side-effect free expressions of the form $x < e$ or $x \leq e$ where x is a clock variable and e evaluates to an integer. Each automaton has exactly one initial location.

Particular automata in the network synchronize using channels and values can be passed between them using shared (global) variables. A state of the system is defined by the locations of all automata and the values of clocks and discrete variables. Valid are only states such that invariants of actual locations of all automata are evaluated to true when using actual values of clocks. The state can be changed in two ways – passing of time (increasing values of all clocks by the same amount) and firing an edge of some automaton (possibly synchronizing with another automaton or other automata).

Some locations may be marked as committed. If at least one automaton is in a committed location, time passing is not possible, and the next change of the state must involve an outgoing edge of at least one of the committed locations.

Each edge may have a select, a guard, a synchronization and an assignment. Select gives a possibility to choose nondeterministically a value from some range. Guard is a side-effect free expression that evaluates to a boolean. The guard must be satisfied when the edge is fired. It can contain not only clocks, constants and logical and comparison operators but also integer and boolean variables and (side-effect free) calls of user defined functions.

Synchronization label is of the form $Expr!$ or $Expr?$ where $Expr$ evaluates to a channel. An edge with $c!$ synchronizes with another edge (of another automaton in the network) with label $c?$. Both edges have to satisfy all firing conditions before synchronization. Sometimes we say that automaton firing an edge labeled by $c!$ sends a message c to the automaton firing an edge labeled by $c?$. There are urgent channels as well – synchronization

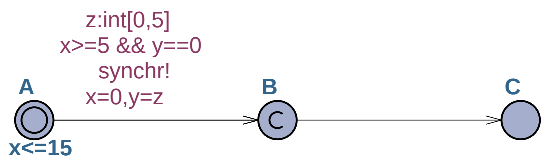


Figure 3.1: Graphical representation of a timed automaton in Uppaal

through such a channel have to be done in the same time instant when it is enabled (it means, time passing is not allowed if a synchronization through urgent channel is enabled) – and broadcast channels (any number of $c?$ labeled edges are synchronized with one $c!$ labeled edge) . An assignment is a comma separated list of expressions with a side-effect. It is used to reset clocks and set values of variables.

Figure 3.1 shows some of the described notions are represented graphically in Uppaal. There are 3 locations named `A`, `B` and `C`. Location `A` is initial and `B` is committed. Moreover `A` has an invariant `x<=15` with the meaning that the automaton can be in this location only when the value of the clock variable `x` is less or equal 15. The edge between `A` and `B` has the select `z:int[0,5]` – it nondeterministically chooses an integer value from the range 0 to 5 and stores it in variable `z`. This edge also has the guard `x>=5 && y==0`. This means that it can be fired only when the value of the clock variable `x` is greater or equal 5 and the integer variable `y` has the value 0. Data types of variables are defined in a declaration section. Further it has synchronization label `synchr!` and an assignment `x=0, y=z` reseting the clock variable `x` and setting the value of `z` to the integer variable `y`. Second edge has only synchronization label `synchr?` hence it can be fired only when some other automaton (possibly of the same template) has an edge labeled `synchr!` enabled.

Uppaal has some other useful features. Templates are automata with parameters. These parameters are substituted with given arguments in the process declaration. This enables easy construction of several alike automata. Moreover, we can use bounded integer variables (with defined minimal and maximal value), arrays and user defined functions. These are defined in declaration sections. There is one global declaration section where channels, constants, user data types etc. are specified. Each automaton template has own declaration section, where local clocks, variables and functions are specified. And finally, there is a system declaration section, where global variables are declared and automata are created using templates.

Uppaal's query language for requirement specification is based on CTL (Computational Tree Logic, [12]). It consist of path formulae and state formulae. State formulae describe individual states and path formulae quantify over paths or traces of the model.

A state formula is an expression that can be evaluated for a state without looking at the behavior of the model. For example it could be a simple comparison of a variable with a constant `x <= 5`. The syntax of state formulae is similar to the syntax of guards. The only difference is that in a state formula disjunction may be used.

There is a special state formula `deadlock`. It is satisfied in all deadlock states. The state is deadlock if there is not any action transition from the state neither from any of its delay successors.

Path formulae can be classified into *reachability*, *safety* and *liveness*. Reachability formulae ask if a given state formula is satisfied by some reachable state. In Uppaal we use $F\Diamond\varphi$ where φ is a state formula and we write it as `E<>` φ .

Safety properties are usually of the form: "something bad will never happen". In Uppaal they are defined positively: "something good is always true". We use $A\Box\varphi$ (written as `A[]` φ) to express, that a state formula φ should be true in all reachable states, and $E\Box\varphi$ (`E[]` φ) to say, that there should exist a maximal path such that φ is always true.

There are two types of liveness properties. Simpler is of the form: "something will eventually happen". We use $A\Diamond\varphi$ (`A<>` φ) meaning that a state formula φ is eventually satisfied. The other form is: "leads to a response". The formula is $\varphi \leadsto \psi$ (written as φ `-->` ψ) with the meaning that whenever φ is satisfied, then eventually ψ will be satisfied.

The simulation and formal verification are possible in Uppaal. The simulation can be random or user assisted. It is more suitable for the user of the tool to see if a model is behaving like he wants and like it corresponds to the real system. Formal verification should confirm that the system has desired properties expressed using the query language. There are many options and settings for verification algorithm in Uppaal. For example we can change representation of reachable states in memory or the order of search in the state space (breadth first, depth first, random depth first search). Some of the options lead to less memory consumption, some of them speed up the verification. But improvement in one of these two characteristic leads to a degradation of the other, usually.

For more exact definitions of modeling and query languages and verification possibilities of Uppaal see [3].

3.5 Modeling of V4DB

First our attempts on modeling and verification of real-time database systems were concerned with models of V4DB parts with emphasis on concurrency control. Uppaal is supposed to be used on so-called reactive systems, which are quite different from database systems. So we need to solve the problem of modeling data records of the database and some other problems. Then we would like to check some important properties of used protocols and algorithms, for example: absence of a deadlock when using an algorithm which should avoid deadlock in the transaction processing, processing transaction with bigger priority instead of transactions with smaller priority and so on.

Big problem of verification tools is so called state space explosion. Uppaal is not able to manage too detailed models. On the other hand, too simple models can not catch important properties of a real system. So we need to find a suitable level of abstraction.

Concurrency control is one of the most important and crucial parts of all database management systems allowing concurrent access to data records. Some of used protocols are described in Section 3.2.

In this section, we will concentrate on a pessimistic protocol called Two phase locking (2PL) (Subsection 3.5.1) and one optimistic protocol called Sacrifice (3.5.2). In the Subsection 3.5.3 we will then discuss some possible formula expressed in query language of Uppaal which can be checked on the suggested models. All this results were published in [23].

3.5.1 Pessimistic Protocol Two Phase Locking

In this subsection we will show one possible model of pessimistic concurrency control protocol. But the ideas used to create this model are general and can be used for creation of models of other protocols as well.

Protocol called Two phase locking (2PL) is based on data locks. The transaction must have a lock before access to a data record. All locks granted to a transaction are released after all operations of this transaction are executed. There are two types of locks – for read and write. The first is used for the operation select and the latter for update, delete and insert. Either one write lock or several read locks can be on a particular record (for simplicity, in our model will be only one read lock allowed for one record). If a transaction can not get a lock for a request it is placed in a queue of this record. After an existing lock is released, a new lock is granted to the first transaction in the queue.

Deadlock can arise in the described protocol when some transactions wait mutually for locks acquired by other transactions. This can be avoided in several different ways. The one used in V4DB is time limit for waiting in a queue for a lock. After the limit, transaction is restarted or aborted if its critical time has passed. Both, restart and abort of the transaction, release all locks granted to this transaction in the past and some other waiting transactions can get them.

The model consists of several timed automata. Each of them represents a part of RTDBMS. Generator and dispatcher are not in our focus hence they are modeled as one, quite simple, automaton. We only need to catch a transaction creation.

In V4DB, there is a maximal number of active transactions at the same time. Predispatcher avoids overload of the system in such a way that transactions beyond the limit are put into the buffer and executed when the execution of some other transactions terminates. If the buffer is full, incoming (in experimental system generated) transactions are discarded.

First automaton represents a dispatcher together with random generator. It is depicted on Figure 3.2 and is quite simple.

The variable `waiting` counts transactions waiting for an execution in the buffer. Two edges represent generation of a new transaction. One of them adds this transaction to a buffer if it is not full. The other one is intended as a representation of the situation where buffer is full and transaction is discarded without processing. The constants `MIN_INTERVAL` and `MAX_INTERVAL` determine minimal and maximal time between generation of two consecutive transactions. If both constants have the same value, the generator is periodic. `a` is a clock variable which measures the interval between two transactions. A created transaction should have some data describing database operations. But, as they are chosen in random, we can afford in this model to choose particular operations randomly during the execution.

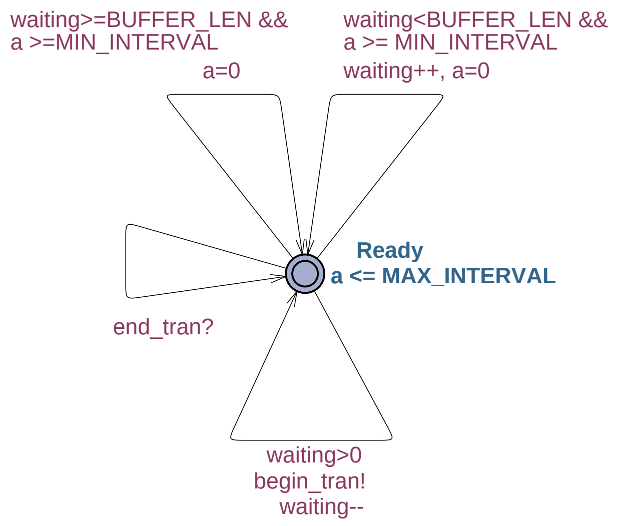


Figure 3.2: Dispatcher automaton

The other two transitions are used for communication with transaction manager. Communication through the channel `begin_tran` represents passing of transaction to the manager. However the transaction is not really generated in the dispatcher/generator automaton and therefore the synchronization without any data passing is enough. The second channel is intended as notification of the end of a transaction to the dispatcher. In the actual model it has no real meaning, it is there only for more realistic behavior and for use in other models (not presented in this thesis, focused on other aspects of Two phase locking protocol).

As number of concurrently active transactions is bounded, we can model each active transaction as one automaton. We use the template shown on Figures 3.3 and 3.4. In Uppaal it is one big automaton template, locations with the same name on both figures are the same in the template. Several copies of this automaton are created in the system declaration. Each copy has an unique integer identification value stored in the variable `id`. There are several user defined functions used in this template. For illustration, the definition of them (from declaration section of this Uppaal template) is shown on Figure 3.5. In the following we will just describe intention of functions in other models without presentation of the full code of them.

An automaton representing transaction starts in the location `Free`. The

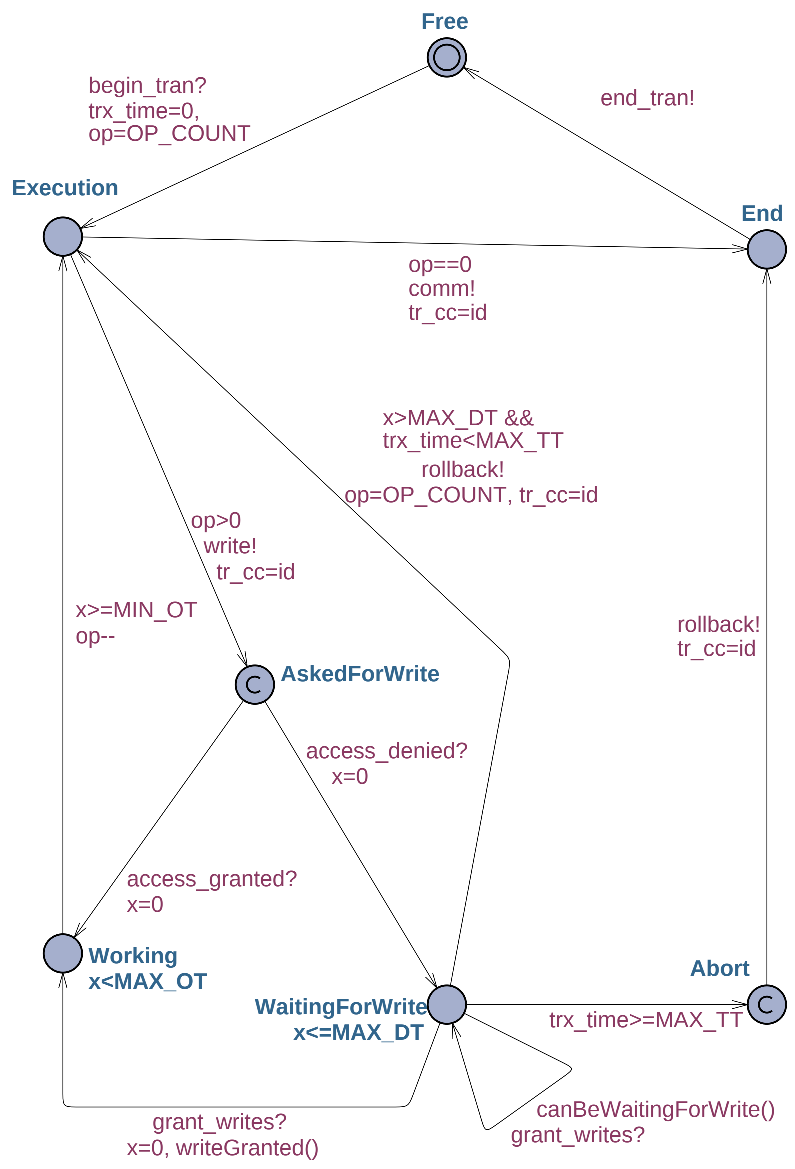


Figure 3.3: Transaction automaton for Two phase locking protocol

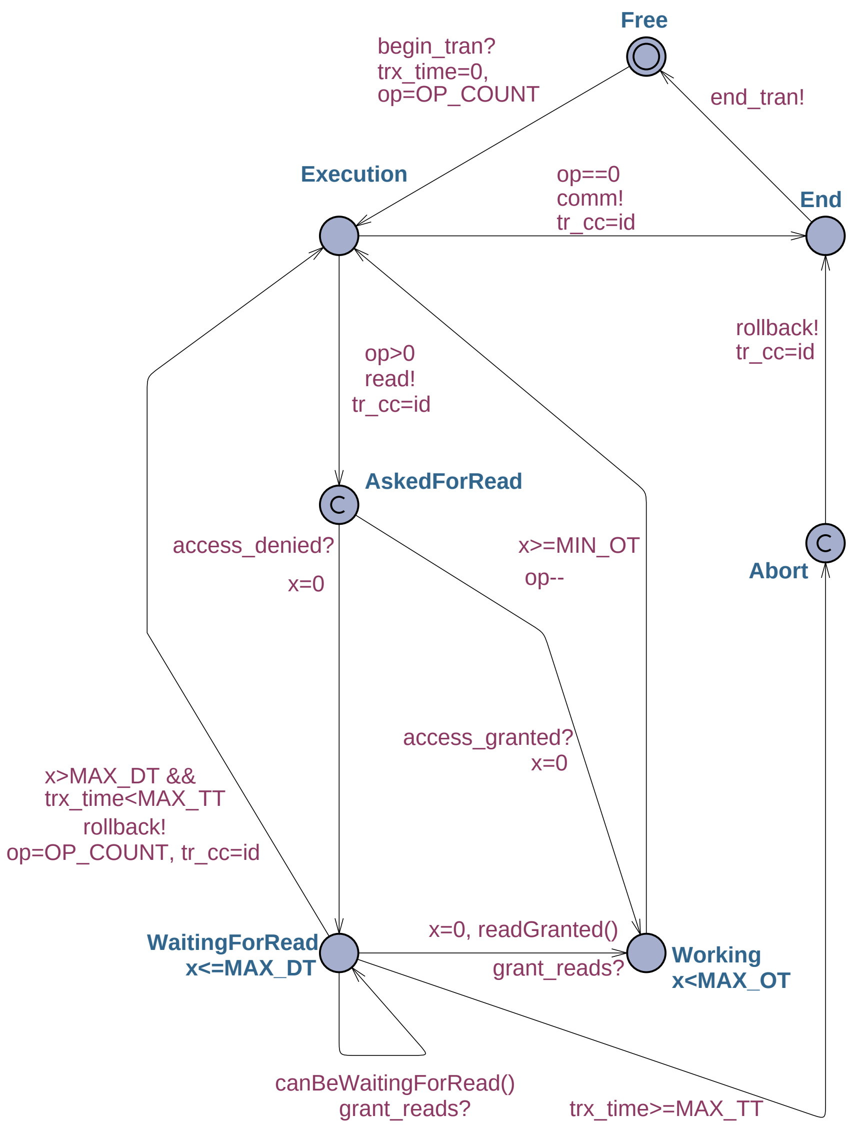


Figure 3.4: Second part of a transaction automaton for Two phase locking protocol

```
bool canBeWaitingForRead(){
  return (write_count>1 || (write_count==1 && !write_list[id]));
}

bool canBeWaitingForWrite(){
  return (read_count>1 || (read_count==1 && !read_list[id]) ||
          write_count>1 || (write_count==1 && !write_list[id]));
}


void readGranted(){
  if(!read_list[id]) read_count++;
  read_list[id]=true;
}

void writeGranted(){
  if(!write_list[id]) write_count++;
  write_list[id]=true;
}
```

Figure 3.5: Definitions of functions used in the template for transactions

edge to the location `Execution` synchronizes with dispatcher automaton. This ensures that the transaction automaton gets active only when some transaction is waiting for an execution. The clock variable `trx_time` will contain the time of the whole execution of the transaction. For the simplicity, we can assume that each transaction contains the same number of operations. This number is given by the constant `OP_COUNT` in the model. An integer variable `op` is used as a counter of already executed operations.

If an automaton is in the location `Execution` and all operations were executed (the value of the variable `op` is 0) the transaction could end. It has to release all granted locks hence it synchronizes with concurrency control automaton using the channel `comm`. The variable `tr_cc` is shared between transaction and concurrency control. Transaction which ends sets its identification to this variable. Therefore the concurrency control knows which transaction it is synchronizing with.

The proper database operation is not important in our model. We only have to differentiate between the two types of locks. Hence we consider two operations – read and write. Through the channels `read` and `write` the transaction asks concurrency control for locks. It is in fact random for a particular transaction whether there is a lock on the record already. Therefore we do not need to represent particular records and pass an identification of record to the concurrency control. We only get a response through the channel `access_granted` if a lock was granted or `access_denied` in the opposite case.

If a lock has been granted, the transaction is in the location `Working`. This location represents execution of one database operation. We could have more states for different operations, but, from our actual point of view, all operations are similar. Hence the execution of an operation is modeled as a time delay only. The time of an execution is measured using a clock variable `x` and an automaton can leave this location only when the time spent there is greater or equal to `MIN_OT` and less than `MAX_OT` (MINimal and MAXimal Operation execution Time).

Locations `WaitingForRead` and `WaitingForWrite` represent the situation when transaction could not get a lock and is waiting for it. The clock variable `x` is intended for measuring time spent in those locations. When the limit `MAX_DT` is exceeded, the transaction is restarted or aborted. Abort comes on when the transaction has exceeded its critical time. This is, for simplicity, for all transactions `MAX_TT` time units since the start of the transaction execution. Transaction which is aborting synchronizes with concurrency control using `rollback` channel. This ensures that all locks, which were granted to the transaction previously, are removed. Then the transaction automaton can represent other transaction. Restart synchronizes with concurrency control too. Hence all previously gained locks are removed as well. The count of operations is set back to `OP_COUNT`. It is not important that in the next run of the restarted transaction other operations may be chosen. In properties we usually ask for something to hold on all possible execution paths – hence the path with the same order of operations is considered too.

A transaction waiting for a lock can get it when some other transaction ends or is restarted. The information about the end of the other transaction is passed to a waiting one through the broadcast channels `grant_reads()` and `grant_writes()`. The demanded record can be free or not. We represent those eventualities using nondeterminism. There are two different edges with the label `grant_reads()` or `grant_writes()`, one leads back to a waiting location and the other to the location `Working`. In the case that finished transaction removed all remaining locks, the edge leading back to the waiting location is not enabled because of the guard `canBeWaitingForRead()` or `canBeWaitingForWrite()`. Those functions return true when some locks granted to the other transactions exist.

The last automaton in our model of Two phase locking protocol is concurrency control. It is depicted on the figure 3.6. The initial location `Ready` is the only one where time delay is possible. All other locations are committed what prevents time delay. It means, that answers to all requests are immediate.

The first possible demand is for a read lock. It is passed through the channel `read`. We suppose that the database is much greater than the count of active transactions. Hence there are unlocked records every time. Therefore it is always possible to grant access to a transaction through the channel `access_granted`. The lock can be refused only when some other transaction has a write lock. Concurrency control maintains two boolean arrays (`read_list` and `write_list`) where it stores for each transaction automaton if it has some read or write lock. The function `canDenyRead()` has identification of transaction as an argument. It checks whether some other transaction has an active write lock and returns answer as a boolean value. When the lock is granted, the function `addRead()` stores information about this lock to `read_list`.

The solution for a write lock demand through the channel `write` is similar. The only difference is in the function `canDenyWrite()`. This function has to check read locks too. If there is some read or write lock granted to a transaction other than the asking one, the lock can be refused. It corresponds to a real situation when accessed record is the locked one.

The last part of concurrency control is responsible for commits and rollbacks of transactions. In the real system, all changes made by a transaction before abort or restart are taken back. We do not have data in our model hence rollback is the same as commit. The only thing to be done is a release of all locks granted to the transaction. After the release, waiting transactions should be informed that the accessed record could be free now. If the transaction had only read locks, transactions waiting for a write lock are informed because read lock can not block read. If the transaction had both types of locks or some read locks, all waiting transactions are informed through the broadcast channels `grant_writes()` and `grant_reads()`. The function `removeWrite()` stores information about released write locks and returns true if there was any such lock. The function `removeRead()` is sim-

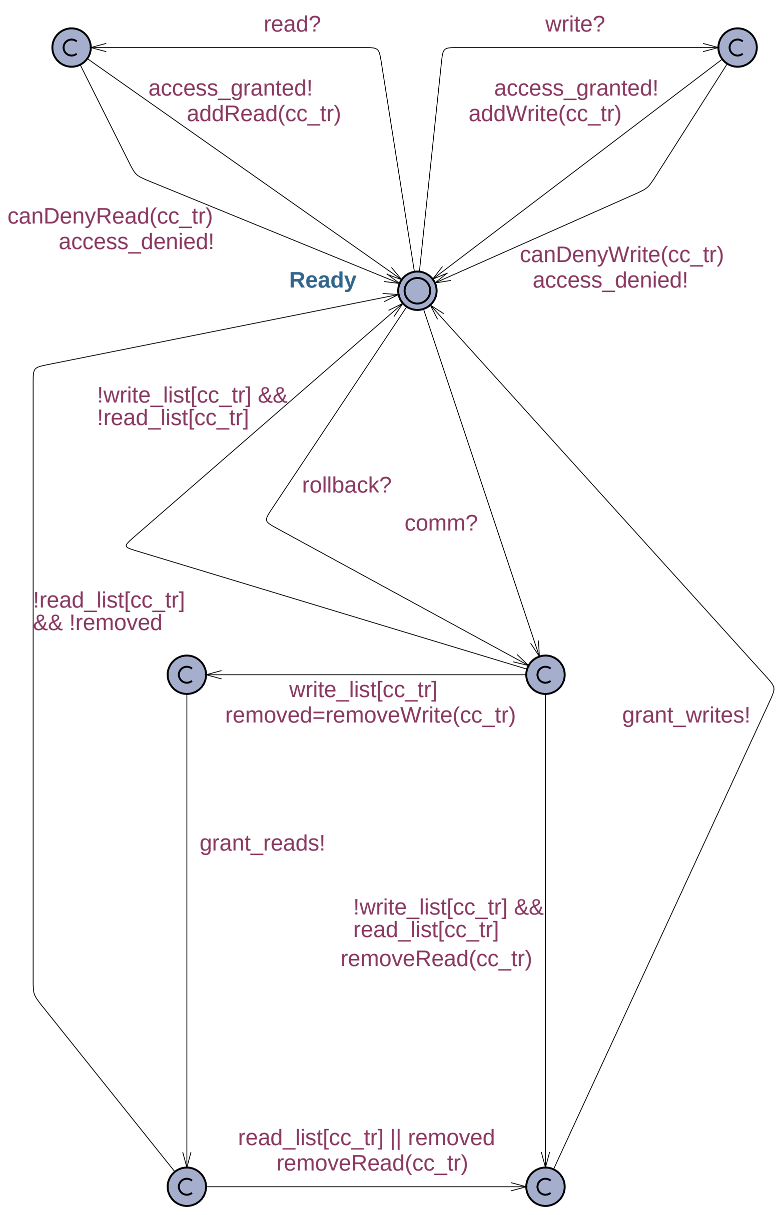


Figure 3.6: Concurrency control manager automaton for Two phase locking protocol

ilar for read locks.

One copy of generator, dispatcher and concurrency control automata and several copies of transaction generator are created using described templates in the system declaration. This completes our model of a simple real-time database system with Two phase locking concurrency control protocol.

3.5.2 Optimistic Protocol Sacrifice

Optimistic concurrency control protocols are quite different from pessimistic protocols. In this section we will show that they can be also modeled to some level of approximation using timed automata of Uppaal. We have chosen the protocol called Sacrifice. Using this protocol, all operations of the transaction are executed. If a conflict is detected in the validation phase and the validating transaction has a smaller priority, it is restarted and all previously made changes of data should be taken back. As in the case of Two phase locking protocol, we will not have any data modeled and rollback is almost the same as commit for us.

Transaction generator and dispatcher are independent on the chosen concurrency control protocol. We can use, in our model of a system with the protocol Sacrifice, dispatcher automata which has been described in the previous section.

An automaton representing transaction (depicted on the figure 3.7) starts in the state `Free`. Edges labeled `begin_tran?` and `end_tran!` synchronize with dispatcher exactly in the same way as in the case of Two phase locking.

The state `Working` represents an execution of one database operation. We could have more states for different operations but from our actual point of view all operations are similar. We use constants `MinOT` and `MaxOT` as bounds on execution time of an operation to model time consumed by database operations in real systems. The integer variable `op` is used as a counter of executed operations. We consider a constant number of operations in transaction. This number is given by `OP_COUNT` again.

When a number of non-executed operations is zero the edge to the location `Validation` may be used. The transaction is done and it has to be validated. Database records accessed by operations of a transaction are chosen randomly by the generator in V4DB. Hence it is in fact random situation that two transactions access the same record and a conflict occurs. In our automaton, we have two edges to represent both situations. The edge to

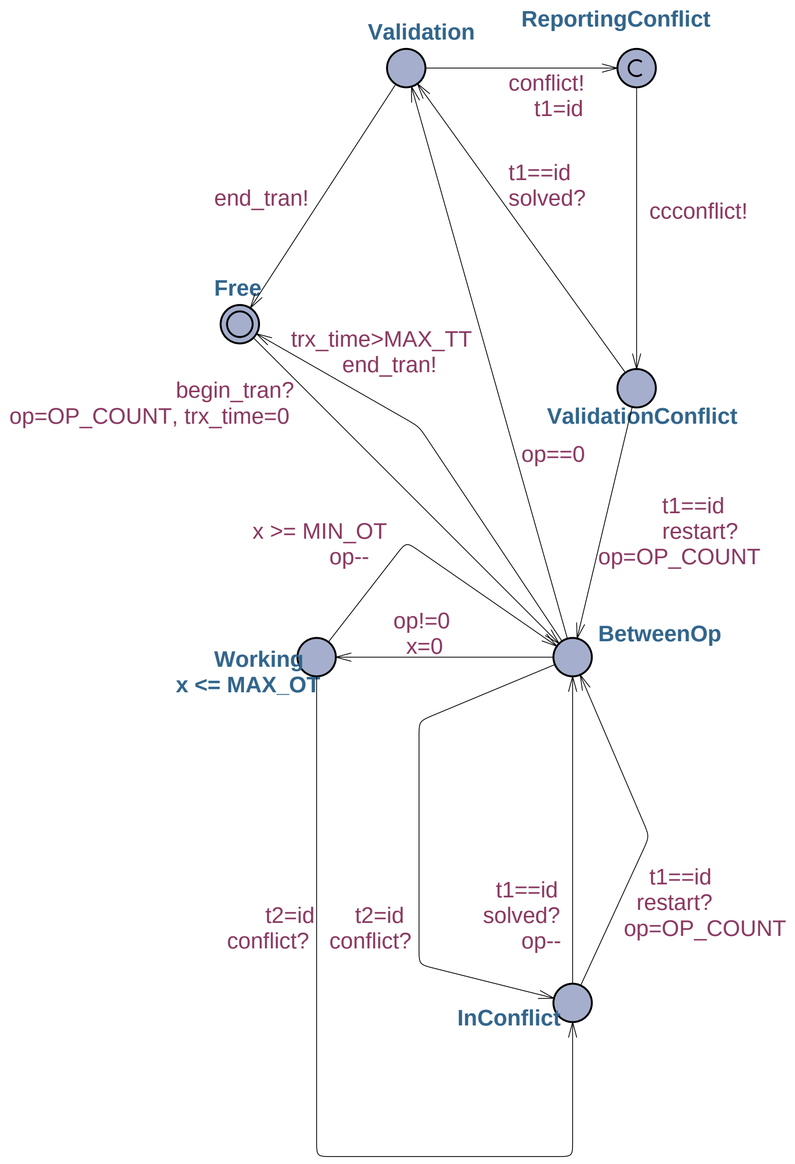


Figure 3.7: Transaction automaton for Sacrifice protocol

`Free` means that validation was successful.

The other edge from `Validation` represents a conflict situation. A conflict occurs only when some other, not yet validated, transaction accessed the same record during the execution of the validating transaction. Hence in our model there has to be some transaction automaton in the location `Working` or `BetweenOp` with nonzero count of executed operations. It is ensured by the use of synchronization through the channel `conflict`. The validating transaction automaton passes its identification to concurrency control manager automaton using a shared variable `t1`. The conflicting transaction passes its identification using a variable `t2`. Only one synchronization is permitted for an edge and we need not to synchronize only two transaction automata, but concurrency control automaton should be synchronized as well. Therefore, a committed location is used and the edge synchronizing with the concurrency control through the channel `ccconflict` is fired without any time delay.

Concurrency control manager automaton sends its responses through the channels `solved` and `restart`. An identification of an automaton for which the response is intended is in the variable `t1`. Synchronization through `solved` means that conflict was successfully solved and transaction can continue the validation or the execution. Synchronization through `restart` means that the transaction will be restarted. In this case `op` is set back to `OP_COUNT` because all operations should be executed again.

A clock variable `trx_time` measures the time of an execution of the transaction. It is reseted on the first transition of each transaction execution. If a deadline of a transaction is exceeded the transaction is aborted (transition from `BetweenOp` to `Free`). For simplicity, deadline is the same for all transactions – it is stored in a constant `MAX_TT`.

Last automaton of our simple model is concurrency control manager depicted on the Figure 3.8. We model Sacrifice protocol, hence the transaction with smaller priority is sacrificed (restarted) when a conflict occurs. In V4DB, priorities are randomly set by the generator. In the model, we could choose priority randomly when transaction automaton begins simulation of the execution. But particular automaton is chosen nondeterministically and it has unique identification number. Hence we can consider identification to be the priority. A bigger identification number means a bigger priority.

Concurrency control automaton synchronizes with validating transaction through the channel `ccconflict`. When this synchronization occurs, an identification of validating transaction is in the variable `t1` and an identification of conflicting transaction is in `t2`. Concurrency control sets the

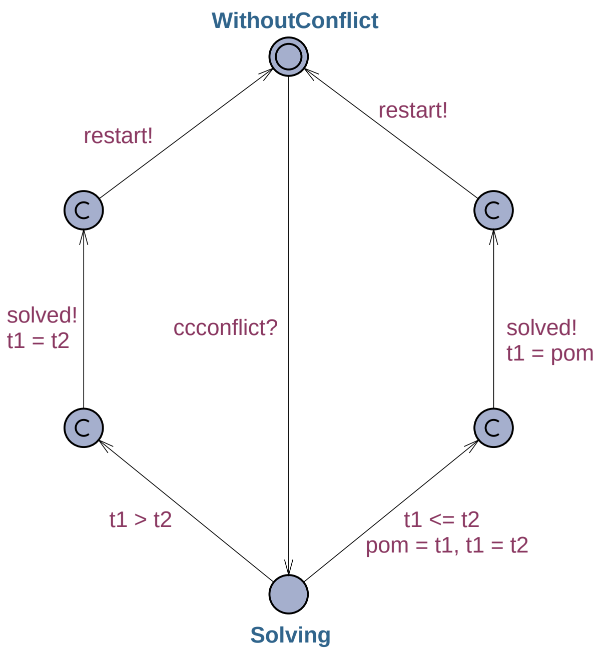


Figure 3.8: Concurrency control manager automaton

bigger of those identifications to `t1` and sends `solved!` and then sets the smaller identification to `t1` and sends `restart!`. Hence one transaction is restarted and one continues the execution. All three edges between locations `Solving` and `WithoutConflict` are fired without any time delay because of committed locations.

3.5.3 Verification

The main purpose of a use of verification tools such as Uppaal is verification, not the modeling. In this section we describe some properties that can be checked on our models.

In the query language of Uppaal we can express properties containing time constraints hence it is possible to express exceeding of critical time of transaction. We can use bounded integer values hence we can ask questions about priorities of transactions. But the language has not full expressivity of CTL logic – there are not operators 'Until' and 'Next', nested path formula are not allowed. Hence it is not possible to express all properties we are interested in.

A problem is also so called state-space explosion. The state is given by the location of all automata and values of all variables. Hence a number of reachable states is very large even for relatively simple model. There are many techniques for a state-space reduction. But they are not perfect and the problem remains. So it is important to choose proper level of abstraction such that model captures desired behavior and the state space is manageable by Uppaal.

Both our models approximate real system in such a way that database records and data are not captured. So we can not formulate queries such as if two transactions are changing same record concurrently. But we can check some other properties.

In the case of Two phase locking protocol, the usual problem are deadlocks. The modification of this protocol modeled in section 3.5.1 uses time limit on waiting time which should avoid deadlock. We can check if this modification is really deadlock-free. Corresponding formula in Uppaal's query language is

```
A[] not deadlock
```

This property is satisfied on the model. Because of the state-space explosion, the answer is given only for a small number of transaction automata (checked for 3 automata) and small number of operations in each transaction (checked for 2 operations).

We can ask if some transaction could exceed its deadline. The query is

```
E<> transactions(0).trx_time > MAX_TT
```

and this property is satisfied.

In the case of the protocol Sacrifice we can ask whether a non-validating transaction can be in conflict when no other transaction is in validation phase. For the simplicity, let us consider a model with two transaction automata only. The query is

```
E<> transactions(0).InConflict &&
    !(transactions(1).Validation ||
      transactions(1).ReportingConflict ||
      transactions(1).ValidationConflict)
```

and this property is not satisfied. We can also ask whether a transaction with smaller priority can be validated successfully and completed when it was in conflict with a transaction with bigger priority. If this situation could occur, there was a state reachable in the model where transaction automaton with bigger priority is in the location `InConflict` and the other automaton is in the location `Validation`. Hence the query is

```
E<> transactions(1).InConflict &&
    transactions(0).Validation
```

and it is not satisfied.

Other formulation of the query is if the transaction with the biggest priority may be restarted. To check this we can slightly modify the model. We add new location called `Restart` of transaction automaton and transition from the location `ValidationConflict` with a guard `t1==id` and a synchronization `restart?` is redirected to the new location. Then the query is

```
E<> transactions(2).Restart
```

This property is not satisfied too.

We tried more queries on those models. For some queries some small modifications were needed. For other queries the model have to be changed almost completely, because chosen abstraction does not capture the demanded behavior. It is not possible to create a model of RTDBMS such that it captures all interesting properties and behavior and all queries on it are manageable by Uppaal.

3.6 Modeling of Concurrency Control Protocols

In the previous section we presented some models of V4DB system with emphasis on concurrency control. There were several parts of the system modeled which are not so important for concurrency control protocols but they are important for RTDBMS system. The problem is that every added automaton increases the state space. This caused that even for small amount of transaction automata some properties were not manageable by Uppaal.

In this section we describe some models that concern just with concurrency control protocol itself. Techniques used in the models can be used in future by authors of new concurrency control to verify their design. In [35] authors also used Uppaal for verification of their new pessimistic protocol, but their models were quite simple aimed on this concrete protocol and does not suggest any ideas and possibilities for modeling of other protocols.

For the models of protocols we have chosen pessimistic 2PL protocol (Subsection 3.6.1), its modification that avoids deadlocks (Subsection 3.6.2) and modification called 2PL High priority (Subsection 3.6.3) and two optimistic protocols – Broadcast commit (Subsection 3.6.4) and Sacrifice (Subsection 3.6.5). Mentioned models of pessimistic protocols were previously published at a conference [24] and all models (pessimistic and optimistic) in journal [25].

3.6.1 Pessimistic Protocol Two Phase Locking

The protocol Two phase locking (2PL) was modeled from other perspective in Section 3.5.1 and there is also short description of its properties. Now we will show one possible model of this protocol itself without other parts of RTDBMS system as dispatcher or transaction generator are.

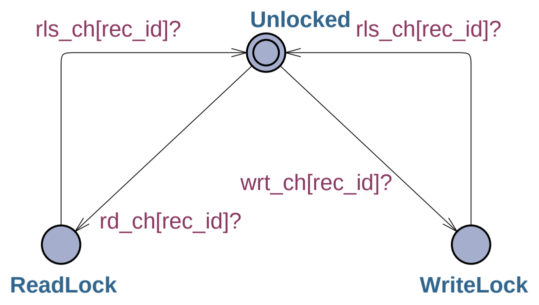


Figure 3.9: Automaton representing a record in a database

Of course, presented model is not the only one possible. It consists of several timed automata created using two templates. One type of automata (i.e. one template) represents data records in a database. A graph of the automaton of this template is shown on the Figure 3.9. From the concurrency control point of view, the value stored in a record is not important. Hence we will not catch it in our model. We need only representation of locks on records.

Each record automaton has an integer ID stored in `rec_id`. There are three locations corresponding to two types of locks and to an unlocked state. Channels `rd_ch[x]` and `wrt_ch[x]` are used for requests for read and write locks on record x. Channel `rls_ch[x]` is for release (unlock) request.

The graph of automaton of the second template is shown on the Figure 3.10 is intended to create automata representing active transactions in the system. In almost all existing database systems, there is a bounded number of active transactions (predispatcher module of RTDBMS holds the queue of incoming transactions and passes them to a dispatcher in such a way that it avoids overloading). So it is possible to represent one active (i.e. currently in execution) transaction as one automaton. After successful end of a transaction the same automaton represents some other transaction.

For simplicity, all transactions are supposed to have the same number of operations (given by a constant `OPERATIONS`). Each operation accesses one record (i.e. needs one lock). A type of operations and an accessed record is for a real RTDBMS in fact random because it is determined outside of the RTDBMS. We do not need to model concrete operations, only locks. The record is chosen nondeterministically using select `rec:rec_id_t`. The operation is then immediately (due to a committed location) chosen nondeterministically by using one of three possible edges. If a transaction owns

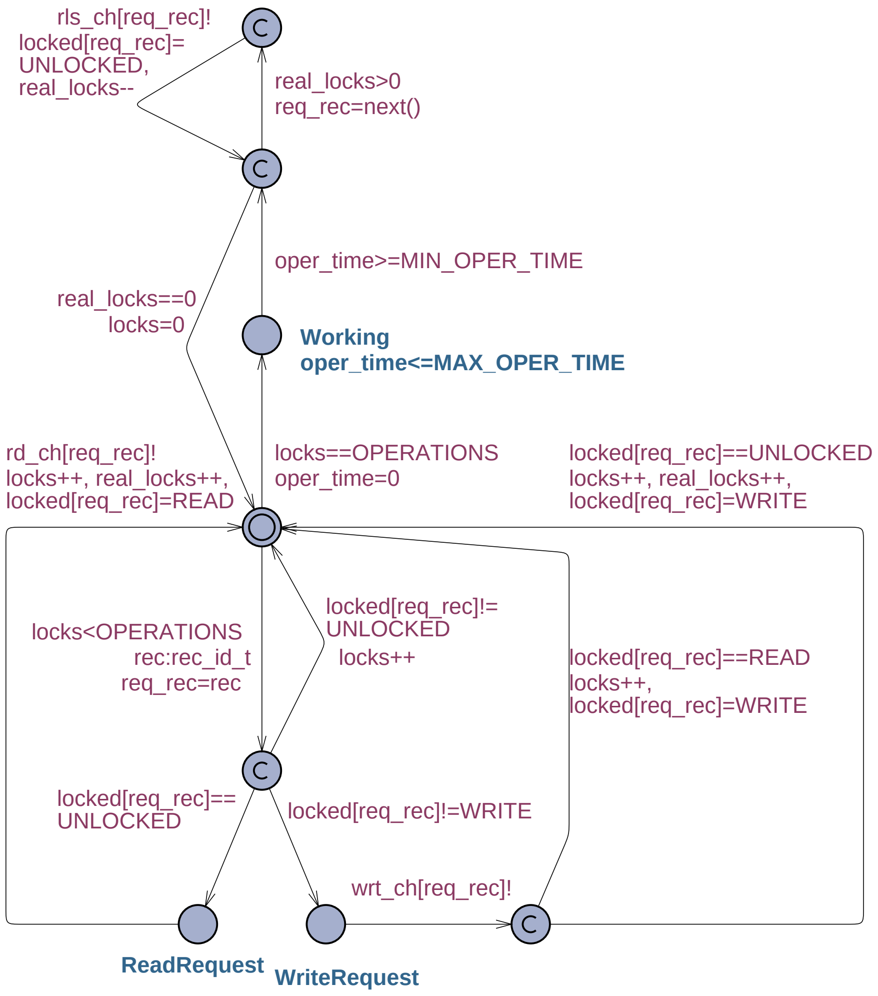


Figure 3.10: Transaction automaton for Two phase locking protocol

the demanded type of a lock on the accessed record, it does not asks the lock again. If it has only a read lock, it can ask a change to a write lock. In the global array `locked` is stored the information about owned locks, local variable `locks` contains the number of operations for which locks are gained and local variable `real_locks` the number of records locked by this transaction.

If the transaction has all necessary locks, the automaton is in a location `Working`. This represents execution of database operations. The time spent in this location is bounded by the constants `MIN_OPER_TIME` and `MAX_OPER_TIME`. After the execution, all locks are released instantly (using committed states and edges between them).

The described model simulates basic variant of 2PL protocol where a deadlock can arise when some transactions wait mutually for locks granted to other transactions. A small modification where transactions exceeding their deadline may be aborted can solve the problem with deadlocks.

3.6.2 Modification of a Model of Two Phase Locking Protocol

It is common in real-time databases that a transaction has some deadline. If this deadline is missed, transaction could get (nearly) useless and can be aborted for the sake of other transactions to meat their deadline. This we can use in the modification of 2PL protocol to avoid deadlocks – some of transactions causing a deadlock reaches its deadline, it is aborted, its locks are released and some waiting transaction can possibly finish successfully.

A template for database record automata for a model of this protocol remains the same as in the previous model (Figure 3.9). A changed transaction automata template is shown on Figure 3.11.

There is a local clock variable `tr_time` added. It measures time from the beginning of transaction execution. If a transaction is waiting for a lock and it reaches its deadline (for simplicity same for all transactions given by a constant `DEADLINE`), it can be aborted. This means that all locks previously granted to this transaction are released. It is done by added state `Abort` and edges guarded by `tr_time>DEADLINE` leading to this state. Function `next()` returns the smallest identification number of a record for which this transaction is holding a lock.

We can use Uppaal to verify that this solution is really sufficient to avoid

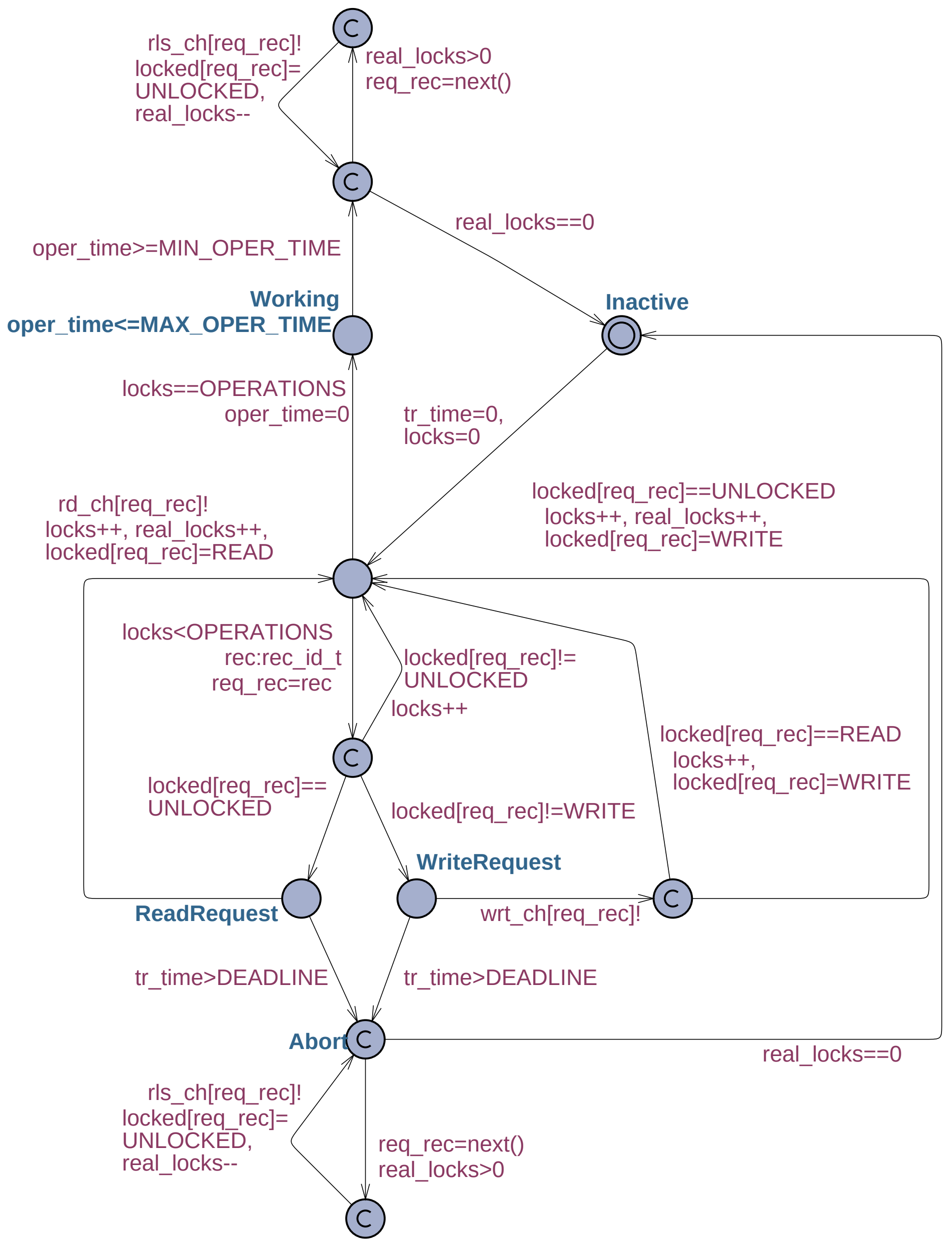


Figure 3.11: Transaction automaton for modification of Two phase locking protocol

deadlock. For Uppaal, reachability properties are more suitable. So the formula

```
E<> deadlock
```

means that deadlock is reachable in the model and this property is not satisfied. Hence it is verified that the system is deadlock-free. Of course, the same formula is not satisfied for the previous model and Uppaal can show a trace leading to a deadlocked state as a counter-example.

3.6.3 Pessimistic Protocol Two Phase Locking High Priority

The last modification of our model is for a protocol Two phase locking high-priority (2PL–HP). If a lock is requested by a transaction with a higher priority, the transaction with a lower priority holding this lock may be restarted.

For this model we change both automata templates. A graph of automaton of the template representing database records is depicted on Figure 3.12.

Two arrays – `locked` and `lock_type` – are defined in the global declarations. The first one contains information about transactions holding locks for particular records and the latter one contains information about types of particular locks. `lock_holder` is a local variable of one record automaton used for the ID of transaction holding the lock on this record. As almost all is chosen nondeterministically (including the order of activating particular transaction automata), we can model priorities using ID numbers of transaction automata – higher ID means higher priority.

If the automaton is in the location `Unlocked`, all requests passed through channels `rd_ch` and `wrt_ch` are answered immediately through a channel `granted` and informations about this lock are saved to above mentioned arrays and variable.

If the automaton is in the location `WriteLock` or `ReadLock` and a new request arrives, it has to restart a transaction holding the lock (priorities are checked before the request in a transaction automaton). Restarted transaction x is contacted using a channel `restart[x]`. Then the lock is granted to a requesting transaction using channel `grant`. If a write lock is requested from the location `ReadLock`, there is a possibility to grant it without any other activity (except for the change of a type of lock in `lock_type` array). This is done when requesting transaction (`req_trans`) is the same as the current holder of the read lock (`lock_holder`).

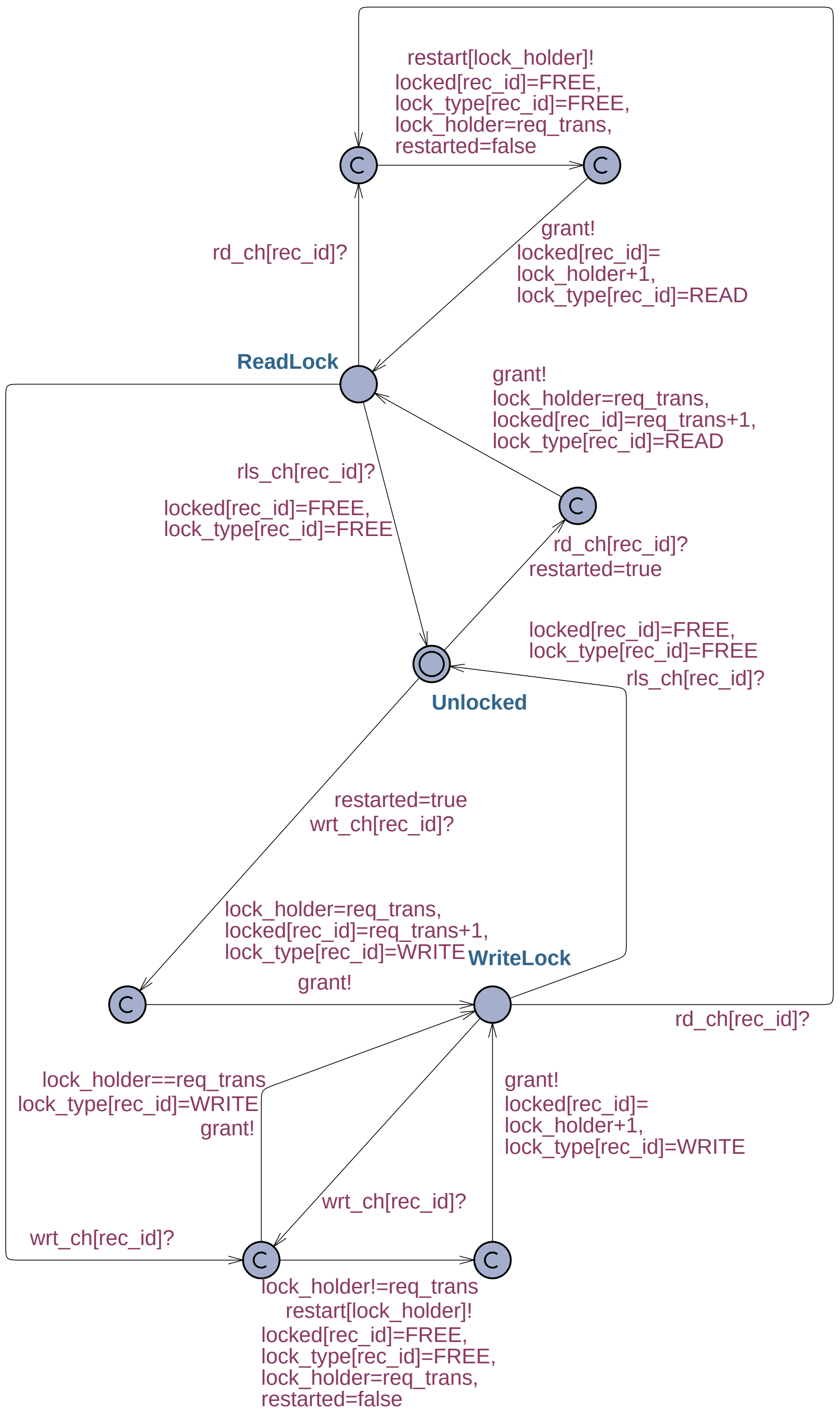


Figure 3.12: Automaton representing a record in a database for 2PL-HP protocol

The transaction automata template has to be changed as well. The modified version is depicted on the Figure 3.13.

There are added edges leading to a new location `Restart` from all locations where an automaton can be during passing of time. All those edges have synchronization label `restart[trans_id]`. In this way a transaction (with an ID stored in a variable `trans_id`) can be restarted anytime by a record automaton. In the location `Restart` all previously gained locks are released and the waiting transaction with higher priority is notified using global boolean variable `restarted`. A function `next()` returns the smallest ID number of a record on which is the transaction actually holding a lock.

Requests for locks are guarded. A requested record (specified by the variable `req_rec`) has to be unlocked or locked by a transaction with smaller priority. It comes handy to use 0 (constant `FREE` is defined as 0) in the array `locked[]` for unlocked records and ID of transaction automaton (i.e. the priority of transaction) plus one for a lock holder. Than the guard

```
trans_id+1 > locked[req_rec]
```

is true whenever the transaction holding a lock on `req_rec` has a smaller priority or this record is unlocked.

As in the previous case, although for this model Uppaal can verify that it is deadlock-free. We can use the same formula

```
E<> deadlock
```

and the answer is negative (i.e. no deadlock is reachable).

Furthermore we can check e.g. if the transaction with the highest priority could be possibly restarted. The number of transaction automata is given using a constant `TRANSACTIONS`. Hence the greatest ID number (this means priority too) is `TRANSACTIONS-1`. The formula is

```
E<> Transaction(TRANSACTIONS-1).Restart
```

and it is not satisfied, i.e. this transaction could not be restarted. For all other transactions x the formula

```
E<> Transaction(x).Restart
```

is satisfied.

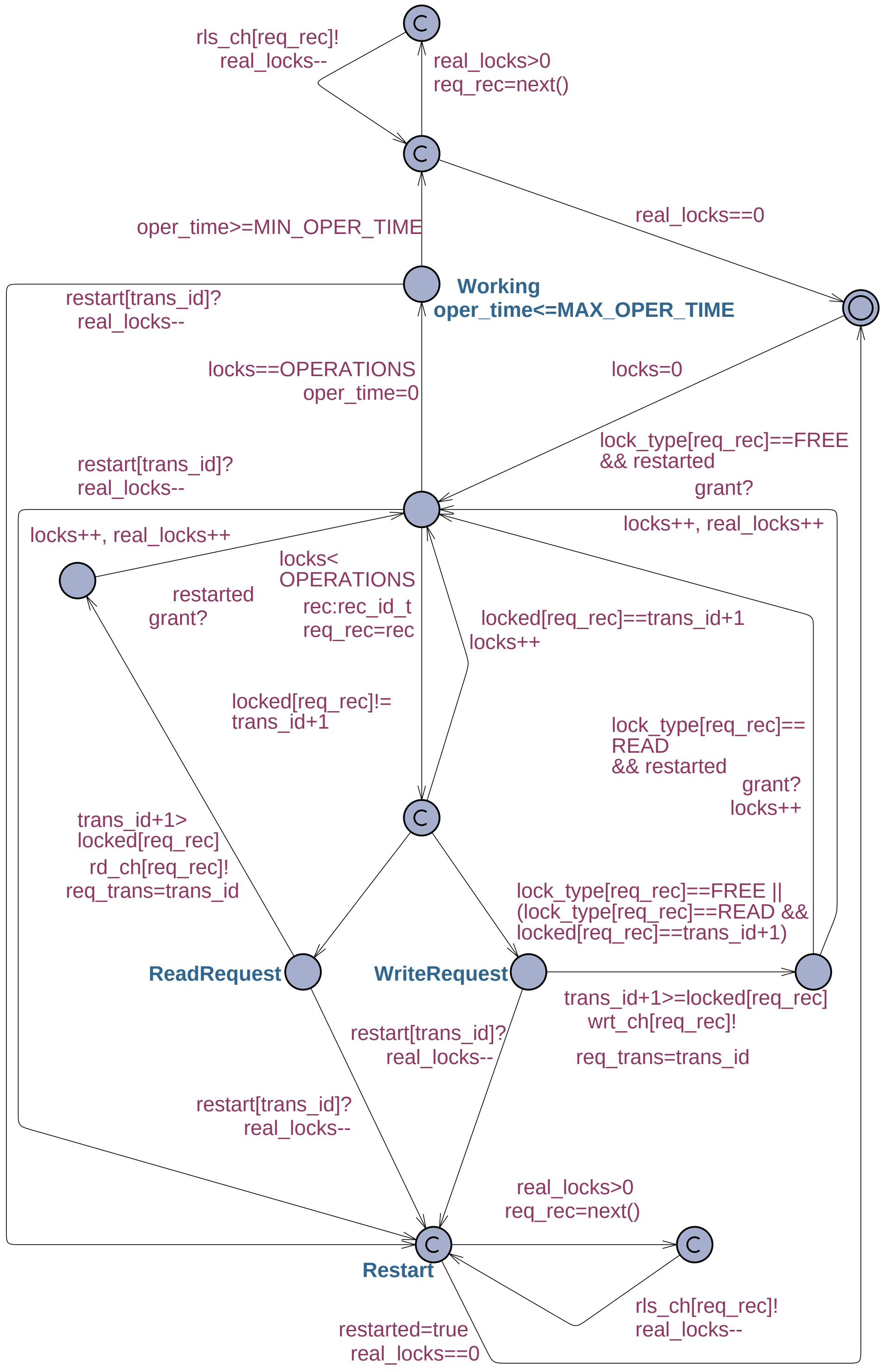


Figure 3.13: Transaction automaton for 2PL-HP protocol

3.6.4 Optimistic Protocol Broadcast Commit

Optimistic concurrency control protocols are quite different from pessimistic protocols. In this section we will show that they can be also modeled to some level of approximation using timed automata of Uppaal. First we have chosen the protocol called Broadcast commit. Using this protocol, all operations of the transaction are executed on a local copy of records. Then a validation phase comes. If a conflict is detected with some running transaction, the running transaction is restarted and all previously made changes of data should be taken back. Validating transaction then writes all its changes on records to the global copy of database. So every transaction which reaches validation phase is finished successfully and cannot be restarted.

For the model of this protocol we use only one template. An automaton representing transaction (depicted on the figure 3.14) starts in the state `Free`.

The state `Working` represents execution of one database operation. We could have more states for different operations but from our actual point of view all operations are similar. We use constants `MinOT` and `MaxOT` as bounds on execution time of an operation to model time consumed by database operations in real systems. The integer variable `op` is used as a counter of executed operations. We consider constant number of operations in a transaction. This number is given by `OP_COUNT`.

When the number of non-executed operations is zero, the edge to the location `Validation` may be used. The transaction is done and it has to be validated. Database records accessed by operations are given from outside of concurrency control unit (by incoming transactions in the real system or randomly by the transaction generator in experimental DBMS V4DB). Hence, for concurrency control protocol, it is in fact random situation that two transactions access the same record and a conflict occurs. In our automaton, we have two edges to represent both situations. The edge to `Free` means that validation was successful. Boolean variable `success` is set to `true` in this case. This variable can be used for verification purposes – we can ask whether particular transaction finished successfully. As we do not represent actual stored data, we can also skip commit operation (transfer of changes from local copy of database to the global one).

The other edge (loop) from `Validation` represents conflict situation. Conflict occurs only when some other, not yet validated, transaction accessed the same record. Hence in our model there has to be some transaction

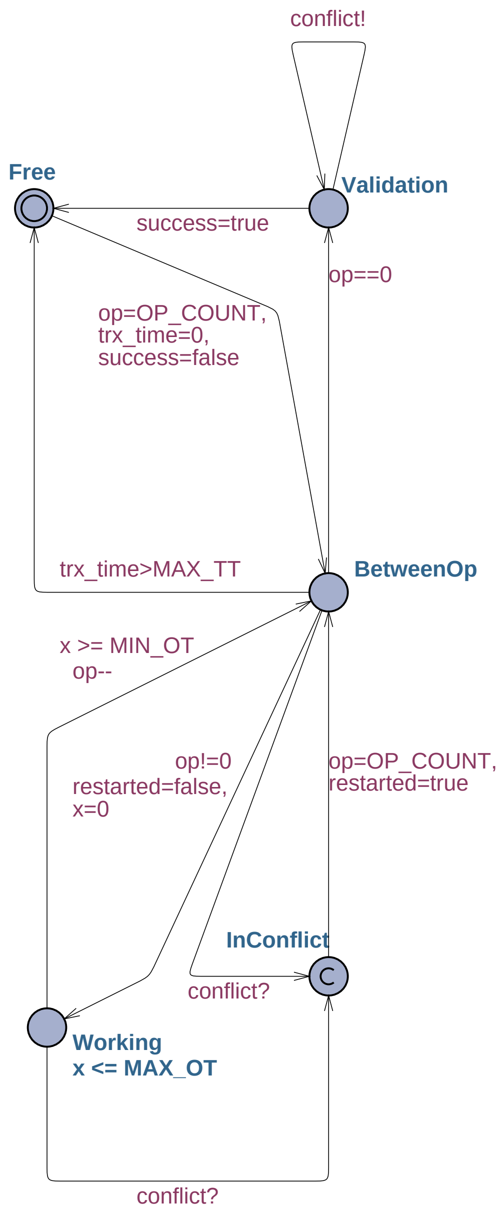


Figure 3.14: Transaction automaton for optimistic protocol Broadcast commit

automaton in the location `Working` or `BetweenOp` with nonzero count of executed operations. It is ensured by the use of synchronization through the channel `conflict`. The other transaction (receiving signal through the channel `conflict?`) is restarted and the validating transaction continues its validation process. A transaction could be in conflict with more than one other transactions and our model enables this (communication through the channel `conflict` can be done more times).

A clock variable `trx_time` measures time of execution of the transaction. It is reseted on the first transition of each transaction execution. If a deadline of a transaction is exceeded the transaction is aborted (transition from `BetweenOp` to `Free`). For simplicity, deadline is the same for all transactions – it is stored in a constant `MAX_TT`.

3.6.5 Optimistic Protocol Sacrifice

Protocol Sacrifice is a modification of Broadcast commit. It introduces sacrifice of a transaction according to its priority. If a conflict is detected in a validation phase and validating transaction has smaller priority, it is restarted and all previously made changes of data should be taken back. If the validating transaction has at least the same priority, the conflicting transaction is restarted. As in the case of Two phase locking protocol or Broadcast commit, we will not have any data modeled and rollback (taking back all changes done by the transaction) is almost the same as commit for us.

A graph of an automaton of a template representing transaction in our model of this protocol is depicted on the Figure 3.15.

This protocol is modification of Broadcast commit hence automata for those two protocols are quite similar. As in the case of 2PL-High priority protocol, we will for simplicity represent priorities using the `id` variable of the automaton.

If a conflict is detected in a validation phase, the automaton reaches a state `ValidationConflict`. There are two possibilities (i.e. outgoing edges). The one with a guard `id>=conflicted` means that the validating transaction has bigger priority than the conflicting one (its ID is stored in the variable `conflicted`) and nothing happens. In the opposite case, validating transaction is restarted. In our simplification it means that we set the number of nonexecuted operations (variable `op`) back to the initial number and we store the information about restart into the boolean variable `restarted`.

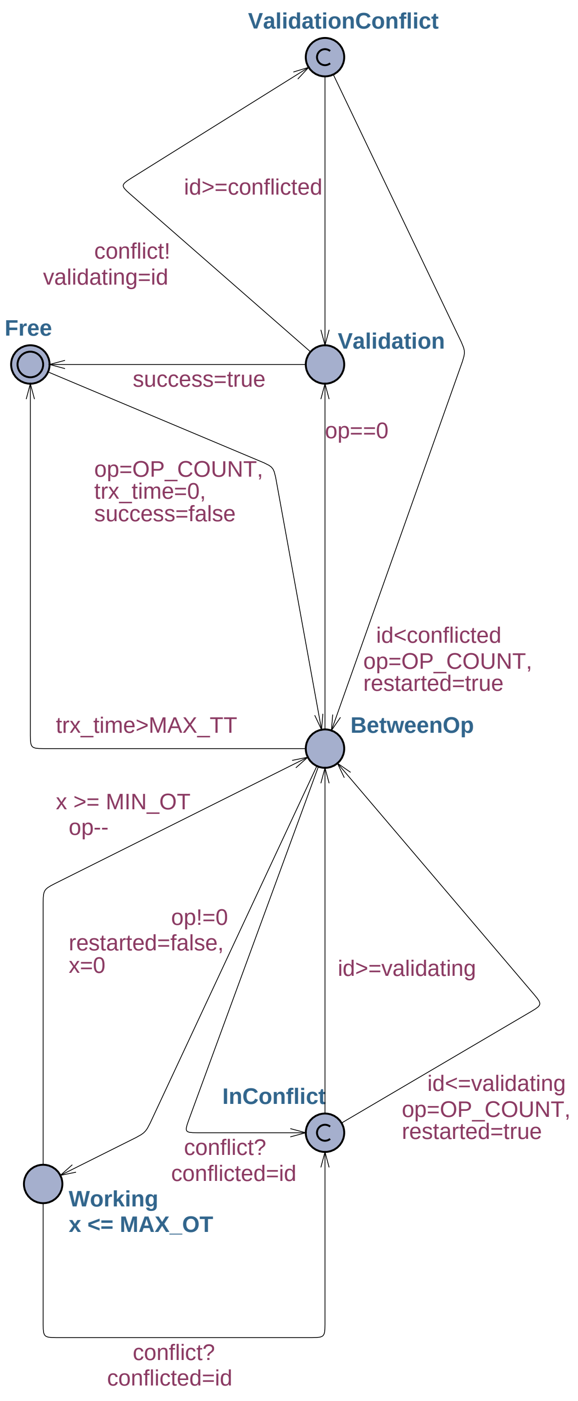


Figure 3.15: Transaction automaton for optimistic protocol Sacrifice

The counterpart for the transaction in execution phase conflicted with some transaction in the validation phase is realized using the state `InConflict` and its adjacent edges. If the running transaction has bigger priority, nothing happens (represented by the edge with a guard `id>validating` where `validating` is global variable containing id of the validating transaction). In the opposite case, this transaction is restarted. This is represented similarly as in the case of restart of validating transaction.

Also for optimistic protocols, it gives sense to verify that they are deadlock-free. The formula remains the same as for pessimistic protocols:

```
E<> deadlock
```

and the answer is negative for both modeled optimistic protocols (i.e. no deadlock is reachable).

Furthermore we can check e.g. if the transaction with the highest priority could possibly be restarted. The number of transaction automata is again given using a constant `TRANSACTIONS`. Hence the greatest ID number (this means priority too) is `TRANSACTIONS-1`. We use also our boolean variable `restarted`. The formula is

```
E<> Transaction(TRANSACTIONS-1).restarted
```

It is satisfied for Broadcast commit and not satisfied for Sacrifice. Using similar queries we can show, that transactions with any priority can possibly be restarted using Broadcast commit. In the case of protocol Sacrifice, the transaction with the greatest priority (ID) can not be restarted in any run.

For both protocols we can verify that all transactions can finish successfully at the same time using formula (in the case of 3 transactions):

```
E<> Transaction(0).success &&
  Transaction(1).success &&
  Transaction(2).success
```

Chapter 4

Conclusion

This thesis concentrates on two quite different areas of formal verification. The first area is complexity of some equivalence-checking problems, i.e., the problems where, for given (descriptions of) transition systems, we ask whether they are equivalent with respect to some notion of equivalence. We concerned with one of the most important equivalences – the bisimulation equivalence and with systems specified primarily as Basic Parallel Processes.

The second area is a practical use of model checking tool Uppaal on some parts of real time database system.

In the next section is an overview of the results presented in the thesis and it is followed by some open problems and suggestions on further research.

4.1 Summary of the Results

Basic Parallel Processes (BPP) were studied in Chapter 2. Using techniques inspired by Jančar's paper [16], some new algorithms were suggested to some problems:

- A polynomial time algorithm for deciding bisimulation equivalence between a BPP and a finite-state system (with time complexity $O(n^4)$).
- $O(n^3)$ algorithm for deciding bisimilarity of two normed BPPs. This problem was known to be polynomial but without known degree of polynomial.

- A polynomial space algorithm for regularity of BPP. This problem was known to be decidable and PSPACE-hard.

Moreover we presented a polynomial time algorithm for deciding bisimilarity between normed BPP and normed BPA with time complexity $O(n^7)$. The best previously known algorithm for this problem was exponential.

In Chapter 3 we were interested in modeling of concurrency control protocols used in real-time database system using a verification tool Uppaal and in checking some simple properties expressed as temporal logic formula on modeled protocols. Several models of different pessimistic and optimistic protocols were suggested. Presented models show some general possibilities how to model parts of real-time database systems in verification tools using modeling language not directly intended for database systems.

4.2 Open Problems and Further Research

In the area of deciding bisimilarity are still some interesting open problems. For example the problem of decidability of *weak* bisimulation equivalence on BPP remains open. There are also some questions concerning complexity. For example the problem of bisimilarity of BPP and BPA processes in general (in this thesis we discussed normed subcase) is known to be decidable but there are not any complexity estimations. We have even some preliminary results for this problem leading to triple exponential upper bound, we hope to improve it.

We prepare a journal article containing summary of our contributions to problems related to bisimilarity of BPP. An algorithm for bisimilarity of BPP and finite state system should be improved there to allow time complexity $O(n^3 \log n)$. The analysis of this algorithm in the case of normed BPP and finite state system seems to lead to time complexity $O(n^2 \log n)$.

In the area of model checking of real-time database systems, there are also many possibilities for further research. It is, e.g., possible to model and verify other parts of RTDBMS or use other verification tool. In the long-term research, it is possible to design some new verification tool specialized on (real-time) databases.

Kapitola 5

Závěr

Tato práce se zaměřuje na dvě, docela rozdílné, oblasti formální verifikace. V prvním případě jde o složitost některých problémů z oblasti ověřování ekvivalencí, tj. problémů, kde se pro dané popisy systémů ptáme, jestli jsou ekvivalentní vzhledem k nějaké zvolené ekvivalenci. Zaměřili jsme se na jednu z nejvýznamnějších ekvivalencí – bisimulační ekvivalenci a na systémy specifikované především jako základní paralelní procesy.

Druhou oblastí je praktické využití verifikačního nástroje Uppaal na některé části real-time databázových systémů.

V následujících sekcích budou shrnuty konkrétní, v práci prezentované, výsledky. Poté následuje sekce věnovaná otevřeným problémům a možnostem dalšího výzkumu.

5.1 Přehled výsledků

V kapitole 2 jsme se zabývali základními paralelními procesy (Basic Parallel Processes – BPP). S využitím technik inspirovaných DD-funkcemi z článku [16] jsme navrhli nové algoritmy pro některé problémy:

- Algoritmus pracující v polynomiálním čase (s horním odhadem časové složitosti $O(n^4)$) pro rozhodování bisimulační ekvivalence mezi BPP a konečně stavovým systémem.
- Algoritmus pracující v čase $O(n^3)$ pro rozhodování bisimilarity mezi dvěmi normovanými BPP systémy. Pro tento problém již byl znám

polynomiální algoritmus (založený na jiných myšlenkách), ale nebyl k dispozici žádný přesný odhad stupně polynomu.

- Algoritmus rozhodující regularitu BPP v polynomiálním prostoru. Protože byla známa PSPACE obtížnost tohoto problému, tak jsme dostali jeho PSPACE úplnost.

Dále byl prezentován polynomiální algoritmus pro rozhodování bisimulační ekvivalence mezi normovanými BPP a normovanými BPA procesy s časovou složitostí $O(n^7)$. Nejlepší předtím známý algoritmus pro tento problém byl exponenciální.

V kapitole 3 jsme se zabývali modelováním protokolů pro řízení souběžného přístupu k datům v real-time databázových systémech. Modelování probíhalo v nástroji Uppaal, který potom za použití metod ověřování modelů umožnil ověřit nějaké jednoduché vlastnosti vyjádřené jako formule temporální logiky. Bylo navrženo několik modelů různých verzí pesimistických i optimistických protokolů. Prezentované modely ukazují nějaké obecné možnosti, jak mohou být modelovány různé části real-time databází v modelovacích jazycích verifikačních nástrojů, které nejsou přímo určeny k práci s databázovými systémy.

5.2 Otevřené problémy a další výzkum

V oblasti rozhodnutelnosti bisimulační ekvivalence jsou stále některé zajímavé otevřené problémy. Například rozhodnutelnost problému tzv. *slabé* bisimulační ekvivalence na BPP je stále otevřená, i když pro (silnou) bisimulační ekvivalenci je na stejných systémech známá PSPACE úplnost. Zajímavé jsou také některé otázky složitosti problémů, jejichž rozhodnutelnost je známá. Například bychom se v budoucnu chtěli věnovat problému rozhodování bisimilarity obecných BPA a BPP systémů (v práci byl tento problém prezentován jen pro normovaný případ těchto systémů). Tento problém je znám jako rozhodnutelný, ale není znám žádný odhad složitosti. Máme v tomto směru nějaké předběžné výsledky vedoucí k trojitě exponenciálnímu hornímu odhadu časové složitosti, ale doufáme ještě v další zlepšení.

Připravujeme také článek do časopisu shrnující náš přínos v oblasti spojené s bisimilaritou na BPP. Algoritmus pro bisimilaritu mezi BPP a konečně stavovým systémem by v tomto článku měl být vylepšen, aby umožnil odhad časové složitosti $O(n^3 \log n)$. Analýza tohoto algoritmu pro případ, kdy dané BPP je normované, povede ke složitosti asi $O(n^2 \log n)$.

V oblasti použití metod ověřování modelů na real-time databázové systémy se také nabízí mnoho možností dalšího výzkumu. Je například možné zkusit modelovat a verifikovat jiné části real-time databázových systémů nebo využít jiný verifikační nástroj. V dlouhodobějším výhledu je možné navrhnou i nějaký verifikační nástroj specializovaný na (real-time) databáze.

Appendix A

List of Publications

The following list contains reviewed publications of the author in the chronological order:

- Jančar, P., Kot, M., *Bisimilarity on normed Basic Parallel Processes can be decided in time* $O(n^3)$, in: *Proceedings of the Third International Workshop on Automated Verification of Infinite-State Systems - AVIS2004, Barcelona, 2004* [17]
- Kot, M., *Some Problems Related to Bisimilarity on BPP*, in: *Proceedings of Movep'04, Universit Libre de Bruxelles, Brussels, 2004, p. 96-102*
- Kot, M., Sawa, Z., *Bisimulation equivalence of a BPP and a finite-state system can be decided in polynomial time*, in: *Electronic Notes in Theoretical Computer Science, 2005, vol. 138, issue 3 (Proceedings of the 6th International Workshop on Verification of Infinite-State Systems –INFINITY 2004), p. 49-60, ISSN 1571-0661* [26]
- Kot, M., *Complexity of some Bisimilarity Problems between BPP and BPA or Finite-State System*, in: *Proceedings of Movep'06, University of Bordeaux-1, Bordeaux, 2006, p. 318-323*
- Kot, M., *Notes on Modeling of Real-Time Database System V4DB in Verification Tool Uppaal*, in: *MEMICS proceedings (MEMICS 2007 - Third Doctoral Workshop on Mathematical and Engineering Methods in Computer Science), Brno, 2007, p. 82-89, ISBN 978-80-7355-077-6* [23]

- Jančar, P., Kot, M., Sawa, Z., *Normed BPA vs. Normed BPP Revisited*, in: *Proceedings of CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, Lecture Notes in Computer Science 5201, Springer, 2008, p. 434-446, ISBN 978-3-540-85360-2, ISSN 0302-9743 (Print), ISSN 1611-3349 (Online)*, Best paper award [19]

- Kot, M., *Modeling Real-Time Database Concurrency Control Protocol Two-Phase-Locking in Uppaal*, in: *Proceedings of the International Multiconference on Computer Science and Information Technology, Volume 3 (2008), IEEE Computer Society Press, 2008, p. 673-678, ISBN 978-83-60810-14-9, ISSN 1896-7094* [24]

- Kot, M., *Modeling selected real-time database concurrency control protocols in Uppaal*, in: *Innovations in Systems and Software Engineering, Volume 5, Number 2, June 2009, Springer, London, p. 129-138, ISSN 1614-5046 (Print), ISSN 1614-5054 (Online)* [25]

- Jančar, P., Kot, M., Sawa, Z., *Complexity of Deciding Bisimilarity between Normed BPA and Normed BPP*, in: *Information and Computation, Elsevier, 28 p., ISSN 0890-5401, to appear* [18]

Some results were also presented at student workshop:

- Kot, M., Onderek, O., *Two known algorithms for checking bisimilarity of normed BPPs*, in: *Sborník semináře Wofex 2003, Ostrava, 2003*

- Kot, M., *Complexity of deciding bisimilarity of nBPP and bisimilarity of BPP with finite-state system*, in: *Proceedings of the 2nd annual workshop WOFEX 2004, Ostrava, 2004, p. 310-315, ISBN 80-248-0596-0*

- Kot, M., *Regularity of BPP is PSPACE-complete*, in: *Proceedings of the 3nd annual workshop WOFEX 2005, Ostrava, 2005, p. 393-398, ISBN 80-248-0866-8* [22]

Bibliography

[1] Aho, A. V., J. E. Hopcroft and J. D. Ullman, "Design and Analysis of Computer Algorithms," Addison-Wesley Reading, 1974.

[2] Baeten, J. C. M. and W. P. Weijland, *Process algebra*, Cambridge Tracts in Theoretical Computer Science **18** (1990).

[3] Behrmann, G., A. David and K. G. Larsen, "A tutorial on Uppaal," Available on-line (September 15, 2009).
URL `http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf`

[4] Bérard, B., M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen and P. McKenzie, "Systems and Software Verification: Model-Checking Techniques and Tools," Springer, 2001.

[5] Bergstra, J. A. and J. W. Klop, *Algebra of communicating processes with abstraction*, Theoretical Computer Science **37** (1985), pp. 77–121.

[6] Burkart, O., D. Caucal, F. Moller and B. Steffen, *Verification on infinite structures*, in: *Handbook of Process Algebra* (2001), pp. 545–623.

[7] Burkart, O., D. Caucal and B. Steffen, *An elementary decision procedure for arbitrary context-free processes*, in: *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, LNCS **969** (1995), pp. 423–433.

[8] Christensen, S., "Decidability and Decomposition in Process Algebras," Ph.D. thesis, The University of Edinburgh (1993).

[9] Clarke, E. M., O. Grumberg and D. A. Peled, "Model Checking," The MIT Press, 1999.

[10] David, A. and T. Amnell, "Uppaal2k: Small Tutorial," Available on-line (September 15, 2009).
URL `http://www.it.uu.se/research/group/darts/uppaal/tutorial.ps`

[11] Emerson, E. A., *Temporal and modal logic*, Handbook of Theoretical Computer Science **B** (1991), pp. 995–1072.

[12] Henzinger, T. A., X. Nicollin, J. Sifakis and S. Yovine, *Symbolic model checking for real-time systems*, Information and Computation **111** (1992), pp. 394–406.

[13] Hirshfeld, Y. and M. Jerrum, *Bisimulation equivalence is decidable for normed process algebra*, in: *Proceedings of 26th International Colloquium on Automata, Languages and Programming (ICALP'99)*, Lecture Notes in Computer Science **1644** (1999), pp. 412–421.

[14] Hirshfeld, Y., M. Jerrum and F. Moller, *A polynomial algorithm for deciding bisimilarity of normed context-free processes*, Theoretical Computer Science **158** (1996), pp. 143–159.

[15] Hirshfeld, Y., M. Jerrum and F. Moller, *A polynomial-time algorithm for deciding bisimulation equivalence of normed basic parallel processes*, Mathematical Structures in Computer Science **6** (1996), pp. 251–259.

[16] Jančar, P., *Strong bisimilarity on basic parallel processes is PSPACE-complete*, in: *Proc. 18th LiCS* (2003), pp. 218–227.

[17] Jančar, P. and M. Kot, *Bisimilarity on normed Basic Parallel Processes can be decided in time $O(n^3)$*, in: *Proceedings of the Third International Workshop on Automated Verification of Infinite-State Systems – AVIS 2004*, 2004, p. 9.

[18] Jančar, P., M. Kot and Z. Sawa, *Complexity of deciding bisimilarity between normed BPA and normed BPP*, Information and Computation, Elsevier (to appear).

[19] Jančar, P., M. Kot and Z. Sawa, *Normed BPA vs. normed BPP revisited*, in: *Proceedings of CONCUR 2008*, Lecture Notes in Computer Science **5201** (2008), pp. 434–446.

[20] Jančar, P., A. Kučera and F. Moller, *Deciding bisimilarity between BPA and BPP processes*, in: *Proceedings of CONCUR 2003*, LNCS **2761** (2003), pp. 159–173.

[21] Kao, B. and H. Garcia-Molina, *An overview of real-time database systems*, in: *Advances in Real-Time Systems*, Prentice-Hall, Inc., 1995 pp. 463–486.

[22] Kot, M., *Regularity of BPP is PSPACE-complete*, in: *Proceedings of the 3rd annual workshop WOFEX 2005* (2005), pp. 393–398.

[23] Kot, M., *Notes on modeling of real-time database system V4DB in verification tool Uppaal*, in: *MEMICS proceedings (MEMICS 2007 - Third Doctoral Workshop on Mathematical and Engineering Methods in Computer Science)* (2007), pp. 82–89.

[24] Kot, M., *Modeling real-time database concurrency control protocol two-phase-locking in Uppaal*, in: *Proceedings of the International Multiconference on Computer Science and Information Technology* (2008), pp. 673–678.

[25] Kot, M., *Modeling selected real-time database concurrency control protocols in Uppaal*, Innovations in Systems and Software Engineering **5** (2009), pp. 129–138.

[26] Kot, M. and Z. Sawa, *Bisimulation equivalence of a BPP and a finite-state system can be decided in polynomial time*, Electronic Notes in Theoretical Computer Science **138** (2005), pp. 49–60, proceedings of Infinity 2004).

[27] Król, V., "Metody ověřování vlastností real-time databázového systému s použitím jeho experimentálního modelu." Ph.D. thesis, Technical university of Ostrava (2006), in czech language.

[28] Król, V., J. Pokorný and J. Černohorský, *Symbolic model checking for real-time systems*, WSEAS Transactions on Information Science & Applications **3** (2006).

[29] Kučera, A. and R. Mayr, *Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time*, Theoretical Computer Science **270** (2002), pp. 667–700.

[30] Kučera, A. and R. Mayr, *A generic framework for checking semantic equivalences between pushdown automata and finite-state automata*, in: *IFIP TCS* (2004), pp. 395–408.

[31] Lam, K.-Y. and T.-W. Kuo, editors, "Real-Time Database Systems: Architecture and Techniques," Kluwer Academic Publishers, 2001.

[32] Lasota, S. and W. Rytter, *Faster algorithm for bisimulation equivalence of normed context-free processes*, in: *Proc. MFCS'06*, Lecture Notes in Computer Science **4162** (2006), pp. 646–657.

[33] Mayr, R., *Process rewrite systems*, Electronic Notes in Theoretical Computer Science **7** (1997), proceedings of Expressivness in Concurrency (EXPRESS'97).

[34] Mayr, R., *Process rewrite systems*, Information and Computation **156** (2000), pp. 264–286.

[35] Nyström, D., M. Nolin, A. Tesanovic, C. Norström and J. Hansson, *Pessimistic concurrency-control and versioning to support database pointers in real-time databases*, in: *Proc. of the 16th Euromicro Conference on Real-Time Systems* (2004), pp. 261–270.

[36] Paige, R. and R. E. Tarjan, *Three partition refinement algorithms*, SIAM Journal on Computing **16** (1987), pp. 973–989.

[37] Srba, J., *Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard*, in: *Proc. STACS'02*, LNCS **2285** (2002), pp. 535–546.

[38] Srba, J., *Strong bisimilarity and regularity of basic process algebra is PSPACE-hard*, in: *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, LNCS **2380** (2002), pp. 716–727.

[39] Srba, J., *Roadmap of infinite results*, in: *Current Trends In Theoretical Computer Science, The Challenge of the New Century, Volume 2: Formal Models and Semantics*, World Scientific Publishing Co., 2004 pp. 337–350, (See an updated version at http://www.brics.dk/∼srba/roadmap/).

[40] Stirling, C., *Modal and temporal logics*, Handbook of Logic in Computer Science **2** (1992), pp. 477–563.

[41] Tarski, A., *A lattice-theoretical fixpoint theorem and its applications*, Pacific Journal of Mathematics **5** (1955), pp. 285–309.

[42] van Glabbeek, R., *The linear time—branching time spectrum*, Handbook of Process Algebra (1999), pp. 3–99.

[43] Černá, I., M. Křetínský and A. Kučera, *Comparing expressibility of normed BPA and normed BPP processes*, Acta Informatica **36** (1999), pp. 233–256.