MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Quantitative Linear-Time Model Checking

PH.D. THESIS PROPOSAL

Jana Tůmová

Brno, January 2010

Advisor: prof. RNDr. Ivana Černá, CSc.

Advisor's signature: _____

Consultant: RNDr. Jiří Barnat, Ph.D.

Consultant's signature: _____

Acknowledgements

I would like to thank my advisor Ivana Černá and consultant Jiří Barnat for their helpful advice and many valuable discussions. I also thank members of ParaDiSe laboratory for their support and creative working environment.

Contents

Chapter 1

Introduction

Complex computer systems have become fundamental part of our daily lives and we rely on them in many fields. Validation of their correctness is therefore a necessary part of hardware and software design and implementation process. Probably the best known and the most widespread technique that helps us to reveal errors is *simulation and testing*. Although this conventional technique is in many cases undoubtedly the most suitable one, it does not guarantee that all bugs in a given system have been found. *Formal verification*, on the other hand, is a technique that can prove or disprove correctness of a system.This is particularly useful when it comes to safety critical systems, where subtle faults may cause loss of a huge amount of money or even human lives. Higher level of confidence in their correctness is thus desired. In such cases, formal verification methods are applied instead of, or in addition to, simulation and testing.

One of the popular fully-automatic approaches in the field of formal verification techniques is *model checking* [3], involving three basic steps. The first one is building an abstraction of a system – a *model*. Then, a desired property is transformed into formal *specification*. Finally, the third step is *verification*, whether the model satisfies the property by exhaustive examination of all its possible behaviours. If the model does not satisfy the property, model checking procedure moreover provides a *counterexample*, i.e. a behaviour of the model that violates the property.

There is a number of modeling formalisms and the choice of the appropriate one depends on the structure of a system under examination and to some extent also on the verified property. Usually, it is some kind of a labeled state transition model representing systems states and their changes. In this work we aim at finite, discrete models only, which can be viewed as graphs and easily handled in algorithms and especially their implementations. A desired property of a system is often expressed as a formula of a *temporal logic*. However, there are also different specification formalisms such as automata. We distinguish two major categories of temporal logics: *branching-time* and *linear-time*. The former one expresses properties that depend on a branching structure of a model. Its typical representative is CTL (Computational Tree Logic), whose formulas reason about the whole computational trees of a model. On contrary, formulas of linear-time logics such as LTL (Linear Temporal Logic) capture properties of single runs of a model. CTL and LTL are incomparable (some properties can be captured by CTL and not by LTL and vice versa). Model checking algorithms are based on different principles for both of them. Linear-time logics are usually more convenient for users, who tend to think easier about single runs rather than whole computational trees. Logics, such as CTL*, combine both branching-time and linear-time operators.

In this work we aim at *quantitative model checking*, where a specific quantitative attribute is incorporated in the model. Verified properties are *quantitative properties* expressing requirements on quantity of this attribute. There are several specialized areas in quantitative model checking and a number of dedicated tools have been developed. For in-

stance, real-time systems can be modelled with the use of timed automata and tools such as UPPAAL [9] or KRONOS [28] can be used to verify properties with timing constraints. MRMC [20] tool analyzes Markov reward models, etc.

A remarkable branch of quantitative model checking is quantitative probabilistic model checking. Probabilistic models, namely *Markov Chains* (MCs) and *Markov Decision Processes* (MDPs) allow us to capture uncertainities or randomization in systems, e.g. randomized protocols or systems with failure rates such as communication systems with lossy channels, etc. Informally, they are discrete state transition models, whose transitions are enhanced with probabilities under which the transitions may happen. The advantage of MDPs is that they combine probability and nondeterminism. They can be used for example for modeling asynchronous parallel composition of probabilistic systems. Widely used formalisms for specifying desired behaviour of probabilistic models include probabilistic temporal logics PLTL, PCTL and PCTL* derived from their nonprobabilistic versions LTL, CTL and CTL*, respectively. Quantitative model checking of MDPs has a long tradition and there exist a number of specialized model checking tools. PRISM [18] is the most widespread probabilistic verification tool today enabling PCTL and PLTL model checking of continuous and discrete MCs and MDPs. The tools dedicated to PLTL model checking include LiQuor [10] and our tool ProbDiVinE-MC [5] taking advantage of multi-core computation.

The probabilistic logic PCTL introduces a new probabilistic quantification operator $P_{\bowtie p}$ to the syntax of CTL. This operator can be nested and more complex quantitative properties can thus be expressed. On contrary, PLTL has the same syntax as LTL and does not adopt any quantification operator. Quantitative PLTL model checking only provides probability that an MC or an MDP satisfies an LTL (PLTL) formula. A natural task arising is to overlay this gap: to find a similar quantitative extension to LTL as in case of CTL that would furthermore capture a new and practically interesting class of properties. Although probabilistic model checking is a deeply studied topic, to our best knowledge, this problem has not been solved yet.

Recent, quite unexplored, but well motivated direction in quantitative verification is model checking of *systems with degradation*, i.e. with an inherent quality that degrades in time. This might be for instance electric charge in some electronic devices, power or quality of a transmitted signal in broadcasting network, memory consistency that degrades during allocation and deallocation of a memory block in computers etc.

We could model such systems as standard finite discrete transition systems with floating point variables keeping the exact amount of degradation and model check them with the use of tools such as SPIN [19], or DiVinE [4]. However, this approach does not allow for verification of some properties that system designers might be interested in. For instance, a property "every time A holds true, B happens before the amount of degradation measured since that moment drops below certain level", requires possibly infinite number of unbounded variables to keep the amount of degradation since the every moment when A holds true. This is beyond the possibilities of current model checkers. The same problem arises when we decide to use Markov decision processes and express degradation by means of probability. Furthermore, in MDPs, a successor function is a probabilistic distribution, which is rather restrictive and unrealistic. In [14] the authors suggest a discounted branching-time logic with possibility of giving more weight to the near future than to the far away future. However, the semantics of the logic is defined over standard transition systems, Markov chains and Markov decision processes and the degradation constant is system-wide fixed. For the same reasons as explained above, this approach is also inappropriate.

The task is not only to introduce a suitable modeling formalism for systems with degradation, but also to develop a logic rich enough to express their distinguished properties, and corresponding model checking algorithms. Pure LTL is not appropriate due to the reasons mentioned above, however, a linear-time logic with a quantification operator might be useful here. Another motivation for development of such logic is the fact that degradation and probability are not completely unrelated. In fact, probability can be somehow viewed as a degrading quality. Progress in quantitative linear-time model checking of systems with degradation could therefore potentially help us to understand the same open problem for probabilistic systems.

The goal of formal verification is to answer, whether a system meets a specification or not. Another, strongly motivated task in system design is to synthetize a *controller* (also called *control strategy*) that affects behaviour of a system to meet given requirements. Controller synthesis is a broad and widely studied topic for various types of systems and specifications. In this work, we will consider a subproblem of controller synthesis formulated as a dual to our primary research subject – model checking. Given a model and a specification (a temporal logic formula), find a control strategy such that if the model is executed according to the strategy, it meets the specification.

In case the model is a standard labeled transition system, the task of a controller is, simply put, to decide which control input (action) to choose in each state so that a desired property is satisfied regardless on which transition labeled with the chosen input is actually executed. In probabilistic settings, the specification for control of Markov decision processes can be given by means of a probabilistic temporal logic, namely PCTL or LTL. In case of PCTL, the task is to find controller, such that a given MDP satisfies a formula. In case of LTL it is to find a controller, such that a MDP meets a specification with a given lower or upper probability bound. LTL control algorithms for both nonprobabilistic and probabilistic systems are in many cases based on, or at least inspired by model checking algorithms.

Control strategy synthesis for systems with degradation is of course an open question due to the fact, that appropriate modeling and specification formalisms have not been yet designed. However, once those are developed, solution to controller synthesis problem in this settings would nicely complement model checking techniques in simplifying software and hardware system design process.

Chapter 2

State of the Art

As we indicated in the introduction, the primary aim of the thesis will be development of new techniques for quantitative linear-time verification of systems with degradation including suitable modeling and specification formalisms, and model checking algorithms. Degradation and probability are closely related topics and as we will see later, our work in progress is inspired by modeling and verification of probabilistic systems. That is why we will first introduce quantitative probabilistic model checking. In the first two sections, we will provide necessary definitions of models of probabilistic systems, logics used to express their properties, and describe model checking algorithms. Although the thesis itself will focus on linear-time verification only, we will introduce also branching-time logic PCTL to understand the concept of quantification operator. This concept might be useful in linear-time model checking of systems with degradation. The second reason for covering probabilistic system is that it is expected that we will design a linear-time logic with quantification operator to capture properties of systems with degradation. Such logic does not exist in probabilistic settings and it might be interesting to investigate its expressive power for probabilistic systems, namely Markov decision processes.

The third section in this chapter will be dedicated to our current results in verification of systems with degradation. In particular, we show a motivation example, and define modeling and specification formalisms. Then, we focus on model checking techniques and the use of the specification formalism in probabilistic settings.

Finally, the fourth section will discuss work related to the second goal of the thesis, to the development of control strategy synthesis methods for systems with degradation and linear-time properties. Here, we will focus on both discrete and probabilistic state transition systems. Controller synthesis algorithms in these areas are strongly inspired by corresponding model checking procedures. We assume that we will reach a solution to control strategy synthesis problem for systems with degradation similarly.

2.1 Preliminaries

Labeled Transition Systems

Let us first shortly introduce labeled transition systems as a basic modeling formalism for nonprobabilistic systems. The variant that we define here is sometimes called Kripke transition system as it adapts state labeling concept from Kripke structures. In general, labeled transition system are also not required to have an initial state.

A finite (nondeterministic) transition system is a tuple $\mathcal{T} = (S, Act, T, s_{init}, AP, L)$, where S and Act are finite sets of states and actions, $T : S \times Act \to 2^S$ is a (nondeterministic) transition function, $s_{init} \in S$ is an initial state, AP is a set of atomic propositions, and $L : S \to 2^{AP}$ is a labeling function; $L(s)$ is the set of atomic propositions that hold true in the

state s. A transition system is deterministic, if $|T(s,a)| \leq 1$ for all $s \in S$ and $a \in Act$.

A run of $\mathcal{T} = (S, Act, \delta, s_{init}, AP, L)$ is an is an infinite sequence of states $\tau = s_0 s_1 s_2 \ldots$, where for all $i \geq 0$ it holds that $s_i \in S$, and $s_{i+1} \in T(s_i, a)$ for some $a \in Act$. A run $\tau = s_0 s_1 s_2 \ldots$ defines a *trajectory* $L(s_0)L(s_1)L(s_2)\ldots$. The set of all trajectories generated by the set of all starting at state s_{init} is called the *language* of $\mathcal{T}$.

Markov Decision Processes

Widespread modeling formalisms for probabilistic systems include discrete Markov chains and Markov decision processes, both of them labeled transition systems enhaced with probabilities. In Markov chains, the successor relation is, simply put, given by a probability distribution. However, with the use of MCs we cannot describe behaviour of systems with nondeterministic choices, such as randomized distributed algorithms. Markov decision processes combine both nondeterminism and probability. In each state of an MDP, an action is chosen nondeterministically and then the successor state is determined according to the probability distribution associated with the action.

A finite *Markov Decision Process* (MDP, see [3]) is a tuple $\mathcal{M} = (S, Act, P, s_{init}, AP, L)$, where S is a finite, nonempty set of states, Act is a finite, nonempty set of actions, $P : S \times Act \times S \to [0,1]$ is a transition probability function, $s_{init} \in S$ is an initial state, AP is a set of atomic propositions, and $L : S \to 2^{AP}$ is a labelling function; $L(s)$ is the set of atomic propositions that are true in the state s. $Act(s)$ denotes the set of actions that are enabled in the state s, i.e. the set of actions $a \in Act$ such that $P(s,a,t) > 0$ for some state $t \in S$. For any state $s \in S$, we require that $Act(s) \neq \emptyset$ and $\forall a \in Act(s)$ it holds $\sum_{s' \in S} P(s,a,s') = 1$.

An infinite *path* in an MDP is a sequence $\pi = s_0 a_0 s_1 a_1 \ldots$ such that $P(s_i, a_i, s_{i+1}) > 0$ for each $i \geq 0$. We say π originates at s_0. A finite path is a finite prefix of an infinite path. A *trajectory* of π is the word $L(s_0)L(s_1)\ldots$ over the alphabet 2^{AP} obtained by the projection of π to the state labels. A *trace* of π in $\mathcal{M}$ is a sequence $(L(s_0), P(s_0,a_1,s_1))\ (L(s_1), P(s_1,a_2,s_2))\ldots$ over the alphabeth $2^{AP} \times [0,1]$ given by the projection of π to the state labels and the probabilities of the transitions under the actions.

The intuitive operational semantics of an MDP is as follows. If s is the current state then an action $a \in Act(s)$ is chosen nondeterministically and is executed leading to a state t with probability $P(s,a,t)$. We refer to t as an *a-successor* of s if $P(s,a,t) > 0$. State s is called *deterministic* if exactly one action is enabled in s. If all states of an MDP are deterministic, the MDP is called *Markov chain*. To resolve the nondeterminism of an MDP a *scheduler* function is used. We consider deterministic history dependent schedulers which are given by a function η assigning an action $\eta(\pi) \in Act(s_n)$ to every finite path $\pi = s_0 a_0 \ldots a_{n-1} s_n$. Given a scheduler η, the behavior of M under η can be formalized as a Markov chain.

Let M be a Markov chain, $s \in S$ be a state of M, and X be a set of paths of M originating at s. We define *the probability of the set X* as a measure of the set X in the set of all paths of M originating at s. A set X of paths of a Markov Chain M is called *basic cylinder set* if there is a prefix $s_0 a_0 \ldots a_{n-1} s_n$ such that X contains exactly all paths of M with that prefix. The probability meassure of a basic cylinder set with prefix $s_0 a_0 \ldots a_{n-1} s_n$ is then $\prod_{i=0}^{n-1} P(s_i, a_i, s_{i+1})$. If the set X of paths of M is not a basic cylinder set, its measure is determined as a sum of measures of maximal (w.r.t. inclusion) basic cylinder sets fully contained in X [12].

Linear Temporal Logic

Formal specification of linear properties is provided by Linear Temporal Logic (LTL) proposed in [23]. The syntax and semantics of LTL that we introduce here is based on the definitions in [16]. (Propositional) LTL formulae over a set of atomic propositions AP are inductivelly defined as follows:

$$\varphi ::= true \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathcal{X}\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where $a \in AP$ and $true$ is a predicate that holds true in each state of the system. Further, we define Boolean connectives disjunction $\vee$ and implication $\Rightarrow$ and derived operators $\mathcal{F}$ (eventually) and $\mathcal{G}$ (globally): $\varphi_1 \vee \varphi_2 \stackrel{def}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \Rightarrow \varphi_2 \stackrel{def}{=} \neg\varphi_1 \vee \varphi_2$, $\mathcal{F}\varphi \stackrel{def}{=} true\ \mathcal{U}\ \varphi$, and $\mathcal{G}\varphi \stackrel{def}{=} \neg\mathcal{F}\neg\varphi$.

Semantics of LTL is defined over paths of an MDP $\mathcal{M} = (S, Act, P, s_{init}, AP, L)$. Let $\pi = s_0 a_0 s_1 a_1 s_2 a_2 \ldots$ be a path in $\mathcal{M}$. π^i denotes a sufix $s_i a_i s_{i+1} a_{i+1} \ldots$ starting at s_i and $\pi(i) = s_i$. The satisfaction relation $\models$ is defined as follows:

$\pi \models true$	always
$\pi \models a$	$\iff a \in L(\pi(0))$
$\pi \models \neg\varphi$	$\iff \pi \not\models \varphi$
$\pi \models \varphi_1 \wedge \varphi_2$	$\iff \pi \models \varphi_1 \wedge \pi \models \varphi_2$
$\pi \models \mathcal{X}\varphi$	$\iff \pi^1 \models \varphi$
$\pi \models \varphi_1 \mathcal{U} \varphi_2$	$\iff \exists k : \pi^k \models \varphi_2 \wedge \forall 0 \leq j < k : \pi^j \models \varphi_1$

Intuitivelly, π satisfies a formula $\mathcal{X}\varphi$, if φ holds in the *next* state on the path, i.e. in s_1. $\varphi_1 \mathcal{U} \varphi_2$ means that φ_1 holds on the path *until* φ_2 will become true. $\mathcal{F}\varphi$ describes that φ will hold true sometimes in future (*eventually*) and $G\varphi$ that φ is true *globally*, in every state on the path π. We denote by $\mathcal{L}_\varphi$ the language of infinite words that satisfy the formula φ.

Given an MDP $\mathcal{M}$ and an LTL formula φ, the task of quantitative model checking is to compute the minimal (or maximal) probability of the set of paths satisfying φ with respect to all schedulers for $\mathcal{M}$. We simply call the probability the minimal (or maximal) probability that $\mathcal{M}$ satisfies φ.

Remark. Given an MDP and an LTL formula, the problem of *qualitative* model checking is to answer whether the minimal (or maximal) probability of the set of paths satisfying the formula is 1 (or 0, respectivelly) with respect to all schedulers for $\mathcal{M}$.

Remark. LTL can be interpreted both over nonprobabilistic and probabilistic models. The syntax and the semantics for the nonprobabilistic ones is defined over runs in analogous way as stated above. The question to be answered is whether *each* run originating at the initial state of a model satisfies a given LTL formula.

Büchi and Rabin Automata

An ω-automaton is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_{init}, F)$, where Q is a finite set of states, Σ is the input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a nondeterministic transition function, $Q_{init} \subseteq Q$ is the set of initial states, and F is the acceptance condition. $\mathcal{A}$ is deterministic iff $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. The semantics of an ω-automaton is defined over infinite input words. A run of $\mathcal{A}$ over a word $\sigma = a_1 a_2 a_3 \ldots \in \Sigma^\omega$ is a sequence $\rho = q_0 q_1 q_2 \ldots$, where $q_0 \in Q_{init}$

and $(q_{i-1}, a_i, q_i) \in \delta$ for all $i \geq 1$. Let $\inf(\rho)$ denote the set of states that appear in the run ρ infinitely often. An input word is accepted by an automaton if some run over σ is accepting. The definition of an accepting run depends on the type of the acceptance condition F of the automaton. If $\mathcal{A}$ is a Büchi automaton, then $F \subseteq Q$ and a run ρ of $\mathcal{A}$ is accepting if and only if $\inf(\rho) \cap F \neq \emptyset$. If $\mathcal{A}$ is a Rabin automaton, then $F = \{(G_1, B_1), \ldots, (G_n, B_n)\}$, where $G_i, B_i \subseteq Q$ for all $i \in \{1, \ldots, n\}$. A run ρ is accepting if $inf(\rho) \cap G_i \neq \emptyset \wedge inf(\rho) \cap B_i = \emptyset$ for some $i \in \{1, \ldots, n\}$. We denote by $\mathcal{L}_\mathcal{A}$ the language accepted by $\mathcal{A}$, i.e. the set of all words accepted by $\mathcal{A}$. An LTL formula φ can be translated into a nondeterministic Büchi automaton $\mathcal{A}$ accepting the language $\mathcal{L}_\mathcal{A} = \mathcal{L}_\varphi$ or a deterministic Rabin automaton $\mathcal{B}$ accepting the language $\mathcal{L}_\mathcal{B} = \mathcal{L}_\varphi$.

Probabilistic Computation Tree Logic

LTL formulae themselves do not contain any constraints on probabilities. In Probabilistic Computation Tree Logic (PCTL, first introduced in [17]), the probabilistic requirements are built directly in its formulae. Whereas LTL captures properties of individual paths originating at a certain state, PCTL is a branching-time logic. Similarly, as in nonprobabilistic setting, LTL and PCTL are incomparable (see [3]).

PCTL *state formulae* over the set of atomic propositions AP are defined inductivelly:

$$\phi ::= true \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathcal{P}_{\bowtie p}(\varphi)$$

where $a \in AP$, $true$ is a predicate true in each state of the system, φ is a PCTL path formula, $\bowtie \in \{<, \leq, >, \geq\}$ and $p \in [0, 1]$ is a rational bound. PCTL *path formulae* are formed according to the grammar:

$$\varphi ::= \mathcal{X}\phi \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{U}^{\leq n} \phi_2$$

where ϕ, ϕ_1 and ϕ_2 are PCTL state formulae and $n \in N$.

Intuitivelly, state formulae express properties of states and path formulae specify properties of paths. For both of them, we define satisfaction relation $\models$. Let us consider an MDP $\mathcal{M} = (S, Act, P, s_{init}, AP, L)$, a state $s \in S$, a path $\pi = s_0 a_0 s_1 a_1 s_2 a_2 \ldots$ in $\mathcal{M}$, state formulae ϕ, ϕ_1, ϕ_2 and a path formula φ.

$s \models true$	always
$s \models a$	$\iff a \in L(s)$
$s \models \neg\phi$	$\iff s \not\models \phi$
$s \models \phi_1 \wedge \phi_2$	$\iff s \models \phi_1 \wedge s \models \phi_2$
$s \models \mathcal{P}_{\bowtie p}(\varphi)$	$\iff$ for all schedulers η for $\mathcal{M} : Pr_\eta(s \models \varphi) \bowtie p$

where $Pr_\eta(s \models \varphi)$ denotes probability of the set of paths originating at s and satisfying a path formula φ under the scheduler η.

$\pi \models \varphi$	$\iff \pi(0) \models \varphi$
$\pi \models \neg\phi$	$\iff \pi \not\models \phi$
$\pi \models \phi_1 \wedge \phi_2$	$\iff \pi \models \phi_1 \wedge \pi \models \phi_2$
$\pi \models \mathcal{X}\phi$	$\iff \pi(1) \models \phi$
$\pi \models \phi_1 \mathcal{U} \phi_2$	$\iff \exists k : \pi^k \models \phi_2 \wedge \forall 0 \leq j < k : \pi^j \models \phi_1$
$\pi \models \phi_1 \mathcal{U}^{\leq n} \phi_2$	$\iff \exists 0 \leq k \leq n : \pi^k \models \phi_2 \wedge \forall 0 \leq j < k : \pi^j \models \phi_1$

Similarly as in LTL, we can define derived PCTL path operators $\mathcal{F}$ and $\mathcal{G}$. We can observe that in a formula with nested operators, the state operator $P_{\bowtie p}$ must alternate with the path operators. Thus, for example $\mathcal{F}\mathcal{G}\phi$ is not a correctly defined PCTL formula. On the other hand $P_{\leq 0.5}\,\mathcal{F}(P_{\geq 0.1}\,\mathcal{G}\phi)$ is.

Remark. A notable extension to PCTL is PCTL*. In PCTL* a path formula does not need to be immediatelly preceded by the state operator $\mathcal{P}_{\bowtie p}$. This causes that any LTL formula φ is a PCTL* formula as well.

2.2 Quantitative Probabilistic Model Checking

The principles of the automata-theoretic approach [26] to quantitative LTL model checking of MDPs are similar to the nondeterministic case (as implemented in the tool SPIN [19]). The maximal probability that an MDP $\mathcal{M}$ satisfies a formula φ is computed as follows. First, φ is transformed into a language equivalent deterministic Rabin automaton $\mathcal{A}$. Then, we build a product $\mathcal{M} \times \mathcal{A}$ of the MDP $\mathcal{M}$and the automaton $\mathcal{A}$. The result is an MDP with a Rabin condition. We identify all the so-called accepting end components [12, 13] in its underlying graph. An End Component (EC) is a strongly connected component such that for each state q in an EC E there exists an enabled action a such that *all* a-successors of q belong to E. An Accepting End Component (AEC) is an end component satisfying the accepting condition of $\mathcal{A}$. States in an AEC satisfy the examined property with the probability one. After computation of AECs, the graph is transformed into a linear programming problem (a set of inequalities over states of the probabilistic system and an objective function to be minimized) [1]. The solution of the linear programming problem gives us maximal probability of reaching an AEC from each state of the product. By projecting the initial states of the product into the states of $\mathcal{M}$, we obtain the desired result.

The minimal probability that $\mathcal{M}$ satisfies φ can be computed via computation of the maximal probability, that $\mathcal{M}$ satisfies negation of φ.

To our best knowledge, the above explained scheme is the only approach to quantitative probabilistic LTL model checking that has been implemented, namely in the tools LiQuor, PROBDIVINE-MC, and PRISM.

Model checking for probabilistic computation tree logic gives a yes/no answer to the question whether a state s in a given MDP satisfies a state formula. We do not describe the verification techniques in more details, as they are not relevant to the thesis topic.

2.3 Quantitative Model Checking of Systems with Degradation

In this section we present our recent results in quantitative model checking of systems with degradation published in [7]. The author of this thesis proposal provided major contributions both to the theoretical research and the text of the paper. So far, we designed a modeling formalism for systems with degradation, which we call a Transition System with Degradation (TSD). Further, we developed an automata-like specification formalism called Büchi automata with Degradation Constants (BADCs). These formalisms are described here in detail as a result closely related to our future research. The next, currently investigated goal is development of a logic with a quantification operator and its translation to BADCs. In [7] we also suggested a model checking method allowing for model checking of transition systems

with degradation against properties expressed with the use of BADCs. Finally, we aimed at the use of BADCs in probabilistic settings.

Modeling Systems with Degradation

Let us start with a motivation example (taken from [7]), which demonstrate the need of quantitative verification of systems with degradation and also a natural way how to model them.

Motivation Example: Signal Coverage Problem

Let us suppose, we want to get some signal from a start point S to an end point E. Unfortunately, the points are too far from each other, so the signal cannot reach the destination without unrepairable signal degradation. A possible solution to the problem is to build relays in between S and E that restore the quality of the signal while the signal is still fully reconstructible. Furthermore, let us assume we have a map of possible places where a relay may be built including pairwise signal degradation values as illustrated in Figure 2.1. For the sake of simplicity, let us assume the signal goes through these places. Using a system with degradation, we can easily check, whether the signal reaches the target point in proper shape if the relays are built at the A-points. Another example of a degradation property might be to check whether some of the A-points are redundant.

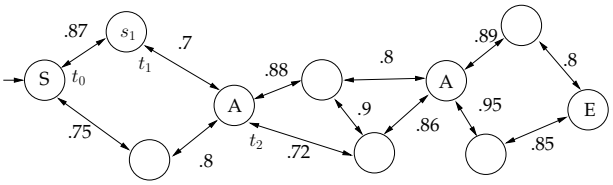

Figure 2.1: Signal coverage map.

Transition Systems with Degradation

Informally, systems with degradation are systems that involve an attribute whose quality degrades (e.g. the signal in the example). We formalize such systems as *Transition Systems with Degradation* (TSDs). Unlike the standard transition systems, every transition is associated with a degradation constant in a TSD. A degradation constant is a rational number from interval $(0, 1]$. The constants may differ for individual transitions in the system. Note that the formal definition of a TSD contains no specification of the attribute that degrades, it only captures how much it degrades along each transition.

A *transition system with degradation* is a tuple $\mathcal{M} = (S, Act, \rightarrow, S_{init}, AP, L)$, where S is a finite, nonempty set of states, Act is a finite, nonempty set of actions, $\rightarrow \subseteq S \times Act \times (0, 1] \times S$ is a transition relation, $S_{init} \subseteq S$ is a set of initial states, AP is a set of atomic propositions, and $L : S \rightarrow 2^{AP}$ is a labeling function; $L(s)$ denotes the set of atomic propositions that are true in state s. Instead of $(s_1, a, d, s_2) \in \rightarrow$ we write $s_1 \xrightarrow{a,d} s_2$. A transition $s_1 \xrightarrow{a,d} s_2$ represents that the model can move from state s_1 to the state s_2 by a (nondeterministic) choice of action a. The degradation constant d associated with the transition gives the fraction to which the

quality is degraded if the transition is executed. If the level of degradation at state s_1 is let us say l and the action is executed, the level of degradation at state s_2 will be $l \cdot d$.

A *path* in a TSD $\mathcal{M} = (S, Act, T, S_{init}, AP, L)$ is an infinite sequence $\pi = s_0 t_0 s_1 t_1 \ldots$ where $s_i \in S$ and $t_i = (s_i, a_i, d_i, s_{i+1}) \in T$ for all $i \geq 0$. A *trajectory* corresponding to the path $\pi = s_0 t_0 s_1 t_1 \ldots$ is given by the projection of π to the state labels, $trajectory(\pi) = L(s_0)L(s_1)\ldots$. A *trace* corresponding to the path $\pi = s_0 t_0 s_1 t_1 \ldots$ is given by the projection of π to the state labels and degradation rates, $trace(\pi) = (L(s_0), d_0)\ (L(s_1), d_1)\ \ldots$.

For instance, let us consider the example in Figure 2.1 and its path $St_0 s_1 t_1 A t_2 \ldots$ with the trace $(S, 0.87), (s_1, 0.7), (A, 0.72), \ldots$. The signal degradation between S and A is $0.87 \cdot 0.7 = 0.609$. This means the quality of the signal in A will be 60.9% of the quality in S.

Specification

To express the quantitative linear properties of systems with degradation we introduce a modification of Büchi automata, the so called *Büchi Automata with Degradation Constraints* (BADC). The standard automata are enriched with a set of bounded variables allowing us to express the amount of degradation.

Let D be a finite set of *degradation variables* ranging over the rational numbers in between $(0, 1]$. A *degradation constraint over* D is of form $\varphi ::= x \bowtie d \mid \varphi \wedge \varphi$, where $\bowtie \in \{<, \leq, >, \geq\}$, $x \in D$, and d is a rational number in $(0, 1]$. Note that degradation constraints exclude disjunction as it can be expressed using two different transitions of a BADC. $DC(D)$ denotes the set of degradation constraints over D. A *degradation valuation* is a function $\nu : D \to (0, 1]$. The set of all possible degradation valuations is $Eval(D)$.

A *Büchi Automaton with Degradation Constraints (BADC)* is a tuple $\mathcal{A} = (Q, \Sigma, D, T, q_{init}, F)$, where Q is a finite nonempty set of states, Σ is a finite alphabet,D is a finite set of degradation variables, $T \subseteq Q \times \Sigma \times DC(D) \times 2^D \times Q$ is a set of transitions, $q_{init} \in Q$ is an initial state, and $F \subseteq Q$ is a finite set of states (Büchi accepting condition).

A 5-tuple $t = (q, \alpha, \varphi, R, q') \in T$ represents the transition from state q to q' labeled with α that is enabled if constraint φ is satisfied. R is a set of degradation variables which are reset to 1 when executing the transition. For the transition $t = (q, \alpha, \varphi, R, q')$ we denote $label(t) = \alpha$, $constraint(t) = \varphi$ and $reset(t) = R$.

The semantics of a BADC $\mathcal{A} = (Q, \Sigma, D, T, q_{init}, F)$ is given by an infinite labeled transition system $\mathcal{M}_\mathcal{A} = (S, \Sigma', \to, S_{init})$, where $S = Q \times Eval(D)$,$\Sigma' = \Sigma \times (0, 1]$, $\to \subseteq S \times \Sigma' \times S$, $S_{init} = \{(q_{init}, \nu_{init}) \mid \nu_{init}(x) = 1 \text{ for all } x \in D\}$, and $(q_1, \nu_1) \xrightarrow{\alpha, d} (q_2, \nu_2)$ whenever there is a transition $(q_1, \alpha, \varphi, R, q_2) \in T$ such that

- $\nu_1 \models \varphi$
- $\nu_2(x) = \begin{cases} d, & \text{if } x \in R \\ \nu_1(x) \cdot d & \text{otherwise} \end{cases}$

A run for a word $\sigma = (\alpha_0, d_0)(\alpha_1, d_1)\ldots \in (\Sigma \times (0, 1])^\omega$ is an infinite sequence $\rho = (q_0, \nu_0)(q_1, \nu_1)\ldots$ such that $(q_0, \nu_0) \in S_{init}$ and $(q_i, \nu_i) \xrightarrow{\alpha_i, d_i} (q_{i+1}, \nu_{i+1})$ for all $i \geq 0$. A run $\rho = (q_0, \nu_0)(q_1, \nu_1)\ldots$ is accepting if $q_i \in F$ for infinitely many indices i. $\mathcal{L}_\omega(\mathcal{A}) = \{\sigma \in (\Sigma \times (0, 1])^\omega \mid \text{there exists an accepting run for } \sigma \text{ in } \mathcal{A}\}$.

Figure 2.2 the "redundant A-point" quantitative linear property for the signal coverage example.

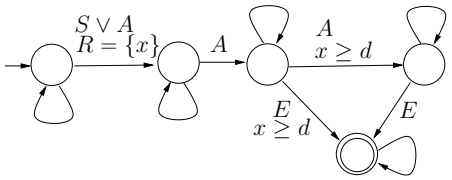

Figure 2.2: Quantitative propertiey of Sender/Receiver example.

Model Checking Algorithm

In standard model checking, a given formula is first negated and instead of verifying, whether all behaviours of a model satisfy the formula, the model checking algorithms try to reveal runs, that violates the formula. Here, we approach the model checking problem similarly. We are given a TSD model of a system with degradation and a BADC automaton specifying prohibited quantitative linear behaviors. In [7] we developed an algorithm deciding whether a given TSD model exhibits the forbidden behavior. Our model checking algorithm follows the automata-based approach to LTL model checking. First, we defined a product automaton and proved that this automaton accepts exactly the the language of the TSD traces accepted by the BADC. Next, we demonstrated that checking language nonemptiness of the product automaton is equivalent to finding an accepting cycle in the product automaton graph. This can be tested effectively by a number of known techniques like the Nested Depth First Search [11] or OWCTY [15].

The main obstacle in the verification process is that the mentioned product automaton graph may be infinite. The key observation allowing for model checking of BADC properties of systems with degradation is that for a special type of BADC automata, the so called *normalized* BADC, it is guaranteed that the product graph is finite. Intuitivelly, in normalized BADCs each cycle contains either an upper bound or a reset for each degradation variable. We defined normalized BADCs and proved that the product automaton of a TSD and a normalized BADC is finite. We also provided an algorithm which transforms any BADC to an equivalent normalized BADC. The author proved correctness of the transformation in [8].

Quantitative Linear Properties of Markov Decision Processes

Part of our work raises the question about the parallel between the systems with degradation and the Markov decision processes as well as about the relationship between probabilistic logic PLTL, PCTL, PCTL* and the quantitative linear properties formalized via BADCs. It is easy to see that an MDP is just a special case of a system with degradation. Probability can be viewed as degradation and paths and traces of an MDP as paths and traces of a Büchi automaton with degradation. With the use of BADCs, we can specify properties of traces of MDPs. We demonstrated, that BADC constraints can distinguish otherwise indistinguishable MDPs.

Using BADCs for expressing properties of MDPs brings us a new possibility to check for the presence of a specific path with a certain probability contribution. For further details and an illustrative example see [7].

2.4 Control Strategy Synthesis for Discrete and Probabilistic Systems

In this section we overview solutions to controller synthesis problem for finite, discrete and probabilistic systems and the requirement specification expressed as an LTL formula.

Discrete systems

Given a labeled transition system $\mathcal{T} = (S, Act, T, s_{init}, AP, L)$ and an LTL formula φ, the control problem is to find a (history dependent) function $\eta : S^+ \to Act$, such that each run $s_0 s_1 s_2 \ldots$, where $s_{i+1} \in T(s_i, \eta(s_0 \ldots s_i))$ satisfies the formula φ.

Remark. A function η might be also history independent, i.e. the problem is then to find $\eta : S \to \Sigma$, such that each path $s_0 s_1 s_2 \ldots$, where $s_{i+1} \in T(s_i, \eta(s_i))$ satisfies the formula φ. Obviously, each history independent function can be easily transformed into a history dependent one.

The problem for deterministic labeled transition systems is relatively easy and can be solved by adapting standard algorithms and tools from LTL model checking. A solution is proposed in [22]. The authors first translate the examined formula φ into a language equivalent (nondeterministic) Büchi automaton $\mathcal{A}$. Then, they build a product of the given deterministic transition system $\mathcal{T}$ and $\mathcal{A}$. The accepting runs of the product correspond exactly the trajectories of $\mathcal{T}$ that satisfy formula φ. The product is then analyzed and the shortest paths to strongly connected components containing an accepting state are found. Finally, these paths are projected into trajectories of $\mathcal{T}$ and an action that need to be chosen in each state of $\mathcal{T}$ is determined.

Nondeterministic systems, on the other hand, cannot be handled the same way. In [21] the problem is solved only for a fragment of LTL accepted by deterministic Büchi automata. The suggested solution is inspired from infinite LTL games, which are played by two players on a graph [24]. The authors treat nondeterminism as an adversary. A given LTL formula φ is first translated into a language equivalent deterministic Büchi automaton $\mathcal{A}$. Then, a product of transition system $\mathcal{T}$ and $\mathcal{A}$ is built, viewed as a 2 player game and a winning strategy is synthetized. Finally, the strategy is projected into $\mathcal{T}$ and the controller is presented in the form of a feedback automaton that reads a current state of $\mathcal{T}$ and generates an a control symbol (i.e. action of $\mathcal{T}$) to be applied. The same problem for full LTL has been addressed in [25] and has not yet been published.

Probabilistic systems

The problem for Markov decision processes is defined as above, except for that the satisfaction of an LTL formula is required to meet a lower or upper probability bound. Given an MDP $\mathcal{M} = (S, Act, P, s_{init}, AP, L)$ and an LTL formula φ, the control problem is to find a (history dependent) function $\eta : S^+ \to \Sigma$, such that each path $s_0 a_0 s_1 a_1 \ldots$, where $a_i = \eta(s_0 \ldots s_i))$ satisfies φ with at least (or at most) given probability under all schedulers.

This problem is solved in [2]. The authors are again inspired by automata-theoretic approach to model checking. First, a product automaton of $\mathcal{M}$ and a deterministic Rabin automaton $\mathcal{A}$ for a given formula φ is built. Then, a variant of accepting end components, called winning components, are identified. The solution is provided by maximizing the probability of reaching a winning component. Then the strategy found for the product is projected into MDP $\mathcal{M}$.

Chapter 3

Aim of the Work

3.1 Objectives and Expected Results

The aim of the PhD thesis is to contribute to the design process of correctly working systems that incorporate a quality degrading in time by development of appropriate quantitative linear-time model checking and contoller synthesis techniques. Part of the work is also study on usability of the newly designed methods in probabilistic settings. The particular goals are as follows.

- Design of a suitable modeling formalism for systems with degradation and design of a new linear-time logic with quantification operator allowing for specification of practically interesting properties of such systems.
- Development of effective procedures for quantitative linear-time model checking of systems with degradation.
- Prototype implementation and preliminary experimental evaluation and case studies showing practical application of the whole technique. The implementation will build on DIVINE framework.
- Investigation on usability of the proposed logic for probabilistic systems and study of its relation to probabilistic logics LTL and PCTL.
- Development of control strategy synthesis methods for systems with degradation and a specification given as a formula of the newly designed logic. The goal of this item is to complement the developed model checking techniques and to provide more complete framework to improve faultlessness of design of systems with degradation.
- Prototype implementation and preliminary experimental evaluation of the controller synthesis algorithms. The implementation will build on DIVINE framework.

3.2 Expected Outputs

- The text of the thesis presenting especially theoretical results.
- Publicly available prototype tool implementing model checking and control synthesis algorithms.
- Web page containing the tool, the results of its evaluation and a set of examples.
- Reviewed publications on relevant international forums.

3.3 Progression Schedule

The plan of my future study and research activities, besides attending courses and teaching, is as follows.

- Internship at Boston University — now – Feb 2010
- Doctoral exam and defence of this thesis proposal — May 2010
- Development of quantitative linear-time logic for expressing properties of systems with degradation and research on its translation to automata-like formalism, usability in probabilistic settings and relation to probabilistic logics — now – Feb 2011
- Implementing the designed techniques — Sep 2010 – Feb 2011
- A prototype version of the model checking tool — Feb 2011
- Development of controller synthesis methods — Feb 2011 – Feb 2012
- Implementing the controller synthesis tool — Sep 2011 – Feb 2012
- A prototype version of the model checking and controller synthesis tool — Feb 2012
- Work on the text of the PhD thesis — Feb 2012 – Sep 2012
- Final version of the thesis — Sep 2012

Bibliography

[1] C. Baier, F. Ciesinski, and M. Grösser. Probmela and verification of markov decision processes. *SIGMETRICS Perform. Eval. Rev.*, 32(4):22–27, 2005.

[2] C. Baier, M. Grösser, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems. In *In Proceedings of IFIP TCSâ2004*. Kluwer, 2004.

[3] C. Baier and J. P. Katoen. *Principles of Model Checking - The MIT Press*.

[4] J. Barnat, L. Brim, and P. Ročkai. DiVinE Multi-Core – A Parallel LTL Model-Checker. In *Automated Technology for Verification and Analysis*, volume 5311 of *LNCS*, pages 234–239. Springer, 2008.

[5] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. Probdivine-mc: Multi-core ltl model checker for probabilistic systems. In *QEST '08: Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 77–78, Washington, DC, USA, 2008. IEEE Computer Society.

[6] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. Local Quantitative LTL Model Checking. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 53–68. Springer-Verlag, 2009.

[7] J. Barnat, I. Černá, and J. Tůmová. Quantitative Model Checking of Systems with Degradation. In *Proceeding of the Sixth International Conference on Quantitative Evaluation of Systems (QEST 2009)*, pages 21–30. IEEE, 2009.

[8] J. Barnat, I. Černá, and J. Tůmová. Quantitative Model Checking of Systems with Degradation (Full Paper). Technical report, Faculty of Informatics Masaryk University FIMU-RS-2009-04, 2009.

[9] G. Behrmann, A. David, K. G. Larsen, O. Möller, P. Pettersson, and W. Yi. UPPAAL - present and future. In *Proc. of 40*th *IEEE Conference on Decision and Control*. IEEE Computer Society Press, 2001.

[10] F. Ciesinski and C. Baier. LiQuor: A tool for Qualitative and Quantitative Linear Time analysis of Reactive Systems. In *Proc. of QEST'06*, pages 131–132. IEEE Computer Society, 2006.

[11] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1(2/3):275–288, 1992.

[12] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[13] L. de Alfaro. *Formal Verification of Stochastic Systems*. PhD thesis, Stanford University, Department of Computer Science, 1997.

[14] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. Model checking discounted temporal properties. *Theor. Comput. Sci.*, 345(1):139–170, 2005.

[15] K. Fisler, R. Fraer, G. Kamhi, M. Y. Vardi, and Zijiang. Is there a best symbolic cycle-detection algorithm. In *In Proc. Tools and Algorithms for Construction and Analysis of Systems, volume 2031 of LNCS*, pages 420–434. Springer, 2001.

[16] O. Grumberg, E. M. Clarke, and D. Peled. *Model Checking*. The MIT Press, 1999.

[17] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:102–111, 1994.

[18] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. of TACAS'06*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[19] G. J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[20] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Quantitative Evaluation of Systems (QEST)*, pages 243–244. IEEE Computer Society, 2005.

[21] M. Kloetzer and C. Belta. Dealing with nondeterminism in symbolic control. In *HSCC*, pages 287–300, 2008.

[22] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[23] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[24] W. Thomas. Infinite games and verification. In *Proceedings of the International Conference on Computer Aided Verification*, pages 58–64, 2002.

[25] J. Tůmová, B. Yordanov, C. Belta, J. Barnat, and I. Černá. A symbolic approach to controlling piecewise affine systems. *Submitted*.

[26] M. Vardi and P. Wolper. Reasoning about infinite computation paths. *Proceedings of 24th IEEE Symposium on Foundation of Computer Science, Tuscan*, pages 185–194, 1983.

[27] B. Yordanov, J. Tůmová, C. Belta, J. Barnat, and I. Černá. Formal analysis of piecewise affine systems through formula-guided refinement. *Submitted*.

[28] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.

Appendix A

Summary of the Study

Courses

List of courses and seminars:

- Informatics Seminar (Spring 2009)
- Informatics Colloquium (Spring 2009)
- Laboratory of Parallel and Distributed Systems (Spring 2009)
- Supercomputer Architecture and Intensive Computations (Spring 2009)
- Formal Methods in Systems Biology (Spring 2009)

Internships

- Boston University (September 2009 - February 2010)

Research and Publications

As a member of Laboratory of Parallel and Distributed Systems, I have participated in the project aimed at research and development of automatic verification techniques supported by grants no. GA 201/09/1389 and no. AV 1ET408050503. I have cooperated on design of effective parallel algorithms for LTL model checking of probabilistic systems [6] and their implementation in the tool PROBDIVINE-MC [5], which builds on the model checking framework DIVINE. We also achieved first results in verification of systems with degradation [7]. We developed a suitable model, automata-like specification formalism, and the corresponding model checking algorithm. In this paper, we also showed, that the developed specification formalism can capture properties of probabilistic systems that cannot be expressed by any PLTL, PCTL or PCTL* formula. The full version of the paper including detailed proofs of correctness is also available as technical report [8].

In addition to my research activities at the ParaDiSe Laboratory, I have also visited HyNeSs laboratory at Boston University since September 2009 till February 2010. During my stay we have aimed at controller synthesis problem for discrete and probabilistic transition systems. At the time of this proposal's writing, our results have been submitted as two conference papers.

My publications and presentations achieved during my PhD studies and related to the topic of my PhD thesis are listed below, numbered according to bibliography.

Publications

[6] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. Local Quantitative LTL Model Checking. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 53–68. Springer-Verlag, 2009.

[7] J. Barnat, I. Černá, and J. Tůmová. *Quantitative Model Checking of Systems with Degradation.* In *Proceeding of the Sixth International Conference on the Quantitative Evaluation of Systems (QEST 2009),* pages 21-30. IEEE, 2009.

[25] J. Tůmová, B. Yordanov, C. Belta, J. Barnat, and I. Černá. A Symbolic Approach to Controlling Piecewise Affine Systems. *Submitted.*

[27] J. Tůmová, B. Yordanov, C. Belta, J. Barnat, and I. Černá. Formal Analysis of Piecewise Affine Systems through Formula-Guided Refinement *Submitted.*

Technical Reports

[8] J. Barnat, I. Černá, and j. Tůmová. Quantitative Model Checking of Systems with Degradation (Full Paper). Technical Report FIMU-RS-2009-04, Faculty of Informatics, Masaryk University, June 2009.

Presentations

- Presentation at HyNeSs seminar, Boston University, Fall 2009.
- Presentation at Seminar of Laboratory of Parallel and Distributed Systems, Spring 2009.

Other Activities

In addition to my research activities, I have assisted in teaching of the following courses.

- IB005 Formal Languages and Automata (Spring 2009)
- IB108 Algorithm Design II (Spring 2009)

Appendix B

Summary/Souhrn

Model checking is an advanced technique that help us to guarantee that a system meets given requirements. In general, it includes three steps: building a model of the system, formalizing the requirements, and finally examining all possible behaviors of the model to verify whether the model satisfies the requirements. In many cases, quantitative properties are an inseparable part of the system specification. The proposed PhD thesis will aim at quantitative model checking of systems with degradation, i.e. with an inherent quality that degrades in time. Currently, to our best knowledge, no appropriate formalisms to model such systems and specify their properties have been developed. Our goal is to design those, develop model checking algorithms and implement the whole solution in a publicly available prototype tool. A part of the work is also to investigate on usability of the designed techniques in probabilistic settings. To extend the usability of the techniques and the tool in system design process, we will study also problem of synthesis of a control strategy. Such strategy affects a given model of a system with degradation to satisfy a desired quantitative property.

Ověřování modelu je pokročilá technika, která nám pomáhá zaručit, že systém splňuje dané požadavky. Obecně jsou jejím základem tři kroky: vytvoření modelu systému, formalní vyjádření požadavků a ověření, zda model splňuje dané požadavky prozkoumáním všech možných chování modelu. V mnoha případech jsou nedílnou součastí specifikace systému i kvantitativní vlastnosti. Disertační práce bude zaměřena na kvantitativní ověřovaní modelu systémů s degradací, tj. s vnitřní vlastností systému, která v čase degraduje. Pokud je nám známo, doposud nebyl vyvinut vhodný formalismus k modelování takových systémů ani ke specifikaci jejich vlastností. Naším cílem je navrhnout tyto formalismy, vyvinout algoritmy ověřovaní modelu a implementovat celé řešení ve veřejně dostupném prototypovém nástroji. Součástí práce bude i studium použití navržených technik v pravděpodobnostním kontextu. Abychom rozšířili použitelnost výše zmíňených technik a nástroje v oblasti návrhu systémů, budeme studovat také problém syntézy řidící strategie. Taková strategie ovlivňuje daný model systému s degradací tak, aby splňoval požadovanou kvantitativní vlastnost.

Appendix C

Publications

Quantitative Model Checking of Systems with Degradation