MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Specification and monitoring of oscillation properties in dynamical systems

MASTER'S THESIS

Petr Dluhoš

Brno, 2012

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: RNDr. David Šafránek, Ph.D.

Acknowledgement

I would like to express my gratitude to Dr. David Šafránek, my supervisor, for his patience, valuable advice and significant amount of time dedicated to our frequent consultations.

Abstract

Formal techiques for analysis and verification of systems are increasingly used in natural and systems sciences. Application of this approach in the new areas requires new methods. This thesis focuses on reasoning about complex biological signals. Existing approaches based on temporal logics are reviewed and a new temporal logic extending the Signal Temporal Logic is introduced. A polynomial monitoring algorithm for piecewise linear signals is introduced and implemented in MATLAB with the Multi Parametric Toolbox.

Keywords

temporal logic, signals, monitoring, formal specification, oscillations, system analysis, parameter estimation, STL

Contents

List of acronyms

CTL	Computational Tree Logic
CTL*	An extension of CTL
LTL	Linear Temporal Logic (=PTL)
MITL	Metric Interval Temporal Logic
MITL$_{[a,b]}$	A fragment of MITL
MPT	Multi-Parametric Toolbox
ODE	Ordinary Differential Equation
ODEs	Ordinary Differential Equations
PTL	Propositional Temporal Logic = (LTL)
STL	Signal Temporal Logic
STL*	An extension of STL, defined in this thesis
TPTL	Timed Propositional Temporal Logic

Chapter 1

Introduction

Nature is incredibly complex. The more we know about it, the more complicated it seems to be. Recent development of computers made it possible not only to acquire, store and process gigantic amount of data, but also to make models of natural processes and study them *in silico*. The rapid development of fields like systems biology, computational biology or synthetic biology creates a demand for tools for automatic performing and evaluation of such experiments.

One of these tools is temporal logic, formalism for description and reasoning about phenomena taking place in time. It originated in philosophy and found its use in computer science for formal description and verification of technical processes. Nowadays, these techniques are increasingly used for formal analysis of natural processes. The usage in this field places new demands on temporal logics.

This thesis deals with a class of temporal logics suited for reasoning about nontrivial real-time processes. As a model of these processes, various types of oscillations (understood in broader sense) are considered, because oscillatory behaviour plays a crucial role in nature.

The thesis is divided into seven chapters. The first chapter is introductory. The main theme of the second chapter are oscillations – why do they interest us and how they can be comprehended and modelled with help of the theory of ordinary differential equations. Next chapter is focused on existing temporal logics for real-time and real-valued signals. In the fourth chapter, a new extension of Signal Temporal Logic is proposed together with an algorithm for its monitoring. The fifth chapter describes a prototype implementation of the newly created logic and

evaluates its performance. The usage of the introduced logic is demonstrated in a case study in the sixth chapter. The last chapter concludes the thesis and discusses the contribution and directions of future development.

Chapter 2

Oscillations and their importance

In mathematics, an oscillation is behaviour of an infinite sequence of numbers which does not converge nor diverge. Another way how to describe an oscillation is as a process in which a displacement from a position of an equilibrium occurs repeatedly over time. There are various definitions of oscillatory behaviour. It reflects the fact that oscillations are a very common phenomenon in various areas of natural sciences and nature itself.

While in the technical fields like physics, mechanics or electronics, the oscillations are of the periodic character – same or very similar event repeats regularly in the same time interval, in biological, social or economical systems, oscillations have more loose character. Their frequency, amplitude or development are often variable in time. However, even in this case, the oscillatory character of the phenomenon plays an important role and predetermines its course. In this thesis, oscillations are understood in a broad sense as a phenomenon of similar behaviour repeated over time.

As an example of systems with oscillation having the key role, we can think about models of several populations of animals with mutual interactions (foundations based independently by Lotka [24] and Volterra [30] in a model of predator and prey in the 1920s). In much smaller scales, tenths of oscillatory processes are taking place in animals, plants and bacteria, e.g., circadian cycles (one day period), ovarian cycles (28 days), hormonal activity, cardiac cycle, neural spikes, cell cycle [20].

In the area of modelling physical processes we encounter oscillations in the whole scale of models from the simplest ones like a bouncing ball to much more complicated ones like in [23] (model of the transition to turbulence in the form of the equation of oscillations of a pendulum with a randomly vibrating suspension axis).

Another example of a process showing importance of oscillations are metabolic processes in an unicellular, nitrogen fixing cyanobacterium *Cyanothece* sp. [6]. Authors show that periodicity in respiratory activity of this bacterium is controlled by its circadian cycle rather then actual level of irradiation.

In [21], Kholodenko shows that a well known mitogen-activated protein kinase (MAPK) cascade which conveys extracellular signals to other parts of a cell can under some conditions exhibit sustained oscillations.

This was a short list of examples showing that scientists encounter oscillations in various forms in many fields and disciplines. To study these processes, it is necessary to understand the substance of oscillatory behaviour and to have the ability to model and examine such phenomena. The former will be discussed in Section 2.1 and the letter in Chapter 3.

2.1 Characterization of oscillations

A very powerful tool for describing, modelling and analysis of the behaviour of biological systems is the theory of ordinal differential equations. This theory enables us to abstract the key properties from a real-world system and analyze them by means of mathematics. However, the analysis and interpretation of more complicated models becomes more difficult and can be hardly automatized. That is the time when usage of logics presented in this thesis becomes suitable.

In this section, there will be described brief extracts of the theory of ordinary differential equations. The purpose is to show where the oscillatory behaviour originate in, what is the most common means for its

modelling and where does the continuous signals (most common output of real-world systems and model for temporal logic presented in Chapter 4) came from. To get the picture of relations between biological systems and their abstractions see [11], for more information about mathematical foundations of the theory see [16] or [18].

Definition 2.1. *Ordinary differential equation (ODE)* is a vector equation:

$$\frac{dx}{dt} = f(x(t), t), \tag{2.1}$$

where f contains ordinary derivatives of the unknown real function x. Both functions f and x depend on an independent variable $t \in \mathbb{R}$ (time).

To extract the fundamental characteristics of oscillatory behaviour, we will deal with simpler systems where the function f in (2.1) does not directly depend on time t. These systems are called *autonomous*.

Definition 2.2. *Autonomous differential equation* is an ODE of the form:

$$\frac{dx}{dt} = f(x(t)), \tag{2.2}$$

where the function f is defined on some subspace $\Omega \subseteq \mathbb{R}^n$.

Definition 2.3. A solution $x = \varphi(t)$ of the autonomous system (2.2) with the initial condition $x(0) = x_0$ interpreted as a curve in the space Ω is called a *trajectory*.

Trajectories can be visualized in plane as *phase portraits*. This visualization can be supplemented by vector fields depicting the direction and speed of change of trajectories (Figure 2.1).

Definition 2.4. Point $x_0 \in \Omega$ is called *steady state* or *equilibrium* of the system (2.2) if $f(x_0) = 0$.

Steady states are the constant solutions of the autonomous system (2.2), i.e., if the system is located in the point x_0 at some time t_0 (i.e., $x(t_0) = x_0$), it remains there for all times $t > t_0$. Steady states play an important role in understanding of dynamics of systems described by ODEs. This role will be discussed in next sections.

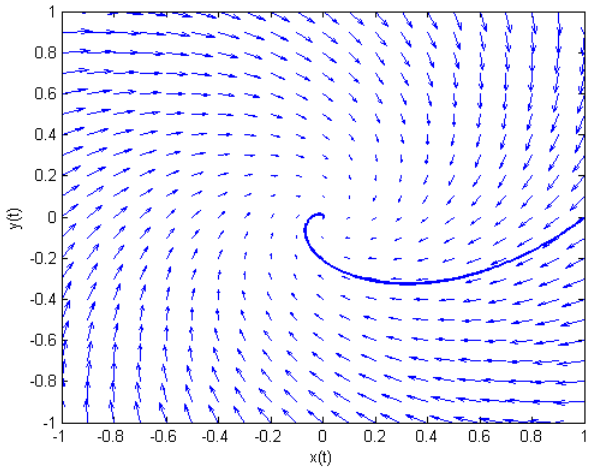


Figure 2.1: Phase portrait depicting a trajectory and a vector field of a system of two linear equations.

2.1.1 Linear systems in plane

There are various kinds of steady states and they have diverse influence on the behaviour of a system in their neighbourhood. We can demonstrate the basic types of steady states on very simple autonomous systems, the linear systems in plane, and then generalize this knowledge to the more complicated ones.

Linear system in plane is a system of two differential equations:

$$\begin{aligned} \dot{x} &= f(x,y) \\ \dot{y} &= g(x,y) \end{aligned}, \tag{2.3}$$

where the functions f, g are linear in both arguments. The system (2.3) can be written in the vector form:

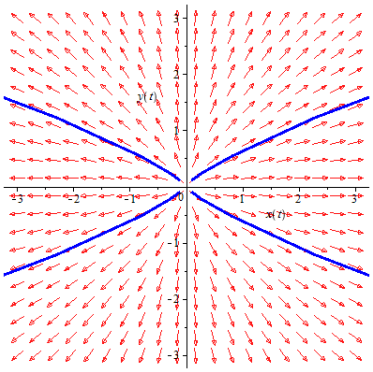
$$\dot{x} = Ax, \tag{2.4}$$

where

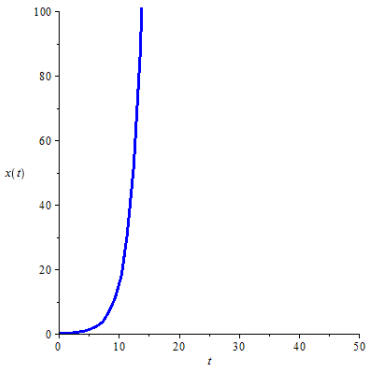
$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, det(A) \neq 0. \tag{2.5}$$

Let λ_1, λ_2 be the eigenvalues of the matrix A. Then the origin $(x, y) = (0, 0)$ is a steady state and we can distinguish the cases depicted in Figures 2.2 – 2.7. [16]

From the knowledge of steady states of a system we can derive its qualitative behaviour without the need of explicitly solving it. Similar rules apply for the systems in higher dimensions and the properties of steady states are extensions and combinations of those in plane.

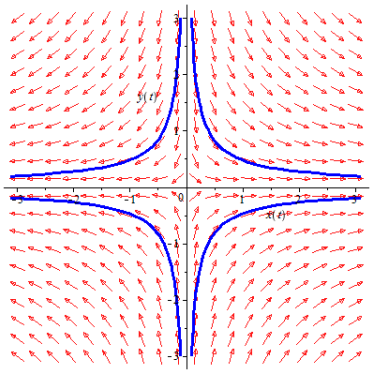


(a) Phase portrait.

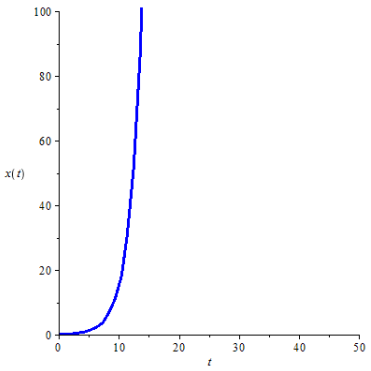


(b) A typical solution $x(t)$.

Figure 2.2: $\lambda_1, \lambda_2 < 0 \Rightarrow$ origin is a *stable node* or a *sink*.

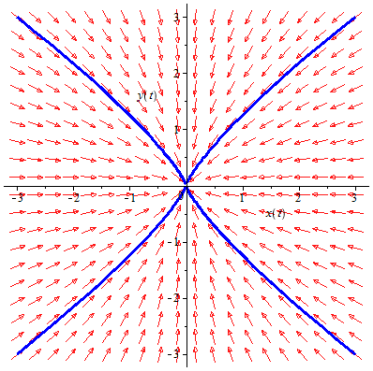


(a) Phase portrait.

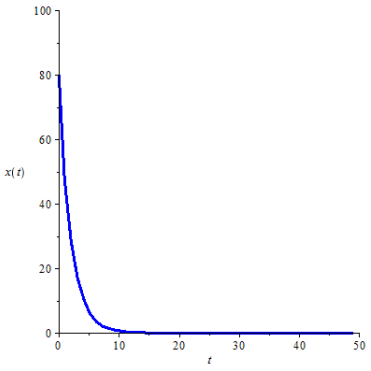


(b) A typical solution $x(t)$.

Figure 2.3: $\lambda_1 > 0, \lambda_2 < 0 \Rightarrow$ origin is an *unstable node* or a *source*.

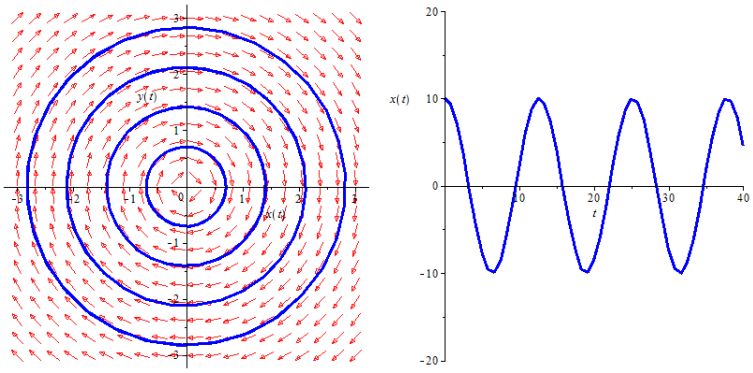


(a) Phase portrait.



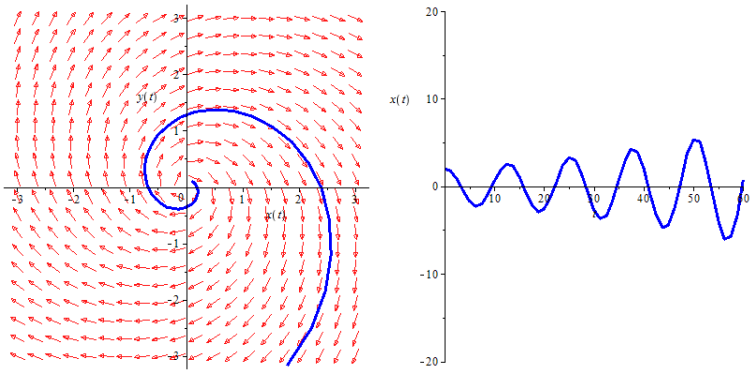
(b) A typical solution $x(t)$.

Figure 2.4: $\lambda_1, \lambda_2 > 0 \Rightarrow$ origin is a *saddle*.



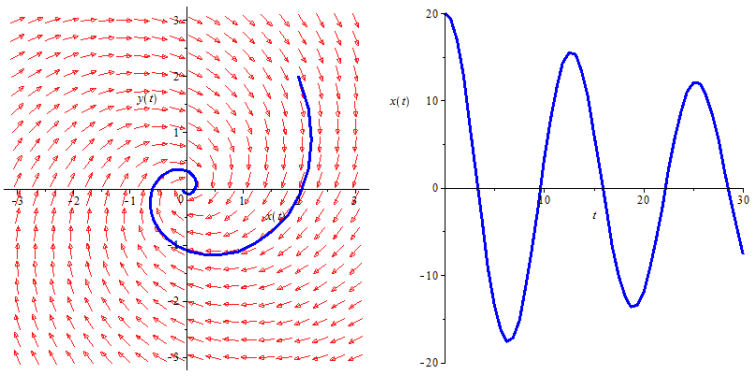
(a) Phase portrait. (b) A typical solution $x(t)$.

Figure 2.5: $\lambda_1 = \alpha + \beta i, \lambda_2 = \alpha - \beta i, \alpha = 0, \beta > 0 \Rightarrow$ origin is a *center*.



(a) Phase portrait. (b) A typical solution $x(t)$.

Figure 2.6: $\lambda_{1,2} = \alpha \pm \beta i, \alpha > 0, \beta > 0 \Rightarrow$ origin is a *stable spiral*.



(a) Phase portrait. (b) A typical solution $x(t)$.

Figure 2.7: $\lambda_{1,2} = \alpha \pm \beta i, \alpha < 0, \beta > 0 \Rightarrow$ origin is an *unstable spiral*.

2.1.2 Nonlinear systems

So far, we have discussed only linear systems, in which case we can analyze their behaviour by finding the stable states and determining their type. We would like to do something similar in the case of nonlinear systems. To do so, we will approximate the behaviour of a nonlinear system near its stable states by a linear system.

Let

$$\begin{aligned} \dot{x} &= f(x,y) \\ \dot{y} &= g(x,y) \end{aligned} \tag{2.6}$$

be an autonomous system in plane with a steady state (x_0, y_0). Assume a linear transformation transforming the point (x_0, y_0) to the origin: $X = x - x_0$, $Y = y - y_0$. We can use the Taylor expansion to get:

$$\dot{X} = f(x + x_0, y + y_0) = f(x_0, y_0) + X\frac{\partial f}{\partial x}\Big|_{x=x_0,y=y_0} + Y\frac{\partial f}{\partial y}\Big|_{x=x_0,y=y_0} + R_x(X,Y),$$
$$\dot{Y} = g(x + x_0, y + y_0) = g(x_0, y_0) + X\frac{\partial g}{\partial x}\Big|_{x=x_0,y=y_0} + Y\frac{\partial g}{\partial y}\Big|_{x=x_0,y=y_0} + R_y(X,Y),$$

where nonlinear terms R_x a R_y satisfy $\frac{R_x}{r} \to 0$ a $\frac{R_y}{r} \to 0$ for $r = \sqrt{X^2 + Y^2} \to 0$. If we drop these terms and substitute for $f(x_0, y_0) = g(x_0, y_0) = 0$ ($[x_0, y_0]$ is a steady state of the system (2.6)), we obtain a *linearized system*:

$$\begin{aligned} \dot{X} &= X\frac{\partial f}{\partial x}\Big|_{x=x_0,y=y_0} + Y\frac{\partial f}{\partial y}\Big|_{x=x_0,y=y_0} \\ \dot{Y} &= X\frac{\partial g}{\partial x}\Big|_{x=x_0,y=y_0} + Y\frac{\partial g}{\partial y}\Big|_{x=x_0,y=y_0} \end{aligned} \tag{2.7}$$

with Jacobian matrix

$$J(x_0, y_0) = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{pmatrix}\Bigg|_{x=x_0,y=y_0}.$$

Definition 2.5. Let (x_0, y_0) be a steady state of the system (2.6), $J_{(x_0,y_0)}$ be its Jacobian matrix at this point. If all eigenvalues of this matrix have nonzero real parts, the point (x_0, y_0) is called *hyperbolic*.

Theorem 2.1 (Hartman-Grobman)**.** *Let (x_0, y_0) be a hyperbolic steady state of the system* (2.6)*. Then there exists some neighbourhood of the point (x_0, y_0) in which the phase portrait of the nonlinear system* (2.6) *is similar to the phase portrait of the linearized one* (2.7) *(i.e., phase portraits are topologically conjugate in the neighbourhood of this point).*

Proof. Proof can be found in [18]. □

In following text we will use the notation:

$$f_1 = \frac{\partial f(x_0, y_0)}{\partial x},\ f_2 = \frac{\partial f(x_0, y_0)}{\partial y},$$
$$g_1 = \frac{\partial g(x_0, y_0)}{\partial x},\ g_2 = \frac{\partial g(x_0, y_0)}{\partial y}.$$

As a corollary of Theorem 2.1 we get the following theorem.

Theorem 2.2. *Let*

$$\begin{aligned} \dot{x} &= f(x, y) \\ \dot{y} &= g(x, y) \end{aligned} \tag{2.8}$$

be an autonomous system, assume $f(x, y)$ and $g(x, y)$ are continuous functions with continuous partial derivatives up to the second order in a neighbourhood of (x_0, y_0). Let $f(x_0, y_0) = g(x_0, y_0) = 0$ and assume

$$det \begin{pmatrix} f_1 & f_2 \\ g_1 & g_2 \end{pmatrix} \neq 0.$$

Then the point (x_0, y_0) is an isolated steady state of the system (2.8)*. Additionally:*

- *If the origin in the linearized system*

$$\begin{aligned} \dot{x} &= f_1 x + f_2 y \\ \dot{y} &= g_1 x + g_2 y \end{aligned} \tag{2.9}$$

  is a node, a spiral or a saddle, then the point (x_0, y_0) in the system (2.6) *is of the same type.*

- *If the origin in the linearized system* (2.9) *is a center, then the point (x_0, y_0) is a center or a spiral in the system* (2.8)*.*

Theorem 2.3. *Let* (x_0, y_0) *be a steady state of the autonomous system* (2.8) *and let*

$$J_{(x_0,y_0)} = \begin{pmatrix} f_1 & f_2 \\ g_1 & g_2 \end{pmatrix} \tag{2.10}$$

be a Jacobian matrix of this system at the point (x_0, y_0)*. Then the steady state* (x_0, y_0) *is stable if all eigenvalues of the matrix* (2.10) *have negative real parts.*

Proof. Proof can be found in [7]. □

The process of transforming a general system of ODEs to a linear one is called *linearization*. This method can be generalized for n-dimensional systems. By using linearization we can study qualitative behaviour of quite complicated systems near their steady states. However, for analysis of behaviour in surroundings of nonhyperbolic steady states, in greater distances of all steady states or in the case of more complicated systems, it is necessary to use more sophisticated methods or it is even impossible.

For this reason we will talk about temporal logics in next chapter. They can be utilized for specification of desired properties of a system, simulation or a real-world measurement. Satisfaction of these properties can be then automatically tested or verified.

Chapter 3

Reasoning about continuous time

In this chapter we discuss an extensively used formalism that enables us to capture and analyze different properties (with some relation to time) of real-time systems. As a system we can understand any entity or set of interacting parts evincing some measurable behaviour or output. As a good abstraction of systems we will use the systems of differential equations described in the the previous chapter. In Chapter 4 a logic designed specially for this abstraction is presented.

The connection between real-time logics and ODEs is straightforward – trajectories as solutions of ODEs are used as a model over which the formulae of a temporal logic are interpreted. It is important to realize that temporal logics provide not only tools for reasoning about various temporal properties, but also a language for their succinct description. This role of logic is discussed in Section 3.3.

3.1 Temporal logics

Temporal logics originated as modal logics [28] with the modal operators understood in a temporal manner. This formalism was utilized by Pnueli [27] for studying the time behaviour of systems. In general, temporal logics deal with ongoing behaviour of a system, actions executed by this system or changes of its states.

There are many branches of temporal logics and the full description of all of them is outside the scope of this paper. While going through some of them, we will follow the main goal – reasoning about nontrivial time dependent behaviour (like oscillations). For a review of temporal logics concerning real-time see [4]. The aspects of temporal logics dis-

cussed in this section follow a frame used in [2] where authors present several main decisions one have to make when creating a logic for description of real-time systems.

We skip some features of temporal logics like division according to the order of the logic (propositional, first- or higher-order) and focus on the aspects related to our goal, which is reasoning about continuous signals with real-valued time domain (e.g., an output of ODEs from Section 2.1). First division of temporal logics can be made by the nature of the system and the structure over which the logic is interpreted.

3.1.1 Nature of models

Linear-time

Linear-time logics are interpreted over linear structures. Each structure corresponds to a single run of a system. A typical example of a linear-time logic is LTL (also called *Propositional Temporal Logic* – PTL) [27]. We will briefly describe its syntax and semantics.

LTL formulae are defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2,$$

where $p \in P$ is and atomic proposition from a countable set P, $\neg$ and $\vee$ are propositional logical operators, $\mathbf{X}$ – *next* is a unary and $\mathbf{U}$ – *until* a binary temporal operator.

LTL formulae are interpreted over infinite sequences of states.

Definition 3.1. *State sequence* $\sigma = s_0 s_1 \ldots$ is an infinite sequence of *states*. Each *state* s_i is a subset $s_i \subseteq P$ of a countable set of propositions P. For propositions $p \in s_i$ we say that p holds in s_i and write $s_i \models p$.

Satisfaction relation is then defined inductively as follows:

$$
\begin{array}{lcl}
\sigma, i \models p & \iff & s_i \models p; \\
\sigma, i \models \neg\varphi & \iff & \sigma, i \not\models \varphi; \\
\sigma, i \models \varphi_1 \vee \varphi_2 & \iff & \sigma, i \models \varphi_1 \texttt{ or } \sigma, i \models \varphi_2; \\
\sigma, i \models \mathbf{X}\varphi & \iff & \sigma, i+1 \models \varphi; \\
\sigma, i \models \varphi_1 \mathbf{U} \varphi_2 & \iff & \exists k \geq i : \sigma, k \models \varphi_2 \wedge \forall j, i \leq j < k : \sigma, i \models \varphi_1.
\end{array}
$$

There are some additional operators derived from the above like *future (or eventually)* – $\mathbf{F}\varphi \equiv \mathsf{true}\mathbf{U}\varphi$, *globally* – $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$ and others.

Branching-time

Branching-time logics, on the other hand, are interpreted over tree structures. They talk about process containing some decision points, i.e., time instants where the future behaviour of a system is not determined. A classical representatives of branching-time logics are Computational Tree Logic (CTL) [8] or CTL* [15], which combines both LTL and CTL. The trajectories of ODE systems defined in the previous chapter (Section 2.1) are linear – the next step is unambiguously determined in every time instant. Therefore we will use the linear-time semantics from now on.

3.2 Logics for real-time

Next question is how to deal with the real-time. We have to be able to speak about time intervals relativized to the time in which the formula is investigated. Consider a property:

Every action a is followed by a reaction b within 3 seconds. (3.1)

This is an example of a property with a time constraint. If we use a fictitious-clock semantics [2] where the time flows only in discrete steps (or we are able to measure the time events with finite precision only), we could describe the property (3.1) in LTL as:

$$\varphi = \mathbf{G}(a \Rightarrow (b \vee \mathbf{X}b \vee \mathbf{XX}b \vee \mathbf{XXX}b)).$$

However, in a world with dense time (applies for both the real world and its abstraction in a form of ODEs), we cannot use directly LTL because the operator next $\mathbf{X}$ is not defined on a dense domain and the operator until $\mathbf{U}$ (or rather future $\mathbf{F}$) is not able to restrict the interval in which the property should hold. Formula:

$$\mathbf{G}(a \Rightarrow \mathbf{F}b)$$

is true even if the action a is followed by the reaction b with a delay grater then 3 seconds. There are several possibilities how to extend the standard temporal logics to deal with temporal constraints:

- **Bounded temporal operators** – the validity of temporal operators is restricted on an interval;
- **Freeze quantification** – a freeze quantifier binds a variable in time;
- **Explicit clock variable** – a new variable represents actual time.

3.2.1 Bounded temporal operators

One way is to incorporate a time interval into the standard temporal operators. By doing so we get bounded until $\mathbf{U}_{[a,b]}$, bounded future $\mathbf{F}_{[a,b]}$, etc. The meaning is straightforward – their validity is restricted to a certain interval only, e.g., $\mathbf{F}_{[1,3]}\,\varphi$ is interpreted as *"the property φ holds at some time between 1 and 3 time units from now"*. With this enriched logic we can express the property (3.1) by the formula:

$$\mathbf{G}_{[0,\infty)}\,(a \Rightarrow\ \mathbf{F}_{[0,3]}\,b).$$

An example of a temporal logic implementing this way of relativizing time is the Metric Interval Temporal Logic (MITL) [1]. Formulas of MITL are inductively defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1\, \mathbf{U}_I\, \varphi_2,$$

where $p \in P$ is an atomic proposition and I is a *nonsingular* interval (i.e., not of the form $[a, a]$) with rational end-points.

MITL formulae are interpreted over *timed state sequences*.

Definition 3.2. *Timed state sequence* τ is a pair $\tau = (s, I)$ where $s = s_0 s_1 s_2 \ldots$ is a *state sequence* from Definition 3.1 and $I = I_0 I_1 I_2 \ldots$ is an infinite sequence of time intervals such that:

- I_0 is left-closed with the left end-point $l(I_0) = 0$;
- for all $i \geq 0$ the intervals $I_i, I_i + 1$ are adjacent, i.e., $r(I_i) = l(I_{i+1})$ and either I_i is right-closed and I_{i+1} is left-closed or vice versa;
- every time $t \geq 0$ belongs to some interval I_i.

We can notice that I partitions the nonnegative real line, i.e., $\bigcup_{i \geq 0} I_i = \mathbb{R}_0^+$ and for every $i, j \geq 0, i \neq j : I_i \cap I_j = \emptyset$.

Given a MITL formula φ, a timed state sequence $\tau = (s, I)$ and a time instant $t \in \mathbb{R}_0^+$, the satisfaction relation $\tau \models \varphi$ is defined inductively as follows:

$$
\begin{array}{lcl}
\tau, t \models p & \iff & p \in s_i, \text{ where } t \in I_i; \\
\tau, t \models \neg\varphi & \iff & \tau, t \not\models \varphi; \\
\tau, t \models \varphi_1 \vee \varphi_2 & \iff & \tau, t \models \varphi_1 \text{ or } \tau, t \models \varphi_2; \\
\tau, t \models \varphi_1 \, \mathbf{U}_I \, \varphi_2 & \iff & \exists t' \in I + t \text{ such that } \tau, t' \models \varphi_2 \wedge \\
 & & \forall t'', t < t'' < t' : \tau, t'' \models \varphi_1.
\end{array}
$$

The expression $I + t$ denotes the interval $\{t + t' \mid t' \in I\}$. The timed state sequence τ satisfies the formula φ iff $\tau, 0 \models \varphi$.

Again, we can define derived operators as $\mathbf{F}_I \, \varphi \equiv \text{true} \, \mathbf{U}_I \, \varphi$ and $\mathbf{G}_I \, \varphi \equiv \neg \, \mathbf{F}_I \, \neg\varphi$.

3.2.2 Freeze quantification

Another way how to involve the real-time into a temporal logic is by a quantifier which freezes (binds) a variable in the corresponding time. This idea was first introduced in the Timed Propositional Temporal

Logic (TPTL) [3, 19]. The freeze quantifier $x.$ freezes the associated variable x to the time of the local temporal context: The formula $x.\varphi(x)$ holds at time t iff $\varphi(t)$ does.

The property (3.1) can be expressed in this formalism as:

$$\psi = \mathbf{G}x.(a \Rightarrow \mathbf{F}y.(b \wedge y \leq x + 3)),$$

which asserts that *"whenever the proposition a holds in a state with time x, then there is a later state with time y, in which b holds and the time y is at most $x + 3$"*.

One should remark that by freeze quantification we can express same properties as by using the bounded temporal operators 3.2.1 and even some additional properties. We can relate nonadjacent temporal contexts like for example *"every stimulus p is followed by a response q and, then, by another response r such that r is within 5 time units of the stimulus p"*. There is no direct way how to express this property by bounded operators, however, it is done by the formula with freeze quantifier:

$$\psi = \mathbf{G}x.(p \Rightarrow \mathbf{F}(q \wedge \mathbf{F}z.(r \wedge z \leq x + 5))).$$

For up to date information about freeze quantification see [12].

3.2.3 Explicit clock variable

The third way of expressing timing constraints is based on the first-order temporal logic. It is done by using a dynamic state variable T, the *clock variable*, representing the current time [17]. The property (3.1) can be expressed as follows:

$$\psi = \mathbf{G}((a \wedge x = T) \Rightarrow \mathbf{F}(b \wedge T \leq x + 3)).$$

The actual value of time in a state where a holds is stored in x and later, when b occurs, it is compared to the actual time in that state.

3.3 Temporal properties over continuous signals

So far, we have discussed temporal logics suitable for expressing non-trivial real-time properties. But they lack an easy way of specifying and

extracting properties of continuous signals, e.g., the output of a system of differential equations (see Section 2.1).

Having a set of continuous signals produced by an ODE system (Section 2.1) or a set of time series acquired by measurement of real world phenomena, we would like to have a tool by which we could specify interesting properties over these signals and automatically monitor whether the system satisfies them. Logic, which enables us to do the above mentioned things, is the *Signal Temporal Logic (STL)* [26].

It is a temporal logic designed to speak about finite-length continuous signals. It does not work with continuous signals directly but rather via predicates describing their properties. These predicates are used as atomic propositions in $\text{MITL}_{[a,b]}$ – a fragment of MITL (see Section 3.2.1) adjusted for finite-length models.

STL offers a way to specify a large amount of properties of continuous signals, however, at the cost of explicitly constructed predicates or additional signal variables. For more complex relations, which cannot be captured by MITL, it is necessary to have an extra variable containing the auxiliary information about the property and a predicate specifying time intervals in which the target property holds (e.g., to have a variable $\dot{x}$ providing information about the derivative of x and a predicate $\dot{x} > 0$ telling when x is increasing).

But this is not a typical situation we are in. Commonly, we have only a few time series of data measured on our system of interest. We have the actual values of the variables at some time instants, we might have their derivatives, but that is all. We are facing a problem to specify the desired properties of the signal with fixed set of signal variables. If we had the possibility to add variables to the signal, we would be able, in an extreme case, to express the whole desired property as a single predicate over some artificial variable. Thus, there would be no need to use a temporal logic.

If we settle on the fixed set of signal variables, we have to look for another tools to express properties like *"the variable x is increasing"* or *"the variable x is bigger than y was 5 time units ago"*. This relations be-

tween values of variables in different time instants cannot be expressed in STL (see Section 4.2.3). In Chapter 4, I am proposing a temporal logic derived from STL. With restrictions placed on the structure of atomic predicates, and, on the other hand, with enhanced expressivity by an added temporal operator.

Chapter 4

Extended temporal logic for continuous signals

In the previous chapter, we have seen some approaches to the problem of specification of properties of signals with possibly both dense domain and range. In my opinion, the mentioned logics are not well suited for the practical use for this problem. They lack two features I attach weight to. First, they do not specify the process of transformation of the analyzed signals to the propositional variables, and second, their expressive power is insufficient for specification of required properties. An exception is STL, which concerns with the first point.

In this chapter, I am proposing a temporal logic to address both problems. The logic is based on STL, from which it draws the idea of continuous signals and their translation to atomic predicates. However, there is given a clear restriction to the form of the predicates so that it is feasible both to compute their values and to reason about the formulae of the logic. The second contribution of this logic is the possibility of succinct specification and distinction of larger classes of signals. This is achieved by one new temporal operator which freezes the values of the signal for the use deeper in the formula.

4.1 Signals

In this section we briefly describe real-valued signals over continuous time. These signals are defined in the same way as in [26] and we will use them in the predicates in STL* logic.

Definition 4.1. Let $T = [0, r], r \in \mathbb{Q}$ be a finite interval. A *finite length signal s over a domain D* is a function $s : T \to D$.

We say that the length of the signal s is r and we denote it as $|s| = r$.

Definition 4.2. *Real-valued signal* is a signal from Definition 4.1 where $D = \mathbb{R}^n$. Given a real-valued signal $s : T \to \mathbb{R}^n$, the *order of the signal* s is the number n. We use s_i to denote the i-th component of a signal s. It is a signal $s_i : T \to \mathbb{R}$ defined as $s_i(t) \stackrel{\text{df}}{=} \pi_i(s(t))$ where π_i is the projection from n-tuples to their i-th component.

A signal of the order n represents one finite run of a continuous-time system with n variables. These variables can be any quantities of the system, e.g., concentrations of species, their derivatives or any other attributes of the signal which can be measured and quantified.

We need a tool to speak about properties of a signal in terms of logic. To do so, we define a function transforming real values of the signal variables in every time instant to the Boolean set $\{0, 1\}$, representing satisfaction or dissatisfaction of one property over one signal. This truth values can be then used as propositional variables in some logic.

We will restrict the allowed operations on signal variables to linear combinations and simple comparisons. Such restriction enables us to algorithmically determine satisfaction of a property of a signal in every time instant. By using better algorithms or less precise solution we could loosen the demands on the form of the properties.

Definition 4.3. *Linear predicate ν of the order n* is a function $\nu : \mathbb{R}^n \to \mathbb{B}$ of the form $\sum_{i=1}^{n} a_i x_i \sim b$ where $a_i, b \in \mathbb{R}$ are real coefficients, x_i are input variables, $\sim \in \{<, \leq, >, \geq, =\}$ is one of the listed relational operators and $\mathbb{B}$ is a Boolean set $\mathbb{B} = \{0, 1\}$. The predicate $\nu(x_1, x_2, \ldots x_n)$ returns 0 if the expression $\sum_{i=1}^{n} a_i x_i \sim b$ is false and 1 if it is true.

However, in the logic defined in this chapter, we need not only to work with the values of variables acquired in one time instant, but also with a second set of values from another time. For this reason, we will define an additional function extracting truth values from the signal.

Definition 4.4. *Atomic predicate* μ is a function $\mu : S_m \times \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{B}$ where S_m is a class of signals of order m. For each signal $s \in S_m$ and

$t, t^* \in [0, |s|]$, μ is defined by the formula:

$$\mu(s, t, t^*) \stackrel{\text{df}}{=} \nu(s_1(t), \ldots, s_m(t), s_1(t^*), \ldots, s_m(t^*)),$$

where ν is a linear predicate of the order $2m$ and s_i are components of the signal s. Values of the atomic predicate μ for $t > |s|$ or $t^* > |s|$ are considered undefined.

An atomic predicate over a signal s describes a Boolean property of the signal s, i.e., if a constrained specified by the linear predicate ν is satisfied with respect to two time instants t and t^*. An example of such property is:

$$\nu(x_1, x_2, x_3, x_4) \stackrel{\text{df}}{=} x_1 - x_2 - x_3 + x_4 < 0,$$

which is equivalent to:

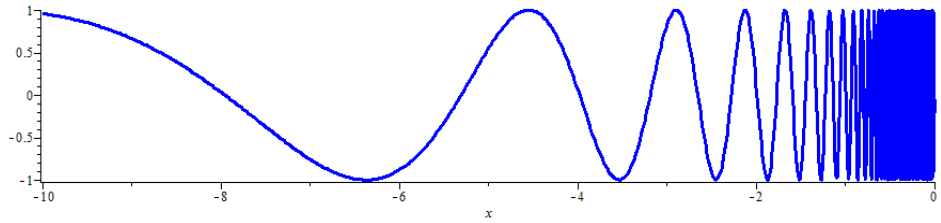
$$x_1 - x_2 < x_3 - x_4.$$

This property tells us that under condition $t^* < t$, the difference between values of variables x_1 and x_2 was higher in the time t^* than in the time t:

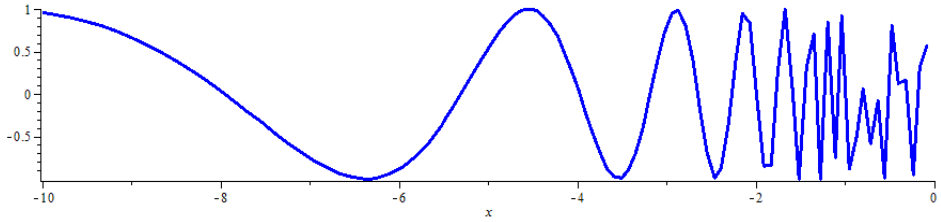
$$\mu(s, t, t^*) \equiv \nu(s_1(t), s_2(t), s_1(t^*), s_2(t^*)) \equiv s_1(t) - s_2(t) < s_1(t^*) - s_2(t^*).$$

We will avoid the undesired cases of atomic predicates where the output values of the predicate are varying infinitely often. From now on, we will deal only with signals which values, with respect to all used atomic predicates, changes at most finitely often. It is a reasonable presumption because the signals of our interest are usually outputs of some systems measured in finite-length time steps. Such a pathological behaviour would vanish during this process of sampling and cannot reappear due to linear form of atomic predicates (Figure 4.1).

Instead of $\mu(s, t, t^*) \equiv s_1(t) - s_2(t) < s_1(t^*) - s_2(t^*)$ we usually write $\mu(s, t, t^*) \equiv x - y < x^* - y^*$, denoting the variables of a signal s in time t by letters $x, y, z, \ldots$ and the same variables in time t^* as $x^*, y^*, z^*, \ldots$.



(a) Pathological continuous signal with infinitely many changes between negative and positive values ($y = sin(\frac{50}{x})$).



(b) Same signal measured with finite sampling frequency. The undesired behaviour disappeared due to the sampling.

Figure 4.1: The pathological behaviour of a signal disappears during the sampling.

4.2 STL* logic

In this section we will describe the syntax and semantics of STL*. It is based on the Signal Temporal Logic (STL) [26] extended with the freeze operator $*$ and with restricted form of atomic predicates.

4.2.1 Syntax

The formulae of the STL* are inductively defined by the following grammar:

$$\varphi ::= \mu_i \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \, \mathbf{U}_{[a,b]} \, \varphi_2 \mid *\varphi,$$

where μ_i are atomic predicates from Definition 4.4, $\neg$ and $\vee$ are standard operators for negation and disjunction taken from the propositional logic, $\mathbf{U}_{[a,b]}$ is a bounded until operator constrained by the non-singular time interval $[a, b]$ with rational end-points. The interpretation of this operator is similar to the one used in the Metric Interval Tempo-

ral Logic (MITL) [1]. Finally, $*$ is the newly added unary freeze operator.

In the standard way, we can derive additional temporal operators such as eventually ($\mathbf{F}_{[a,b]}\,\varphi \equiv \text{true}\ \mathbf{U}_{[a,b]}\,\varphi$) and globally ($\mathbf{G}_{[a,b]}\,\varphi \equiv \neg\,\mathbf{F}_{[a,b]}\,\neg\varphi$). These operators are then also bounded by an interval $[a, b]$ and their interpretation follows from the interpretation of $\mathbf{U}_{[a,b]}$.

4.2.2 Semantics

Formulae of STL* are interpreted over triplets (s, t, t^*) where s is a real-valued signal and $t, t^* \in [0, |s|]$ are two time instants. We write:

$$(s, t, t^*) \models \varphi,$$

iff a formula φ is satisfied on the triplet (s, t, t^*). The satisfaction of an STL* formula is defined inductively to its structure:

$$\begin{array}{lcl}
(s,t,t^*) \models \mu & \iff & \mu(s,t,t^*) = 1; \\
(s,t,t^*) \models \neg\varphi & \iff & (s,t,t^*) \not\models \varphi; \\
(s,t,t^*) \models \varphi_1 \vee \varphi_2 & \iff & (s,t,t^*) \models \varphi_1 \text{ or } (s,t,t^*) \models \varphi_2; \\
(s,t,t^*) \models \varphi_1\ \mathbf{U}_{[a,b]}\,\varphi_2 & \iff & \exists t' \in [a+t, b+t] : (s,t',t^*) \models \varphi_2 \wedge \\
 & & \forall t'' \in [t,t'] : (s,t'',t^*) \models \varphi_1; \\
(s,t,t^*) \models *\varphi & \iff & (s,t,t) \models \varphi.
\end{array} \tag{4.1}$$

Because models for STL* are signals, we also speak about satisfaction of a formula over a signal.

Definition 4.5. The satisfaction of a formula φ over a signal s is defined as:

$$s \models \varphi \iff (s, 0, 0) \models \varphi.$$

The meaning of the operator $*\varphi$ is to freeze the values of the signal over which the formula is interpreted in the current time point so we can use these values inside the subformula φ. In combination with other temporal operators, interesting properties can be expressed such as *"the value of the variable x is nondecreasing on the interval I"*:

$$\varphi_1 \equiv\ \mathbf{G}_I * \big[\, \mathbf{G}_{[0,\epsilon]}\,(x^* \leq x)\big],$$

where $\epsilon > 0$ is some small constant. Or *"at some point in the future (during interval I) the value x increases by the value of 5 within two time units"*:

$$\varphi_2 \equiv \mathbf{F}_I * \big[\mathbf{F}_{[0,2]} \, (x^* + 5 = x) \big].$$

Determination of satisfaction of a formula φ over a signal s for $t > |s|$ or $t^* > |s|$ is impossible due to lack of information about values of atomic predicates at these time instants. This follows from the finite nature of considered signals and it is also the reason why the temporal operator $\mathbf{U}_I$ is bounded only by constrained interval. To avoid problems with undefined values, we can compute the sufficient length $l(\varphi)$ of a signal inductively according to the structure of the interpreted formula φ:

$$\begin{array}{lcl} l(\mu) & \stackrel{\text{df}}{=} & 0; \\ l(\neg\varphi) & \stackrel{\text{df}}{=} & l(\varphi); \\ l(\varphi_1 \vee \varphi_2) & \stackrel{\text{df}}{=} & \max(l(\varphi_1), l(\varphi_2)); \\ l(\varphi_1 \, \mathbf{U}_{[a,b]} \, \varphi_2) & \stackrel{\text{df}}{=} & \max(l(\varphi_1), l(\varphi_2)) + b; \\ l(*\varphi) & \stackrel{\text{df}}{=} & l(\varphi). \end{array} \tag{4.2}$$

From Definition 4.5 we can see that if a formula φ contains an atomic predicate μ_i and there is no operator $*$ wrapping this predicate then all variables concerning the frozen time instant t^* are handled as if $t^* = 0$.

On the other hand, in a formula with several nested operators $*$, the meaning of frozen variable x^* is local, i.e., it relates to the nearest operator $*$. For example, in a formula:

$$*\big[x^* \leq y \, \mathbf{U}_I \, (x^* = y \wedge *\big[\, \mathbf{G}_{[0,5]} \, x^* \geq y \big])\big], \tag{4.3}$$

the first and the second occurrence of the variable x^* relates to the first usage of $*$ and addresses the time $t = 0$, in which the formula is interpreted. The third occurrence of x^*, in contrast to the previous two, relates to the second operator $*$ and addresses the time instant in which the subformula on the right side of $\mathbf{U}_I$ becomes true.

The property described by the formula (4.3) can be roughly expressed as *"when y becomes larger than the actual value of x on the interval I for the first time (say at time $t = t_0$), it will not fall under the value of $x(t_0)$ for the next 5 time units"*.

4.2.3 Comparison with other logics

In this section we have a think about different types of oscillations and how they can or cannot be expressed in STL and STL*. We restrict ourselves to predicates of the form of inequations with linear combinations of measured values on both sides. As a model, we take time series of values of continuous variables or as the case may be their first derivatives. This is typically all for the use of an experimenter or any scientist who wants to reason about their model. Our goal in this section is to justify the need of an additional temporal operator to enhance the expressivity of the new logic described in Chapter 4.

All features are demonstrated on a signal s of one variable x of the length $|s| = 80$ (i.e., defined on the interval $[0, 80]$ time units) and the logic STL* is compared to STL only. Other real-time temporal logics mentioned in Section 3.1 can express similar properties when bounded with the same restrictions on atomic predicates.

First, start with a simple property *"the signal s is oscillating"*. This property could be true for all possible signals because we have not specified the period of the oscillations and thus we could consider even a constant signal to be oscillating with a very long period. The property has to be formulated more precisely:

The signal s is oscillating with a period at most 10 time units. (4.4)

Denote this property as φ. We would like to check if the signal s satisfies the property φ. First, we have to formalize the property in some temporal logic. If we knew the derivatives in time of the variable x (denote them as $\dot{x} \stackrel{\text{df}}{=} \frac{dx}{dt}$), we could express the property φ in STL as:

$$\mathbf{G}_{[0,70]}\,((\dot{x} \geq 0 \Rightarrow\ \mathbf{F}_{[0,10]}\,\dot{x} < 0) \wedge (\dot{x} \leq 0 \Rightarrow\ \mathbf{F}_{[0,10]}\,\dot{x} > 0)). \tag{4.5}$$

This is a robust formula describing many different oscillatory signals (Figure 4.2). However, we do not know the derivatives of the variable x and even if we knew, the problem would be only shifted. What if we wanted to know if the derivatives oscillate?

Neither STL, nor other temporal logic, as far as I know, provide any possibility to express the whole class of oscillations like those depicted in Figure 4.2 or to distinguish among them. It could be done for particular signals with fixed parameters, but not universally without referencing to concrete values. In STL*, there are multiple ways how to achieve that. For example, we could define a property similar to $\dot{x} < 0$, but without the need of the derivative $\dot{x}$:

$$*\big[\, \mathbf{F}_{[0,\epsilon]}\, x^* > x\big] \tag{4.6}$$

for sufficiently small $\epsilon > 0$.

Substituting the derivatives in (4.5) for (4.6) we get

$$\mathbf{G}_{[0,70-\epsilon]}\,((*\big[\, \mathbf{F}_{[0,\epsilon]}\, x^* < x\big] \Rightarrow \mathbf{F}_{[0,10]}\, *\big[\, \mathbf{F}_{[0,\epsilon]}\, x^* > x\big]) \wedge (*\big[\, \mathbf{F}_{[0,\epsilon]}\, x^* > x\big] \Rightarrow \mathbf{F}_{[0,10]}\, *\big[\, \mathbf{F}_{[0,\epsilon]}\, x^* < x\big])).$$

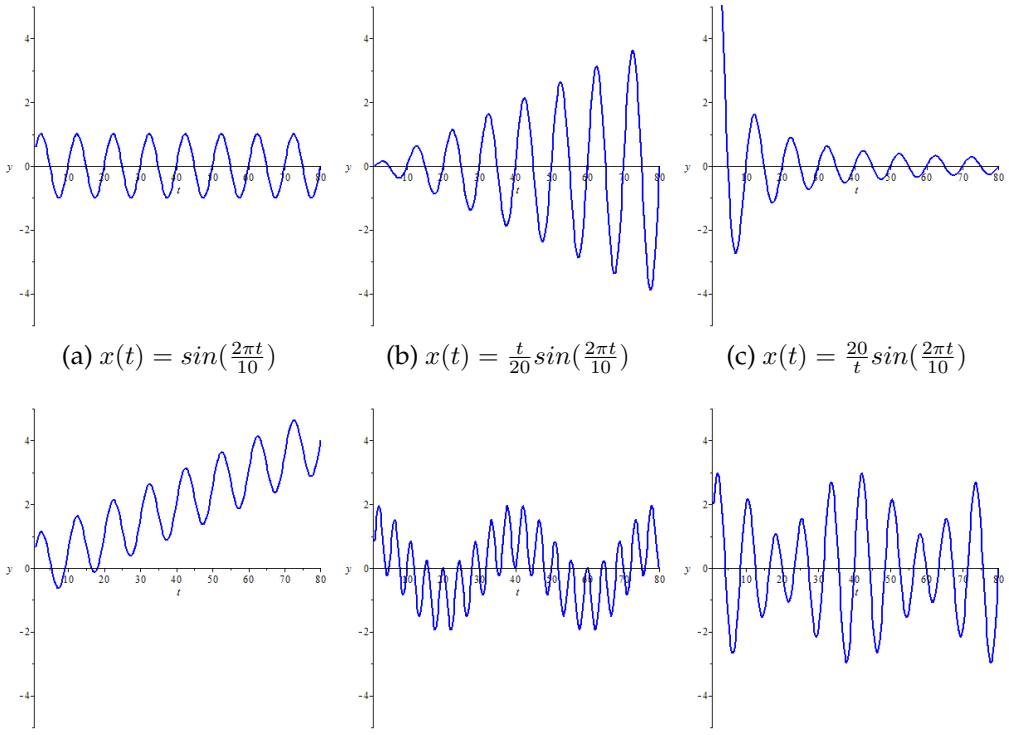
Notice that by working on an interval $[0, \epsilon]$ instead of in one precise point, we would need a slightly longer signal. Or better, as in this formula, we have to slightly reduce the interval on which the satisfaction of the operator $\mathbf{G}$ is monitored.

Another way is to express the oscillations as follows:

$$\mathbf{G}_{[0,60]}\ \mathbf{F}_{[0,10]}\, *\big[(\, \mathbf{F}_{[0,10]}\, x^* < x) \wedge\ \mathbf{F}_{[0,10]}\, x^* > x)\big],$$

which says *"for every time instant, there is a near future, when the value of x will be both smaller than after some time, but also greater than after another time"*. In other words, *"a system repeatedly displaces from some states and returns to them afterwards"*. Which corresponds to the definition of oscillations from Chapter 2.

One could think of more different formulae describing the class of oscillatory signals and they could vary in some marginal cases. This depends on what exactly do we accept as the desired behaviour.



(a) $x(t) = sin(\frac{2\pi t}{10})$ (b) $x(t) = \frac{t}{20} sin(\frac{2\pi t}{10})$ (c) $x(t) = \frac{20}{t} sin(\frac{2\pi t}{10})$

(d) $x(t) = sin(\frac{2\pi t}{10}) + \frac{t}{20}$ (e) $x(t) = sin(\frac{2\pi t}{10}) 2sin(\frac{2\pi t}{8})$ (f) $x(t) = sin(\frac{2\pi t}{10}) + 2sin(\frac{2\pi t}{8})$

Figure 4.2: Various types of oscillations.

A typical phenomenon in biological systems is a *damped oscillation*. It is an oscillation with gradually decreasing amplitude (Figure 4.2c). This property can be expressed by the formula:

$$\mathbf{G}_{[0,60]} \left((\mathbf{F}_{[0,10]} * \left[\mathbf{G}_{[0,10]}\, x^* \geq x \right]) \wedge (\mathbf{F}_{[0,10]} * \left[\mathbf{G}_{[0,10]}\, x^* \leq x \right]) \right). \tag{4.7}$$

It says that *"there is always a time instant in near future which is a local maximum for some future interval and there is also another time instant in near future which is a local minimum for some future interval"*. In these intervals, another local maxima and minima for more distant intervals occur, and so on. This implies that the signal values can be surrounded by a nonincreasing function from top and a nondecreasing function from bottom.

Among the signals in Figure 4.2, only the signals a) and c) satisfy the formula (4.7). The signal a) is a marginal case of damped oscillation with zero damping. We cannot get rid of this case by simply substituting strict inequalities for the nonstrict in (4.7) because the condition $\mathbf{G}_{[0,10]}\, x^* > x$ would never hold for $x^* = x$. However, we could shift the left point of the interval and add a constant $c > 0$ to the formula (4.7) to ensure the damping would proceed with some speed:

$$\mathbf{G}_{[0,60]}\,((\,\mathbf{F}_{[0,10]} * \big[\,\mathbf{G}_{[1,10]}\, x^* \geq x + c\big])\wedge(\,\mathbf{F}_{[0,10]} * \big[\,\mathbf{G}_{[1,10]}\, x^* \leq x - c\big])). \tag{4.8}$$

Another class of properties which cannot be expressed in STL are the relations between variables in different times. An example is the property *"x copies the values of y with a delay of 4 seconds"* or equally *"x equals to the value y had 4 seconds ago"*. In STL*, it corresponds to the formula:

$$\mathbf{G}_{[0,80-(4+\delta)]} * \big[\,\mathbf{G}_{[4-\delta,4+\delta]}\, y^* = x\big], \tag{4.9}$$

for a small $\delta > 0$ (recall temporal operators can be bounded only by nonsingular intervals. It is due to the same restriction in MITL, which is a necessary condition for MITL to be decidable [1]).

4.3 STL* formulae monitoring procedure

In this section we deal with monitoring of an STL* formula, which means to determine the satisfaction of a formula over a finite length signal. Unlike model checking (i.e., determining if a formula is satisfied by all runs of a system [8, 9]), monitoring is much simpler task [26]. It does not guarantee the satisfaction over the whole space of solutions of a system, but it can be performed repeatedly to sample the searched space to increase the knowledge about the system. It can be also used for parameter estimation (Chapter 6) or more complex machine learning methods [5].

My monitoring procedure for STL* introduced in this chapter is inspired by constrained LTL model-checking [5] and monitoring of STL formulae [26], but needed to be extended into 2-D space due to freeze

operator $*$. It is because the task is solved simultaneously for two time instants, actual time and frozen time.

The idea of the monitoring procedure is to construct a parse tree of the formula and check the satisfaction in a bottom-up manner. First step in checking a formula φ over a signal s is to construct a set of time points in which the formula φ is satisfied.

Definition 4.6. Let φ be a STL* formula and s be a real-valued signal. A set $S_{\varphi,s} = \{(t, t^*) \in [0, |s|] \times [0, |s|] \mid (s, t, t^*) \models \varphi\}$ is called *the satisfaction set of formula φ over signal s*. We omit the s and use short-form notation S_φ if the signal s is obvious from the context.

Now we inductively construct satisfaction sets for nodes on higher levels of the parse tree. The last step of the procedure is to decide if $s \models \varphi$ from satisfaction set for the formula φ. According to Definition 4.5, it is equivalent to checking whether the satisfaction set contains the point $(0, 0)$.

Constructing a set of satisfaction for a general signal can be a very difficult task. For this reason we will construct the monitoring algorithm only for piecewise linear signals. This is a reasonable requirement because in most cases we deal with time series produced by numerical simulations of modelled systems or by some measurements. These series can be interpreted as piecewise linear signals considering the values changing linearly between two adjoining points. If required, another points can be generated in between the existing points to make the signal more precise. The assumption of piecewise linear signals in combination with linearity of atomic predicates (Definitions 4.3, 4.4) enable us to construct the satisfaction sets for atomic predicates in polynomial time and to easily work with these sets in the process of construction of sets belonging to higher formulae.

4.3.1 Algorithm for exact monitoring of piecewise linear signals

In this section, the idea of an exact monitoring algorithm for piecewise linear signals is presented.

Definition 4.7. A *piecewise linear signal* s is a real-valued signal of order n where all the projections $s_i(t) : [0, |s|] \to \mathbb{R}, i = 1 \dots n$ are piecewise linear functions defined on a finite set of intervals $J = \{I_1, I_2, \dots, I_n\}$, $I_j \subseteq [0, |s|], \bigcap J = \emptyset, \bigcup J = [0, |s|]$.

Theorem 4.1 (Inductive construction of the satisfaction set)**.** *Let s be a piecewise linear signal and φ a formula in STL*. The satisfaction set S_φ of a formula φ over a signal s can be constructed inductively with respect to the parse tree of the formula φ:*

$$\begin{array}{lcl} S_\mu & = & \{(t, t^*) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \mid \mu(s, t, t^*) = 1\}; \\ S_{\neg\varphi} & = & \overline{S_\varphi}; \\ S_{\varphi_1 \vee \varphi_2} & = & S_{\varphi_1} \cup S_{\varphi_2}; \\ S_{\varphi_1 \, \mathbf{U}_{[a,b]} \, \varphi_2} & = & \{(t, t^*) \in S_{\varphi_1} \mid \exists t' \in [t + a, t + b] : (t', t^*) \in S_{\varphi_2} \wedge \\ & & \forall t'' \in [t, t'] : (t'', t^*) \in S_{\varphi_1}\}; \\ S_{*\varphi} & = & \{(t, t^*) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \mid (t, t) \in S_\varphi\}. \end{array}$$

Proof. Each of the equations follows directly from the definition of semantics of STL* (4.1). □

Definition 4.8. *Convex polytope* $P \subseteq \mathbb{R}^n$ is a set $\{x = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n \mid Ax \leq b\}$ where A is a $m \times n$ matrix and b is a $n \times 1$ column, both in real numbers. A convex polytope in one dimension is called a *line segment*, in 2 dimensions a *convex polygon*.

Each row in $Ax \leq b$ in Definition 4.8 represents one linear inequality which can be imagined as hyperplane dividing the space $\mathbb{R}^n$ into two subspaces. The convex polytope is the intersection of one half of these subspaces.

Lemma 4.1. Suppose we have a piecewise linear signal s and let J be the set of intervals on which s behaves linearly. Let μ be an atomic predicate over the signal s. The predicate $\mu(s, t, t^*) = \sum_{i=1}^n a_i s_i(t) + \sum_{i=1}^n b_i s_i(t^*) \sim b, \sim \in \{<, \leq, >, \geq, =\}$ is a linear function over the signal s in variables t and t^* according to Definition 4.4. Denote $S_{i,j}^+ \stackrel{\text{df}}{=} \{(t, t^*) \in I_i \times I_j \mid \mu(s, t, t^*) = 1\}$ the set of time instants at which the atomic predicate μ is satisfied over signal s on rectangular area $I_i \times I_j$ and $S_{i,j}^- \stackrel{\text{df}}{=} (I_i \times I_j) \backslash S_{i,j}^+ = \{(t, t^*) \in I_i \times I_j \mid \mu(s, t, t^*) = 0\}$ its complement. For $t \in I_i, t^* \in I_j; I_i, I_j \in J$ there can arrive two situations:

1. if $\sim \equiv =$, then the set $S^+_{i,j}$ makes a line segment (possibly degenerated to a single point or an empty set);

2. if $\sim \not\equiv =$, then the space $I_i \times I_j$ is divided by the line $\sum_{i=1}^{n} a_i s_i(t) + \sum_{i=1}^{n} b_i s_i(t^*) = 0$ into two subspaces $S^+_{i,j}$ and $S^-_{i,j}$ (Figure 4.3). (Again, there might be a degenerated case when the line $\sum_{i=1}^{n} a_i s_i(t) + \sum_{i=1}^{n} b_i s_i(t^*) = 0$ does not cross the rectangle $\{(t, t^*) \in I_i \times I_j\}$ in which case $S^+_{i,j} = \emptyset$ and $S^-_{i,j} = I_i \times I_j$ or vice versa).

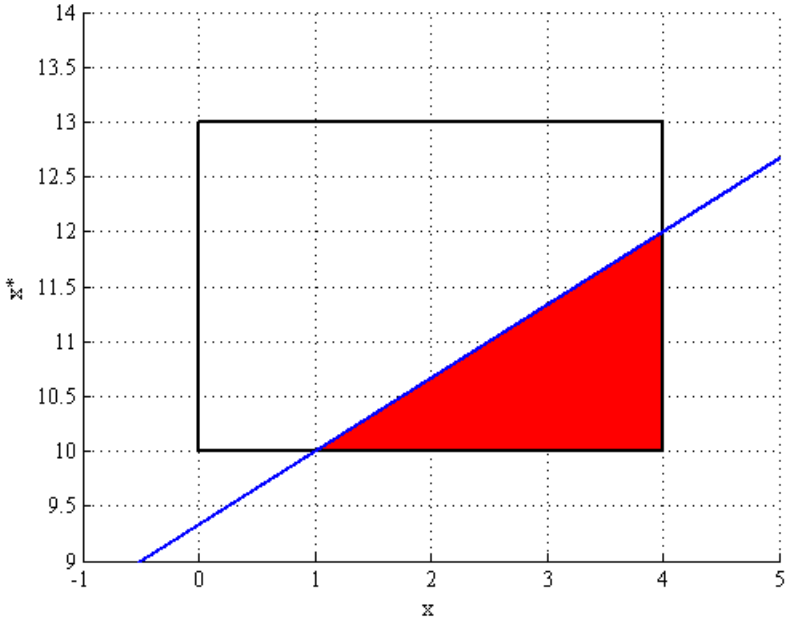


Figure 4.3: An illustration of the set $S^+_{i,j}$ for a formula $x^* < \frac{2}{3}x + \frac{28}{3}$ on the area $I_i \times I_j = [0, 4] \times [10, 13]$.

In both cases, the set $S^+_{i,j}$ makes a convex set easily describable by set operations over convex polytopes. In the first case, a line itself is a convex polytope (two inequalities $\sum_{i=1}^{n} a_i x_i \leq b$ and $-\sum_{i=1}^{n} a_i x_i \leq -b$ degrade one dimension into a line and a line segment can be acquired by intersection with the rectangle $I_1 \times I_2$), in the second case, either the set $S^+ \cap I_1 \times I_2$ is a polytope or (when $\sim \equiv <$ or $\sim \equiv >$) we have to

exclude one bordering line segment which is a polytope by the former case.

Using this algorithm we can express the satisfaction set $S_{\mu,s}$ as $S_{\mu,s} = \bigcup_{I_i,I_j \in J} S^+_{i,j}$, where $S^+_{i,j}$ are the sets described above represented as set operations on a few polytopes. The set $S_{\mu,s}$ can be further simplified by joining adjacent polytopes and removing redundant conditions. Having satisfaction sets for atomic predicates, we could inductively construct satisfaction sets for composite formulae from Theorem 4.1 again as operations on polytopes, which could be achieved in polynomial time [10].

However, working with precise satisfaction sets as defined in Theorem 4.1 can be quite difficult task. We have to work with polytopes of different dimensions (polygons, lines and points) and operations over sets of these objects can be unnecessarily demanding. In the real-world applications, we could end up performing an expensive monitoring over noisy signals measured on imperfect devices or computed on computers with finite numerical precision. Hence think about less precise, yet faster monitoring algorithm.

4.3.2 Approximative monitoring algorithm for piecewise linear signals

We can imagine a *noisy signal* as a signal measured or computed with finite precision. The actual values of the variables can differ from the values presented by the signal up to some error $\epsilon > 0$. This inaccuracy in the domain of values implies also inaccuracy in the time domain. When investigating the time instant in which some event occurred (e.g., $x > 0$) we cannot be sure when exactly it happened. That is because the critical value $x = 0$ is burdened by an error and it is possible that $x = 0$, but also that $x > 0$ or $x < 0$ in that particular time instant.

Suppose we have a noisy piecewise linear signal s measured or computed with some error $\epsilon > 0$, let J be the set of intervals on which s behaves linearly. Let $\mu(s, t, t^*)$ be an atomic predicate over the signal s. For $t \in I_i, t^* \in I_j; I_i, I_j \in J$ there could arrive two situations as discussed in

Lemma 4.1:

1. if $\sim \equiv =$, then the set $S^+_{i,j}$ makes a line segment;
2. if $\sim \not\equiv =$, then the space $I_i \times I_j$ is divided by the line $\sum_{i=1}^{n} a_i s_i(t) + \sum_{i=1}^{n} b_i s_i(t^*) = 0$ into two subsets $S^+_{i,j}$ and $S^-_{i,j}$.

However, we cannot be sure about the border of the set $S^+_{i,j}$ due to the error ϵ. The condition $\sum_{i=1}^{n} a_i s_i(t) + \sum_{i=1}^{n} b_i s_i(t^*) = b$ defining points on the border of $S^+_{i,j}$ could be easily broken by changing the values of the signal s by arbitrary small values from $(0, \epsilon)$. For this reason, we will ignore these border points and count them as they were in $S^+_{i,j}$ or in $S^-_{i,j}$ depending on what would be computationally easier. As a consequence, in the first case above, $S^+_{i,j} = \emptyset$, and in the second case, it does not matter if $\sim \ \equiv \ <$ or $\sim \ \equiv \ \leq$.

It is necessary to show that this simplification is justified and what are the consequences of it. First, the set of allowed operators in atomic predicates shrunk to $\{<, >\}$. The remaining three lost their sense. And it is true that with signals burden with some error $\epsilon > 0$ the satisfaction of atomic predicates can be determined only for time instants inside the satisfaction set (Figure 4.4). Hence the satisfaction of a formula using the operator $=$, e.g., $x = b$ or $x = y + 2$, cannot be determined. Every reference to precise value have to be replaced by an sufficiently large interval, e.g., formula $x = b$ can be replaced by $x \geq (b - \delta) \wedge x \leq (b + \delta)$ for some $\delta > \epsilon/2$.

Based on these assumptions, monitoring algorithm is more simple. For every pair of intervals $I_i, I_j \in J$, the satisfaction set on this area can be described by a single convex polygon $S^+_{i,j}$. For the whole satisfaction set $S_{\mu,s}$ we get $S_{\mu,s} = \bigcup_{I_i, I_j \in J} S^+_{i,j}$. The problem of construction of satisfaction sets for composite formulae from Theorem 4.1 is converted to operations with sets of convex polytopes in plane for which efficient algorithms exist [10]. During the process, adjacent polygons can be connected into bigger ones if this resulting polygons are convex, which could significantly improve computational time.

Algorithm 4.1 (Approximative monitoring algorithm for piecewise linear signals)**.**

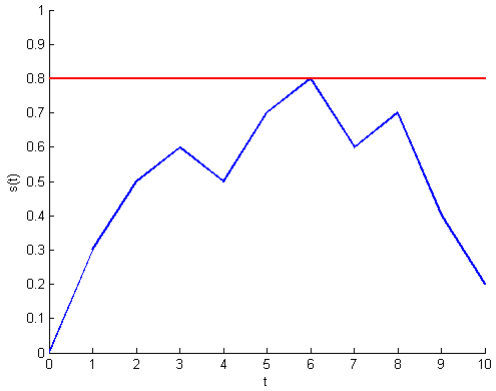


Figure 4.4: A signal measured with some error $\epsilon > 0$. The satisfaction of a formula $\mathbf{F}_{[0,10]}(s = 0.8)$ cannot be determined. Condition $s = 0.8$ might be satisfied in some time in a neighbourhood of $t = 6$ (size of the neighbourhood depends on ϵ), but it cannot be said reliably.

Input: A piecewise linear signal s and a STL* formula φ.
Output: Answer to the question $s \models \varphi$.

Algorithm inductively constructs the satisfaction set S_φ:

All the computations are performed on parts of the signal s of sufficient length. This can be computed according to (4.2). If the signal does not have the sufficient length, the answer returned by the algorithm might be wrong.

1. $S_\mu = \bigcup_{I_i, I_j \in J} S^+_{i,j}$. The computation of the satisfaction sets $S^+_{i,j}$ for individual intervals is performed according to Lemma 4.1, but without determining the satisfaction of the borders (as was justified in this section).

2. $S_{\neg\varphi} = (J \times J) \setminus S_\varphi$. The result can be computed as a simple Boolean operation on polygons. It is necessary to ensure that the resulting set consists only of convex polygons, which could be achieved for example by triangulation [10].

3. $S_{\varphi_1 \vee \varphi_2} = S_{\varphi_1} \cup S_{\varphi_2}$. Again a Boolean polygonal operation.

4. $S_{*\varphi} = \{(t, t^*) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \mid (t, t) \in S_\varphi\}$. The satisfaction set for the freeze operation can be computed by: (1) finding an intersection between the line $t^* = t$ and the satisfaction set S_φ, (2) making a projection to the first component t and then a Cartesian product with J (Figure 4.5).

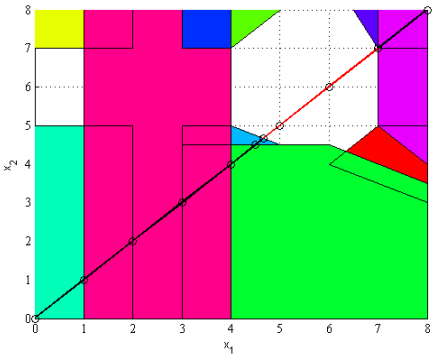
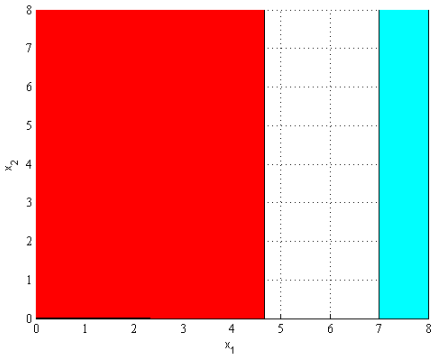
a)  b) 

Figure 4.5: Example of computing satisfaction set for the formula $*\varphi$. a) The satisfaction set represented as a set of (overlapping) convex polygons and a line $x_1 = x_2$. The regions of intersection of the line and the satisfaction set are depicted in black color. b) Resulting set $S_{*\varphi}$ obtained by projecting the intervals of intersection to the axis x_1 and making a Cartesian product with the axis x_2.

5. $S_{\varphi_1 \mathbf{U}_{[a,b]} \varphi_2} = \{(t, t^*) \in S_{\varphi_1} \mid \exists t' \in [t + a, t + b] :$
$(t', t^*) \in S_{\varphi_2} \wedge \forall t'' \in [t, t'] : (t'', t^*) \in S_{\varphi_1}\}$. For each convex polygon $P_k \in S_{\varphi_1} \cap S_{\varphi_2}$ the following procedure is performed:

   (a) Vertices of polygons $\{P_k\} \cup S_{\varphi_1}$ are ordered by the second component t^* (duplications can be removed). Denote the ordered set $V \stackrel{\text{df}}{=} (V_1, V_2, \ldots V_n)$;

   (b) For $i = \{1, 2, \ldots n - 1\}$ every neighbouring pair of vertices $V_i = (t_i, t_i^*), V_{i+1} = (t_{i+1}, t_{i+1}^*) \in V$ specifies a rectangular polygon

$$R_{k,i} \stackrel{\text{df}}{=} \{(t, t^*) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \mid t_i^* \leq t^* \leq t_{i+1}^*\}$$

(we could use $>$ instead of $\geq$ because we do not care about border points. For the same reason, $R_{k,i}$ in a form of line segment is considered empty.) Notice that $\bigcup_i (R_{k,i} \cap P_k) = P$.

(c) The task is solved in every stripe $R_{k,i}$ separately. For every nonempty $R_{k,i}$ denote $P_{k,i} \stackrel{\text{df}}{=} R_{k,i} \cap P_k$ and $S_{k,i} \stackrel{\text{df}}{=} R_{k,i} \cap S_{\varphi_1}$. Due to the way of construction of the stripe $R_{k,i}$, all the polygons in $P_{k,i}$ and the one polygon in $S_{k,i}$ have all their vertices on the upper or lower border of the stripe. In fact, they can take only the shape of a triangle or a trapezoid (Figure 4.6).

Now we get rid of the polygons lying on the right side of $S_{k,i}$ because their points cannot be in the solution (they represent the time past the event φ_2). To do so, we isolate the upper and lower rightmost points of the polygon $S_{k,i}$. Denote them as $u = (u_t, u_{t^*})$ and $l = (l_t, l_{t^*})$. We get a new area

$$R'_{k,i} \stackrel{\text{df}}{=} \{(t, t^*) \in R_{k,i} \mid (t, t^*) \text{ lies on the left side of the line } \overrightarrow{ul}\}.$$

The solution can lie only in this area, so we can restrict $P_{k,i}$ to $P'_{k,i} \stackrel{\text{df}}{=} R'_{k,i} \cap P_{k,i}$.

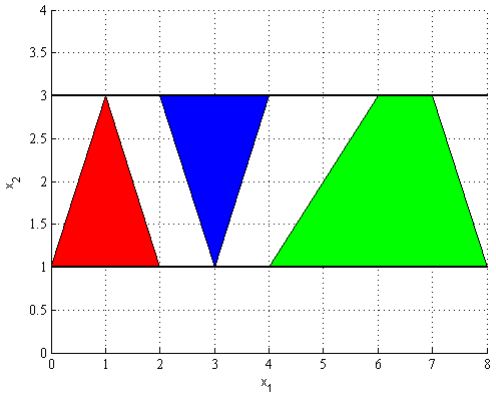


Figure 4.6: An example of a stripe and all possible shapes of polygons inside a stripe.

(d) All the adjoining polygons (sharing an edge) in $P'_{k,i}$ are connected to form maximal seamless convex polygons.

(e) Let $A_{i,k}$ be the rightmost polygon in $P'_{k,i}$ (even after connecting some polygons in $P'_{k,i}$ their shape can still be only triangular or trapezoidal (Figure 4.6) so there is only one rightmost polygon in $P'_{k,i}$). Then $A'_{i,k} = A_{i,k} \cap (S_{k,i} \ominus (b, a))$ is the solution for the stripe $R_{k,i}$.

The operation $\ominus$ is defined as $A \ominus (a, b) \stackrel{\text{df}}{=} \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid \exists c \in (a, b) : (x + c, y) \in A\}$ which is equal to $A \oplus (-a, -b)$, where $\oplus$ is the Minkowski sum.

(f) The final satisfaction set is given by $S_{\varphi_1 \, \mathbf{U}_{[a,b]} \, \varphi_2} = \bigcup_{i,k} A'_{i,k}$.

6. The result $s \models \varphi$ which is equivalent to $(0, 0) \in S_\varphi$ is returned.

4.3.3 Example of a run of the approximative monitoring algorithm

We demonstrate the computation of Algorithm 4.1 on the signal s from Figure 4.2c. The formula to be monitored is:

$$\varphi \stackrel{\text{df}}{=} \mathbf{G}_{[1,58]} \underbrace{((\mathbf{F}_{[0,11]} * \big[\mathbf{G}_{[0,11]} \, x^* \geq x \big]) \wedge (\mathbf{F}_{[0,11]} * \big[\mathbf{G}_{[0,11]} \, x^* \leq x \big]))}_{\psi}. \tag{4.10}$$

The first step is the computation of satisfaction sets for $\mu_1(s, t, t^*) \stackrel{\text{df}}{=} x^* \geq x$ and $\mu_2(s, t, t^*) \stackrel{\text{df}}{=} x^* \leq x$. In fact, we could compute only one of them and the second get by polygonal operations, because one is the complement to the other (due to the simplification explained in 4.3.2). One of the resulting sets after connecting some neighbouring polygons is depicted in Figure 4.7.

Next step is the computation of the satisfaction set for the formulae $\mathbf{G}_{[0,11]} \, \mu_1$ and $\mathbf{G}_{[0,11]} \, \mu_2$. It is transformed according to definition to $\mathbf{G}_{[0,11]} \, \mu_i \equiv \neg \mathbf{F}_{[0,11]} \, \neg \mu_i \equiv \neg(\text{true} \, \mathbf{U}_{[0,11]} \, \neg \mu)$. For the result see Figure 4.9.

The freeze operator is applied on both sets and the corresponding satisfaction sets are computed (Figure 4.8). After that, a future operator $\mathbf{F}_{[0,11]}$ is applied (Figure 4.10), resulting in full sets $[1, 58] \times [1, 58]$. The operation $\wedge$ corresponds to the intersection of these two sets, which

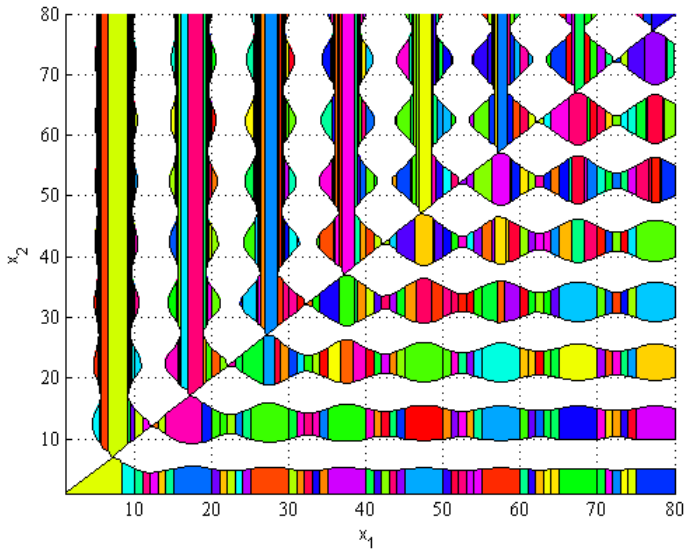
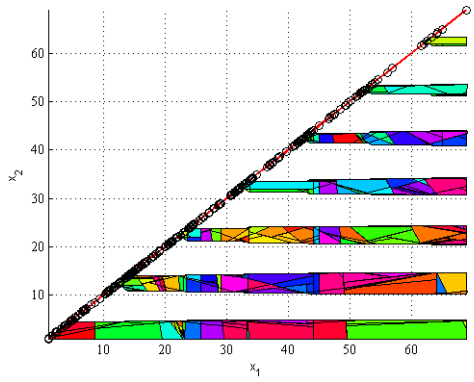
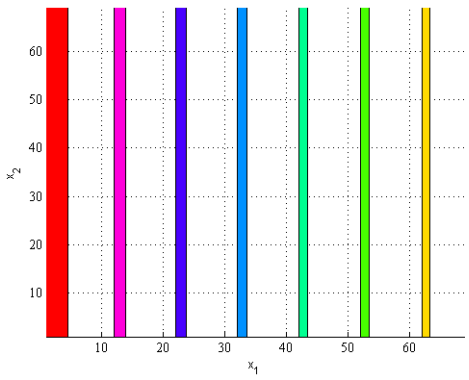


Figure 4.7: The satisfaction set for the atomic predicate $\mu_1(s,t,t^*) = x^* \geq x$ over the signal 4.2c.

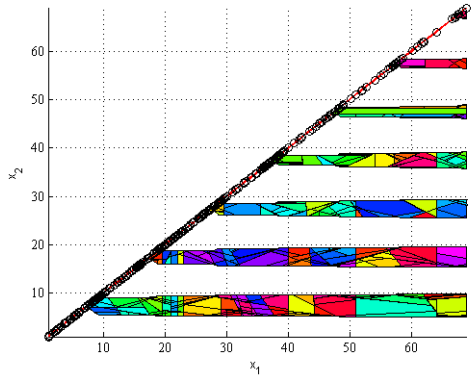
gives again a full set $S_\psi = [1,58] \times [1,58]$. After application of the last operator $\mathbf{G}_{[1,58]}$ we get the satisfaction set for the formula φ over the signal s: $S_\varphi = (0,0)$ (only the point $(0,0)$ is valid because only in the time 0 the signal has sufficient length). The result "TRUE" is returned because $(0,0) \in S_\varphi$.



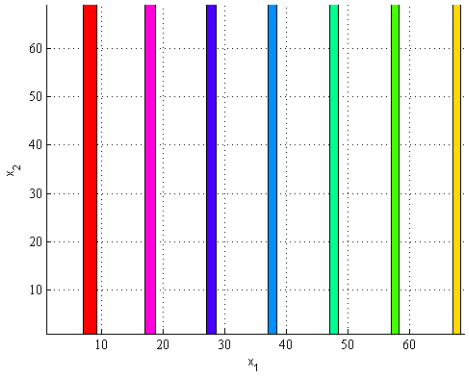
(a) Intersection of a line $t = t^*$ and $S_{\mathbf{G}_{[0,11]}\,\mu_1}$.



(b) Satisfaction set $S_{*\left[\mathbf{G}_{[0,11]}\,\mu_1\right]}$

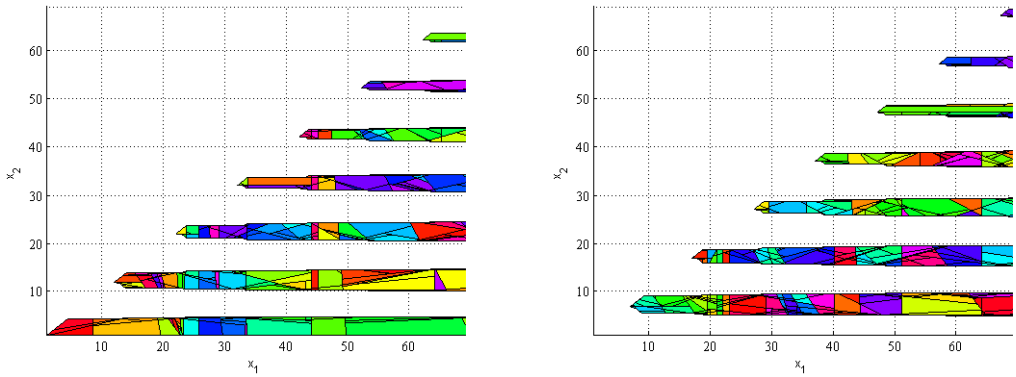


(c) Intersection of a line $t = t^*$ and $S_{\mathbf{G}_{[0,11]}\,\mu_2}$.



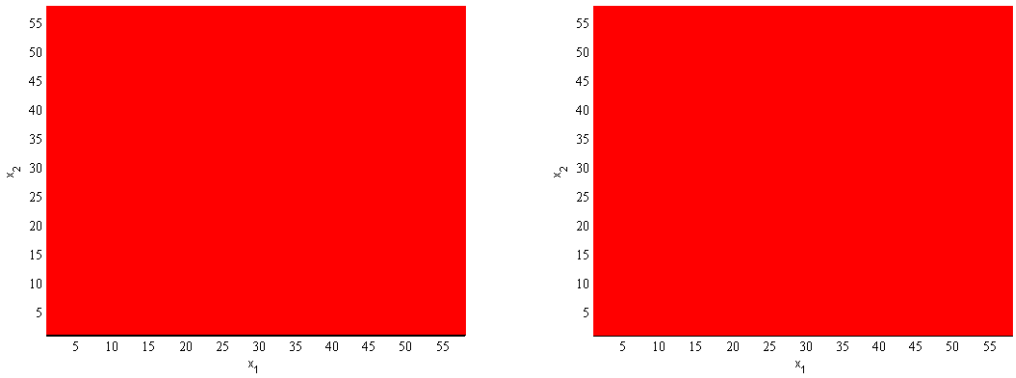
(d) Satisfaction set $S_{*\left[\mathbf{G}_{[0,11]}\,\mu_2\right]}$

Figure 4.8: Satisfaction sets for the operator $*$.



(a) $S_{\mathbf{G}_{[0,11]}\,\mu_1}$ (b) $S_{\mathbf{G}_{[0,11]}\,\mu_2}$

Figure 4.9: Satisfaction sets for the operator $\mathbf{G}_I$.



(a) $S_{\mathbf{F}_{[0,11]}*\left[\mathbf{G}_{[0,11]}\,\mu_1\right]}$ (b) $S_{\mathbf{F}_{[0,11]}*\left[\mathbf{G}_{[0,11]}\,\mu_2\right]}$

Figure 4.10: Satisfaction sets for the operator $\mathbf{F}_I$. The results are full sets.

Chapter 5

Implementation

A prototype of the approximative monitoring Algorithm 4.1 was implemented in MATLAB [25] (version R2011b). The Multi-Parametric Toolbox (MPT) package [22] (version 2.6.3) was used for the polygonal operations. Commented source code of the program is on CD attached to this thesis.

5.1 Algorithm

The implemented algorithm works as described in Algorithm 4.1. The list of all functions and their short descriptions follows, for the exact information and documentation see the CD.

5.1.1 List of main functions

- `[ P ] = checkPoints( f, f2, condition, interval, debug )`

  Computes the satisfaction set for atomic predicate $f_2(t^*) \sim f(t)$, where $\sim =$ `condition`. The computation is performed on interval `interval`. The optional parameter `debug` enables/disables the debug mode. The satisfaction set is returned as polytope `P`. This function corresponds to Algorithm 4.1.

- `[ P3 ] = intersectionSS( P, P2 )`

  Computes the intersection of two satisfaction sets `P`, `P2`.

- `[ P3 ] = unionSS( P, P2 )`

  Computes the union of two satisfaction sets `P`, `P2`.

- `[ P2 ] = complementSS( P, interval )`

  Computes the complement of the satisfaction set `P` on the interval `interval`.

- `[ P3 ] = untilSS( P1, P2, interval, area, debug )`

  Computes the satisfaction set for $S_{\varphi_1 \, \mathbf{U}_I \, \varphi_2}$, where $S_{\varphi_1} =$ `P1`, $S_{\varphi_2} =$ `P2` and $I =$ `interval`. The computation is performed on the area `area` $\times$ `area`.

- `[ P2 ] = futureSS( P, interval, area, debug )`

  Computes the satisfaction set for $S_{\mathbf{F}_I \, \varphi}$, where $S_\varphi =$ `P` and $I =$ `interval`. The computation is performed on the area `area` $\times$ `area`.

- `[ P2 ] = globallySS( P, interval, area, debug )`

  Computes the satisfaction set for $S_{\mathbf{G}_I \, \varphi}$, where $S_\varphi =$ `P` and $I =$ `interval`. The computation is performed on the area `area` $\times$ `area`.

- `[ P2 ] = freezeSS( P, interval, debug )`

  Computes the satisfaction set for $S_{*\varphi}$, where $S_\varphi =$ `P`. The computation is performed on the interval `interval`.

- `plotSS( P, interval )`

  Plots the satisfaction set `P` restricted on the area `interval` $\times$ `interval`.

- `satisfactionSS( P, debug )`

  Checks whether the satisfaction set P is satisfied over the signal, i.e., whether $[0, 0] \in P$. Returns 1 if true and 0 if false.

- `[ P ] = checkPoints2( f, f2, condition, interval, debug )`

  Computes the satisfaction set for atomic predicate $f_2(t^*) \sim f(t)$, where $\sim =$ `condition`. The computation is performed on interval `interval`. The optional parameter `debug` enables/disables the debug mode. The satisfaction set is returned as polytope `P`.

  This is an alternative version of the function `checkPoints`. The satisfaction set is not computed exactly as in Algorithm 4.1. Instead, some more points are added to the functions f, f_2 using an auxiliary function `slice()` so that the vertices of the convex polygons can lie only in these points. The resulting polygons can be in some cases bigger which can lead to their smaller count and hence faster computation. However, in the worst case scenario, this function is slower than the function `checkPoints`.

5.1.2 Time and space complexity

Time and space complexity of all the implemented functions is proportional to the number of polygons they are working with. The function `checkPoints` takes two piecewise linear functions defined on the interval of interest by n_1 and n_2 points. Number of generated polygons is at most $(n_1 + 1)(n_2 + 1) = n_1 n_2 + n_1 + n_2 + 1$. Which is asymptotically $O(n^2 + 2n + 1) = O(n^2)$, where $n = \max(n_1, n_2)$.

The number of polygons does not asymptotically change during the computation of `intersectionSS`, `unionSS` and `complementSS` and all these functions do not work slower than $O(n^2)$ (for more details see the source code of this functions and corresponding functions from MPT). The output polygons also keep small number of vertices during the process.

The function `untilSS` (and hence also functions `futureSS` and `globallySS`) works in $O(n^2)$ and it does not asymptotically change number of output polygons. The function `freezeSS` works in linear time and can only lower the number of polygons, the function `plotSS` works in linear time.

The upper bound on the total computation time is therefore $O(kn^4)$, where n is the number of intervals on which the signal is defined and k is the size of the investigated formula.

5.2 Evaluation

All the computations for this thesis were done by the implemented algorithm (Section 5.1). Both the computations and tests were performed on a notebook with an Intel Core i5-430M processor (2.26 GHz) and 4 GB RAM.

Speed of individual computations depends not only on the length of the signal and the complexity of the monitored formula, but also on the character of the signal (e.g., fast oscillating signal can cause more frequent changing of the monitored properties leading to higher number of polygons).

Although the theoretical time complexity of the algorithm is polynomial and was approximately cubic in practical cases (Figure 5.1), the computations take quite a long time (over an hour for signals defined on 100 points). It can be attached to the slow computations of Boolean polygonal operations and especially of the algorithm for merging small polygons into larger ones. Functions from MPT package are used for these tasks. This package is not specialized for polygonal operations, but was chosen because it contained all the functions needed for the prototype implementation. The computations could be accelerated by using or implementing more specialized algorithms.

Tests comparing performance of the functions checkPoints and checkPoints2 (Tables 5.1,5.2) shows that the first function is faster, but produces more polygons.

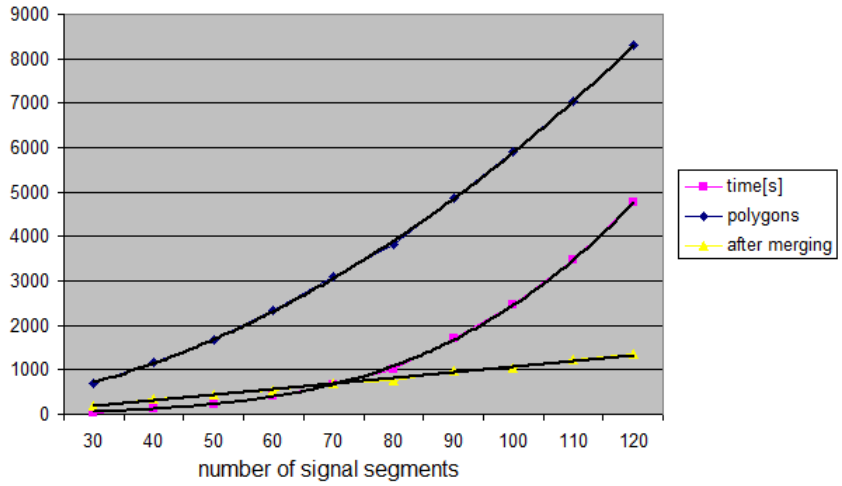


Figure 5.1: Graph of performance characteristics of the checkPoints algorithm with interpolated polynomials. The tested property was $x^* < x$. Time is interpolated by a cubic polynomial, number of generated polygons by a parabola and number of polygons after merging is interpolated by a line.

	checkPoints			checkPoints2		
segments	polygons	after merging	time [s]	polygons	after merging	time [s]
20	346	98	13	1060	168	87
30	702	203	47	1947	263	263
40	1161	351	114	2874	310	545
50	1685	436	226	3474	403	781
60	2339	536	424	4262	472	1156
70	3080	706	675	4991	558	1588
80	3813	748	1022	3884*	552*	1015*
90	4867	987	1717	6488	708	2669
100	5904	1039	2459	7133	780	3168
110	7038	1229	3460	7765	849	3766
120	8290	1351	4757	8629	931	4929

Table 5.1: Comparison between two implementations of algorithm for creating satisfaction set for atomic predicates. A property $x^* \leq x$ was monitored over the signal depicted in Figure 4.2c. In this case (values corresponding to both actual and frozen time are taken from one variable), there are no significant differences in running time between both functions, but the function checkPoints2 produces less polygons. *Segments* = number of linear segments the signal consists of, *polygons* = number of initially generated polygons, *after merging* = number of polygons after merging, *time* = running time of the algorithm. * = the unexpected acceleration for signal of 80 line segments was caused by concurrence of sampling frequency and signal frequency. This resulted in smaller number of sliced points (see Section 5.1.1 or documentation for the function checkPoints2).

	checkPoints			checkPoints2		
segments	polygons	after merging	time [s]	polygons	after merging	time [s]
20	256	73	10	1247	77	116
30	534	121	27	2720	124	477
40	836	198	61	4246	187	1167
50	1232	277	137	5273	214	1752
60	1710	386	226	6817	266	2716
70	2266	467	381	8293	337	4108
80	2829	499	597	7765*	316*	3663*
90	3549	606	882	11000	434	7233
100	4305	653	1334	12460	507	9190

Table 5.2: Comparison between two implementations of algorithm for creating satisfaction set for atomic predicates. A property $x^* \leq y$ was monitored over a signal of two oscillating functions. The function checkPoints ran ten times faster, but created more polygons than the function checkPoints2. *Segments* = number of linear segments of both variables of the signal, *polygons* = number of initially generated polygons, *after merging* = number of polygons after merging, *time* = running time of the algorithm. * = the unexpected acceleration for signal of 80 line segments was caused by concurrence of sampling frequency and signal frequency. This resulted in smaller number of sliced points (see Section 5.1.1 or documentation for the function checkPoints2).

Chapter 6

Case study

In this chapter I demonstrate a typical use of STL* and the monitoring algorithm. The main contribution is the possibility to express nontrivial properties of continuous signals in combination with ability to automatically test them. Imagine we have a model of some system and we want to find the parameters of the model such that its behaviour meets desired requirements. One way would be to perform a qualitative analysis of the underlying ODEs as described in Section 2.1. But it is not always feasible, especially for more complex systems. Also, the investigated model does not have to be in the form of differential equations. We can study parameters of systems defined in other formalisms or even existing only in the real world.

In these cases we have to relay on manual verification of satisfaction of desired properties. However, with the tool for algorithmic monitoring we can run a lot of experiments with different parameters and automatically verify which combination satisfies our needs. I demonstrate this on a biological case study.

We will study a biological system of three transcriptional repressors which was designed and built into bacteria *Escherichia coli* [14]. I have chosen this system because it is a biological oscillatory system, it is simple enough to perform the computations with the prototype algorithm (Section 5.1) and it is a well-studied system. The goal of this case study is primarily to demonstrate the ways of usage of STL*.

In [14], the authors designed a network of three genes influencing each other so that it causes periodical oscillation behaviour (Figure 6.2). Concentrations of involved proteins are periodically oscillating which

causes periodic production of a green fluorescent protein. Intensity of fluorescence of this protein, which can be measured, gives evidence of the ongoing activity of the network.

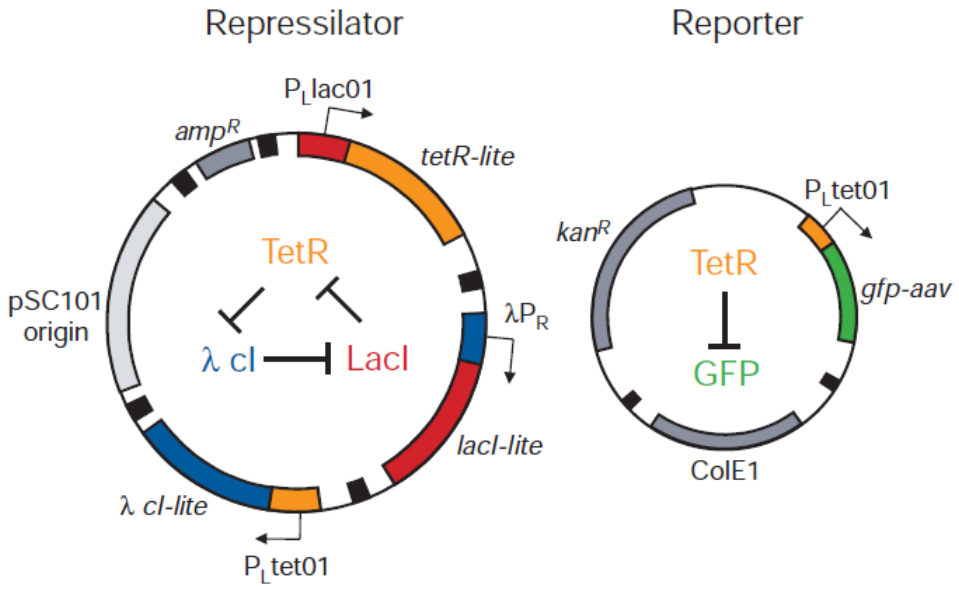


Figure 6.1: The repressilator network. Position of three genes on a plasmid (left circle). In the middle of the left plasmid is a scheme of gene interactions. On the second plasmid (right circle) is a gene for green fluorescent protein which is inhibited by one of the products of oscillating genes. Fluctuations in concentration of the product of tetR gene cause fluctuations in concentration of green fluorescent protein and hence in fluorescence level, which can be measured. Picture was taken from [14].

The network was initially modelled by a system of differential equations to estimate parameters producing the desired behaviour. The whole network was then introduced into bacteria *Escherichia coli* and the predicted oscillatory behaviour was tested by measuring the intensity of the fluorescence (Figure 6.2).

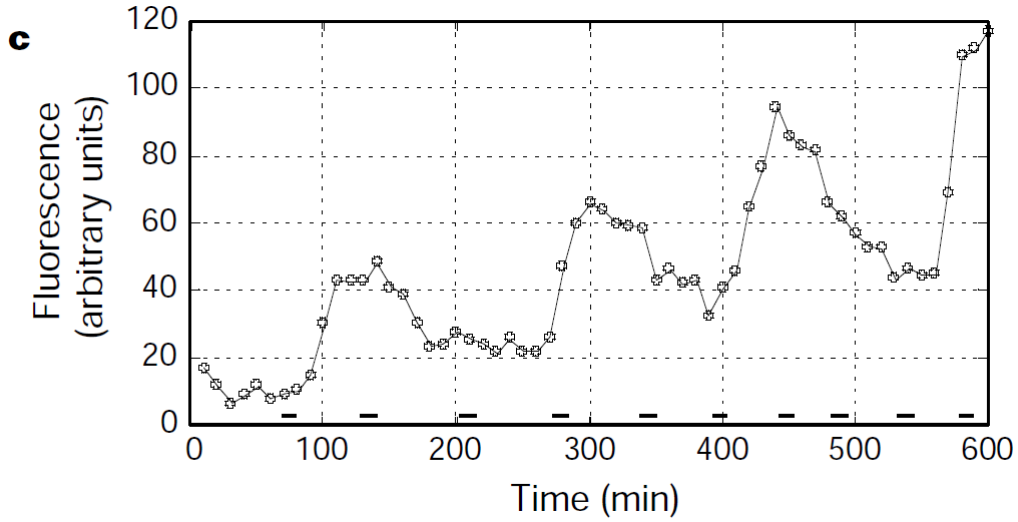


Figure 6.2: The fluorescent intensity of a colony of *Escherichia coli* containing the repressilator. The intensity is increasing overall due to the increasing number of individuals in the colony. The period of oscillations is lower than the time between cell division, but the whole repressilator is transmitted from generation to generation. Picture was taken from [14].

The network was modelled as a system of six differential equations:

$$\begin{aligned} \dot{m}_1 &= -m_1 + \tfrac{\alpha}{1+p_3^n} + \alpha_0; \\ \dot{m}_2 &= -m_2 + \tfrac{\alpha}{1+p_1^n} + \alpha_0; \\ \dot{m}_3 &= -m_3 + \tfrac{\alpha}{1+p_2^n} + \alpha_0; \\ \dot{p}_1 &= -\beta(p_1 - m_1); \\ \dot{p}_2 &= -\beta(p_2 - m_2); \\ \dot{p}_3 &= -\beta(p_3 - m_3). \end{aligned} \tag{6.1}$$

Where $\dot{m}_i \overset{\text{df}}{=} \frac{dm_i}{dt}, \dot{p}_i \overset{\text{df}}{=} \frac{dp_i}{dt}; m_1, m_2$ and m_3 correspond to activity of genes $lacl, tetR$ and cl; p_1, p_2 and p_3 are concentrations of corresponding proteins and $\alpha, \beta, \alpha_0, n$ are constants (Figure 6.3).

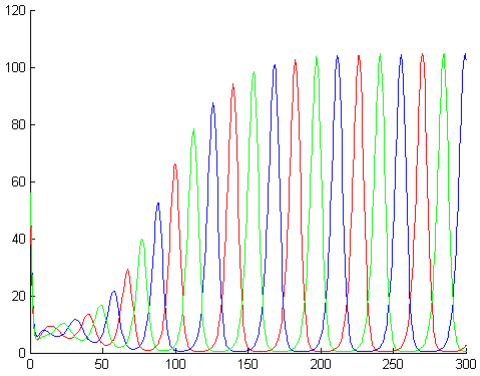


Figure 6.3: A typical signal produced by the system (6.1). Only variables m_1, m_2 and m_3 are depicted.

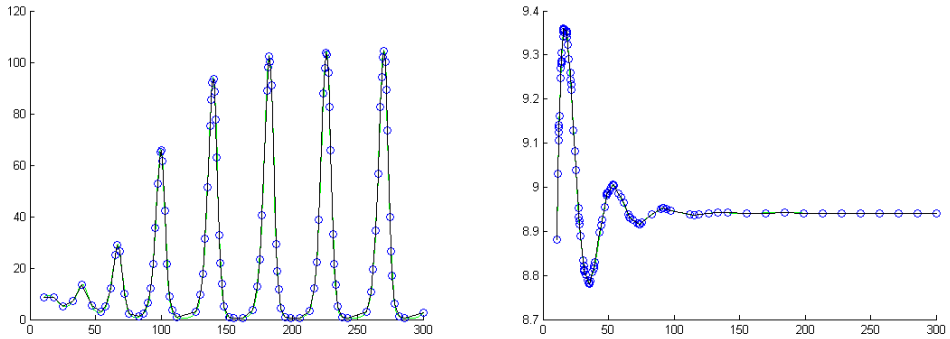
The analysis of the system (6.1) and the estimation of parameters $\alpha, \beta, \alpha_0, n$ producing the oscillatory behaviour was originally done by means of qualitative analysis of ODEs (Section 2.1). However, we could do the same using the monitoring of STL* formulae. We could specify the desired oscillatory property by a formula:

$$\varphi \stackrel{\text{df}}{=} \mathbf{G}_{[10,190]}\ \mathbf{F}_{[0,50]} * \big[(\ \mathbf{F}_{[1,50]}\, m_i^* < m_i) \wedge\ \mathbf{F}_{[1,50]}\, m_i^* > m_i)\big] \tag{6.2}$$

and test the satisfaction for different values of parameters on runs of the length at least 300 time minutes. The expected period has to be lower than 50 minutes. We start the testing 10 minutes later after the beginning to avoid the initial swing and end the testing 50 minutes before the end of the measurement because the signal might be cut in the middle of a period.

Formula (6.2) is satisfied by both runs in Figure 6.4. To avoid the case of damped oscillations, we can add a formula:

$$\psi \stackrel{\text{df}}{=} \mathbf{G}_{[10,200]} * \big[(\ \mathbf{F}_{[1,50]}\, m_i^* \leq m_i)\big]. \tag{6.3}$$



(a) $\alpha = 400, \beta = 0.2, \alpha_0 = 0.2, n = 2$ (b) $\alpha = 400, \beta = 0.2, \alpha_0 = 2, n = 2$

Figure 6.4: Sample runs of the system (6.1) for different sets of parameters. The initial values were $m_1 = 0.1, m_2 = 0.3, m_3 = 0.2, p_1 = 0.2, p_2 = 0.1, p_3 = 0.3$. Only the values of m_1 are depicted.

It ensures that the values reached in each period are not decreasing. By connecting formulae (6.2) and (6.3) we get the formula $\varphi \wedge \psi$. It does express the desired oscillatory behaviour in Figure 6.4a, but is not satisfied by signals of the type depicted in Figure 6.4b.

Another property of the repressilator can be expressed by the formula:

$$\mathbf{G}_{[0,270]} * \big[(\mathbf{F}_{[0,30]} \, (m_1^* + 1 > m_3 \wedge m_1^* - 1 < m_3) \big], \tag{6.4}$$

which means that *"the values of m_1 precede the values of m_3 in such sense that for every value of m_1, m_3 reaches similar value in short time after m_1 did"* (Figure 6.5).

Full estimation of parameters could not be performed due to high time demands of the implemented monitoring algorithm (Section 5.2). Source files with scripts for the computations presented in this chapter are on attached CD, the figures depicting satisfaction sets during the computations can be found in Appendix A.2.

Different task would be to ensure that the behaviour of concentrations of the fluorescent protein in the bacteria (Figure 6.2) is in corre-

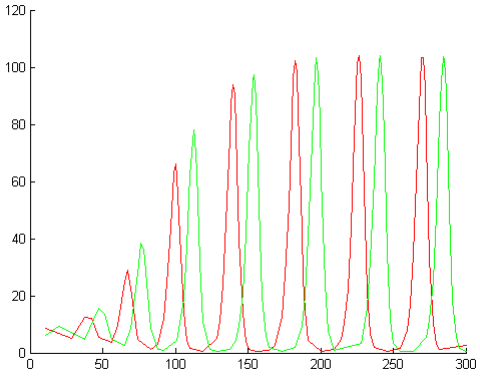


Figure 6.5: Sample run of the system (6.1) satisfying the formula (6.4). The parameters were $\alpha = 400, \beta = 0.2, \alpha_0 = 0.2, n = 2$ and the initial values $m_1 = 0.1, m_2 = 0.3, m_3 = 0.2, p_1 = 0.2, p_2 = 0.1, p_3 = 0.3$. Depicted variables are m_1 – red and m_3 – green.

spondence with the model. While it could be done only manually in the paper [14], if we had the data, we would be able to test the properties of the measurements identically as the properties of the signals produced by ODEs.

Chapter 7

Conclusion

This thesis has concerned systems evincing oscillatory behaviour and their description by temporal logics. I have formulated demands on temporal logic to be able to express and analyze such nontrivial properties. Following that, I have examined existing logics and introduced more succinct and expressive temporal logic for reasoning about continuous signals. This logic is able to capture and distinguish more classes of oscillatory behaviour.

As a part of this thesis, I have also described and implemented a polynomial monitoring algorithm for this new logic. This algorithm determines satisfaction of any formula, but only when interpreted over piecewise linear signal. It is, however, a reasonable limitation, because practical signals occur mostly in this form. Second disadvantage of the implemented algorithm is long computation time. For the practical applications, it will be necessary to improve the performance of the functions for polygonal operations.

The introduced logic can be used not only for checking whether one formula is satisfied over one signal, but also in the process of machine learning to infer rules or estimate parameters of unknown systems [5, 29]. A different utilization of the logic can be found in the area of knowledge representation. Formulae can succinctly represent whole classes of behaviour corresponding to various systems.

One straightforward direction of future development, besides improving computation efficiency, is introduction of a measure of robustness [13]. Quantification of the degree of satisfaction of a formula enables us to use better optimization techniques in parameter estimation or to identify possible weak points of the designed model.

Bibliography

[1] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J ACM*, 43(1):116–146, 1996.

[2] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Proc. of the Real-Time: Theory in Practice, REX Workshop*, pages 74–106, London, UK, UK, 1992. Springer-Verlag.

[3] R. Alur and T. A. Henzinger. A really temporal logic. *J ACM*, 41(1):181–203, 1994.

[4] P. Bellini, R. Mattolini, and P. Nesi. Temporal logics for real-time system specification. *ACM Comput Surv*, 32(1):12–42, 2000.

[5] L. Calzone, N. Chabrier-rivier, F. Fages, and S. Soliman. Machine learning biochemical networks from temporal logic properties. *Trans Comput Syst Biol*, 4220:68–94, 2006.

[6] J. Cerveny and L. Nedbal. Metabolic rhythms of the cyanobacterium cyanothece sp. atcc 51142 correlate with modeled dynamics of circadian clock. *J Biol Rhythms*, 24(4):295–303, 2009.

[7] C. Chicone. *Ordinary Differential Equations with Applications*. Springer, USA, 2 edition, 2006.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM T Progr Lang Sys*, 8:244–263, 1986.

[9] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[10] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 3 edition, 2008.

[11] G. de Vries, T. Hillen, M. Lewis, J. Müller, and B. Schönfisch. *A Course in Mathematical Biology: Quantitative Modeling with Mathematical and Computational Methods*. SIAM, Philadelphia, 2006.

[12] S. Demri, D. D'Souza, and R. Gascon. Temporal logics of repeating values. *J Logic Comput*, 2011.

[13] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS'10*, pages 92–106, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[15] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J ACM*, 33(1):151–178, 1986.

[16] J. K. Hale. *Ordinary Differential Equations*. Wiley-Interscience, New York, 1969.

[17] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 402 –413, 1990.

[18] P. Hartman. *Ordinary Differential Equations*. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 2002.

[19] T. Henzinger. Half-order modal logic: how to prove real-time properties. In *Proc. of the 9th ACM sym. on PODC*, PODC '90, pages 281–296, New York, NY, USA, 1990. ACM.

[20] B. Hess. Periodic patterns in biology. *Naturwissenschaften*, 87:199–211, 2000.

[21] B. N. Kholodenko. Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. *Eur J Biochem*, 267(6):1583–1588, 2000.

[22] M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT). http://control.ee.ethz.ch/~mpt/about.php#geometry, 2004.

[23] P. S. Landa. Universality of oscillation theory laws. types and role of mathematical models. *Discrete Dyn Nat Soc*, 1(2):99–110, 1997.

[24] A. J. Lotka. *Elements of Physical Biology*. Williams and Wilkins, Baltimore, 1925.

[25] software MATLAB, (version 2011b). http://www.mathworks.com/products/matlab/.

[26] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proc. of FORMATS-FTRTFT*, pages 152–166. Springer, 2004.

[27] A. Pnueli. The temporal logic of programs. In *Proc. of 18th Sym. on FOCS*, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[28] A.N. Prior. *Time and Modality*. Oxford University Press, 1957.

[29] A. Rizk, G. Batt, F. Fages, and S. Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In M.Heiner and A.M.Uhrmacher, editors, *Proc. of the 6th conf. on CMSB'08*, pages 251–268. Springer, 2008.

[30] V. Volterra. In R. N. Chapman, editor, *Animal Ecology*. McGraw-Hill, 1926.

Appendix A

Appendix

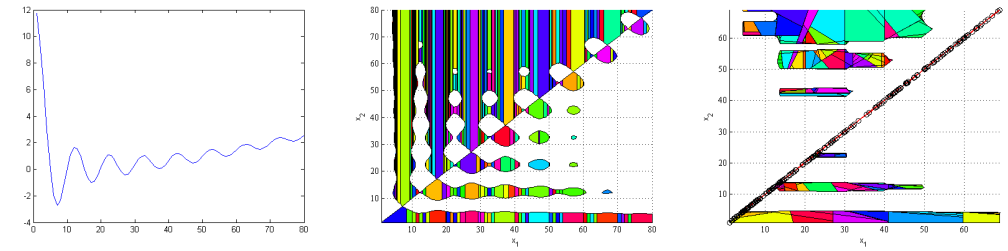
A.1 Summary of the attached CD

The source code of the prototype algorithm and scripts for computations and tests performed in this thesis are on the attached CD. All the functions and scripts contain documentation. Files on the attached CD are organized into several directories:

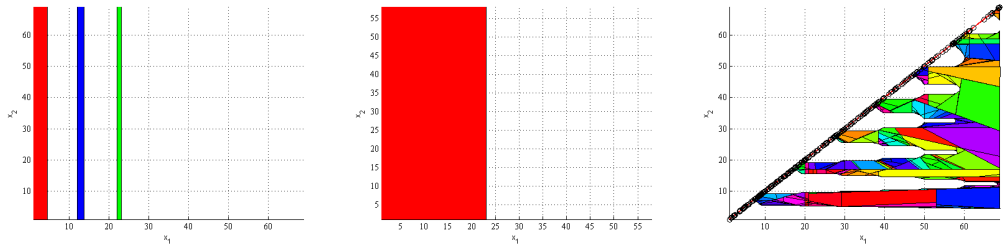
1. **main** — main functions used by user for computations
2. **auxiliary** — auxiliary functions used by main functions
3. **scripts** — scripts containing the computations of the examples and tests in this thesis

A.2 Graphical output of the computations in this thesis

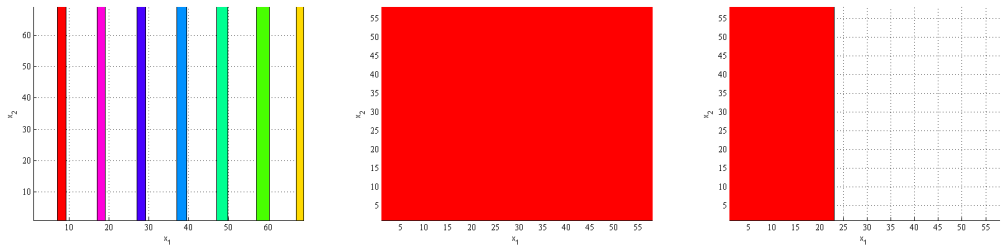
Additional graphical output of some of the computations described in this thesis is depicted here.



(a) Monitored signal. (b) Satisfaction set for $x^* \geq x$. (c) $\mathbf{G}_{[0,11]}\, x^* \geq x$.

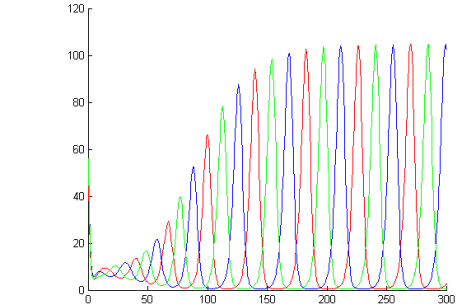


(d) $*\big[\, \mathbf{G}_{[0,11]}\, x^* \geq x \big]$. (e) $\mathbf{F}_{[0,11]} *\big[\, \mathbf{G}_{[0,11]}\, x^* \geq x \big]$. (f) $\mathbf{G}_{[0,11]}\, x^* \leq x$.

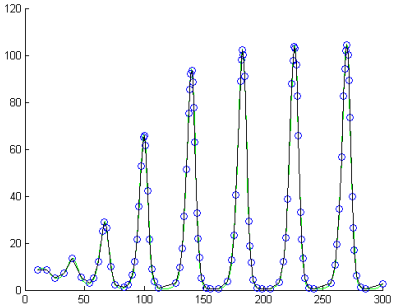


(g) $*\big[\, \mathbf{G}_{[0,11]}\, x^* \leq x \big]$. (h) $\mathbf{F}_{[0,11]} *\big[\, \mathbf{G}_{[0,11]}\, x^* \leq x \big]$. (i) Intersection of (e) and (h).

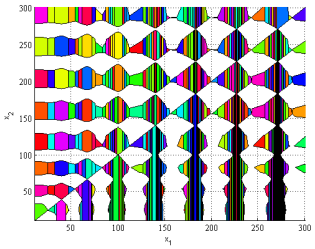
Figure A.1: Monitoring of a formula for damping oscillations (4.10) over the signal (a). The formula is not satisfied because we get an empty set after applying $\mathbf{G}_{[1,58]}$ on (i).



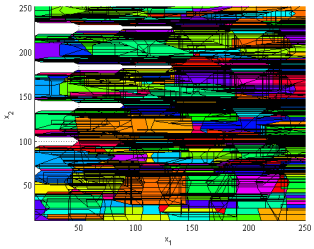
(a) Concentrations of proteins.



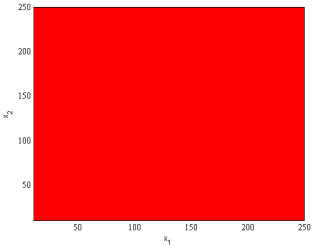
(b) Sampling of tested species.



(c) Satisfaction set for $m_1^* \leq m_1$.

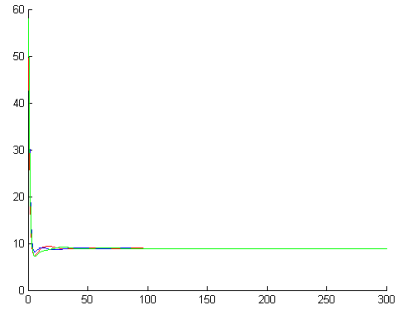


(d) $\mathbf{F}_{[1,50]}\, m_1^* \leq m_1$.

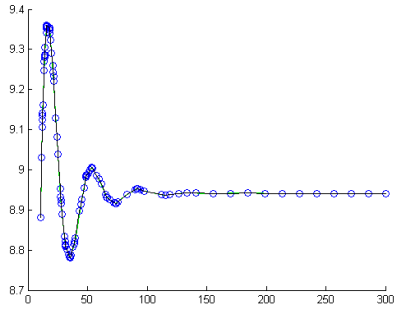


(e) $*\big[\, \mathbf{F}_{[1,50]}\, m_1^* \leq m_1 \big]$.

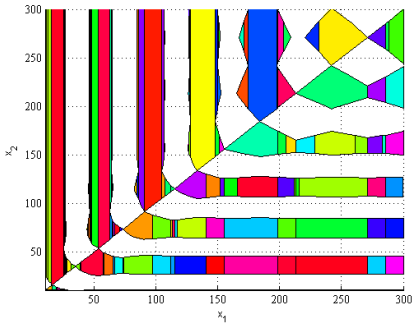
Figure A.2: Monitoring of a formula for nondecreasing oscillations (6.3) over the signal (b). The formula is satisfied.



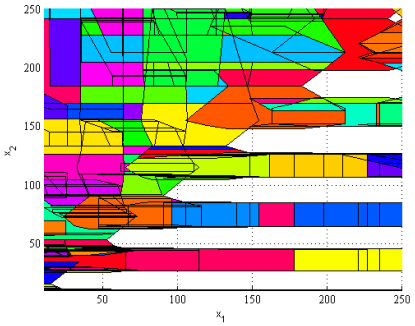
(a) Concentrations of proteins.



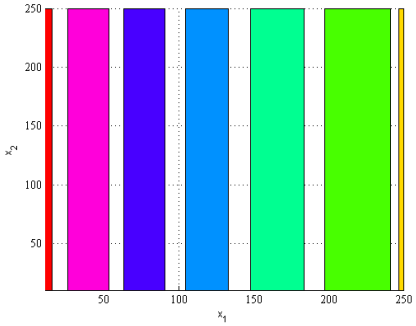
(b) Sampling of tested species.



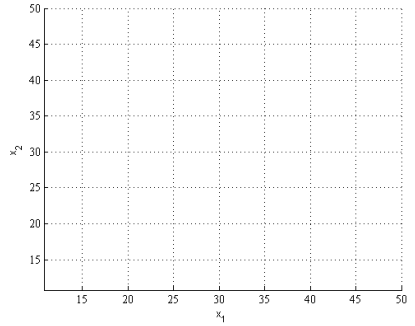
(c) Satisfaction set for $m_1^* \leq m_1$.



(d) $\mathbf{F}_{[1,50]}\, m_1^* \leq m_1$.

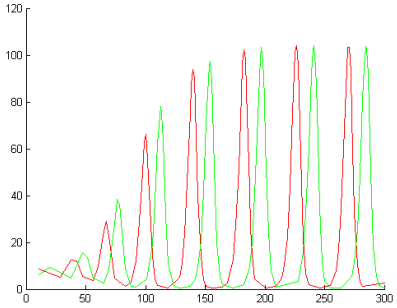


(e) $*\big[\, \mathbf{F}_{[1,50]}\, m_1^* \leq m_1 \big]$.

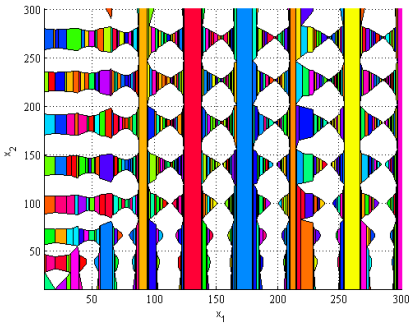


(f) $\mathbf{G}_{[10,200]} *\big[\, \mathbf{F}_{[1,50]}\, m_1^* \leq m_1 \big]$.

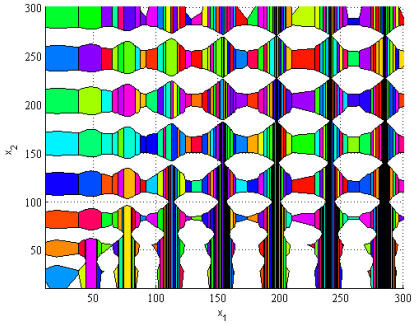
Figure A.3: Monitoring of a formula for nondecreasing oscillations (6.3) over the signal (b). The formula is not satisfied because the oscillations are damped.



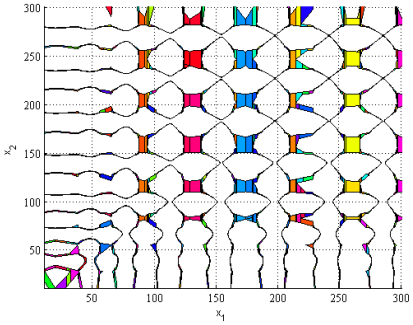
(a) Sampling of tested species.



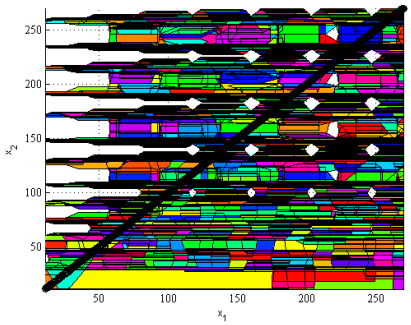
(b) Satisfaction set for $m_1^* + 1 > m_3$.



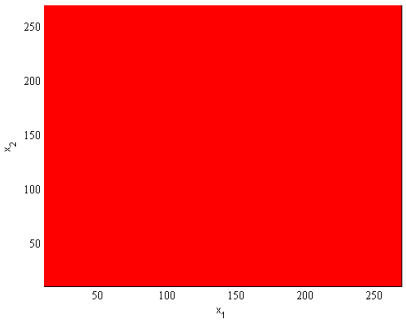
(c) Satisfaction set for $m_1^* - 1 < m_3$.



(d) Intersection of (b) and (c).



(e) $\mathbf{F}_{[0,30]}$ (d).



(f) $*\big[\,\mathbf{F}_{[0,30]}\,\big]$ (d).

Figure A.4: Monitoring of the formula (6.4) over the signal (a). The formula is satisfied.