Guaranteed control synthesis for continuous systems in Uppaal Tiga

Kim G. Larsen, Adrien Le Coënt, Marius Mikučionis, Jakob Haahr Taankvist

Guaranteed control synthesis for continuous systems in Uppaal Tiga

Kim Guldstrand Larsen[1], Adrien Le Coënt[1], Marius Mikučionis[1], and Jakob Haahr Taankvist[1]

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg Øst, Denmark
{kgl,adrien,marius,jht}@cs.aau.dk

Abstract. We present a method for synthesising control strategies for continuous dynamical systems. We use Uppaal Tiga for the synthesis in combination with a set-based Euler method for guaranteeing that the synthesis is safe. We present both a general method and a method which provides tighter bounds for monotone systems. As a case-study, we synthesize a guaranteed safe strategy for a simplified adaptive cruise control application. We show that the guaranteed strategy is only slightly more conservative than the strategy generated in the original adaptive cruise control paper which uses a discrete non guaranteed strategy. Also, we show how reinforcement learning may be used to obtain optimal sub-strategies.

Keywords: Continuous systems · Euler method · Control synthesis · Timed games.

1 Introduction

The goal of this work is to introduce a new approach for the synthesis of *correct-by-construction* control strategies for continuous-time sampled switched systems, based on the synthesis tool Uppaal Tiga [2]. Sampled switched systems constitute a sub-class of hybrid systems, and the synthesis problem for such systems is still an important issue, particularly when considering safety critical systems. The model of sampled switched systems has been used in various domains, such as power electronics [9], green housing [17, 25], automotive industry [30, 23]. The approach we develop is motivated by a cruise control application introduced in [18]. In a few words, the objective of the case-study is to compute a controller choosing the acceleration of a car (through the throttle and brake), in order to avoid hitting the car in front of it. Obviously, one does not control the front car, and such a system can easily be modelled as a two-player game. Furthermore, an accurate modelling of the dynamics of the cars should be done with differential equations. Our main goal is to ensure safety properties for the controlled system, *e.g.*, that the distance between the cars stays above a given limit. The system can actually be formulated as a continuous-time switched system, however, the difficulty comes from uncontrollable components, which prevents us from using standard switched control synthesis methods.

Control synthesis for switched systems has been extensively studied in the past years. One of the current major approaches is symbolic methods, which basically aim at representing the continuous and infinite state-space of the system with a finite number of symbols, *e.g.* discrete points [11, 29], sets of states [21], etc. This type of approaches is particularly adapted for safety critical systems, since it exhaustively ensures that an interest set is safe. However, dealing with uncontrollable components in this setting is particularly difficult. One could cite robust approaches [12], distributed computations [22], or contract-based design [31], but they usually consider the

adversary as a bounded perturbation. In the case of the cruise control case-study, considering the front car as a bounded perturbation would be overly conservative, the reachability computations would lead to extremely pessimistic sets, and thus inapplicable controllers. Moreover, the state-space to be considered in the cruise control example is too large to be handled by most symbolic methods, using a discretisation or tiling based approach would be computationally infeasible.

The model of timed automata and games is particularly well suited for modelling the behaviour of the car. The tool UPPAAL TIGA allows to compute strategies for such systems, but for computability reasons, requires integers in the guards of the models. Thereby, the dynamics of the system should be described using only integers. A naive way of doing so is to discretise the system, for example with a numerical scheme, properly scaled or approximated so that the discrete approximation is described with integers. This is the approach which was used in [18]. The problem of this approach is that the properties are guaranteed only at discrete times, which compromises the safety of the controlled system. This is illustrated in Figure 1, in which the true continuous distance can be compared to the integer approximation used in [18]. We clearly observe that the continuous distance between the cars goes below the integer approximation between two time steps. The safety property is not guaranteed in this case, since the cars could hit each other between two time steps.

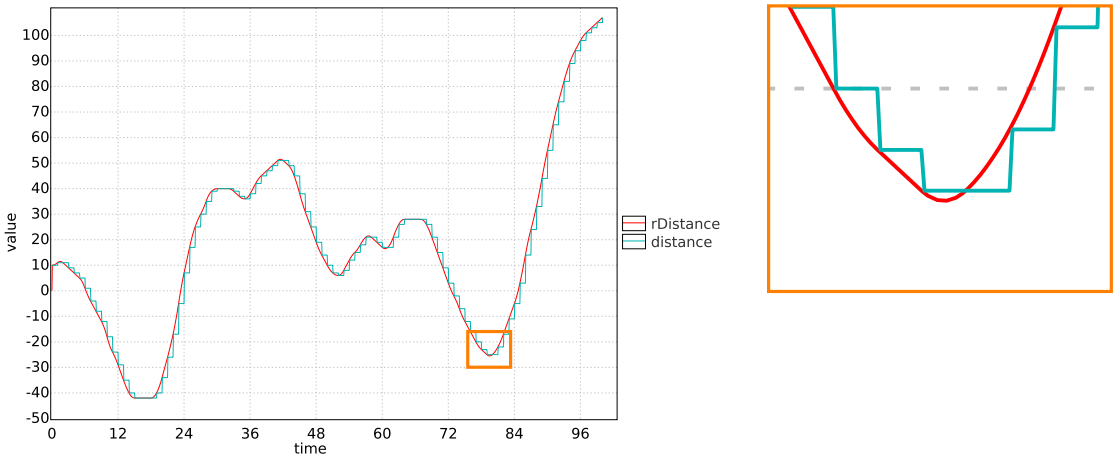


Fig. 1: The problem of using a discrete approximation for the dynamics as in [18]: the continuous distance (`rDistance`) can go below the integer approximation (`distance`) between the time steps.

We present an approach based on the synthesis tool UPPAAL TIGA, which allows to compute guaranteed safe strategies for timed game models of continuous dynamical systems. By merging safe reachability computations based on guaranteed numerical schemes, with state-of-the-art verification of timed automata and games, we extend the field of application of both continuous reachability computations, and timed game verification tools. The guaranteed solution of ordinary differential equations have been widely studied in the last decades. The main approaches rely on interval arithmetic, in the framework of Taylor series in [26, 27], and Runge-Kutta schemes in [4, 10, 3]. In this paper, we choose to use a guaranteed Euler method, introduced in [20], because of its

simplicity of implementation which allows its use directly in UPPAAL TIGA, without using any external libraries.

The principle of our approach is the following: instead of using a standard scheme or discretisation for computing successor states in the timed game model, we use a guaranteed set-based Euler scheme. Since we use a set-based approach, we need to use lower and upper over-approximations, this allows us to keep the continuous state in a convex set. The set-based approach is implemented in functions which take integers as inputs, and return integers, thus ensuring that we keep the decidability results of the verification tool. Since set-based methods usually lead to growing sets within time, we develop a refinement method for monotone systems in order to keep a tight approximation during long simulations. We then demonstrate the usability of the method by further exploring the safe strategies computed. In our case, we use a learning method implemented in the tool UPPAAL STRATEGO [7] to optimise the controller while keeping the strategy safe.

The paper is divided as follows. In Section 2, we present the synthesis tool UPPAAL TIGA, its underlying limitations, and the gap to be filled in order to synthesize safety controllers for continous-time switched systems. We present a set-based reachability method for continuous systems based on the Euler method in Section 3. In Section 4, we combine the latter to functions that can be handled by the UPPAAL framework, maintaining the associated decidability results and ensuring safety for the continuous system. We propose a refinement method for monotone systems in Section 5, allowing to tighten the reachability approximations. We further exploit the safe strategies in Section 6, in which we optimise the strategy with a learning algorithm. We conclude in Section 7.

Problem statement

We are interested in continuous-time switched systems described by the set of nonlinear ordinary differential equations:

$$\dot{x} = f_j(x), \tag{1}$$

where $x \in \mathbb{R}^n$ is the state of the system, and $j \in U$ is the mode of the system. The finite set $U = \{1, \ldots, N\}$ is the set of switching modes of the system. The functions $f_j : \mathbb{R}^n \longrightarrow \mathbb{R}^n$, with $j \in U$, are the vector fields describing the dynamics of each mode j of the system. The system can be in only one mode at a time. We focus on sampled switched systems: given a sampling period $\tau > 0$, switchings will occur periodically at times τ, 2τ, ...

For $t \in [0, \tau)$, we denote by $\phi_j(t; x_0)$ the state reached by the system at time t from the initial condition x_0 at time $t_0 = 0$ and under mode $j \in U$. A controller is a function $C : \mathbb{R}^+ \longrightarrow U$, constant on the intervals $[k\tau, (k+1)\tau]$ with $k \in \mathbb{N}$, which associates to any time $t \in \mathbb{R}^+$ a switched mode $j \in U$. Given a controller C, we denote by $\phi_C(t; x_0)$ the state reached by the system at time $t \in \mathbb{R}^+$ from the initial condition x_0 and under controller C, i.e. the active mode at time $t' \leq t$ is $C(t')$. One should include the initial time $t_0 = 0$ in the notation: $\phi_j(t; t_0, x_0)$, but in the remainder of this paper, we omit it for the sake of simplicity.

We focus on synthesizing controllers for the class of system introduced, and we aim at ensuring safety properties. The safety properties we consider is defined as follows.

Definition 1. *Consider a switched system of the form* (1)*. Consider a safety set* $S = [s_1^{min}, s_1^{max}] \times \cdots \times [s_n^{min}, s_n^{max}]$ *given as a box of* $\mathbb{R}^n$*. Given a controller* C*, system* (1) *is said to be safe with respect to* S *if, for any initial condition* $x_0 \in S$ *and for all time* $t \in \mathbb{R}^+$*, we have:*

$$\phi_C(t; x_0) \in S.$$

Case study

To illustrate our approach, we consider a cruise control application introduced in [18]. Two cars *Ego* and *Front* are driving on a road shown in Figure 2. We are capable of controlling *Ego* but not *Front*. Both cars can drive a maximum of 20 m/s forward and a maximum of 10 m/s backwards. The cars have three different possible accelerations: -2m/s^2, 0 m/s^2 and 2m/s^2, between which they can switch instantly. For the cars to be safe, there should be a distance of at least 5 m between them. Any distance less than 5 m between the cars is considered unsafe. *Ego*'s sensors can detect the position of *Front* only within 200 meters. If the distance between the cars is more than 200 meters, then *Front* is considered to be *far away*. *Front* can reenter the scope of *Ego*'s sensor with arbitrary velocity as it desires, as long as the velocity is smaller or equal to that of *Ego*.

Fig. 2: Distance, velocity and acceleration between two cars [18].

In this example, the aim is to synthesize a strategy for controllable car *Ego* such that it always stays far enough from uncontrollable car *Front*. The continuous dynamics of the resulting system is as follows:

$$\dot{v}_f = a_f \tag{2}$$

$$\dot{v}_e = a_e \tag{3}$$

$$\dot{d} = v_f - v_e \tag{4}$$

where a_f and a_e can take the values -2, 0, and 2, resulting in an 8-mode switched system of the form (1). The safety set to consider is thus $S = [-10, 20] \times [-10, 20] \times [5, 200]$. We suppose that the switching period of the system is $\tau = 1$. Note that the dynamics of this system is linear, therefore, given a controller C, we can analytically compute the exact trajectory of the system.

2 From continuous switched systems to stochastic priced timed games

2.1 Modeling of the system

We model the system as a timed game in Uppaal Tiga, later, in Section 6 we annotate the model with continuous behaviour and probabilities in Uppaal Stratego [7] to be able to do performance optimisation. The system has two players, *Ego* and *Front*, with two different behaviors. We use an already existing model for these players, and we refer the reader to [18] for more information on this model. We see the two players in Figures 4 and 5. We see that these two do not do anything except when they get a synchronisation call `chooseEgo` respectively `chooseFront`. In Figure 3 we see a system component which waits one time unit and then sends the two signals. The fact that the controller only makes choices once every time unit means the system is a switched system as defined above.

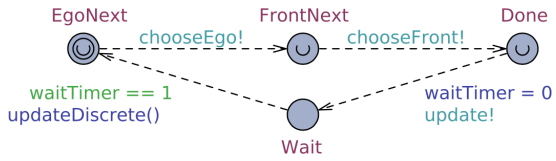


Fig. 3: Model of the system component.

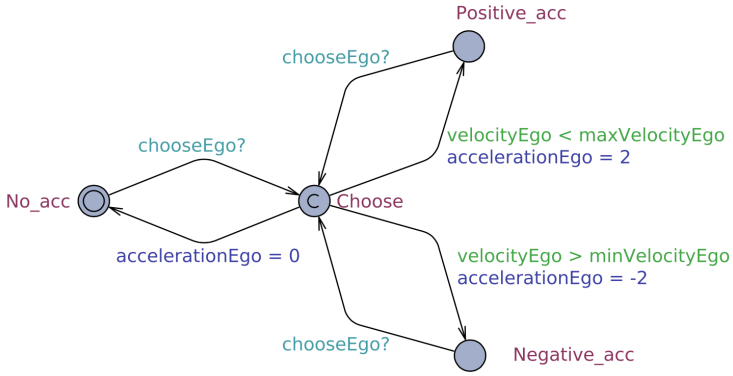


Fig. 4: Model of *Ego*.

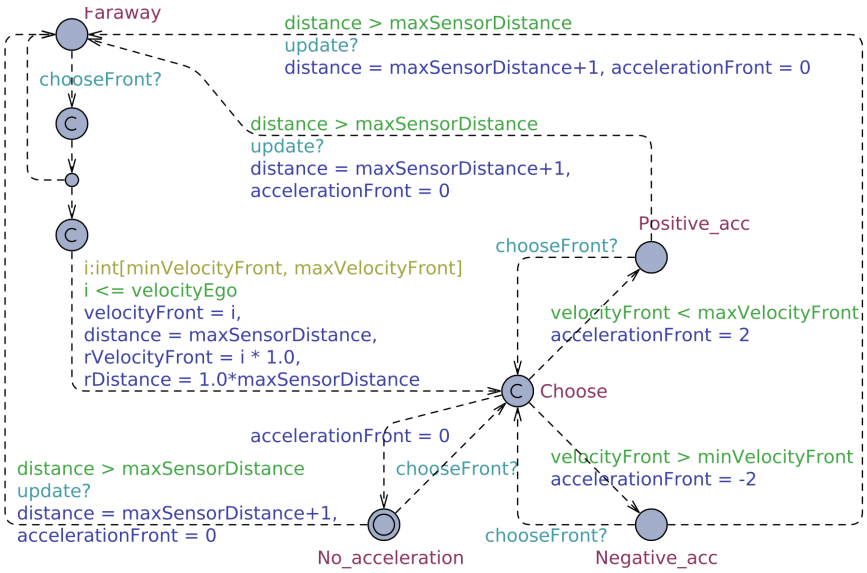


Fig. 5: Model of *Front*.

In Figure 3 we see that once every time unit we call `updateDiscrete()`. This function can be seen in Appendix A. A remaining question is the computation of function `updateDiscrete()`, which, given the current state of system (1) at time t, returns the successor state at time $t + \tau$. Using a proper scaling of the system, we can approximate system (1) by a discrete-time discrete-state system, where the state and time is described with integers. The approximate discrete system can be written

$$x(t + \tau) = F_j(x(t)), \tag{5}$$

where $x \in \mathbb{N}^n$ is the state, $t \in \mathbb{N}$ (with $\tau = 1$) and $F_j(x(t)) = x(t) + \int_0^\tau f_j(x(t))dt$.

2.2 Synthesis using Uppaal Tiga

As shown above, we model the system using only integers, and as shown in Figure 1 this is not always safe as the continuous trajectory is not taken into account between discrete points. In Section 3 we introduce a method for calculating a tube in the state space, which we can guarantee the system will stay in. In the Uppaal model we update the `updateDiscrete()` (see Appendix A) function to do the computation of this tube. We add the bounds of the tube to the state space in the form of arrays (vectors) containing the lower and upper guaranteed integer approximations. The function internally uses doubles. Generally, if real numbered variables (which are not clocks) are used, synthesis of strategies in timed games is undecidable. However, the doubles are *only* used in the calculation of the transition and the result of the function is integers[1]. This means the state space only contains integers, and as the model guarantees bounds on these integers the synthesis is possible.

As we now have guaranteed lower and upper bounds for the continuous state, we can ask Uppaal Tiga to synthesize a strategy which ensures that the guaranteed lower and upper bounds always stay in the safety set S. It is clear that this is a conservative approach and if the guaranteed bounds are not reasonably tight, it might not be possible to synthesize a strategy. We will see in Section 5 how to tighten these bounds and in Section 6 we will explore the generated strategy using Uppaal Stratego.

3 Reachability tubes using the Euler method

3.1 Lipschitz and one-sided Lipschitz condition

We make the following hypothesis:

$(H0)$ For all $j \in U$, f_j is a locally Lipschitz continuous map on S.

We recall the definition of locally Lipschitz:

Definition 2. *A function $f : A \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is* locally Lipschitz *at $x_0 \in A$ if there exist constants $\eta > 0$ and $M > 0$ such that*

$$\|x - x_0\| < \eta \rightarrow \|f(x) - f(x_0)\| \leq M\|x - x_0\|$$

[1] The method described actually calculates a box which is specified using reals, we then round these to obtain integers to make it possible to do the synthesis.

As in [13], we make the assumption that the vector field f_j is such that the solutions of the differential equation (1) are defined, e.g. by assuming that the support of the vector field f_j is compact.

We denote by T a compact overapproximation of the image by ϕ_j of S for $0 \leq t \leq \tau$ and $j \in U$, i.e. T is such that

$$T \supseteq \{\phi_j(t; x_0) \mid j \in U, 0 \leq t \leq \tau, x_0 \in S\}.$$

The existence of T is guaranteed by assumption $(H0)$ and the compactness of S. We know furthermore by $(H0)$, Definition 2 and the compactness of the support of f_j that, for all $j \in U$, there exists a constant $L_j > 0$ such that:

$$\|f_j(y) - f_j(x)\| \leq L_j \|y - x\| \quad \forall x, y \in S. \tag{6}$$

Let us define C_j for all $j \in U$:

$$C_j = \sup_{x \in S} L_j \|f_j(x)\| \quad \text{for all} \quad j \in U. \tag{7}$$

We make the additional hypothesis that the mappings f_j are *one-sided Lipschitz* (OSL) [8].

Formally:

$(H1)$ For all $j \in U$, there exists a constant $\lambda_j \in \mathbb{R}$ such that

$$\langle f_j(y) - f_j(x), y - x \rangle \leq \lambda_j \|y - x\|^2 \quad \forall x, y \in T, \tag{8}$$

where $\langle \cdot, \cdot \rangle$ denotes the scalar product of two vectors of $\mathbb{R}^n$. Constant $\lambda_j \in \mathbb{R}$ is called one-sided Lipschitz (OSL) constant, and can also be found in the literature as Dahlquist's constant [32]. Note that in practice, hypotheses H0 and H1 are not strong. Hypothesis H0 just ensures the existence of solutions for the system, and constants L_j and λ_j can always be found if the state of the system stays in a compact set (e.g. the set T).

Computation of constants λ_j, L_j and C_j In the general case, computation of constants L_j, C_j, λ_j ($j \in U$) can be realised with constrained optimisation algorithms such as the "sqp" algorithm of Octave, applied on the following optimisation problems:

- Constant L_j:

$$L_j = \max_{x,y \in S,\ x \neq y} \frac{\|f_j(y) - f_j(x)\|}{\|y - x\|}$$

- Constant C_j:

$$C_j = \max_{x \in S} L_j \|f_j(x)\|$$

- Constant λ_j:

$$\lambda_j = \max_{x,y \in T,\ x \neq y} \frac{\langle f_j(y) - f_j(x), y - x \rangle}{\|y - x\|^2}$$

We could point out that such algorithms do not guarantee that an underapproximation of the constants is computed. However, some works have been done for computing over and under approximation of Lipschitz constants in [28], and could be used here. This approach can be extended to the OSL constant. Yet, for linear systems, constants L_j, C_j, λ_j ($j \in U$) can be computed exactly, such as in [19], and we use this approach for the cruise control example.

3.2 Euler approximate solutions

Having defined OSL conditions, we now present the method introduced in [20], allowing to compute reachability sets and tubes, relying on the Euler method. The introduction of OSL conditions actually allows to establish a new global error bound, permitting the computation of overapproximation of reachability sets and tubes, precise enough to be used for control synthesis. In the remainder of this section, we consider, without loss of generality, that $t_0 = 0$, and omit its notation in the trajectory ϕ_j.

Given an initial point $\tilde{x}_0 \in S$ and a mode $j \in U$, we define the following "linear approximate solution" $\tilde{\phi}_j(t; \tilde{x}_0)$ for t on $[0, \tau]$ by:

$$\tilde{\phi}_j(t; \tilde{x}_0) = \tilde{x}_0 + t f_j(\tilde{x}_0). \tag{9}$$

Note that formula (9) is nothing else but the explicit forward Euler scheme with "time step" t. It is thus a consistent approximation of order 1 in t of the exact trajectory of (1) under the hypothesis $\tilde{x}_0 = x_0$.

We define the closed ball of center $x \in \mathbb{R}^n$ and radius $r > 0$, denoted $B(x, r)$, as the set $\{x' \in \mathbb{R}^n \mid \|x' - x\| \leq r\}$.

Given a positive real δ, we now define the expression $\delta_j(t)$ which, as we will see in Theorem 1, represents (an upper bound on) the error associated to $\tilde{\phi}_j(t; \tilde{x}_0)$ (i.e. $\|\tilde{\phi}_j(t; \tilde{x}_0) - \phi_j(t; x_0)\|$).

Definition 3. *Let us consider a switched system verifying hypotheses (H0) and (H1), associated to constants λ_j, L_j and C_j for each mode $j \in U$, such that equations* (6), (7) *and* (8) *hold. Let δ be a positive constant. We define, for all $0 \leq t \leq \tau$, function $\delta_j(\rho, t)$ as follows:*

- *if $\lambda_j < 0$:*

$$\delta_j(\rho, t) = \left(\rho^2 e^{\lambda_j t} + \frac{C_j^2}{\lambda_j^2} \left(t^2 + \frac{2t}{\lambda_j} + \frac{2}{\lambda_j^2} \left(1 - e^{\lambda_j t} \right) \right) \right)^{\frac{1}{2}}$$

- *if $\lambda_j = 0$:*

$$\delta_j(\rho, t) = \left(\rho^2 e^t + C_j^2 (-t^2 - 2t + 2(e^t - 1)) \right)^{\frac{1}{2}}$$

- *if $\lambda_j > 0$:*

$$\delta_j(\rho, t) = \left(\rho^2 e^{3\lambda_j t} + \frac{C_j^2}{3\lambda_j^2} \left(-t^2 - \frac{2t}{3\lambda_j} + \frac{2}{9\lambda_j^2} \left(e^{3\lambda_j t} - 1 \right) \right) \right)^{\frac{1}{2}}$$

Note that $\delta_j(\rho, t) = \rho$ for $t = 0$.

Theorem 1. *Given a sampled switched system satisfying (H0-H1), consider a point $\tilde{x}_0$ and a positive real ρ. We have, for all $x_0 \in B(\tilde{x}_0, \delta)$, $t \in [0, \tau]$ and $j \in U$:*

$\phi_j(t; x_0) \in B(\tilde{\phi}_j(t; \tilde{x}_0), \delta_j(\rho, t))$.

See proof in [20]

Remark 1. In Theorem 1, we have supposed that the step size h used in Euler's method was equal to the sampling period τ of the switching system. Actually, in order to have better approximations, it is sometimes convenient to consider a uniform subdivision of $[0, \tau]$ and apply the Euler's method for a time step h equal to e.g. $h = \frac{\tau}{k}$, where $k \in \mathbb{N}$ is the number of sub-steps used in the interval $[0, \tau]$.

Corollary 1. *Given a sampled switched system satisfying (H0-H1), consider a point $\tilde{x}_0 \in S$, a real $\rho > 0$ and a mode $j \in U$ such that:*

1. $B(\tilde{x}_0, \rho) \subseteq S$,
2. $B(\tilde{\phi}_j(\tau; \tilde{x}_0), \delta_j(\rho, \tau)) \subseteq S$, *and*
3. $\frac{\partial^2 \delta_j(\rho,t)}{\partial t^2} > 0$ *for all* $t \in [0, \tau]$.

Then we have, for all $x_0 \in B(\tilde{x}_0, \rho)$ and $t \in [0, \tau]$: $\phi_j(t; x_0) \in S$.

Proof. By items 1 and 2, $B(\tilde{\phi}_j(t; \tilde{x}_0), \delta_j(\rho, t)) \subseteq S$ for $t = 0$ and $t = \tau$. Since $\delta_j(\rho, \cdot)$ is convex on $[0, \tau]$ by item 3, and S is convex, we have $B(\tilde{\phi}_j(t; \tilde{x}_0), \delta_j(\rho, t)) \subseteq S$ for all $t \in [0, \tau]$. It follows from Theorem 1 that $\phi_j(t; x_0) \in B(\tilde{\phi}_j(t; \tilde{x}_0), \delta_j(\rho, t)) \subseteq S$ for all $0 \leq t \leq \tau$.

Note that condition 3 of Corollary 1 on the convexity of $\delta_j(\rho, \cdot)$ on $[0, \tau]$ can be established again using an optimisation function. Since we have an exact expression for $\delta_j(\cdot)$, its second derivative (w.r.t. time) can be computed using a computer algebra software. Using an optimisation algorithm then allows to verify that its minimum is positive. Nevertheless, we believe that the convexity of $\delta_j(\rho, \cdot)$ with respect to time could be shown analytically. For the remainder of this paper, we state this condition as a hypothesis:

$$(H2) \quad \text{For all } j \in U,\ \rho > 0, \quad \frac{\partial^2 \delta_j(\rho,t)}{\partial t^2} > 0 \text{ for all } t \in [0, \tau].$$

4 Guaranteed synthesis using the Euler method

Before stating the main result of this section, let us extend the definitions of the floor and ceiling functions to the n-dimensional setting as follows:

Definition 4. *Let $x = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$. The floor function maps a vector $x \in \mathbb{R}^n$ to a vector of $\mathbb{Z}^n$, denoted by $\lfloor x \rfloor$, with the following coefficients:*

$$\lfloor x \rfloor = (\max\{m \in \mathbb{Z} \mid m \leq x_1\}, \ldots, \max\{m \in \mathbb{Z} \mid m \leq x_n\})^\top \tag{10}$$

Similarly, the ceiling function maps a vector $x \in \mathbb{R}^n$ to a vector of $\mathbb{Z}^n$, denoted by $\lceil x \rceil$, with the following coefficients:

$$\lceil x \rceil = (\min\{m \in \mathbb{Z} \mid m \geq x_1\}, \ldots, \min\{m \in \mathbb{Z} \mid x_n \leq m\})^\top \tag{11}$$

In the following, we denote by $\mathbb{1}$ the vector of $\mathbb{R}^n$ which coefficients are all one. We also consider the ordering of $\mathbb{R}^n$ with the relation $\leq$ defined as follows, for two vectors $x = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$ and $y = (y_1, \ldots, y_n)^\top \in \mathbb{R}^n$:

$$x \leq y \quad \text{if and only if} \quad x_i \leq y_i \ \forall i = 1, \ldots, n. \tag{12}$$

We now introduce the main functions allowing to compute safe approximations of the state of the continuous system using integers:

Definition 5. *Let $j \in U$ be the active mode for the time interval $[t, t+\tau]$. Consider a given initial condition $x_0 \in S$ and initial radius $\rho_0 > 0$, let $h = \tau/k$ be the step size of the Euler method, k thus being the number of time steps used for the interval. For $i \in \{0, \ldots, k\}$, we define the sequence of radiuses of the Euler method as follows:*

$$\rho_0^j = \rho_0 \tag{13}$$

$$\rho_{i+1}^j = \delta_j(\rho_i^j, h) \tag{14}$$

Similarly, we define the sequence of points computed by the Euler method as:

$$x_0^j = x_0 \tag{15}$$

$$x_{i+1}^j = \tilde{\phi}_j(h; x_i^j) = x_i^j + h f_j(x_i^j) \tag{16}$$

We define the integer under-approximation function $H_j^k(x_0, \rho_0)$ over time time interval $[t, t+\tau]$ as follows:

$$H_j^k(x_0, \rho_0) = \min_{i \in \{0,\ldots,k+1\}} \lfloor x_i^j - \rho_i^j \mathbb{1} \rfloor \tag{17}$$

We define the low integer successor state function $h_j^k(x_0, \rho_0)$ as follows:

$$h_j^k(x_0, \rho_0) = \lfloor x_{k+1}^j - \rho_{k+1}^j \mathbb{1} \rfloor \tag{18}$$

Similarly, we define the integer over-approximation function $G_j^k(x_0, \rho_0)$ over time time interval $[t, t+\tau]$ as follows:

$$G_j^k(x_0, \rho_0) = \max_{i \in \{0,\ldots,k+1\}} \lceil x_i^j + \rho_i^j \mathbb{1} \rceil \tag{19}$$

We define the high integer successor state function $g_j^k(x_0, \rho_0)$ as follows:

$$g_j^k(x_0, \rho_0) = \lceil x_{k+1}^j + \rho_{k+1}^j \mathbb{1} \rceil \tag{20}$$

Theorem 2. *Suppose that system (1) satisfies (H0), (H1), (H2). Consider a given initial condition $x_0 \in S$ and initial radius $\rho_0 > 0$, let $h = \tau/k$ be the step size of the Euler method, k thus being the number of time steps used for the interval. Functions H and G return safe integer under and over-approximations of the continuous state of system (1) on the time interval $[t, t+\tau]$, for initial conditions in ball $B(x_0, \rho_0)$ at time t.* I.e., *for any initial condition $x \in B(x_0, \rho_0) \subseteq S$ at initial time t, for all mode $j \in U$ and any $k \in \mathbb{N}_{>0}$, we have:*

$$H_j^k(x_0, \rho_0) \in S \wedge G_j^k(x_0, \rho_0) \in S \Rightarrow \phi_j(t'; x) \in S \quad \forall t' \in [t, t+\tau]. \tag{21}$$

If $H_j^k(x_0, \rho_0) \in S \wedge G_j^k(x_0, \rho_0) \in S$ holds, we furthermore have:

$$H_j^k(x_0, \rho_0) \leq \phi_j(t'; x) \leq G_j^k(x_0, \rho_0), \quad \forall t' \in [t, t+\tau]. \tag{22}$$

Proof. The proof relies on Corollary 1. Because of the convexity with respect to time of function $\delta(\cdot, \cdot)$, it is sufficient to verify that a reachability tube stays inside safety set S only by verifying that balls $B(x_0^i, \rho_0^i)$ belong to S at each discrete instant $t + \frac{i\tau}{k}$ with $i = 0, \ldots, k+1$. Functions H and G return integer vectors which bound the Euler tubes on the whole time interval $[t, t+\tau]$, thus containing all the possible trajectories issued from $B(x_0, \rho_0)$.

Theorem 2 allows to compute under and over-approximations of the minimum and maximum integer visited by the system on a given time-interval. If it allows to ensure safety properties, one also needs an accurate way of computing the successor state. The following corollary gives a general way to compute this successor state:

Corollary 2. *Suppose that system* (1) *satisfies (H0), (H1) and (H2). Consider a given initial condition* $x_0 \in S$ *and initial radius* $\rho_0 > 0$*, let* $h = \tau/k$ *be the step size of the Euler method,* k *thus being the number of time steps used for the interval. We have, for any initial condition* $x \in B(x_0, \rho_0) \subseteq S$ *at initial time* t*, for all mode* $j \in U$ *and any* $k \in \mathbb{N}_{>0}$*:*

$$H_j^k(x_0, \rho_0) \in S \wedge G_j^k(x_0, \rho_0) \in S \Rightarrow h_j^k(x_0, \rho_0) \leq \phi_j(t + \tau; x) \leq g_j^k(x_0, \rho_0) \tag{23}$$

The main result allowing to compute safety controllers is the following:

Theorem 3. *Consider system* (1) *satisfying (H0), (H1) and (H2), and an initial condition* $x_0 \in S$ *at time* 0*.*

- *If* x_0 *is an integer, let* $x'_0 = x_0$
- *Ohterwise, let* $x'_0 = \frac{\lfloor x_0 \rfloor + \lceil x_0 \rceil}{2}$

Let $h = \tau/k$ *be the step size of the Euler method with* $k \in \mathbb{N}_{>0}$*. Given a controller* C*, let us compute iteratively two sequences of integer states as follows:*

$$y_0^{min} = \lfloor x'_0 \rfloor \quad at\ t = 0$$

$$y_0^{max} = \lceil x'_0 \rceil \quad at\ t = 0$$

and

$$y_{i+1}^{min} = h_{C(i\tau)}^k\left(\frac{y_i^{max} + y_i^{min}}{2}, \left\|\frac{y_i^{max} - y_i^{min}}{2}\right\|\right) \quad at\ t = (i+1)\tau$$

$$y_{i+1}^{max} = g_{C(i\tau)}^k\left(\frac{y_i^{max} + y_i^{min}}{2}, \left\|\frac{y_i^{max} - y_i^{min}}{2}\right\|\right) \quad at\ t = (i+1)\tau$$

If controller C *verifies, for all* $t = i\tau$ *with* $i \in \mathbb{N}$*:*

$$H_{C(t)}^k\left(\frac{y_i^{max} + y_i^{min}}{2}, \left\|\frac{y_i^{max} - y_i^{min}}{2}\right\|\right) \in S \wedge G_{C(t)}^k\left(\frac{y_i^{max} + y_i^{min}}{2}, \left\|\frac{y_i^{max} - y_i^{min}}{2}\right\|\right) \in S, \tag{24}$$

then system (1) *is safe with respect to* S*.*

Proof. The main idea of Theorem 3 is that we compute two sequences of integers which bound the continuous trajectory at discrete times, and functions H and G guarantee the correctness between discrete times. Corollary 1 guarantees that the sequences y_i^{min} and y_i^{max} do bound the continuous trajectory, since they are built as the bounds of the Euler tubes, but made wider using only integers. Theorem 2 then ensures that H and G bound the continuous trajectory between the time steps. If, for a controller C, they return values always belonging to S, we know that the controlled continuous trajectory stays inside S .

Note that functions H and G take integers as inputs (the sequences y_i^{min} and y_i^{max}), and return integers. Thus, functions H and G can be implemented in UPPAAL and used for computing guards in the time automata models, and the decidability results still hold, which ensures the termination of the computations. Due to the lack of space, we do not present how the strategy synthesis engine of UPPAAL TIGA works, and we refer the reader to [2]. In a nutshell, one has to exhaustively explore all the possible transitions, and eliminate the controllable actions leading to unsafe states. With the above theorem, we eliminate a transition as soon as (24) does not hold. Theorem 3 thus gives a general way of computing safety controllers for any system satisfying (H0)-(H1)-(H2), which are weak hypotheses. However, the accuracy of the Euler method can lack on some systems, particularly when looking far in the future, which leads us to the development of the refinement technique presented in the following section.

5 Refinement for monotone systems

The monotonicity property has been widely used in symbolic methods and control synthesis [25, 1]. The monotonicity property can be expressed using an ordering of the state, of the input, or of a perturbation. In our case, we only need to consider the state ordering, or, more precisely, that each mode of the switched system is monotone with respect to the state. The monotonicity property is then expressed as follows:

Definition 6 (Monotonicity). *System* (1) *is monotone with respect to ordering* $\leq$ *if the following implication holds for all* $j \in U$*:*

$$x_0 \leq x_0' \Rightarrow \forall t > 0, \phi_j(t; x_0) \leq \phi_j(t; x_0') \tag{25}$$

We refer the reader to [25, 24, 16] for more information on monotone control systems and applications of the monotonicity property. If system (1) is monotone, we can refine Theorem 3 by computing more accurate sequences of integer points as follows:

Corollary 3. *Consider system* (1) *satisfying (H0), (H1), (H2) and the monotonicity property. Consider an initial condition* $x_0 \in S$ *at time* 0*.*

- *If* x_0 *is an integer, let* $x_0' = x_0$
- *Ohterwise, let* $x_0' = \frac{\lfloor x_0 \rfloor + \lceil x_0 \rceil}{2}$

Let $h = \tau/k$ *be the step size of the Euler method with* $k \in \mathbb{N}_{>0}$*. Given a controller* C*, let us compute iteratively two sequences of integer states as follows:*

$$y_0^{min} = \lfloor x_0' \rfloor \quad at\ t = 0$$

$$y_0^{max} = \lceil x_0' \rceil \quad at\ t = 0$$

and

$$y_{i+1}^{min} = \lfloor \phi_{C(i\tau)}(\tau; y_i^{min}) \rfloor \quad at\ t = (i+1)\tau$$

$$y_{i+1}^{max} = \lceil \phi_{C(i\tau)}(\tau; y_i^{max}) \rceil \quad at\ t = (i+1)\tau$$

If controller C verifies, for all $t = i\tau$ with $i \in \mathbb{N}$:

$$H^k_{C(t)}(\frac{y_i^{max}+y_i^{min}}{2}, \left\|\frac{y_i^{max}-y_i^{min}}{2}\right\|) \in S \wedge G^k_{C(t)}(\frac{y_i^{max}+y_i^{min}}{2}, \left\|\frac{y_i^{max}-y_i^{min}}{2}\right\|) \in S, \qquad (26)$$

then system (1) *is safe with respect to* S.

Proof. Let us consider a controller C. We first show by induction on $i \in \mathbb{N}$ that for any initial condition $x_0 \in S$ we have: $y_i^{min} \leq \phi_C(i\tau; x_0) \leq y_i^{max}$. This is trivially true for $i = 0$ ($\lfloor x_0' \rfloor \leq x_0 \leq \lceil x_0' \rceil$). If this is true for $i \in \mathbb{N}$, then, from the monotonicity hypothesis with respect to ordering $\leq$, we have:

$$\phi_{C(i\tau)}(\tau; y_i^{min}) \leq \phi_{C(i\tau)}(\tau; \phi_C(i\tau; x_0)) \leq \phi_{C(i\tau)}(\tau; y_i^{max})$$

i.e.:

$$\lfloor \phi_{C(i\tau)}(\tau; y_i^{min}) \rfloor \leq \phi_C((i+1)\tau; x_0) \leq \lceil \phi_{C(i\tau)}(\tau; y_i^{max}) \rceil$$
$$y_{i+1}^{min} \leq \phi_C((i+1)\tau; x_0) \leq y_{i+1}^{max}$$

which proves the induction. Thus, at every discrete time $t = i\tau$, we have:

$$\phi_C(i\tau; x_0) \in B(\frac{y_i^{max}+y_i^{min}}{2}, \left\|\frac{y_i^{max}-y_i^{min}}{2}\right\|).$$

This allows us to apply Theorem 2 on each interval $[i\tau, (i+1)\tau]$: under hypotheses (H0), (H1), (H2), for any step size $h = \tau/k$ of the Euler method, if controller C verifies, for all $t = i\tau$ with $i \in \mathbb{N}$:

$$H^k_{C(t)}(\frac{y_i^{max}+y_i^{min}}{2}, \left\|\frac{y_i^{max}-y_i^{min}}{2}\right\|) \in S \wedge G^k_{C(t)}(\frac{y_i^{max}+y_i^{min}}{2}, \left\|\frac{y_i^{max}-y_i^{min}}{2}\right\|) \in S, \qquad (27)$$

then any (continuous) trajectory issued from $B(\frac{y_i^{max}+y_i^{min}}{2}, \left\|\frac{y_i^{max}-y_i^{min}}{2}\right\|)$ is safe with respect to S in the time interval $[i\tau, (i+1)\tau]$. Since $\phi_C(i\tau; x_0)$ does belong to $B(\frac{y_i^{max}+y_i^{min}}{2}, \left\|\frac{y_i^{max}-y_i^{min}}{2}\right\|)$, we conclude that $\phi_C(t; x_0) \in S$ for all $t \in [i\tau, (i+1)\tau]$, which is true for all $i \in \mathbb{N}$.

Corollary 3 uses the fact that we can use tighter integer bounds than those used in Theorem 3 when computing the integer sequences. Indeed, in Theorem 3, the sequences y_i^{min} and y_i^{max} are computed from the Euler tubes, which are quite pessimistic approximations of the trajectories. The sequences computed here do bound the trajectories of the continuous system at discrete instants because of the monotonicity property, but are this time computed from the exact solution.

Note that Corollary 3 requires the computation of the exact solution $\phi_C(\cdot;\cdot)$. If the system considered is subject to a linear dynamics, such as the cruise control example, the exact solution can be computed analytically. In other cases, an accurate numerical method can be used, such as a Runge-Kutta integration scheme with appropriate (fine) time-stepping. One could think about computing functions H and G using a similar fine-stepped numerical method, but it would require much more tests for each simulation step, or some numerically expensive optimisation methods, making it irrelevant in terms of computation times.

A simulation of a safe strategy computed with the refinement method[2] is given in Figure 6. We observe a good accuracy of the integer bounds, the induced controller is thus not overly conservative.

[2] In order to obtain a monotone system from the cruise control model, simply right (3) as $-\dot{v}_e = -a_e$ in order to have an addition in (4) which will indeed verify the monotonicity hypothesis.

It illustrates that the refinement method keeps the approximations very close to the exact solution. Note that without the refinement method, it is be possible to obtain such accurate results if the OSL constants are negative. Indeed, when an OSL constant is negative, it is possible to choose a time step small enough to make the radius of the approximations decrease. This is due to the fact that a system with negative OSL constant presents trajectories getting exponentially closer together within time, and this behaviour can be captured with a time step small enough [19].

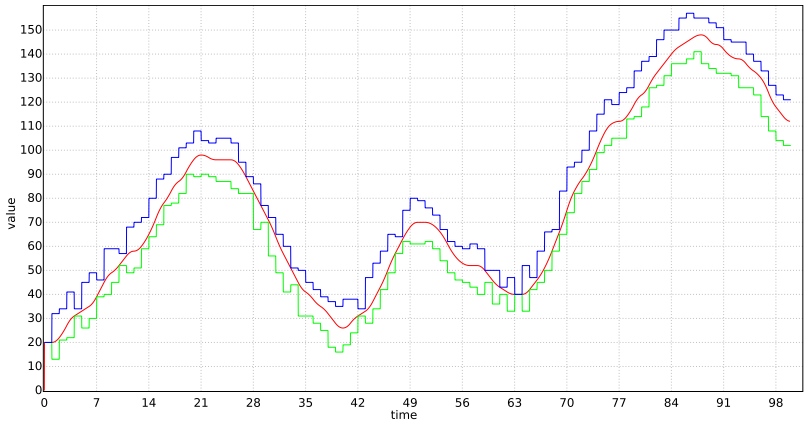


Fig. 6: Simulation of the real distance (red) and the guaranteed lower and integer distance (green and blue respectively) within time, using the refinement procedure.

6 Exploration of the Guaranteed Safe Strategy

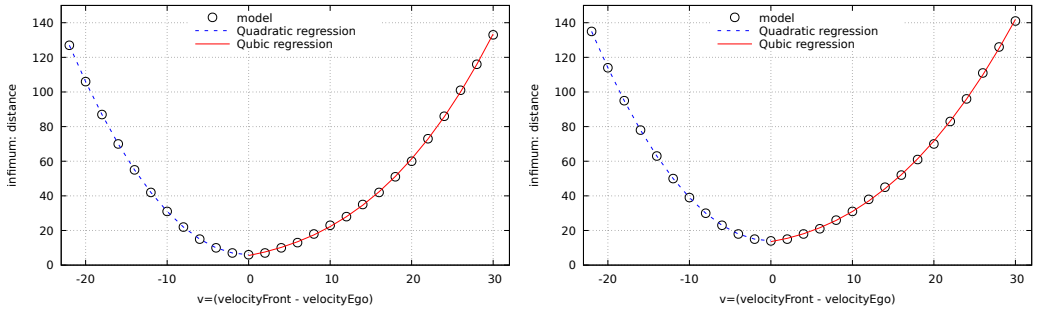
We showed above how we can compute a guaranteed safe strategy for our system. We can now use Uppaal Stratego to explore this new strategy and compare it with the discrete strategy computed in [18].

6.1 Model checking under the new guarenteed strategy

Using Uppaal's model checker [5] we can explore which distances are possible at different relative velocities between the two cars, in Figure 7 we see a plot of the different minimum possible distances of the two cars at different relative velocities, one for the strategy computed in [18] and one for the guaranteed safe strategy. We see that the two strategies are very similar, but the guaranteed safe strategy is slightly more conservative than the old non-safe strategy.

6.2 Optimisation

Uppaal Stratego also enables us to optimise the safe strategies using a learning algorithm, with the aim of minimising a given cost (*e.g.* fuel consumption, relative distance between the cars). In a given state, the safe strategy enables different (safe) transitions for *Egos*, but some of them



(a) Strategy computed in [18]. Figure from [18].

(b) Safe and guaranteed strategy.

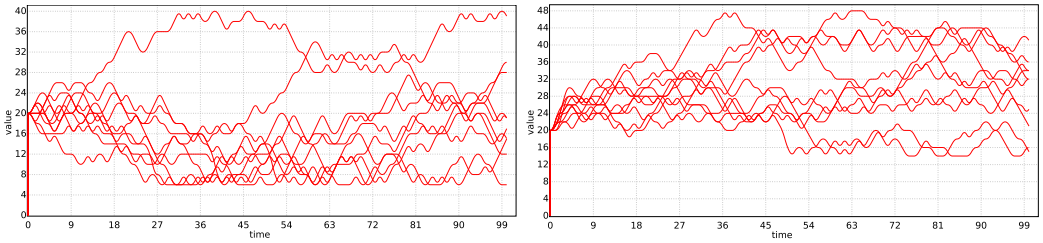
Fig. 7: Smallest achievable distance under the (guaranteed) safe strategy as a function of the relative velocity of the two cars.

might lead to a better cost. Reinforcement-learning techniques are used to create a sequence of safe strategies converging to an optimal one (minimising the cost). Despite the lack of theoretical guarantees on the speed of convergence to optimum, a near-optimal strategy is typically obtained after a few iterations. We refer the reader to [7, 6] for more information on the tool and the learning techniques used. This learning algorithm supports the use of continuous information. To use this algorithm we annotate the model with the continuous information using differential equations as seen in Figure 8.

Fig. 8: The monitor component defines a set of differential equation over real variables, these variables can then be used by the learning algorithm.

The learning algorithm also requires us to annotate the choices of *Front* with probabilities, in this case we assume that *Front* always chooses the acceleration and any other choice from a uniform distribution.

The measure we wish to optimise is the accumulated distance, we use the continuous variable `D` defined using the differential equation `D' == rDistance`, as seen Figure 8. Our goal is to minimize this measure, which means to minimize the distance between the two cars. Clearly, just minimising the distance between the cars would lead to the cars crashing, so what we do is to constrain *Egos* allowed actions to only those which are safe according to the strategy computed above. In Figure 9 we see ten simulations under respectively the optimised strategy from [18] and the optimised guaranteed safe strategy. Again, we see that the guaranteed strategy is slightly more conservative as it was not possible for the learning algorithm to optimise as well as it did in [18].



(a) Optimised strategy computed in [18]. Figure from [18].

(b) Optimised safe and guaranteed strategy.

Fig. 9: Distance over time for 10 simulations under the two different optimised strategies.

7 Conclusion

In this paper, we presented a guaranteed approach for the synthesis of control strategies for continuous-time sampled switched systems. Our approach relies on the tool Uppaal Tiga, which field of application can be extended with the use of guaranteed numerical schemes. This approach is made effective by enforcing the numerical scheme to bound the continuous trajectory using integers. We developed a refinement method for monotonic systems which allows to use a guaranteed Euler scheme even when the error grows rapidly.

We illustrated that once a safe strategy in computed, the strategy can be refined with further developments such as optimisation using learning algorithms. This showed that the safe strategy is, as expected, more conservative than the approach previously used, but does not over-constrain the system, leaving room for further optimisations. The safe strategies we compute thus constitute a sound basis for more specific computations.

We plan on testing this approach on more case studies, and implementing all the presented methods in the core of Uppaal's analytical methods, which does not currently support continuous dynamical systems. In this work, we did not explore time dependent specifications, for example with a moving safety set, or an obstacle to avoid, but this approach could handle this kind of specifications thanks to the timed game framework. This is a noticeable improvement from standard symbolic control synthesis methods which should be explored.

Another issue which could be raised is the synchronisation of the players. In a real cruise control application, we cannot guarantee that the two players choose their acceleration at the same time. We plan on developing a time robust Euler scheme, which would guarantee that if the switch does not occur simultaneously, safety is still ensured. Note that this issue has already been studied in the literature in different frameworks, an interested reader might refer to [14, 15].

Acknowledgements

This work is supported by the LASSO project financed by an ERC adv. grant; the DiCyPS project funded by Innovation Fund Denmark; and the Eurostars project Reachi.

References

1. David Angeli and Eduardo D Sontag. Monotone control systems. *IEEE Transactions on automatic control*, 48(10):1684–1698, 2003.
2. Gerd Behrmann, Agnes Cougnard, Alexandre David, Emmanuel Fleury, Kim G Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! In *International Conference on Computer Aided Verification*, pages 121–125. Springer, 2007.
3. Olivier Bouissou, Alexandre Chapoutot, and Adel Djoudi. Enclosing temporal evolution of dynamical systems using numerical methods. In *NASA Formal Methods*, number 7871 in LNCS, pages 108–123. Springer, 2013.
4. Olivier Bouissou and Matthieu Martel. GRKLib: a Guaranteed Runge Kutta Library. In *Scientific Computing, Computer Arithmetic and Validated Numerics*, 2006.
5. Alexandre David, Huixing Fang, Kim Guldstrand Larsen, and Zhengkui Zhang. Verification and performance evaluation of timed game strategies. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 100–114. Springer, 2014.
6. Alexandre David, Peter G. Jensen, Kim Guldstrand Larsen, Axel Legay, Didier Lime, Mathias Grund Sørensen, and Jakob H. Taankvist. On time with minimal expected cost! In *ATVA 2014*, 2014.
7. Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. Uppaal stratego. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 206–211, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
8. Tzanko Donchev and Elza Farkhi. Stability and euler approximation of one-sided lipschitz differential inclusions. *SIAM J. Control Optim.*, 36(2):780–796, 1998.
9. Laurent Fribourg and Romain Soulat. *Control of Switching Systems by Invariance Analysis: Applcation to Power Electronics*. John Wiley & Sons, 2013.
10. Karol Gajda, Małgorzata Jankowska, Andrzej Marciniak, and Barbara Szyszka. A survey of interval Runge–Kutta and multistep methods for solving the initial value problem. In *Parallel Processing and Applied Mathematics*, volume 4967 of *LNCS*, pages 1361–1371. Springer Berlin Heidelberg, 2008.
11. Antoine Girard. Controller synthesis for safety and reachability via approximate bisimulation. *Automatica*, 48(5):947–953, 2012.
12. Antoine Girard and Samuel Martin. Synthesis for constrained nonlinear systems using hybridization and robust controllers on simplices. *IEEE Transactions on Automatic Control*, 57(4):1046–1051, 2012.
13. Antoine Girard, Giordano Pola, and Paulo Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control*, 55(1):116–126, 2010.
14. Zohra Kader, Antoine Girard, and Adnane Saoud. Symbolic models for incrementally stable switched systems with aperiodic time sampling? 2018.
15. Kengo Kido, Sean Sedwards, and Ichiro Hasuo. Bounding errors due to switching delays in incrementally stable switched systems. *IFAC-PapersOnLine*, 51(16):247–252, 2018.
16. Eric S Kim, Murat Arcak, and Sanjit A Seshia. Symbolic control design for monotone systems with directed specifications. *Automatica*, 83:10–19, 2017.
17. Kim G Larsen, Marius Mikučionis, Marco Muniz, Jiří Srba, and Jakob Haahr Taankvist. Online and compositional learning of controllers with application to floor heating. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 244–259. Springer, 2016.
18. Kim Guldstrand Larsen, Marius Mikučionis, and Jakob Haahr Taankvist. Safe and optimal adaptive cruise control. In *Correct System Design*, pages 260–277. Springer, 2015.
19. Adrien Le Coënt. *Guaranteed control synthesis for switched space-time dynamical systems*. Theses, Université Paris Saclay, October 2017.
20. Adrien Le Coënt, Florian De Vuyst, Ludovic Chamoin, and Laurent Fribourg. Control synthesis of nonlinear sampled switched systems using euler's method. In Erika Ábrahám and Sergiy Bogomolov, editors, Proceedings 3rd International Workshop on *Symbolic and Numerical Methods for Reachability Analysis,* Uppsala, Sweden, 22nd April 2017, volume 247 of *Electronic Proceedings in Theoretical Computer Science*, pages 18–33. Open Publishing Association, 2017.

21. Adrien Le Coënt, Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Laurent Fribourg. An improved algorithm for the control synthesis of nonlinear sampled switched systems. *Formal Methods in System Design*, pages 1–21, 2017.
22. Adrien Le Coënt, Laurent Fribourg, Nicolas Markey, Florian De Vuyst, and Ludovic Chamoin. Compositional synthesis of state-dependent switching control. *Theoretical Computer Science*, 2018.
23. Hai Lin and Panos J Antsaklis. Stability and stabilizability of switched linear systems: a survey of recent results. *IEEE Transactions on Automatic control*, 54(2):308–322, 2009.
24. Pierre-Jean Meyer, Antoine Girard, and Emmanuel Witrant. Safety control with performance guarantees of cooperative systems using compositional abstractions. *IFAC-PapersOnLine*, 48(27):317–322, 2015.
25. Pierre-Jean Meyer, Antoine Girard, and Emmanuel Witrant. Robust controlled invariance for monotone systems: application to ventilation regulation in buildings. *Automatica*, 70:14–20, 2016.
26. Ramon Moore. *Interval Analysis*. Prentice Hall, 1966.
27. Nedialko S. Nedialkov, Kenneth Jackson R., and Georges F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. and Comp.*, 105(1):21 – 68, 1999.
28. János D Pintér. *Global optimization in action: continuous and Lipschitz optimization: algorithms, implementations and applications*, volume 6. Springer Science & Business Media, 2013.
29. Matthias Rungger and Majid Zamani. Scots: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 99–104. ACM, 2016.
30. Adnane Saoud, Antoine Girard, and Laurent Fribourg. Contract based design of symbolic controllers for vehicle platooning. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 277–278. ACM, 2018.
31. Adnane Saoud, Antoine Girard, and Laurent Fribourg. On the composition of discrete and continuous-time assume-guarantee contracts for invariance. 2018.
32. Gustaf Söderlind. On nonlinear difference and differential equations. *BIT Numerical Mathematics*, 24(4):667–680, 1984.

A UPPAAL functions

```
void updateDiscrete(){

    velocityEgo_gua_evol[0] = velocityEgo_gua_evol[0] + accelerationEgo;
    velocityEgo_gua_evol[1] = velocityEgo_gua_evol[1] + accelerationEgo;
    velocityFront_gua_evol[0] = velocityFront_gua_evol[0] + accelerationFront
    ;
    velocityFront_gua_evol[1] = velocityFront_gua_evol[1] + accelerationFront
    ;

    if (distance_gua_evol[0] > maxSensorDistance) {
        distance_gua_evol[0] = maxSensorDistance + 1;
        distance_gua_evol[1] = maxSensorDistance + 1;
    } else {
        eulerDiscrete();
        distance_gua_evol[0] = distance_gua_evol[0] + (velocityFront_gua_evol
    [0]-accelerationFront) - (velocityEgo_gua_evol[0]-accelerationEgo) + (
    accelerationFront - accelerationEgo)/2;
        distance_gua_evol[1] = distance_gua_evol[1] + (velocityFront_gua_evol
    [0]-accelerationFront) - (velocityEgo_gua_evol[0]-accelerationEgo) + (
    accelerationFront - accelerationEgo)/2;

        if (distance_gua_evol[1] > maxSensorDistance) {
            distance_gua_evol[1] = maxSensorDistance + 1;
        }
    }
}
```

Listing 1.1: The function updating the discrete variables.

```
void eulerDiscrete(){
    //double velEgo, velFront,
    double dist, velEgo, velFront, delta;
    double memdist_min, memdist_max, memVF_min, memVF_max, memVE_min,
    memVE_max;

    int i;

    dist = (distance_gua_evol[0]+distance_gua_evol[1])/2;
    velFront = (velocityFront_gua_evol[0]+velocityFront_gua_evol[1])/2;
    velEgo = (velocityEgo_gua_evol[0]+velocityEgo_gua_evol[1])/2;

    delta = sqrt((distance_gua_evol[1]-distance_gua_evol[0])*(
    distance_gua_evol[1]-distance_gua_evol[0])/4 + (velocityFront_gua_evol
    [1]-velocityFront_gua_evol[0])*(velocityFront_gua_evol[1]-
    velocityFront_gua_evol[0])/4 + (velocityEgo_gua_evol[1]-
    velocityEgo_gua_evol[0])*(velocityEgo_gua_evol[1]-velocityEgo_gua_evol
    [0])/4);
```

```
    memdist_min = dist - delta;
    memdist_max = dist + delta;
    memVF_min = velFront - delta;
    memVF_max = velFront + delta;
    memVE_min = velEgo - delta;
    memVE_max = velEgo + delta;

    for (i=0;i<=euler_sub_step;i++){
        dist = dist + tau*(velFront - velEgo);
        velEgo = velEgo + tau*accelerationEgo;
        velFront = velFront + tau*accelerationFront;
        delta = delta_mode(delta,find_mode(accelerationFront,accelerationEgo)
    );
        memdist_min = mini(memdist_min,dist-delta);
        memdist_max = maxi(memdist_max,dist+delta);
        memVF_min = mini(memVF_min,velFront-delta);
        memVF_max = maxi(memVF_max,velFront+delta);
        memVE_min = mini(memVE_min,velEgo-delta);
        memVE_max = maxi(memVE_max,velEgo+delta);

    }

    distance_gua_evol[0] = floor(dist-delta);
    distance_gua_evol[1] = ceil(dist+delta);
    velocityFront_gua_evol[0] = floor(velFront-delta);
    velocityFront_gua_evol[1] = ceil(velFront+delta);
    velocityEgo_gua_evol[0] = floor(velEgo-delta);
    velocityEgo_gua_evol[1] = ceil(velEgo+delta);

    distance_gua[0] = floor(memdist_min);
    distance_gua[1] = ceil(memdist_max);
    velocityFront_gua[0] = floor(memVF_min);
    velocityFront_gua[1] = ceil(memVF_max);
    velocityEgo_gua[0] = floor(memVE_min);
    velocityEgo_gua[1] = ceil(memVE_max);

}
```

Listing 1.2: The function using the Euler method and returning the lowest integer part visited by the continuous system.