Efficient On-the-fly Algorithm for Checking Alternating Timed Simulation [⋆]

Peter Bulychev[1], Thomas Chatain[2], Alexandre David[3], and Kim G. Larsen[3]

[1] Lomonosov Moscow State University, Russia
`peter.bulychev@gmail.com`
[2] LSV, ENS Cachan, France
`chatain@lsv.ens-cachan.fr`
[3] CISS, Aalborg University, Denmark
{`adavid,kgl`}`@cs.aau.dk`

Abstract In this paper we focus on property-preserving preorders between timed game automata and their application to control of partially observable systems. We define timed weak alternating simulation as a preorder between timed game automata, which preserves controllability. We define the rules of building a symbolic turn-based two-player game such that the existence of a winning strategy is equivalent to the simulation being satisfied. We also propose an on-the-fly algorithm for solving this game. This simulation checking method can be applied to the case of non-alternating or strong simulations as well. We illustrate our algorithm by a case study and report on results.

1 Introduction

Since the introduction of timed automata [3] the technology and tool support [18,8,6] for model-checking and analysis of timed automata based formalisms have reached a level mature enough for industrial applications as witnessed by a large and growing number of case studies. Most recently, efficient on-the-fly algorithms for solving reachability and safety games based on timed game automata have been put forward [9] and made available within the tool Uppaal-Tiga. The tool has been recently used in an industrial case study [17] with the company Skov A/S for synthesizing climate control programs to be used in modern pig and poultry stables. Also Uppaal-Tiga has been used for autonomous robot control [1]. Despite this success, the state-space explosion problem is a reality preventing the tools from scaling up to arbitrarily large and complex systems. We need complementary techniques allowing for the verification and analysis efforts to be carried out on suitable abstractions.

Assume that S is a timed (game) automaton and assume that ϕ is a property to be established (solved) for S. Now S may be a timed automaton too complex

[⋆] This work has been supported by the EC FP7 under grant numbers INFSO-ICT-224249 (Multiform *www.ict-multiform.eu*) and ICT-214755 (Quasimodo *www.quasimodo.aau.dk*), and the VKR Center of Excellence MT-LAB (*www.mtlab.dk*).

for our verification tool to settle the property ϕ, or S may be a timed automaton with a number of unobservable features that can not be exploited in any realizable strategy for solving the game. The goal of abstraction is to replace the complex (or unobservable) model S with an abstract timed automaton A being smaller in size, less complex and fully observable. This method requires the user not only to supply the abstraction but also to argue that the abstraction is *correct* in the sense that all relevant properties established (controllable) for A also hold (are controllable) for S; i.e. it should be established that $\mathsf{S} \geq \mathsf{A}$ for some property-preserving relationship $\geq$ between timed (game) automata. If one wants to detect errors in S (i.e to show that ϕ is not satisfied on S) rather than show its correctness, he can prove that ϕ is not satisfied on A and show that $\mathsf{A} \geq \mathsf{S}$.

The possible choices for the preorder $\geq$ obviously depend heavily on the class of properties to be preserved as well as the underlying modeling formalism. In this paper we introduce the logic ATCTL being a universal fragment of the real-time logic TCTL [2]. We introduce the notion of weak alternating timed simulation between timed game automata. This relation is proved to preserve controllability with respect to ATCTL. We reduce the problem of checking given notion of simulation to the problem of solving timed reachability turn-based game of two players. Also we provide symbolic on-the-fly algorithm for solving this game, which is a derivative of the algorithm of solving timed games proposed in [9]. It should be mentioned that in our algorithm we have exploited the fact that simulation-checking game is turn-based, i.e., in each game state only one player is permitted to make a move. This makes our algorithm simpler than the one, proposed in [9]. This algorithm can be also applied to checking non-alternating weak simulation, i.e., to the case when there are no uncontrollable transitions in timed automata.

Related work. Decidability for timed (bi)simulation between timed automata was given in [10] using a "product" region construction. This technique provided the computational basis of the tool Epsilon [11]. In [21] a zone-based algorithm for checking (weak) timed bisimulation – and hence not suffering the region-explosion in Epsilon – was proposed though never implemented in any tool.

For fully observable and deterministic abstract models timed simulation may be reduced to a reachability problem of S in the context of a suitably constructed testing automaton monitoring that the behavior exhibited is within the bounds of A [16].

Alternating temporal logics were introduced in [4] and alternating simulation between finite-state systems was introduced in [5].

The application of our method using weak alternating simulation for the problem of timed control under partial observability improves the direct method proposed in [13] to solve the same problem.

The current paper has been preceeded by the paper [12]. In that paper the simulation-checking task has been reduced to the task of reachability analysis of timed game automata, which in turn can be performed using the tool

UPPAAL-TIGA [9]. This reduction was tricky because the formalism of timed game automata is not well suited for expressing simulation-checking games. In the current paper we use another more natural formalism for defining simulation-checking games. The new simulation-checking games are turn-based (only one player owns each game state), and that makes them easier to solve. The current paper is self-contained and does not refer to [12].

Overview of the paper. In Section 2 we present the models of timed automata and timed game automata as well as the logic ATCTL. In Section 3 we define weak alternating timed simulation preorder. We prove that it preserves controllability with respect to ATCTL, and propose the algorithm of solving it.

2 Timed Games and Preliminaries

2.1 Timed Automata

Let X be a finite set of real-valued variables called clocks. We denote $\mathcal{B}(X)$ the set of constraints φ generated by the grammar: $\varphi ::= x \sim k \mid \varphi \wedge \varphi$ where $k \in \mathbb{Z}$, $x \in X$ and $\sim \in \{<, \leq, =, >, \geq\}$. A *valuation* of the variables in X is a mapping $v : X \mapsto \mathbb{R}_{\geq 0}$. We write $\vec{0}$ for the valuation that assigns 0 to each clock. For $Y \subseteq X$, we denote by $v[Y]$ the valuation assigning 0 (*resp.* $v(x)$) for any $x \in Y$ (*resp.* $x \in X \setminus Y$). We denote $v + \delta$ for $\delta \in \mathbb{R}_{\geq 0}$ the valuation *s.t.* for all $x \in X$, $(v + \delta)(x) = v(x) + \delta$. For $g \in \mathcal{B}(X)$ and $v \in \mathbb{R}^X_{\geq 0}$, we write $v \models g$ if v satisfies g. We denote $[\![g]\!] = \{v \in \mathbb{R}^X_{\geq 0} \mid v \models g\}$.

Definition 1 (Timed Automaton [3]). *A* Timed Automaton *(TA) is a tuple* $A = (L, l_0, \Sigma, X, E, Inv)$ *where* L *is a finite set of* locations*,* $l_0 \in L$ *is the* initial location*,* Σ *is the set of* actions*,* X *is a finite set of* real-valued clocks*,* $Inv : L \to \mathcal{B}(X)$ *associates to each location its* invariant *and* $E \subseteq L \times \mathcal{B}(X) \times \Sigma \times 2^X \times L$ *is a finite set of* transitions*, where* $t = (l, g, a, R, l') \in E$ *represents a transition from the location* l *to* l'*, labeled by* a*, with the guard* g*, that resets the clocks in* R*. One special label* τ *is used to code the fact that a transition is not observable.*

A *state* of a TA is a pair $(l, v) \in L \times \mathbb{R}^X_{\geq 0}$ that consists of a discrete part and a valuation of the clocks. From a state $(l, v) \in L \times \mathbb{R}^X_{\geq 0}$ *s.t.* $v \models Inv(l)$, a TA can either let time progress or do a discrete transition and reach a new state. This is defined by the transition relation $\longrightarrow$ built as follows: for $a \in \Sigma$, $(l, v) \xrightarrow{a} (l', v')$ if there exists a transition $e = (l, g, a, Y, l')$ in E *s.t.* $v \models g$, $v' = v[Y]$ and $v' \models Inv(l')$, we denote the fact that e is enabled from v by $Enabled(e, v)$ and abbreviate $v' = Post_e(v)$. For $\delta \geq 0$, $(l, v) \xrightarrow{\delta} (l, v')$ if $v' = v + \delta$ and $v, v' \in [\![Inv(l)]\!]$. Thus the semantics of a TA is the labeled transition system $S_A = (Q, q_0, \longrightarrow)$ where $Q = L \times \mathbb{R}^X_{\geq 0}$, $q_0 = (l_0, \vec{0})$ and the set of labels is $\Sigma \cup \mathbb{R}_{\geq 0}$. A *run* of a timed automaton A is a sequence $(q_0, \delta_1, q_1, t_1, q'_1, \delta_2, q_2, t_2, q'_2 \dots)$ of alternating time and discrete transitions in S_A. We use $\mathsf{Runs}(q, A)$ for the set of runs that start in state q. We write $\mathsf{Runs}(A)$ for $\mathsf{Runs}((l_0, \vec{0}), A)$. If ρ is a finite run we denote $\mathsf{Duration}(\rho)$ the total elapsed time all along the run.

Let's define $q \xrightarrow{a+} q'$ iff $q \xrightarrow{a} q'$ or there exists a sequence of states $q_1, \ldots, q_n$ such, that $q \xrightarrow{a} q_1 \xrightarrow{a} \ldots \xrightarrow{a} q_n \xrightarrow{a} q'$, and define $q \xrightarrow{a*} q'$ iff $q \xrightarrow{a+} q'$ or $q = q'$. We use the superposition of transitions: $q \xrightarrow{a}\xrightarrow{b} q'$ iff $\exists q'' \cdot q \xrightarrow{a} q'' \xrightarrow{b} q'$.

2.2 Symbolic Operations on Clock Valuations

The state space of timed automaton is infinite and thus it should be handled using symbolic methods. Consider set of clocks X and a set of clock valuations $Z \subseteq R^X_{\geq 0}$. We use the following abbreviations $Z^{\nearrow} = \{v + \delta | v \in Z \wedge \delta \in \mathbb{R}_{\geq 0}\}$ and $Z^{\searrow} = \{v | \{v\}^{\nearrow} \cap Z \neq \emptyset\}$, these operations are called future and past correspondingly. Difference bounded matrixes (DBM) can be used to encode $[\![g]\!]$ for every $g \in \mathcal{B}(X)$ [7]. The set of DBMs is closed under the future ($^{\nearrow}$), past($^{\searrow}$), intersection ($\bigcap$) union ($\bigcup$) and complementary ($\neg$) operations.

We use symbolic extension of $Post_e$ function: $Post_e(Z) = \{Post_e(v) | v \in Z \wedge Enabled(e, v)\}$, also we use $Pred_e(Z) = \{v | Post_e(v) \in Z\}$. These two functions can be implemented using operations on DBMs.

2.3 ATCTL

In this article, we consider the restricted subset of universal fragment of the real-time logic TCTL [2], we call it ATCTL.

Definition 2 (ATCTL). *A formula of ATCTL is either $\mathcal{A}\ \phi_1 \mathcal{U}_t \phi_2$ or $\mathcal{A}\ \phi_1 \mathcal{W}_t\ \phi_2$, where $\mathcal{A}$ denotes the quantifier "for all path" and $\mathcal{U}_t$ (*resp. *$\mathcal{W}_t$) denotes the temporal operator "until" (*resp. *"weak until"), $t \in \mathbb{Z} \cup \{+\infty\}$ and the ϕ_i's are sets of observable actions.*

A run ρ of a timed automaton A satisfies $\phi_1\, \mathcal{U}_t\, \phi_2$ iff there exists a prefix ρ' of ρ such that: 1) all observable actions of ρ' are in ϕ_1 and 2) the last action of ρ' is in ϕ_2 and 3) $\mathsf{Duration}(\rho') \leq t$. Then we write $\rho \models \phi_1\, \mathcal{U}_t\, \phi_2$.

A run ρ of a timed automaton A satisfies $\phi_1\, \mathcal{W}_t\, \phi_2$ iff either it satisfies $\phi_1\, \mathcal{U}_t\, \phi_2$ or only actions of ϕ_1 occur in ρ. Then we write $\rho \models \phi_1\, \mathcal{W}_t\, \phi_2$. When all the runs of a timed automaton A satisfy a property ϕ, we write $A \models \mathcal{A}\ \phi$. In the following, the proposed notions of strategies and outcome are similar to the setting of asymmetric concurrent games in [14].

2.4 Timed Games

Definition 3 (Timed Game Automaton [19]). *A* Timed Game Automaton (TGA) *G is a timed automaton with its set of transitions E partitioned into* controllable *(E^c) and* uncontrollable *(E^u) transitions. We assume that a controllable transition and an uncontrollable transition never share the same observable label. In addition, invariants are restricted to $Inv : L \to \mathcal{B}'(X)$ where $\mathcal{B}'$ is the subset of $\mathcal{B}$ using constraints of the form $x \leq k$, which is needed to handle* forced *actions.*

Given a TGA G and a control property $\phi \equiv \mathcal{A}\ \phi_1\ \mathcal{U}_t\ \phi_2$ (*resp.* $\mathcal{A}\ \phi_1\ \mathcal{W}_t\ \phi_2$) of ATCTL, the *reachability (*resp. *safety) control problem* consists in finding a *strategy* f for the controller such that all the runs of G supervised by f satisfy the formula. By "the game (G, ϕ)" we refer to the control problem for G and ϕ.

The formal definition of the control problems is based on the definitions of *strategies* and *outcomes*. In any given situation, the strategies suggest to do a particular transition e after a given delay δ. A strategy [19] is described by a function that during the course of the game constantly gives information as to what the players want to do, in the form of a pair $(\delta, e) \in (\mathbb{R}_{\geq 0} \times E) \cup \{(\infty, \perp)\}$. $(\infty, \perp)$ means that the strategy wants to delay forever.

The environment has priority when choosing its actions: If the controller and the environment want to play at the same time, the environment actually plays. In addition, the environment can decide not to take action if an invariant requires to leave a state and the controller can do so.

Assumptions. We consider only runs that are infinite and contain infinitely many observable transitions. We assume that from every state, either a delay action with positive duration or a controllable action can occur.

Definition 4 (Strategies). *Let $G = (L, l_0, \Sigma, X, E, Inv)$ be a TGA. A* strategy *over G for the controller (*resp. *the environment) is a function f from the set of runs* $\mathsf{Runs}((l_0, \vec{0}), G)$ *to* $(\mathbb{R}_{\geq 0} \times E^c) \cup \{(\infty, \perp)\}$ *(*resp. $(\mathbb{R}_{\geq 0} \times E^u) \cup \{(\infty, \perp)\}$*). We denote $(\delta(\rho), e(\rho)) \overset{\text{def}}{=} f(\rho)$ and we require that for every run ρ leading to a state q,*

- *if $\delta(\rho) = 0$ then the transition $e(\rho)$ is possible from q.*
- *for all $\delta' \leq \delta(\rho)$, waiting δ' time units after ρ is possible and the augmented run $\rho' = \rho \xrightarrow{\delta'}$ (abusing notation) satisfies: $f(\rho') = (\delta(\rho) - \delta', e(\rho))$.*

Furthermore, the controller is forced to play if an invariant expires, (and, by assumption it can always play). This can be specified as follows: if no positive delay is possible from q, then the strategy of the controller satisfies $\delta(\rho) = 0$.

A strategy is called a memoryless strategy if it depends only on the last state of a run. We assume only memoryless strategies.

The restricted behavior of a TGA G when the controller plays a strategy f_c and the opponent plays a strategy f_u is defined by the notion of *outcome*.

Definition 5 (Outcome). *Let $G = (L, l_0, \Sigma, X, E, Inv)$ be a TGA and f_c,* resp. *f_u, a strategy over G for the controller,* resp. *the environment. The* outcome $\mathsf{Outcome}(q, f_c, f_u)$ *from q in G is the (possibly infinite) maximal run $\rho = (\rho_0, \ldots, \rho_i, \ldots)$ such that for every $i \in \mathbb{N}$ (or $0 \leq i < \frac{|\rho|}{2}$ for finite runs),*

- $\rho_{2i} = \min\{\delta_c(\rho_0, \ldots, \rho_{2i-1}), \delta_u(\rho_0, \ldots, \rho_{2i-1})\}$
- $\rho_{2i+1} = \begin{cases} e_u(\rho_0, \ldots, \rho_{2i}) & \text{if } \delta_u(\rho_0, \ldots, \rho_{2i}) = 0 \\ e_c(\rho_0, \ldots, \rho_{2i}) & \text{otherwise} \end{cases}$

A strategy f_c for the controller is *winning* in the game $(A, \mathcal{A}\ \phi)$ if for every f_u, $\mathsf{Outcome}(q_0, f_c, f_u)$ satisfies ϕ. We say that a formula ϕ is *controllable* in A, and we write $A \models \mathcal{A}\ \phi$, if there exists a winning strategy for the game $(A, \mathcal{A}\ \phi)$.

3 Playing Games with Timed Games

In this section we let A and B be two timed game automata. We want to find conditions that ensure that any property of ATCTL that is controllable in B is also controllable in A.

In the context of model-checking, simulation relations allow us to verify some properties of a concrete model using a more abstract version of the model, after checking the fact that the abstract model simulates the concrete one.

Here we are also considering the more general problem of controller synthesis: Some actions are controllable (the models A and B are TGA) and we want to use an abstraction of the model to build controllers for some properties of the concrete model. For this we define weak alternating simulation relation, such that if $A \geq B$, then any property of ATCTL that is controllable in B is also controllable in A.

3.1 Weak Alternating Simulation

We define weak alternating simulation relation as a relation R between the states of A and those of B such that if $(q_A, q_B) \in R$, then every property that is controllable in B from q_B is also controllable in A from q_A. Thus every controllable transition that can be taken from q_B must be matched by an equally labeled controllable transition from q_A. And on the other hand, every uncontrollable transition in A tends to make A harder to control than B; then we require that it is matched by an equally labeled uncontrollable transition in B.

Progress of time. It is necessary to check that if the controller of B is able to avoid playing any action during a given delay, then the controller of A is able to do the same. To understand why this is required, think of a control property where the goal is simply to reach a given time without playing any observable action, unless the environment plays an uncontrollable action. If the controller of B is able to wait, then it has a winning strategy for this property. So the controller of A must be able to win too.

Symmetrically, we should in principle check that if the environment of A is able to avoid playing any action during a given delay, then the environment of B is able to do the same. Actually this property does not need to be checked since, by assumption, the environments are never forced to play.

τ actions. We consider the case when there are no τ transitions in the model A. It is a natural limitation, because abstract models usually do not have any invisible behavior. Besides that, permitting the τ transitions in A complicates the definition and the computation of the corresponding simulation relation, because in this case any delay in B can be matched by a series of delays in A separated by τ transitions.

Model B can contain τ transitions. Controllable τ transitions in B don't have to be matched in A, but the state, which is reachable by controllable τ transition from q_B, should be still simulated by q_A. Additionally any uncontrollable observable transition in B is allowed to be preceded by a finite amount of uncontrollable τ transitions. The last point makes our simulation weaker (i.e. more pairs of models satisfy our definition) but still preserves the observable features.

Definition 6 (Timed Weak Alternating Simulation). *A* weak alternating simulation relation *between two TGAs $A = (L_A, l_{A0}, \Sigma \setminus \{\tau\}, X_A, E_A, Inv_A)$ and $B = (L_B, l_{B0}, \Sigma, X_B, E_B, Inv_B)$ is a relation $R \subseteq Q_A \times Q_B$ such that $({q_0}_A, {q_0}_B) \in R$ and for every $(q_A, q_B) \in R$ and for every observable action a*

- $(q_B \xrightarrow{\tau}_c q'_B) \implies ((q_A, q'_B) \in R)$ *(τ action)*
- $(q_B \xrightarrow{a}_c q'_B) \implies \exists q'_A \quad (q_A \xrightarrow{a}_c q'_A \wedge (q'_A, q'_B) \in R)$ *(controllable)*
- $(q_A \xrightarrow{a}_u q'_A) \implies \exists q'_B \quad (q_B \xrightarrow{\tau *}_u \xrightarrow{a}_u q'_B \wedge (q'_A, q'_B) \in R)$ *(uncontrollable)*
- $(q_B \xrightarrow{\delta} q'_B) \implies \exists q'_A \quad (q_A \xrightarrow{\delta} q'_A \wedge (q'_A, q'_B) \in R)$ *(delay)*

We write $A \geq B$ if there exists a weak alternating simulation relation between A and B.

Theorem 1. *If A and B are two timed games such that $A \geq B$, then for every formula $\mathcal{A}\ \phi \in ATCTL$, if $B \models \mathcal{A}\ \phi$, then $A \models \mathcal{A}\ \phi$.*

Proof (Proof Outline). We show how to build a winning strategy f^c_A for the controller in A from a winning strategy f^c_B for the controller in B using the relation R. The strategy f^c_A that we build is such that for every strategy f^u_A for the environment in A, there exists a strategy f^u_B (that we build also from f^u_A using R) such that the outcome of f^u_A and f^c_A in A matches the outcome of f^u_B and f^c_B in B (w.r.t. the observations) and one can play the two games simultaneously such that all along the plays the current state q_A in A is related by R to the current state q_B in B.

The strategy f^c_A is built by playing a fake game in B that imitates (w.r.t R) the game in A.

- When the environment of A plays a transition, play an equally labeled uncontrollable transition in the fake game B such that the states in A and B are still related by R.
- When the controller of A plays an observable transition, play an equally labeled controllable transition in the fake game B such that the states in A and B are still related by R.
- Otherwise let time elapse in B as it elapses in A.

The rest of the proof consists in showing that the strategy f^c_A is well defined, i.e. the required actions are possible. This is done by induction on the length of the finite runs of the games.

Like the case of untimed weak simulation, timed weak simulation doesn't respect the branching structure of the models [20]. Thus if we allow nested quantifiers in the considered logic ATCTL, then weak timed simulation will not preserve the satisfiability of its formulas.

In the current paper we don't study a characterization of timed alternating simulation in terms of logic, it is planned for a future work.

Theorem 1 can be used to show that all the formulas that are uncontrollable for the abstract model A are also uncontrollable for the concrete model S (by checking $\mathsf{S} \geq \mathsf{A}$). However, if we want to show that all the formulas controllable in A are also controllable in model S that contains invisible behaviour, then we can't apply theorem 1 straightforwardly. For this case we can check the simulation between the *inverted* models according to the following theorem:

Theorem 2. *If TGAs A' and B' are obtained from TGAs A and B by replacing controllable transitions by uncontrollable transitions and vice versa (i.e. $E^c_{B'} = E^u_B$, $E^u_{B'} = E^c_B$, $E^c_{A'} = E^u_A$, $E^u_{A'} = E^c_A$) and $B' \geq A'$, then for every formula $\mathcal{A}\ \phi \in ATCTL$, if $B \models \mathcal{A}\ \phi$, then $A \models \mathcal{A}\ \phi$.*

3.2 Simulation Checking Game

In this section we consider the task of checking weak alternating simulation between TGAs $A = (L_A, l_{A0}, \Sigma \setminus \{\tau\}, X_A, E_A, Inv_A)$ and $B = (L_B, l_{B0}, \Sigma, X_B, E_B, Inv_B)$, i.e. $A \geq B$. We assume that $L_A \cap L_B = \emptyset$ and $X_A \cap X_B = \emptyset$.

There is an uncontrollable τ loop in TGA M iff there is a reachable state $q \in Q_M$ such that $q \xrightarrow{\tau+}_u q$. We assume that there are no uncontrollable τ loops in B.

We use one of the well known methods of checking simulation relations. The method reduces the simulation checking task to the task of solving a two-players game [15]. In this game one player, *Spoiler*, tries to put the models in an inconsistent state by taking controllable transitions in B and uncontrollable in A. The other player, *Duplicator*, tries to prevent *Spoiler* from doing that by repeating *Spoiler*'s transition in the opposite model. The simulation checking game is infinite for our case because the state spaces of the original models are infinite. Therefore we solve our game using symbolic methods.

Definition of Simulation Checking Game. The simulation checking game is turn-based, which means that in each game state only one player is allowed to make a move. The game states are represented by the tuples (l_A, l_B, Z, a), where $l_A \in L_A$, $l_B \in L_B$, $a \in \{\bot\} \cup \Sigma \setminus \{\tau\}$ and $Z \subseteq R_{\geq 0}^{X_A \cup X_B}$. All the game states (l_A, l_B, Z, a) such that $a = \bot$ are states of the player *Spoiler* and we use the shorthand $(l_A, l_B, Z)_S$ for identifying them. All other states belong to player *Duplicator* and we'll use the shorthand $(l_A, l_B, Z, a)_D$ for them. We use the function *Type*() which given a game state as an input returns its owner. We use an abbrevation $Z(S)$ for the third component of a game state S.

For two game states S_1 and S_2 we write $S_1 \rightarrow S_2$ if there is a game transition from S_1 to S_2. The game transition relation is constructed as follows:

- $(l_A, l_B, Z)_S \rightarrow (l_A, l'_B, Z')_S$ iff $e = (l_B, g, \tau, Y, l'_B) \in E^c_B$ and $Z' = Post_e(Z)^{\nearrow} \cap [\![Inv_B(l'_B)]\!]$

- $(l_A, l_B, Z)_S \to (l_A, l'_B, Z', a)_D$ iff $e = (l_B, g, a, Y, l'_B) \in E^c_B$, $a \neq \tau$ and $Z' = Post_e(Z)$
- $(l_A, l_B, Z)_S \to (l'_A, l_B, Z', a)_D$ iff $e = (l_A, g, a, Y, l'_A) \in E^u_A$ and $Z' = Post_e(Z)$
- $(l_A, l_B, Z, a)_D \to (l'_A, l_B, Z')_S$ iff $e = (l_A, g, a, Y, l'_A) \in E^c_A$, $Z' = Post_e(Z)^{\nearrow} \cap [\![Inv_B(l_B)]\!]$
- $(l_A, l_B, Z, a)_D \to (l_A, l'_B, Z')_S$ iff $e = (l_B, g, a, Y, l'_B) \in E^u_B$, $Z' = Post_e(Z)^{\nearrow} \cap [\![Inv_B(l'_B)]\!]$
- $(l_A, l_B, Z, a)_D \to (l_A, l'_B, Z', a)_D$ iff $e = (l_B, g, \tau, Y, l'_B) \in E^u_B$, $Z' = Post_e(Z)$

Remind that observable controllable and uncontrollable transitions of one model never share the same label, thus controllable transition in B can't be matched by uncontrollable transition in B and uncontrollable transition in A can't be matched by controllable transition in A.

Consider the initial game state $S_0 = (l_{A0}, l_{B0}, \{\vec{0}\}^{\nearrow} \cap [\![Inv_B(l_{B0})]\!])_S$. The set of all game states, which are reachable from S_0 by the game transition relation, is finite and can be built by forward exploration.

For each game transition $\alpha = (S' \to S'')$ we store the transition e of one of the models, which has been used in it. It allows us to define function $Pred_\alpha : 2^{Z(S'')} \to 2^{Z(S')}$ (*resp.* $Post_\alpha : 2^{Z(S')} \to 2^{Z(S'')}$), such that it returns the set of all the predecessors (*resp.* successors) of Z''. For instance if $\alpha = (S' \to S'')$ is a transition of the first type, in which transition e of the model B has been used, and $Z'' \subseteq Z(S'')$, then $Pred_\alpha(Z'') = Z(S') \cap \{(s_A, s_B) | (s_A, s'_B) \in Z'' \wedge s_B \in Pred_e(s'_B)\}$.

The fourth (*delay*) requirement of the simulation definition is not presented in the game rules, because this requirement can be checked by player *Spoiler* alone and thus he can modify the set of concrete winning states without interacting with *Duplicator*.

As it has been mentioned before model B is not allowed to have uncontrollable τ loops. It makes sense for the transitions of the last type because otherwise *Duplicator* could move along such a loop forever and thus win even if there is no simulation. Conservative static analysis can be used to detect τ loops before running simulation checking algorithm.

Simulation-checking games for finite-state automata are finite and thus can be solved by back-propagating the set of game states which are known to be winning for *Spoiler* [15]. In our case each game state S includes a possibly infinite set of clock valuations $Z(S)$, some of them are winning for *Spoiler*. We use function $Win(S) \subseteq Z(S)$ to store them.

Definition 7. *Let us say that function Win defines the set of winning states of the player Spoiler, if the following requirements are fulfilled:*

- *if* $Type(S) = Spoiler$ *and* $S = (l_A, l_B, Z)_S$, *then*
  $Win(S) = Z(S) \cap \Big(Z(S) \cap \big(\neg[\![Inv_A(l_A)]\!] \cup \bigcup_{\alpha = S \to S'} Pred_\alpha(Win(S'))\big)\Big)^{\searrow}$,
- *if* $Type(S) = Duplicator$, *then*
  $Win(S) = Z(S) \setminus \bigcup_{\alpha = S \to S'} Pred_\alpha(Z(S') \setminus Win(S'))$,

– *Win is the least such function according to the preorder $f \leq g \equiv \forall S(f(S) \subseteq g(S))$.*

The first point of this definition stands for the fact that *Spoiler* wins in a state of the concrete game if the invariant of A is violated or if he can delay and move to some other winning state. The second point means that *Duplicator* loses in a state of concrete game if he can't move to some other game state, which is winning for him.

Theorem 3. *For every two models A and B there is one and only one function Win, such that it satisfies definition 7.*

Theorem 4. *Let Win be a function which satisfies definition 7. Then A simulates B with respect to weak alternating simulation iff $\vec{0} \notin Win(S_0)$.*

Solving Simulation Checking Game Our goal is to build the function Win which defines the set of winning states for *Spoiler*, and then checks whether *Spoiler* wins in the initial state. We can do it by firstly building game graph in the forward manner and than back-propagating the values of the Win function. If we use this approach, we have to build the entire game graph even if there is no simulation and it can be proved after a few steps. To avoid this we develop an efficient on-the-fly algorithm based on the algorithm for solving timed games [9]. This algorithm combines forward exploration of a state-space and backward propagation of a winning set. Being on-the-fly, the symbolic algorithm may terminate before having explored the entire state-space, i.e. as soon as a winning strategy has been identified.

The symbolic on-the-fly algorithm for computing simulation-checking game is given in Figure 1. The algorithm is based on a waiting-list, $Waiting$, of edges in the game graph to be explored, and a passed-list, $Passed$, containing all the symbolic states of the simulation-graph encountered so far by the algorithm.

For each explored game state S the symbolic representation of the set of corresponding winning states is stored in $Win[S] \subseteq Z(S)$. The set $Depend[S]$ indicates the set of predecessors of S which must be reevaluated when new information about $Win[S]$ is obtained. Each time when an edge $e = (S, S')$ is considered with $S' \in Passed$ and $Win[S'] \subsetneq Z(S')$ the edge e is added to the dependency set of S' in order that possible future information about additional winning states within S' may also be back-propagated to S.

There is a major difference between the original algorithm [9] and the derivative algorithm, presented in this paper. The function $Pred_t(X, Y)$ has been used in the original algorithm for back-propagation of winning states. It returns the set of the states, from which we can reach X by time elapsing and along the path we avoid Y. The simulation-checking game is turn-based, it means that in each game state only one player can make a move. Thus the set of the states to be avoided during the back-propagation is empty and the past operator ($X^{\searrow}$) can be used instead of the $Pred_t(X, Y)$ function. The $Pred_t(X, Y)$ function is more expensive to compute than operator $X^{\searrow}$ (when Y is nonempty).

```
Initialization:
    $S_0 = (l_{A0}, l_{B0}, \vec{0}^{\nearrow} \cap \llbracket Inv_B(l_{B0}) \rrbracket)_S$
    $Win[S_0] = \llbracket Inv_B(l_{B0}) \rrbracket \cap (\llbracket Inv_B(l_{B0}) \rrbracket \cap \neg \llbracket Inv_A(l_{A0}) \rrbracket)^{\searrow}$
    $Waiting = \{(S_0, S) | S_0 \rightarrow S\}$
    $Depend[S_0] = \emptyset$
    $Passed = \emptyset$
Main:
while $((Waiting \neq \emptyset) \wedge (\vec{0} \notin Win[S_0]))$ do
    $(S, S') = pop(Waiting)$
    if $S' \notin Passed$ then
      $Passed = Passed \cup \{S'\}$
      $Depend[S'] = \{(S, S')\}$
      $(l_A, l_B, Z, a) = S'$
      if $Type(S') = Spoiler$
        $Win[S'] = Z(S') \cap (Z(S') \cap \neg \llbracket Inv_A(l_A) \rrbracket)^{\searrow}$
      else if $\{S'' | S' \rightarrow S''\} = \emptyset$ then
        $Win[S'] = Z(S')$
      else
        $Win[S'] = \emptyset$
      end if
      $Waiting = Waiting \cup \{(S', S'') | S' \rightarrow S''\}$
      if $Win[S'] \neq \emptyset$ then
        $Waiting = Waiting \cup \{(S, S')\}$
    else (* reevaluate *)
      if $Type(S) = Spoiler$ then
        $Win^* = Win[S] \cup (Z(S) \cap (\bigcup_{\alpha = S \rightarrow S''} Pred_\alpha(Win[S'']))^{\searrow})$
      else
        $Win^* = Z(S) \setminus \bigcup_{\alpha = S \rightarrow S''} Pred_\alpha(Z(S'') \setminus Win[S''])$
      end if
      if $Win^* \not\subseteq Win[S]$ then
        $Waiting = Waiting \cup Depend[S]; Win[S] = Win^*;$
      end if
      $Depend[S'] = Depend[S'] \cup \{(S, S')\}$
    end if
end while
```

Figure 1. On-The-Fly Algorithm for Solving Simulation Checking Games

Theorem 5. *The given algorithm terminates. Upon its termination there are two possible situations:*

- $\vec{0} \in Win[S_0]$ *is fulfilled and* A *doesn't simulate* B
- Win *defines the set of winning states of Spoiler in the simulation checking game*

It follows from the theorem 1 that A simulates B if and only if $\vec{0} \notin Win[S_0]$ upon termination of the given algorithm.

3.3 Handling forced transitions

If we remove the assumption that a controller can always make a transition, then it will be possible for a model to come to a state where an invariant has expired and only an environment is able to play. We consider infinite runs only and thus environment will be forced to play in this case. Theorem 1 is not valid for models with such kind of forced transitions. However this special case can be handled by complementing these uncontrollable transitions by controllable transitions which are enabled only when the invariants of their source states expire. From a controllability point of view the transformed models are equivalent to the original models. Thus it is correct to apply theorem 1 to transformed models to prove that all the ATCTL formulas that are controllable for one model with forced environment transitions are also controllable for the other model.

3.4 Handling unobservable transitions

Consider the task of checking whether $A \geq B$. In the current paper A is not allowed to have any τ transitions. This restriction seems to be reasonable because A is typically an abstraction and doesn't contain any invisible activity. If this restriction is removed, then the simulation relation is more complex to define and to compute, because in this case we have to allow to match one delay in B by series of delays and τ transitions in A. For instance, the delay for 2 time units in B could be matched in A by delay for 1 time unit, τ transition to some other state and than again by delay for 1 time unit in that state. This makes it impossible to apply the proposed algorithm and more complex to develop the appropriate one.

4 Experimental Results

The problem of timed controllability under partial observability has been studied in the paper [13]. In this task a controller has only imperfect or partial information on the state of the system. This imperfect information is given in terms of a finite number of *observations* on the state of the system. The controller can only use such observations to distinguish states and base its strategy on.

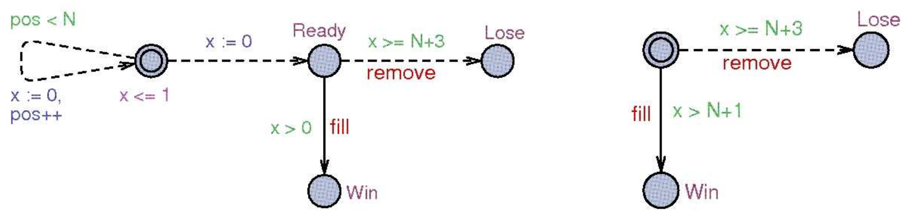

Figure 2. Concrete model of a box (left) and abstract model (right).

In paper [12] we showed that this problem can be solved by generating controller strategy for an abstract and fully observable model and proving that this abstract model simulates the original one.

We use the same model as we used in the case study in the paper [12]. A box is placed on a moving conveyor belt to reach a place where it is filled. The box has to go through a number of steps, that is a parameter N in the model. Each step takes a variable duration (0 to 1 time unit); consequently, the exact time when the box arrives in the state `Ready` is unknown. And the box might stay only $N + 3$ time units in the state `Ready`.

Thus the challenge for the controller is to fill the box while it is in the state `Ready`. This would be easy if the controller observes the progress of the box on the conveyor belt. But we assume precisely that this is not the case. Then the controller has to fill the box at a time where it is sure that the box is in the state `Ready`, however the box has progressed on the conveyor belt.

Figure 2 (left part) shows a model of the system as a timed game automaton. The loop represents the progress on the conveyor belt, incrementing the variable `pos`, which represents the position on the belt. We therefore introduce a fully observable, abstract model, shown in Figure 2 (right part). In both models all the transitions are non observable except transitions which end at the states `Win` and `Loose`, and they are marked by `fill` and `remove` actions correspondingly. We can use UPPAAL-TIGA to check this model for controllability. To guarantee that the strategy obtained from this abstract model also correctly controls our original concrete model we use proposed algorithm to establish a timed weak alternating simulation between the two models and apply theorem 2.

We have checked the simulation between the abstract and the concrete model for different values of N using the algorithm proposed in the current paper and using the algorithm proposed in the paper [12]. Table 1 shows the execution time obtained experimentally for these two algorithms. It could be seen that the new algorithm is better than the old one as it checks the simulation relation in linear time, while the timed game obtained using the old method is solved only in quadratic time. The main reason for that is that we avoid computing expensive $Pred_t$ function, as we are using turn-based simulation checking games.

proposed algorithm

N	time (in seconds)
1000	0.051
2000	0.098
3000	0.147
4000	0.196
5000	0.247
6000	0.304
7000	0.356
8000	0.411
9000	0.468
10000	0.525
15000	0.805
20000	1.085

reduction-based algorithm [12]

N	time (in seconds)
100	0.3
200	0.9
300	2.0
400	3.5
500	5.4
600	7.7
700	10.4
800	13.6
900	17.1
1000	21.1

Table 1. Experimental results

5 Conclusion

In the current paper weak alternating timed simulation has been defined which preserves controllability w.r.t. ATCTL. We have proposed the algorithm for checking this simulation by finding a winning strategy in a symbolic two-players simulation checking game.

We showed that this simulation checking game can be solved using a modification of the on-the-fly algorithm, proposed in the paper [9]. We have exploited the fact that our simulation-checking game is turn-based, i.e. in each game state only one player is permitted to make a move. It made it possible to use more effective algorithm than the original algorithm for solving arbitrary timed games.

The proposed algorithm has been implemented as a part of the tool Uppaal-Tiga. It has a user-friendly GUI and a user is allowed to play a simulation checking game against the tool to find the reason for the simulation to be fullfilled or to be not fulfilled.

We compared the algorithm proposed in the current paper with the algorithm proposed in the paper [12] and demonstrated that the new algorithm performs better than the old one.

The methods developed in this paper allow user to check for refinement of timed models, which can be useful in CEGAR-like approaches or iterative development of models.

References

1. Y. Abdeddaïm, E. Asarin, M. Gallien, F. Ingrand, C. Lessire, and M. Sighireanu. Planning Robust Temporal Plans A Comparison Between CBTP and TGA Approaches. In *Proceedings of the International Conference on Automated Planning and Scheduling International Conference on Automated Planning and Scheduling (ICAPS'07)*. AAAI, September 2007.

2. R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.
3. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. R. Alur, Th. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *FOCS*, pages 100–109, 1997.
5. R. Alur, Th. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *CONCUR*, volume 1466 of *LNCS*, pages 163–178. Springer, 1998.
6. G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. *SIGMETRICS Performance Evaluation Review*, 2005.
7. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lecture Notes in Computer Science*, pages 87–124, New York, NY, USA, 2004. Springer.
8. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In *CAV*, volume 1427 of *LNCS*, pages 546–550, 1998.
9. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, volume 3653 of *LNCS*, pages 66–80, 2005.
10. K. Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *CAV*, volume 663 of *LNCS*, pages 302–315. Springer, 1992.
11. K. Cerans, J. C. Godskesen, and K. G. Larsen. Timed modal specification – theory and tools. In *CAV*, volume 697 of *LNCS*, pages 253–267. Springer, 1993.
12. T. Chatain, A. David, and K.G. Larsen. Playing games with timed games. Research Report LSV-08-34, Laboratoire Spécification et Vérification, ENS Cachan, France, December 2008. 15 pages.
13. A. David, F. Cassez, K. G. Larsen, D. Lime, and J.-F. Raskin. Timed control with observation based and stuttering invariant strategies. In *5th International Symposium on Automated Technology for Verification and Analysis (ATVA 2007)*, Lecture Notes in Computer Science. Springer, 2007.
14. Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *CONCUR 2003 - Concurrency Theory*, volume 2761 of *LNCS*, pages 144–158, 2003.
15. K. Etessami, T. Wilke, and R.A. Schuller. Fair simulation relations, parity games, and state space reduction for buchi automata. In *SIAM Journal on Computing*, volume 34, pages 1159–1175, 2001.
16. H. E. Jensen, K. G. Larsen, and A. Skou. Scaling up Uppaal automatic verification of real-time systems using compositionality and abstraction. In *FTRTFTS*, volume 1926 of *LNCS*, pages 19–30. Springer, 2000.
17. J. J. Jessen, J. I. Rasmussen, K. G. Larsen, and A. David. Guided controller synthesis for climate controller using UPPAAL-TIGA. In *Proceedings of the 19th International Conference on Formal Modeling and Analysis of Timed Systems*, number 4763 in LNCS, pages 227–240. Springer, 2007.
18. K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer (STTT)*, 1(1-2):134–152, 1997.
19. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS*, volume 900, pages 229–242. Springer, 1995.
20. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.
21. C. Weise and D. Lenzkes. Efficient scaling-invariant checking of timed bisimulation. In *STACS*, volume 1200 of *LNCS*, pages 177–188, 1997.