Verification and Performance Evaluation of Timed Game Strategies*

Alexandre David[1], Huixing Fang[2], Kim G. Larsen[1], and Zhengkui Zhang[1]

[1] Department of Computer Science, Aalborg University, Denmark
{adavid,kgl,zhzhang}@cs.aau.dk
[2] Software Engineering Institute, East China Normal University, China
wxfang@sei.ecnu.edu.cn

Abstract. Control synthesis techniques, based on timed games, derive strategies to ensure a given control objective, e.g., time-bounded reachability. Model checking verifies correctness properties of systems. Statistical model checking can be used to analyse performance aspects of systems, e.g., energy consumption. In this work, we propose to combine these three techniques. In particular, given a strategy synthesized for a timed game and a given control objective, we want to make a deeper examination of the consequences of adopting this strategy. Firstly, we want to apply model checking to the timed game under the synthesized strategy in order to verify additional correctness properties. Secondly, we want to apply statistical model checking to evaluate various performance aspects of the synthesized strategy. For this, the underlying timed game is extended with relevant price and stochastic information. We first explain the principle of translating a strategy produced by UPPAAL-TIGA into a timed automaton, thus enabling the deeper examination. However, our main contribution is a new extension of UPPAAL that automatically synthesizes a strategy of a timed game for a given control objective, then verifies and evaluates this strategy with respect to additional properties. We demonstrate the usefulness of this new branch of UPPAAL using two case-studies.

1 Introduction

Model checking (MC) of real-time systems [12] has been researched for over 20 years. Mature tools such as UPPAAL [3] and KRONOS [5] have been applied to numerous industrial case studies. Nowadays, more interesting formal methods for real-time systems are inspired by or derived from model-checking. Two remarkable ones are controller synthesis and statistical model checking. Controller synthesis techniques [6], based on games, derive strategies to ensure some given objective while handling uncertainties of the environment. Statistical model-checking (SMC) [14], based on statistical analysis of simulations, is used to analyse reliability and performance aspects of systems, e.g., energy consumption.

* This work has been supported by Danish National Research Foundation – Center for Foundations of Cyber-Physical Systems, a Sino-Danish research center.

In the Uppaal toolbox, efficient implementations of these new techniques are found in the branches Uppaal-tiga [2] and Uppaal-smc [9].

We believe the three techniques can complement each other. Given a timed game and a control objective, controller synthesis will generate a strategy if the game is controllable. The strategy may ensure hard timing guarantees for a controller to win the game. We aim at verifying additional correctness properties by applying MC to the timed game under this strategy. Similarly, SMC should allow to infer more refined performance consequences (cost, energy consumption etc) of the synthesized strategy. For this, we extend the underlying timed game with prices and stochastic semantics.

There have been a few previous attempts to combine modelling, synthesis, verification and performance evaluation in a single paradigm. In [7] Franck et al. presented a tool chain – Uppaal-tiga for synthesis, PHAVer for verification, Simulink for simulation – to solve the energy consumption and wear control problem of an industrial oil pump case-study. In [10] Uppaal-tiga was combined with Matlab and Simulink to achieve synthesis, simulation and executable code generation for the climate controller of a pig stable. These tool chains are not integrated inside one tool and require translations to let the different tools interact.

As the first contribution in this paper, we propose the principle of translating a synthesized strategy, as obtained from Uppaal-tiga, into a controller timed automaton. One can build a closed system using the controller and do model-checking in Uppaal or statistical model-checking in Uppaal-smc. The second contribution is an extension of the semantics and algorithms of MC and SMC to use a synthesized strategy when exploring the state space (for MC) or generating random runs (for SMC). The third contribution is an implementation of this extension based on Uppaal referred here as Control-SMC, which allows users to synthesize a timed game strategy then verify and evaluate this strategy automatically. It is worth noting that Uppaal-tiga may not guarantee that the synthesized strategy is time optimal and here we are interested in evaluating a given strategy w.r.t. a number of different cost measures.

The rest of the paper is organized as follows. Section 2 defines timed games and strategies. Section 3 provides the stochastic semantics of SMC. Section 4 describes the translation of a strategy to a timed automaton. Section 5 presents the extended SMC semantics and implementation of Control-SMC. Section 6 gives the experiment results on two case-studies using Control-SMC. The paper concludes with the future work in Section 7.

2 Timed Game

This section recalls the basic theory of timed game and controller synthesis. Controller synthesis aims at solving the following problem: Given a system S and an objective ϕ, synthesize a controller C such that C can supervise S to satisfy ϕ ($C(S) \models \phi$) regardless how the environment behaves. The problem can be formulated as a two-player game between the controller and the environment.

2.1 Timed Game Automata

Let $X = \{x, y, ...\}$ be a finite set of clocks. We define $\mathcal{B}(X)$ as the set of clock constraints over X generated by grammar: $g, g_1, g_2 ::= x \bowtie n \,|\, x - y \bowtie n \,|\, g_1 \wedge g_2$, where $x, y \in X$ are clocks, $n \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$.

Definition 1. *A* Timed Automaton (TA) [1] *is a 6-tuple* $\mathcal{A} = (L, \ell_0, X, \Sigma, E, Inv)$ *where:* L *is a finite set of locations,* $\ell_0 \in L$ *is the initial location,* X *is a finite set of non-negative real-valued clocks,* Σ *is a finite set of actions,* $E \subseteq L \times \mathcal{B}(X) \times \Sigma \times 2^X \times L$ *is a finite set of edges,* $Inv : L \to \mathcal{B}(X)$ *sets an invariant for each location.*

Definition 2. *The semantics of a timed automaton* $\mathcal{A}$ *is a* Timed Transition System (TTS) $S_{\mathcal{A}} = (Q, Q_0, \Sigma, \to)$ *where:* $Q = \{(\ell, v) \,|\, (\ell, v) \in L \times \mathbb{R}^X_{\geq 0}$ and $v \models Inv(\ell)\}$ *are states,* $Q_0 = (\ell_0, 0)$ *is the initial state,* Σ *is the finite set of actions,* $\to \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ *is the transition relation defined separately for action* $a \in \Sigma$ *and delay* $d \in \mathbb{R}_{\geq 0}$ *as:*
(i) $(\ell, v) \xrightarrow{a} (\ell', v')$ *if there is an edge* $(\ell \xrightarrow{g,\alpha,r} \ell') \in E$ *such that* $v \models g$, $v' = v[r \mapsto 0]$ *and* $v' \models Inv(\ell')$,
(ii) $(\ell, v) \xrightarrow{d} (\ell', v + d)$ *such that* $v \models Inv(\ell)$ *and* $v + d \models Inv(\ell)$.

A timed game automaton is an extension of a timed automaton whose actions are partitioned into controllable actions for the controller and uncontrollable actions for the environment. Besides discrete actions, each player can decide to wait in the current location. As soon as one player decides to play one of his available actions, time will stop elapsing and the action will be taken.

Definition 3. *A* Timed Game Automaton (TGA) [13] *is a 7-tuple* $\mathcal{G} = (L, \ell_0, X, \Sigma_c, \Sigma_u, E, Inv)$ *where:* Σ_c *is the finite set of controllable actions,* Σ_u *is the finite set of uncontrollable actions,* Σ_c *and* Σ_u *are disjoint, and* $(L, \ell_0, X, \Sigma_c \cup \Sigma_u, E, Inv)$ *is a timed automaton.*

Let $S_{\mathcal{G}}$ be the timed transition system of $\mathcal{G}$. A *run* ρ of $\mathcal{G}$ can be expressed in $S_{\mathcal{G}}$ as a sequence of alternative delay and action transitions: $\rho = q_0 \xrightarrow{d_1} q'_0 \xrightarrow{a_1} q_1 \xrightarrow{d_2} q'_1 \xrightarrow{a_2} \cdots \xrightarrow{d_n} q'_{n-1} \xrightarrow{a_n} q_n \cdots$, where $a_i \in \Sigma_c \cup \Sigma_u$, $d_i \in \mathbb{R}_{\geq 0}$, q_i is state (ℓ_i, v_i), and q'_i is reached from q_i after delay d_{i+1}. $Exec_{\mathcal{G}}$ denotes the set of runs of $\mathcal{G}$ and $Exec^f_{\mathcal{G}}$ denotes the set of its finite runs.

Definition 4. *Given a timed game automaton* $\mathcal{G}$ *and a set of states* $K \subseteq L \times \mathbb{R}^X_{\geq 0}$*, the* control objective ϕ *can be: (i) a* reachability control problem *if we want* $\mathcal{G}$ *supervised by a strategy to reach* K *eventually, or (ii) a* safety control problem *if we want* $\mathcal{G}$ *supervised by a strategy to avoid* K *constantly.*

We can define a run $\rho \in Exec_{\mathcal{G}}$ as *winning* in terms of its control objective. For a reachability game, ρ is winning if $\exists k \geq 0, (\ell_k, v_k) \in K$. For a safety game, ρ is winning if $\forall k \geq 0, (\ell_k, v_k) \notin K$.

Definition 5. *A* strategy *for a controller in the timed game $\mathcal{G}$ is a mapping $s : Exec^f_{\mathcal{G}} \to \Sigma_c \cup \{\lambda\}$ satisfying the following conditions: given a finite run ρ ending in state $q = last(\rho)$, if $s(\rho) = a \in \Sigma_c$, then there must exist a transition $q \xrightarrow{a} q'$ in $S_{\mathcal{G}}$, or if $s(\rho) = \lambda$, λ being the delay action, then there must exist a positive delay $d \in \mathbb{R}_{>0}$ such that $q \xrightarrow{d} q'$ in $S_{\mathcal{G}}$.*

When a strategy only depends on the current state of the game, that is $\forall \rho, \rho' \in Exec_{\mathcal{G}}, last(\rho) = last(\rho')$ implies $s(\rho) = s(\rho')$, it is called a *positional* or *memoryless* strategy. The strategies for reachability and safety games, as the ones handled by Uppaal-tiga, are memoryless.

The analysis of TA and TGA is based on the exploration of a finite *symbolic reachability graph*, where the nodes are *symbolic states*. A symbolic state S is a pair (ℓ, Z), where $\ell \in L$, and $Z = \{v \mid v \models g_z, g_z \in \mathcal{B}(X)\}$ is a *zone* [12], which is normally efficiently represented and stored in memory as *difference bound matrices* (DBM) [4]. Uppaal-tiga uses efficient on-the-fly algorithms [6] that manipulate zones to solve timed games. The winning strategy $\hat{s}$ produced by Uppaal-tiga is also represented using zones. More precisely, for each location ℓ, $\hat{s}$ gives a finite set of pairs as $\hat{s}(\ell) = \{(Z_1, a_1), \ldots, (Z_n, a_n)\}$, where $a_i \in \Sigma_c \cup \{\lambda\}, Z_i \cap Z_j = \emptyset$ if $i \neq j$.

2.2 A Running Example

Fig. 1 [6] shows a timed game automaton named `Main` which has one clock x and two types of edges: controllable (solid) and uncontrollable (dashed). The control objective is to find a strategy that can supervise `Main` to reach `goal`, regardless of the environment's behavior. The object is expressed as `control: A<> Main.goal`. The game is controllable, and Uppaal-tiga provides a strategy as shown in Fig. 2 if running the command line version of Uppaal-tiga– `verifytga` with the option `-w0`. The strategy is a list of (zone, action) pairs indexed by locations.

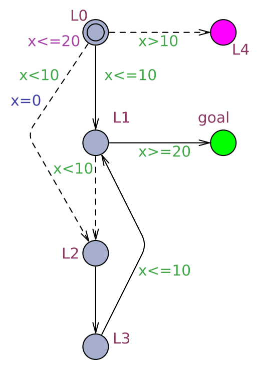


Fig. 1. TGA `Main`

```
State: ( Main.L1 )
While you are in (10<=Main.x && Main.x<20), wait.
When you are in (20<=Main.x), take transition
    Main.L1->Main.goal { x >= 20, tau, 1 }
State: ( Main.L3 )
While you are in (Main.x<10), wait.
When you are in (Main.x==10), take transition
    Main.L3->Main.L1 { x <= 10, tau, 1 }
State: ( Main.L0 )
When you are in (Main.x==10), take transition
    Main.L0->Main.L1 { x <= 10, tau, 1 }
While you are in (Main.x<10), wait.
State: ( Main.L2 )
When you are in (Main.x<=10), take transition
    Main.L2->Main.L3 { 1, tau, 1 }
State: ( Main.goal )
While you are in true, wait.
```

Fig. 2. A Strategy for `Main`

For example when `Main` is at `L1`, the action is to wait if $10 \leq x < 20$, or to take the action to reach `goal` if $x \geq 20$.

3 Stochastic Priced Timed Automata

In this section, we briefly recall the definition of priced timed automata and stochastic semantics of SMC. We borrow the definitions from [8].

3.1 Priced Timed Automata

Priced timed automata are a generalization of timed automata where clocks may have different rates in different locations. We note by $R(\ell) : X \to \mathbb{N}$ the *rate vector* assigning a rate to each clock of X at location ℓ. For $v \in \mathbb{R}^X_{\geq 0}$ and $d \in \mathbb{R}_{\geq 0}$, we write $v + R(\ell) \cdot d$ to denote the clock valuation defined by $(v + R(\ell) \cdot d)(x) = v(x) + R(\ell)(x) \cdot d$ for any $x \in X$.

Definition 6. *A* Priced Timed Automaton (PTA) *is a tuple* $\mathcal{P} = (L, \ell_0, X, \Sigma, E, R, I)$ *where: (i)* L *is a finite set of locations, (ii)* $\ell_0 \in L$ *is the initial location, (iii)* X *is a finite set of clocks, (iv)* $\Sigma = \Sigma_i \uplus \Sigma_o$ *is a finite set of actions partitioned into inputs* (Σ_i) *and outputs* (Σ_o)*, (v)* $E \subseteq L \times \mathcal{B}(X) \times \Sigma \times 2^X \times L$ *is a finite set of edges, (vi)* $R : L \to \mathbb{N}^X$ *assigns a rate vector to each location, and (vii)* $I : L \to \mathcal{B}(X)$ *assigns an invariant to each location.*

3.2 Stochastic Semantics

Consider a closed network of PTAs $\mathbf{A} = (\mathcal{P}_1 | \dots | \mathcal{P}_n)$ with a state space $St = St_1 \times \dots \times St_n$. For a concrete global state $q = (q_1, \dots, q_n) \in St$ and $a_1 a_2 \dots a_k \in \Sigma^*$ we denote by $\pi(q, a_1 a_2 \dots a_k)$ the set of all maximal runs from q with a prefix $t_1 a_1 t_2 a_2 \dots t_k a_k$ for some $t_1, \dots, t_2 \in \mathbb{R}_{\geq 0}$, that is, runs where the i'th action a_i has been output by the component $\mathcal{P}_{c(a_i)}$. We give the probability for getting such sets of runs as:

$$\mathbb{P}_{\mathbf{A}}(\pi(q, a_1 a_2 \dots a_k)) = \int_{t \geq 0} \mu^c_q(t) \cdot \Big(\prod_{j \neq c} \int_{\tau > t} \mu^j_q(\tau) d\tau \Big) \cdot \gamma^c_{q^t}(a_1) \cdot \mathbb{P}_{\mathbf{A}} \big(\pi((q^t)^{a_1}, a_2 \dots a_n) \big) dt$$

where $c = c(a_i)$ is the *index* of component taking a_1, μ^c_q is the *delay density function* for component c to choose a delay t_i at q, and $\gamma^c_{q^t}$ is the *output probability function* for component c to choose an action a_i after q is delayed by t. The above nested integral reflects that the stochastic semantics of the network is defined based on race among components. All components are independent in giving their delays which are decided by the given delay density functions. The player component who offers the minimum delay is the winner of the race, and takes the turn to make a transition and (probabilistically) choosing the action to output.

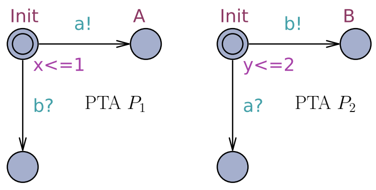


Fig. 3. A Tiny Example

Fig. 3 gives the intuition of the SMC semantics. Two PTAs P_1 and P_2 race to reach locations A or B. If P_1 enters A, it blocks P_2 to enter B, and vice versa. Furthermore, either PTA can delay uniformly within the invariants from its initial state before firing its output transition. We can use the SMC semantics to calculate the probability for P_1 to enter location A within 2 time units as:

$$\mathbb{P}(\pi(q_0, a)) = \int_{x=0}^{1} 1 \cdot \Big(\int_{y=x}^{2} \frac{1}{2} dy \Big) dx = \frac{1}{2} \int_{x=0}^{1} (2 - x) dx = \frac{3}{4}$$

where q_0 is the initial state of the network of P_1 and P_2, and the delay density functions for P_1 and P_2 at q_0 are 1 and $\frac{1}{2}$ respectively. P_1 can reach A only if it takes its transition before P_2.

4 Translating Strategies to Timed Automata

In this section, we provide a systematic way to translate a synthesized strategy of a timed game $\mathcal{G}$ produced by UPPAAL-TIGA into a controller timed automaton C. Once the controller is built, we can verify additional correctness properties or evaluate performance aspects of the closed system $C(\mathcal{G})$ in UPPAAL.

4.1 The Method

We recall from Section 2.1 that strategies have the form $\hat{s}(\ell) = \{(Z_1, a_1), \ldots, (Z_n, a_n)\}$. Given a concrete state $q = (\ell, v)$, one can lookup which action a_i to take by finding Z_i such that $v \in Z_i$. Fig. 4 illustrates how to translate the strategy from a location ℓ with the schematic zone representation (left) into a basic

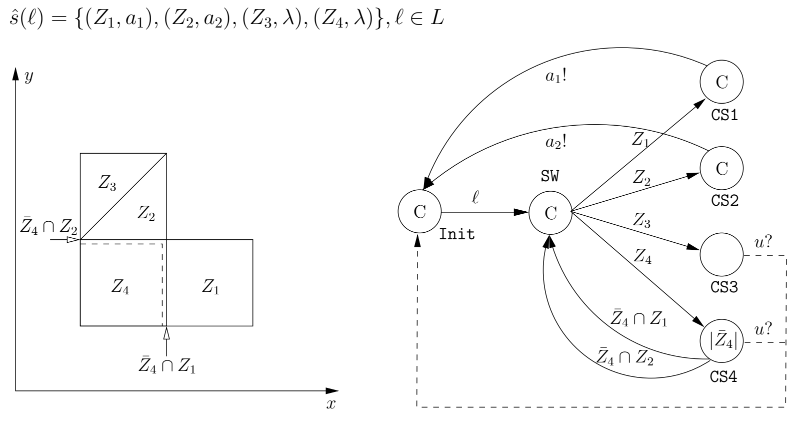


Fig. 4. Translating the Strategy

controller TA (right). The complete controller TA is obtained by repeating the same translation procedure for all locations and connecting all resulting basic controller TAs to the same initial state. The symbol "C" inside states indicates committed states. Time does not elapse in committed states, and the outgoing transitions are taken atomically. We use $\bar{Z}$ to denote the *closure* of the zone Z.

The small controller TA on the right is constructed as follows. For a given discrete state (ℓ) (location only), a transition from `Init` to a switch state `SW` is added with a guard encoding ℓ. From there we add transitions guarded by Z_i for each (Z_i, a_i) entry of $\hat{s}(\ell)$ to a choice state `CS`i. Then, we have tree basic cases: Either (1) a_i is a controllable action, (2) a_i is an unbounded delay, or (3) it is a bounded delay. In case (1), the controller takes a_i immediately with the synchronization $a_i!$ (e.g. from `CS1` and `CS2` in Fig. 4). In case (2) corresponding to $a_i = \lambda$, the controller stays idle waiting for a move from the environment with the synchronization $u?$. Finally, case (3) is similar to case (2) except for the upper bound on the delay (encoded with an invariant) and additional transitions to go back to `SW` whenever the upper bound is reached and a controllable action is enabled.

4.2 The Running Example

We translate the strategy in Fig. 2 into a controller TA C. Before translating, we need to synchronize C and $\mathcal{G}$ so that C can observe the state of $\mathcal{G}$ and control it. To observe the locations, we assign unique IDs and use global flags for each component to keep track of the current active location. Then we rename the local clocks to be global to make them visible. To monitor every uncontrollable transition in $\mathcal{G}$, we use a unique channel u and the synchronizations $u!$ in $\mathcal{G}$ and $u?$ in C. Similarly, to control $\mathcal{G}$, controllable actions a_i use the corresponding channel synchronizations $a_i!$ in C and $a_i?$ in $\mathcal{G}$.

In Fig. 5, we define location IDs for `Main.L0` – `Main.L4` and `Main.goal` from 0 to 5. Then we use the global location flag *loc* to keep track of the current location of `Main`, and the global clock x to replace the local one, then the broadcast channels $u1, u2, a1 - a4$ to synchronize `Main` and its controller TA `MyCon` in Fig. 6. In `MyCon`, by testing *loc* on the predefined location IDs, transitions from `Init` lead to the switch states `L0` – `L3` and `L5`, which correspond to the strategies at locations `Main.L0` – `Main.L3` and `Main.goal`. Choice states `M00`, `M10`, `M20` and `M30` depict case (1) in Fig. 4. `Accept` corresponds to case (2). `M01`, `M11` and `M31` match case (3).

We also add price and a delay distribution to `Main` for performance evaluation in SMC. This essentially turns `Main` into a priced timed automaton. We use an integer s to count the number of transitions to reach `goal`, and a clock e to measure the energy consumption to reach `goal`. The rate of the clock e is specified at all locations as $e' == n, n \in \mathbb{N}$ except at `L4` because `L4` is not reachable under the strategy. e' is stopped at `goal` by setting to 0. Besides, an exponential rate of 3 is defined for the delay density function at `L1`. Now a closed system can be made from `Main` and `MyCon`. We can verify correctness properties and evaluate performance aspects of this strategy as shown in Table 1.

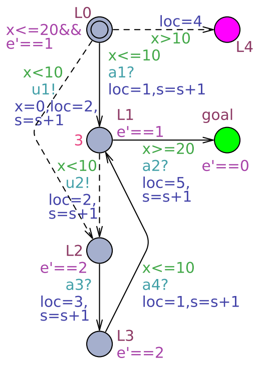


Fig. 5. Decorated TGA `Main`

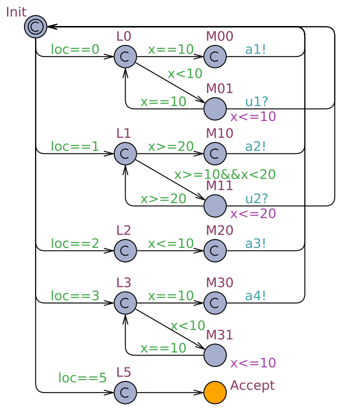
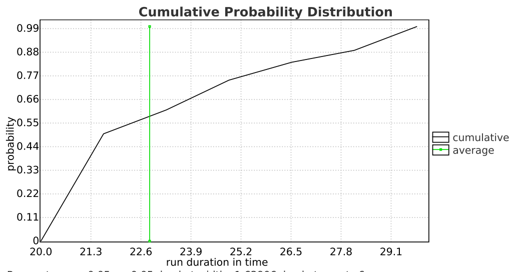


Fig. 6. Controller TA `MyCon`

Table 1. MC & SMC Experiments of the Running Example

	#	Queries	Results
MC	1	`A<> Main.goal`	Yes
	2	`A<> Main.goal and time<=20`	No
SMC	3	`Pr[<=30] (<> Main.goal)`	[0.902606,1]
	4	`E[<=30;200] (max: Main.s)`	3.05
	5	`E[<=30;200] (max: Main.e)`	27.5137

Experiment 1 verifies the original control objective that is satisfied (`Yes`) for sure. Experiment 2 verifies if the strategy ensures `Main` to reach `goal` within 20 time units, where `time` is a global clock. The result is not satisfied (`No`). We evaluate reachability of `goal` within 30 time units under the strategy in experiment 3. The probability is `[0.902606,1] with confidence 0.95` if the probability uncertainty factor ϵ is 0.05. Besides, several kinds of statistical plots can be generated by Uppaal-smc such as probability distribution, probability density distribution, cumulative probability distribution, and frequency histogram. Fig. 7 shows the cumulative probability distribution of 36 runs. The curve shows that over 55% of runs reach `goal` between 20.0 and 22.6 time units, and almost 90% runs can reach `goal` within 29.1 time units. The last two experiments report the expected number of steps and energy consumption to reach `goal` for 200 simulated runs within 30 time units.



Parameters: α=0.05, ε=0.05, bucket width=1.62906, bucket count=6
Runs: 36 in total, 36 (100%) displayed, 0 (0%) remaining
Span of displayed sample: [20, 29.7744]
Mean of displayed sample: 22.8265 ± 1.12486 (95% CI)

Fig. 7. Distribution on Time to Reach `goal`

5 MC and SMC under Strategies

Control-SMC is a new extension of UPPAAL. It automatically synthesizes a strategy of a timed game, keeps the strategy in memory, then verifies and evaluates the strategy on a number of SMC properties. We extended the semantics and algorithms of MC and SMC to apply the synthesized strategy when exploring the state space (for MC) and generating random runs (for SMC).

5.1 Extended Stochastic Semantics

Let $\mathbf{A} = (\mathcal{P}_1|\dots|\mathcal{P}_n)$ be a network of priced timed automata modelling an environment to be controlled. That is $\mathbf{A}$ may be seen as a timed game with global state space $St = St_1 \times \cdots \times St_n$, and with sets Σ_c and Σ_u of controllable and uncontrollable actions, respectively. Now assume that – using UPPAAL-TIGA– we have synthesized a strategy $s : St \rightarrow (\mathbb{R} \times \Sigma_c) \cup \{\lambda\}$ for $\mathbf{A}$ ensuring some desired reachability or safety objective. That is $s(q) = (d, a)$ indicates that the strategy s in state q proposes to perform controllable action a after a delay of d; $s(q) = \lambda$ indicates that the strategy will delay indefinitely until the environment has performed an uncontrollable action. Now we may view the *extended* network:

$$\mathbf{A}^e = (\mathcal{P}_1|\dots|\mathcal{P}_n|A_s)$$

as a closed *stochastic* network over $\Sigma_u \cup \Sigma_c$, where the components $\mathcal{P}_1, \dots, \mathcal{P}_n$ have been given delay density functions $\mu^1, \dots, \mu^n$ and output probability functions $\gamma^1, \dots, \gamma^n$. Now A_s is a one-state component implementing the strategy s. That is s has delay density function $\mu^s_q = \delta_d$, when $s(q) = (d, a)$ and δ_d is the Dirac delta function with probability mass concentrated at time-point d[3]. Moreover the output probability function γ^s_q for s is given by:

$$\gamma^s_q(b) = \begin{cases} 1 & ;\ s(q) = (0, a), a = b \\ 0 & ;\ s(q) = (0, a), a \neq b \\ \perp & ;\ s(q) = (d, a), d > 0 \\ \perp & ;\ s(q) = \lambda \end{cases}$$

In this way $\mathbf{A}^e$ may be subject to statistical model checking provided. We extend the capability of UPPAAL-SMC to generate random runs for networks of environment components extended with control strategies.

5.2 Implementation

Fig. 8 shows the work-flow of Control-SMC. The UPPAAL-TIGA engine receives the timed game model $\mathcal{G}$ and the control objective ϕ. It synthesizes a strategy that is kept in memory if $\mathcal{G}$ is controllable. The strategy can be printed out with the option `-w0`. If the option `-X` is used then subsequent MC or SMC queries

[3] which should formally be treated as the limit of a sequence of delay density functions with decreasing, non-zero support around d.

ρ_i are checked under this strategy. For the purpose of evaluating performance, the model $\mathcal{G}$ can be extended with costs to $\mathcal{G}'$. These costs are modeled with clocks that must be declared as *hybrid clock*. They are ignored for the purpose of symbolic model-checking (synthesis or MC) and taken into account for SMC. Furthermore, floating-point variables can be used in the same way. These additional variables may not be *active* for the purpose of controlling the behavior.

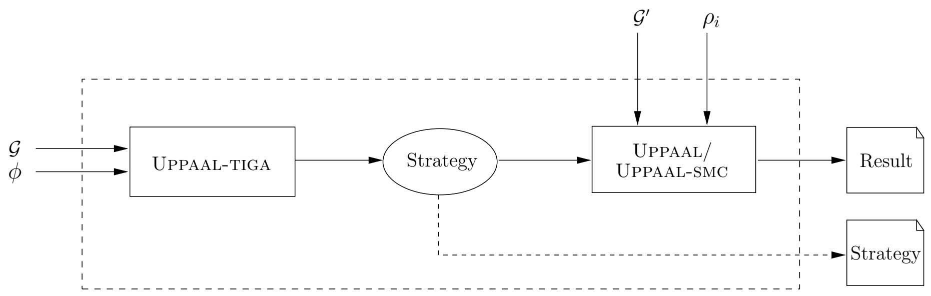


Fig. 8. Workflow of Control-SMC

The exploration under a given strategy is similar to standard MC or SMC when considering uncontrollable transitions since they are played by an opponent. The opponent is stochastic for the purpose of SMC and when doing MC, all possible successors are tried. However, only the controllable transitions allowed by the strategy are allowed. In addition, delay is constrained by the delays of the strategy, e.g., if a controllable transition is to be taken after 5 time units, Uppaal will not delay more. For SMC, this is resolved naturally through the semantics with a race between components. For the symbolic exploration, the strategy specifies how much delay is allowed and this constrains the standard delay operation. Furthermore, we have to add the upper border of bounded delays to enable following transitions. More precisely Uppaal-tiga maintains a partition so we could have the case to wait while in $x \in [0, 5[$ and take a transition at $x = 5$, but $x = 5$ is then unreachable. Therefor we have to wait while $x \in [0, 5]$. Finally, when an action follows a delay it has an urgent semantics, i.e., the states in which such an action is enabled are not allowed to delay.

5.3 The Running Example

We demonstrates how to use Control-SMC on the running example described in Section 2.2 and 4.2 without the need to translate the strategy. We add prices and stochastic information directly on the TGA `Main` as shown in Fig. 9. The clocks used for cost are declared as `hybrid clock` (e.g. e), while counters for SMC evaluation are declared as `double` (e.g. s).

Fig. 10 shows the query file we use here. A control query that expresses the control objective starts on the first line with a list of MC and SMC queries on

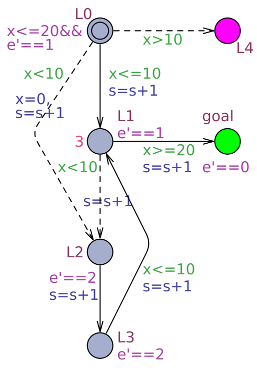


Fig. 9. TGA Main with Prices

```
control: A<> Main.goal
A<> Main.goal
A<> Main.goal and time<=20
Pr[<=30] (<> Main.goal)
E[<=30;200] (max: Main.s)
E[<=30;200] (max: Main.e)
```

Fig. 10. Combined Query File

the following lines. For now, Control-SMC is available only from the command line checker `verifytga` and is enabled with the option `-X`. Given the model as in Fig. 9 and the query file as in Fig. 10 as inputs, it first synthesizes a strategy for the control query, then processes the rest MC and SMC queries in a batch fashion, and gives the same results as in Table 1 in Section 4.2.

6 Experiments Results

We show the experiments of two case-studies by first using the Control-SMC method of Section 5, then using the strategy translation method in Section 4 as a cross-check. The two methods gave the same results for MC and SMC queries. We also measured the execution time of the queries for both methods, because we want to know the runtime benefit of applying a strategy in time and memory compared with using a translated controller from a strategy. All models in the experiments are available on our SMC web-page[4].

6.1 Case Study 1: Jobshop

The Jobshop problem is about scheduling a set of machines for a set of jobs, where each job needs to use those machines in a particular order for a particular time limit. This case-study involves two professors Kim and Jan who want to read a single piece of four-section newspaper. Each person has his own preferred order on sections, and can spend different times on different sections. The control objective, which is expressed as `control: A<> Kim.Done and Jan.Done and time<=80`, is to find a scheduling strategy that guarantees both people finish reading within 80 time units. UPPAAL-TIGA finds such a strategy. The full explanation about this model can be found on web-page of examples [11]. The model is down-sized for the purpose of the manual conversion to a controller automaton.

[4] Section Control-SMC at `http://people.cs.aau.dk/~adavid/smc/cases.html`

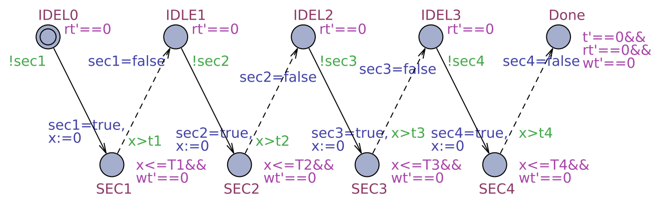


Fig. 11. Job Template with Prices

Table 2. MC & SMC Experiments of Jobshop

	#	Queries	Results	T (CS)	T (M)
MC	1	`A[] Jan.Done imply Kim.Done`	`Yes`	-	-
	2	`E<> Kim.Done and Jan.Done and time<=45`	`Yes`	-	-
	3	`E<> Kim.Done and Jan.Done and time<=44`	`No`	-	-
SMC	4	`Pr[<=80] (<> Kim.Done and Jan.Done)`	1*	76.5s	148.3s
	5	`E[time<=80 ; 2000000] (max: Kim.wt)`	`5.40221`	62.1s	132.5s
	6	`E[time<=80 ; 2000000] (max: Kim.rt)`	`22.7469`	61.7s	138.7s
	7	`E[time<=80 ; 2000000] (max: Jan.wt)`	`11.5652`	60.9s	136.8s
	8	`E[time<=80 ; 2000000] (max: Jan.rt)`	`47.3951`	62.1s	138.8s

1* `in [0.999998,1] with confidence 0.95.`

Fig. 11 shows the TGA template with prices for each person for Control-SMC. The availability of four sections are maintained by four global boolean variables. During the initialization of Kim and Jan, the references to the boolean variables are assigned to `sec1` − `sec4` according to each person's preferred order of reading. The strategy tells a person when to acquire a section (controllable, solid edge), while a person can release a section at any time within a time bound (uncontrollable, dashed edge). We add respectively three stop-watches[5] wt, rt and t to measure the accumulated time on waiting, reading and finishing the newspaper respectively.

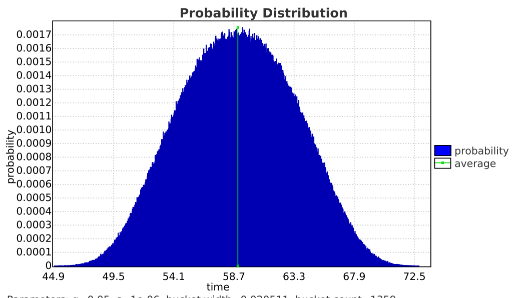
We obtain the same results when checking the MC and SMC queries in Control-SMC and UPPAAL. Thus in Table 2 we use the single column Result to show the MC results (`Yes` for satisfied or `No` for not satisfied), and SMC results (probabilities or evaluations). The T (CS) column shows the execution time of a query in seconds by Control-SMC, while the T (M) column shows that by using a manually translated controller. We do not compare the runtime of MC queries because the size of this model is not big enough to make the runtime distinguishable. But we compare the runtime of SMC queries, because we can let the SMC engine to generate a large number of runs to make the runtime difference noticeable.

Experiment 1 shows that Kim always finishes reading before Jan. We get the shortest time (= 45 time units) for both to finish from experiments 2 and 3.

[5] Stop-watches are clocks whose rates are reset to zero.

Experiment 4 measures the probability for Kim and Jan to finish reading within 80 time units if the probability uncertainty $\epsilon = 0.000001$.

In UPPAAL-SMC we can get the plot of probability distribution of this query as shown in Fig. 12. The plot gives the mean value of around 59 time units. The remaining SMC experiments show the expected time for Kim and Jan to wait and read the newspaper individually. The strategy biases Kim because Kim waits less than Jan. The runtime experiments of SMC queries were carried out on a PC with Intel i7-2640M CPU @ 2.80GHz, 8GB main memory and Ubuntu 12.04 x86_64 with the upcoming version 0.18 of UPPAAL-TIGA. Experiment 4 set $\epsilon = 0.000001$ to force the SMC engine to generate a large number of runs (1844438 runs). In experiments 5 − 8, we set the number of runs to 2000000. We can conclude that applying a strategy in memory improves the performance of SMC engine inside Control-SMC by a factor of two. This is due to the strategy look-up in a hash table instead of simulating it within the model.



Parameters: α=0.05, ε=1e-06, bucket width=0.020511, bucket count=1359
Runs: 1844438 in total, 1844438 (100%) displayed, 0 (0%) remaining
Span of displayed sample: [44.9821, 72.8566]
Mean of displayed sample: 59.003 ± 0.00626518 (95% CI)

Fig. 12. Distribution on Time to Finish Reading for Both People

6.2 Case Study 2: Train-Gate

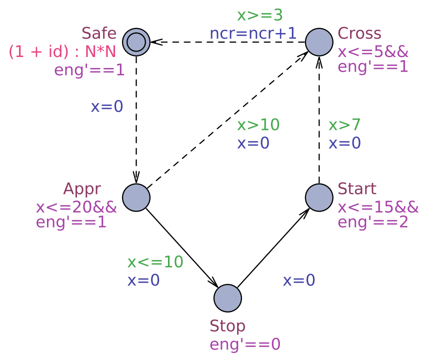


Fig. 13. Train Template with Prices

Train-Gate is a classical case-study for real-time model checking. It is distributed with UPPAAL with an detailed explanation in [3]. Fig. 13 shows the game version of it with prices and stochastic extensions. The control objective, which is expressed as `control: A[] forall (i : id_t) forall (j : id_t) Train(i).Cross and Train(j).Cross imply i == j`, is finding a strategy to guarantee the exclusive access to `Cross` by two trains. If necessary, the strategy should stop a train at `Appr` in time ($x \leq 10$) by the controllable solid edge to `Stop`, otherwise the train goes to `Cross` directly by the uncontrollable dashed edge. The train can resume at `Stop` by the other controllable solid edge to `Start`. The exponential rate `((1+id):N*N)` appears at `Safe` for specifying the delay density function. A counter *ncr* records the throughput at `Cross`.

Table 3. MC & SMC Experiments of Train-Gate

	#	Queries	Result		T (CS)	T (M)
			Syn	Que		
MC	1	`E<> Train(0).Cross && Train(1).Start`	`Yes`	`No`	-	-
SMC	2	`Pr[<=100] (<> Train(0).Cross)`	1^*	1^*	45.9s	88.5s
	3	`E[<=100 ; 1000000] (max: ncr)`	`8.0665`	`5.8065`	72.3s	173.3s
	4	`E[<=100 ; 1000000] (max: Train(0).eng)`	`124.938`	`88.402`	69.3s	169.5s

1^* `in [0.999998,1] with confidence 0.95.`

A hybrid clock e measures the energy consumption of a train. The interesting point of this case-study is that we compare the behavior and performance of the synthesized strategy with the manually programmed queue-based controller available in the train-gate example provided in the distribution of Uppaal.

Table 3 shows the comparative experiments of the synthesized strategy `Syn` and the queue-based controller `Que`. Experiment 1 shows `Syn` allows `Train(1)` to approach `Cross` while `Train(0)` is still crossing. This is forbidden by `Que`. Experiment 2 measures the probability for `Train(0)` to reach `Cross` within 100 time units with the probability uncertainty $\epsilon = 0.000001$. Experiment 3 shows that `Syn` gives a bigger throughput from `Que`, because `Syn` allows different trains to approach `Cross` concurrently as witnessed by experiment 1. Experiment 4 gives the expected energy consumption for `Train(0)`. We compare the execution time of SMC queries in seconds by Control-SMC in the T (CS) column with that using a manually translated controller in the T (M) column. In experiment 2, we set $\epsilon = 0.000001$ to force the SMC engine to generate a large number of runs (1844438 runs). In experiments 3 and 4, we set the number of runs to 1000000. We can conclude that applying a strategy in memory improves the performance of SMC engine inside Control-SMC by a factor of two.

7 Future Work

The future work are in three directions. Our first goal is to merge Uppaal and Uppaal-tiga, which will enable Control-SMC from the graphical interface with all its capabilities, in particular the plot composer. Next, we aim to make the clocks for measuring prices in Control-SMC to become real hybrid as in Uppaal-smc. The clock rates can be floating-point, negative, or in the form of ordinary differential equations (ODE). The third direction is exploring more potential use of the synthesized strategy in memory. We can try to refine or optimize the strategy using machine learning methods.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)

2. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV. Lecture Notes in Computer Science, vol. 4590, pp. 121–125. Springer (2007)
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM. Lecture Notes in Computer Science, vol. 3185, pp. 200–236. Springer (2004)
4. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. Lecture Notes in Computer Science, vol. 3098, pp. 87–124. Springer (2003)
5. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: A model-checking tool for real-time systems. In: Hu, A.J., Vardi, M.Y. (eds.) CAV. Lecture Notes in Computer Science, vol. 1427, pp. 546–550. Springer (1998)
6. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR. Lecture Notes in Computer Science, vol. 3653, pp. 66–80. Springer (2005)
7. Cassez, F., Jessen, J.J., Larsen, K.G., Raskin, J.F., Reynier, P.A.: Automatic synthesis of robust and optimal controllers - an industrial case study. In: Majumdar, R., Tabuada, P. (eds.) HSCC. Lecture Notes in Computer Science, vol. 5469, pp. 90–104. Springer (2009)
8. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Stochastic semantics and statistical model checking for networks of priced timed automata. CoRR abs/1106.3961 (2011)
9. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV. Lecture Notes in Computer Science, vol. 6806, pp. 349–355. Springer (2011)
10. Jessen, J.J., Rasmussen, J.I., Larsen, K.G., David, A.: Guided controller synthesis for climate controller using uppaal tiga. In: Raskin, J.F., Thiagarajan, P.S. (eds.) FORMATS. Lecture Notes in Computer Science, vol. 4763, pp. 227–240. Springer (2007)
11. Larsen, K.G.: Quantitative model checking exercise. `http://people.cs.aau.dk/~kgl/QMC2010/exercises/` (2010), 28. Job Shop Scheduling
12. Larsen, K.G., Pettersson, P., Yi, W.: Model-checking for real-time systems. In: Reichel, H. (ed.) FCT. Lecture Notes in Computer Science, vol. 965, pp. 62–88. Springer (1995)
13. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: STACS. pp. 229–242 (1995)
14. Younes, H.L.S.: Planning and verification for stochastic processes with asynchronous events. In: McGuinness, D.L., Ferguson, G. (eds.) AAAI. pp. 1001–1002. AAAI Press / The MIT Press (2004)