

Model Checking, Performance Analysis, Synthesis and Learning for Cyber Physical Systems

Kim G. Larsen¹  and Marius Mikučionis¹ 

Aalborg University, Aalborg, Denmark {kg1,marius}@cs.aau.dk

Abstract. During a period of 30 years, the tool UPPAAL has evolved from being a Timed Automata model checker to a tool allowing reinforcement learning of near-optimal control strategies for hybrid Markov Decision Processes. In this tutorial, we present the various capabilities of UPPAAL, with a detailed explanation of the syntax and semantics of the various supported modeling formalisms. We demonstrate through examples the type of analysis supported, which includes model checking, performance analysis, controller synthesis, and reinforcement learning. In addition, we provide details of the various engines that allow for these analyses to be carried out efficiently.

Keywords: Model Checking · Performance · Synthesis · Learning · Optimization · Timed Automata · UPPAAL

1 Introduction

Cyber-Physical Systems (CPS) are systems that combine computational elements with dedicated hardware and software in order to monitor and control physical processes. In other words, CPS tightly integrate the physical world with the digital world. This integration forms the foundation for many of today's smart solutions in areas such as transportation, healthcare, energy systems, and manufacturing.

However, designing CPS is far from trivial. These systems are expected to deliver ever-increasing functionality and performance, while being developed under strict time and cost constraints. At the same time, they must remain dependable—failures are often unacceptable in safety-critical applications. This combination of demands makes CPS development a major software engineering challenge.

To meet these challenges, the research community has developed mathematically rigorous and scalable methods, tools, and techniques. Over the past three decades, one influential approach has been model-based design, where abstract models of a system are created and analyzed before implementation. Our work in this area has led to the development of the UPPAAL tool suite (uppaal.org) [79], a family of tools based on efficient algorithms and data structures for analyzing well-defined system models. UPPAAL has been successfully applied

in a wide range of industrial case studies, particularly within Embedded and Cyber-Physical Systems. For a thorough account of industrial applications of the various branches of UPPAAL, we refer the reader to the papers [58,78] as well as the very recent review [57].

The first version of UPPAAL was presented at the inaugural TACAS conference in 1995 in Aarhus, Denmark. Initially developed through a collaboration between Uppsala University (Sweden) and Aalborg University (Denmark), UPPAAL has since evolved into several specialized branches. In 2023, many of these branches were consolidated into a unified platform: UPPAAL 5.0. In 2016 UPPAAL received the CAV Award for “being the foremost tool-suite for automated analysis and verification of real-time systems”.

The goal of this tutorial is to introduce the different capabilities of UPPAAL 5.0 through examples, explain their underlying modeling formalisms, demonstrate the analyses they make possible, and provide sufficient implementation details to enable effective usage of the tool.

2 Timed Automata

The underlying formalism of UPPAAL is that of timed automata as introduced by Rajeev Alur and David Dill [2]. Timed automata have proven very convenient for modeling and reasoning about real-time systems: they combine a powerful formalism with advanced expressiveness and efficient algorithmic and tool support, and have become a model of choice. Timed automata extend traditional automata with real-valued clocks, making them well suited for representing and analyzing systems where timing plays a crucial role. This is especially important in embedded, real-time and cyber-physical systems, where correctness often depends not only on what the system does, but also on when it does it.

Today, the timed automata formalism is routinely applied in several domains covering

- analysis of real-time control programs,
- timing analysis of software and asynchronous circuits,
- verification of real-time communication protocols, and
- schedulability and response-time analysis.

Since the introduction of the UPPAAL tool in 1995, significant effort has been put into development and implementation of improved data structures and algorithms for the efficient analysis of timed automata.

This includes guided search algorithms using AI heuristics [13,52] [74,75,9], fully symbolic data structures [9], minimal constraint normal forms [76], as well as a new symbolic DART data structure [70,72] making useful trade-offs between the effectiveness of discrete and symbolic semantics. In addition, this research has included the development of a series of exact abstractions (or extrapolations) that not only ensure the finiteness of symbolic semantics, but also provide significant performance improvements [10,11,63]. In addition to these advances with respect to the verification engine, significant effort has been made over the years

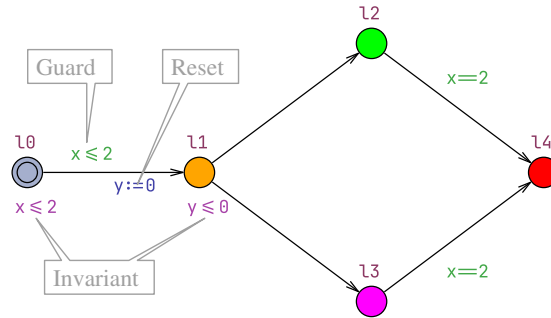


Fig. 1: Timed automaton with constraints over x and y clocks.

on the graphical interface of the tool (e.g. [14]), and on the modeling side, the introduction of user-defined data types and procedures has undoubtedly made the tool significantly more usable in modeling real control programs and communication protocols [13]. For a more detailed account of the timed automata formalism and the main model-checking results in this framework, we refer to the corresponding chapter in the Handbook of Model Checking [26].

2.1 Syntax and Semantics

A timed automaton is essentially a finite-state automaton extended with a finite set of real-valued clocks. In Fig. 1 we see a timed automaton with five locations (l_0 to l_4) and two clocks x and y . Locations as well as edges may have logical constraints (so-called invariants and guards) on the value of the clocks. In Fig. 1 the location l_0 is marked as the initial location by the inscribed circle and it has an invariant $x \leq 2$. Also, location l_1 uses an invariant $y \leq 0$. The edge from l_0 to l_1 has a guard $x \leq 2$. Also on this edge $y := 0$ indicates a reset of the clock y . In the following, we give the formal definition of a timed automaton:

The clock constraints that may be used in a timed automaton are quite restrictive: being essentially Boolean combinations of bounds on individual clocks and the difference between two clocks. This restriction is by necessity, as even small extensions will lead to undecidability of most problems related to timed automata. Formally, clock constraints must be expressions $g \in G(C)$ of the form:

$$g ::= \text{true} \mid \text{false} \mid x \sim c \mid x - y \sim c \mid g \wedge g$$

for clocks $x, y \in C$, constants $c \in \mathbb{Z}$ and comparisons $\sim \in \{\leq, <, =, >, \geq\}$. We may now define the syntax of timed automata:

Definition 1. A timed automaton $\mathcal{A}$ is a tuple (L, ℓ_0, C, A, E, I) :

- L is a finite set of locations,
- $\ell_0 \in L$ is an initial location,
- C is a finite set of $\mathbb{R}_{\geq 0}$ -valued clocks which evolve at the same rate,

- A is a set of actions,
- $E \subseteq L \times G(C) \times A \times 2^C \times L$ is a set of directed edges between two locations decorated with:
 - a guarding expression G over clocks C ,
 - an action from A , and
 - reset expressions $r \subseteq C$ of the form: $\bigcup_{x \in r} x := 0$,
- $I : L \rightarrow G(C)$ is a mapping providing an invariant expression for each location.

Intuitively, guards and invariants control the timed behavior of an automaton: a guard expression enables an edge when a certain clock reaches a specified integer value, an invariant expression forces to leave a location when a clock reaches a certain integer value, and the behavior is *non-deterministic* anywhere in between (some edge can be traversed, or automaton can stay in the same location as long as its invariant is satisfied). Reset expressions are used to update variable values, and action labels A can be used to identify action transitions (just like in finite state automaton), signify certain features and synchronize multiple automata.

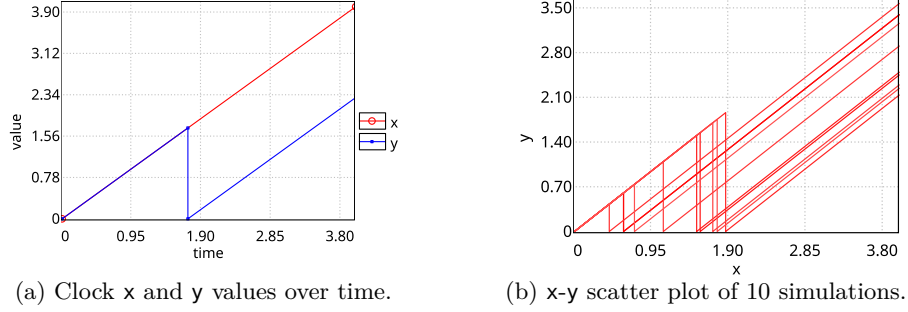
Semantically, a timed automaton $\mathcal{A} = (L, \ell_0, C, A, E, I)$ describes labeled transition systems, where states are pairs of the form $\langle \ell, v \rangle$ with ℓ being a location and $v \in \mathbb{R}_{\geq 0}^C$ is a clock valuation, i.e. a function that assigns non-negative real values to clocks. Transitions between states are divided into two types: delay transitions and action transitions. For example, for the timed automaton of Fig. 1 $\langle \mathbf{l0}, x = 0.3, y = 0.3 \rangle$ is a state that may be reached after a delay of 0.3 from the initial state $\langle \mathbf{l0}, x = 0, y = 0 \rangle$. For convenience we will write $v \models g$ if the valuation v satisfies the expression g ; we will write $v + d$ for the valuation with $(v + d)(x) = v(x) + d$ for any clock $x \in C$; and finally for a reset set $r \subseteq C$ we write $v[r]$ for the valuation $v[r](x) = 0$ if $x \in r$ and $v[r](x) = v(x)$ if $x \notin r$. We may now provide the formal semantics:

Definition 2. *The semantics of a timed automaton $\mathcal{A} = (L, \ell_0, C, A, E, I)$ is the Timed Labeled Transition System $(S, s_0, A, \rightarrow)$, where:*

- State set $S = \{ \langle \ell, v \rangle \mid \ell \in L, v \in \mathbb{R}_{\geq 0}^C, v \models I(\ell) \}$ with initial state $s_0 = \langle \ell_0, v_0 \rangle$, where v_0 maps all clocks to 0.
- Delay-transition ($d \in \mathbb{R}_{\geq 0}$): $\langle \ell, v \rangle \xrightarrow{d} \langle \ell, v + d \rangle$ iff $v + d \models I(\ell)$.
- Action-transition ($\alpha \in A$): $\langle \ell, v \rangle \xrightarrow{\alpha} \langle \ell', v' \rangle$ iff $\langle \ell, g, \alpha, r, \ell' \rangle \in E, v \models g, v' = v[r], v' \models I(\ell')$.

Now reconsidering the timed automaton of Fig. 1, we observe the following transition sequence leading from the initial state to a final goal state:

$$\begin{aligned}
 \langle \mathbf{l0}, x = 0, y = 0 \rangle &\xrightarrow{1.9} \langle \mathbf{l0}, x = 1.9, y = 1.9 \rangle \rightarrow \langle \mathbf{l1}, x = 1.9, y = 0 \rangle \\
 &\rightarrow \langle \mathbf{l2}, x = 1.9, y = 0 \rangle \xrightarrow{0.1} \langle \mathbf{l2}, x = 2.0, y = 0.1 \rangle \\
 &\rightarrow \langle \mathbf{l4}, x = 2.0, y = 0.1 \rangle
 \end{aligned}$$


 Fig. 2: Trajectories of x and y clock values.

In Fig. 2a we illustrate the continuous evolution of the values of x and y as individual functions over time for one given (randomly generated) run. Also in Fig. 2b we provide a scatter plot from 10 random runs illustrating the relationship between the values of the two clocks.

The most basic question about timed automata is that of *reachability*. This problem asks, given a timed automaton $\mathcal{A} = (L, \ell_0, C, A, E, I)$ and a designated set of goal locations $F \subseteq L$, is there a run from the initial state of $\mathcal{A}$ that reaches a state $\langle \ell, v \rangle$ where $\ell \in F$. Clearly **14** is reachable in the timed automaton of Fig. 1.

2.2 Symbolic Techniques

Clearly, the semantics of timed automata results in transition systems with infinite (even uncountably infinite) state spaces. This makes decidability of even simple properties such as reachability non-obvious. Fortunately, there exist symbolic techniques that essentially allow one to reason about properties (reachability, model-checking) of timed automata through finite abstractions.

Regions: From Infinite to Finite One such abstraction is given through a partitioning of the infinite state-space into finite so-called *regions*. Here we give a simplified account.

Now let $\mathcal{A} = (L, \ell_0, C, A, E, I)$ be a fixed timed automaton and let us by M denote the maximum constant appearing in any constraint of $\mathcal{A}$. Now, let $G_M(C)$ be the (finite) subset of clock constraints in $G(C)$, where clocks and clock differences are only compared to integers between $-M$ and M .

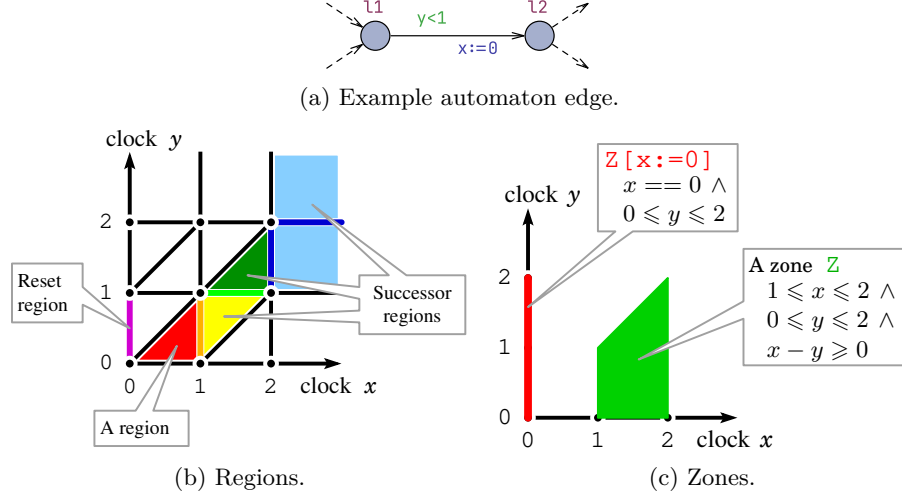


Fig. 3: Geometric interpretation of symbolic techniques.

Definition 3. Two valuations $v, v' : C \rightarrow \mathbb{R}_{\geq 0}$ are said to be region equivalent w.r.t. the maximal constant M , denoted $v \simeq_M v'$, if v and v' satisfy the exact same constraints in $G_M(C)$ ¹.

Now the equivalence classes under $\simeq_M$ are called regions (with maximal constant M). The number of regions is finite and bounded above by $n! \cdot 2^n \cdot (2M+2)^n$ – an exponentially growing number in the number of clocks n , and in the binary (or any non-unary) syntactic representation of M . Region equivalence of valuations is extended to states of $\mathcal{A}$ by declaring that $\langle \ell, v \rangle \simeq_M \langle \ell', v' \rangle$ whenever $\ell = \ell'$ and $v \simeq_M v'$.

Now, Fig. 3b illustrates the number of regions resulting from a timed automaton $\mathcal{A}$ with two clocks x, y and with maximum constant $M = 2$. Each point in the grid, each line segment, and each open triangular area is a region (you can try to figure out the constraint that characterizes the region indicated in red).

Now equivalence enjoys many nice properties, the most important is that it provides a *time-abstracted bisimulation*. This means that two region-equivalent states agree on which regions may be reached by action-transitions as well as delay-transitions. Reconsidering Fig. 3b we see that all valuations in the red region agree on the fact that the edge indicated in Fig. 3a is enabled and that the result of taking the edge will be a valuation in the purple “reset” region. Also, delays from any valuation in the red region will go through the same sequence of “successor regions”: orange, yellow, light green, green, blue, light blue, etc.

¹ Note that this is a simpler definitions of regions compared to that in [26], where each clock x has an associated maximum constant M_x . The result is a larger number of regions in the presented definition.

In fact, the stability of regions under delays and discrete actions results in a finite-state region graph.

The number of regions grows exponentially large in size but provides the basis for showing decidability of a number of important problems such as reachability, model checking, and timed (untimed) bisimulation checking. In fact, checking a reachability property can be done on the fly on the region graph, hence this can be done in polynomial space. One of the most fundamental theorems in the model checking of timed automata is stated below:

Theorem 1 ([2]). *The reachability problem for timed automata is PSPACE-complete.*

Furthermore, time-optimal reachability has been shown to be PSPACE-complete [7]² as well.

Zones: From Finite to Efficient As shown above, regions provide a finite and elegant abstraction of the infinite state space of timed automata, enabling us to prove decidability of a wide range of problems, including (timed and untimed) bisimilarity, reachability, and TCTL model-checking (as we shall expand on later). Unfortunately, the number of states obtained from the region partitioning is extremely large, growing exponentially in the number of clocks and in (the binary representation of) the maximal constants of a given timed automaton.

Therefore, substantial efforts have been made to develop a more efficient representation of the state space, using the coarser notion of *zone*. This underlying idea is that one can in practice cover the reachable state space of most timed automata with substantially fewer zones than regions.

Definition 4. *Let C be a finite set of clocks. A subset $Z \subseteq \mathbb{R}_{\geq 0}^C$ is called a zone if there exists a constraint $g \in G(C)$ such that $Z = \{v \in \mathbb{R}_{\geq 0}^C \mid v \models g\}$.*

For reachability analysis, we need the following operations on zones: for a zone $Z \subseteq \mathbb{R}_{\geq 0}^C$ and $r \subseteq C$ we denote:

- the *delay* of Z by $Z^\uparrow = \{v + d \mid v \in Z, d \in \mathbb{R}_{\geq 0}\}$
- the *reset* of Z under r by $Z[r] = \{v[r] \mid v \in Z\}$.

Most importantly, it can be shown that zones are closed under delay, reset, and intersection. Consider the green zone Z in Fig. 3c and its corresponding constraint. Then the red set of valuations $Z[x := 0]$ – being the reset of Z under x – is also a zone as witnessed by the characterizing constraint. These closure properties allow for efficient symbolic zone-based exploration of timed automata. For example, Fig. 4a shows a zone of clock values acquired while being in location **l0**: it starts at $(0; 0)$ and the delay extends the zone line diagonally (the clocks

² That is given a timed automaton $\mathcal{A}$, a designated set of goal locations F , and a time-bound T it is possible to reach a state with location in F within a total time of T .

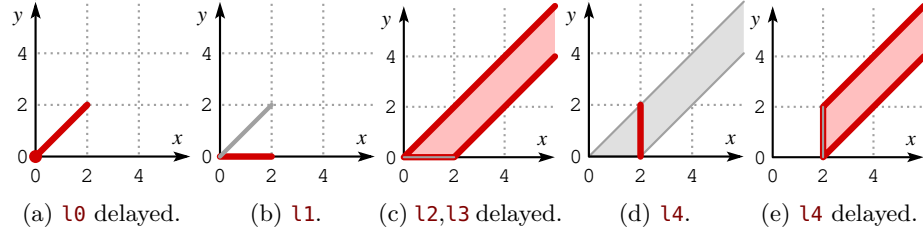


Fig. 4: Zones for each location of TA in 1.

progress at the same rate) until $(2; 2)$ while the invariant $x \leq 2$ stops it from going further. Figure 4b shows the effect of switching to location **l1**, where the reset $y := 0$ projects the zone line onto x -axis and no delay is not permitted by the invariant $y \leq 0$. Location **l2** and **l3** work the same way by allowing an infinite delay, thus extending the possible clock values diagonally as shown in Fig. 4c. Then both edges leading to location **l4** use the guard $x == 2$ which cuts the zone at $x = 2$ as shown in Fig. 4d and the delay operation extends the zone diagonally as showed in Fig 4e.

Extrapolation Apriori, however, the number of zones for a given set of clocks is infinite – simply due to the fact that $G(C)$ is infinite. Hence, an additional abstraction yielding finiteness is needed. This is provided by extrapolation with respect to the maximal constant M of the given timed automata. Here, the extrapolation of a zone $Z \subseteq \mathbb{R}_{\geq 0}^C$ is the set $\{v : C \rightarrow \mathbb{R}_{\geq 0} \mid \exists v' \in Z. v \simeq_M v'\}$. The extrapolation of a zone is not in general a zone, hence in practice other extrapolation operators are used [10,20,21,11,63,54] [27,67,5,6]. Figure 5 illustrates how zones are closed under various operations as well as the extrapolation operation ensuring finiteness of the number of zones encountered during symbolic exploration. UPPAAL uses a DBM library³ to perform zone operations with low polynomial algorithms.

2.3 Logical Specifications

Given a Timed Automata model UPPAAL supports model checking with respect to a fragment of the branching time logic TCTL [62]. In particular temporal operators are not allowed to be nested⁴. Below we list and exemplify the supported TCTL properties.

The properties are specified using the following grammar:

$$\phi ::= E \Diamond p \mid E \Box p \mid A \Diamond p \mid A \Box p \mid p \dashrightarrow q$$

where E and A are resp. the existential and universal *path quantifiers*, $\Diamond$ and $\Box$ are resp. the existential and universal *state quantifiers*, and $\dashrightarrow$ is a special

³ <https://github.com/UPPAALModelChecker/UDBM>

⁴ In the recent paper [68] support model checking full TATL (thus subsuming TCTL) has been implemented, but not yet been integrated into UPPAAL 5.0.

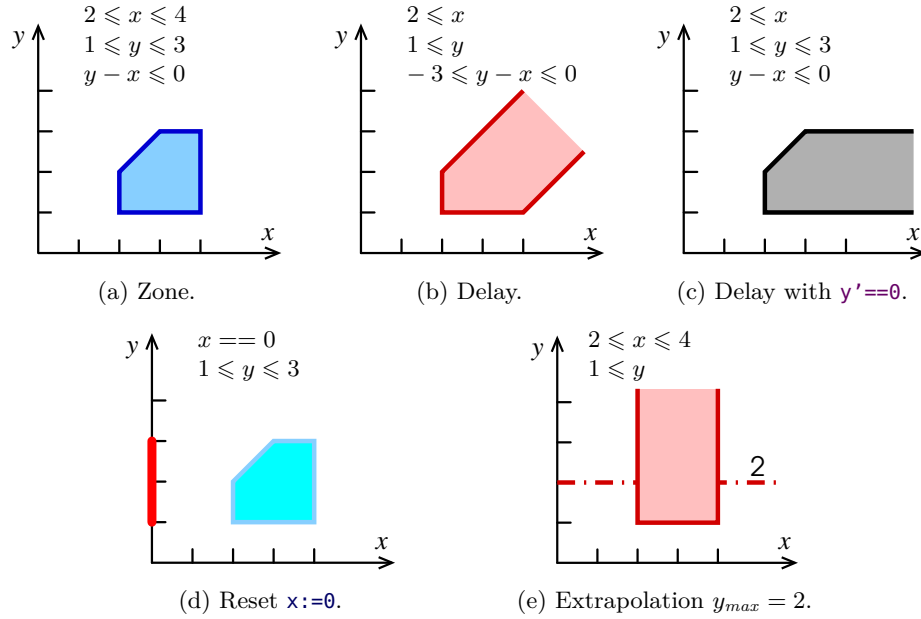


Fig. 5: Zone operations.

shorthand for $A \Box (p \rightarrow A \Diamond q)$. p and q are state predicates: expressions, which are type safe, side effect free, and evaluate to a boolean value. Only references to constants, integer variables, clocks and process locations are allowed.

Validation properties:

- Possibly $E \Diamond p$: there exists a path where a state satisfies p , e.g. $E \langle \rangle P.\text{Possible3}$ is *satisfied*, and $E \langle \rangle P.\text{Error}$ is *not satisfied* in Fig. 6a.

Safety properties:

- Invariantly $A \Box p$: in every path, all states satisfy p , e.g. $A \Box \text{not } P.\text{Error}$ is *satisfied* in Fig. 6d since **Error** is not reachable in process **P**.
- Possibly invariant $E \Box p$: there exists a path, where all states satisfy p , e.g. $E \Box P.\text{Busy}$ and $E \Box P.\text{Busy} \text{ or } P.\text{Future2}$ are *satisfied* in Fig. 6b since **P** can delay indefinitely in **Busy** and **Future2**.

Liveness properties:

- Eventually $A \Diamond p$: for all paths there exists a state satisfying p , e.g. $A \langle \rangle P.\text{good} == 1$ is *satisfied* in Fig. 6c because the local variable **good** in process **P** will eventually be assigned a value 1.
- Leads-to $p \dashrightarrow q$ means that whenever a state satisfies p , for every subsequent path there exists a state satisfying q , e.g.: $P.\text{Fault} \dashrightarrow P.\text{Recover}$ is *satisfied* in Fig. 7a because process **P** cannot stay indefinitely in **Fault** due to invariant $x \leq 20$ and thus will surely transition into **Recover** (and it does so repeatedly every time it visits

Fault). The property is *unsatisfied* in Fig. 7b due to an infinite loop alternating through location **Workaround**, as unfolded in Fig. 7c where the infinite loop⁵ is enabled only after a repair as controlled by the value of variable **repaired**.

For validation and safety properties, the model checking engine of UPPAAL uses a symbolic reachability algorithm based on zones. For liveness properties, UPPAAL performs a symbolic nested depth-first reachability algorithm again using zones [16]. Both algorithms generate the state space and evaluate the state predicates on demand (“on-the-fly”), which allows the exploration to terminate early without exploring the full state space once a witness trace is discovered, saving a lot of time and memory. Given the timed automaton in Fig. 1, the following specifications are satisfied:

- $A \langle \rangle \text{ l4}$, i.e. eventually the goal-location **l4** will be reached;
- $A[] y \leq x$, i.e. the value of x is always larger than that of y .

2.4 Train Gate Example

As we shall see in the following example a UPPAAL model is *not just* a single timed automaton but rather a network of interacting, channel-synchronizing timed automata extended with discrete variables over bounded-range integers, Boolean as well as user-defined types.

A train gate controls mutual exclusion over the bridge crossing where multiple train lines meet as shown in Fig. 8. The gate controller receives **appr** and **leave** signals when trains approach and leave the crossing respectively. It can issue **stop** and **go** signals to trains to stop them and restart to ensure safety. However, the trains can stop safely only if they receive the **stop** signal while they are in the yellow zone (within 10 time units after the **appr** signal is triggered). Once stopped, the train must wait until a **go** signal is issued to resume and it takes 7 to 15 time units to reach the crossing. If a train is not stopped, then it takes 10 to 20 time units to get to the crossing. A train takes 3 to 5 time units to leave the crossing where the **leave** signal is sent to the gate controller. The gate controller maintains a list of approached train identifiers to ensure that the trains are let **go** when it is safe for them to enter the crossing.

UPPAAL timed automata templates for train and gate controller are shown in Fig. 9, where gate controller calls functions listed in Listing 1.1. Each train has an identifier **id** and dedicated channels in arrays of channels **appr**, **leave**, **stop**, **go**. The exclamation marks are for sending and question marks are for receiving. The timings are encoded using guards and invariants, for example, when a train goes from the **Safe** location to **Appro** it signals **appr[id]!** and resets its local clock x , then the train is in **Approaching** state for at most 20 time units (controlled by the invariant $x \leq 20$), where the train can be stopped by **stop[id]?** signal within 10 time units (controlled by the guard $x \leq 10$) or the train can enter **Crossing**

⁵ UPPAAL simulator uses red font to display transitions belonging to an infinite loop.

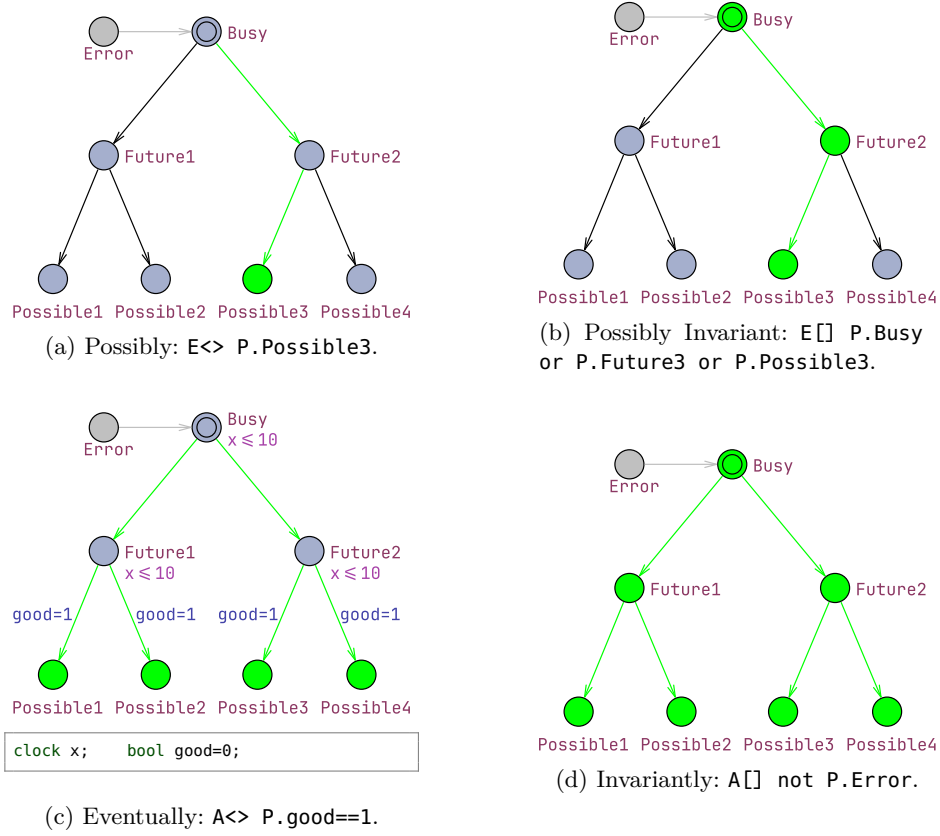


Fig. 6: TCTL quantifiers highlighted over a simple process state space.

location after 10 time units (controlled by the guard $x \geq 10$). The local clock x is then being reused in other locations by being reset $x=0$ before it enters another location.

Similarly, the gate controller shown in Fig 9b in location **Free** awaits for any `appr[e]?` signal where e denotes the train identifier in the select statement `e : id_t`, which is then appended to the end of the list by calling the `enqueue(e)` function. In the **Occupied** location the gate controller lets this first train go without stopping, while awaiting for other signals: if another `appr[e]?` is received, then it adds the train identifier e to the list, moves into **Stopping** location which is *committed* (with C encircled) to exit immediately without any interruption to **Occ** location by issuing `stop[tail()]!` signal to that last train. The gate controller stays in the **Occ** location until the train in front of the queue leaves the crossing and `leave[e]?` is received. The train e is then removed from the front of the queue and the gate moves back to the **Free** location, where its stay is preempted by an urgent option to produce `go[front]!` output when the list is not empty ($len > 0$) and thus the next train in the queue is signaled to enter

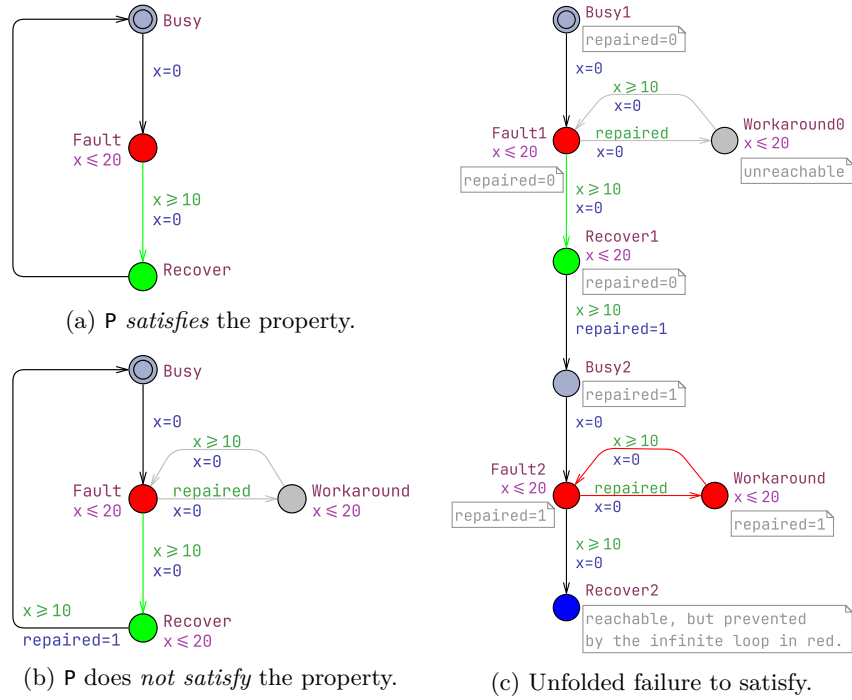
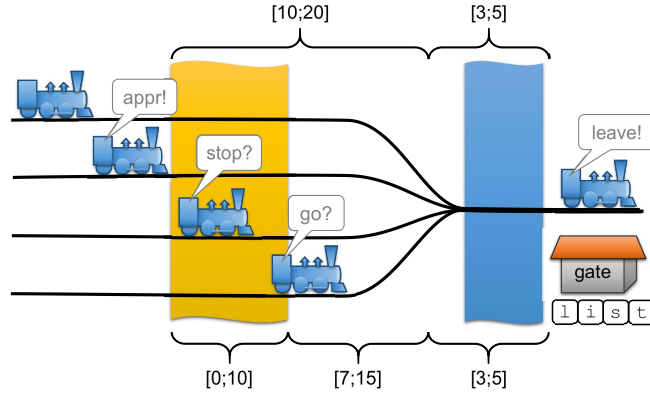
Fig. 7: Example TA for liveness property $P.$ $\text{Fault} \rightarrow \text{Recover}$.

Fig. 8: Crossing over blue river where trains are stoppable in yellow zone.

the crossing. The urgency is modeled by declaring `go` channel as *urgent* which means that such transition is taken as soon as it becomes enabled.

The gate controller functions shown in Listing 1.1 are implemented using C syntax: Square brackets (`[]`) on array variables denote subscripting (element lookup), brackets on `int` type denote the bounded range (array `list` on line

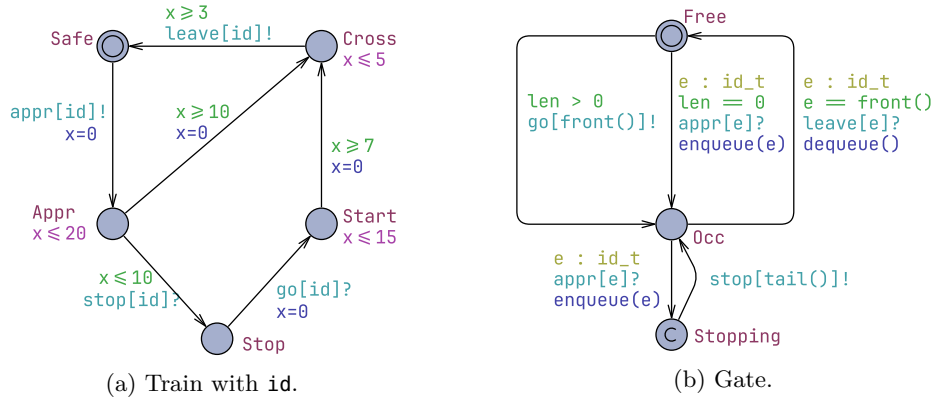


Fig. 9: UPPAAL TA templates for train-gate system.

7 allocates one extra element so that queue overflow could be checked). Pre-increment $++i$ increments the variable i by one before evaluating a statement, and post-increment $i++$ in $\text{list}[i++] = \text{element}$ uses the old i value when assigning element to the list and increments i afterwards. UPPAAL also supports abbreviated assignments, like assignment by subtraction: $\text{len} -= 1$. On line 19, the last element in the list is reset to zero as part of *active variable reduction* – a model-checking optimization technique which brings the variable state to its initial value making it irrelevant when checking if the system state has already been explored (otherwise the state may appear as different and require exploring more states than necessary). Note that UPPAAL implements static analysis for *active clock reduction* and thus the clock variables do not need to be reset, but *discrete variables* are not analyzed as the expressions over them are much more complicated.

The **Train** template is parameterized by a constant identifier **const id_t id** of the bounded integer type **id_t**, which is then filled by automatic instantiation of N trains, allowing easy change of the number of train instances simply by defining a different value for N .

Table 1 summarizes properties one can check on the train-gate model shown in Fig. 9. All of the properties are formulated with a positive expectation in mind and are verified by UPPAAL as satisfied.

2.5 Extensions

It turns out that the same symbolic techniques are also applicable for stopwatch automata where some clocks can be stopped on demand. UPPAAL uses Lagrangian derivative syntax to express constraint over the clock rate in location invariants, e.g. $x' == c$ states that the clock x is stopped when an expression c evaluates to 0 and increases with rate 1 like regular clock when c is 1. Useful to model preemptive schedulers. However, the precise reachability is not decidable

Listing 1.1: UPPAAL code declarations for train-gate model.

```

1  const int N = 6;           ///< the number of trains
2  typedef int[0,N-1] id_t; ///< identifier range is 0..N-1
3
4  chan      appr[N], stop[N], leave[N]; ///< handshake channel arrays
5  urgent chan go[N]; ///< urgent channels synchronize without delays
6
7  id_t list[N+1]; ///< array of N+1 train IDs (one more for overflow check)
8  int[0,N] len;   ///< queue length: integer bounded to [0,N] range
9
10 ///< Adds an element at the end of the queue
11 void enqueue(id_t element) { list[len++] = element; }
12
13 void dequeue() { ///< removes the first element from the queue
14     int i;
15     len -= 1;    ///< shorten the length
16     for (i = 0; i < len; ++i)
17         list[i] = list[i+1]; ///< shift the elements
18     list[i] = 0; ///< reset the garbage for active variable reduction
19 }
20
21 id_t front() { return list[0]; } ///< the first element in the queue
22 id_t tail() { return list[len-1]; } ///< the last element in the queue
23
24 system Train, Gate; ///< process instantiation

```

for stopwatch automata, and hence the resulting reachability analysis is over-approximate in a sense that it includes potentially more states than the actual state space, potentially leading to spurious counter-example traces (signified by “*maybe (un)satisfied*” results).

In this tutorial we use the following extensions not covered by a timed automata definition:

- Urgent locations (with inscribed U) must be left without time passage. The feature avoids additional clock resets and invariants, and thus lowering the cost of symbolic operations.
- Committed locations (with inscribed C) must be left without interruption from other processes. Useful to express atomic sequence of edge transitions.
- **urgent** channels force synchronizing transitions to be taken without time delay: *as soon as* the edge guard becomes enabled.
- **broadcast** channels synchronize one sender with all receivers that are ready to synchronize (could be zero). Allows modeling non-blocking sending, or asynchronous communication.

To ease modeling UPPAAL supports many other features, such as:

Table 1: Properties for model-checking the train-gate system in Fig. 9.

Properties	Description
Validation	
$E \langle \rangle \text{ Gate.0cc}$	Can Gate get to Occupied location?
$E \langle \rangle \text{ Train(1).Cross}$	Can Train get to Crossing ?
$E \langle \rangle \text{ Train(0).Cross and Train(1).Stop}$	Can Train(0) be Crossing and Train(1) Stopped at the same time?
$E \langle \rangle \text{ Train(0).Cross and } (\text{forall}(i:\text{id_t}) i \neq 0 \text{ imply Train}(i).\text{Stop})$	Can Train(0) be Crossing when all other trains are Stopped ?
Safety	
$A[] \text{ forall}(i:\text{id_t}) \text{ forall}(j:\text{id_t}) \text{ Train}(i).\text{Cross} \ \&\& \ \text{Train}(j).\text{Cross} \text{ imply } i=j$	No two trains can be Crossing the bridge at the same time.
$A[] \text{ Gate.list}[N] == 0$	The queue can never overflow beyond N elements.
$A[] \text{ not deadlock}$	There is always an edge-transition available.
Liveness	
$\text{Train}(0).\text{Appr} \rightarrow \text{Train}(0).\text{Cross}$	If Train(0) Approaches it <i>eventually</i> Crosses
$\text{Train}(1).\text{Appr} \rightarrow \text{Train}(1).\text{Cross}$	If Train(1) Approaches it <i>eventually</i> Crosses

- **select** statement to express many similar edges counted by a temporary variable with an integer range.
- Bounded integer variables up to $[-2^{31}; 2^{31} - 1]$ range (the default **int** range is limited to $[-2^{15}; 2^{15} - 1]$ to facilitate early termination in case of run-away counters).
- Arrays of integers, clocks and multidimensional arrays.
- User defined structures (non-homogeneous arrays) of integers, arrays and structures.
- User defined type aliasing using **typedef** statements.
- Reference types to bind variables directly (pointers and dynamic memory are not supported).
- User defined functions with inner scope and **for** and **while** loops.
- Ranged operations:
 - for-loop: **for**($i:\text{int}[0,9]$) $v[i]=i*i$,
 - summation: **sum**($i:\text{int}[0,9]$) $v[i]$,
 - conjunction: **forall**($i:\text{int}[0,9]$) $x[i]'==c[i]$,
 - disjunction: **exists**($i:\text{int}[0,9]$) $v[i]==i*i$.
- **assert** statements to check additional conditions during verification.
- Special functions **before_update** and **after_update** to be executed before transition is taken and after respectively.
- Progress measure to facilitate sweep-line heuristics of exploring state space in waves and reusing verification memory.

- Parameterized timed automata templates and function-call-like process instantiation/construction by supplying arguments to parameters.
- Partial template instantiation creates TA templates from other TA templates by adding additional parameters, or supplying some arguments to another template parameters.
- Automatic instantiation of TA process from TA template counted by a constant parameter from an integer range allows instantiating many processes at once.

UPPAAL implements strong type and range checking to help with imperative code.

Besides TCTL properties, we often want to know the earliest and the latest times some process can complete a task, for this purpose UPPAAL can be instructed to report minimal and maximal values while exploring the model state space:

- **supremum** of an expression, e.g.: $\text{sup}:x,y$ and $\text{sup}\{p\}:x,y$;
- **infimum** of an expression, e.g. $\text{inf}:x,y$ and $\text{inf}\{p\}:x,y$;
- **bounds** infimum and supremum at once: $\text{bounds}:x,y$ and $\text{bounds}\{p\}:x,y$;

where x and y are arbitrary expressions (such as a clock variable) and p is an optional state predicate (such as a process location) which tells when the value of the mentioned expressions are to be recorded.

Clock bounds are particularly useful at extracting earliest and latest time margins for scheduling problems.

3 Priced Timed Automata

So far, the symbolic techniques considered have been restricted to properties reducible to reachability analysis of timed automata. However, in practical settings one is not only interested in whether a system can reach a given target state within timing constraints, but also in quantifying and optimizing the cost of operating the system. To this end, timed automata are extended with a distinguished cost variable. Like clock variables, the cost variable ranges over dense real values, but unlike clocks it may evolve with an arbitrary integer rate (rather than being restricted to rate 1, or 0 in the case of stopwatch automata). Moreover, the cost variable can be increased using integer-valued expressions on transitions. By reusing the same temporal property language, this extension enables the computation of (potentially infinite) witness traces that achieve the system's operational objectives at minimal accumulated cost. As an example Figure 10 provides a priced extension of the simple timed automaton of Fig. 1. Here we see that the cost-variable C grows with different rates depending on which location is being active: we see that **l2** is the most expensive location (with C growing with rate 10), whereas location **l3** is the cheapest (with C growing with rate 1). Moreover we see that C increase by 7 (1), when transitioning from **l3** to **l4** (from **l2** to **l4**).

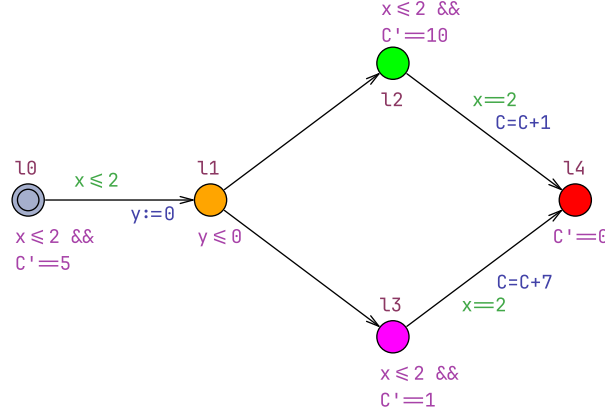


Fig. 10: Simple Priced Timed Automaton.

Motivated by the need for addressing (optimal) usage of resource, the extension of priced timed automata was introduced in 2001 with [4,15] (independently) demonstrating decidability of cost-optimal reachability. Soon after, efficient priced extension of the symbolic zone data structures (so-called priced zones) was implemented in the branch UPPAAL CORA, which combined with a symbolic A* algorithm provides a new generic tool for cost-optimal planning competitive with traditional OR methods such as Mixed-Integer Linear Programming [77]. Later the problem of optimal mean cost infinite runs was considered [23]. Most recently, further improvement for near-optimal planning has been made using Monte Carlo Tree Search [69]. Also, priced zones have been applied for efficient computation of cost-optimal *infinite* plans in terms of long-term average cost [83]. Efficient extrapolation methods for priced timed automata have been introduced [25] and UPPAAL CORA has been used for the optimal planning of missions for battery-powered nano-satellites [17].

3.1 Syntax and Semantics

A priced timed automaton (PTA) consists of a timed automaton, where locations and edges have been decorated with cost rates and cost increments. Below we give the formal definition:

Definition 5. A priced timed automaton $\mathcal{P}$ is a tuple $(L, \ell_0, C, A, E, I, P)$, where (L, ℓ_0, C, A, E, I) is a timed automaton, and

- $P : (L \cup E) \rightarrow \mathbb{N}$ assigns costs to both locations and edges.

The semantics of a PTA is a cost extension of the labeled transition system of the underlying TA given in Definition 2 as defined below:

- Delay-transition $(d \in \mathbb{R}_{\geq 0})$: $\langle \ell, v \rangle \xrightarrow{d, p} \langle \ell, v + d \rangle$ iff $v + d \models I(\ell)$ and $p = P(\ell) \cdot d$

- Action-transition ($\alpha \in \mathbf{A}$): $\langle \ell, v \rangle \xrightarrow{\alpha, p} \langle \ell', v' \rangle$ iff $\langle \ell, g, \alpha, r, \ell' \rangle \in E$, $v \models g$, $v' = v[r]$, $v' \models I(\ell')$, $p = P(\langle \ell, g, \alpha, r, \ell' \rangle)$

Thus the cost of an action transition is simply that of the corresponding edge, and the cost of a delay transition is based on the cost-rate of the location.

Now consider the simple PTA of Fig. 10. We observe the following priced transition sequence leading from the initial state to a final goal state:

$$\begin{aligned} \langle \mathbf{l0}, x = 0, y = 0 \rangle &\xrightarrow{1.9, 9.5} \langle \mathbf{l0}, x = 1.9, y = 1.9 \rangle \xrightarrow{0} \langle \mathbf{l1}, x = 1.9, y = 0 \rangle \\ &\xrightarrow{0} \langle \mathbf{l3}, x = 1.9, y = 0 \rangle \xrightarrow{0.1, 0.1} \langle \mathbf{l3}, x = 2.0, y = 0.1 \rangle \\ &\xrightarrow{7} \langle \mathbf{l4}, x = 2.0, y = 0.1 \rangle \end{aligned}$$

We note that this run reaches the final location $\mathbf{l4}$ with a total cost of $16.6 = 9.5 + 0.1 + 7$. Among all the runs reaching the final location $\mathbf{l4}$, the minimal total cost is given by the following expression:

$$\inf_{0 \leq t \leq 2} \min \left\{ 5t + 10(2 - t) + 1, 5t + (2 - t) + 7 \right\} = 9$$

The expression t in the "inf" expression refers to the delay made at the location $\mathbf{l0}$ (with the rate 5). The two terms in the "min" expression corresponds to choosing between reaching $\mathbf{l4}$ through $\mathbf{l2}$ or through $\mathbf{l3}$. The minimal cost is obtained by the optimal run that leaves $\mathbf{l0}$ immediately, goes to $\mathbf{l2}$, and waits there for 2 time units.

3.2 Symbolic Techniques

Natural questions of a PTA include optimal reachability of a given location, or optimal mean cost of infinite runs. Formally, the associated decision problems, respectively, ask whether the target location can be reached with total weight less than a given threshold, and whether there is an infinite run along which the average weight is less than the given bound.

These problems turn out to be decidable. The main technique used in the algorithms is a refinement of the region equivalence of Sect. 2.2 called corner-point equivalence [15]. Intuitively, for these two optimization problems, optimal schedules will mean staying as long as possible in interesting locations. Corner-point regions extend classical regions with an extra corner of this region, i.e., an integer valuation that belongs to the closure of the region. A corner-point (r, c) represents a clock valuation v that is *close* to the corner c but lies within the region r . The *corner-point automaton* extends the region graph by using corner-point regions and having weighted transitions between these, see Fig. 11. The main observation in the paper is that the cost-optimal path in a PTA actually goes through the corner-points of regions (hence corner-point regions). This is closely related to the fact that the optimal solution of a Linear Programming Problem is obtained in a corner-point of the constraints. The optimal reachability and optimal-ratio problems for PTA actually have the same complexity as plain reachability of TAs as stated below:

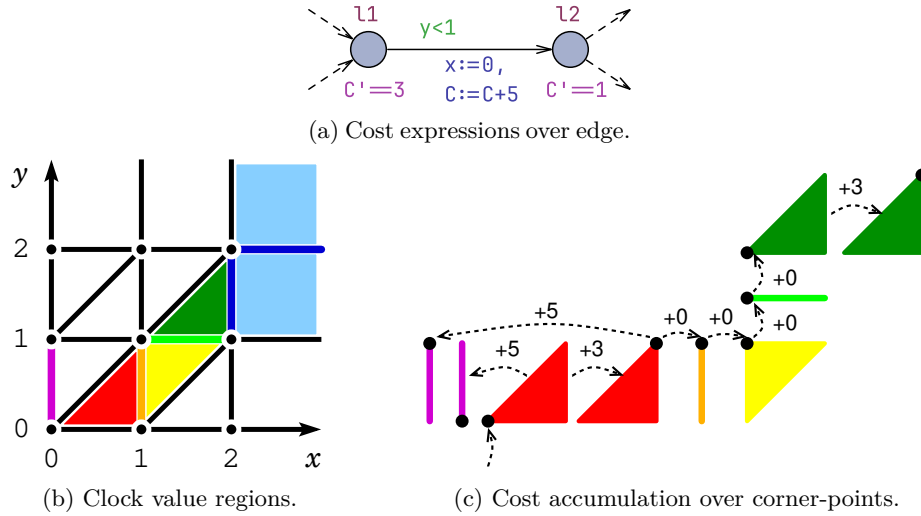


Fig. 11: Corner-point regions with weighted transitions.

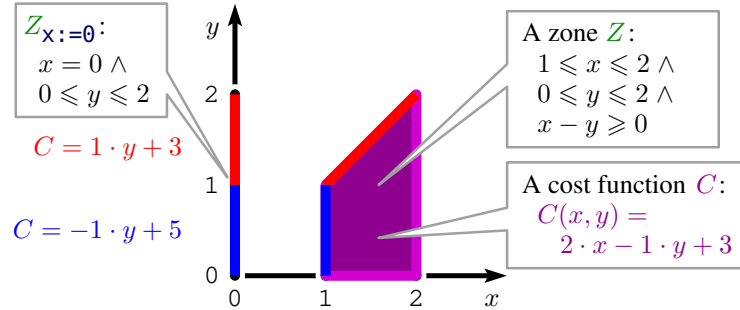


Fig. 12: Priced zones.

Theorem 2 ([15,4,23]). *The optimal-reachability and the optimal-ratio problems for priced timed automata are PSPACE-complete.*

Soon after the above decidability results was proven using corner-point regions, efficient symbolic A^* algorithms was given and implemented in UPPAAL CORA for optimal reachability [77]. Here, the notion of *priced zone* was used as a pair (Z, c) , where Z is a zone and c is an affine, non-negative function over the set of clocks C giving a symbolic encoding of the cost of any valuation in Z . See Fig. 12 for a priced zone and how it is modified by exploring an edge of the PTA. Most recently, we have implemented a Monte Carlo Tree Search method that obtains near-optimal schedules for PTAs with significantly improved efficiency [69]. As for the optimal-ratio problem, it is only now – after 20 years – that we have succeeded in presenting an efficient algorithm using priced zones [83].

4 Stochastic Timed Automata

We have demonstrated how symbolic techniques enable exhaustive exploration of the system's behavior, including the computation of extreme timing bounds and cost values. However, beyond worst-case and best-case analyses, we are also interested in identifying the most likely and practically effective execution times and costs, which more accurately characterize the system's operational performance. As an example, consider the TA in Fig. 13 we can find that **l4** is reachable with $x \in [1, \infty)$ and $y \in [1, \infty)$ using `bounds{P.l4}:x,y` query. However, for the concrete system more information about the stochastic distribution underlying the delays in **l0**, **l2** and $[0,2]$ **l3** may be known, e.g. that the delay in **l0** is uniformly distributed in $[0, 2]$, or the delay in **l3** is governed by an exponential distribution with rate $\frac{1}{2}$. Also the choice between proceeding to **l2** or **l3** may be known to be made probabilistically using weights 1 and 2. Given this additional stochastic information the model actually describes an (infinite-state) stochastic process, a so-called Stochastic Timed Automaton (STA). For the STA of Fig. 13 it can be shown that the mean time of reaching **l4** is about 3.832 ± 0.003 (95% CI⁶) and in **l4** the estimated mean values $x \approx 3.836 \pm 0.003$ and $y \approx 2.935 \pm 0.003$ (95% CI). What is needed is a revision of the concrete semantics of the timed automata extending the set of possible runs with a well-defined probability measure.

In addition, parallel composition of timed automata contains lots of implicit non-determinism and poses a specification challenge: what probabilities should we assign to synchronizing transitions? Here we follow the principles of simplicity and the least surprise: the receiving processes wait passively and thus do not need extra information to be deterministic (they have no control and simply delay the time). Instead, the sending processes dictate when the output takes place, and thus the instance of the entire synchronization, therefore, any non-determinism must be resolved probabilistically on the sender side. The internal (non-synchronizing) transitions can progress independently, thus the same prin-

⁶ Confidence interval with level 95%: 5% chance that the mean is outside the interval.

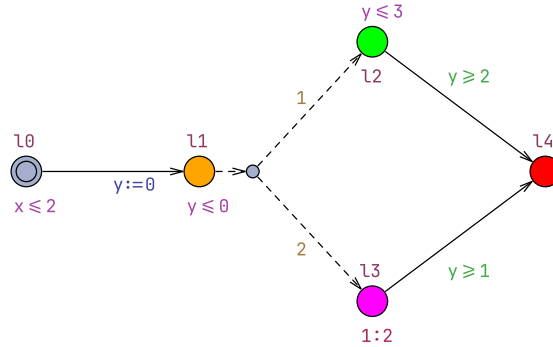


Fig. 13: Simple Stochastic Timed Automaton.

ciple as for sending edges is applied and for convenience we include the internal actions into the set of output actions. In order to avoid conditional probabilities (which are hard to reason with and specify intentions correctly), UPPAAL does not support handshake synchronizations and supports only broadcast synchronizations which are conveniently controlled by a sending process alone.

For overall simulation, we adopt an intuitive race protocol: each process picks their delay independently (requires only local reasoning), the smallest delay wins, and the system state is updated by the proposed delay and transition dictated by the winning process.

Interestingly, the number of simulation runs required does not depend on the complexity of the model (which is not affected by the state space explosion problem), it depends only on the precision (the interval width) and confidence level.

A drawback is that the simulation runs are sampled and finite (not exhaustive as in symbolic methods). Generating simulation runs are relatively cheap and often find witness traces faster than symbolic methods (comparable to a random-depth-first search) if they exist.

The branch UPPAAL SMC (Statistical Model Checking) introduced in 2011 allows performance evaluation based on the notion of stochastic timed automata and stochastic hybrid automata [47,48] through the use of a highly efficient statistical model checking engine. This branch has now been widely applied to analyze a variety of case studies ranging from biological examples [46], scheduling for mixed-critical systems [19,49], controller evaluation for energy-sensitive buildings [50], social-technical attacks in security [51], predictive maintenance [81] as well as performance evaluation of a variety of wireless communication protocols [55]. In addition, the statistical model check engine of UPPAAL SMC is supported by a distributed implementation [36], and allows the statistical model check of a large subset of MITL [37,35]. For a full account of UPPAAL SMC we refer the reader to the tutorial [45].

4.1 Syntax and Semantics

The modeling formalism of UPPAAL SMC is based on a stochastic interpretation and extension of the timed automata used in the classical model checking version of UPPAAL. For individual components, the stochastic interpretation replaces the nondeterministic choices between multiple enabled transitions by probabilistic choices (that may or may not be user defined). Similarly, the nondeterministic choices of time delays are refined by continuous probability distributions. In this work, we assume that these delay distributions are given either as *uniform distributions* in cases with time-bounded delays or as *exponential distributions* (with user-defined rates) in cases of unbounded delays. Below we give the formal definition of a stochastic timed automaton (STA).

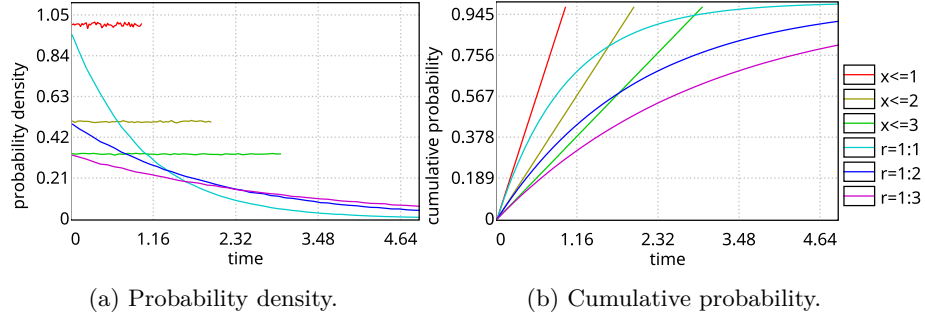


Fig. 14: Distributions obtained from simulations with invariants $x \leq 1$, $x \leq 2$, $x \leq 3$, rates $1:1$, $1:2$, $1:3$.

Definition 6. A stochastic timed automaton $\mathcal{S}$ is a tuple $(L, \ell_0, C, \mathbf{A}, E, I, P, W)$, where $(L, \ell_0, C, \mathbf{A}, E, I)$ is a timed automaton, and

- $P : L \rightarrow \mathbb{Q}_{>0}$ provides the rates of exponential distributions for locations, and
- $W : E \rightarrow \mathbb{N}_{>0}$ provides the weights assigned to given edges.

Moreover, the action set $\mathbf{A}$ is divided into input actions $\mathbf{A}^i$ and output actions $\mathbf{A}^o$ such that $\mathbf{A} = \mathbf{A}^i \cup \mathbf{A}^o$ and $\mathbf{A}^i \cap \mathbf{A}^o = \emptyset$.

In addition, it is assumed that an STA is input-enabled, thus for all states (ℓ, v) and for all input actions $i \in \mathbf{A}^i$, there exists a transition $(\ell, v) \xrightarrow{i} (\ell', v')$. Also the STA must be deterministic, so that whenever $(\ell, v) \xrightarrow{a} (\ell', v')$ and $(\ell, v) \xrightarrow{a} (\ell'', v'')$, then $(\ell', v') = (\ell'', v'')$.

Semantically, an STA specifies the density of delay transitions and the probability of discrete output-labeled transitions for a given state $s = (\ell, v)$ by the following functions:

$$\textbf{Delay density: } \mu_s : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}, \text{ where } \int_t \mu_s(t) dt = 1, \text{ and} \quad (1)$$

$$\textbf{Output probability: } \gamma_s : \mathbf{A}^o \rightarrow [0, 1], \text{ where } \sum_{o \in \mathbf{A}^o} \gamma_s(o) = 1. \quad (2)$$

For the given STA, μ_s is either a *uniform* or an *exponential* distribution based on the invariant of the current location ℓ (bounded delay) and its rate annotation $P(\ell)$ (unbounded delay). Figure 14 shows probability distributions for various invariant and rate expressions. We write $G_\ell = \{g \mid (\ell, g, o, r, \ell') \in E \wedge o \in \mathbf{A}^o\}$ to denote all the guards of output-labeled edges from ℓ and $E_\ell = \bigvee_{g \in G_\ell} g$ to denote the disjunction of the guards of G_ℓ . Now, the minimum possible delay $d_{\min}(s)$ in state $s = (\ell, v)$ required to enable some output action o out of ℓ is defined as $d_{\min}(\ell, v) = \inf\{d \in \mathbb{R}_{\geq 0} \mid v + d \models E_\ell\}$. Similarly, the maximum delay is defined as $d_{\max}(\ell, v) = \inf\{d \in \mathbb{R}_{\geq 0} \mid v + d \models I(\ell)\}$. If $d_{\max}(\ell, v) < \infty$ the delay density

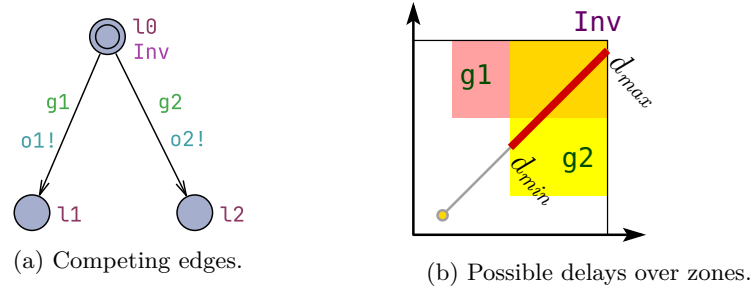


Fig. 15: Stochastic semantics for two competing edges.

$\mu_{(\ell,v)}$ is defined as a uniform distribution on the interval $[d_{min}(\ell, v), d_{max}(\ell, v)]$. However, if $d_{max}(\ell, v) = \infty$, the delay density $\mu_{(\ell,v)}$ is defined by an exponential distribution with rate $P(\ell)$. Fig. 15 illustrates the above definition of μ_s for an STA with three locations, two outgoing edges, two clocks and a state $s = (\ell, v)$.

The output probability distribution γ_s is given by the relative weights of the edges enabled from s ⁷. We denote by $E_{(\ell,v)} = \{(\ell, g, o, r, \ell') \in E \mid o \in A^o \wedge v \models g\}$ the set of output-labeled edges that are enabled in (ℓ, v) . Also we denote by $O_{(\ell,v)} = \{o \in A^o \mid (\ell, g, o, r, \ell') \in E\}$ the set of output actions that are enabled in (ℓ, v) . Finally, for an edge $e = (\ell, g, o, r, \ell') \in E$ we denote by $O(e)$ the output o . Now:

$$\gamma_{(\ell,v)}(o) = \begin{cases} 0, & \text{if } o \notin O_{(\ell,v)} \\ \frac{W(e)}{\sum_{e' \in E_{(\ell,v)}} W(e')} & \text{where } e \in E_{(\ell,v)} \text{ and } o \in O(e) \end{cases}$$

Consider the stochastic timed automaton of Fig. 13. Here all edges are assumed to be labeled with the same single output actions, which for simplicity is omitted here. In **l0** a delay is *uniformly* chosen in the interval $[0, 2]$ due to the invariant $x \leq 2$. In **l1** no delay is possible, but a *discrete probabilistic choice* is made between transitioning to **l2** respectively **l3** with probability $\frac{1}{3}$ respectively $\frac{2}{3}$ due to weights **1** and **2** respectively. In location **l2** a delay in the interval $[2, 3]$ will be *uniformly* chosen before reaching **l4**. Location **l3** does not have an upper bound for delays due to no invariant, thus the delay is chosen starting with $y \geq 1$ and using *exponential distribution* with probability density function $\text{PDF}(\lambda, t) = \lambda e^{-\lambda t}$ with $\lambda = \frac{1}{2}$ as specified with rate expression **1:2**. A useful intuition behind the exponential distribution rate λ is that the expected mean time is exactly $\frac{1}{\lambda}$, that is the greater the rate the shorter the delays and the more eager the process is to leave the location, so the mean time of leaving **l3** is $1 + \frac{1}{\lambda} = 3$.

For the purpose of simplification, we shall in the exposition below restrict to STAs with only output actions. Such STAs are completely stochastic, and the delay density and output probability functions induce a probability measure

⁷ Note that the enabled edges uniquely determine the output due to the determinism.

on measurable sets of runs from a sigma-algebra generated by cylinder sets of the form $\pi = I_1 o_1 I_2 o_2 \dots o_n$, where I_i are open intervals of reals given by rational end points, and o_i are output actions. Semantically, the set of cylinders associated with π is the set:

$$\Pi = \{t_1 o_1 t_2 o_2 \dots t_n o_n \mid \forall i. t_i \in I_i\}$$

Given an STA $\mathcal{S}$, the following equation inductively defines the probability of the set of runs in Π as ($n > 0$):

$$\mathbb{P}_{\mathcal{S}}(s, I_1 o_1 I_2 o_2 \dots I_n o_n) = \int_{t \in I_1} \mu_s(t) \cdot \gamma_{s^t}(o_1) \cdot \mathbb{P}_{\mathcal{S}}(s^{t, o_1}, I_2 o_2 \dots I_n o_n) dt \quad (3)$$

and where $\mathbb{P}_{\mathcal{S}}(s, \epsilon) = 1$. Here, ϵ denotes the empty sequence covering the base-case when $n = 0$. Here, we use the notation s^t and $s^{t, o}$ to denote the unique state obtained from s after a delay of time t or a delay of time t followed by action o . Now the sigma-algebra of measurable sets of runs is obtained by closing the set of generating cylinder sets under complement and countable union.

Now, reconsider the STA from Fig. 13. Let $\Pi_{\leq 3}^{\mathbf{l2}}$ denote the set of runs that via location $\mathbf{l2}$ ends in $\mathbf{l4}$ within 3 time units, i.e.:

$$\begin{aligned} \Pi_{\leq 3}^{\mathbf{l2}} = \{ & \langle \mathbf{l0}, x = 0, y = 0 \rangle \xrightarrow{t_1} \langle \mathbf{l1}, x = t_1, y = 0 \rangle \rightarrow \\ & \langle \mathbf{l2}, x = t_1, y = 0 \rangle \xrightarrow{t_2} \langle \mathbf{l4}, x = t_1 + t_2, y = t_2 \rangle \mid \\ & t_1 + t_2 \leq 3 \wedge t_1 \leq 2 \wedge 1 \leq t_2 \leq 3 \} \end{aligned}$$

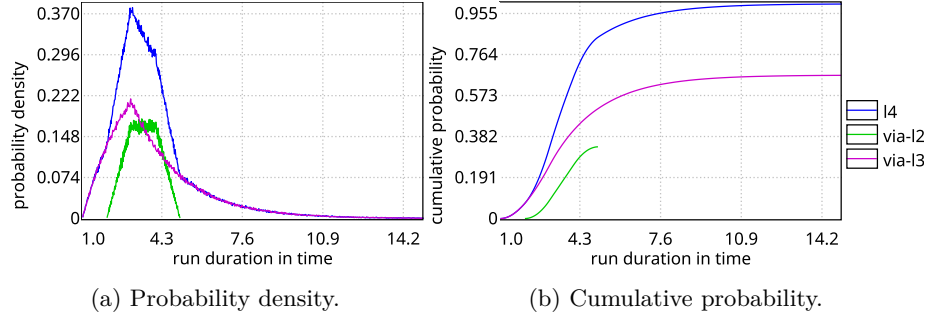
Now, the probability of $\Pi_{\leq 3}^{\mathbf{l2}}$ is given by the following nested integral in the style of (3):

$$\mathbb{P}_{\mathcal{S}}(\Pi_{\leq 3}^{\mathbf{l2}}) = \int_{t_1=0}^{t_1=2} \frac{1}{2} \cdot \left(\frac{1}{3} \cdot \int_{t_2=2}^{t_2=3-t_1} 1 dt_2 \right) dt_1 = \frac{1}{12}$$

In Figure 16 we consider the total probability of reaching $\mathbf{l4}$ within 15 time units, providing probability density and cumulative probability for obtained using the statistical model checking engine of UPPAAL to be explained in the following section.

4.2 Networks of Stochastic Timed Automata

In the previous section, we have focused on a single STA with only output actions. However, a full model consists of a *closed network* of STAs, where the stochastic behavior is obtained through a race of producing outputs between components. Here, it is assumed that the set of actions A is identical for all components, but that the output actions are disjoint, i.e. $A_i^o \cap A_j^o = \emptyset$ whenever $i \neq j$. Being a closed system is reflected by the requirement that $\cup_i A_i^o = A$. Now each component $\mathcal{S}_i$ chooses independently of the others at which time t_i and which action o_i to output – of course using the delay density and output

Fig. 16: Probability estimate over time: $\Pr[\leq 15; 1000000] (< P.l4)$

probability functions of the component. In our race-semantics of the network, the component $\mathcal{S}_i$ with the minimal t_i wins, and will broadcast the chosen output o_i (recall that all components are input-enabled avoiding o_i to be blocked).

Now, for any state $\mathbf{s} = (s_1, \dots, s_n)$ of the network, we define the probability of the set of runs from $\mathbf{s}$ within a cylinder $\Pi = (I_1 o_1 \dots I_n o_n)$ as follows:

$$\mathbb{P}_{\mathcal{S}}(s, I_1 o_1 I_2 o_2 \dots I_n o_n) = \int_{t \in I_1} \mu_{s_c}^c(t) \cdot \left(\prod_{j \neq c} \int_{\tau > t} \mu_{s_j}^j(\tau) d\tau \right) \cdot \gamma_{s_c}^c(o_1) \cdot \mathbb{P}_{\mathcal{S}}(s^{t, o_1}, I_2 o_2 \dots I_n o_n) dt \quad (4)$$

Note that this extends the definition of Eq. 3 to the setting of several components. In the above equation, it is assumed that c is the index of the component for which o_1 is an output, $o_1 \in A_c^o$. In the above equation $\mu_{s_c}^c(t)$ is the density that component c will delay t time-units. What follows in the equation is the probability that all other components will choose a delay later than t .

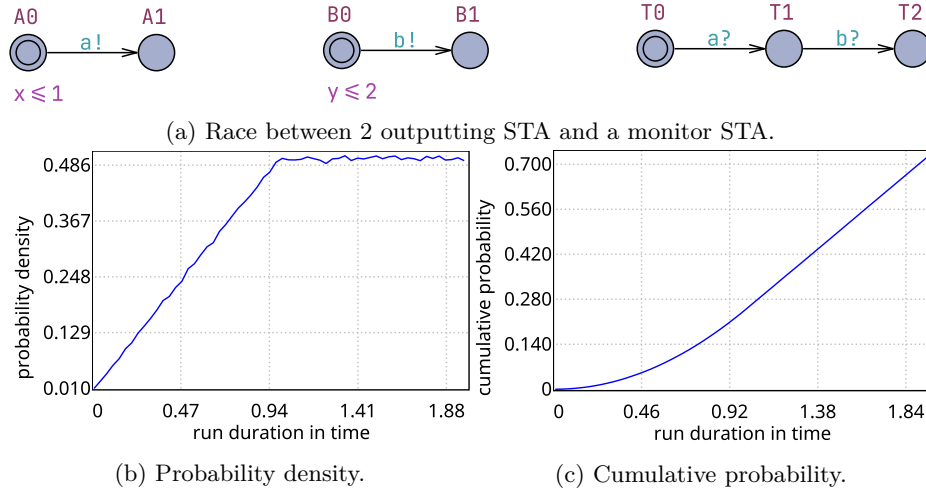
Now consider the network of three STA in Fig. 17. Here, the component **A** will output **a** after a uniformly chosen delay from $[0, 1]$. Similarly **B** will output **b** after a uniformly chosen delay from $[0, 2]$. The component **T** is just a monitoring component, but will end in location **T2** if **a** is sent before **b**. Now, using Equation 4 the probability that the monitor will end in **T2** is given by the following nested integral expression:

$$\mathbb{P}(\Diamond \mathbf{T2}) = \int_{t_a=0}^1 1 \cdot \int_{t_b=t_a}^2 \frac{1}{2} dt_b dt_a = \frac{3}{4}$$

In Figures 17b and 17c we show the plots for the probability density and cumulative probability obtained using the statistical model checking engine of UPPAAL to be explained in the following section.

4.3 Statistical Model Checking

Using the cylinder-based stochastic semantics of STAs it can be shown that time-bounded reachability problems have well-defined probabilities as illustrated with

Fig. 17: Probability estimate of $\Pr[\leq 2; 1000000] (\langle \rangle T.T2)$

the simple STA of Fig. 13. Actually, for any MITL [3] property one can show that the set of satisfying (infinite) runs have a well-defined probability. For acyclic STA's these probabilities may be computed using nested integrals as illustrated on the simple STA. This approach was pursued in [80] guaranteeing exact probabilities but leading to an exploding complexity in the length of runs of the STA. Instead, UPPAAL SMC offers the use of a *statistical model checking* engine [48], where the probability of the queried property ϕ is estimated through repeated generation of random runs of the STA according to its stochastic semantics, see Fig. 18. For each run, π , it is checked whether π satisfies ϕ or not. Now core statistical algorithms will allow us to estimate $\mathbb{P}_{\mathcal{S}}(\phi)$ with a confidence interval with a given error margin ε and confidence level μ . Alternatively, given a threshold probability p one may dismiss (or not) a hypothesis of the form $\mathbb{P}_{\mathcal{S}}(\phi) \geq p$ with a specified strength α . We refer to the tutorial [45] for more information.

4.4 Probabilistic Specification

Given an STA model UPPAAL SMC supports estimation (hypothesis) testing or properties of a fragment of MITL [3] using the queries of Listing 1.2. Let us now detail the five different UPPAAL SMC queries:

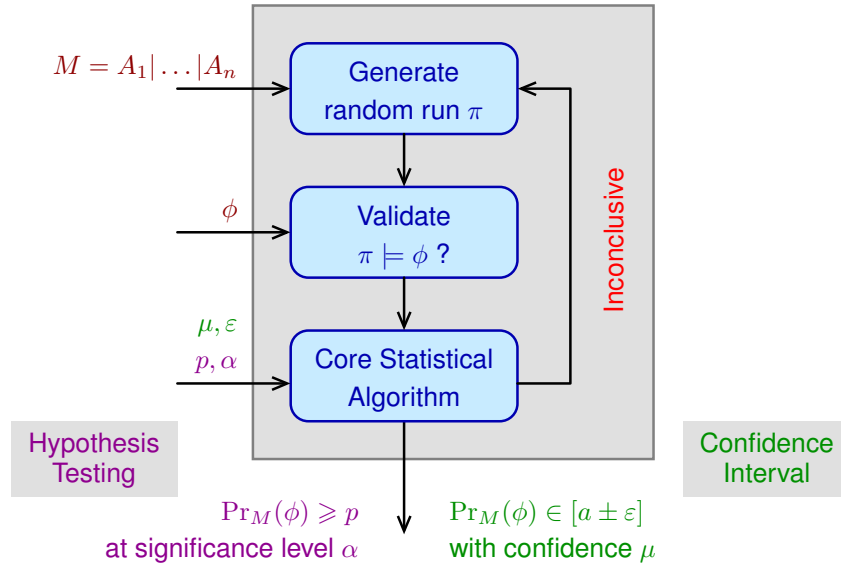


Fig. 18: Statistical model checking framework.

Listing 1.2: Syntax of UPPAAL SMC queries.

```

1 simulate[BoundExpr;Runs] { expr, ... } : SuccRuns : Pred
2 Pr[BoundExpr;Runs](PathFormula)
3 Pr[BoundExpr](PathFormula) <=> PrBound
4 Pr[BoundExpr](PathFormula1) ≥ Pr[BoundExpr](PathFormula2)
5 E[BoundExpr;Runs](aggr: Expr)

```

Simulate query (line 1) requests to generate a number of stochastic runs and record the trajectories of specified state expressions:

- **BoundExpr** can be one of the following forms:
 - **Clock** $\leq$ **Bound** is an invariant controlling how long the simulated run should be. **Clock** is an optional **clock** variable (an absolute time is used if omitted) and **Bound** is an upper bound (usually an integer constant).
 - **#** $\leq$ **EdgeCount** specifies that the run should contain at most **EdgeCount** edge transitions.
- **Runs** is an optional positive integer denoting the number of runs (1 is used if omitted).
- **expr, ...** is a comma-separated list of state expressions whose values are collected along the run.
- **SuccRuns** is an optional integer specifying the maximum number of runs satisfying the **Pred**.

- **Pred** is an optional state predicate specifying that only the runs satisfying $\langle \rangle$ **Pred** should be considered. If the predicate is supplied, then the query also computes a confidence interval $(1 - \alpha \text{ CI})$ for probability of $\langle \rangle$ **Pred** being satisfied.

Probability query (line 2) estimates a confidence interval $(1 - \alpha \text{ CI})$ using the Clopper-Pearson algorithm [41] for probability of some **PathFormula** being satisfied over a number of runs:

- **Runs** can be omitted, in which case a sequential algorithm will be used: generate runs until the estimated confidence interval $(1 - \alpha \text{ CI})$ width falls below 2ε where ε is the error margin parameter around the probability mean.
- **PathFormula** can be:
 - $\langle \rangle$ **Pred** means that the run satisfies the state predicate **Pred** eventually.
 - $[]$ **Pred** means that the entire run has to satisfy the state predicate **Pred**.

Hypothesis Testing query (line 3) tests a hypothesis where a probability is lower than $\text{PrBound} - \delta_-$ or greater than $\text{PrBound} + \delta_+$ with false positive rate α and false negative rate β using Wald's algorithm [84]:

- $\langle \Rightarrow \rangle$ is either $\leq$ or $\geq$.
- **PrBound** is a floating point constant denoting the probability bound.

Comparison query (line 4) compares two probabilities without estimating them. UPPAAL compares the probabilities over the time/cost/transitions and yields 0 (**false**), 1 (**true**) and 0.5 (not enough evidence).

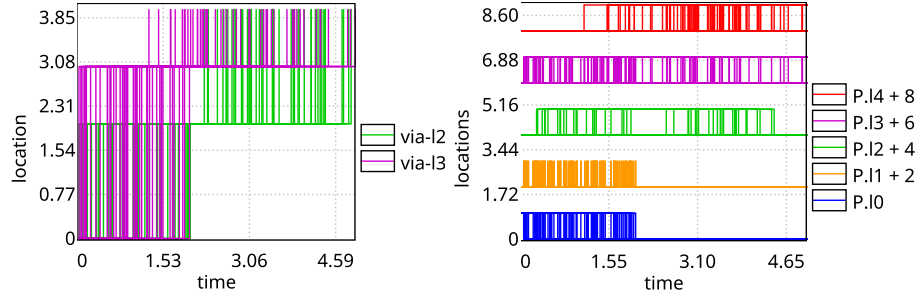
Expected Value query (line 5) computes a confidence interval $(1 - \alpha \text{ CI})$ of minimum or maximum value of an expression over specified number of runs by assuming that the value follows Student- t distribution:

aggr is an aggregating function of **Expr** value over an entire run which can be either **min** for the minimum, or **max** for maximum.

Table 2 provides a number of statistical model-checking queries on the simple STA of Fig. 13.

4.5 Stochastic Priced Timed Automata

Given that UPPAAL SMC is based on a simulation engine, extensions to more expressive, even undecidable, modeling formalisms, e.g. hybrid automata, are possible and are supported in UPPAAL [42,45]. Here, we consider the Stochastic Priced Timed Automata (SPTA) of Fig. 21. Like in a PTA we see the addition of a cost variable **C** that is growing with different rates in different locations. Formally, **C** may be seen as a random variable which given a maximal run (ending in 14) will have a given value. In Fig. 22 we see the probability distribution over values of **C**. We may also obtain the expected value of **C** within a given time-bound using the query of Table 3.



(a) $\text{simulate}[\leq 5;1000]\{P.l_1+P.l_2*2+ P.l_3*3+P.l_4*4\} : 50 : \text{via}==\{2,3\}$ (b) $\text{simulate}[\leq 5;100]\{P.l_4+8,P.l_3+6, P.l_2+4,P.l_1+2,P.l_0\}$

Fig. 19: Time spent in locations using `simulate` query.

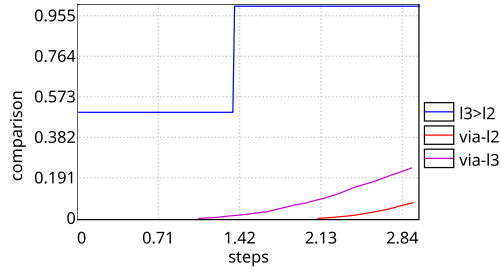


Fig. 20: Probabilities of reaching l_4 via l_2 and l_3 and their comparison.

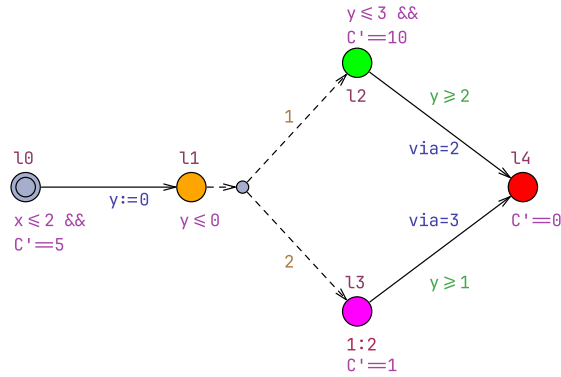


Fig. 21: Simple Stochastic Priced Timed Automaton.

Table 2: Statistical model checking query examples for simple STA (Fig. 13).

Query	Description
<code>simulate[≤5;200]{P.l1+ P.l2*2+P.l3*3+P.l4*4}</code>	How does P location evolve over time? The result is in Fig. 19a.
<code>simulate[≤5;100]{P.l4+8, P.l3+6,P.l2+4,P.l1+2,P.l0}</code>	Where does P spend time? The result is in Fig. 19b.
<code>Pr[≤3](<> P.l4)</code>	What is the probability of P reaching l4 within 3 time units? Result is 0.329 ± 0.049 (95% CI) (used 376 runs). Probability over longer time is shown in Fig. 16.
<code>Pr[≤3](<> P.l4) ≤ 0.5</code>	Is the probability of P reaching l4 within 3 t.u. less than 0.5? The result is ≤ 0.49 (95% conf.) (used 180 runs).
<code>Pr[≤3](<> via==3) ≥ Pr[≤3](<> via==2)</code>	Is the probability of P reaching l4 greater by going through l3 than l2 ? The result is inconclusive up to 1.36 t.u. (blue line is at 0.5 in Fig 20) and true (blue line is at 1) with 95% confidence (used 4546 runs).

Table 3: UPPAAL SMC queries for Simple SPTA in Fig. 21.

Query	Description
<code>E[≤3;1000](max: C)</code>	What is the estimated maximum C within 3 t.u.? The result is 12.30 ± 0.54 (95% CI) and distribution is shown in Fig. 23.

4.6 Performance Analysis of the Train Gate

In order to adopt the train-gate model from Fig. 9 for performance analysis we need to: 1) determine delay probability distribution for each location, 2) ensure the input-enabledness, and 3) determinize input edges. Most **Train** locations (**Appr**, **Start**, **Cross**) already use clock invariants, therefore a uniform delay distribution is assigned which follows the principle of least surprises and thus no changes are needed. Location **Stop** waits for inputs, thus no delay distribution is needed. Location **Safe** has no invariant, therefore the delay is unbounded and an exponential delay distribution is assigned, which requires an exponential rate to be defined. In order to guarantee input-enabledness by simple construction, we change the handshake **channel** types into non-blocking **broadcast channel** types. Inputs are handled by a unique outgoing edge per location and channel, therefore no changes are needed for input determinization. Finally **Gate** from Fig. 9b can be used as it is: **Occ** waits for inputs deterministically, **Free** is committed (neither delay nor interruption are permitted) and **Free** either waits for **appr[e]?** synchronization or takes output transition *as soon as* **len>0** is true since **go** is an array of **urgent broadcast channels** – either way no delay distribution is needed. Fig. 24 shows the resulting stochastic extension of the train timed automaton, where trains approach with a delay governed by an exponential

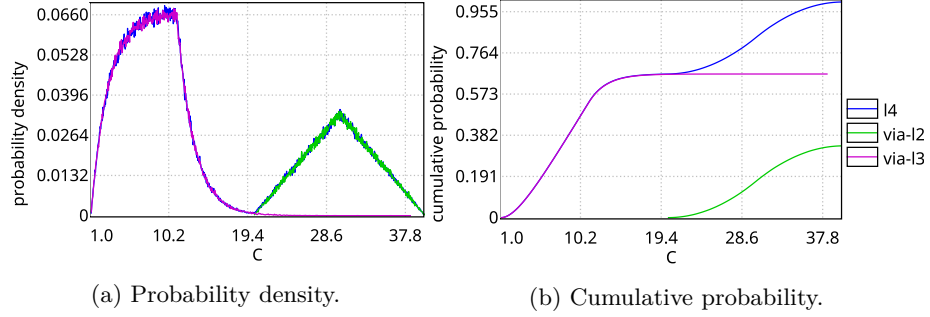
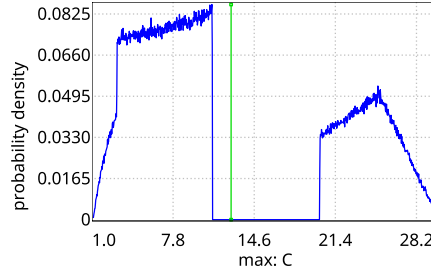

 Fig. 22: Probability estimate over cost C : $\Pr[C \leq 100; 1000000] (<> P.l4)$


Fig. 23: Expected max. cost distribution.

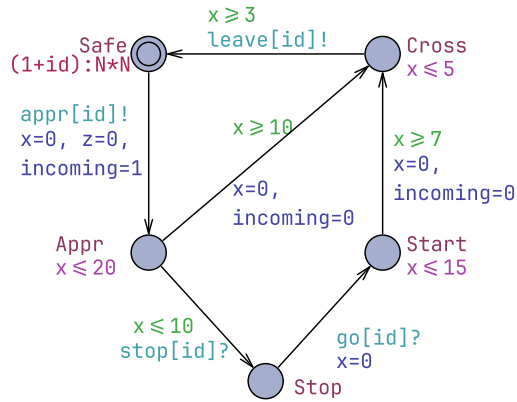


Fig. 24: Stochastic timed automaton for a train.

distribution of rate $\frac{1+id}{N \cdot N}$ at **Safe**, where **id** is the train identifier, and **N** is the number of trains. The use of identifier **id** in the exponential rate expression makes every train to behave in a unique way. Specifically **Train(0)** is the slowest and **Train(N)** is the fastest to leave location **Safe**.

In order to track the waiting time we add an additional local clock **z** and a local boolean variable **incoming**.

Table 4 shows the performance queries. The first two queries demonstrate

Table 4: Performance queries for train gate shown in 24.

Property	Result
Is waiting time always less than 200?	
$A[] \text{ Train}(i).\text{incoming} \text{ imply } \text{Train}(i).z \leq 200$	For all $i \in 0, \dots, 5$: yes
What is the worst waiting time?	
$\sup\{\text{Train}(i).\text{incoming}\} : \text{Train}(i).z$	For all $i \in 0, \dots, 5$: 120
Estimated probability of train entering Crossing	See Fig. 25
$\Pr[\leq 100; 1000](\langle \rangle \text{ Train}(0).\text{Cross})$	0.60 ± 0.03 (95% CI)
$\Pr[\leq 100; 1000](\langle \rangle \text{ Train}(1).\text{Cross})$	0.84 ± 0.02 (95% CI)
$\Pr[\leq 100; 1000](\langle \rangle \text{ Train}(2).\text{Cross})$	0.93 ± 0.02 (95% CI)
$\Pr[\leq 100; 1000](\langle \rangle \text{ Train}(3).\text{Cross})$	0.97 ± 0.01 (95% CI)
$\Pr[\leq 100; 1000](\langle \rangle \text{ Train}(4).\text{Cross})$	0.99 ± 0.006 (95% CI)
$\Pr[\leq 100; 1000](\langle \rangle \text{ Train}(5).\text{Cross})$	0.99 ± 0.005 (95% CI)
Expected maximum waiting time to enter Crossing	See Fig. 26
$E[\leq 1000; 100000](\max:\text{Train}(0).z * \text{Train}(0).\text{incoming})$	85.36 ± 0.03 (95% CI)
$E[\leq 1000; 100000](\max:\text{Train}(1).z * \text{Train}(1).\text{incoming})$	86.78 ± 0.03 (95% CI)
$E[\leq 1000; 100000](\max:\text{Train}(2).z * \text{Train}(2).\text{incoming})$	87.65 ± 0.03 (95% CI)
$E[\leq 1000; 100000](\max:\text{Train}(3).z * \text{Train}(3).\text{incoming})$	88.32 ± 0.03 (95% CI)
$E[\leq 1000; 100000](\max:\text{Train}(4).z * \text{Train}(4).\text{incoming})$	88.84 ± 0.03 (95% CI)
$E[\leq 1000; 100000](\max:\text{Train}(5).z * \text{Train}(5).\text{incoming})$	89.27 ± 0.03 (95% CI)

that the model can still be analyzed by the symbolic engine (the model changes are compatible), but the results are quite limited: “yes”/“no” answer means that the exact time bound can be found though many queries via manual binary search (inefficient), and a more efficient supremum query yields an upper bound (120 time units) at once, but that this upper bound is an extreme/unlikely case. Meanwhile, SMC engine estimates the most likely, average performance: the *probability of arriving to Crossing within 100 time units* (results vary a lot across trains as expected: higher arrival rates lead to higher probability of reaching destination), and the *maximum waiting times* vary for every **Train** as expected due to different rates of interruptions by arriving trains, but only slightly. In addition to the mean value estimates, UPPAAL SMC also reports probability distributions of the estimated quantities as collected in Fig. 25 and 26 which reveal a complicated competition at the gate while the waiting times are relatively undisturbed and approach a family of normal distributions (likely a sum of uniform delays).

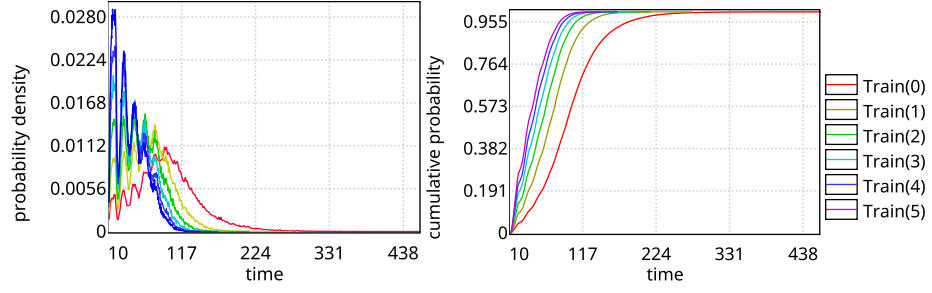


Fig. 25: Estimated probability of each train entering the crossing:
 $\text{Pr}[\leq 1000; 100000](\langle \rangle \text{Train}(i).\text{Cross})$.

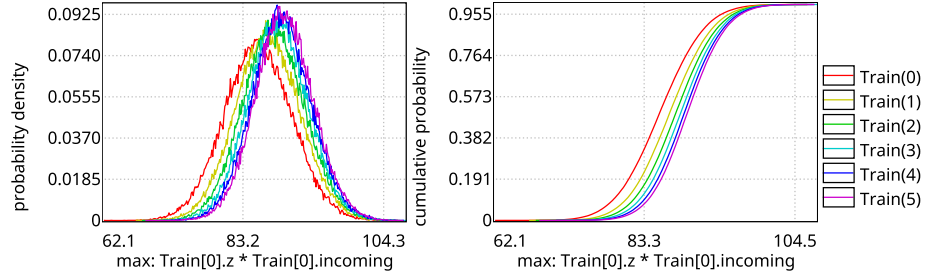


Fig. 26: Estimated waiting time for each train.

5 Timed Game Automata

In 2005—encouraged by suggestions from Thomas A. Henzinger—the branch UPPAAL TIGA was released, enabling the synthesis of control strategies from timed games, i.e., two-player games played on timed automata [8,61]. This branch implements an efficient symbolic on-the-fly algorithm for synthesizing winning strategies for reachability, safety, and Büchi objectives, while also accounting for partial observability [38].

UPPAAL TIGA advances the development of control programs for embedded systems. Instead of manually constructing a controller, subsequently checking and correcting it, it is possible to fully automatically generate correct-by-construction control programs. Moreover, the customization of a control program becomes straightforward, requiring only modification of the objective.

5.1 Syntax and Semantics

We consider two-player timed games, in which transitions are partitioned into controllable and uncontrollable (i.e. under the control of an environment). The problem is then to synthesize a strategy telling when to take which (enabled) controllable transitions in order that a given objective is guaranteed regardless of the behavior of the environment:

Definition 7. A timed game $\mathcal{G}$ is a tuple $(L, \ell_0, C, A_c, A_u, E, I)$ with $A_c \cap A_u = \emptyset$ for which the tuple $\mathcal{A}_{\mathcal{G}} = (L, \ell_0, C, A_c \cup A_u, E, I)$ is a timed automaton. We require this automaton to be deterministic. Edges with actions in A_c are said to be controllable and edges with actions in A_u are said to be uncontrollable.

In the following, we shall assume a set F of accepting states (specified syntactically as a union of symbolic states). A *strategy* in a timed game $\mathcal{G}$ provides instructions on which controllable edges are permissible to take, and also whether it is permissible to wait in any given state. A (memoryless, non-deterministic) strategy is a mapping σ from the states $\text{St}_{\mathcal{A}_{\mathcal{G}}} = L \times (C \rightarrow \mathbb{R}_{\geq 0})$ of the underlying timed automaton $\mathcal{A}_{\mathcal{G}}$ to non-empty sets of “instructions”, i.e.

$$\sigma : \text{St}_{\mathcal{A}_{\mathcal{G}}} \rightarrow \mathcal{P}(A_c \cup \{\lambda\}) \setminus \emptyset$$

with $\lambda \notin A_c \cup A_u$ indicating “wait”. For a strategy σ a run in $\mathcal{A}_{\mathcal{G}}$:

$$\rho = (\ell_0, v_0) \xrightarrow{\alpha_0} (\ell_1, v_1) \xrightarrow{\alpha_1} \dots (\ell_n, v_n) \xrightarrow{\alpha_n} (\ell_{n+1}, v_{n+1})$$

is an *outcome* of σ if the following holds for all $i = 1 \dots n$:

- if $\alpha_i \in A_c$, then $\alpha_i \in \sigma(\ell_i, v_i)$
- if $\alpha_i = d_i \in \mathbb{R}_{\geq 0}$, then $\lambda \in \sigma(\ell_i, v_i + d)$ for all $d < d_i$.

A run is *maximal* if it stops in an accepting (or rejecting) location, or no controllable actions are available at its end. The aim of reachability games is to find strategies all of whose maximal outcomes end in an accepting location; dually, the aim of a safety game is to find strategies all of whose (not necessarily maximal) outcomes avoid rejecting locations. A strategy σ_1 is *more permissive* than a strategy σ_2 , if $\sigma_1(s) \subseteq \sigma_2(s)$ for any state s .

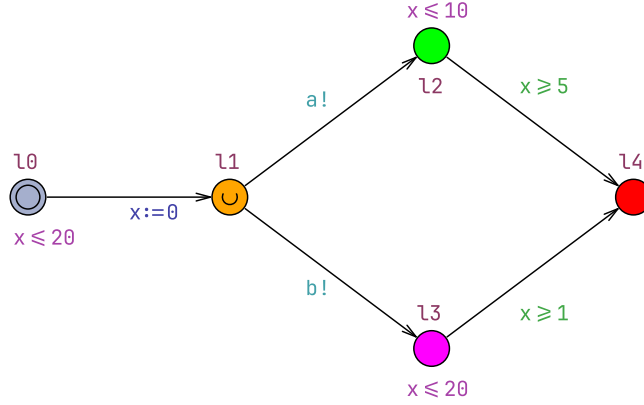


Fig. 27: Simple Timed Game.

Definition 8. A strategy is said to be winning for a reachability game if all of its maximal outcomes end in an accepting location. A strategy is said to be winning for a safety game if none of its outcomes visits a rejection location.

Consider the timed game in Fig. 27 with two controllable actions, a and b . We assume that in addition to the single clock x there is an additional clock y that is never tested or reset; thus y measures the total amount of time elapsed from the initial state. Now, let the set of *unsafe* states F be specified by the following specification: $l4 \wedge (y \geq 31)$. For a winning safety strategy, this essentially requires that $l4$ is reached at the latest after 31 time units. The following two strategies σ_1, σ_2 are winning with respect to the safety game for F :

$$\sigma_1(l1, x = 0, y = d) = \{a\} \quad \text{if } d \leq 20$$

$$\sigma_2(l1, x = 0, y = d) = \begin{cases} \{a, b\} & \text{if } d < 11, \\ \{a\} & \text{if } d \geq 11. \end{cases}$$

We observe that σ_2 is more permissive than σ_1 . Note that the action b is unsafe in $l1$ when $d = 11$, as this would make the unsafe state $(l3, x = 20, y = 31)$ reachable.

Theorem 3. [8,61] *Reachability and safety timed games are decidable and EXPTIME-complete.*

Theorem 4. [7,31,73] *Optimal-time reachability timed games are decidable and EXPTIME-complete.*

It can be proven that memoryless and region-based strategies suffice for reachability, time-optimal and safety timed games. Moreover, for safety timed games, which includes time-bounded reachability timed games, there is a maximum permissive safety strategy.

5.2 Symbolic Techniques for Strategy Synthesis

The decidability and complexity results for timed games of Theorems 3 and 4 are region-based. However, an efficient on-the-fly zone-based algorithm has been developed [12,39] and implemented in UPPAAL TIGA. The algorithm is based on the following operations:

$$\begin{aligned} \text{cPred}(X) &= \{ q \in Q \mid \exists q' \in X. q \xrightarrow{c} q' \} && (\text{controllable}) \\ \text{uPred}(X) &= \{ q \in Q \mid \exists q' \in X. q \xrightarrow{u} q' \} && (\text{uncontrollable}) \\ \text{Pred}_t(X, Y) &= \{ q \in Q \mid \exists t, q^t \in X \text{ and } \forall s \leq t. q^s \in Y^C \} && (\text{time}) \end{aligned}$$

Here $\text{cPred}(X)$ denotes all the controllable predecessors of a given state set X , that is the set of state (ℓ, v) , where $(\ell, v) \xrightarrow{d, c} (\ell', v')$ for some delay d , controllable action c with $(\ell', v') \in X$. Similarly $\text{uPred}(X)$ denotes the set of uncontrollable predecessors of X . Finally, $\text{Pred}_t(X, Y)$ denotes all states that by

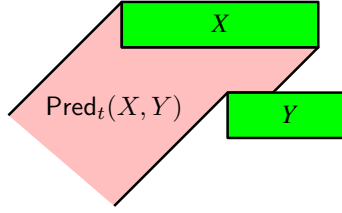


Fig. 28: Backwards fixed-point computation.

delay d may reach a state in X without visiting states in Y during this delay. Importantly, it has been shown that so-called *federations* (finite set of zones) are closed under these operators, enabling efficient zone-based realization. Now let $\pi(X)$ be the operator $\pi(X) = \text{Pred}_t[X \cup \text{cPred}(X), \text{uPred}(X^C)]$, then the following theorem holds:

Theorem 5. *Let Goal be a set of goal states. Then the set of winning states with respect to the reachability timed game of Goal is the least fixed point of the function: $X \mapsto \pi(X) \cup \text{Goal}$.*

5.3 Train Gate Example as a Game

We adopt the train-gate example shown previously to demonstrate controller synthesis. In this case, the gate controller environment consists of a number of trains as depicted in timed game automaton in Fig. 29a, where only **stop?** and **go?** edges are controllable (solid arrows) and others are not (dashed arrows), because the gate controls when **stop** and **go** are issued while the rest of the behavior is up to the individual train to decide.

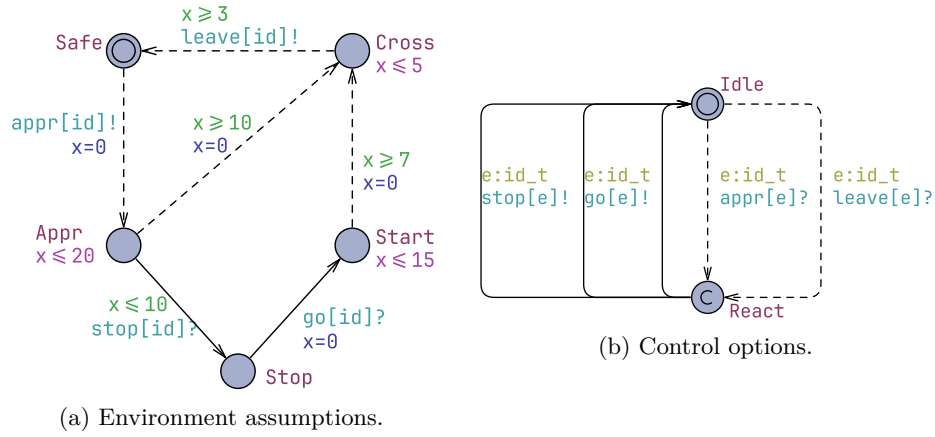


Fig. 29: Train-gate as a timed game.

The gate controller (Fig. 29b) then models all possible (re-)actions the controller can take when trains **approach** and **leave** uncontrollably: issue **stop**, **go**, or simply ignore (solid arrow without channel synchronization).

The objective then is to find *safe* strategy for controllable actions such that the behavior satisfies predicate ϕ , where ϕ is “never two trains at the crossing at the same time”, which can be formulated as the following alternative control queries:

```
control: A[] not exists(i:id_t) exists(j:id_t) i ≠ j and
           Train(i).Cross and Train(j).Cross
control: A[] forall(i:id_t) forall(j:id_t)
           Train(i).Cross and Train(j).Cross imply i==j
```

In Fig. 30 we have used the **simulate** query to provide information about the location of 4 trains under the *Safety* strategy. For each train the maximal value is achieved when it is in the **Cross**-ing. It can be noted that, as expected, there are never two trains in the **Cross**-ing at the same time (magenta line is always at zero, moreover the location trajectories are never at their top value $5 \cdot \text{Train}[i].\text{Cross}$ at the same time).

5.4 Applications

UPPAAL TIGA been used in industrial applications ranging from automatic synthesis of control strategies for zone-based climate control in pigsties [71] to safe and optimal operation of hydraulic pumps [40].

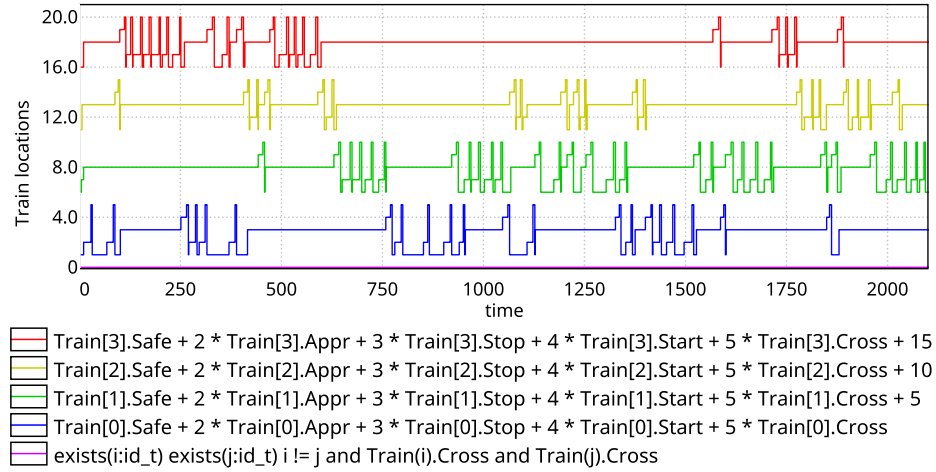


Fig. 30: Simulation of 4 Trains under Safety Strategy

6 Timed Markov Decision Processes

STRATEGO, introduced in 2014 [43,44] is the most recent branch of the UPPAAL tool suite. It enables users to generate, optimize, compare, and analyze the performance and consequences of strategies synthesized for Timed Markov Decision Processes (TMDP) in a user-friendly way.

A key feature of STRATEGO is its extended query language (see Table 5), in which strategies are treated as first-class objects. Strategies can be constructed, compared, optimized, and directly employed when performing (statistical) model checking of a game under the constraints of a synthesized strategy.

In this sense, STRATEGO can be regarded a superset of UPPAAL TIGA and UPPAAL SMC, combining their capabilities within a unified framework, now all collected in UPPAAL 5.0.

6.1 Syntax and Semantics

A Timed Markov Decision Process (TMDP) is syntactically identical to a Timed Game (TG). However, while the environment is viewed as being antagonistic in a TG, it is considered stochastic in a TMDP. As for Stochastic Timed Automata default stochastic choices of delays of then environment will be made by either uniform or exponential distributions.

To illustrate the notion of TMDP let us consider an appropriate version of our simple running example, here given in Fig. 31.

Compared to the TG version of Fig. 27, we have added the cost-variable C growing with different rates in the different locations. The controller can only affect the behavior in location $l1$ choosing between going to $l2$ (with action a) or going to $l3$ (with action b). Again, we assume the existence of an additional clock y that is never tested or reset.

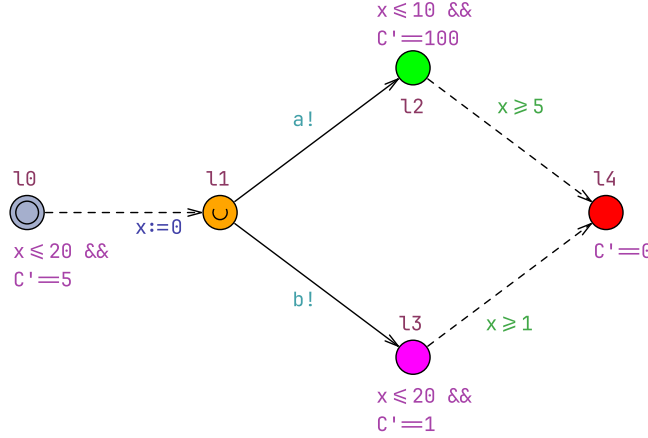


Fig. 31: Simple Priced Timed Markov Decision Process.

One interpretation of the model of Fig. 31 is that of a Priced Timed Game (PTG), where the objective is to find a strategy that minimizes the worst-case cost of reaching **l4**. However, cost-optimal winning strategies for priced games are generally undecidable when the underlying timed automata have three or more clocks [22]. Decidability results have been provided for one-clock priced timed games [29] and for so-called strongly cost-non-zeno priced timed games [1,24,82,59,30]; also approximate algorithms have been proposed [28].

However, being a Priced Timed Markov Decision Process (PTMDP), the environment is viewed as stochastic. In particular, the delays chosen by the environment in locations **l0**, **l2** and **l3** are chosen uniformly from the intervals $[0, 20]$, $[5, 10]$, respectively $[1, 20]$. The objective now is to find the control strategy that minimizes the expected overall completion cost. For the simple example of Fig. 31 this is clearly given by the following expression:

$$\int_{t=0}^{20} \frac{1}{20} \cdot 5 \cdot t \, dt + \min \left\{ \int_{t=5}^{10} \frac{1}{5} \cdot 100 \cdot t \, dt, \int_{t=1}^{20} \frac{1}{19} \cdot 1 \cdot t \, dt \right\} = 60.5$$

obtained by the controller that always chooses to proceed from **l1** to **l3**.

6.2 Queries of STRATEGO

In this section, we will demonstrate the usage of the STRATEGO query language in Table 5 on the simple PTMDP of Fig. 31. Now, posing the STRATEGO query:

strategy Cheap = minE(C) [≤ 100] : <> P.l4

will obtain by reinforcement learning the strategy **Cheap** that minimizes the expected value of **C** upon reaching **l4**. Now the query

Table 5: Various types of STRATEGO queries: “**strategy S =**” means strategy assignment and “**under S**” is strategy usage via strategy identifier S. Here the variables NS, DS and SS correspond to non-deterministic, deterministic and stochastic strategies respectively; **bound** is a bound expression on time or cost like $x \leq 100$ and **n** is the number of simulations.

Strategy generators using [43]:	
Minimize objective:	<code>strategy DS = minE (expr) [bound]: <> prop</code>
Maximize objective:	<code>strategy DS = maxE (expr) [bound]: <> prop under NS</code>

Strategy generators using UPPAAL TIGA:	
Guarantee objective:	<code>strategy NS = control: A<> prop</code>
Guarantee objective:	<code>strategy NS = control: A[] prop</code>

Statistical Model Checking Queries:	
Hypothesis testing:	<code>Pr[bound](<> prop) ≥ 0.1 under SS</code>
Evaluation:	<code>Pr[bound](<> prop) under SS</code>
Comparison:	<code>Pr[bound](<> prop1) under SS1 $\geq$ Pr[bound](<> prop2) under SS2</code>
Expected value:	<code>E[bound;n](min: prop) under SS</code>
Simulations:	<code>simulate[bound;n] { expr1, expr2 } under SS</code>

Symbolic model checking queries:	
Safety:	<code>A[] prop under NS</code>
Liveness:	<code>A<> prop under NS</code>
Infimum of value:	<code>inf { condition } : expression</code>
Supremum of value:	<code>sup { condition } : expression</code>

`E[ $\leq 100; 1000000$ ](max:C) under Cheap`

generates 10 million runs to estimate the expected value of C under the Cheap strategy. Here, 60.4864 ± 0.576115 is returned as a confidence interval 95% consistent with the analytical value of 60.5.

Now clearly, the learned strategy Cheap does not guarantee that **l4** is reached within 31 time units. For this, we pose the following query:

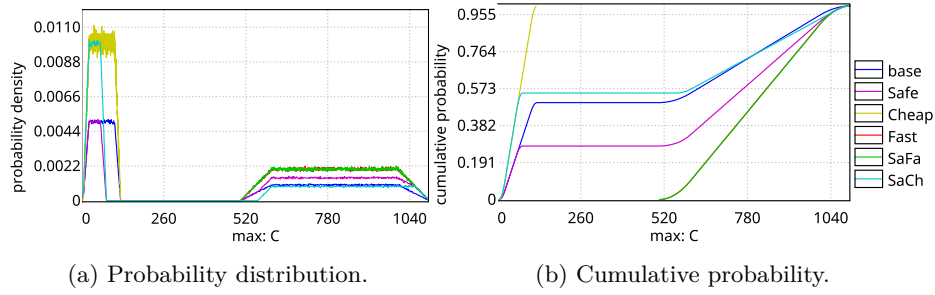
`strategy Safe = control: A[] (time  $\geq 31$  imply P.l4)`

which symbolically synthesize the most permissive strategy Safe guaranteeing this. Now, under the constraints of this permissive strategy, the following query will find the C-minimal sub-strategy using “shielded” reinforcement learning:

`strategy SafeCheap = minE(C)[ $\leq 31$ ] : <> P.l4 under Safe`

Now `E[ $\leq 100; 1000000$ ](max:C) under SafeCheap` returns 393.326 ± 0.73471 as a 95% CI for the expected reachability cost under SafeCheap. Being significantly larger than the minimal cost of an unconstrained strategy, SafeCheap guarantees that the upper 31 upper time-bound of reaching **l4** is guaranteed.

Below, we list a number of STRATEGO queries on the simple PTMDP model – include the ones we already discussed.

Fig. 32: Cost estimate of **l4** by PTMDP in Fig. 31.

- Optimize time for reaching **l4**:
`strategy Fast = minE(time)[ ≤ 100 ] : <> P.l4`
- Optimize time for reaching **l4** while using **Safe** strategy:
`strategy SafeFast = minE(time)[ ≤ 100 ] : <> P.l4 under Safe`
- Optimize cost C for reaching **l4**:
`strategy Cheap = minE(C)[ ≤ 100 ] : <> P.l4`
- Optimize cost for reaching **l4** while using **Safe** strategy:
`strategy SafeCheap = minE(C)[ ≤ 100 ] : <> P.l4 under Safe`
- Baseline cost estimate of stochastic behavior $E[\leq 100; 1000000](\text{max:C})$
- Cost estimate of **Cheap** strategy:
 $E[\leq 100; 1000000](\text{max:C})$ under **Cheap**
- Cost estimate of **Safe** strategy:
 $E[\leq 100; 1000000](\text{max:C})$ under **Safe**
- Cost estimate of **SafeCheap** strategy:
 $E[\leq 100; 1000000](\text{max:C})$ under **SafeCheap**
- Cost estimate of **SafeFast** strategy:
 $E[\leq 100; 1000000](\text{max:C})$ under **SafeFast**
- Performance of **Cheap** strategy:
 $\text{Pr}[\leq 2000; 1000000](<> \text{P.l4})$ under **Cheap**
- Performance of **SafeCheap** strategy:
 $\text{Pr}[\leq 2000; 1000000](<> \text{P.l4})$ under **SafeCheap**
- Is **Cheap** strategy also safe? Not always, only 89.3%:
 $\text{Pr}[\leq 31; 1000000](<> \text{P.l4})$ under **Cheap**

The plots in Fig. 32 show that **Cheap** strategy has the lowest expected cost overall, which seems to use only one specific path to reach **l4**, while other strategies juggle between **l2** and **l3** locations. The second best is **SafeCheap** (**SaCh** in legend) which follows the **Cheap** strategy but sometimes has to switch through a different path, so it looks like the **Cheap** strategy might have safety issues. **Fast** and **SafeFast** (**SaFa** in the legend) strategy have overlapping distributions again, thus their costs is indistinguishable.

The performance of all strategies is shown in Fig. 33, where all strategies have similar timing performance. **Cheap** strategy has a triangular distribution

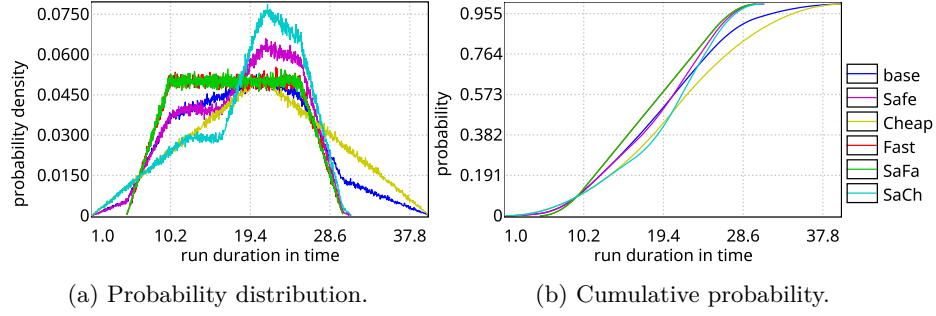


Fig. 33: Probability reaching 14 by PTMDP in Fig. 31.

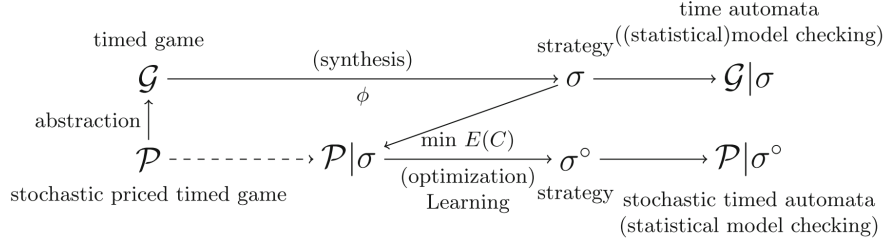


Fig. 34: Overview of workflows in STRATEGO.

which is consistent with two consecutive uniform delay distributions. The cumulative probability plot shows that **Cheap** strategy is even slower to complete than uniform choices (**base** in the legend), which is definitely not safe.

In general, as shown in the overview Fig.34, STRATEGO will start from a PTMDP $\mathcal{P}$. It can then abstract $\mathcal{P}$ into a timed game (TG) $\mathcal{G}$ by simply ignoring prices and stochasticity in the model. Using $\mathcal{G}$, UPPAAL TIGA may now be used to (symbolically) synthesize a (most permissive) strategy σ meeting a required safety or (time-bounded) liveness constraint ϕ . The TG $\mathcal{G}$ under σ (denoted $\mathcal{G}|\sigma$) may now be subject to additional (statistical) model checking using classical UPPAAL and UPPAAL SMC. Similarly, the original PTMDP $\mathcal{P}$ under σ may be subject to statistical model checking. Now using reinforcement learning, we may synthesize near-optimal strategies that minimize (maximize) the expectation of a given cost-expression cost. In case the learning is performed from $\mathcal{P}|\sigma$, we obtain a sub-strategy σ° of σ that optimizes the expected value of cost subject to the hard constraints guaranteed by σ . Finally, given σ° , one may perform additional statistical model checking of $\mathcal{P}|\sigma^\circ$.

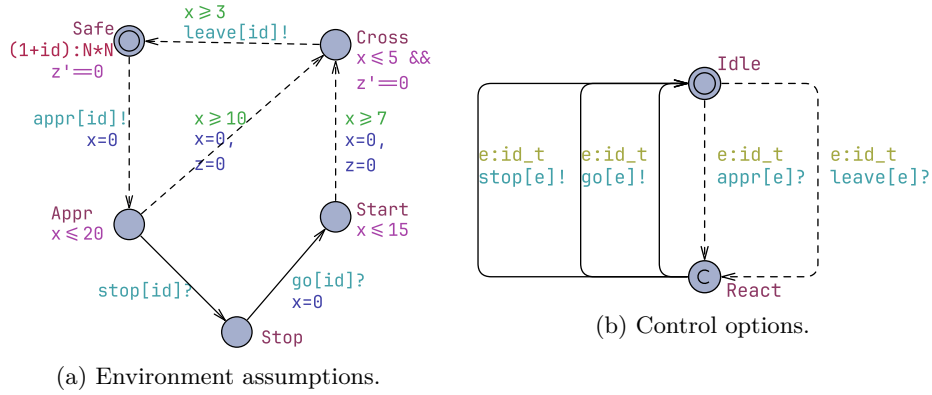


Fig. 35: Train-gate as a stochastic timed game.

6.3 Train Gate Example and Reinforcement Learning

Figure 35 provides a PTMDP model of the Train Gate, refining the TG model of Fig. 29 with stochastic choices for delay; note the exponential rate expression in location **Safe**.

Now the following query generates the named safety strategy **GoSafe** that ensures that two different trains will never be in the **Cross**-ing at the same time.

```
strategy GoSafe =
  control: A[] not exists(i:id_t) exists(j:id_t) i ≠ j and
    Train(i).Cross and Train(j).Cross
```

Now, for each train, we have added an additional stopwatch z that is stopped in locations **Safe** and **Cross**. Thus z measures the total time that the train is waiting to enter the crossing. Now the following query learns the sub-strategy **GoSafeFast** that minimizes the expected waiting time for **Train(0)** within 200 time-units.

```
strategy GoSafeFast =
  minE(Train(0).z) [≤200]: <> time ≥ 200 under GoSafe
```

In Fig. 36 we see a simulation of the position of the 4 trains under the learned **GoSafe** strategy. Note, that while **Train(0)** repeatedly enters the **Cross**-ing, the other trains are largely ignored – indicating that safety is guaranteed. It is also possible to minimize the sum of waiting times by replacing **Train(0).z** expression with **sum(i:id_t) Train(i).z**.

6.4 Reinforcement Learning in STRATEGO

The first simulation-based method implemented in STRATEGO for “shielded” learning of strategies with optimal expected cost, yet guaranteed completion time, was presented in [43,44]. Here, starting from a uniform version of the

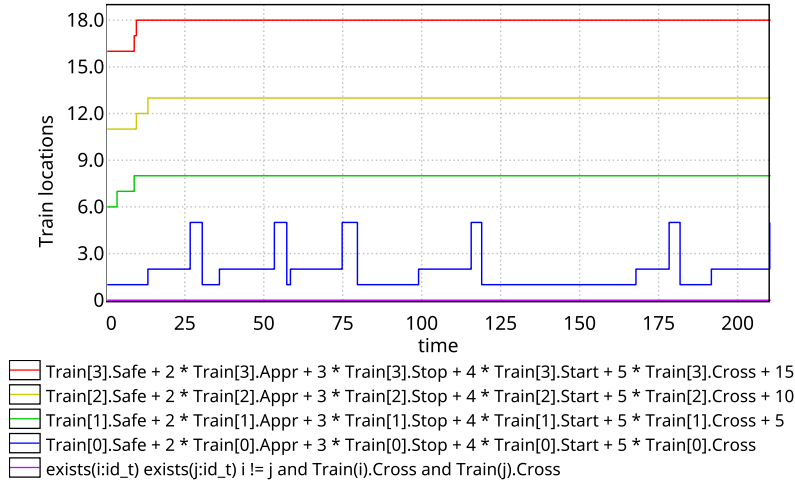


Fig. 36: Simulation of 4 Trains under GoSafeFast strategy

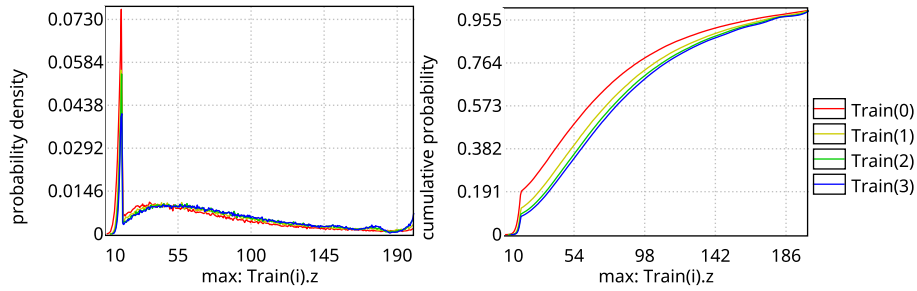


Fig. 37: Distribution of waiting times for each train with stochastic controller.

most permissive strategy guaranteeing the given time bound, the learning technique iteratively improves the strategy – by observing the effect of control choices in sampled runs – until a strategy with satisfactory expected reachability cost is found.

Later, in [65], the reinforcement learning method currently available in STRATEGO was presented. Here, we developed and performed an exhaustive experimental evaluation of two versions of a basic partition refinement approach: one based on Q-learning and one related to real-time dynamic programming (RTDP).

Despite its high performance in practice, the method in [65] lacks a proof of convergence due to the infinite state space of the MDP considered. This has been the subject of investigation in [64] and in [66], where an alternative reinforcement learning method for Euclidean MDPs with proven convergence properties has been proposed.

From the very first papers [43,44], methods in UPPAAL have supported reinforcement learning for infinite state MDPs “shielded” with respect to already

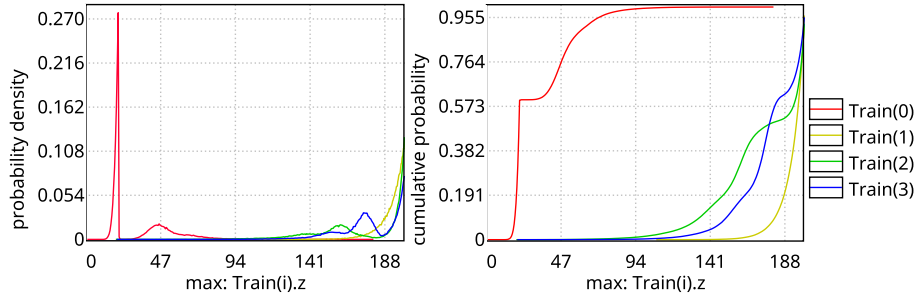


Fig. 38: Distribution of waiting times for each train under GoSafeFast.

synthesized, permissive safety strategies. Since, several researchers have contributed to this notion of “shielded” reinforcement learning [18].

Within STRATEGO shielded synthesis for Hybrid Markov Decision Processes has been developed using axis-aligned partitioning [33], as well as state-space transformations [32] and semantic preservation of state-partioning. This work is now available in the new branch UPPAAL COSHY: *Automatic Synthesis of Compact Shields for Hybrid Systems*.

Most recently, work on shielded reinforcement learning for multi-agent systems has been developed and implemented [34].

6.5 Applications

STRATEGO has been applied in several industrial applications, including safe and optimal adaptive cruise control, optimal control of traffic lights in intersections [53], safe and optimal control strategies for storm water detention ponds [56], as well as control of domestic heat-pumps [60]. In two cases, the effort has resulted in spin-out companies: ATS⁸ for intelligent traffic control and CEDAR⁹ for control of heat-pumps.

7 Acknowledgment

The authors would like to thank all our colleagues that have contributed to the theory, implementation and applications of UPPAAL over the years. Also thanks to the anonymous reviewers for their constructive and insightful comments and suggestions.

⁸ <https://ambolt.io/>

⁹ <https://cedar-heat.dk/#hero>

References

1. Alur, R., Bernadsky, M., Madhusudan, P.: Optimal reachability for weighted timed games. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) *Automata, Languages and Programming*. pp. 122–133. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27836-8_13
2. Alur, R., Dill, D.L.: A theory of timed automata. *Theoretical Computer Science* **126**(2), 183–235 (1994). [https://doi.org/10.1016/0304-3975\(94\)90010-8](https://doi.org/10.1016/0304-3975(94)90010-8)
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. *J. ACM* **43**(1), 116–146 (Jan 1996). <https://doi.org/10.1145/227595.227602>
4. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A. (eds.) *Hybrid Systems: Computation and Control*. pp. 49–62. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-45351-2_8
5. Arcile, J., André, É.: Zone extrapolations in parametric timed automata. In: Deshmukh, J.V., Havelund, K., Perez, I. (eds.) *NASA Formal Methods*. pp. 451–469. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-06773-0_24
6. Arcile, J., André, É.: Zone extrapolations in parametric timed automata. *Innovations in Systems and Software Engineering* **21**, 707–726 (June 2025). <https://doi.org/10.1007/s11334-024-00554-5>
7. Asarin, E., Maler, O.: As soon as possible: Time optimal control for timed automata. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) *Hybrid Systems: Computation and Control*. pp. 19–30. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48983-5_6
8. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. *IFAC Proceedings Volumes* **31**(18), 447–452 (1998). [https://doi.org/10.1016/S1474-6670\(17\)42032-5](https://doi.org/10.1016/S1474-6670(17)42032-5), 5th IFAC Conference on System Structure and Control 1998 (SSC’98), Nantes, France, 8–10 July
9. Behrmann, G., Larsen, K.G., Pearson, J., Weise, C., Yi, W.: Efficient timed reachability analysis using clock difference diagrams. In: Halbwachs, N., Peled, D. (eds.) *Computer Aided Verification*. pp. 341–353. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48683-6_30
10. Behrmann, G., Bouyer, P., Fleury, E., Larsen, K.G.: Static guard analysis in timed automata verification. In: Garavel, H., Hatcliff, J. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems*. pp. 254–270. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). https://doi.org/10.1007/3-540-36577-X_18
11. Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer* **8**, 204–215 (June 2006). <https://doi.org/10.1007/s10009-005-0190-0>
12. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-TIGA: Time for playing games! In: *Proceedings of the 19th International Conference on Computer Aided Verification*. pp. 121–125. No. 4590 in LNCS, Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_14
13. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: *Third International Conference on the Quantitative Evaluation of Systems - (QEST’06)*. vol. 6, pp. 125–126 (2006). <https://doi.org/10.1109/QEST.2006.59>

14. Behrmann, G., David, A., Larsen, K.G., Pettersson, P., Yi, W.: Developing UPPAAL over 15 years. *Softw., Pract. Exper.* **41**(2), 133–142 (2011). <https://doi.org/10.1002/spe.1006>
15. Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-cost reachability for priced time automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A. (eds.) *Hybrid Systems: Computation and Control*. pp. 147–161. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-45351-2_15
16. Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Beyond liveness: Efficient parameter synthesis for time bounded liveness. In: Pettersson, P., Yi, W. (eds.) *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings. Lecture Notes in Computer Science*, vol. 3829, pp. 81–94. Springer (2005). https://doi.org/10.1007/11603009_7
17. Bisgaard, M., Gerhardt, D., Hermanns, H., Krčál, J., Nies, G., Stenger, M.: Battery-aware scheduling in low orbit: The GomX-3 case. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) *FM 2016: Formal Methods*. pp. 559–576. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-48989-6_34
18. Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis: Runtime enforcement for reactive systems. In: Baier, C., Tinelli, C. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems*. pp. 533–548. Springer Berlin Heidelberg, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_51
19. Boudjadar, A.J., David, A., Kim, J.H., Larsen, K.G., Mikučionis, M., Nyman, U., Skou, A.: Degree of schedulability of mixed-criticality real-time systems with probabilistic sporadic tasks. In: *2014 Theoretical Aspects of Software Engineering Conference*. pp. 126–130 (Sep 2014). <https://doi.org/10.1109/TASE.2014.27>
20. Bouyer, P.: Untameable timed automata! In: Alt, H., Habib, M. (eds.) *STACS 2003*. pp. 620–631. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). https://doi.org/10.1007/3-540-36494-3_54
21. Bouyer, P.: Forward analysis of updatable timed automata. *Formal Methods in System Design* **24**, 281–320 (May 2004). <https://doi.org/10.1023/B:FORM.0000026093.21513.31>
22. Bouyer, P., Brihaye, T., Markey, N.: Improved undecidability results on weighted timed automata. *Information Processing Letters* **98**(5), 188–194 (2006). <https://doi.org/10.1016/j.ipl.2006.01.012>, <https://www.sciencedirect.com/science/article/pii/S0020019006000652>
23. Bouyer, P., Brinksma, E., Larsen, K.G.: Optimal infinite scheduling for multi-priced timed automata. *Formal Methods Syst. Des.* **32**(1), 3–23 (2008). <https://doi.org/10.1007/S10703-007-0043-4>
24. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.G.: Optimal strategies in priced timed game automata. In: Lodaya, K., Mahajan, M. (eds.) *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*. pp. 148–160. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30538-5_13
25. Bouyer, P., Colange, M., Markey, N.: Symbolic optimal reachability in weighted timed automata. In: Chaudhuri, S., Farzan, A. (eds.) *Computer Aided Verification*. pp. 513–530. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_28

26. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Ouaknine, J., Worrell, J.: Model checking real-time systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) *Handbook of Model Checking*, pp. 1001–1046. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_29
27. Bouyer, P., Gastin, P., Herbreteau, F., Sankur, O., Srivathsan, B.: Zone-based verification of timed automata: Extrapolations, simulations and what next? In: Bogomolov, S., Parker, D. (eds.) *Formal Modeling and Analysis of Timed Systems*. pp. 16–42. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-15839-1_2
28. Bouyer, P., Larsen, K.G., Markey, N., Rasmussen, J.I.: Almost optimal strategies in one clock priced timed games. In: *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science*. p. 345–356. FSTTCS’06, Springer-Verlag, Berlin, Heidelberg (2006). https://doi.org/10.1007/11944836_32, https://doi.org/10.1007/11944836_32
29. Brihaye, T., Bruyère, V., Raskin, J.F.: On optimal timed strategies. In: Pettersson, P., Yi, W. (eds.) *Formal Modeling and Analysis of Timed Systems*. pp. 49–64. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11603009_5
30. Brihaye, T., Geeraerts, G., Narayanan Krishna, S., Manasa, L., Monmege, B., Trivedi, A.: Adding negative prices to priced timed games. In: Baldan, P., Gorla, D. (eds.) *CONCUR 2014 – Concurrency Theory*. pp. 560–575. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44584-6_38
31. Brihaye, T., Henzinger, T.A., Prabhu, V.S., Raskin, J.F.: Minimum-time reachability in timed games. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) *Automata, Languages and Programming*. pp. 825–837. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73420-8_71
32. Brorholt, A.H., Høeg-Petersen, A.H., Larsen, K.G., Schilling, C.: Efficient shield synthesis via state-space transformation. In: Steffen, B. (ed.) *Bridging the Gap Between AI and Reality*. pp. 206–224. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-75434-0_14
33. Brorholt, A.H., Jensen, P.G., Larsen, K.G., Lorber, F., Schilling, C.: Shielded reinforcement learning for hybrid systems. In: Steffen, B. (ed.) *Bridging the Gap Between AI and Reality*. pp. 33–54. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-46002-9_3
34. Brorholt, A.H., Larsen, K.G., Schilling, C.: Compositional shielding and reinforcement learning for multi-agent systems. In: *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*. p. 399–407. AAMAS ’25, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2025). <https://doi.org/10.5555/3709347.3743554>
35. Bulychev, P., David, A., Guldstrand Larsen, K., Legay, A., Li, G., Bøgsted Poulsen, D., Stainer, A.: Monitor-based statistical model checking for weighted metric temporal logic. In: Bjørner, N., Voronkov, A. (eds.) *Logic for Programming, Artificial Intelligence, and Reasoning*. pp. 168–182. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28717-6_15
36. Bulychev, P., David, A., Guldstrand Larsen, K., Legay, A., Mikučionis, M., Bøgsted Poulsen, D.: Checking and distributing statistical model checking. In: Goodloe, A.E., Person, S. (eds.) *NASA Formal Methods*. pp. 449–463. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28891-3_39

37. Bulychev, P., David, A., Larsen, K.G., Legay, A., Li, G., Poulsen, D.B.: Rewrite-based statistical model checking of wmtl. In: Runtime Verification. LNCS, vol. 7687, pp. 260–275 (2012). https://doi.org/10.1007/978-3-642-35632-2_25
38. Bulychev, P.E., Cassez, F., David, A., Larsen, K.G., Raskin, J., Reynier, P.: Controllers with minimal observation power (application to timed systems). In: Chakraborty, S., Mukund, M. (eds.) Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7561, pp. 223–237. Springer (2012). https://doi.org/10.1007/978-3-642-33386-6_19
39. 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 2005 – Concurrency Theory. pp. 66–80. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11539452_9
40. Cassez, F., Jessen, J.J., Larsen, K.G., Raskin, J., Reynier, P.: Automatic synthesis of robust and optimal controllers - an industrial case study. In: Majumdar, R., Tabuada, P. (eds.) Hybrid Systems: Computation and Control, 12th International Conference, HSCC 2009, San Francisco, CA, USA, April 13-15, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5469, pp. 90–104. Springer (2009). https://doi.org/10.1007/978-3-642-00602-9_7
41. Clopper, C.J., Pearson, E.S.: The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika* **26**(4), 404–413 (1934), <http://www.jstor.org/stable/2331986>
42. David, A., Du, D., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. In: Bartocci, E., Bortolussi, L. (eds.) Hybrid Systems and Biology (HSB). EPTCS, vol. 92, pp. 122–136. Newcastle Upon Tyne, UK (September 2012). <https://doi.org/10.4204/EPTCS.92.9>
43. David, A., Jensen, P.G., Larsen, K.G., Legay, A., Lime, D., Sørensen, M.G., Taankvist, J.H.: On time with minimal expected cost! In: Cassez, F., Raskin, J.F. (eds.) Automated Technology for Verification and Analysis. pp. 129–145. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_10
44. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal Stratego. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9035, pp. 206–211. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_16
45. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.* **17**(4), 397–415 (2015). <https://doi.org/10.1007/s10009-014-0361-y>
46. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for biological systems. *International Journal on Software Tools for Technology Transfer* **17**, 351–367 (June 2015). <https://doi.org/10.1007/s10009-014-0323-4>
47. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Vliet, J.V., Wang, Z.: Statistical model checking for networks of priced timed automata. In: FORMATS. pp. 80–96. LNCS, Springer (2011). https://doi.org/10.1007/978-3-642-24310-3_7

48. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science*, vol. 6806, pp. 349–355. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_27
49. David, A., Larsen, K.G., Legay, A., Mikučionis, M.: Schedulability of Herschel-Planck revisited using statistical model checking. In: Margaria, T., Steffen, B. (eds.) *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*. pp. 293–307. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34032-1_28
50. David, A., Du, D., Larsen, K.G., Mikučionis, M., Skou, A.: An evaluation framework for energy aware buildings using statistical model checking. *Science China Information Sciences* **55**, 2694–2707 (December 2012). <https://doi.org/10.1007/s11432-012-4742-0>
51. David, N., David, A., Hansen, R.R., Larsen, K.G., Legay, A., Olesen, M.C., Probst, C.W.: Modelling social-technical attacks with timed automata. In: *Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats*. p. 21–28. MIST '15, Association for Computing Machinery, New York, NY, USA (2015). <https://doi.org/10.1145/2808783.2808787>
52. Dierks, H., Kupferschmid, S., Larsen, K.G.: Automatic abstraction refinement for timed automata. In: Raskin, J.F., Thiagarajan, P.S. (eds.) *Formal Modeling and Analysis of Timed Systems*. pp. 114–129. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75454-1_10
53. Eriksen, A.B., Huang, C., Kildebogaard, J., Lahrmann, H.S., Larsen, K.G., Muñiz, M., Taankvist, J.H.: Uppaal Stratego for intelligent traffic lights. In: *12th ITS European Congress. ERTICO - ITS Europe (2017)*, <https://vbn.aau.dk/en/publications/uppaal-stratego-for-intelligent-traffic-lights>, 12th ITS European Congress : ITS Beyond Borders ; Conference date: 19-06-2017 Through 22-06-2017
54. Gastin, P., Mukherjee, S., Srivathsan, B.: Fast algorithms for handling diagonal constraints in timed automata. In: Dillig, I., Tasiran, S. (eds.) *Computer Aided Verification*. pp. 41–59. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_3
55. van Glabbeek, R., Höfner, P., Portmann, M., Tan, W.L.: Modelling and verifying the AODV routing protocol. *Distributed Computing* **29**, 279–315 (August 2016). <https://doi.org/10.1007/s00446-015-0262-7>
56. Goorden, M.A., Larsen, K.G., Nielsen, J.E., Nielsen, T.D., Rasmussen, M.R., Srba, J.: Learning safe and optimal control strategies for storm water detention ponds. *IFAC-PapersOnLine* **54**(5), 13–18 (2021). <https://doi.org/10.1016/j.ifacol.2021.08.467>, 7th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2021
57. Grobelna, I., Gajewski, K., Karatkevich, A.: A systematic review on the applications of uppaal. *Sensors* **25**(11) (2025). <https://doi.org/10.3390/s25113484>, <https://www.mdpi.com/1424-8220/25/11/3484>
58. Guldstrand Larsen, K., Lorber, F., Nielsen, B.: 20 years of real time model validation. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) *Formal Methods*. pp. 22–36. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_2

59. Hansen, T.D., Ibsen-Jensen, R., Miltersen, P.B.: A faster algorithm for solving one-clock priced timed games. In: D’Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013 – Concurrency Theory. pp. 531–545. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40184-8_37
60. Hasrat, I.R., Jensen, P.G., Larsen, K.G., Srba, J.: A toolchain for domestic heat-pump control using Uppaal Stratego. *Science of Computer Programming* **230**, 102987 (2023). <https://doi.org/https://doi.org/10.1016/j.scico.2023.102987>
61. Henzinger, T.A., Kopke, P.W.: Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science* **221**(1), 369–392 (1999). [https://doi.org/10.1016/S0304-3975\(99\)00038-9](https://doi.org/10.1016/S0304-3975(99)00038-9)
62. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. In: *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS ’92)*, Santa Cruz, California, USA, June 22–25, 1992. pp. 394–406. IEEE Computer Society (1992). <https://doi.org/10.1109/LICS.1992.185551>
63. Herbreteau, F., Srivathsan, B., Walukiewicz, I.: Better abstractions for timed automata. *Information and Computation* **251**, 67–90 (2016). <https://doi.org/https://doi.org/10.1016/j.ic.2016.07.004>
64. Jaeger, M., Bacci, G., Bacci, G., Larsen, K.G., Jensen, P.G.: Approximating euclidean by imprecise markov decision processes. In: Margaria, T., Steffen, B. (eds.) *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles*. pp. 275–289. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_15
65. Jaeger, M., Jensen, P.G., Guldstrand Larsen, K., Legay, A., Sedwards, S., Taankvist, J.H.: Teaching Stratego to play ball: Optimal synthesis for continuous space MDPs. In: Chen, Y.F., Cheng, C.H., Esparza, J. (eds.) *Automated Technology for Verification and Analysis*. pp. 81–97. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-31784-3_5
66. Jaeger, M., Larsen, K.G.: Reinforcement learning for discretized euclidean mdps. In: Steffen, B. (ed.) *Bridging the Gap Between AI and Reality*. pp. 312–335. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-75434-0_22
67. Jensen, N.Ø., Jensen, P.G., Larsen, K.G.: Dynamic extrapolation in extended timed automata. In: Li, Y., Tahar, S. (eds.) *Formal Methods and Software Engineering*. pp. 83–99. Springer Nature Singapore, Singapore (2023). https://doi.org/10.1007/978-981-99-7584-6_6
68. Jensen, N.Ø., Larsen, K.G., Lime, D., Srba, J.: On-the-fly symbolic algorithm for timed ATL with abstractions. In: Bouyer, P., van de Pol, J. (eds.) *36th International Conference on Concurrency Theory, CONCUR 2025, August 26–29, 2025, Aarhus, Denmark. LIPIcs*, vol. 348, pp. 25:1–25:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2025). <https://doi.org/10.4230/LIPICS.CONCUR.2025.25>
69. Jensen, P.G., Kiviriga, A., Guldstrand Larsen, K., Nyman, U., Mijačika, A., Høiriis Mortensen, J.: Monte carlo tree search for priced timed automata. In: Ábrahám, E., Paolieri, M. (eds.) *Quantitative Evaluation of Systems*. pp. 381–398. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-16336-4_19
70. Jensen, P.G., Larsen, K.G., Srba, J., Sørensen, M.G., Taankvist, J.H.: Memory efficient data structures for explicit verification of timed systems. In: Badger, J.M.,

- Rozier, K.Y. (eds.) *NASA Formal Methods*. pp. 307–312. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-06200-6_26
71. Jessen, J.J., Rasmussen, J.I., Larsen, K.G., David, A.: Guided controller synthesis for climate controller using uppaal tiga. In: Raskin, J., Thiagarajan, P.S. (eds.) *Formal Modeling and Analysis of Timed Systems*, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3–5, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4763, pp. 227–240. Springer (2007). https://doi.org/10.1007/978-3-540-75454-1_17
 72. Jørgensen, K.Y., Larsen, K.G., Srba, J.: Time-darts: A data structure for verification of closed timed automata. *Electronic Proceedings in Theoretical Computer Science* **102**, 141–155 (Nov 2012). <https://doi.org/10.4204/eptcs.102.13>
 73. Jurdziński, M., Trivedi, A.: Reachability-time games on timed automata. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) *Automata, Languages and Programming*. pp. 838–849. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73420-8_72
 74. Kupferschmid, S., Hoffmann, J., Dierks, H., Behrmann, G.: Adapting an ai planning heuristic for directed model checking. In: Valmari, A. (ed.) *Model Checking Software*. pp. 35–52. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1007/11691617_3
 75. Kupferschmid, S., Wehrle, M., Nebel, B., Podelski, A.: Faster than Uppaal? In: Gupta, A., Malik, S. (eds.) *Computer Aided Verification*. pp. 552–555. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70545-1_53
 76. Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Efficient verification of real-time systems: compact data structure and state-space reduction. In: *Proceedings Real-Time Systems Symposium*. pp. 14–24 (Dec 1997). <https://doi.org/10.1109/REAL.1997.641265>
 77. Larsen, K.G., Behrmann, G., Brinksma, E., Fehnker, A., Hune, T., Pettersson, P., Romijn, J.: As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In: Berry, G., Comon, H., Finkel, A. (eds.) *Proceedings of CAV 2001*. pp. 493–505. No. 2102 in *Lecture Notes in Computer Science*, Springer-Verlag (2001). https://doi.org/10.1007/3-540-44585-4_47
 78. Larsen, K.G., Lorber, F., Nielsen, B.: 20 years of uppaal enabled industrial model-based validation and beyond. In: Margaria, T., Steffen, B. (eds.) *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*. pp. 212–229. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-03427-6_18
 79. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. *Int. Journal on Software Tools for Technology Transfer* **1**(1–2), 134–152 (Oct 1997). <https://doi.org/10.1007/s100090050010>
 80. Maler, O., Larsen, K.G., Krogh, B.H.: On zone-based analysis of duration probabilistic automata. In: Chen, Y., Rezzini, A. (eds.) *Proceedings 12th International Workshop on Verification of Infinite-State Systems, INFINITY 2010*, Singapore, Singapore, 21st September 2010. EPTCS, vol. 39, pp. 33–46 (2010). <https://doi.org/10.4204/EPTCS.39.3>
 81. Ruijters, E., Stoelinga, M.: Better railway engineering through statistical model checking. In: Margaria, T., Steffen, B. (eds.) *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*. pp. 151–165. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_10

82. Rutkowski, M.: Two-player reachability-price games on single-clock timed automata. *Electronic Proceedings in Theoretical Computer Science* **57**, 31–46 (Jul 2011). <https://doi.org/10.4204/eptcs.57.3>, <http://dx.doi.org/10.4204/EPTCS.57.3>
83. Tollund, R.G., Johansen, N.S., Nielsen, K.Ø., Álvaro Torralba, Larsen, K.G.: Optimal infinite temporal planning: Cyclic plans for priced timed automata. In: Bernardini, S., Muise, C. (eds.) *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*. pp. 588–596. AAAI Press (2024). <https://doi.org/10.1609/ICAPS.V34I1.31521>
84. Wald, A.: Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics* **16**(2), 117–186 (1945), <http://www.jstor.org/stable/2235829>