Quantitative analysis of real-time systems

Patricia Bouyer[*]
bouyer@lsv.ens-cachan.fr

Uli Fahrenberg[†]
uli@cs.aau.dk

Kim G. Larsen[†]
kgl@cs.aau.dk

Nicolas Markey[*]
markey@lsv.ens-cachan.fr

[*]LSV – CNRS & ENS Cachan
61 avenue du Président Wilson
94230 Cachan – France

[†]Department of Computer Science
Aalborg Universitet
9220 Aalborg Ø – Denmark

ABSTRACT

The problems of time-dependent behavior in general, and dynamic resource allocation in particular, pervade many aspects of modern life. Prominent examples range from reliability and efficient use of communication resources in a telecommunication network to the allocation of tracks in a continental railway network, from scheduling the usage of computational resources on a chip for durations of nano-seconds to the weekly, monthly or longer range reactive planning in a factory or a supply chain. But how can we model, analyze or even optimize such problems in an efficient manner?

1. INTRODUCTION

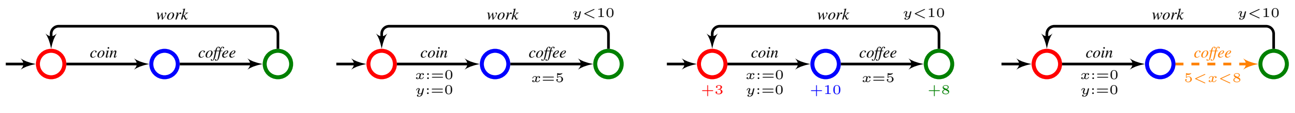
Timing: Twenty years ago, Rajeev Alur and David Dill introduced the notion of *timed automata* which, together with its later extensions, has proven a highly expressive and uniform formalism for formulating and analysing timing and resource problems arising in a variety of application domains. As a witness for the importance of the formalism one may consider the 2008 CAV award given to Alur and Dill for their seminal 1990 article *Automata for modeling real-time systems*,[3] which provided the theoretical foundation for the computer aided verification of real-time systems.

Real-time systems and resource allocation problems have manifested themselves under different names in application domains such as manufacturing, transport, communication networks, embedded systems, and digital circuits, and have been treated using theories and methods so-far scattered around many disciplines. All of these applications involve distributed, reactive systems of considerable complexity, and with a number of real-time constraints in the sense that correctness not only depends on the logical ordering of events of the systems, but also on the relative timing between these.

State-based models have been the basis of a wide range of successful computer-supported verification methodologies allowing the efficient prediction of functional properties, *e.g.* absence of deadlock or memory overflow. However, many of the models used in this methodology are purely discrete and their treatment of time is purely qualitative, that is, behaviors are just sequences of events appearing one after the other but without any quantitative timing information about the duration of actions and the time between events. Timed automata allow such timing constraints to be expressed, while being amenable to computer-aided analysis methods such as *simulation*, *verification*, *optimization* and *controller synthesis*.

Performance: In all of the above applications, an explicit constraint on timing is only one of a number of quantitative aspects of importance. Within embedded systems additional key quantities include *energy* and *memory* consumption, in communication networks required *band-width* is a key quantity, and within the factory and supply chain applications need for *storage* and overall *cost* for a given production are crucial quantities. The extended notion of *priced* or *weighted* timed automata has been put forward as a formalism allowing for such additional and time-dependent quantities to be modelled, without hampering efficient analysis and even permitting *optimization*.

Uncertainty: Classical models for scheduling in manufacturing, such as job-shop problems, are somewhat detached from industrial practise and reality. They assume that the duration of every step as well as the arrival times are fixed and known with certainty; in practice however, it is rarely the case that a schedule is executed as planned. The problem of coping with uncertainty is identified by providers of scheduling tools and by their clients as one of the major problems in the domain. There have been various attempts to model and solve such problems, but no unified approach has emerged. Using so-called *timed games* with controlled



(a) A finite automaton (b) A timed automaton (c) A priced timed automaton (d) A timed game

Figure 1: Several refinements of a model (a) of the working mathematician according to Erdős: After insertion of a coin into the coffee dispenser, coffee can be collected, and the scientist can go back to work. In the timed-automaton model (b), precisely five time units pass between coin insertion and coffee collection, and the time which passes between coin insertion and going back to work is less than ten time units. In the priced timed automaton (c), cost rates (modeling *e.g.* energy consumption) are associated with the three states. In the timed game (d), uncertainty as to precisely when coffee is delivered is modeled as an uncontrolled edge.

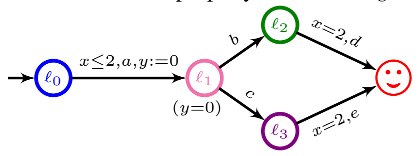
and uncontrolled transitions for representing the uncertainty coming from the plant, we can model a large class of such problems, and provide efficient off-line algorithms for synthesizing reactive schedulers. Such algorithms can plan for the best, worst or average case, but the scheduling strategies they produce are adaptive and can take advantage, for example, of the fact that a task has terminated before it was expected to.

In this paper we present the formalism of timed automata and its priced and game extension as a unifying mathematical framework for the modelling, analysis, optimization and synthesis of real-time related phenomena.

2. TIMED AUTOMATA

2.1 A model for time

Timed automata[3,4] are a powerful model for representing and reasoning about systems where the notion of time is essential. They are an extension of classical finite-state automata with real-valued variables called *clocks*. These clocks all increase at the same rate, and their values can be used to restrict availability of transitions and how long one can stay in a location (or state). Also, clocks can be reset to zero when a transition is taken. To this end, each transition has associated with it a *guard* (which must be satisfied for the transition to be enabled) and a set of clocks to be reset, and each location carries an *invariant* which must be continuously satisfied when the system is in the location. Below we show an example of a timed automaton with two clocks, x and y, and label set $\{a, b, c, d, e\}$. Note that no time can elapse in location ℓ_1 due to the invariant $(y = 0)$; locations with this property are called *urgent*.



Guards and invariants are given as comparisons $x \bowtie c$ of a clock value with an integer constant, or as conjunctions of these. In some contexts, also so-called *diagonal* constraints $x - y \bowtie c$ are allowed as parts of a guard (or invariant), but other extensions quickly lead to undecidability issues, see below.

A configuration of the system is made of a location and a clock valuation (in our case, values for both clocks x and y). A possible execution in our example is:

$$\begin{bmatrix}\ell_0\\0\\0\end{bmatrix}\xrightarrow[1.3]{\text{delay}}\begin{bmatrix}\ell_0\\1.3\\1.3\end{bmatrix}\xrightarrow[a]{\text{action}}\begin{bmatrix}\ell_1\\1.3\\0\end{bmatrix}\xrightarrow[c]{\text{action}}\begin{bmatrix}\ell_3\\1.3\\0\end{bmatrix}\xrightarrow[0.7]{\text{delay}}\begin{bmatrix}\ell_3\\2\\0.7\end{bmatrix}\xrightarrow[e]{\text{action}}\begin{bmatrix}\text{☺}\\2\\0.7\end{bmatrix}$$

where the first component of a configuration is the location and the second and third components give the values of clocks x and y, respectively. This execution corresponds to a delay of 1.3 time units in ℓ_0, the firing of transition a (this is possible because the value of clock x is 1.3, which is less than 2; clock y is then set to 0), a delay of 0 time unit in ℓ_1 (which we did not represent as it is a no-op), the firing of transition c, *etc.*

In the context of verification, several problems are of interest, like the model-checking of safety properties ("*Can a distinguished set of states be avoided?*"), reachability/liveness properties ("*Can/will a distinguished goal state be reached?*"), or more involved properties such as response properties ("*Is any request eventually granted?*"). As a model for real-time systems, these properties can include quantitative constraints, for instance time-bounded reachability, or time-bounded response properties ("*Is any request granted within two minutes?*"). It is also relevant to compute optimal time-bounds for these properties, like optimal-time reachability ("*What is the minimum time required for reaching a distinguished set of states?*").

2.2 The region abstraction

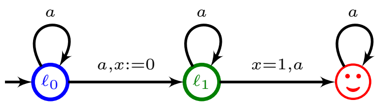
A timed automaton is a finite representation of an infinite transition system, since clocks take (nonnegative) real values. However, there is a way to deal with this infinity of configurations by reasoning symbolically: the main theoretical ingredient for solving problems on timed automata is the notion of *regions*,[4] which provide a finite partitioning of the state space such that states within a given region are behaviorally indistinguishable.

The precise definition of regions is such that inside a region, integral parts of clock values do not change, and also the ordering of clocks according to their values' fractional parts stays the same. Special consideration has to be given to the cases where one or more clock values are integers, and finiteness of the region partitioning is ensured by considering as equivalent all clock values which exceed the maximal constant appearing in guards and invariants of the timed automaton in question. In the left part of Figure 2 we show the fourty-four regions for two clocks x and y with maximal constant equal to 2. In this two-clock case, regions can be points (both clocks have integer values), open line segments (one clock has integer value, or their fractional parts are equal), open triangles, or open unbounded rectangles.

From two equivalent configurations (same location, region equivalent valuations), by delaying or by taking a transition, similar regions will be visited and similar behaviors will be possible. Regions are thus a way to finitely abstract the behaviors of a timed automaton. There are finitely (but exponentially) many regions, and by considering as abstract configurations pairs of locations and regions, we get a finite automaton, called the *region automaton*, which preserves many properties including reachability, liveness and safety. Hence verification of those properties on the original timed automaton can be transferred to the finite region automaton, and can thus be checked using exponential time. Furthermore, it is possible to design algorithms that work without constructing the whole region graph (using *on-the-fly* techniques), thus using only polynomial space.

2.3 The limits of the region abstraction

Not all properties can be decided on timed automata using the region abstraction, and language-based problems such as checking inclusion ("*Are all behaviors of a timed automaton also behaviors of another timed automaton?*") and universality ("*Can all behaviors be realized in a given timed automaton?*") are undecidable. Regarding language-theoretic questions, the set of languages (*i.e.*, set of behaviors) accepted by timed automata is not closed under complement. For instance, the following timed automaton accepts all behaviors with at least two a's separated by one time unit, but it can be proved that no timed automaton can accept the complement language. The same counterexample can be used to show that timed automata are not determinizable.



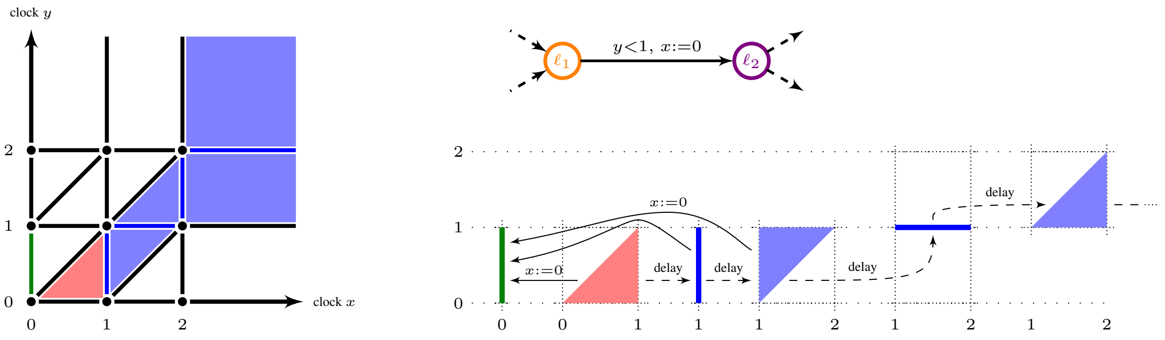


Figure 2: The region abstraction is a finite representation of all possible behaviors of the timed automaton. Consider the timed automaton on top of the picture, and assume we enter location ℓ_1 with clock values (x_0, y_0) such that $0 < y_0 < x_0 < 1$ (that is, somewhere in the red triangle, see the picture on the left); as clock y has value strictly less than 1, we have the option to switch to location ℓ_2, which would reset clock x and end up in the green region. We also have the option to delay in ℓ_1; in that case, we will exit the red triangle and reach the blue line, where $x = 1$ and $0 < y < 1$. Here again we have two options: switching to ℓ_2, or delaying to the next clock-region where $0 < y < 1 < x < 2$ and $y > 1 - x$ (that is, the fractional part of y is larger than that of x). In case we still decide to wait in that region, we reach the horizontal blue region where $1 = y < x < 2$. From that region on, the transition to ℓ_2 is not available anymore. This description of the possible behaviors starting from the red region, which has been represented on the picture to the right, does not depend on the precise values of the clocks: region equivalence preserves enough information to encode exactly the behaviors of the underlying timed automaton.

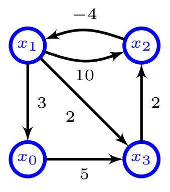
2.4 Timed automata in practice

In practice, algorithms based on the region automaton are infeasible because of their exponential time complexity. A rectification of this is provided by the so-called *zone graph* abstraction: A zone is a set of clock valuations defined by a clock constraint and can hence be represented by such, and the zone graph has as vertices pairs of locations and zones which satisfy the location's invariant, and its edges are derived from the transitions of the timed automaton under investigation. The number of zones is unbounded, so unlike the region graph, the zone graph is infinite. Finiteness can be enforced using a technique known as normalization;[11] however the number of zones is still much larger than the number of regions, and moreover the same zone can be represented using many different clock constraints.

Despite the above, zone-based algorithms have shown to be very efficient in practice, and they form the basis of a number of successful tools. The reason is again that the algorithms used have no need to explore all of the zone graph (they work *on-the-fly*), and that operations on zones can be implemented very efficiently (in time cubic in the number of clocks).

Zones are usually represented using *difference-bound matrices*, or DBMs. The DBM representation of a zone on a set of k clocks has $(k+1)\times(k+1)$ entries, where an entry $c_{i,j}$ represents a clock constraint $x_i - x_j \le c_{i,j}$ and an extra clock x_0 is added to represent clock constraints which do not involve differences of clocks. DBMs in turn can be represented as directed weighted graphs; see below for an example of a zone and its DBM (graph) representation. Canonical representations of zones can be obtained using shortest-path closure or shortest-path reduction of their DBM graphs.

$$Z = \begin{cases} x_1 \le 3 \\ x_1 - x_2 \le 10 \\ x_1 - x_2 \ge 4 \\ x_1 - x_3 \le 2 \\ x_3 - x_2 \le 2 \\ x_3 \ge -5 \end{cases}$$



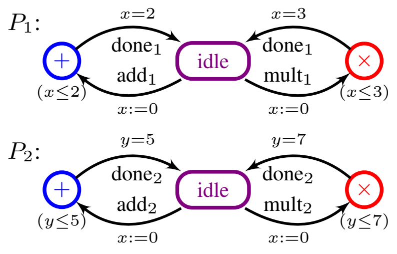
Zone-based algorithms are implemented in a number of tools, *e.g.* UPPAAL, KRONOS, HYTECH, and others, which by now have been successfully applied to numerous industrial case studies ranging from verification of real-time control programs and communication protocols to optimal real-time scheduling, as illustrated in the next section.

2.5 Task graph scheduling: time optimality

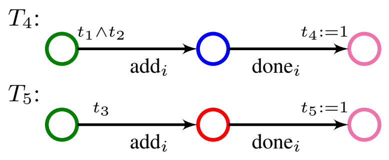
A task graph problem involves a number of tasks $T_1, \ldots, T_m$, a number of machines or processors $P_1, \ldots, P_n$, and a (partial) mapping d giving, for each task T_i and processor P_j, the time $d(i, j)$ for computing T_i on P_j. In addition there is a partial order on the tasks used for describing dependencies. Figure 3 is an example of a task graph problem.

We want to determine a schedule of when to start the execution of tasks, and on which processors, that minimizes the total execution time while being feasible in respecting the following conditions: (a) a task can be executed only if all its predecessors have completed; (b) each machine can process at most one task at a time; (c) tasks cannot be preempted.

Task graph scheduling problems may be easily modelled as networks of timed automata so that every run corresponds to a feasible schedule and the fastest run gives the time-optimal schedule: For each processor we construct a small timed automaton able — when idle— to handle within the appropriate amount of time the requests from the tasks. For the processors of Figure 3, these are as follows:



Each task is modelled as a timed automaton waiting to be served by either of the processors, conditioned by the completion of its predecessors. Tasks T_4 and T_5 of our example can be represented as follows:



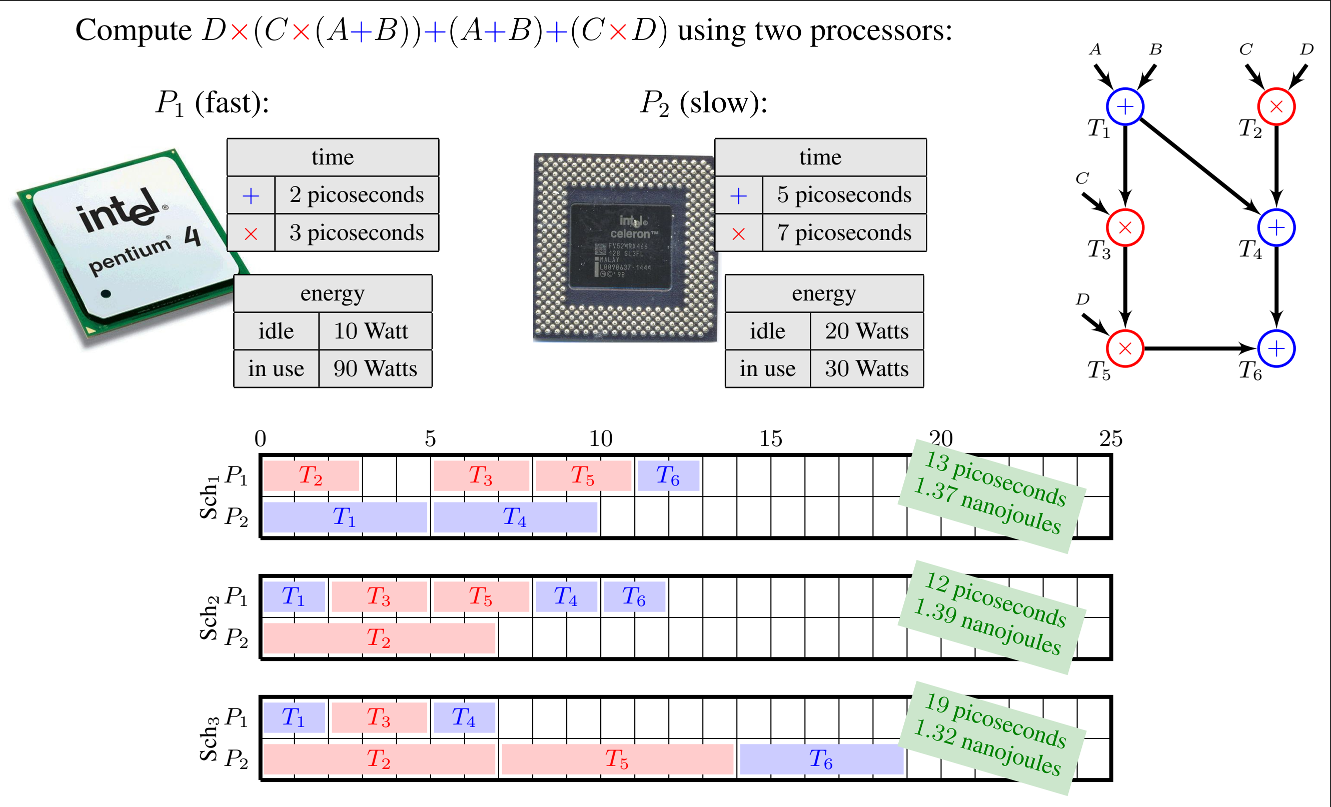


Figure 3: Task graph problem with 6 tasks, where each task corresponds to the computation of a given sub-expression of $(D \times (C \times (A + B)) + ((A + B) + (C \times D))$. Given the execution platform with two processors, P_1 and P_2, and corresponding computation times for addition and multiplication, as well as their energy consumption, Sch_1 to Sch_3 provide three feasible schedules, where Sch_2 is in fact time-optimal, and Sch_3 is energy-optimal.

Extensive experiments on benchmarks have demonstrated that the above timed automata approach to task graph scheduling is competitive compared with more traditional approaches from operations research (*e.g.* mixed-integer linear programming) as well as specialized, heuristic algorithms from planning and scheduling.[1] Furthermore the generic approach of timed automata admits easy incorporation of more specialized features (*e.g.* release times, deadlines) to the models and scheduling.

2.6 Extensions of timed automata

We have seen that timed automata are a rich formalism with efficient tool support. As such, they are often cited as the model of choice for representing and reasoning about embedded and real-time systems. This success has naturally led to several extensions of the model, for instance with more general guards or resets being allowed (*e.g.* additive guards[10] or non-deterministic udpates of clocks[15]), or with more involved dynamics measuring other quantities than time. Unfortunately, these extensions quickly lead to undecidability; *e.g.* for timed automata in which clocks can be stopped (so-called *stopwatch automata*), even basic properties such as safety or liveness are undecidable.[23]

On the other hand, the model of *hybrid automata*,[22,23] though suffering from the same undecidability problems as mentioned for other classes above, has emerged as a popular formalism for which semi-decision and approximation procedures have been developed. The *priced timed automata* which we shall discuss in the next section form an intermediate class between timed and hybrid automata for which some of the good decidability properties of timed automata are retained.

3. PRICED TIMED AUTOMATA

3.1 A model for resources

Time is not the only quantitative notion of interest when designing embedded systems; other quantities such as energy or memory consumption, required bandwidth, or accumulated cost can be of interest in such systems.

These notions are intimately connected to time, because the longer the device is operating, the more resources it consumes. This makes timed automata the model of choice to reason about those quantities, and has led to the definition of *priced timed automata*,[5,9] extending timed automata with *cost* (which is the general name we will use in the sequel to refer to the various quantities which can be modeled within this formalism).

With a look to decidability, we only allow *linearly evolving costs*: in each location, costs evolve linearly with respect to elapsed time. The important restriction which differentiates priced timed automata from linear hybrid automata (for which reachability is undecidable) is that *cost information cannot be used in the guards of the automaton*: it can only be used as an *observer variable* for evaluating the price of executions of the underlying timed automaton.

An example of a priced timed automaton, extending the timed automaton of the previous section, is depicted below:

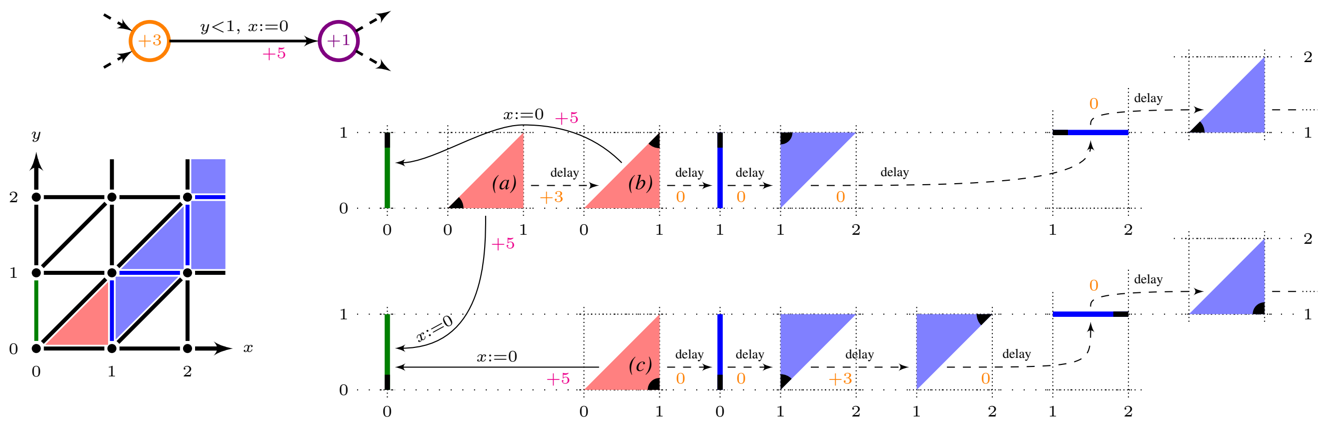
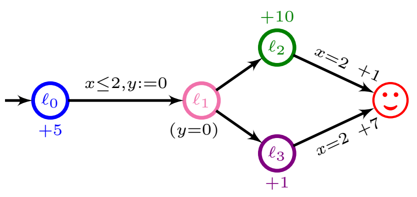


Figure 4: The corner-point abstraction refines the region abstraction by also keeping track of the corner point close to which an execution runs. This is needed to measure costs: for instance, if we are in the orange location (with cost rate $+3$) and in the red region where $0 < y < x < 1$, the price of delaying depends on the value of the clocks. From *(a)*, where both x and y are close to 0, we can let almost one time-unit elapse and reach *(b)*. The resulting cost is almost $+3$. On the other hand, from (c), where x is close to 1 and y close to 0, letting time elapse takes us "in almost no time" to the subsequent region, so that the cost is close to 0. Of course, we still have enough information to keep track of which transitions of the timed automaton are available (notice that resetting transitions from the blue regions have not been represented for the sake of readability).



A decoration $+10$ on a location indicates that cost increases by 10 per time unit in the location; a decoration $+7$ on a transition indicates that taking the transition increases overall cost by 7. The executions of such an automaton are those of the underlying timed automaton. The total cost of the example execution given in Section 2.1 is hence given by

$$1.3 \times (+5) + 0.7 \times (+1) + 7 = 14.2$$

3.2 Optimizing the resources

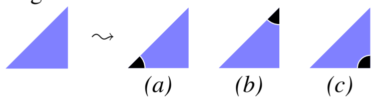
Natural optimization questions can be posed on that model, *e.g.* the *optimal reachability* problem (minimum cost for reaching a given goal), the *mean-cost optimization* problem (mean cost used in the long run), or the *discounted-cost optimization* problem (where costs are discounted exponentially as time elapses).

As an example, we compute the minimum cost that is required for reaching location ☺ in the previous example. There are two families of executions: those that go through ℓ_2 and those that go through ℓ_3. Furthermore, in each family, there is a single parameter t: the time elapsed in location ℓ_0, everything else being determined by the guards in the automaton. Hence the minimum cost is:

$$\inf_{0\le t\le 2} \min \begin{pmatrix} 5t+10(2-t)+1 \\ 5t+(2-t)+7 \end{pmatrix} = 9 \qquad (1)$$

where the expressions $5t + 10(2 - t) + 1$ and $5t + (2 - t) + 7$ give the cost of executions going through ℓ_2 respectively ℓ_3 after delaying t time units in location ℓ_0.

The standard region construction is not accurate enough to properly keep track of cost information, and a refinement of the region abstraction, the *corner-point abstraction*,[13] has to be used to solve the optimization problems mentioned above. For this abstraction, regions are refined by distinguishing their corner points. As an example, the two-dimensional region depicted below is refined into three region-corner pairs; the meaning of a region-corner pair is that the current clock valuation is in the region close to the distinguished corner:



Similar to the refinement of regions, the transitions in the region automaton have to be refined to keep track of the corners. In the example above, there is a (delay) transition from region-corner pair *(a)* to *(b)*, whereas *(c)* cannot be reached from neither *(a)* nor *(b)*. Figure 4 illustrates the corner-point abstraction of an example priced timed automaton. This graph has two types of delay edges: either within a region, from one corner to another, or from a corner of a region to the corresponding corner in the subsequent region. The first case corresponds to a delay of "almost" one time unit, while the second case corresponds to a delay of "almost" zero time units. In addition, there are edges representing transitions of the timed automaton (which reset clock x in our example of Figure 4). In that case as well, there is a natural mapping between corners.

The edges of the corner-point abstraction are labeled with discrete cost information: if the cost rate in the current location is $+3$, all one time-unit edges have label $+3$, and all zero time-unit edges get label 0. Edges coming from discrete transitions are labeled with the cost of the transition ($+5$ in the example).

The corner-point abstraction can be used to solve many optimization problems, as it can be shown that in these cases, optimal total cost is obtained for runs which always take transitions close to integer clock values. Hence the optimization problem reduces to a problem on a finite graph which can be solved using different standard techniques. This is the case for the mean-cost optimization problem[13] and the discounted-cost problem.[21] For optimal reachability, another technique of *priced regions* has been used[9] which also extends to a setting of multiple independent cost variables.[25]

As for algorithm and tool support, the zone-based approach has been successfully extended to solve the optimal reachability problem,[24] by introducing *priced zones*, and tool support is available in UPPAAL CORA. For mean-cost and discounted-cost optimization, active research is being conducted in developing efficient zone-based algorithms,

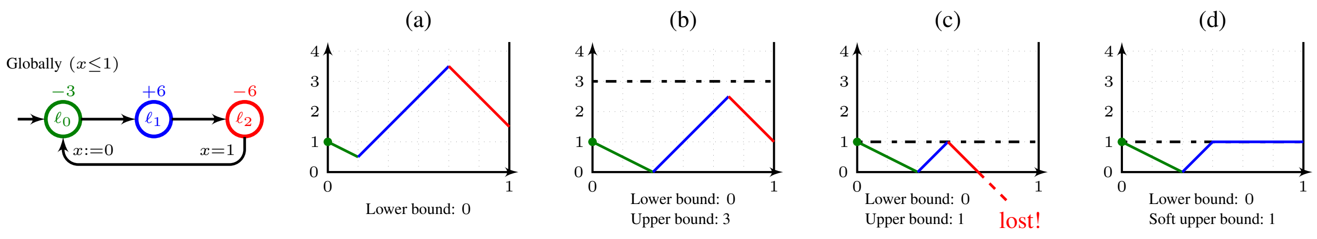


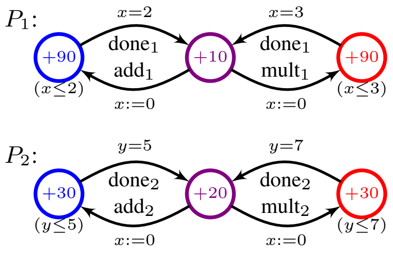
Figure 5: The resource management problem asks whether it is possible to maintain the cost level within fixed bounds. There can be only a lower bound (a), a lower and an upper bound (b, c), or a strong lower and a soft upper bound. Figures (a), (b) and (d) represent solutions to the respective problems for the priced timed automaton depicted on the left: there is an infinite path that satisfies the global constraint. In case (a) for instance, we have depicted a possible schedule for the first cycle, and this path can be repeated because at the start of the second cycle, the cost level is larger than at the start of the first cycle. In figure (c), the proposed schedule violates the lower bound, and it can be shown that there exists no infinite path which maintains cost level within the specified bounds.

or alternatively showing that no such algorithms exist.

3.3 Task graph scheduling: energy optimality

Reconsidering our running task graph scheduling problem of Section 2.5, cost-optimal reachability for priced timed automata may be used to provide energy-optimal schedules.

For the task graph scheduling instance of Figure 3, energy consumption of the two processors is reflected in the respective timed automata by suitable cost-rates in the locations corresponding to the processor being idle or in use. The processors can then be represented by the following two priced timed automata:



Thus the energy needed for performing computation steps is modeled as cost in the priced timed automaton model, and optimal reachability techniques can be employed for finding an energy-optimal schedule.

3.4 Managing the resources

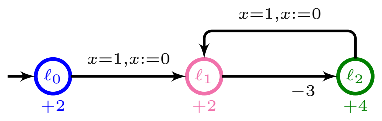
Up to this point we have only employed priced timed automata as a formalism for modeling time-dependent *consumption* of resources. However, in several situations resources may not just be consumed but also occasionally regained, *e.g.* in rechargeable batteries, autonomous robots equipped with solar cells for energy harvesting, or in tanks which may not only be emptied but also filled. Extending priced timed automata to allow for both positive (regaining) and negative (consumption) rates provides a natural modelling formalism.

However, a new question now emerges related to the appropriate management of resources: "*Is it possible to maintain the level of resources within fixed bounds?*" Such resource-bound problems are highly relevant to the analysis of several embedded systems, *e.g.* it is natural to plan the usage of a device with rechargeable batteries or solar cells so that one never runs out of energy, nor exceeds the maximum capacity for energy storage.

This problem goes one step further towards (linear) hybrid automata: while costs can still not be used directly in guards, they restrict the possible behaviors of the timed automaton by forbidding transitions that would make the total cost go below the lower bound or above the upper bound. There are several variants to this problem: there can be no upper bound (when representing the evolution of a bank account for instance), a "soft" upper bound (once reached, the cost cannot increase anymore, as is the case for a rechargeable laptop battery), or a strong upper bound (for instance when modeling the level of oil in a closed tank which might explode if it exceeds the maximum capacity). Figure 5 shows a priced timed automaton together with some resource management problems.

Few results have been obtained on this problem so far: only the case of *one-clock* priced timed automata has been investigated.[16] This restriction has two important consequences: Cycle detection can be done statically, as each resetting transition leads to a configuration with clock value 0, and the region automaton can be coarsened so that the partition consists of intervals with endpoints given by the constants in the automaton's guards. As a consequence, there are only polynomially many regions.

For priced timed automata with more than one clock, no results are known, but even for one-clock automata, there are some difficulties which mean that abstractions like the region automaton or the corner-point abstraction are insufficient. As an example, consider the following priced timed automaton:



Assuming that we start with initial cost 0, this automaton has exactly one feasible execution in which the cost level remains non-negative: after spending 1 time unit in location ℓ_0, we alternately spend half a time unit in ℓ_1 and half a time unit in ℓ_2. Any other execution eventually violates the lower bound.

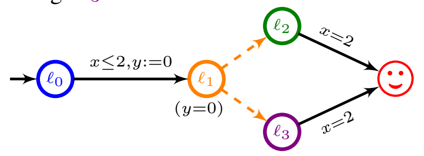
Hence with discrete costs on transitions, the abstractions introduced above do not suffice even for one-clock priced timed automata. If one restricts to automata *without discrete costs*, the corner-point abstraction can be used to show decidability of the management problem for lower bound only, and for strong lower and soft upper bound; in these cases, feasible executions proceed close to corners of regions, and the problem is solvable in polynomial time.

4. PRICED TIMED GAMES

4.1 A model for uncertainties

The systems we have considered so far are *closed* in the sense that we have a complete description of the system. This is not sufficient to model embedded systems where interaction with the environment is crucial, or systems with some imprecisions. These can be modelled using (two-player) *timed games*,[7] in which some actions are triggered by the environment (we can think of signals received by sensors, or of unexpected events). The aim is to *control*, or *guide*, the system so that it will be safe or correct regardless of the way the environment interferes. An example of a timed game is depicted below.

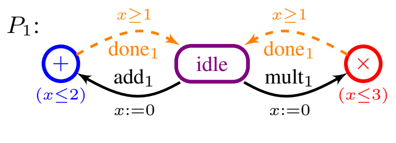
The dashed edges are those of the environment (said to be *uncontrollable*): when they are firable, the system cannot prevent (nor force) them to be fired. Here, the system cannot decide whether it goes through ℓ_2 or through ℓ_3.

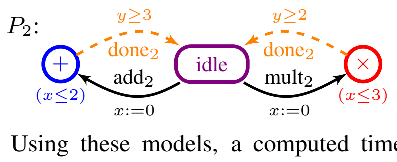


For simple correctness criteria, *e.g.* reachability or safety, the set of *winning states* (*i.e.*, states from which the system can be controlled under the safety constraint) and also *winning strategies* (*i.e.*, policies for how to control the system) can be computed using the region abstraction.[7] Also computability of time-optimal strategies,[6] as well as strategies under partial observability based on discrete observations,[20] has been demonstrated. For efficient algorithms, a zone-based approach for solving timed games with reachability and safety objectives has been developed,[19] and tool support is now available in UPPAAL-TIGA.[8] Using a subset construction, also strategies under partial observability can be computed with this approach.

4.2 Task graph scheduling: timing uncertainty

Returning to our running task graph scheduling example, we can use the formalism of timed games to model uncertainty in precisely how much time a certain computation on a given processor takes. In Section 2.5 we modeled computation times by precise numbers, whereas we now can make the model more realistic by only providing *interval bounds* within which computation times are prescribed to lie. The timed game models below provide version of the processors P_1 and P_2 from Figure 3 in which computation times are prescribed to lie in the intervals $[1, 2]$ for addition and $[1, 3]$ for multiplication on P_1, and similarly for P_2. Note that uncertainty of precise computation times has been modeled by uncontrollable edges:

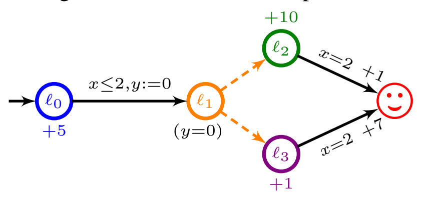




Using these models, a computed time-optimal schedule will no longer be a simple fixed assignment of tasks and time slots to processors, but rather a flexible dynamic assignment, where task scheduling can be adapted on-line according to actual completion times of previous tasks.

4.3 Cost-optimal strategies

It is natural to extend the timed game framework with cost information, hence making it possible to model uncertainty as well as resource use, and to ask for *controllability under resource constraints*, or for *optimal* controllability. The model of *priced timed games* is a synthesis of priced timed automata and timed games; we show an example below:



In the above example we may *e.g.* want to compute the minimum cost for reaching location ☺ regardless of the moves of the environment (which is in charge of the edges out of ℓ_1 as before). As the system cannot control whether execution goes through ℓ_2 or ℓ_3, the minimum cost is given by the formula

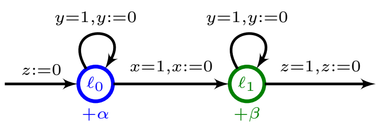
$$\inf_{0\le t\le 2} \max\begin{pmatrix} 5t+10(2-t)+1 \\ 5t+(2-t)+7 \end{pmatrix} = 14+\frac{1}{3}$$

and the strategy is to wait $\frac{4}{3}$ time units in ℓ_0. Hence techniques based on the region automaton or the corner-point abstraction are not sufficient for computing optimal-reachability strategies, even in case of one-clock priced timed games.

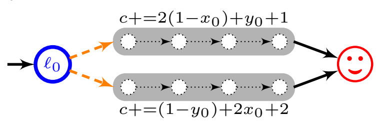
Generally, priced timed games are much more difficult to analyse than priced timed automata. Cost-optimal strategies are undecidable,[18] even when restricted to priced timed games with only three clocks.[12]

Decidability has been shown for classes of priced timed games with strong conditions on the cost evolution[2,14] and for one-clock priced timed games.[17] The reason for the latter is the same as for one-clock priced timed automata above: resetting the clock leads to a configuration with a known clock valuation.

The mentioned undecidablity results have been shown by reduction from the halting problem for two-counter, or *Minsky*, machines. The essence of the reduction is that given three clocks x, y and z, it can be checked whether the value of y is twice the value of x. Consider the following module, which is to be part of a larger priced timed game:



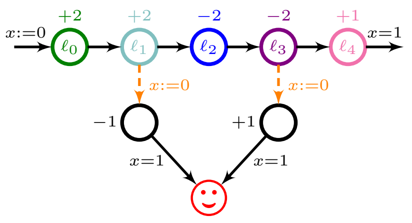
Along an execution in this module, total cost is increased by $\alpha(1-x_0)+\beta x_0$, where x_0 is the initial value of x when entering the gadget. Using this and similar other constructions, modules which increase total cost by $2(1-x_0)+y_0+1$ respectively $(1-y_0)+2x_0+2$ can be designed. These will be integrated in another module and depicted as gray boxes.



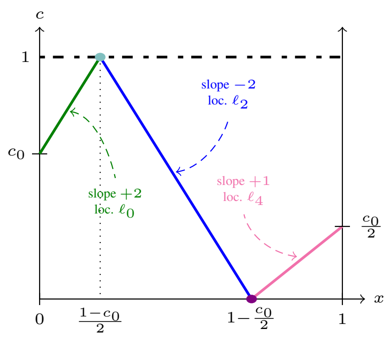
By requiring that the optimal strategy has total cost no more than 3, this module can be used to ensure that the value of y is twice the value of x. Indeed, if $y_0 > 2x_0$, the opponent can choose the upper of the two gray sub-modules to increase total cost above 3, and if $y_0 < 2x_0$, the opponent can choose the lower sub-module. Similar modules can be used to ensure other arithmetic properties of clock values, and collecting these appropriately, a two-counter machine can be encoded as a priced timed game.

4.4 Resource management

The formalism of priced timed games can also be used to model resource management problems under uncertainty, such as *e.g.* the problem of programming an autonomous robot so that it always is able to reach its base before running out of energy, even if some obstacles can interfere. As in the automaton framework, this problem has been addressed only quite recently,[16] and many problems are still open. Preliminary results are rather negative: finding a strategy that keeps the total cost within a given interval is undecidable, even if the timed game has only one clock. The reason is again that it is possible to implement basic arithmetic operations, but this time on the value c of the total cost. Consider the following module:



If c is to be kept within the interval $[0, 1]$, then the total cost when leaving this module must be exactly half of what it was when entering. Indeed, the only winning strategy is to wait in location ℓ_0 until c reaches value 1, then switch to ℓ_1 and immediately pass to ℓ_2 and wait until c reaches 0, and then pass over ℓ_3 and spend the remaining time in ℓ_4. Any other strategy lets the opponent take the system to the ☺ location with total cost below 0 or above 1. The resulting behavior can be depicted as follows:



This is the basic step in encoding a two-counter machine as a resource management problem on a one-clock priced timed game, entailing that this problem is undecidable.

5. CONCLUSIONS

Timed automata and their priced and game extensions provide a uniform and expressive formalism for dynamic resource allocation, allowing for analysis of a wide collection of performance and optimization problems, with results competitive with respect to more traditional approaches such as mixed-integer linear programming or others.

With the trend towards multi-core and multi-processor architectures in embedded systems, classical scheduling theory for one-processor systems no longer applies, and there is an increasing usage of timed-automata technology to schedulability and performance analysis in these settings.

Particularly challenging problems remaining to be settled include decidability of synthesis for priced timed games under partial observability, as well as a range of resource management problems in the setting of priced timed automata and games with both consumption and regaining of resources.

Acknowledgments

The authors are partly supported by the European project QUASIMODO (FP7-ICT-STREP-214755). The French authors are also supported by project DOTS (ANR-06-SETI-003).

6. REFERENCES

[1] Y. Adbeddaïm, A. Kerbaa, and O. Maler. Task graph scheduling using timed automata. In *Proc. 17th International Parallel and Distributed Processing Symposium (IPDPS'03)*, page 237. IEEE Computer Society Press, Apr. 2003.

[2] R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

[3] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[4] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[5] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001.

[6] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Proc. 2nd International Workshop on Hybrid Systems: Computation and Control (HSCC'99)*, volume 1569 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[7] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier Science, 1998.

[8] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. UPPAAL-TIGA: Time for playing games! In *Proc. 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007.

[9] G. Behrmann, A. Fehnker, Th. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[10] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1–2):1–7, 2000.

[11] P. Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

[12] P. Bouyer, Th. Brihaye, and N. Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.

[13] P. Bouyer, E. Brinksma, and K. G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):2–23, 2008.

[14] P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2004.

[15] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2–3):291–345, 2004.

[16] P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, Lecture Notes in Computer Science. Springer, 2008.

[17] P. Bouyer, K. G. Larsen, N. Markey, and J. I. Rasmussen. Almost optimal strategies in one-clock priced timed automata. In *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

[18] Th. Brihaye, V. Bruyère, and J.-F. Raskin. On optimal timed strategies. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005.

[19] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.

[20] F. Cassez, A. David, K. G. Larsen, D. Lime, and J.-F. Raskin. Timed control with observation based and stuttering invariant strategies. In *Proc. 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, volume 4762 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2007.

[21] U. Fahrenberg and K. G. Larsen. Discount-optimal infinite runs in priced timed automata. In *Proc. 10th International Workshop on Verification of Infinite-State Systems (INFINITY'08)*, 2008. To appear.

[22] Th. A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual Symposim on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.

[23] Th. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[24] K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, Th. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 493–505. Springer, 2001.

[25] K. G. Larsen and J. I. Rasmussen. Optimal reachability for multi-priced timed automata. *Theoretical Computer Science*, 390(2-3):197–213, 2008.