On Zone-Based Analysis of Duration Probabilistic Automata

Oded Maler
CNRS-VERIMAG
University of Grenoble
France
Oded.Maler@imag.fr

Kim G. Larsen
CISS and CS
Aalborg University
Denmark
kgl@cs.aau.dk

Bruce H. Krogh
Department of EC
Carnegie Mellon University
USA
krogh@ece.cmu.edu

We propose an extension of the zone-based algorithmics for analyzing timed automata to handle systems where timing uncertainty is considered as *probabilistic* rather than *set-theoretic*. We study *duration probabilistic automata* (DPA), expressing *multiple* parallel processes admitting *memoryfull* continuously-distributed durations. For this model we develop an extension of the zone-based forward reachability algorithm whose successor operator is a *density transformer*, thus providing a solution to verification and performance evaluation problems concerning acyclic DPA (or the bounded-horizon behavior of cyclic DPA).

1 Introduction

Timed automata [4] handle temporal uncertainty in a set-theoretic manner consistent with the *worst-case* spirit of safety-critical verification. Performance evaluation of systems of a less dramatic nature is typically based on a stochastic interpretation of temporal uncertainty. A well-studied class of such systems are *continuous-time Markov chains* (CTMC) where durations are distributed exponentially and model-checking against temporal properties is well understood [7]. More general distributions fall under the category of *generalized semi-Markov processes* (GSMP) [18, 17, 15] and other similar models such as *stochastic timed automata* [16, 10] or *stochastic Petri nets* [21, 8]. Good overviews of these issues can be found in [12, 11]. Some approaches for verifying such systems against qualitative [3] and quantitative [22] properties have been proposed based on partitioning the state space into equivalence classes in the spirit of the *region graph* [4] and performing the analysis on the finite quotient which can be viewed as a discrete-time Markov chain. Although the region graph underlies the fundamental decidability results for timed automata, it is not used in any existing verification tool, due to its prohibitive size. Verification tools [27, 24] use reachability computation on *zones* [19], a class of polyhedra that represent reachable sets of states and clock valuations.[1]

We extend the zone-based reachability computation to handle timed automata with probabilistic durations. We use a variant of stochastic timed automata that we call *duration probabilistic automata* inspired by the class of timed automata encountered while modeling scheduling problems [1]. Such automata can model tasks admitting precedences and resource constraints, with the duration of each task being probabilistically distributed. We focus on *uniform* distributions but the proposed approach will work with any polynomial distributions with bounded support. To analyze such systems we decorate zones with clock *densities*, and define *successor* operators that act as *density transformers* that allow us to compute the clock distribution upon taking a particular transition from state q based on the clock distribution at the entrance into q. As a result we can assign probabilities to interesting subsets of the *timed language* generated by the automaton.

[1] Theoretically the number of zones can be even higher than the number of regions but in practice it is much lower.

Submitted to:
INFINITY 2010

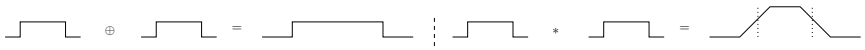

Figure 1: A Minkowski sum of intervals versus a convolution of two probability density functions. The area outside the dotted vertical lines corresponds to low-probability behaviors that can be ignored in certain circumstances.

The rest of the paper is organized as follows. Section 2 is a self-contained introduction to the modeling of timing uncertainty in concurrent systems and its algorithmic analysis. In Section 3 we define duration-probabilistic automata. Section 4 is devoted to a summary of the reachability graph construction used to compute the semantics of timed automata. In Section 5 we present our contribution, the extension of this technique for DPA using density transformers while Section 6 mentions related and future work.

2 Timing Uncertainty: Modeling and Analysis

Discrete concurrent processes can be analyzed at different levels of abstraction with respect to time. To illustrate this point consider two concurrent systems, one that performs two tasks sequentially and one that preforms a third task in parallel and let events a, b, and c denote the respective *terminations* of these tasks. At the most abstract level one assumes nothing about the relative *durations* of the processes and hence all the sequences in the shuffle $ab||c = \{abc, acb, cab\}$ are considered feasible. The first refinement of the model is provided by models such as *timed automata* or *timed Petri nets*, where the durations of a, b and c are specified to be bounded in the intervals $[l_a, u_a]$, $[l_b, u_b]$ and $[l_c, u_c]$, respectively. In this model, knowing, for example, that $l_c > u_a$ we conclude that c cannot occur before a and hence cab is impossible. Likewise, abc is impossible when $u_c < l_a + l_b$.

While this refinement of the untimed model adds a lot of information, this *set-theoretic nondeterminism* which states only *what* is possible but does not quantify the likelihood of different possibilities, is still too *qualitative* for certain purposes as the following example demonstrates. Consider a sequence of k processing steps, each of which with duration in $[l,u]$. From a purely "measureless" set-theoretic viewpoint, the termination time of the whole sequence of steps can be anywhere in $[kl,ku]$. Intuition tells us, however, that a duration of ku, whose realization requires that each of the steps takes the maximal time to terminate, is less likely than, say, an "average" duration of $k(l+u)/2$.[2] On the other hand if we interpret the interval $[l,u]$ as, say, a uniform distribution with density $1/(u-l)$, the total duration of the k-step sequence is still restricted to the interval $[kl,ku]$, but with probability which is larger in the middle of the interval and smaller toward the boundaries. In a nutshell, this is the difference between a Minkowski sum of intervals $[l,u] \oplus [l,u]$ and the *convolution* $\psi_1 * \psi_2$ of two functions defined over those intervals, see Fig. 1. Assigning probabilities to the runs of the automaton we can, for example, distinguish between different degrees of property violations or compute the expected value over all runs of some performance measure.

The use of automata with clocks has some advantages over the standard language of stochastic processes, in particular, the ability to express more sophisticated synchronization mechanisms between processes, such as schedulers that resolve resource conflicts. These are expressed naturally in the language

[2] Another example of a more discrete nature is the modeling of computer memory access where worst case duration (cache miss) is orders of magnitude larger than the normal case (cache hit) and if we want to be conservative and assume that both cases are possible in each and every instance, our performance estimation will be overly pessimistic and practically useless. Timed automata with probabilities on transitions have been studied in [20, 23].

Figure 2: A process that takes time: (a) standard description; (b) decoupling the non-deterministic choice from the *end* transition.

of states and transitions while translating them into conditionals based on inequalities over values of random variables may be cumbersome. Computationally, a state-based approach provides for iterative forward or backward computations, for both analysis and scheduler synthesis, more flexible than methods based on a holistic analytical solution.[3] For timed automata, this iterative computation works on *sets* of clock valuations (zones) [19, 27, 24] that each qualitative sequence of events may lead to, where clock values eliminate qualitative behaviors which are infeasible due to timing. In the probabilistic setting, we decorate zones with additional probability information concerning runs and clock values. Thus we can eliminate classes of behaviors which are feasible but unlikely. Hopefully, the non-negligible overhead associated with computing probabilities will be compensated by the liberty not to explore paths of low probability.

We consider processes constructed from very simple components such as the automaton of Fig. 2(a). Such a process is in a waiting state, until it takes a *start* transition and moves to an active state. Clock x, which is set to zero upon the transition, measures the time elapsed since the activation. A *start* transition is instantaneous and is initiated by some external scheduler/supervisor. The timing of an *end* transition is based on the clock value and the temporal guard $\phi(x)$, which in the case of timed automata, is simply the condition $x \in [l,u]$. In duration probabilistic automata we associate a probability density with the duration of each step which is technically expressed as the distribution over the values of clock x when the *end* transition is taken, (note that once started, a process cannot be aborted). We want to analyze the behavior of *multiple* such systems running *concurrently*, each with its own clock.

We use a slightly modified (but equivalent) version of the basic automaton, as shown at Fig. 2(b). Rather than having the *start* transition deterministic and delegating the non-determinism to the *end* transition, we use an auxiliary variable y which is assigned non-deterministically upon *start* and which should be equal to x upon *end*. In the set-theoretic setting this means an assignment $y \in [l,u]$ while for DPA this means drawing a value for y according to ϕ, which we denote by $y := \phi()$.

The fundamental phenomenon in the analysis of continuous-time stochastic processes is that of a *race* which occurs in a global state where two or more processes are active. We would like to know which process terminates first, in other words, via which of the pending *end* transitions will the automaton leave the state. The outcome of a race depends on two factors: the random choices of the respective task durations (the y variables) and the values of the clocks upon *entering* the global state. Fig. 3 shows a fragment of a global automaton representing two parallel processes, both active at state q. Clock x_2 was reset upon entering q, while clock x_1, corresponding to a different process that has not yet terminated, was reset in a preceding global state. The gap between the two starting times is maintained by the difference $x_1 - x_2$ which remains constant throughout the sojourn in q. The larger is this difference, the more likely is clock x_1 to satisfy its temporal guard by reaching y_1 before x_2 reaches y_2.[4]

[3] We use *automata* here as a generic term for discrete transitions systems. Some of the advantage attributed to them in terms of modeling expressivity and analysis techniques apply, at least in principle, to other similar formalisms such as Petri nets for which an approach similar to ours has been developed in [26], see Section 6.

[4] It is interesting to note that in the stochastic processes literature [15] the role of x and y is taken by a *single* timer $z = y - x$

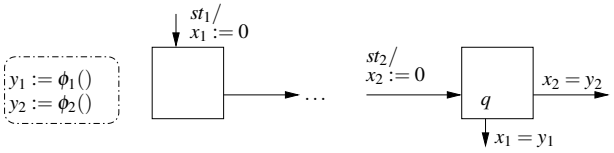

Figure 3: A race.

In the probabilistic setting, this is rephrased as follows. Suppose we enter a global state with some probability over clock values, and in this state there are several pending *end* transitions guarded by probabilistically-chosen durations. The probabilities over the clock values upon entrance together with the probabilities over the durations determine the probability that a certain transition wins the race and is taken, as well as the probabilities over the clock values upon taking each of the transitions. We develop a computational scheme for computing for every finite sequence of events the probability that it occurs and the probability over the clock values upon the occurrence of its last event. Technically this is achieved via the concept of a *density transformer* which extends the symbolic *successor* operator of timed automata (which deserves to be called a subset/zone transformer) to an operator on (partial) densities over clock values.

3 Definitions

Throughout this paper we use $\mathbb{T} = [0,\infty)$ as a time domain on which we define probabilities. We use a fixed set of clock variables $X = \{x_1,\ldots,x_n\}$ all ranging over $\mathbb{T}$ or a bounded subset of it.

Definition 1 (Clock Constraints and Zones) *The set of clock constraints over X, is defined by the following grammar: $\varphi ::= true \,|\, x_i \prec k \,|\, x_i - x_j \prec k \,|\, \varphi \wedge \varphi'$, where $x_i, x_j \in X$, $k \in \mathbb{N}$ and $\prec \in \{<, \leq, =, \geq, >\}$. The set of points satisfying a clock constraint is called a zone*

Each zone is a convex polytope in some dimension $m \leq n$ defined as the intersection of half-spaces which are either orthogonal ($x_i \prec k$) or diagonal ($x_i - x_j \prec k$) with integer k. There are finitely many zones in any bounded subset of $\mathbb{T}^n$ or any of its subspaces. We use $\perp$ to denote the zone associated with dimension zero (where no clock is active).

Definition 2 (Time Densities) *A piecewise-continuous function $\phi : \mathbb{T} \to \mathbb{T}$ is a time density if it satisfies*

$$\int_0^\infty \phi(\tau) d\tau = 1.$$

A density has a bounded support $[a,b] \subset \mathbb{T}$ if $\phi(\tau) \neq 0 \leftrightarrow \tau \in [a,b]$. A bounded support density is uniform if $\phi(\tau) = 1/(b-a)$ when $\tau \in [a,b]$.

The generalization to higher dimension is:

---

which is a clock going with derivative -1 to zero, after being assigned a random duration. The difference between the two formulations is that ours distinguishes the information that is observable at any time, the value of x, from the information that is observed only upon termination, the actual duration y. This two-variable representation may provide for more refined *dynamic* schedulers that can base their decisions on the value of x, as demonstrated in [1].

Definition 3 (Clock Densities) *A function $\psi : \mathbb{T}^m \to \mathbb{T}$ is a clock density if it satisfies*

$$\int_0^\infty \dots \int_0^\infty \psi(\tau_1, \dots, \tau_m) d\tau_1 \dots d\tau_m = 1.$$

We will consider clock densities whose supports are zones.[5]

Abusing terminology we call ψ a *partial density* if the above integral is smaller than 1.

To define the behaviors of our automata we will use timed words (the *time-event sequences* of [5]) over an alphabet Σ of events which will correspond to the various *start* and *end* actions.

Definition 4 (Timed Words and Languages) *A timed word over a finite alphabet Σ is a concatenation of the form $\xi = t_1 \cdot w_1 \cdot t_2 \cdot w_2 \cdots$ where $t_i \in \mathbb{T}$ and $w_i \in \Sigma^+$. The untiming of ξ is $\mu(\xi) = w_1 \cdot w_2 \cdots$ and its duration is $\lambda(\xi) = \sum_i t_i$. A timed language is a set of timed words.*

Intuitively this object represents an alternation between passages of time of duration t_i, followed by sequences w_i of one or more instantaneous events. The events will be *start* and *end* transitions and time passages correspond to time elapsing in active states. All events in w_i occur at the same absolute time instant $\sum_{j=1}^{i} t_j$ but in order not to extend the alphabet to 2^Σ we will consider them as occurring sequentially. We use ε for the empty word. A timed word ξ' such that $\mu(\xi') = \mu(\xi)$ agrees with ξ on the *order* of events. All such behaviors form an equivalence class $[\xi]$ that we sometime refer to as a *qualitative behavior*.

Duration probabilistic automata (DPA) constitute a well-structured class of timed automata obtained as products of simple DPA and a scheduler. They can model most situations encountered in the analysis of scheduling problems such as job-shop or task-graph [1] and are free from notorious anomalies such as Zeno behaviors. For economy of expression, we use as our building blocks processes that admit several processing steps where the *end* transition of step j leads to the waiting state of step $j+1$. Although practically, the same clock can be reused in subsequent steps, conceptually we prefer sometimes to view each step j as using a distinct clock x^j. Let $N = \{1, \dots, n\}$ and $K = \{1, \dots, k\}$.

Definition 5 (SDPA) *A simple duration probabilistic automaton (SDPA) of k steps is a tuple $\mathscr{A} = (\Sigma, Q, X, Y, \Delta, \overline{q}^1)$ where $\Sigma = \Sigma_s \uplus \Sigma_e$ is the alphabet of* start *and* end *actions with $\Sigma_s = \{s^1, \dots, s^k\}$ and $\Sigma_e = \{e^1, \dots, e^k\}$. The state space is an ordered set $Q = \{\overline{q}^1, q^1, \overline{q}^2, \dots, q^k, \overline{q}^{k+1}\}$ with $\overline{q}^j$ states considered* idle *and q^j states are* active*, $X = \{x^1, \dots, x^k\}$ is a set of clock variables and $Y = \{y^1, \dots, y^k\}$ is a set of auxiliary random variables, each distributed according to a bounded and uniform time density ϕ^j. The transition relation Δ consists of two types of transitions:*

1. Start *transitions: for every idle state $\overline{q}^j$, $j \in K$, there is one transition of the form $(\overline{q}^j, s^j, \{x^j\}, q^j)$. When the transition is taken, clock x^j is reset to zero and becomes active. Such transitions take no time;*
2. End *transitions: for every active state q^j, $j \in K$, there is a transition of the form $(q^j, x^j = y^j, e^j, \overline{q}^{j+1})$. This transition renders clock x inactive.*

State $\overline{q}^1$ is the initial state of $\mathscr{A}$.

The SDPA just defined is acyclic. A cyclic version of this definition, employs addition modulo k with the last transition going back to $\overline{q}^1$, see Fig. 4. In this paper we restrict ourselves to acyclic automata.

The operational interpretation is the following: for each step j we draw a duration y^j according to ϕ^j. Inside an active state q^j, clock x^j advances with derivative 1 and the *end* transition is taken when

[5] More precisely, due to resets that put all the probabilistic mass of some clocks at zero, we have to deal with hybrid objects that combine discrete and continuous probabilities and can be framed in terms of densities using impulse functions.

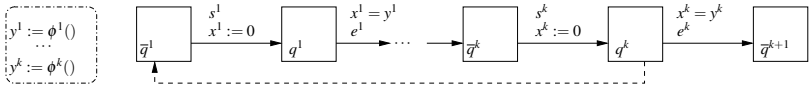

Figure 4: A simple DPA: acyclic and cyclic (dashed transition).

$x^j = y^j$, that is, y^j time after the corresponding *start* transition. A generalized state (configuration) of the automaton in an active state is a pair (q,v) consisting of a discrete state and a clock value v which represents the time elapsed since the last *start* transition. Note the difference between transition labels s^j and e^j: the former is an external command coming from a *scheduler* outside the SDPA, while the latter is emitted by the SDPA itself when it terminates a step within a randomly chosen duration. When such a scheduler is not specified, the automaton can be viewed as non-deterministic, generating behaviors of the form

$$r^1 \cdot s^1 \cdot t^1 \cdot e^1 \cdot r^2 \cdot s^2 \cdot t^2 \cdot e^2 \cdot \cdots \cdot s^k \cdot t^k \cdot e^k \cdot \infty$$

with each $r^j \in \mathbb{T}$ being an arbitrary waiting period and each t^j is in the support of ϕ^j.

Duration probabilistic automata (DPA) are obtained by composing a set $\{\mathscr{A}_i = (\Sigma_i, Q_i, X_i, \Delta_i, \overline{q}_i^1)\}_{i\in N}$ of SDPA with a *scheduler*. To simplify notations we assume all $\mathscr{A}_i$ to admit the same number k of steps. The event alphabet is the union of the event alphabets Σ_i, that we write as $\Sigma = \Sigma_s \uplus \Sigma_e$ with $\Sigma_s = \{s_i^j : i \in N, j \in K\}$ and $\Sigma_e = \{e_i^j : i \in N, j \in K\}$. The state space of the product automaton is $Q = Q_1 \times \cdots \times Q_n$. The composition of automata, which is fairly standard in the non-deterministic setting, often employs an *interleaving semantics* where independent transitions can occur in any order. Applying this approach to several *start* transitions that take place *simultaneously*, introduces an annoying artificial non-determinism that we avoid by combining all transitions that occur simultaneously into a single transition. However in order to maintain the semantics of the automaton as a set of timed words over Σ we will associate with such a transition a unique *sequence* of labels. This is done via a *sequentialization function* which maps every $E \subseteq \Sigma$ into a sequence $\alpha(E) \in \Sigma^+$ consisting of the elements of E concatenated according to some fixed order relation over the alphabet. We say that transition s_i^j is *enabled* in global state q if the i^{th} component of q is $\overline{q}_i^j$.

Definition 6 (Scheduler) *A scheduler for a set $\{\mathscr{A}_i\}_{i\in N}$ of SDPA is a function $S : Q \to 2^{\Sigma_s}$, satisfying:*

- *$s_i^j \in S(q)$ only if s_i^j is enabled in q;*
- *$S(q) = \emptyset$ only if q is the global final state or admits at least one active component.*

The scheduler plays two roles in our model. First, it guarantees mathematical sanity with a single run for every value of the random variables and a non-blocking behavior where all prefixes of runs have continuations that reach the final state in a bounded amount of time. In a world of unlimited resources where each SDPA may progress independently, $S(q)$ is the set of all transitions enabled in q and the scheduler is restricted to this mathematical role. The more interesting case is when the scheduler has to resolve resource conflicts and keep some processes waiting while giving priority to others. Abusing notation we say that $i \in S(q)$ if $s_i^j \in S(q)$ for some j.

Definition 7 (Duration Probabilistic Automata) *A duration probabilistic automaton (DPA) is a composition $\mathscr{A} = \mathscr{A}_1 \circ \cdots \circ \mathscr{A}_n \circ S = (Q, X, Y\Delta, q^0)$ of n SDPA and a scheduler. The state space is $Q \subseteq Q_1 \times \cdots Q_n$ with initial state $q^1 = (\overline{q}_1^1, \ldots, \overline{q}_n^1)$ the set of clocks[6] is $X = \bigcup_i X_i$ and the auxiliary variables*

[6]Since at any time there is at most one clock active for each $\mathscr{A}_i$, we will sometimes refer to the set of clocks as $\{x_1, \ldots, x_n\}$ where x_i refers to some x_i^j depending on the state of $\mathscr{A}_i$. Likewise we will compare it with y_i denoting the appropriate y_i^j.

$Y = \bigcup_i Y_i$. *The transition relation* Δ *consists of two types: multiple* start *transitions of the form* (q, w, R, q') *where* $w \in \Sigma_s^+$ *is a sequence of labels and* R *is a set of initialized clocks, as well as* end *transitions of the form* $(q, x_i = y_i, e_i, q')$*, one for each* $\mathscr{A}_i$ *active in* q.

- *For every state* $q = (q_1, \ldots, q_n)$ *such that* $S(q) = E \neq \emptyset$ *we define a transition*

$$((q_1, \ldots, q_n), w, R, (p_1, \ldots, p_n))$$

  where $w = \alpha(E)$ *is the sequentialization of* E *and* $R = \{x_i^j : s_i^j \in E\}$. *When* $i \notin E$ $p_i = q_i$ *otherwise* $p_i = q_i'$ *where* $(q_i, s_i, \{x_i\}, q_i') \in \Delta_i$ *is the corresponding* start *transition;*

- *For every* q *such that* $S(q) = \emptyset$ *and for every* i *such that* q_i *is active and* $(q_i, x_i = y_i, e_i, q_i') \in \Delta_i$ *is its corresponding* end *transition, we define a transition*

$$((q_1, \ldots, q_i, \ldots, q_n), x_i = y_i, e_i, (q_1, \ldots, q_i', \ldots, q_n))$$

This definition gives priority to the immediate *start* transitions while the pending *end* transitions are allowed only in a state where no immediate transitions are admitted by the scheduler.

4 Behaviors and their Computation

The set of all complete behaviors that a DPA $\mathscr{A}$ may generate constitutes a timed language $L = L(\mathscr{A})$. The probabilistic semantics of $\mathscr{A}$ is a probability distribution over subsets of L. We will not give at this point a detailed formal definition of this semantics but rather convey sufficient intuition to relate it to the zone-based computation that we develop in the sequel. For the sake of simplicity, we temporarily assume a most liberal scheduler which executes every s_i^j immediately after e_i^{j-1}. The untiming $\mu(L)$ of the language consists of words satisfying some well-formedness condition, that is, $\mu(L) \subseteq M$ where $M = M_1 || \cdots || M_n$ is the shuffle of the SPDA local languages, each of the from $M_i = \{s_i^1 \cdot e_i^1 \cdots s_i^k \cdot e_i^k\}$. By construction, there is a one-to-one correspondence between sequences of events in M and complete paths in $\mathscr{A}$. Hence L can be written as a union $\bigcup_{w \in M} L_w$ of languages, each corresponding to a subset of L corresponding to a particular *order* of events. Elements of L_w are obtained from w by inserting time durations between the events.

Each choice y of values for the duration random variables determines a *unique* behavior of the system that we denote $\xi(y)$ and the probability of a set of behaviors is the probability of the y values that induce them. The density of this distribution at a complete timed word $\xi = t_1\sigma_1, \ldots, t_{nk}\sigma_{nk}$ under a liberal scheduler is defined as follows. For every step $(i, j) \in N \times K$, let r_i^j be the sum of all duration occurring between s_i^j and e_i^j. Then the density at ξ is:

$$\prod_{i \in N, j \in K} \phi_i^j(r_i^j). \tag{1}$$

Unfortunately (1) cannot be exported as is to the case of non-trivial schedulers where we have to resort to *incremental* computations that derive the probability of $\xi \cdot t \cdot \sigma$ from the probability of its *prefix* ξ. To this end we need to consider *incomplete* behaviors that correspond to a word w in which *not every* s has been followed by a matching e. The probability of $\xi \cdot t \cdot e$ for each of the pending *end* events depends on the probability of the corresponding step to terminate within a duration equal to the sum of t and the duration in ξ occurring after s *and* the probability of the *other* steps already started in w to terminate *after* that.

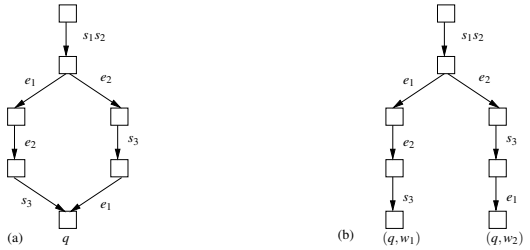

Figure 5: (a) Two commuting paths; (b) splitting a state into two copies according to the history.

An incomplete behavior ξ can be associated with two other objects, the first being the subset of L consisting of complete words having ξ as a *prefix* and the second is a global configuration of the automaton reached while generating ξ. A global state in a timed automaton is a mixture of active and idle local states with active clocks defined naturally according to the state, and this determines the dimensionality of clock space in that state. Thus a configuration is a pair (q,v) with $v \in \mathbb{T}^m$ for some $m \leq n$, and the time evolution inside the state consists of all active clocks advancing in the same pace, keeping the *difference* between any pair of active clocks constant throughout the sojourn in a state. The set of *time predecessors* of a clock valuation is $\pi(v) = \{v - \tau\mathbf{1} : \tau \geq 0\} \cap \mathbb{T}^m$, where $\mathbf{1}$ is a vector $(1,\ldots,1)$ of dimension m. A configuration (q,v) can be reached via time passage *only* from configurations of the form (q,v') with $v' \in \pi(v)$.

Let us just comment on the issue of commuting paths in the automaton. Why can we merge two such paths into a single state despite their differing past histories? The reason is that the past events that occurred in different orders along the two paths are of two types: 1) events related to *completed* steps that do not affect the future beyond what is already encoded in the state; 2) *start* transitions of steps which are still active in q. These events do affect the future but the order of their occurrence is captured already, at a finer level of detail, by the values of the active clocks and their pairwise *differences*. This is illustrated in the two commuting paths depicted in Fig. 5(a), assuming step 3 to follow step 2 in the same SDPA. The qualitative languages associated with the paths are the singletons $w_1 = s_1s_2e_1e_2s_3$ and $w_2 = s_1s_2e_2s_3e_1$, respectively, while the qualitative language of the whole state q is $s_1s_2(e_1||e_2s_3)$ and the only information that still affects the future is the time elapsed since s_3, captured by a clock (see also [25]). Despite this fact, for convenience reasons, we split states according to their respective histories, that is, work with *extended discrete states* of the form (q,h) where $h \in \Sigma^*$. A transition from q to q' labeled by some $w \in \Sigma^+$ thus extends into a transition from (q,h) to $(q',h \cdot w)$, and the transition graph of the automaton becomes a tree, see Fig. 5(b).

We will present our method to compute the probabilistic semantics gradually starting with its *support*, which is the set of all timed words which are possible if we interpret each ϕ as an interval, as in timed automata. Although what is described in the sequel is standard material underlying the *practice* of TA verification tools [27, 24], it is our perception that it is not sufficiently known to the more general public. We assume that for every component i active in state q, the duration of its corresponding step is distributed with a uniform density ϕ_i of support $[a_i,b_i]$. We use $R(v)$ to denote the setting to zero of clocks in R and the continuation of the clocks in v that are not in R. Note that by the definition of SPDA all clocks in R are inactive in q before the transition.

Definition 8 (Steps and Runs) *A step of a DPA $\mathscr{A}$ is one of the following:*

- *A* start step*: $(q,h,v) \xrightarrow{w} (q',h \cdot w,v')$, for some $(q,w,R,q') \in \Delta$ such that $v' = R(v)$;*

- *A* time step*:* $(q,h,v) \xrightarrow{\tau} (q,h,v+\tau\mathbf{1})$*; for some* $\tau > 0$ *such that for every i active in q,* $v_i + \tau \leq b_i$*;*
- *An* end step*:* $(q,h,v) \xrightarrow{e_i} (q,h\cdot e_i,v')$ *where* $v_i \in [a_i,b_i]$ *and* v' *is obtained from* v *by deactivating* x_i*.*

A run of the automaton is a sequence of steps which starts at $(q^1,\varepsilon,\perp)$ *and alternates between single time steps and one or more transition steps.*

The behavior associated with a run is the timed word obtained by concatenating the labels (transitions and durations) of its steps. We use the notation $(q,h,v) \xrightarrow{\xi} (q',h',v')$ to denote a run from (q,h,v) to (q',h',v') generating the timed word ξ (note that $h' = h\cdot\lambda(\xi)$). We also use the notation $(q,h,v) \xrightarrow{\xi} (\cdots)$ to denote an infinite run starting from (q,h,v). For acyclic DPA, all such runs terminate with an infinite time step inside the final state.

Definition 9 (State Languages) *With every extended configuration* (q,h,v) *we associate the following timed languages:*

- *The set of behaviors associated with runs whose last event is a* σ*-labeled transition to* $(q,h\cdot\sigma)$*:*

$$L^{\rightarrow\circ}(q,h,v) = \{\xi\cdot\sigma : (q^1,\varepsilon,\perp) \xrightarrow{\xi\cdot\sigma} (q,h\cdot\sigma,v)\}$$

- *The infinite behaviors generated by runs that start from* (q,h,v)*:*

$$L^{\circ\rightarrow}(q,h,v) = \{\xi : (q,h,v) \xrightarrow{\xi} (\cdots)\}$$

- *The set of all infinite behaviors of* $\mathscr{A}$ *with prefixes in* $L^{\rightarrow\circ}(q,h,v)$*:*

$$L(q,h,v) = L^{\rightarrow\circ}(q,h,v)\cdot L^{\circ\rightarrow}(q,h,v).$$

Observation 1 *If* $\xi \in L^{\rightarrow\circ}(q,h,v)$ *then* v_i *is equal, for every i active in q, to the time elapsed since the last* s_i *event in* ξ*.*

Definition 10 (Symbolic States) *An (extended) symbolic state is a triple* (q,h,Z) *with* $q \in Q$*,* $h \in \Sigma^*$ *and* Z *is a zone of dimensionality compatible with q.*

Intuitively, Z will be the set of all possible clock values that runs along the path to (q,h) may have. We will lift the definition of state languages to symbolic states by letting $L(q,h,Z) = \bigcup_{v\in Z} L(q,h,v)$. We associate with time passage and with every transition a successor operator over symbolic states.

Definition 11 (Successor Operator) *Successor operators admit three types:*

- *Time successors:* $post^t(q,h,Z) = (q,h,Z')$ *where*

$$Z' = \{v' : \exists v \in Z\, \exists \tau \in \mathbb{T}\ (q,h,v) \xrightarrow{\tau} (q,h,v')\}$$

- *Start successors:* $post^s(q,h,Z) = (q',h\cdot w,R(Z))$ *for every* start *transition* $(q,w,R,q') \in \Delta$*;*
- *End successors:* $post^e(q,h,Z) = (q',h\cdot e,Z')$ *for every transition* $(q,x=y,e,q') \in \Delta$ *where* Z' *is obtained from* Z *by eliminating the appropriate de-activated clock.*

The reachability graph, also known as the simulation graph, is what timed automata verification tools [27, 24] compute as a symbolic representation of the semantics of the automaton.

Definition 12 (Reachability Graph) *The reachability graph associated with a DPA $\mathscr{A}$ is a graph of symbolic states obtained by successive application of successor operators to* $(q^1, \varepsilon, \perp)$.

The fundamental property of the reachability graph is the following.

Theorem 1 *A symbolic state (q,h,Z) is part of the reachability graph iff for every $v \in Z$, the language $L^{\rightarrow\circ}(q,h,v)$ is not empty.*

In other words there is a timed word ξ generated by the automaton with $\mu(\xi) = h$ such that for every active component $\mathscr{A}_i$, the duration of the suffix of ξ starting with the last s_i event is v_i. Note that since all runs of a DPA have a continuation, $L^{\rightarrow\circ}(q,h,v) \neq \emptyset$ implies that $L^{\circ\rightarrow}(q,h,v) \neq \emptyset$ and $L(q,h,v) \neq \emptyset$. Moreover, $L(q,h,Z)$ is exactly L_h, the set of timed words in L whose untiming is h.

In the sequel we will extend the reachability graph with probabilities and work with symbolic states of the form (q,h,Z,ψ) where ψ is a partial density function over the clock values in Z, which can be used to compute the probability of $L(q,h,Z)$ or its subsets. To this end we need to extend the successor operators to become density transformers.

5 Density Transformers

The major issue in our computational approach is to determine, in a state where several processes are active, the probability of each of the pending *end* transitions to be taken and how the clock values are distributed when the transition is taken. As an informal illustration consider state q in the automaton of Fig. 3 admitting two competing active processes whose durations are distributed with densities ϕ_1 and ϕ_2, respectively. Assuming both ϕ_1 and ϕ_2 are uniform with a bounded interval support, their joint density $\phi(y_1,y_2) = \phi_1(y_1)\phi_2(y_2)$ is supported by a rectangle of the form $[a_1,b_1] \times [a_2,b_2]$. The clock values with which the state can be entered are restricted to the rectangle $[0,b_1] \times [0,b_2]$ and the two transitions can be taken in the rectangles $[a_1,b_1] \times [0,b_2]$ and $[0,b_1] \times [a_2,b_2]$, respectively, see Fig. 6(a). Note that the points of exit need not be inside the (joint) support of ϕ.

What is the probability $\rho_i(u|v)$ that transition i is taken at some point $u = (u_1,u_2)$, i.e., $u_i = y_i$, *given* that the state has been entered at some v? First of all, this probability is non-zero only if $v \in \pi(u)$, that is, v is a time-predecessor of u. Secondly, for transition 1 to be taken, it should be the case that process 1 chooses duration u_1 while process 2 chooses some $y_2 > u_2$ (the vertical thick line in Fig. 6(b)). Transition 2 will be taken at u when process 2 chooses a duration u_2 and process 1 some $y_1 > u_1$ (the horizontal thick line in the figure). Thus $\rho_1(u|v)$ is obtained by summing up the duration probabilities *above* u and $\rho_2(u|v)$ by summing up the probabilities *to the right* of u. Note that $\rho_i(u|v) = \rho(u|v')$ for any other $v' \in \pi(u)$ and that for points like u' outside the support of ϕ_1 we will have $\rho_1(u'|v) = 0$ and $\rho_2(u'|v) = 1$. Assuming that the state has been entered with some density ψ over clock values, we can sum up $\rho_i(u|v)$ over $v \in \pi(u)$ according to ψ and obtain the expected $\rho_i(u)$ as well as new densities ψ_i reflecting the distribution of the clock values upon taking each of the transitions.

With every extended state (q,h) in which m processes are active we associate a partial density function of the form $\psi(x_1,\ldots,x_m,y_1,\ldots,y_m)$ whose intended meaning is to capture the probability over clock values upon *entering* the state. Although the y variables are static and do not vary during execution, we need to keep them in the picture because they do not distribute evenly as time goes by. In other words, certain combinations of choices of durations will make some transitions impossible. We associate density transformers with every *start* and *end* transition as follows.

Start: Let q be a state with l active components and let s be a *start* transition which activates processes

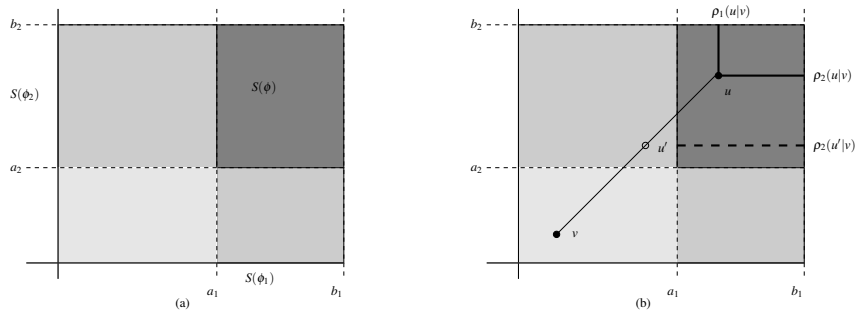

Figure 6: A race: (a) a state can be entered at any point in the shaded area; transition can be taken only in the darker area; (b) the probabilities ρ_1 and ρ_2.

$\{l+1,\ldots,m\}$.[7] We associate with s the density transformer $\mathscr{T}_s$ such that $\psi' = \mathscr{T}_s(\psi)$ if

$$\psi'(x_1,\ldots,x_l,0,\ldots,0,y_1,\ldots,y_l,y_{l+1},\ldots,y_m) = \psi(x_1,\ldots,x_l,y_1,\ldots,y_l) \cdot \phi(y_{l+1},\ldots,y_m)$$

with $\phi(y_{l+1},\ldots,y_m) = \phi_{l+1}(y_{l+1})\cdots\phi_m(y_m)$. When one of $\{x_{l+1},\ldots,x_m\}$ is non-zero, $\psi' = 0$. This operation just reflects the setting of the new clocks to zero and the introduction of their respective durations.
End: For every *end* transition e_i outgoing from a state q with m active processes we define two density transformers $\mathscr{T}_{r_i}$ and $\mathscr{T}_{\perp_i}$. As explained previously, the transformer $\mathscr{T}_{r_i}$ computes the clock density at the time when process i wins the race, given the density was ψ upon entering the state. It is defined as $\psi_i = \mathscr{T}_{r_i}(\psi)$ if

$$\psi_i(x_1,\ldots,x_m,y_1,\ldots,y_m) =$$

$$\begin{cases} \int_{\tau>0} \psi(x_1-\tau,\ldots,x_m-\tau,y_1,\ldots,y_m)d\tau & \text{if } x_i = y_i \wedge \forall i' \neq i\ x_{i'} < y_{i'} \\ 0 & \text{otherwise} \end{cases}$$

The transformer $\mathscr{T}_{\perp_i}$, which just deactivates clock x_i and projects it away from the clock space is defined as $\psi' = \mathscr{T}_{\perp_i}(\psi)$ if

$$\psi'(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_m,y_1,\ldots,y_{i-1},y_{i+1},\ldots,y_m) = \int_{u_i} \psi(x_1,\ldots,u_i,\ldots,x_m,y_1,\ldots,u_i,\ldots,y_m)du_i.$$

We can now define a probabilistic version of the successor operators. Note that for timed automata we had a unique time successor operator for each state, while for DPA time successor operators are specific for each of the transitions that participates in the race. A probabilistic symbolic state is a tuple (q,h,Z,ψ).

Definition 13 (Probabilistic Successor Operator) *Probabilistic successor operators admit two types:*

[7] The restriction to these indices is just to simplify notation. Recall also our previous remark that our probabilities are in reality hybrid, mixing discrete probabilities and distributions.

- *Start successors:* $post^s(q,h,Z,\psi) = (q',h\cdot w,R(Z),\psi')$ *for every* start *transition* $(q,w,R,q') \in \Delta$ *where* $\psi' = \mathscr{T}_s(\psi)$*;*
- *End successors:* $post^{e_i}(q,h,\psi,Z) = (q',h\cdot e_i,Z',\psi')$ *for every transition* $(q,x=y,e_i,q') \in \Delta$ *where* $\psi' = \mathscr{T}_{\perp_i}(\mathscr{T}_{r_i}(\psi))$ *and* Z' *is the support of* ψ'*.*

The *probabilistic reachability graph* is computed by starting with the initial probabilistic symbolic state $(q^1,\varepsilon,\perp,\psi_\perp)$ and then applying the appropriate successor operators. Computing this graph, as in the case of timed automata, allows us to compute everything of interest for DPA as we show below.

Recall that every y valuation induces a complete run $\xi(y)$ with an untiming h. Grouping all the y values resulting in the same h we have a mapping from the duration space $\mathbb{R}^{nk}$ to the finite set Σ^{nk} which defines the probability of each path. To extend this notion to incomplete behaviors one could define a sequence of functions $\{f_\alpha : \alpha = 0,\ldots,nk\}$ over the duration space, each mapping y into a prefix $\xi(y)_\alpha$ of $\xi(y)$ admitting *exactly* α discrete transitions. As mentioned earlier, to compute $f_{\alpha+1}$ from f_α it is sufficient to know the qualitative prefix h_α and the time elapsed since the non-terminated *start* events. For each α we have then a *hybrid* (discrete-continuous) probability distribution on $\Sigma^\alpha \times \mathbb{R}^m$ which can be expressed as a finite set of densities $\{\eta_h : h \in \Sigma^\alpha\}$. Our main claim is that if (q,h,Z,ψ) is part of the probabilistic reachability graph then

$$\eta_h = \int_y \psi(x,y)dy.$$

This holds trivially for the root $(q^1,\varepsilon,\perp,\psi_\perp)$ which corresponds to η_ε where all the probability is concentrated in the empty sequence. The inductive step, showing that if a node (q,h,Z,ψ) satisfies $\eta_h(x) = \int \psi(x,y)dy$ than any successor (q',h',Z',ψ') satisfies $\eta_{h'}(x) = \int \psi'(x,y)dy$, is immediate for a *start* successor because it just concatenates some s-labels without changing probabilities. For an *end* successor e_i, observe that for every $v \in Z$ and $v' \in Z'$, the corresponding run leads from (q,h,v) to $(q',h\cdot e_i,v')$, concatenating to the language a timed word $\tau\cdot e_i$ with $\tau = v'_i - v_i$ and the probability of v', the time elapsed since the remaining uncompleted *start* events, is captured by ψ'.

Thus we can compute the probability for each interesting set of paths, for example those in which some event precedes another. Moreover, by adding an auxiliary clock which is never reset and measures absolute time, we can retrieve the evolution of these probabilities over time and compute the distribution and expected value of the total termination times. This provides for an effective comparison between the performance of different scheduling policies.

6 Past and Future Work

We have shown how timed automata verification techniques can be extended to handle durations which are distributed probabilistically. We conclude by mentioning some related work as well as some of the many open issues that remain.

The works closest to ours are those of Alur and Bernadsky [2, 9] and Vicario et al. [13, 14, 26], each using a different models. The work of [2, 9] is concerned with verifying temporal properties for some classes of GSMPs, where the hard part is the treatment of the unbounded *until* operator which is achieved by putting restrictions on the number of concurrently active clocks. They also deal with computational issues related to symbolic computation of integrals over exponential-polynomial distributions. The work of [13, 14, 26] is concerned with certain classes of stochastic Petri nets for which they develop a computational framework similar to ours which includes both exact and approximate computation of

the distributions. The major difference is that our formulation that separates the x and y variables, provides for more sophisticated scheduling policies, such as those described in [1], that take clock values into consideration.

The most urgent topics in our agenda are the implementation and the extension to cyclic DPA. From a computational standpoint, since we start with uniform distribution, all our density transformers result in piecewise-polynomial functions that can be computed analytically using a mixture of zone-based algorithms and computer algebra tools. Of course, the obtained expressions will become increasingly complex due to case splitting and may require approximation. An alternative (but not scalable) way would be to work using discrete-time approximations of the duration distributions. The present results allow us to compute reachable symbolic states forward to any desired horizon, but since densities are much richer than zones, there is no immediate proof of convergence to a fixed point. Given that the density transformer can be phrased as a linear operator over state-related densities, we intend to investigate functional analysis techniques like those used in [6] to establish convergence and approximate termination.

Acknowledgment: This work benefitted from discussions with E. Asarin and from numerous anonymous referees.

References

[1] Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[2] R. Alur and M. Bernadsky. Bounded model checking for GSMP models of stochastic real-time systems. In *HSCC*, pages 19–33, 2006.

[3] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for probabilistic real-time systems (extended abstract). In *ICALP*, pages 115–126, 1991.

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

[5] E. Asarin, P. Caspi, and O. Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.

[6] E. Asarin and A. Degorre. Volume and entropy of regular timed languages: Analytic approach. In *FORMATS*, pages 13–27, 2009.

[7] C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[8] F. Bause and P.S. Kritzinger. *Stochastic Petri Nets*. Vieweg, 2002.

[9] M. Bernadsky and R. Alur. Symbolic analysis for GSMP models with one stateful clock. In *HSCC*, pages 90–103, 2007.

[10] H.C. Bohnenkamp, P.R. D'Argenio, H. Hermanns, and J.-P. Katoen. Modest: A compositional modeling formalism for hard and softly timed systems. *IEEE Trans. Software Eng.*, 32(10):812–830, 2006.

[11] P. Bouyer. *From Qualitative to Quantitative Analysis of Timed Systems*. Mémoire d'habilitation, Université Paris 7, Paris, France, January 2009.

[12] E. Brinksma, H. Hermanns, and J.-P. Katoen, editors. *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *LNCS*. Springer, 2001.

[13] G. Bucci, R. Piovosi, L. Sassoli, and E. Vicario. Introducing probability within state class analysis of dense-time-dependent systems. In *QEST*, pages 13–22, 2005.

[14] L. Carnevali, L. Grassi, and E. Vicario. State-density functions over DBM domains in the analysis of non-Markovian models. *IEEE Trans. Software Eng.*, 35(2):178–194, 2009.

[15] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2nd edition, 2008.

[16] P.R. D'Argenio and J.-P. Katoen. A theory of stochastic systems part i: Stochastic automata. *Inf. Comput.*, 203(1):1–38, 2005.

[17] R. German. Non-markovian analysis. In Brinksma et al. [12], pages 156–182.

[18] P.W. Glynn. A GSMP formalism for discrete event systems. *Proceedings of the IEEE*, 77(1):14–23, 1989.

[19] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[20] H.E. Jensen. Model checking probabilistic real time systems. In *7th Nordic Workshop on Programming Theory*, pages 247–261, 1996.

[21] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte. *Modelling with generalized stochastic Petri nets*. John Wiley & Sons, Inc. New York, NY, USA, 1994.

[22] M.Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *CONCUR*, pages 123–137, 2000.

[23] M.Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theor. Comput. Sci.*, 282(1):101–150, 2002.

[24] K.G Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[25] R. Ben Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *CONCUR*, pages 465–476, 2006.

[26] E. Vicario, L. Sassoli, and L. Carnevali. Using stochastic state classes in quantitative evaluation of dense-time reactive systems. *IEEE Trans. Software Eng.*, 35(5):703–719, 2009.

[27] S. Yovine. Kronos: A verification tool for real-time systems. *STTT*, 1(1-2):123–133, 1997.