L. Fribourg

A Closed–Form Evaluation for Extended Timed Automata

Research Report LSV–98–2, Mar. 1998

Laboratoire Spécification et Vérification

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

http://www.lsv.ens–cachan.fr/Publis/
Research Report LSV–98–2, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, Mar. 1998

A Closed-Form Evaluation for Extended Timed Automata

Laurent Fribourg

Ecole Normale Supérieure Cachan/CNRS - 61 av. Pdt. Wilson - 94235 Cachan - France
email: fribourg@lsv.ens-cachan.fr

Abstract. We explain here how the bottom-up evaluation method of Revesz for recursive programs with arithmetic constraints, can be adapted from the domain of integers to the domain of reals. We apply the procedure to characterize the set of ancestors of a given state H for timed automata. (An ancestor of H is a state linked to H by a sequence of transitions.) Such a procedure also applies to extensions of timed automata where, in particular, real variables are used not only as clocks but as data registers. The procedure has been successfully experimented for proving automatically the correctness of a sophisticated reactive program that controls dataflow rates on ATM networks.

1 Introduction

Revesz's procedure is a bottom-up evaluation method that allows to compute the least-fixed point of recursive programs with arithmetic constraints over integers. We explain here how to adapt Revesz's procedure from integers to reals, and how to use it, in the framework of timed automata, for characterizing finitely the set of *ancestors* of a given state H. (An ancestor of H is a state linked to H by a sequence of transitions.) This provides us with an alternative method to the calculus of region graphs [2]. A major interest of the method lies in the fact that it can be applied to extended models of timed automata where data registers are used, in couple with clocks, for keeping track of data transmitted during transitions. The procedure has been successfully experimented for proving the correctness of a reactive program that controls dataflow rates on ATM networks.

The plan of the paper is the following: section 2 presents Revesz's bottom-up evaluation procedure; section 3 presents the notion of timed automata; section 4 describes an encoding of the state successor relation as a program with arithmetical constraints; section 5 explains how to finitely compute the set of ancestor states for timed automata using Revesz's procedure; section 6 explains how to apply the method to extended models of timed automata; a conclusion is given in section 7.

2 Bottom-Up Evaluation of Gap Programs

The following definitions are mainly borrowed from [10] (see [10,11] for formal details). A *gap-atom* is an expression of the form $x_i + a < x_j$, $x_i + a \leq x_j$ or $x_i = x_j$, where x_i, x_j denote real variables and a a nonnegative integer. The expression $x_i + a < x_j$ will be sometimes abbreviated as $x_i <^a x_j$. Likewise the expression $x_i + a \leq x_j$ will be sometimes abbreviated as $x_i \leq^a x_j$. The constant a will be called the gap assigned to the ordering relation between x_i and x_j ($<^0$ means $<$, and $\leq^0$ means $\leq$). A gap-atom $x_i <^a x_j$ (resp. $x_i \leq^a x_j$) can be graphically represented by two vertices labelled x_i and x_j linked by an oriented edge from x_j to x_i labelled with symbol $<^a$ (resp. $\leq^a$). Likewise a gap-atom $x_i = x_j$ is graphically represented as an unoriented edge linking x_i and x_j.

A *gap-configuration* G is a conjunction of gap-atoms. A gap-configuration over $x_1, ..., x_n$ is graphically represented as a graph whose vertices are labelled $x_1, ..., x_n$, and whose edges correspond to the gap-atoms of G.

The *merging* of a gap-configurations G over a vector of variables x and a gap-configuration G' over a vector of variables x' (non necessarily distinct from x) is the gap-configuration over $x \cup x'$ whose graph is obtained by connecting together the graphs of G and G' together (see [11], p.131, for technical details). The resulting gap-configuration H is equivalent to $G \wedge G'$. Merging includes the following test of unsatisfiability: if, in the graph associated with H, there is a cyclic circuit with an edge labelled $<^a$, then merging answers *failure*. For example the merging of $x_1 < x_2$ together with $x_2 < x_1$ fails.

As usual, an expression of the form $x_i \preceq^a x_j \preceq^b x_k$ with $\preceq^a \in \{=, <^a, \leq^a\}$, $\preceq^b \in \{=, <^b, \leq^b\}$ means $x_i \preceq^a x_j \wedge x_j \preceq^b x_k$. In this paper, a gap-configuration G over $\{x_1, ..., x_n\}$ will always contain implicitly the information $0 \leq x_i$ for $i = 1, ..., n$. Additionally, it may contain explicitly the information $0 = x_i$ for some i (case where x_i is bound to 0).

We define the notion of "shortcut" (or variable elimination) of $x_i \preceq^a x_j \preceq^b x_k$ over x_j as follows. The *shortcut* [1] of $x_i <^a x_j <^b x_k$ over x_j is $x_i <^{a+b} x_k$. Likewise, the *shortcut of* $x_i \leq^a x_j \leq^b x_k$ is $x_i \leq^{a+b} x_k$. The *shortcut of* $x_i \leq^a x_j <^b x_k$ or $x_i <^a x_j \leq^b x_k$ is $x_i <^{a+b} x_k$. Shortcuts are finally defined in the obvious manner in the case where $\preceq^a$ or $\preceq^b$ denote '='. More generally, shortcuts are defined on graphs representing gap-configurations (see [11], p.130).

The *projection* of a gap-configuration G onto a subset x' of x is the result of applying iteratively shortcuts over the variables of $x - x'$. (The result is independent of the order chosen for eliminating the variables of $x - x'$ via shortcuts.) For example the projection of $x_1 <^4 x_2 \leq^5 x_3 = x_4$ onto $\{x_1, x_4\}$ is $x_1 <^9 x_4$.

A *logic program with gap constraints* (or, more simply, a *gap program*) is defined by a base clause of the form: $p(x_1, ..., x_n)$:-$\phi(x_1, ..., x_n)$ where ϕ is a gap-configuration over $x_1, ..., x_n$ (called *initial gap-configuration*), and a set of recursive clauses r:

$p(x'_1, ..., x'_n)$:-$\chi(x_1, ..., x_n, x'_1, ..., x'_n), p(x_1, ..., x_n)$.

where χ denotes a gap-configuration over $x_1, ..., x_n, x'_1, ..., x'_n$.

Let G be a gap-configuration over $x = \{x_1, ..., x_n\}$. The application of a recursive clause r to G yields a gap-configuration H over $x' = \{x'_1, ..., x'_n\}$ obtained as follows:

1. *Merge* G and χ. If merge fails, then application of r to G fails. Otherwise, let G' be the resulting gap-configuration.
2. *Shortcut* variables $x_1, ..., x_n$ in G'. The resulting gap-configuration is H.

For example, suppose that r is $p(x'_1, x'_2)$:-$\chi(x_1, x_2, x'_1, x'_2), p(x_1, x_2)$, where χ is $x'_1 <^2 x_1 <^5 x_2 <^3 x'_2$, and let G be $x_1 <^8 x_2$. Merging G and χ yields
$x'_1 <^2 x_1 <^8 x_2 <^3 x'_2$. Then shortcutting variables x_1, x_2 yields: $x'_1 <^{13} x'_2$.

---

[1] Note that this differs from the situation on integers where the shortcut is $x_i <^{a+b+1} x_k$.

Now we can define a "bottom-up" evaluation procedure for logic programs with gap-configurations:

initially $\mathcal{D} := \{\phi(x_1, ..., x_n)\}$
while can add a new gap-configuration to $\mathcal{D}$ **do**
begin
select any recursive clause r of the program,
select any gap-configuration $G(x_1, ..., x_n)$ of $\mathcal{D}$,
apply r to G, thus producing $H(x'_1, ..., x'_n)$,
rename variables $x'_1, ..., x'_n$ of H as $x_1, ..., x_n$, and add H to $\mathcal{D}$ if $H \notin \mathcal{D}$
end

In this paper the set of generated gap-configurations will be forced to belong to a restricted class Γ, called the class of "closed-forms": gap-configurations generated by Revesz's bottom-up procedure which are not in Γ will be rejected by a specific additional "closure" test. Furthermore, this class Γ will represent a *finite* set of gap-configurations. This means that Revesz's bottom-up evaluation procedure coupled with this closure test, is guaranteed to terminate. [2] The output $\mathcal{D}$ of the procedure is a set of gap-configurations that characterizes the *least fixed-point* associated with the gap program defining p. We will use a bottom-up evaluation procedure in order to show that the set of tuples x defined by p is contained into a given set Π of gap-configurations characteristic of a certain property. There are two manners of showing the inclusion into Π (see, e.g., [5]). The *forward* manner consists in computing $\mathcal{D}$ and showing that $\mathcal{D}$ is included into Π. The *backward* manner consists in computing the least fixed-point, say $\mathcal{D}^{-1}$, of the "inverse" program of p. This inverse program is obtained by exchanging the head and recursive call of each recursive clause for p, and taking $\neg\Pi$ as a starting gap-configuration instead of ϕ. We say that $\mathcal{D}^{-1}$ is the set of *ancestors* of $\neg\Pi$. In such a case, the property is proved by showing that the intersection of $\mathcal{D}^{-1}$ and $\neg\phi$ is empty. In the following we will focus on the backward manner.

In the following we will consider recursive clauses of the form
$p(s', x'_1, ..., x'_n)$:-$\zeta(x_1, ..., x_n, x'_1, ..., x'_n), p(s, x_1, ..., x_n)$.
where ζ denotes a gap-configuration over variables $x_1, ..., x_n, x'_1, ..., x'_n$, and s (resp. s') denotes an additional "location" argument s included into p: this argument takes its value onto a finite special set S of constants. In this context, a *gap-configuration (with location)* is a couple made of a location s and a gap-configuration G. We will sometimes however omit to mention explicitly location s. Context will make it clear.

3 Timed Automata

This presentation is inspired from [3]. A timed automaton A is a tuple $\langle S, x, \tau, s_{init}, L, T, \Theta\rangle$ where:

1. S is a finite set of *locations*,
2. $x \equiv (x_1, ..., x_n)$ is a n-tuple of real-values variables, called *clocks*, and τ a real-value variable, called *universal clock*.
3. $s_{init} \in S$ is an initial location,
4. L is a finite set of *labels*.

[2] In [10,11], bottom-up evaluation also terminates because it is coupled with a "subsumption" test, which is useless here.

5. T is a finite set of *transitions*: a transition, labelled l from s to s', is a tuple of the form $\langle s, l, \psi, I, s' \rangle$ where $s, s' \in S$, $l \in L$, ψ is the *guard*, and $I \subseteq \{1, ..., n\}$ is the subset of indices of clocks to be reset to 0.

6. Θ associates each location $s \in S$ with an *invariant* condition θ_s.

The guard $\psi(x)$ of a transition is a conjunction of inequalities of the form:

$$\bigwedge_{i=1,\dots,n} a_i^{\psi} \ll_i' x_i \ll_i b_i^{\psi}$$

where $\ll_i', \ll_i \in \{<, \leq\}$ and $a_i^{\psi} \in \mathbb{N} \cup \{-\infty\}$, $b_i^{\psi} \in \mathbb{N} \cup \{+\infty\}$, for all $i \in \{1, ..., n\}$. The above formula will be often abbreviated as: $a^{\psi} \ll x \ll b^{\psi}$. The invariant condition $\theta_s(x)$ is a conjunction of inequalities of the form:

$$\bigwedge_{i=1,\dots,n} x_i \ll_i c_i^s$$

where $\ll_i \in \{<, \leq\}$ and $c_i^s \in \mathbb{N} \cup \{+\infty\}$, for all $i \in \{1, ..., n\}$. The above formula will be often abbreviated as: $x \ll c^s$.

A *state* of A is a triple (s, x, τ) where $s \in S$, $x \in (\mathbb{R}^+)^n$, $\tau \in \mathbb{R}^+$. The automaton A starts with the control at the location s_{init}, and all its clocks initialized to 0 (including τ). The state is thus $(s_{init}, 0, 0)$. Then the values of all the clocks increase uniformly with time, and the states become of the form (s_{init}, t, t). We define Q_{init} to be the subset of those states which satisfy the invariant associated with s_{init}.

$$Q_{init} = \{(s_{init}, t, t) \mid t \in \mathbb{R}^+ \wedge \theta_{s_{init}}(t)\ \}$$

Note that $\theta_{s_{init}}(t)$ implies $\theta_{s_{init}}(t')$, for all $t' \leq t$. We say that state $(s', x' + t, \tau + t)$ is a *successor* of state (s, x, τ) iff there exists a transition $\langle s, l, \psi, I, s' \rangle$ such that:

1. $\psi(x)$, i.e.: $a^{\psi} \ll x \ll b^{\psi}$,
2. $x_i' = 0$ if $i \in I$, otherwise $x_i' = x_i$,
3. $\theta_{s'}(x' + t)$, i.e.: $x' + t \ll c^{s'}$.

Here again, note that $\theta_{s'}(x' + t)$ implies $\theta_{s'}(x' + t')$, for all $t' \leq t$. In the following we will use a special set, called *upperbound set*, and denoted J_A (or more simply J). This set is made of the couples (i, d) where $i \in \{1, ..., n\}, d \in \mathbb{N}$ such that d appears as a finite upperbound value of variable x_i, either within a guard ψ of some transition (i.e., $d = b_i^{\psi}$ for some ψ), or within an invariant θ of some location s (i.e., $d = c_i^s$ for some s). We also denote by K_A (or more simply by K) the set of values taken by the second component d of couples of J. [3]

Example 1

We consider an example corresponding to the train-controller example of [7], p.404. The timed automaton has 6 locations: $\{00, 11, 12, 22, 03, 21\}$. The initial location is 00. There are three clocks x_1, x_2, x_3. [4] The invariants are:

$\theta_{00} \equiv true$, $\quad\theta_{11} \equiv x_1 \leq 5 \wedge x_2 < 2$, $\quad\theta_{12} \equiv x_1 \leq 5$,
$\theta_{21} \equiv x_1 \leq 5 \wedge x_2 < 2$, $\quad\theta_{22} \equiv x_1 \leq 5$, $\quad\theta_{03} \equiv x_2 \leq 1$.

There are 6 transitions labeled *app* from 00 to 11, *down* from 11 to 12, *in* from 12 to 22, *out* from 22 to 03, *up* from 03 to 00, and *in'* from 11 to 21 (see figure 1).

[3] K is the set of upperbound constants appearing in guards or invariants of the system.
[4] With respect to [7], an additional clock x_3 is introduced and used as an "observer".

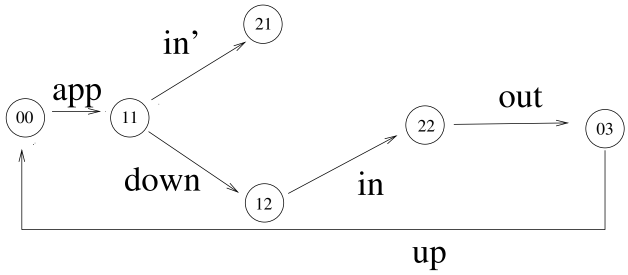

Fig. 1. Train-controller automaton

The associated guards ψ and sets I of indices of clocks to be reset are respectively:

$$\begin{array}{ll}
\psi_{app} \equiv true, \quad I_{app} = \{1,2,3\} & \psi_{down} \equiv 1 \leq x_2 < 2, \quad I_{down} = \{3\} \\
\psi_{in} \equiv 2 \leq x_1 < 5, \quad I_{in} = \{\} & \psi_{in'} \equiv 2 \leq x_1 < 5, \quad I_{in'} = \{\} \\
\psi_{out} \equiv x_1 \leq 5, \quad I_{out} = \{2\} & \psi_{up} \equiv x_2 \leq 1, \quad I_{up} = \{\}
\end{array}$$

The set of upperbound values for x_1 (resp. x_2) appearing within invariants θ and guards ψ is $\{5\}$ (resp. $\{1,2\}$). There is no upperbound values for x_3. So J is a set of the form $\{(1,5),(2,1),(2,2)\}$, and K is $\{5,1,2\}$.

4 Defining The Successor Relation via a CLP Program

As shown, e.g. in [12], one can define recursively the *reachability set*, i.e. the set of states deducible from the initial state by applying iteratively the successor relation, via a recursive logic program with arithmetical constraints (in short: a CLP program). Before showing this, we perform preliminarily a change of clock variables that will ease subsequent program transformations. Henceforth, instead of clock variable x_i $(1 \leq i \leq n)$, we will use y_i defined to be equal to $\tau - x_i$. The vector $(y_1, ..., y_n)$ will be denoted y. Each component y_i $(1 \leq i \leq n)$ of y is initially set to 0. (This will be enforced through a gap-configuration of the form $0 = y_1 = y_2 = ... = y_n$, abbreviated as $y = 0$.) Now, when time elapses, only τ increases while all the variables y_i keep unchanged. When a transition is performed, variables y_i (for $i \in I$) are reset not to 0, but to the current value of τ. A state is now a triple (s, y, τ) with $s \in S, y \in (\mathbb{R}^+)^n$, and $\tau \in \mathbb{R}^+$. The set of initial states is:

$$Q_{init} = \{(s_{init}, y, t) \mid t \in \mathbb{R}^+ \wedge \theta_{s_{init}}(t) \ \wedge \ y = 0 \ \}$$

Accordingly, a state (s', y', τ') is a successor of a state (s, y, τ) iff there exists $t \in \mathbb{R}^+$ such that $\tau' = \tau + t$, and a transition $\langle s, l, \psi, I, s' \rangle$ such that:

1. $\psi(\tau - y)$, i.e.: $y + a^{\psi} \ll \tau \ll y + b^{\psi}$,
2. $y'_i = \tau$ if $i \in I$, otherwise $y'_i = y_i$,
3. $\theta_{s'}(\tau' - y')$, i.e.: $\tau' \ll y' + c^{s'}$.

We introduce the notation $\rho_I(y, \tau, y')$ in order to express item 2. Note that the existence of $t \in \mathbb{R}^+$ such that $\tau' = \tau + t$ is equivalent to the relation $\tau \leq \tau'$. It follows that a state (s', y', τ') is a *successor* of a state (s, y, τ), written $(s, y, \tau) \rightarrow_A (s', y', \tau')$, iff:

$$\tau \leq \tau' \ \wedge \ \rho_I(y, \tau, y') \ \wedge \ y + a^{\psi} \ll \tau \ll y + b^{\psi} \ \wedge \ \tau' \ll y' + c^{s'},$$

for some transition $\langle s, \psi, l, I, s' \rangle$.

Let $\rightarrow_A^*$ denote the reflexive-transitive closure of $\rightarrow_A$. The *reachability set* associated with A, denoted $reach(A)$, is:

$$reach(A) = \{(s', y', \tau') \mid \exists (s, y, \tau) \in Q_{init} : \quad (s, y, \tau) \rightarrow_A^* (s', y', \tau')\}$$

We have: $(s, y, \tau) \in reach(A)$ iff $p_A(s, y, \tau)$, where p_A is defined by the following CLP clauses:

$$p_A(s_{init}, y, \tau) \text{ :- } y = 0, \quad \tau \ll c^{s_{init}}.$$

$$\{ \ p_A(s', y', \tau') \text{ :- } \quad \tau \leq \tau', \quad \rho_I(y, \tau, y'), \quad y + a^\psi \ll \tau \ll y + b^\psi, \quad \tau' \ll y' + c^{s'}, \quad p_A(s, y, \tau). \quad \}$$

for all transition $\langle s, l, \psi, I, s' \rangle$ of T.

Example 2
The encoding of the automaton of example 1 as a CLP programs gives [5] :
%init
$p_A(00, y_1, y_2, y_3, \tau)$:- $\quad y_1 = y_2 = y_3 = 0.$
%app
$p_A(11, y_1', y_2', y_3', \tau')$:- $\tau \leq \tau', \quad y_1' = \tau, y_2' = \tau, y_3' = \tau, \quad \tau' \leq y_1' + 5, \tau' < y_2' + 2,$
$p_A(00, y_1, y_2, y_3, \tau).$
%down
$p_A(12, y_1', y_2', y_3', \tau')$:- $\tau \leq \tau', \quad y_1' = y_1, y_2' = y_2, y_3' = \tau, \quad y_2 + 1 \leq \tau < y_2 + 2, \tau' \leq y_1' + 5,$
$p_A(11, y_1, y_2, y_3, \tau).$
%in
$p_A(22, y_1', y_2', y_3', \tau')$:- $\tau \leq \tau', \quad y_1' = y_1, y_2' = y_2, y_3' = y_3, \quad y_1 + 2 \leq \tau < y_1 + 5, \tau' \leq y_1' + 5,$
$p_A(12, y_1, y_2, y_3, \tau).$

5 A Closed Gap-Form Evaluation

We would like now to transform the above CLP program into a gap program in order to apply Revesz's bottom-up evaluation procedure, and characterize the set of ancestors as a *finite* set of gap-configurations. In a gap program, the constraints of the form $\tau \ll y + b^\psi$ and $\tau' \ll y' + c^{s'}$ within the body of the CLP clauses, are not allowed. In order to treat this problem, we are going to introduce new variables of the form z_i^d, $z_i'^d$ and τ^d which will represent terms $y_i + d$, $y_i' + d$ and $\tau + d$ respectively. Constraints $\tau \ll y + b^\psi$ and $\tau' \ll y' + c^{s'}$ will be then replaced by constraints of the form $\tau \ll z^{b^\psi}$ and $\tau' \ll z'^{c^{s'}}$. Besides, using the fact that $\rho_I(y, \tau, y')$ stands for $\bigwedge_{i \notin I} y_i' = y_i \wedge \bigwedge_{i \in I} y_i' = \tau$, we will set $z_i'^d$ to be equal to z_i^d for all $i \notin I$, and $z_i'^d$ to be equal to τ^d for all $i \in I$. This is formalized below.

5.1 Transformation of the CLP program into a(n almost) gap program

Recall that : $\tau \ll y + b^\psi \equiv \bigwedge_{i \in \{1,\ldots,n\}} \tau \ll_i y_i + b_i^\psi$, and $\tau' \ll y' + c^{s'} \equiv \bigwedge_{i \in \{1,\ldots,n\}} \tau' \ll_i y_i' + c_i^{s'}$. More precisely, one has: $\tau \ll y + b^\psi \equiv \bigwedge_{i \in dom(\psi)} \tau \ll_i y_i + b_i^\psi$, and $\tau' \ll y' + c^{s'} \equiv \bigwedge_{i \in dom(\theta_{s'})} \tau' \ll_i y_i' + c_i^{s'}$, where $dom(\psi)$ (resp. $dom(\theta_{s'})$) denotes the subset of elements $i \in \{1, ..., n\}$ such that $b_i^\psi \neq +\infty$ (resp. $c_i^{s'} \neq +\infty$). Consider the upperbound set J: it is the set of couples (i, d) (with

[5] for the sake of conciseness, only the initial gap-configuration and transitions *app*, *down* and *in* are given here.

$i \in \{1,...,n\}$,$d \in \mathbb{N}$) such that $\tau \ll_i y_i + d$ or $\tau' \ll_i y'_i + d$ occurs within some recursive clause of the CLP program. We introduce a new vector, denoted z^J, made of variables of the form z_i^d, for all couple (i,d) of J. We now extend predicate p_A with z^J as an additional tuple of arguments, and transform the above CLP program as follows:

$$
\begin{array}{l}
p_A(s_{init}, y, \tau, z^J) \text{ :- } \; y = 0, \; \bigwedge_{i \in dom(\theta_{s_{init}})} \tau \ll_i z_i^{c_i^{s_{init}}}, \; \bigwedge_{(i,d) \in J} z_i^d = y_i + d. \\
\{ \; p_A(s', y', \tau', z'^J) \text{ :- } \tau \leq \tau', \rho_I(y, \tau, y'), \; y + a^\psi \ll \tau, \quad \bigwedge_{i \in dom(\psi)} \tau \ll_i z_i^{b_i^\psi}, \\
\hfill \bigwedge_{i \in dom(\theta_{s'})} \tau' \ll_i z_i'^{c_i^{s'}}, \\
\hfill \bigwedge_{(i,d) \in J \wedge i \notin I} z_i'^d = z_i^d, \; \bigwedge_{(i,d) \in J \wedge i \in I} z_i'^d = \tau^d, \\
\hfill \bigwedge_{d \in K} \tau^d = \tau + d, \\
\hfill p_A(s, y, \tau, z^J). \quad \}
\end{array}
$$

for all transition $\langle s, l, \psi, I, s' \rangle$.

It is easy to prove that equations $z_i^d = y_i + d$ (for all $(i,d) \in J$) are invariants of this program. It should be clear that this CLP program is equivalent to the former one. In the new CLP program, the only constraints that are not allowed within gap programs, are $\bigwedge_{d \in K} \; \tau^d = \tau + d$ in the recursive clauses. We call these constraints the *upperbound τ-requirements* of the CLP program. When applying a clause, the elimination of these constraints will require a specific treatment (see next subsection). It is convenient also to assume that the conjunction $\bigwedge_{i=1,\ldots,n} y_i \leq \tau$ is included in the body of each recursive clause.[6] This will be implicit in the following.

Example 3

Recall that in the transition system of example 1, J is: $\{(1,5),(2,1),(2,2)\}$, and K is $\{1,2,5\}$. Therefore, w.r.t. the CLP program of example 2, there will be 6 additional arguments z_1^5, z_2^1, z_2^2, and τ^1, τ^2, τ^5. The program becomes:

%init

$$
\begin{array}{l}
p_A(00, y_1, y_2, y_3, \tau, z_1^5, z_2^1, z_2^2) \text{ :- } \; y_1 = y_2 = y_3 = 0, \\
\hfill z_1^5 = y_1 + 5, z_2^1 = y_2 + 1, z_2^2 = y_2 + 2.
\end{array}
$$

%app

$$
\begin{array}{l}
p_A(11, y'_1, y'_2, y'_3, \tau', z_1'^5, z_2'^1, z_2'^2) \text{ :- } \tau \leq \tau', \; y'_1 = \tau, y'_2 = \tau, y'_3 = \tau, \\
\hfill \tau' \leq z_1'^5, \tau' < z_2'^2, \\
\hfill z_1'^5 = \tau^5, z_2'^1 = \tau^1, z_2'^2 = \tau^2, \\
\hfill \tau^1 = \tau + 1, \tau^2 = \tau + 2, \tau^5 = \tau + 5, \\
\hfill p_A(00, y_1, y_2, y_3, \tau, z_1^5, z_2^1, z_2^2).
\end{array}
$$

%down

$$
\begin{array}{l}
p_A(12, y'_1, y'_2, y'_3, \tau', z_1'^5, z_2'^1, z_2'^2) \text{ :- } \tau \leq \tau', \; y'_1 = y_1, y'_2 = y_2, y'_3 = \tau, \\
\hfill y_2 + 1 \leq \tau < z_2^2, \tau' \leq z_1'^5, \\
\hfill z_1'^5 = z_1^5, z_2'^1 = z_2^1, z_2'^2 = z_2^2, \\
\hfill \tau^1 = \tau + 1, \tau^2 = \tau + 2, \tau^5 = \tau + 5, \\
\hfill p_A(11, y_1, y_2, y_3, \tau, z_1^5, z_2^1, z_2^2).
\end{array}
$$

%in

$$
\begin{array}{l}
p_A(22, y'_1, y'_2, y'_3, \tau', z_1'^5, z_2'^1, z_2'^2) \text{ :- } \tau \leq \tau', \; y'_1 = y_1, y'_2 = y_2, y'_3 = y_3, \\
\hfill y_1 + 2 \leq \tau < z_1^5, \tau' \leq z_1'^5, \\
\hfill z_1'^5 = z_1^5, z_2'^1 = z_2^1, z_2'^2 = z_2^2, \\
\hfill \tau^1 = \tau + 1, \tau^2 = \tau + 2, \tau^5 = \tau + 5, \\
\hfill p_A(12, y_1, y_2, y_3, \tau, z_1^5, z_2^1, z_2^2).
\end{array}
$$

[6] Recall that y_i represents $\tau - x_i$, and that we have $0 \leq x_i \leq \tau$ since x_i evolves like τ, except it may be reset to 0.

5.2 A closed-form for ancestor states

We are going now to apply the above program in a backward manner in order to compute the set of ancestor states for a given input gap-configuration. The recursive clauses are obtained from those above, by exchanging the head $p(s', y', \tau', z'^J)$ and the recursive call $p(s, y, \tau, z^J)$. They are thus of the form:

$$\begin{aligned}\{\ p_A(s, y, \tau, z^J) \text{ :- } & \tau \leq \tau', \rho_I(y, \tau, y'),\ \ y + a^\psi \ll \tau,\ \ \bigwedge \tau \ll z^{b^\psi},\ \ \bigwedge \tau' \ll z^{c^{s'}}, \\ & \textstyle\bigwedge_{(i,d) \in J \wedge i \notin I}\ z_i'^d = z_i^d,\quad \bigwedge_{(i,d) \in J \wedge i \in I}\ z_i'^d = \tau^d,\quad \bigwedge_{d \in K} \tau^d = \tau + d, \\ & p_A(s', y', \tau', z'^J).\quad \}\end{aligned}$$

for all transition $\langle s, l, \psi, I, s' \rangle$.

We will apply these clauses iteratively, starting from an the gap-configuration $H_{\neg\Pi}$ which corresponds to the negation of the property Π to be proved. We now assume that $H_{\neg\Pi}$ belongs to a class $\Gamma(y', \tau', z'^J)$ of gap-configurations made only of gap-atoms of the following form:

$y_i' + e_{i,j} \preceq y_j'$ where $e_{i,j}$ is a nonnegative constant bounded by a certain value, say $\alpha_{\neg\Pi}$,
$y_i' + e_i \preceq \tau'$ where e_i is a nonnegative constant bounded by $\alpha_{\neg\Pi}$,
$y_j' + e_{i,j,d} \preceq z_i'^d$ where $e_{i,j,d}$ is a nonnegative constant bounded by $\alpha_{\neg\Pi} + d$,
$\tau' + e_{i,d} \preceq z_i'^d$ where $e_{i,d}$ is a nonnegative constant bounded by d.

Recall that an expression of the form $t_i + e_{i,j} \preceq t_j$ denotes $t_i + e_{i,j} < t_j$, $t_i + e_{i,j} \leq t_j$ or $t_i = t_j$. For the sake of conciseness, we will use simply e in the following instead of $e_i, e_{i,j}, e_{i,d}, e_{i,j,d}$. Context will make it clear. Without loss of generality, we will assume also that the upper bound value $\alpha_{\neg\Pi}$ is greater than or equal to any constant a_i^ψ appearing in expressions $y_i + a_i^\psi \ll \tau$ of any recursive clause of the program. (If it is not the case, one takes the greater constant a_i^ψ as for $\alpha_{\neg\Pi}$.) In the following, we will simply write α instead of $\alpha_{\neg\Pi}$. We would like to show that the above class Γ of gap-configurations is *closed* by application of the recursive clauses. This is to say that, given a gap-configuration $H(y', \tau', z'^J)$ made of the constraints:
$y_i' + e \preceq y_j'$ with e bounded by α,
$y_i' + e \preceq \tau'$ with e bounded by α,
$y_j' + e \preceq z_i'^d$ with e bounded by $\alpha + d$,
$\tau' + e \preceq z_i'^d$ with e bounded by d,
the application of an above recursive clause to $H(y', \tau', z'^J)$ yields a gap-configuration $G(y, \tau, z^J)$, which (after renaming of y, τ, z^J as y', τ', z'^J) still belongs to Γ. We say that Γ is the *closed class of gap-configurations*.

We also introduce a set Δ_K, called the *set of forbidden τ^K-atoms*, that is made of the following gap-atoms (involving variables of τ^K):
$\tau + e < \tau^d$ with $d \leq e$,
$\tau + e \leq \tau^d$ with $d < e$.

Remark 1

It is easy to that Γ is a *finite* set. A rough upperbound for $|\Gamma|$ is: $(2\alpha + 2)^{O(n^2)}(2\alpha + 2)^n(2k + 2)^{nk}(2(\alpha + k) + 2)^{O(n^2 k)}$ where n is the number of clocks, k the maximum element of the set K of upperbound constants for clock values, and α is as defined above. One thus sees that $|\Gamma|$ is simply exponential in n and in k. (This complexity result is consistent with the complexity result about the size of the "region graph" for timed-automata [1].)

Remark 2

It is immediate to see that any gap-atom of Δ_K is *unsatisfiable* together with the upperbound τ-requirement $\bigwedge_{d \in K} \tau^d = \tau + d$.

We explain hereafter how to use Γ and Δ_K in combination with Revesz's procedure in order to construct the set of ancestor states of a given input gap-configuration.

5.3 Application of recursive clauses with τ-requirements

Let us consider a gap-configuration $H(y', \tau', z'^J)$ of Γ. and let us consider a recursive clause of the form:

$$p_A(s, y, \tau, z^J) \coloneq \chi(y, \tau, z^J, y', \tau', z'^J, \tau^K), \bigwedge_{d \in K} \tau^d = \tau + d, \quad p_A(s', y', \tau', z'^J).$$

where $\chi(y, \tau, z^J, y', \tau', z'^J, \tau^K)$ denotes the gap-configuration $\tau \leq \tau' \wedge \rho_I(y, \tau, y') \wedge y + a^\psi \ll \tau \wedge \tau \ll z^{b^\psi} \wedge \tau' \ll z^{c^{s'}} \wedge \bigwedge_{(i,d) \in J \wedge i \notin I} z_i'^d = z_i^d \wedge \bigwedge_{(i,d) \in J \wedge i \in I} z_i'^d = \tau^d$.

The application of such a clause to H consists in taking the conjunction $H(y', \tau', z'^J) \wedge \chi(y, \tau, z^J, y', \tau', z'^J, \tau^K) \wedge \bigwedge_{d \in K} \tau^d = \tau + d$, then eliminating variables y', τ', z'^J and τ^K. More precisely, this is done in 3 steps.

Step 1: Elimination of y', τ', z'^J

This first step is performed using just Revesz's procedure: the gap-configuration $H \wedge \chi$ is constructed from gap-configuration H and gap-configuration χ through *merging*; then variables y', τ', z'^J are eliminated by *shortcutting*.

One thus gets either the answer *failure* when merging fails, or a conjunction $H'(y, \tau, z^J, \tau^K)$ of gap-inequalities of the form:

$y_i + e \preceq y_j$ with e bounded by α
$y_i + e \preceq \tau$ with e bounded by α
$y_j + e \preceq z_i^d$ with e bounded by $\alpha + d$
$\tau + e \preceq z_i^d$
$\tau + e \preceq \tau^d$
$y_j + e \preceq \tau^d$.

Step 1 has the following properties. Assuming that $H(y', \tau', z'^J)$ belongs to Γ and satisfies $\bigwedge_{(i,d) \in J} z_i'^d = y_i' + d \wedge \bigwedge_{i=1,\ldots,n} y_i' \leq \tau'$, then step 1 either fails, or generates a gap-configuration $H'(y, \tau, z^J, \tau^K)$ satisfying $\bigwedge_{(i,d) \in J} z_i^d = y_i + d \wedge \bigwedge_{i=1,\ldots,n} y_i \leq \tau$. Besides step 1 is *equivalence-preserving* in the following sense:

If step 1 leads to failure, this means that $H \wedge \chi$ is unsatisfiable. Otherwise, the output H' is such that:

$$(\exists y', \tau', z'^J \; H(y', \tau', z'^J) \wedge \chi(y, \tau, z^J, y', \tau', z'^J, \tau^K)) \quad \Leftrightarrow \quad H'(y, \tau, z^J, \tau^K).$$

Step 2: Elimination of τ^K

We now eliminate variables of τ^K occurring in H' as follows:

If there exists a gap-atom of H' specified by set Δ_K (viz., an inequation of the form $\tau + e < \tau^d$ with $d \leq e$, $\tau + e \leq \tau^d$ with $d < e$), then the clause application terminates with *failure*. Otherwise: every inequation of the form $y_j + e \preceq \tau^d$ of H' with $d < e$, is replaced by $y_j + e' \preceq \tau$ with $e' = e - d$, and all the other inequations involving variables of τ^K are dropped.

This yields a new system of inequations H'' of the form:

$y_i + e \preceq y_j$ with e bounded by α
$y_i + e \preceq \tau$ with e bounded by α
$y_j + e \preceq z_i^d$ with e bounded by $\alpha + d$
$\tau + e \preceq z_i^d$.

One can easily show that step 2 is *equivalence-preserving* in the following sense:

If step 2 leads to failure, this means that $H' \wedge \bigwedge_{d \in K} \tau^d = \tau + d$ is unsatisfiable. Otherwise, assuming $\bigwedge_{i=1,,,.n} y_i \leq \tau$, we have:
$(\exists \tau^K \; H' \wedge \bigwedge_{d \in K} \tau^d = \tau + d) \quad \Leftrightarrow \quad H''$.

Step 3: Test of closure

We now filter out gap-configurations H'', which violate the closed-form, as follows:

If H'' does not belong to $\Gamma(y, \tau, z^J)$ (i.e., if H'' contains a gap-atom of the form $\tau + e \preceq z_i^d$ with $e > d$), then the clause application terminates with *failure*. Otherwise H'' is the sought gap-configuration $G \in \Gamma$.

One can easily show that step 3 is *equivalence-preserving* in the following sense:

If step 3 leads to failure, this means that $H'' \wedge \bigwedge_{(i,j) \in J} z_i^d = y_i + d \wedge \bigwedge_{i=1,,,.n} y_i \leq \tau$ is unsatisfiable. Otherwise H'' is (equivalent to) G.

Recapitulation

By linking steps 1,2,3 together we obtain a procedure of application for a clause r of constraint χ with the following properties. Given a gap-configuration $H(y', \tau', z'^J) \in \Gamma(y', \tau', z'^J)$ satisfying $\bigwedge_{(i,d) \in J} z_i'^d = y_i' + d \wedge \bigwedge_{i=1,\ldots,n} y_i' \leq \tau'$, then the application of r to H either fails, or generates a gap-configuration $G(y, \tau, z^J) \in \Gamma(y, \tau, z^J)$, such that $\bigwedge_{(i,d) \in J} z_i^d = y_i + d \wedge \bigwedge_{i=1,\ldots,n} y_i \leq \tau$ holds. Besides, the procedure is *equivalence-preserving* in the following sense:

If application of r to H fails, it means that $H \wedge \chi \wedge \bigwedge_{d \in K} \tau^d = \tau + d$ is unsatisfiable. Otherwise, we have: $(\exists y', \tau', z'^J, \tau^K \; H \wedge \chi \wedge \bigwedge_{d \in K} \tau^d = \tau + d) \quad \Leftrightarrow \quad G$.

Assume now given a gap-configuration $H_{\neg\Pi}$ belonging to Γ and satisfying $\bigwedge_{(i,d) \in J} z_i'^d = y_i' + d \wedge \bigwedge_{i=1,\ldots,n} y_i' \leq \tau'$. Then by iterating the process of clause application (and renaming $G(y, \tau, z^J)$ as $G(y', \tau', z'^J)$), one generates an increasing set of gap-configurations G belonging to Γ satisfying $\bigwedge_{(i,d) \in J} z_i'^d = y_i' + d \wedge \bigwedge_{i=1,\ldots,n} y_i' \leq \tau'$. Since Γ is a *finite* set of gap-configurations (see remark 1 of section 5.2). The procedure is guaranteed to terminate, and yields a finite set of gap-configurations characterizing *exactly* all the ancestor states of the gap-configuration $H_{\neg\Pi}$.

Example 4

Let us consider program of example 3, and let us show that in location 22, one has always the property $\Pi : \quad \tau < y_3 + 5$. In order to do this, we consider the gap-configuration $H(y', \tau', z'^J)$ associated with the negation $\neg\Pi$, viz: $y_3' + 5 \leq \tau'$, and we compute the set of ancestors of H by applying clauses *in*, *down*, *app* in a backward manner. With clause *in* applied to H, we go backward from location 22 to location 12, and generate gap-configuration $G : y_1 + 2 \leq \tau < z_1^5 \wedge y_3 + 5 \leq z_1^5$. This derivation is illustrated figure 2, where the rightmost graph represents H, the leftmost graph G, and the middle graph the constraint χ of clause *in*. The dashed edge in the middle graph corresponds to the edge added by merging with H. G is obtained by shortcutting y', τ', z'^J from the merged graph.

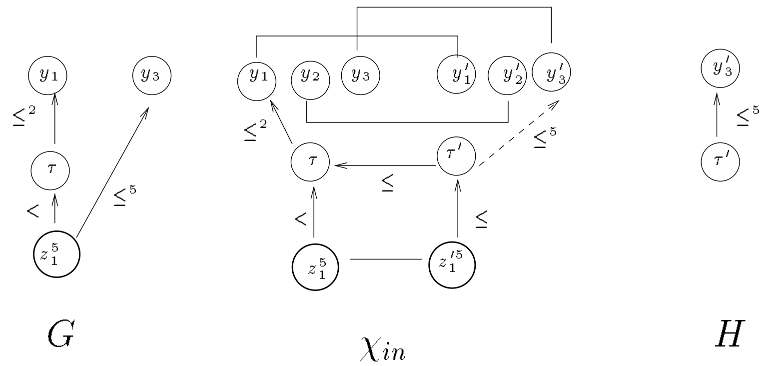

Fig. 2. Backward application of rule *in* (step 1)

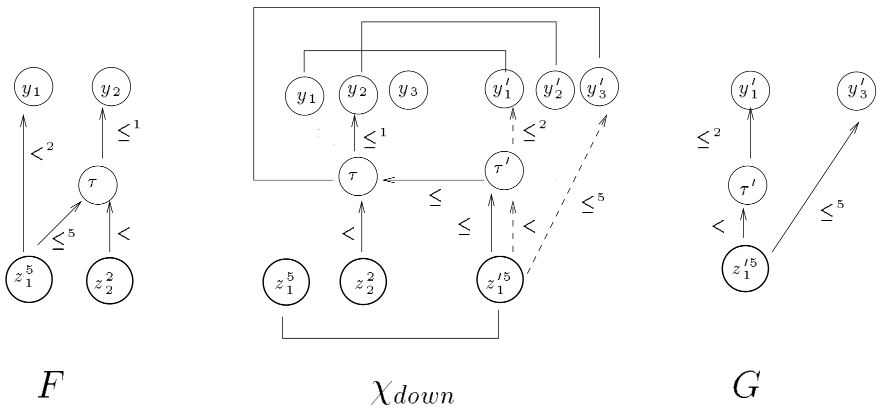

Fig. 3. Backward application of rule *down* (step 1)

After renaming of variables y, τ, z^J of G as y', τ', z'^J, one proceeds by applying clause *down* to G. We thus reach location 11 and the associated gap-configuration is $F : y_2 + 1 \leq \tau < z_2^2 \wedge \tau + 5 \leq z_1^5 \wedge y_1 + 2 < z_1^5$, This is illustrated on figure 3.

Then, after renaming of variables of F, one proceeds by applying clause *app* to F. We reach location 00, and the associated gap-configuration is $E : \tau + 1 < \tau^2 \wedge \tau + 6 \leq \tau^5$. This is illustrated on figure 4.

But now step 2, applied to E, leads to failure because constraint $\tau + 6 \leq \tau^5$ of E belongs to Δ_K. The state corresponding to E is thus rejected, and no more clause is applicable. The set of ancestor states is therefore made of H coupled with location 22, G coupled with 12, and F coupled with 11. Since none of these states contains the initial location 00, we conclude that property Π is true.

6 Extended Models of Timed Automata

Recall that a timed-automaton transition, labelled l from s to s', is a tuple of the form $\langle s, l, \psi, I, s' \rangle$ where $s, s' \in S$, $l \in L$, ψ is the guard, and $I \subseteq \{1, ..., n\}$ is the subset of indices of clocks to be reset to 0. We have represented such a transition as a recursive clause of the form:

$$p_A(s', y', \tau', z'^J) \text{ :- } \chi(y, \tau, z^J, y', \tau', z'^J, \tau^K), \bigwedge_{d \in K} \tau^d = \tau + d, \quad p_A(s, y, \tau, z^J).$$

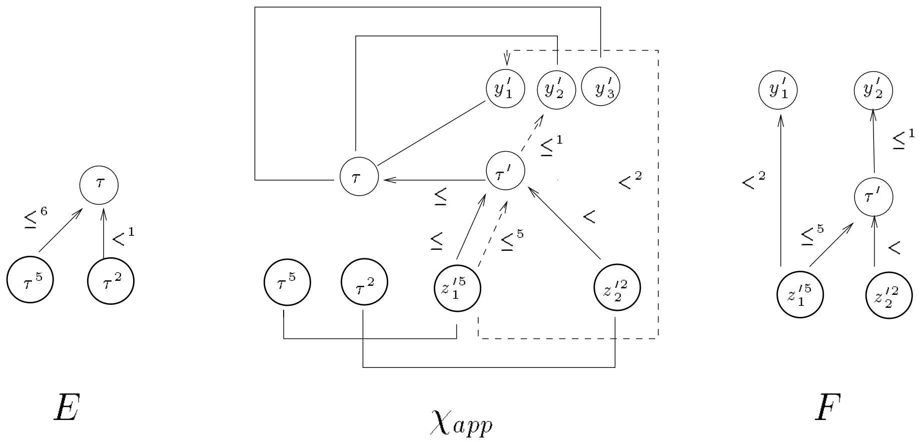

Fig. 4. Backward application of rule *app* (step 1)

where $\chi(y, \tau, z^J, y', \tau', z'^J, \tau^K)$ denotes the gap-configuration $\tau \leq \tau' \wedge \rho_I(y, \tau, y') \wedge y + a^{\psi} \ll \tau \wedge \tau \ll z^{b^{\psi}} \wedge \tau' \ll z^{c^{s'}} \wedge \bigwedge_{(i,d) \in J \wedge i \notin I} z'^d_i = z^d_i \wedge \bigwedge_{(i,d) \in J \wedge i \in I} z'^d_i = \tau^d$.

Remark 3

Let us notice that, in the definition given in section 3 for timed automata, a transition is viewed as a composition of an instantaneous "state"-transition from s to s' occurring at time τ, followed by a "time"-transition from τ to τ'. One could define alternatively a transition, in a symmetrical way, as a composition of a "time"-transition from τ to τ', followed by an instantaneous "state"-transition from s to s' occurring at time τ'. In such a case, a vector τ'^K should be used instead of τ^K, and recursive clauses associated with transitions, would be of the following form:

$$p_A(s', y', \tau', z'^J) \text{ :- } \chi'(y, \tau, z^J, y', \tau', z'^J, \tau'^K), \bigwedge_{d \in K} \tau'^d = \tau' + d, \quad p_A(s, y, \tau, z^J).$$

This variant will be used in section 6.2.

6.1 Extended locations

In our context of CLP programs, it is straightforward to extend the notion of location in the following way: s and s' are not any longer *constants* over a finite domain S, but are now *tuple of nonnegative real variables* w and w' that are constrained by an additional gap-configuration say $\sigma(w, w')$. A transition is now represented by the recursive clause:

$$\{\ p_A(w', y', \tau', z'^J) \text{ :- } \chi(y, \tau, z^J, y', \tau', z'^J, \tau^K), \sigma(w, w'), \bigwedge_{d \in K} \tau^d = \tau + d, \quad p_A(w, y, \tau, z^J). \ \}$$

The tuple of variables w can be considered as a tuple of *data registers* that are used to store some information while the the system evolves. For example, one can suppose that some transitions correspond to the sending or receipt of some messages, and one can use some variables of the state control w for keeping track of the latest transmitted messages.

The procedure described in the previous subsection extends naturally to take into account such extended locations. For example suppose that $\sigma(w, w')$ denotes a gap-configuration containing expressions of the form:

$w_i \odot w_j$, $w_i \odot w'_j$ and $w'_i \odot w'_j$, where $\odot$ denotes $=, <, \leq, >$ or $\geq$.

Then one defines a class Γ of closed-forms as a set of gap-configurations containing only gap-atoms of the form:

$y'_i + e \preceq y'_j$ with e bounded by α,
$y'_i + e \preceq \tau'$ with e bounded by α,
$y'_j + e \preceq z'^d_i$ with e bounded by $\alpha + d$,
$\tau' + e \preceq z'^d_i$ with e bounded by d,
$w'_i \odot w'_j$, with $\odot \in \{=, <, \leq, >, \geq\}$.

The set Δ_K of forbidden τ^K-atoms is left unchanged. The procedure for generating G from H using a recursive clause is exactly the same as in section 5.3, except that now, in step 1, H is merged with $\chi \wedge \sigma$ (instead of χ) and variables y', z'^J, w' are shortcut (instead of y', z^J). By iterating the procedure of clause application, one still generates in finite time the *exact* set of all the ancestors of the given input gap-configuration $H_{\neg\Pi}$.

Remark 4

As far as we know, in spite of the simplicity of its statement, the result seems new. In tools like those described in [6][8], one can handle a form of extended location (by the way, e.g., of parameters), but the resulting set of generated states is not guaranteed to be the exact set of ancestors as here, but simply an overapproximation.

6.2 Another scheme of timed-constraints

It is possible to apply our idea of closed-form evaluation, to timed transition systems modelled by recursive clauses, which incorporate different schemes for timed-constraints χ. However, in general, when applying a recursive clause to a gap-configuration H, the property of equivalence-preserving will be lost: roughly speaking, one will obtain a gap-configuration G, which will satisfy only the implication $(H \wedge \chi \wedge \bigwedge_{d \in K} \tau'^d = \tau' + d) \Rightarrow G$, but not the converse. As a consequence, when starting from a gap-configuration $H_{\neg\Pi}$, the iterative application of the recursive clauses will give a finite *overapproximation* of the set of ancestors of $H_{\neg\Pi}$ instead of the exact set of ancestors as before.

We explain the method on a special scheme of timed system that we have encountered in practice. Recursive clauses (expressed here in a backward manner, with $p_A(w, y, \tau)$ as a head and $p_A(w', y', \tau')$ as a recursive call) are of the form:
$\{\ p_A(w, y, \tau)$:- $\chi(y, \tau, y', \tau', \tau'^K), \sigma(w, w'), \bigwedge_{d \in K} \tau'^d = \tau' + d,\ p_A(w', y', \tau').\ \}$

Note that there is no vector of variables z^J (resp. z'^J). Note also that we use τ'^K rather than τ^K (see remark 3). Constraint $\sigma(w, w')$ is defined, as previously, as a conjunction of atoms of the form: $w_i \odot w_j$, $w_i \odot w'_j$ and $w'_i \odot w'_j$ with $\odot \in \{=, <, \leq, >, \geq\}$. Constraint $\chi(y, \tau, y', \tau', \tau'^K)$ is now assumed to be a conjunction of gap-atoms of the form
$\tau \leq \tau'$, $y_i \odot \tau$, $y_i \odot \tau'$, $y'_i \odot \tau$, $y'_i \odot \tau'$, $y_i \odot y_j$, $y_i \odot y'_j$, $y'_i \odot y'_j$, $y_i \odot \tau'^d$, $y'_i \odot \tau'^d$ with $\odot \in \{=, <, \leq, >, \geq\}$.

The class $\Gamma(w, y, \tau)$ of closed-forms is now defined to be the set of gap-configurations containing expressions of the form: $\tau \odot y_i$, $y_i \odot y_j$, $w_i \odot w_j$ with $\odot \in \{=, <, \leq, >, \geq\}$. Note that Γ is still finite: an upperbound of $|\Gamma|$ is $5^n \times 5^{O(n^2)} \times 5^{O(m^2)}$ where n is the number of variables y_i, and m the number of variables w_i.
The set Δ_K of forbidden τ'^K-atoms is defined as follows: every gap-atom δ of Δ_K

is an expression of the form $t_1 + e \ll t_2$, with $t_1, t_2 \in \tau'^K \cup \{\tau'\}$, $\ll \in \{<, \leq\}$ and $e \in \mathbb{N}$, such that $\delta \wedge \bigwedge_{d\in K} \tau'^d = \tau' + d$ is unsatisfiable.

The application of a recursive clause to a gap-configuration H of Γ is then adapted as follows:

Step 1 merges $H(w', y', \tau')$ with $\chi(y, \tau, y', \tau', y'^K) \wedge \sigma(w, w')$, and shortcuts variables w', y'; if merging fails, then the clause application fails, otherwise let $H'(w, y, \tau, \tau', \tau'^K)$ the resulting gap-configuration.

Step 2 tests if $H'(w, y, \tau, \tau', \tau'^K)$ contains some forbidden gap-atoms specified by Δ_K, and leads to failure in such a case; otherwise it proceeds by shortcutting variables τ', τ'^K, thus yielding a gap-configuration $G(w, y, \tau)$.

Note that step 3 (test of closure) has been omitted because it is superfluous: due to the general form of χ and Γ, one can easily show that a gap-configuration G produced at step 2 always belongs to Γ. Note however that step 2 is not any longer equivalence-preserving (step 2 ensures only: $H' \wedge \bigwedge_{d\in K} \tau'^d = \tau' + d \Rightarrow G$ but the converse does not hold any longer in general.) This is why the iterated application of recursive clauses generates an overapproximation of the set of ancestor states, and not any longer the exact set. The iterated application process is still guaranteed to terminate because Γ is finite.

We illustrate hereafter such a scheme with an example.

Example 5

This example corresponds to the modelling of a reactive program used for controlling dataflow rates in an ATM network [9]. There are 3 time variables y_{tfi}, y_{tla}, y_T and 4 data variables $w_{efi}, w_{fla}, w_{acr}, w_{approx}$. The program is defined by the following initial clause:

$$p_A(w_{efi}, w_{ela}, w_{acr}, w_{approx}, y_{tfi}, y_{tla}, y_T, \tau) :- \quad y_{tfi} = y_{tla} = \tau, \quad w_{efi} = w_{ela} = w_{acr} = w_{approx}.$$

and 9 recursive clauses among which are:

%1

$$\begin{aligned} &p_A(w'_{efi}, w'_{ela}, w'_{acr}, w'_{approx}, y'_{tfi}, y'_{tla}, y'_T, \tau') :- \\ &\qquad y_{tfi} \leq \tau \leq \tau' < y'_{tfi} = y'_{tla} = \tau'^{\lambda_3}, y_T = y'_T, \\ &\qquad w_{acr} = w'_{acr} < w'_{ela} = w'_{efi}, \ w_{efi} = w_{ela}, \ \eta(w_{approx}, w'_{approx}, y_T, \tau', \tau'^{\lambda_2}, \tau'^{\lambda_3}), \\ &\qquad \tau'^{\lambda_2} = \tau' + \lambda_2, \tau'^{\lambda_3} = \tau' + \lambda_3, \\ &\qquad p_A(w_{efi}, w_{ela}, w_{acr}, w_{approx}, y_{tfi}, y_{tla}, y_T, \tau). \end{aligned}$$

%2

$$\begin{aligned} &p_A(w'_{efi}, w'_{ela}, w'_{acr}, w'_{approx}, y'_{tfi}, y'_{tla}, y'_T, \tau') :- \\ &\qquad y_{tfi} \leq \tau \leq \tau' < y'_{tfi} = y'_{tla} = \tau'^{\lambda_2}, y_T = y'_T, \\ &\qquad w'_{efi} = w'_{ela} \leq w_{acr} = w'_{acr}, \quad w_{efi} = w_{ela}, \quad \eta(w_{approx}, w'_{approx}, y_T, \tau', \tau'^{\lambda_2}, \tau'^{\lambda_3}), \\ &\qquad \tau'^{\lambda_2} = \tau' + \lambda_2, \tau'^{\lambda_3} = \tau' + \lambda_3, \\ &\qquad p_A(w_{efi}, w_{ela}, w_{acr}, w_{approx}, y_{tfi}, y_{tla}, y_T, \tau). \end{aligned}$$

%3

$$\begin{aligned} &p_A(w'_{efi}, w'_{ela}, w'_{acr}, w'_{approx}, y'_{tfi}, y'_{tla}, y'_T, \tau') :- \\ &\qquad \tau \leq \tau' = y_{tfi} \leq y_{tla} = y'_{tfi} = y'_{tla}, y_T = y'_T, \\ &\qquad w'_{efi} = w'_{ela} = w_{ela}, \quad w'_{acr} = w_{efi}, \quad w_{approx} = w'_{approx}, \\ &\qquad \tau'^{\lambda_2} = \tau' + \lambda_2, \tau'^{\lambda_3} = \tau' + \lambda_3, \\ &\qquad p_A(w_{efi}, w_{ela}, w_{acr}, w_{approx}, y_{tfi}, y_{tla}, y_T, \tau). \end{aligned}$$

where λ_2 and λ_3 denote constants (with $0 < \lambda_3 < \lambda_2$), and $\eta(w_{approx}, w'_{approx}, y_T, \tau', \tau'^{\lambda_2}, \tau'^{\lambda_3})$ is the disjunctive [7] formula

[7] As usual, a clause with a disjunctive body of the form $p_A(x')$:-$\zeta_1(x, x') \vee \zeta_2(x, x'), p_A(x)$ abbreviates for the set of clauses: $\{\ p_A(x')$:-$\zeta_1(x, x'), p_A(x).\ \ p_A(x')$:-$\zeta_2(x, x'), p_A(x)\ \}$.

$(w'_{approx} = w_{approx} \wedge y_T < \tau'^{\lambda_3})$
$\vee \ (w'_{approx} = w_{approx} \wedge w'_{ela} \leq w_{approx} \ \wedge \ \tau'^{\lambda_3} \leq y_T < \tau'^{\lambda_2})$
$\vee \ (w'_{approx} = w'_{ela} \wedge w_{approx} < w'_{ela} \ \wedge \ \tau'^{\lambda_3} \leq y_T < \tau'^{\lambda_2})$
$\vee \ (w'_{approx} = w'_{ela} \wedge \tau'^{\lambda_2} \leq y_T)$.

The elements of the closed-form class Γ are gap-configurations containing expressions of the form:
$y_i \odot y_j, \tau \odot y_i, w_i \odot w_j$
with $y_i, y_j \in \{y_{tfi}, y_{tla}, y_T\}$, $w_i, w_j \in \{w_{efi}, w_{ela}, w_{acr}, w_{approx}\}$ and $\odot \in \{=, <, \leq, >, \geq\}$.
The set Δ_K of forbidden τ'^K-atoms is: $\{\tau'^{\lambda_2} \leq \tau', \tau'^{\lambda_3} \leq \tau', \tau'^{\lambda_2} \leq \tau'^{\lambda_3}\}$.
It is easy to see that any atom δ of Δ_K together with the τ-requirement constraint $\tau'^{\lambda_2} = \tau' + \lambda_2 \wedge \tau'^{\lambda_3} = \tau' + \lambda_3$ is unsatisfiable (because of the given assumption $0 < \lambda_3 < \lambda_2$).

6.3 Simulation of step 2 using Revesz's procedure

We explain here that it is possible to enrich the body of the clauses with gap-order constraints in order to simulate step 2 through Revesz's evaluation procedure. This is done by taking the conjunction of the negated atoms $\neg\delta$ of Δ_K (for those expressible as gap-atoms), and adding it within the body of each recursive clause. The failure detection of step 2 is now performed by means of the merging operation.
The whole procedure can be thus simulated using only Revesz's mechanisms of merging and shortcutting. Let us explain this in a more direct way. Consider the (backward) recursive clauses modelling the program:
$\{\ p_A(w, y, \tau) \text{ :- } \chi(y, \tau, y', \tau', \tau'^K), \sigma(w, w'), \bigwedge_{d \in K} \tau'^d = \tau' + d,\ p_A(w', y', \tau').\ \}$

We can consider the conjunction of negated atoms $\neg\delta$ of Δ_K as a gap-configuration which *overapproximates* the τ'^K-requirement constraints $\bigwedge_{d \in K} \tau'^d = \tau' + d$ (that are not under gap form). By replacing the τ'^K-requirements with this conjunction of $\neg\delta$, we obtain an *abstraction* of the program that is now a pure gap-program. One can compute the least-fixed point of this abstract program by applying Revesz's bottom-up evaluation procedure. This yields an overapproximation of the set of ancestors. This is illustrated on the example below.

Example 6

Consider the example of section 6.2. Here the conjunction of the negated atoms of Δ_k is $\tau' < \tau'^{\lambda_3} < \tau'^{\lambda_2}$. We add this conjunction within the body of each recursive clause defining the transitions of the reactive system (and drop the τ'^K-requirement constraints). Suppose now that one wants to prove the property Π:
$\tau \leq y_T < y_{tfi} \Rightarrow w_{approx} \leq w_{acr}$.
We attempt to prove it by considering the gap-configuration $H_{\neg\Pi} \in \Gamma$ corresponding to the negation of Π, i.e: $\tau \leq y_T < y_{tfi} \wedge w_{acr} < w_{approx}$, and by applying Revesz's bottom-up procedure to the modified program. This yields a set of 46 gap-configurations (in 2400 sec on a SparcStation). [8]

After eliminating subsumed gap-configurations (i.e., gap-configurations which are consequences of other ones), one gets the simple formula $\mathcal{D}^{-1}$:
$(\tau \leq y_T < y_{tfi} \ \wedge \ w_{acr} < w_{approx})$
$\vee \ (\tau \leq y_{tfi} \leq y_T < y_{tla} \ \wedge \ w_{efi} < w_{approx})$
$\vee \ (y_{tfi} \leq \tau \leq y_T \ \wedge \ w_{efi} = w_{ela} \ \wedge \ w_{acr} < w_{approx})$
$\vee \ (\tau \leq y_{tfi} = y_{tla} < y_T \ \wedge \ w_{efi} < w_{approx})$

[8] We use the implementation of Revesz's procedure written by J. Richardson [4].

$\vee \quad (\tau \leq y_{tfi} \leq y_{tla} \leq y_T \;\; \wedge \;\; w_{ela} < w_{approx})$
The proof is completed by checking that the initial gap-configuration $y_{tfi} = y_{tla} = \tau \wedge w_{efi} = w_{ela} = w_{approx} = w_{acr}$ is not an instance of any disjunct of $\mathcal{D}^{-1}$. Property Π belongs to a set of properties stating the correctness of the reactive program. The other properties have been proved as well using our method.

7 Conclusion

We have presented an original method based on Revesz's bottom-up evaluation procedure (adapted from integers to reals) in order to treat the reachability problem for timed-automata. This provides us with an interesting alternative to methods based on region graphs [1,2]. Our method applies naturally to two extensions of timed-automata. The first extension deals with locations that do not take any longer their values on a finite domain, but are represented as real variables satisfying additional constraints. This allows to model the notion of unbounded data registers. Our method is guaranteed to compute in finite time the exact set of ancestors of a given input state for this extension. A further extension of timed-automata consists in combining data variables and time variables inside a new scheme of arithmetic constraints modelling transition guards. In order to treat the reachability problem in this latter case, we have however to perform a static *abstraction* of the program encoding the reachability relation. The method accordingly is only able to construct an *overapproximation* of the set of ancestor states instead of the exact set. We have nevertheless demonstrated the practical interest of the extended method by using it to mechanize the correctness proof of a sophisticated reactive program designed for controlling dataflow rates on ATM networks.

Acknowledgement. I am very grateful to Béatrice Bérard and François Laroussinie for numerous helpful discussions on timed automata.

References

1. R. Alur, C. Courcoubetis and D. Dill. "Model-Checking in Dense Real-Time". *Information and Computation 104:1*, 1993, pp. 2–34.
2. R. Alur and D. Dill. "Automata for Modeling Real-Time Systems". *Proc. 17th ICALP*, LNCS 443, Springer-Verlag, 1990, pp. 322–335.
3. G. Daws and S. Yovine. "Two Examples of Verification of Multirate Timed Automata with Kronos". *Proc. 2nd European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, 1995, pp.49–67.
4. L. Fribourg and J. Richardson. *Symbolic Verification with Gap-Order Constraints.* Technical Report LIENS-96-3, Ecole Normale Supérieure, Paris, February 1996 (available by anonymous ftp on ftp.ens.fr in /pub/reports/liens or on http://www.dmi.ens.fr/dmi/preprints).
5. N. Halbwachs. "About Synchronous Programming and Abstract Interpretation", *Proc. SAS'94*, LNCS 864, Springer-Verlag, 1994, pp.179–192.
6. T. Henzinger, P.-H. Ho and H. Wong-Toi. "A user Guide to HYTECH". *Proc. TACAS'95*, LNCS 1019, Springer-Verlag, 1995, pp. 41–71.
7. T. Henzinger, X. Nicollin, J. Sifakis and S. Yovine. "Symbolic Model Checking for Real-Time Systems", IEEE Symp. on Logic in Computer Science, 1992, pp. 394–406.
8. N. Halbwachs. P. Raymond and Y.-E. Proy, "Verification of Linear Hybrid Systems by Means of Convex Approximation", *Proc. SAS'94*, LNCS 864, Springer-Verlag, 1994.
9. J.-F. Monin and F. Klay. "Correction of an Algorithm for ABR Conformance". *Internal Note*, CNET, December 1996.
10. P.Z. Revesz. "A Closed Form for Datalog Queries with Integer Order". *Proc. Intl. Conf. on Database Theory*, LNCS 470, Springer-Verlag, 1990, pp. 187–201.

11. P.Z. Revesz. "A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints", *Theoretical Computer Science*, 1993, vol. 116, pp. 117-149.
12. L. Urbina and G. Riedewald. "Simulation and Verification in Constraint Logic Programming". *Proc. 2nd European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, 1995, pp.85–104.