URL: http://www.elsevier.nl/locate/entcs/volume81.html 14 pages

The Geometry of Timed PV Programs

Ulrich Fahrenberg

Dept. of Mathematical Sciences, Aalborg University
9220 Aalborg East, Denmark. Email: uli@math.auc.dk

Abstract

We introduce a real-time extension of the PV language: A timed PV program consists of a number of timed automata which synchronize by locking and releasing common resources. We give a geometric semantics to such programs in terms of local po-spaces, and we work towards making the established geometric techniques available for detecting deadlocks and unsafe configurations in timed PV programs.

1 Introduction

The PV formal language was introduced by E. Dijkstra in [3] and has since been applied in various areas of Computer Science. In [2], a geometric understanding of PV programs is developed, in terms of *progress graphs*. In [6], these ideas are pursued further to develop an algorithm which is *geometric* in spirit, to detect deadlocks and unsafe states in simple PV programs without loops and branching. The case of PV programs with loops is treated in [5] and [8], and in [5] it is noted that treating branching is easier than treating looping, hence the geometric techniques are applicable to the full calculus of untimed PV programs.

A PV process is commonly defined to be a regular expression on an alphabet $\{\mathtt{P}_a, \mathtt{V}_a \mid a \in \mathcal{O}\}$, subject to certain restrictions. Here $\mathcal{O}$ is a finite set of resources which can be locked (P) or released (V) by the processes, and a PV program then consists of a number of PV processes which synchronize by locking and releasing the common resources. In this article we consider finite PV *automata* rather than PV processes, i.e. finite automata on the alphabet $\{\tau, \mathtt{P}_a, \mathtt{V}_a \mid a \in \mathcal{O}\}$. This makes the transition to geometric objects much more simple than in [8], and it also enables us to introduce time into the PV formalism, by passing from finite automata to *timed* automata.

After a review of the geometric realization technique for untimed PV programs from [5] and [8], rewritten to treat PV automata instead of PV processes, in section 2, we introduce our timed PV formalism in sections 3 and 4. In section 5 we define a geometric realization mapping from timed PV programs to local po-spaces, and in section 6 we elaborate on how this geometric

realization technique could be applied to yield results similar to the ones of [5,6,8].

This article is based on the author's Master's thesis [4]; note however that notation and terminology have been changed slightly.

2 Untimed PV Programs

Throughout this article, we fix a finite set $\mathcal{O}$ of resources, which can be locked ($\mathtt{P}$) and released ($\mathtt{V}$) by the PV automata in question, and a semaphoricity mapping $s : \mathcal{O} \to \mathbb{N}_+$. We also let $\Sigma = \{\tau, \mathtt{P}_a, \mathtt{V}_a \mid a \in \mathcal{O}\}$.

2.1 PV Automata

A PV automaton (on $\mathcal{O}$) is a finite automaton $P = (Q_P, q_P^0, E_P, f_P, L_P)$ on the alphabet Σ. Here Q_P is a finite set of locations, $q_P^0 \in Q_P$ is the initial location, E_P is a finite set of edges, $f_P : E_P \to Q_P \times Q_P$ is the edge attaching mapping, and $L_P : E_P \to \Sigma$ is the edge labeling. For the mapping f_P, if $f_P(e) = (q_1, q_2)$, we will write $q_1 = f_P^-(e)$, $q_2 = f_P^+(e)$. If no confusion is possible, we will omit the indices P. Note that we allow multiple edges between any pair of locations. We declare all locations to be *accepting*.

To justify our approach, we show that the formalism of [8] embeds nicely into it: [8] is concerned with (non-branching) PV *processes*, which are $+$-free regular expressions on the set $\{\mathtt{P}_a, \mathtt{V}_a \mid a \in \mathcal{O}\}$. These are given the semantics that the set of (possible) executions of a PV process is the set of *prefixes* of the regular language generated by the expression. Hence the transition from PV processes to PV automata is achieved by first translating the regular expression to a finite automaton generating the same language, and then declaring all locations of the automaton to be accepting.

We introduce a successor relation $\leq \subseteq Q \times Q$ by letting $q_1 \leq q_2$ if and only if $q_1 = q_2$ or there exists $e \in E$ such that $f(e) = (q_1, q_2)$. Without loss of generality we can assume our PV automata to be *connected* in the sense that there exists a path $q^0 \leq \cdots \leq q$ for any $q \in Q$.

PV automata are subject to a *well-behavedness* condition: During any of their executions, resources are only to be released if they have been previously locked, and once they have been locked, they cannot be locked again without being released first. Hence execution sequences like, e.g., $\mathtt{V}_a.\mathtt{P}_a$ and $\mathtt{P}_a.\mathtt{P}_a$ are to be disallowed.

Imposing the well-behavedness condition on a given PV automaton $P = (Q, q^0, E, f, L)$ is achieved by associating with it a *resource-use characteristics* mapping $r : \mathcal{O} \times Q \to \mathbb{Z}$ as follows:

(i) Let $r_a(q^0) = 0$ for all $a \in \mathcal{O}$.

(ii) For all $e \in E$, $f(e) = (q_1, q_2)$, such that $r_a(q_1)$ has been defined and $r_a(q_2)$

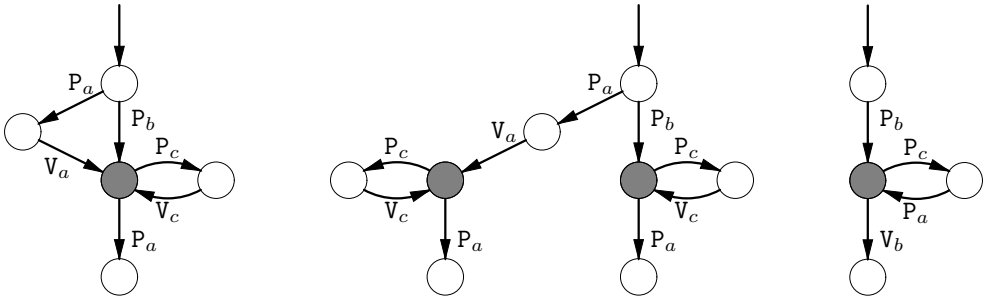

Fig. 1. Examples of resource-use conflicts. On the left, the shaded location has a conflict in r_b: Coming from the left, $r_b = 0$, coming from above, $r_b = 1$. The conflict is resolved by splitting up the offending location in the second automaton. The automaton on the right has an unresolvable resource-use conflict.

has not for some $a \in \mathcal{O}$, let

$$r_a(q_2) = \begin{cases} r_a(q_1) + 1 & \text{if } L(e) = \mathtt{P}_a, \\ r_a(q_1) - 1 & \text{if } L(e) = \mathtt{V}_a, \\ r_a(q_1) & \text{else} \end{cases}$$

for all $a \in \mathcal{O}$.

Step (ii) is to be repeated until no more such e exist. As P is connected, the algorithm terminates with resource-use characteristics assigned to every location in Q. We assume that P has no *resource-use conflicts*, that is, the resource-use characteristics of a location is independent of which path one takes to it from q^0. This condition can easily be verified within the above algorithm, and some of the conflicts can be resolved by splitting up the offending locations, cf. figure 1.

With resource-use characteristics assigned to every location, the automaton is said to be well-behaved if and only if $r_a(q) \in \{0, 1\}$ for all $a \in \mathcal{O}$, $q \in Q$.

2.2 PV Programs

A PV program is a finite set $\mathcal{P} = \{P_1, \dots, P_n\}$ of well-behaved PV automata. The semantics of a PV program is, as in [8], given by associating a transition system with it:

A configuration of $\mathcal{P}$ is a function $\kappa : \mathcal{P} \to \bigcup_{i=1}^{n} Q_{P_i}$ mapping each automaton in $\mathcal{P}$ to one of its locations, i.e. such that $\kappa P_i \in Q_{P_i}$ for all $P_i \in \mathcal{P}$. The initial configuration is κ^0 given by $\kappa^0 P_i = q^0_{P_i}$ for all $P_i \in \mathcal{P}$. We transfer the successor relation $\leq$ to configurations of PV programs by letting $\kappa_1 \leq \kappa_2$ if and only if $\kappa_1 P_i \leq_{P_i} \kappa_2 P_i$ for all $P_i \in \mathcal{P}$, and we let $\leq^*$ denote the transitive closure of $\leq$.

The resource-use characteristics of a configuration κ is $r_a(\kappa) = \sum_{i=1}^n r_a(\kappa P_i)$, i.e. $r_a(\kappa)$ is the number of locks the processes together hold on resource a in the configuration κ.

A configuration κ is *allowed* iff $r_a(\kappa) \leq s(a)$. If $\kappa_1 \leq \kappa_2$ and both κ_1 and κ_2 are allowed, then $\kappa_1 \leq \kappa_2$ is an *allowed computation step*. An allowed computation step is denoted $\kappa_1 \mapsto \kappa_2$; the transitive closure of $\mapsto$ we indicate by $\mapsto^*$.

If $\mathrm{Con_a}\,\mathcal{P}$ denotes the set of allowed configurations of the PV program $\mathcal{P}$, then the transition system $(\mathrm{Con_a}\,\mathcal{P}, \kappa^0, \mapsto)$ gives the semantics of $\mathcal{P}$.

2.3 Geometric Realization

The underlying digraph (Q, E, f) of a PV automaton $P = (Q, q^0, E, f, L)$ can be understood as a geometric object: The nodes in Q are discrete points, and the edges in E are directed unit intervals $\vec{I}$. This assigns a local po-space to the digraph (Q, E, f) called its *geometric realization* and denoted $\mathsf{B}P$. [1]

We carry over the resource-use characteristics r_a to the geometric realization as follows: Given $x \in \mathsf{B}P$, let

$$\tilde{r}_a(x) = \begin{cases} r_a(x) & \text{if } x \in Q, \\ r_a(f^+(e)) & \text{if } x \in e \in E \text{ and } L(e) = \mathtt{P}_a, \\ r_a(f^-(e)) & \text{else.} \end{cases}$$

If x is a point on an edge, the above means that $r_a(x)$ is 1 as soon as r_a equals 1 in one of the endpoints of the edge, that is, any locations in which the automaton holds a lock on some resource are "fattened up" such that it also holds the lock on the edges incident with these locations.

The geometric realization of a PV *program* $\mathcal{P} = \{P_1, \ldots, P_n\}$ is the product space $\mathsf{B}\mathcal{P} = \prod_{i=1}^n \mathsf{B}P_i$, which as a product of local po-spaces again is a local po-space. Resource-use characteristics is carried over to $\mathsf{B}\mathcal{P}$ by letting $\tilde{r}_a(x) = \sum_{i=1}^n \tilde{r}_a(\pi_i\, x)$, where $\pi_i\, x = x_i$ is projection on the ith coordinate space. Configurations of $\mathcal{P}$ are mapped into $\mathsf{B}\mathcal{P}$ by defining $\tilde{\kappa} = (\kappa P_1, \ldots, \kappa P_n) \in \mathsf{B}\mathcal{P}$.

A point $x \in \mathsf{B}\mathcal{P}$ is said to be allowed if $\tilde{r}_a(x) \leq s(a)$ for all $a \in \mathcal{O}$. It is straightforward to see that $\tilde{r}_a(\tilde{\kappa}) = r_a(\kappa)$ for all configurations κ, hence κ is allowed if and only if $\tilde{\kappa}$ is allowed. The set of all allowed points in $\mathsf{B}\mathcal{P}$ is denoted $\mathsf{B}_\mathrm{a}\mathcal{P}$. It can be shown that $\mathsf{B}\mathcal{P}$ is a compact local po-space, and that $\mathsf{B}_\mathrm{a}\mathcal{P}$ is a closed subspace of $\mathsf{B}\mathcal{P}$.

If $\trianglelefteq \subseteq \mathsf{B}\mathcal{P} \times \mathsf{B}\mathcal{P}$ denotes the local order [2] on the local po-space $\mathsf{B}\mathcal{P}$, it is easy to see that $\kappa_1 \leq^* \kappa_2$ if and only if $\tilde{\kappa}_1 \trianglelefteq \tilde{\kappa}_2$. The following proposition, where $x \preccurlyeq^* y$ denotes the property that there exists a dipath from x to

[1] More precisely, the digraph (Q, E, f) is a semi-cubical set in the sense of [7], with Q being the set of 0-cubes, E the set of 1-cubes, and f^-, f^+ the first boundary mappings; and the local po-space we have in mind is the geometric realization $|\,(Q, E, f)\,|$ as defined in [7].

[2] Compare [10, def. 26] for a definition of local order.

y in $\mathsf{B_a}\mathcal{P}$, is much more difficult to prove and constitutes the core of the applicability of the geometric realization technique for untimed PV programs.

Proposition 2.1 *Given two configurations κ_1, κ_2, then $\kappa_1 \mapsto^* \kappa_2$ if and only if $\tilde{\kappa}_1 \preccurlyeq^* \tilde{\kappa}_2$, and as a consequence, κ_1 is a deadlock or unsafe if and only if $\tilde{\kappa}_1$ is a deadlock or unsafe.*

2.4 Linear PV Programs

A PV automaton $P = (Q, q^0, E, f, L)$ is said to be linear if there exist labelings $Q = (q_0, \dots, q_m)$, $E = (e_1, \dots, e_m)$, with $q_0 = q^0$, such that $f(e_j) = (q_{j-1}, q_j)$ for all $j = 1, \dots, m$. Linear PV automata correspond to the *loopless processes* of [8].

A PV *program* is called linear if all the automata that constitute it are linear. The geometric realization of a linear PV program is a (global) po-space, and in [6], an efficient algorithm is developed for finding deadlocks and unsafe points in the geometric realization of linear PV programs.

In [5], a technique called *delooping* is introduced, which makes the algorithm of [6] applicable to PV programs *with loops*. This is done by associating with a looping PV program a number of linear PV programs such that the unsafe region of the original program can be found as the intersection of the unsafe regions of the linear programs.

3 Timed PV Automata

As hinted in the introduction, we define a timed PV automaton to be a timed automaton $P = (Q, q^0, E, f, L, C, \varphi, \rho)$ on the alphabet $\Sigma = \{\tau, \mathtt{P}_a, \mathtt{V}_a \mid a \in \mathcal{O}\}$. In this expression, Q, q^0, E, f, and L are as in the untimed case, C is a finite set of clocks, $\varphi : Q \cup E \to \Phi(C)$ is a mapping assigning a *clock constraint* to each location and every edge, and $\rho : E \to 2^C$ assigns to every edge a set of clocks to be reset.

As before, all locations are accepting. For clock constraints, clock valuations, and valuation resets we use the terminology of [1], except that we also allow constraints on differences of clocks.

The following is standard in the timed automata formalism, see e.g. [1]; we state it here only to fix notation: A *state* of a timed PV automaton P is an element (q, v) of the set $S = Q \times \mathbb{R}^C_{\geq 0}$. A state (q, v) is *allowed* if $v \vDash \varphi(q)$. If (q_1, v_1), (q_2, v_2) are allowed states, then $(q_1, v_1) \mapsto (q_2, v_2)$ if either

- $q_2 = q_1$, and there exists $t \in \mathbb{R}_{\geq 0}$ such that $v_2 = v_1 + t$ and for all $0 \leq t' \leq t$, $v + t' \vDash \varphi(q_1)$, or
- there is $e \in E$ such that $f(e) = (q_1, q_2)$, $v_1 \vDash \varphi(e)$, and $v_2 = v_1[\rho(e) \leftarrow 0]$.

To maintain analogy with the untimed case, we need to define a "successor" relation $\leq$ on S such that the *allowed-successor* relation $\mapsto$ is a subset of $\leq$. This is done by declaring that $(q_1, v_1) \leq (q_2, v_2)$ if there is $t \in \mathbb{R}_{\geq 0}$ such that

$q_2 = q_1$ and $v_2 = v_1 + t$, or there is $e \in E$ such that $f(e) = (q_1, q_2)$ and $v_2 = v_1[\rho(e) \leftarrow 0]$. Note that $\leq$ coincides with $\mapsto$ if and only if $\varphi(Q \cup E) = \{\texttt{true}\}$, i.e. if the automaton has no timing constraints at all.

The *initial state* of the timed PV automaton P is $(q^0, v^0) \in S$, where v^0 is the clock valuation given by $v^0(c) = 0$ for all $c \in C$. We demand the initial state to be allowed; denoting the set of allowed states by S_a, the semantics of P is given by the transition system $(S_\mathrm{a}, \mapsto, (q^0, v^0))$.

The resource-use characteristics mapping $r : \mathcal{O} \times Q \to \mathbb{Z}$ we define as in the untimed case; this also gives us a notion of well-behaved timed PV automata. The mapping is extended to the *states* of P by decreeing that $r_a(q, v) = r_a(q)$.

4 Timed PV Programs

A timed PV program is a finite set $\mathcal{P} = \{P_1, \dots, P_n\}$ of well-behaved timed PV automata. The semantics of a timed PV program is again given by introducing a set of configurations and defining a transition system on it:

A configuration of $\mathcal{P}$ is a mapping $\kappa : \mathcal{P} \to \bigcup_{i=1}^n S_{P_i}$ such that $\kappa P_i \in S_{P_i}$ for all $P_i \in \mathcal{P}$. The initial configuration κ^0 is given by $\kappa^0 P_i = (q^0_{P_i}, v^0)$ for all $P_i \in \mathcal{P}$. Again we define a successor relation $\leq$ on configurations by declaring that $\kappa_1 \leq \kappa_2$ if and only if $\kappa_1 P_i \leq_{P_i} \kappa_2 P_i$ for all $P_i \in \mathcal{P}$.

Also as before, the resource-use characteristics of a configuration κ is defined to be $r_a(\kappa) = \sum_{i=1}^n r_a(\kappa P_i)$. κ is *allowed* if κP_i is allowed in S_{P_i} for all $P_i \in \mathcal{P}$ *and* $r_a(\kappa) \leq s(a)$ for all $a \in \mathcal{O}$. As initial states of timed PV automata are defined to be allowed, and $r_a(\kappa^0) = 0$ for all $a \in \mathcal{O}$, the initial configuration κ^0 is itself allowed.

A successor relation $\kappa_1 \leq \kappa_2$ is an *allowed computation step*, again denoted $\kappa_1 \mapsto \kappa_2$, if κ_1, κ_2 are allowed and $\kappa_1 P_i \mapsto_{P_i} \kappa_2 P_i$ for all $P_i \in \mathcal{P}$. Again letting $\mathrm{Con}_\mathrm{a} \mathcal{P}$ denote the set of allowed configurations of $\mathcal{P}$, the semantics of $\mathcal{P}$ is given by the transition system $(\mathrm{Con}_\mathrm{a} \mathcal{P}, \kappa^0, \mapsto)$.

Note that, compared to the untimed case, the definition of allowed configurations and allowed computations now has an extra component stemming from the individual automata the program is composed of.

5 Geometric Realization

To obtain an analogy of the geometric realization notion for timed PV programs, we have to apply the technique of section 2.3 *twice*. The geometric realization of a single (untimed) PV automaton was simply a digraph, i.e. a one-dimensional local po-space, whereas the geometric realization of a timed PV automaton with d clocks will be a $(d+1)$-dimensional local po-space. Also, we will have the notion of allowed and forbidden points already for the geometric realization of timed PV *automata*, not only for the programs.

Given a set $C = \{c_1, \dots, c_d\}$ of clocks, there is a bijective correspondence between clock valuations in $\mathbb{R}^C_{\geq 0}$ and points of the space $\mathbb{R}^d_{\geq 0}$ given by $\tilde{v} =$

$(v(c_1),\dots,v(c_d))$. Also, given a clock constraint $\varphi \in \Phi(C)$, we can define an associated subset $\tilde{\varphi} \in \mathbb{R}^d_{\geq 0}$—called a *clock zone* by some authors—by $\tilde{\varphi} = \{\tilde{v} \mid v \vDash \varphi\}$. Dividing out logical equivalence in $\Phi(C)$, the mapping $\varphi \mapsto \tilde{\varphi}$ also becomes bijective. With a valuation reset $v[D \leftarrow 0]$ we associate a projection mapping $\pi_D : \mathbb{R}^d_{\geq 0} \to \mathbb{R}^d_{\geq 0}$ setting all coordinates x_i with $c_i \in D$ to 0 and leaving the others untouched. For later use we record the following basic facts about the interplay between these three mappings:

Lemma 5.1 *Given $v \in \mathbb{R}^C_{\geq 0}$, $D \subseteq C$, and $\varphi \in \Phi(C)$, then*

- *$v \vDash \varphi$ if and only if $\tilde{v} \in \tilde{\varphi}$,*
- *$(v[D \leftarrow 0])^\sim = \pi_D(\tilde{v})$, and*
- *$v[D \leftarrow 0] \vDash \varphi$ if and only if $\tilde{v} \in \pi_D^{-1}(\tilde{\varphi})$.*

Proof. The first two assertions are clear from the definitions. As for the last, $v[D \leftarrow 0] \vDash \varphi$ if and only if $(v[D \leftarrow 0])^\sim \in \tilde{\varphi}$, which in turn is the case if and only if $\pi_D(\tilde{v}) \in \tilde{\varphi}$, and the latter is equivalent to $\tilde{v} \in \pi_D^{-1}(\tilde{\varphi})$. □

5.1 Timed PV Automata

The geometric realization of a timed PV automaton $P = (Q, q^0, E, f, L, C, \varphi, \rho)$, with $C = \{c_1,\dots,c_d\}$, is the space $\mathsf{B}P = (Q, E, f) \times \vec{\mathbb{R}}^d_{\geq 0}$, where (Q, E, f) again is to be understood as a local po-space, and $\vec{\mathbb{R}}^d_{\geq 0}$ is the space $\mathbb{R}^d_{\geq 0}$ with the (standard) order

$$(x_1,\dots,x_d) \leq (y_1,\dots,y_d) \quad \text{iff} \quad x_i \leq y_i \text{ for all } i = 1,\dots,d.$$

The space $\mathsf{B}P$ is a local po-space, however unless $d = 0$, it is not compact. We call (Q, E, f) the *location space*, $\vec{\mathbb{R}}^d_{\geq 0}$ the *clock space*.

Given a state $(q, v) \in S$, its geometric realization is defined to be the point $(q, \tilde{v}) \in \mathsf{B}P$. A point $(x, \tilde{v}) \in \mathsf{B}P$ is said to be *allowed* if either $x \in Q$ and $\tilde{v} \in \tilde{\varphi}(x)$, or $x \in e$ for some $e \in E$ and

$$\tilde{v} \in \tilde{\varphi}(f^-(e)) \cap \tilde{\varphi}(e) \cap \pi^{-1}_{\rho(e)}(\tilde{\varphi}(f^+(e))). \tag{1}$$

The following proposition, where $\mathsf{B}_\mathrm{a}P \subseteq \mathsf{B}P$ again denotes the set of allowed points in $\mathsf{B}P$, shows that our definition of $\mathsf{B}_\mathrm{a}P$ is the "right" one:

Proposition 5.2 *Given $(x, \tilde{v}) \in \mathsf{B}P$, then $(x, \tilde{v}) \in \mathsf{B}_\mathrm{a}P$ if and only if, either $x \in Q$ and $v \vDash \varphi(x)$, or $x \in e$ for some $e \in E$ and*

- *$v \vDash \varphi(f^-(e))$,*
- *$v \vDash \varphi(e)$, and*
- *$v[\rho(e) \leftarrow 0] \vDash \varphi(f^+(e))$.*

Proof. The case $x \in Q$ is trivial. If $x \in e$ for some $e \in E$, the first two items correspond to $\tilde{v} \in \tilde{\varphi}(f^-(e)) \cap \tilde{\varphi}(e)$ in equation (1) above. Hence we are left

with showing that $\tilde{v} \in \pi_{\rho(e)}^{-1}(\tilde{\varphi}(f^+(e)))$ if and only if $v[\rho(e) \leftarrow 0] \vDash \varphi(f^+(e))$, which however is clear by the last item of lemma 5.1. □

Resource-use characteristics is defined on $\mathsf{B}P$ by first introducing it on the *space* (Q, E, f) as in section 2.3, and then "fattening it up" by declaring that $r_a(x, \tilde{v}) = r_a(x)$.

5.2 Timed PV Programs

The definitions for timed PV programs resemble the ones from the untimed case: The geometric realization of a timed PV program $\mathcal{P} = \{P_1, \dots, P_n\}$ is the local po-space $\mathsf{B}\mathcal{P} = \prod_{i=1}^n \mathsf{B}P_i$, which in general is not compact; the resource-use characteristics at a point $x \in \mathsf{B}\mathcal{P}$ is defined to be $\tilde{r}(x) = \sum_{i=1}^n \tilde{r}_a(\pi_i\, x)$; and configurations are mapped into $\mathsf{B}\mathcal{P}$ by defining $\tilde{\kappa} = (\kappa P_1, \dots, \kappa P_n)$, where the correspondence between states (q, v) and points $(q, \tilde{v})$ is implicit.

As for allowed points in $\mathsf{B}\mathcal{P}$, we again have the duality between points being forbidden due to over-use of resources and points being forbidden in the respective timed PV automata: We say that a point $x \in \mathsf{B}\mathcal{P}$ is allowed if $\pi_i\, x$ is allowed in all $\mathsf{B}P_i$ and $\tilde{r}_a(x) \leq s(a)$ for all $a \in \mathcal{O}$. As the equality $\tilde{r}_a(\tilde{\kappa}) = r_a(\kappa)$ also holds in the timed case, and by proposition 5.2, a configuration κ is allowed if and only if $\tilde{\kappa}$ is allowed.

6 Applying the Geometric Realization

So far we have introduced a timed PV formalism and defined a geometric realization function in close analogy to what has been done previously for untimed PV programs. To actually make our proposed geometric realization technique *work*, we should provide an analog of proposition 2.1 of section 2.3: Given configurations κ_1, κ_2, there should be an (allowed) execution path from κ_1 to κ_2 if and only if there is a dipath (in B_a) from $\tilde{\kappa}_1$ to $\tilde{\kappa}_2$. In the present section we shall see that this is *not* the case, and we shall propose different ways to handle the problems encountered.

For sake of simplicity, we confine ourselves to treat only timed PV *automata* in this section; let P be a given timed PV automaton with d clocks.

6.1 The Reset Problem: Not All Execution Paths Correspond to Dipaths

An essential feature of timed automata is their ability to reset clocks. Indeed, if the timed automaton in question never resets its clocks, then all clocks have the same value in any reachable state, hence we might as well have only *one* clock—and the results of [9] imply that a timed automaton with one clock is strictly less expressive than one with two or more clocks.

However if clocks can be reset, there exist execution paths which are neither continuous nor directed, cf. figure 2.

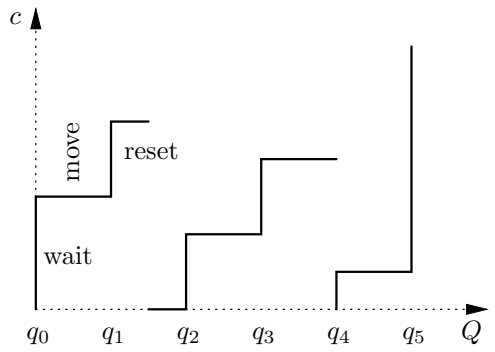

Fig. 2. A typical execution path in a timed automaton with one clock. As the clock is reset during the transition from q_1 to q_2, the execution path is not continuous.

This problem can be solved in several different ways. A first attempt is to simply change the order relation on $\mathsf{B}P$, such that now

$$(p, x_1, \dots, x_d) \leq (q, y_1, \dots, y_d) \quad \text{iff} \quad p \leq q \text{ and } \forall i : (x_i \leq y_i \text{ or } y_i = 0).$$

This makes execution paths directed, but still not continuous; however it also means that $\mathsf{B}P$ is not a local po-space anymore: Given any point $x = (q, 0, \dots, 0)$, there exists no neighbourhood of x in which $\leq$ is a partial order. This last problem might be avoided by defining the new order relation in some other way, but execution paths are still not continuous.

Our second proposal is to identify certain points in $\mathsf{B}P$: If $\check{e}$, for any $e \in E$, denotes the *midpoint* of e (where e is seen as a directed unit interval), define an equivalence relation $\sim$ on $\mathsf{B}P$ by

$$(\check{e}, \tilde{v}) \sim (\check{e}, \pi_{\rho(e)}(\tilde{v}))$$

for any $e \in E$, $\tilde{v} \in \mathbb{R}^d_{\geq 0}$, and pass to the quotient $\mathsf{B}P/\!\sim$. Certainly execution paths in $\mathsf{B}P/\!\sim$ are continuous (if the convention is applied that clocks are reset at the midpoints of edges); we believe that $\mathsf{B}P/\!\sim$ is a local po-space, and that execution paths in $\mathsf{B}P/\!\sim$ are dipaths.

This approach however has the caveat that $\mathsf{B}P/\!\sim$, even though it might be a local po-space, is a rather involved space which is likely to be difficult to handle in applications. A third way to attack this problem is to enhance the timed automata formalism such that the values of certain clocks can remain zero while the others are already started, an approach which only applies to *linear* timed automata as defined in section 6.3.

6.2 The Global-Time Problem: Not All Dipaths Correspond to Execution Paths

In the timed automata formalism, time is global, i.e. all clocks proceed at the same speed. This implies that execution paths run *diagonally* through the clock space, hence not all dipaths correspond to execution paths, cf. figure 3.

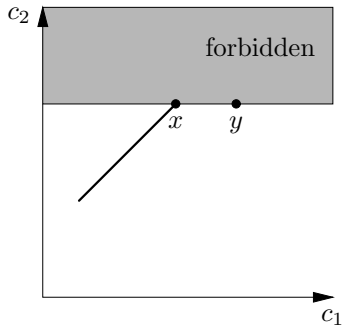

Fig. 3. An example of an execution path through a clock space constituted of two clocks. Both x and y are allowed, and there is a dipath from x to y, yet there is no execution path from x to y. In fact, x is a deadlock, as is y.

This situation can again be remedied by changing the order relation on $\mathsf{B}P$, the new one being

$$(p, x_1, \dots, x_d) \leq (q, y_1, \dots, y_d) \quad \text{iff} \quad \begin{aligned} &p \leq q, \\ &\forall i : x_i \leq y_i, \text{ and} \\ &y_1 - x_1 = \dots = y_d - x_d. \end{aligned}$$

With this order relation, all dipaths are execution paths. However this approach fits badly with our solution to the Reset Problem proposed in the next section.

If we want to stay with the standard order on $\mathbb{R}^d_{\geq 0}$, there is no other solution to this problem than abandoning the global-time approach altogether and consider local-time formalisms instead: As long as at least some of the clocks are synchronized with each other, there will be dipaths which do not correspond to execution paths.

6.3 Linear Timed PV Automata

In analogy to the approach in the untimed case, we should develop techniques to find deadlocks and unsafe configurations in *linear* timed PV programs, and we should attempt to transfer the delooping techniques of [5] to the timed case. In this paper we concentrate on the former; the latter is left open for future research.

As in the untimed case, a timed PV automaton $P = (Q, q^0, E, f, L, C, \varphi, \rho)$ is said to be linear if there exist labelings $Q = (q_0, \dots, q_m)$, $E = (e_1, \dots, e_m)$, with $q_0 = q^0$, such that $f(e_j) = (q_{j-1}, q_j)$ for all $j = 1, \dots, m$. The geometric realization of a linear timed PV automaton is a po-space.

For linear timed PV automata, the Reset Problem of section 6.1 has a third and much more elegant solution: to avoid resets altogether by introducing new clocks. Let P be the automaton from above, e_j one of its edges, and c_k one of its clocks, and assume that $c_k \in \rho(e_j)$. Introduce a new clock $\tilde{c}_{k,j}$, and in all invariants $\varphi(q_i)$, $\varphi(e_{i+1})$, $i \geq j$, replace any occurrence of c_k by $\tilde{c}_{k,j}$. The clock

$\tilde{c}_{k,j}$ is to be started only when location q_j is reached, i.e. its value remains 0 before q_j; it is clear that this replacement does not alter the semantics of P.

After all resets have been resolved by the procedure above, we have an automaton without resets, but instead with a new *start function* $\sigma : C \to Q$, assigning to every clock the location in which it is started. In the geometric realization of this new automaton, execution paths are dipaths.

6.4 Deadlocks and Unsafe Configurations

In the untimed case, the notions of deadlock and unsafe configuration are very intuitive: A non-final configuration κ is a deadlock if and only if there does not exist any κ' such that $\kappa \mapsto \kappa'$, and κ is unsafe if and only if no final configuration can be reached from κ, i.e. if $\uparrow\kappa \cap \mathcal{F} = \emptyset$. Here $\mathcal{F}$ is the set of final configurations, and $\uparrow\kappa = \{\kappa' \mid \kappa \mapsto^* \kappa'\}$.

For *linear* untimed PV programs, these notions are connected in that *a configuration is unsafe if and only if any execution from it reaches a deadlock.*[3] The algorithm of [6] finds all unsafe configurations in a given linear untimed PV program by recursively finding all deadlocks and "tracing them back" to find their associated unsafe configurations.

For timed PV automata the situation is somewhat more complicated. First, there are two kinds of deadlocks: Applying the definition from above gives a notion of *state deadlock*; a state is a state deadlock if no location switch can occur and time cannot increase. However there is also a second notion which is of interest, where the automaton is locked in the present location, but time might increase indefinitely:

Definition 6.1 A state $(q, v) \in Q \times \mathbb{R}^C_{\geq 0}$ is called a *location deadlock* if, for all $t \in \mathbb{R}_{\geq 0}$ such that $(q, v) \mapsto (q, v + t)$ and for all $e \in E$ such that $f^-(e) = q$,

- $(q, v + t)$ is non-final, and
- $v + t \not\models \varphi(e)$ or $(v + t)[\rho(e) \leftarrow 0] \not\models \varphi(f^+(e))$.

It is clear that every state deadlock is also a location deadlock.

Second, turning our attention to *linear* timed PV automata and assuming that the final states are exactly those whose location is q_m, i.e. the "last" location, we see that *a state is unsafe if and only if any execution from it reaches a location deadlock.* Hence we should find all *location* deadlocks and trace them back to find their associated unsafe states.

6.5 What We End Up With

In this last section we spell out the geometric realization technique for linear timed PV automata. We avoid the Reset Problem by the changes proposed in

[3] Here it is to be assumed that the only final configuration is the one consisting of the final locations of the individual automata.

section 6.3, and we work around the Global-Time Problem by stating explicitly when a dipath corresponds to an execution path.

Let $P = (Q, q^0, E, f, L, C, \varphi, \sigma)$ be a linear timed PV automaton, where we instead of the reset mapping ρ now have a start function $\sigma : C \to Q$, and let $C = \{c_1, \dots, c_d\}$, $Q = \{q_0, \dots, q_m\}$, $E = \{e_1, \dots, e_m\}$, where $q_0 = q^0$ and $f(e_j) = (q_{j-1}, q_j)$ for all $j = 1, \dots, m$. The geometric realization of P is

$$\mathsf{B}P = \overrightarrow{[0,m]} \times \vec{\mathbb{R}}^d_{\geq 0},$$

which is a po-space.

In the present setting, execution paths correspond to dipaths, however as not all clocks of the automaton are running in all locations, it is not the case anymore that execution paths run diagonally through the clock space. We keep track of which way execution paths are allowed to run by translating the start function σ to a mapping $\tilde{\sigma} : Q \times \mathbb{R}^d_{\geq 0} \to \mathbb{R}^d$, thereby defining a vector field on each instance of the $m + 1$ clock spaces. Execution paths are then (piecewise linear) curves through $\mathsf{B}P$ which are integral curves to the vector fields in the clock spaces and run horizontally in $[0, m]$.

First, define an accumulated start function $R : Q \to 2^C$, where $R(q)$ contains all clocks which are *running*, i.e. have been started already, in q:

$$R(q_j) = \{c \in C \mid \sigma(c) = q_i, i \leq j\}$$

This mapping translates to a function $\tilde{\sigma} : Q \to \{0,1\}^d$, by $\tilde{\delta}_i(q) = 1$ if and only if $c_i \in R(q)$, and we "fatten it up" to $Q \times \mathbb{R}^d_{\geq 0}$ by declaring that $\tilde{\sigma}(q, \tilde{v}) = \tilde{\sigma}(q)$.

With this in place, we are now able to give an exact characterization of the execution paths in $\mathsf{B}P$: A dipath $\gamma : \vec{I} \to \mathsf{B}P = \overrightarrow{[0,m]} \times \vec{\mathbb{R}}^d_{\geq 0}$ is an execution path if and only if there exists a partition $0 = t_0, t_1, \dots, t_n = 1$ of I such that

- $\gamma(t_i) \in Q \times I^d$ for all $i = 0, \dots, n$,
- $\gamma_{|[t_i, t_{i+1}]}$ is smooth for all $i = 0, \dots, n-1$, and
- for all $i = 0, \dots, n-1$, either

$$\mathrm{d}\gamma_{|[t_i,t_{i+1}]}|_{\gamma(t)} = \big((t_{i+1} - t_i)^{-1}, 0, \dots, 0\big)$$

  for all $t \in [t_i, t_{i+1}]$, or

$$\mathrm{d}\gamma_{|[t_i,t_{i+1}]}|_{\gamma(t)} = \big(0, \tilde{\sigma}_1(\gamma(t_i)), \dots, \tilde{\sigma}_d(\gamma(t_i))\big)$$

  for all $t \in [t_i, t_{i+1}]$.

Here $\mathrm{d}\gamma_{|[t_i,t_{i+1}]}|_{\gamma(t)}$ denotes the differential of γ restricted to the interval $[t_i, t_{i+1}]$, taken in the point $\gamma(t)$. Note that 1) any dipath satisfying the above constraints is actually piecewise *linear*, not just smooth, and 2) any *path* fulfilling these constraints is automatically a dipath.

7 Future Work

We believe that in the setting laid out in section 6.5, we are not far from being able to apply the algorithm of [6] to find deadlocks and unsafe configurations in timed PV programs, and we plan to do this in a future paper. There are two issues still to consider; however to us they are mainly of a computational, rather than conceptual, nature:

First, in the algorithm of [6], deadlocks are critical intersection points of *isothetic hyperrectangles*, whereas in our setting they are critical intersection points of "skew" regions in space, i.e. polyhedra where some of the edges are parallel to the coordinate axes, some are "partially diagonal". Computing intersections of hyperrectangles is easy, computing intersections of skew regions is more difficult. Also, in [6] deadlocks are critical intersection points of n hyperrectangles, where n is the dimension of the po-space in question, whereas in our setting, *all* clocks are stopped from running by a restriction on just *one* clock (since time is global), hence for us, deadlocks are critical intersection points of just *two* forbidden regions.

Second, in the original algorithm the set of unsafe points associated to a certain deadlock is a hyperrectangle "below" the critical intersection point. In our setting, the unsafe set would again be (the interior of) a "skew" region in space. Computing this set might be computationally expensive, but it is certainly possible.

As for the further future, we think that our approach of adding time to the "geometry of concurrency" world is sort of complementary to S. Sokołowski's concept of "continuous resources" as in [10], and we believe that a combination of these two approaches should be attempted.

We should however mention that in our approach to the geometry of timed PV automata, we were not able to use a lot of the machinery available from algebraic ditopology. We believe that this is inherent to the timed automata formalism we were using, and that another attempt to incorporate time into the "geometry of concurrency" world should be tried using some other timed formalism.

Acknowledgement

I am indebted to Lisbeth Fajstrup, Martin Raussen, Kim G. Larsen, and Eric Goubault for sharing many valuable thoughts with me, and to John Leth for correcting an early version of this article.

References

[1] R. Alur. Timed automata. In *Proc. CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer-Verlag, 1999.

[2] E. G. Coffman, M. J. Elphick, and A. Shoshani. System deadlocks. *ACM Comput. Surv.*, 3(2):67–78, June 1971.

[3] E. Dijkstra. Co-operating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–112. Academic Press, New York, 1968.

[4] U. Fahrenberg. Towards an efficient algorithm for detecting unsafe states in timed concurrent systems. Master's thesis, Aalborg University, Dept. of Mathematical Sciences, 9220 Aalborg East, Denmark, May 2002.

[5] L. Fajstrup. Loops, ditopology and deadlocks. *Mathematical Structures in Computer Science*, 10:459–480, 2000.

[6] L. Fajstrup, E. Goubault, and M. Raussen. Detecting deadlocks in concurrent systems. In *Proc. CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag, 1998.

[7] L. Fajstrup, E. Goubault, and M. Raussen. Algebraic topology and concurrency. Report R-99-2008, Department of Mathematical Sciences, Aalborg University, November 1999.

[8] L. Fajstrup and S. Sokołowski. Infinitely running concurrent processes with loops from a geometrical viewpoint. *Electronic Notes in Theoretical Computer Science*, 39(2), 2000.

[9] T. A. Henzinger, P. W. Kopke, and H. Wong-Toi. The expressive power of clocks. In *Proc. ICALP'95*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer-Verlag, 1995.

[10] S. Sokołowski. A case for po-manifolds. In *Preliminary Proceedings GETCO'02*, volume NS-02-5 of *BRICS Notes Series*. BRICS, Aarhus, October 2002.