Languages of Higher-Dimensional Timed Automata

Amazigh Amrane[1], Hugo Bazille[1], Emily Clement[2], and Uli Fahrenberg[1]

[1] EPITA Research Laboratory (LRE), France
hugo@lrde.epita.fr
[2] Université Paris Cité, CNRS, IRIF, Paris, France

Abstract. We present a new language semantics for real-time concurrency. Its operational models are higher-dimensional timed automata (HDTAs), a generalization of both higher-dimensional automata and timed automata. We define languages of HDTAs as sets of interval-timed pomsets with interfaces. As an application, we show that language inclusion of HDTAs is undecidable. On the other hand, using a region construction we can show that untimings of HDTA languages have enough regularity so that untimed language inclusion is decidable.

Keywords: higher-dimensional timed automaton, real-time concurrency, timed automaton, higher-dimensional automaton

1 Introduction

In order to model non-interleaving concurrency, models such as Petri nets [46], event structures [45], configuration structures [51, 52], or higher-dimensional automata (HDAs) [26, 47, 48] allow several events to happen simultaneously. The interest of such models, compared to other models such as automata or transition systems, is the possibility to distinguish concurrent and interleaving executions; using CCS notation [44], parallel compositions $a \parallel b$ are not the same as choices $a.b + b.a$.

Semantically, concurrency in non-interleaving models is represented by the fact that their languages do not consist of words but rather of partially ordered multisets (*pomsets*). As an example, Fig. 1 shows Petri net and HDA models which execute the parallel composition of $a.c$ and b; their language is generated by the pomset $[\begin{smallmatrix} a \to c \\ b \end{smallmatrix}]$ in which there is no order relation between a and b nor

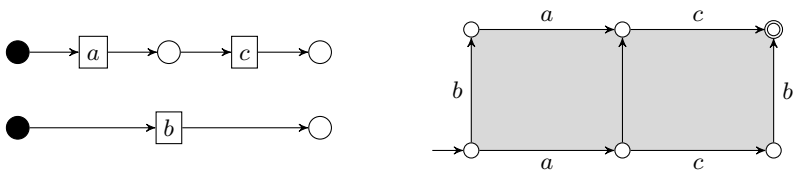

Fig. 1. Petri net and HDA models for $a.c \parallel b$.

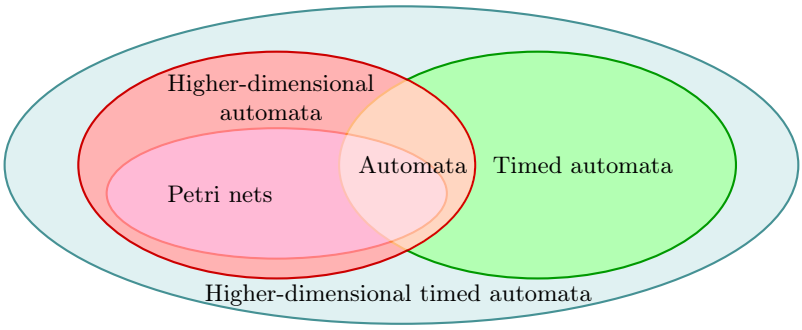

Fig. 2. Taxonomy of some models for time and concurrency

between b and c. However, these models and pomsets use logical time and make no statements about the precise durations or timings of events.

When we consider models for real-time systems, such as for example timed automata [3] which can model precise durations and timings of events, the distinction between concurrency and interleaving is usually left behind. Their languages are sets of *timed words*, that is, sequences of symbols each of which is associated with a timestamp that records when the associated event took place.

In this article, our goal is to propose a language-based semantics for concurrent real-time systems. Our aim is to combine the two semantics above, timed words for interleaving real time and pomsets for non-interleaving logical time.

Another such proposal was developed in [12], where, going back to [11], languages of time Petri nets [43] are given as sets of pomsets with timestamps on events, see also [18–20, 34]. Nevertheless, this creates problems of causality, as explained in [20] which notes that *"[t]ime and causality [do] not necessarily blend well in [...] Petri nets"*.

We put forward a different language-based semantics for real-time concurrency, inspired by recent work on interval-order semantics of higher-dimensional automata [5,26–28,32]. We use pomsets with *interval timestamps* on events, that is, every event has a start time and an end time, and the partial order respects these timestamps.

Our operational models for real-time concurrent systems are higher-dimensional timed automata (HDTAs), a simultaneous extension of timed automata [2,3] and higher-dimensional automata [26,47,48] (which in turn generalize (safe) Petri nets [49]), see Fig.2. These have been introduced in [25], where it is shown among other things that reachability for HDTAs may be decided using zones like for timed automata. We adapt the definition of HDTAs to better conform with the event-based setting of [26] and introduce languages of HDTAs as sets of pomsets with interval timestamps.

This article is organised as follows. We begin in Sect. 2 by recalling timed automata and expressing their language semantics using two perspectives: delay words and timed words. In Sect. 3, we revisit higher-dimensional automata and their languages, again focusing on two complementary perspectives, of step

sequences and pomsets with interfaces. In Sect. 4 we recall the definition of higher-dimensional timed automata and give examples.

The following sections present our proper contributions. In Sect. 5, we present two formalisms for languages for real-time concurrency: interval delay words and timed pomsets with interfaces, generalizing the dual view on languages of timed automata and of HDAs and showing their equivalence. Then in Sect. 6, we define languages of higher-dimensional timed automata using the formalisms previously introduced, and prove two main results: language inclusion is undecidable for higher-dimensional timed automata, but untimed language inclusion is decidable.

2 Timed automata and their languages

Timed automata extend finite automata with clock variables and invariants which permit the modeling of real-time properties. For a set C (of *clocks*), $\Phi(C)$ denotes the set of *clock constraints* defined as

$$\Phi(C) \ni \phi_1, \phi_2 ::= c \bowtie k \mid \phi_1 \wedge \phi_2 \qquad (c \in C, k \in \mathbb{N}, \bowtie \in \{<, \leq, \geq, >\}) .$$

Hence a clock constraint is a conjunction of comparisons of clocks to integers.

A *clock valuation* is a mapping $v : C \to \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ denotes the set of non-negative real numbers. The *initial* clock valuation is $v^0 : C \to \mathbb{R}_{\geq 0}$ given by $v^0(c) = 0$ for all $c \in C$. For $v \in \mathbb{R}_{\geq 0}^C$, $d \in \mathbb{R}_{\geq 0}$, and $R \subseteq C$, the clock valuations $v + d$ and $v[R \leftarrow 0]$ are defined by

$$(v + d)(c) = v(c) + d \qquad v[R \leftarrow 0](c) = \begin{cases} 0 & \text{if } c \in R, \\ v(c) & \text{if } c \notin R. \end{cases}$$

For $v \in \mathbb{R}_{\geq 0}^C$ and $\phi \in \Phi(C)$, we write $v \models \phi$ if v satisfies ϕ.

A *timed automaton* is a structure $(\Sigma, C, Q, \bot, \top, I, E)$, where Σ is a finite set (alphabet), C is a finite set of clocks, Q is a finite set of locations with initial and accepting locations $\bot, \top \subseteq Q$, $I : Q \to \Phi(C)$ assigns invariants to states, and $E \subseteq Q \times \Phi(C) \times \Sigma \times 2^C \times Q$ is a set of guarded transitions. We will often take the liberty to omit Σ and C from the signature of timed automata.

Timed automata have a long and successful history in the modeling and verification of real-time computing systems. Several tools exist such as Uppaal[3] [13, 14, 41], TChecker[4], IMITATOR[5] [6, 7], and Romeo[6] [35, 42], some of which are routinely applied in industry. The interested reader is referred to [1, 16, 40].

The *operational semantics* of a timed automaton $A = (Q, \bot, \top, I, E)$ is the (usually uncountably infinite) transition system $[\![A]\!] = (S, S^\bot, S^\top, \rightsquigarrow)$, with $\rightsquigarrow \subseteq$

[3] `https://uppaal.org/`
[4] `https://www.labri.fr/perso/herbrete/tchecker/`
[5] `https://www.imitator.fr/`
[6] `https://romeo.ls2n.fr/`

$S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$, given as follows:

$$\begin{aligned} S &= \{(q, v) \in Q \times \mathbb{R}^C_{\geq 0} \mid v \models I(q)\} \\ S^\perp &= \{(q, v^0) \mid q \in \perp\} \qquad S^\top = S \cap \top \times \mathbb{R}^C_{\geq 0} \\ \rightsquigarrow &= \{((q, v), d, (q, v + d)) \mid \forall 0 \leq d' \leq d : v + d' \models I(q)\} \\ &\quad \cup \{((q, v), a, (q', v')) \mid \exists (q, \phi, a, R, q') \in E : v \models \phi, v' = v[R \leftarrow 0]\} \end{aligned}$$

Tuples in $\rightsquigarrow$ of the first type are called *delay moves* and denoted $\rightsquigarrow^d$, tuples of the second kind are called *action moves* and denoted $\overset{a}{\rightsquigarrow}$.

The definition of $\rightsquigarrow$ ensures that actions are immediate: for any $(q, \phi, a, R, q') \in E$, then A passes from (q, v) to (q', v') without any delay. Time progresses only during delays $(q, v) \rightsquigarrow (q, v + d)$ in locations. A path π in $[\![A]\!]$ is a finite sequence of consecutive moves of $\rightsquigarrow$:

$$\pi = (l_0, v_0) \rightsquigarrow (l_1, v_1) \rightsquigarrow \cdots \rightsquigarrow (l_{n-1}, v_{n-1}) \rightsquigarrow (l_n, v_n) \tag{1}$$

It is accepting if $(l_0, v_0) \in S^\perp$ and $(l_n, v_n) \in S^\top$.

The *language semantics* of timed automata is defined in terms of timed words. There are two versions of these in the literature, and we will use them both. The first, which we call *delay words* here, is defined as follows. The label of a delay move $\delta = (q, v) \rightsquigarrow (q, v + d)$ is $\mathsf{ev}(\delta) = d$. That of an action move $\sigma = (l, v) \overset{a}{\rightsquigarrow} (l', v')$ is $\mathsf{ev}(\sigma) = a$. Finally, the label $\mathsf{ev}(\pi)$ of π as in (1) is

$$\mathsf{ev}((l_0, v_0) \rightsquigarrow (l_1, v_1)) \cdots \mathsf{ev}((l_{n-1}, v_{n-1}) \rightsquigarrow (l_n, v_n)).$$

Delay words are elements of the quotient of the free monoid on $\Sigma \cup \mathbb{R}_{\geq 0}$ by the equivalence relation $\sim$ which allows to add up subsequent delays and to remove zero delays. Formally, $\sim$ is the congruence on $(\Sigma \cup \mathbb{R}_{\geq 0})^*$ generated by the relations

$$dd' \sim d + d', \qquad 0 \sim \epsilon. \tag{2}$$

The *delay language* $\mathcal{L}(A)$ of the timed automaton A is the set of delay words labeling accepting paths in $[\![A]\!]$:

$$\mathcal{L}(A) = \{\mathsf{ev}(\pi) \mid \pi \text{ accepting path in } [\![A]\!]\} \subseteq (\Sigma \cup \mathbb{R}_{\geq 0})^*{}_{/\sim}$$

Any equivalence class of delay words has a unique representative of the form $d_0 a_0 d_1 a_1 \dots a_n d_{n+1}$ in which delays $d_i \in \mathbb{R}_{\geq 0}$ and symbols $a_i \in \Sigma$ alternate.

The second language semantics of timed automata is given using words with timestamps, which we will call *timed words* here. In the literature [1, 16, 40] these are usually defined as elements of the free monoid on $\Sigma \times \mathbb{R}_{\geq 0}$ in which the real components form an increasing sequence. Formally, this is the subset $\mathsf{TW}' \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^*$ given as

$$\mathsf{TW}' = \{w = (a_0, t_0) \dots (a_n, t_n) \mid \forall i = 0, \dots, n-1 : t_i \leq t_{i+1}\}.$$

The notions of delay words and timed words do not match completely, as delay words allow for a delay *at the end* of a run while timed words terminate

with the last timestamped symbol. In order for the language semantics to better match the operational semantics, we prefer to allow for these extra delays. Let $\mathsf{TW} \subseteq (\Sigma \times \mathbb{R}_{\geq 0})^* \, \mathbb{R}_{\geq 0}$ be the subset

$$\mathsf{TW} = \{w = (a_0, t_0) \dots (a_n, t_n) \, t_{n+1} \mid \forall i = 0, \dots, n : t_i \leq t_{i+1}\}.$$

Concatenation in TW is defined by shifting timestamps. For $w = (a_0, t_0) \dots (a_n, t_n) \, t_{n+1}$, $w' = (a'_0, t'_0) \dots (a'_n, t'_n) \, t'_{n+1} \in \mathsf{TW}$:

$$ww' = (a_0, t_0) \dots (a_n, t_n)(a'_0, t_{n+1} + t'_0) \dots (a'_n, t_{n+1} + t'_n)(t_{n+1} + t'_{n+1}).$$

The monoids $(\Sigma \cup \mathbb{R}_{\geq 0})^*/_\sim$ and TW are then isomorphic via the mapping

$$\begin{aligned} &d_0 a_0 d_1 a_1 \dots a_n d_{n+1} \mapsto \\ &\qquad (a_0, d_0) \, (a_1, d_0 + d_1) \dots (a_{n-1}, d_0 + \dots + d_n) \, (d_0 + \dots + d_{n+1}), \end{aligned}$$

and the *timed language* of a timed automaton A is the image of its delay language $\mathcal{L}(A)$ under this isomorphism.

3 Higher-dimensional automata and their languages

Higher-dimensional automata (HDAs) extend finite automata with extra structure which permits to specify independence or concurrency of events. We focus in this section on the *languages* of HDAs and refer to [26, 32] for more details.

3.1 Higher-dimensional automata

An HDA is a set X of *cells* which are connected by *face maps*. Each cell has a list of events which are active, and face maps permit to pass from a cell to another in which some events have not yet started or are terminated.

We make this precise. A *conclist* (*concurrency list*) over a finite alphabet Σ is a tuple $U = (U, \dashrightarrow, \lambda)$, consisting of a finite set U (of events), a strict total order $\dashrightarrow \subseteq U \times U$ (the event order),[7] and a labeling $\lambda : U \to \Sigma$. Let $\square$ denote the set of conclists over Σ.

A *precubical set* on a finite alphabet Σ,

$$X = (X, \mathsf{ev}, \{\delta^0_{A,U}, \delta^1_{A,U} \mid U \in \square, A \subseteq U\}),$$

consists of a set of cells X together with a function $\mathsf{ev} : X \to \square$. For $U \in \square$ we write $X[U] = \{x \in X \mid \mathsf{ev}(x) = U\}$. Further, for every $U \in \square$ and $A \subseteq U$ there are face maps $\delta^0_{A,U}, \delta^1_{A,U} : X[U] \to X[U \setminus A]$ which satisfy

$$\delta^\nu_{A,U} \delta^\mu_{B,U\setminus A} = \delta^\mu_{B,U} \delta^\nu_{A,U\setminus B} \tag{3}$$

for $A \cap B = \emptyset$ and $\nu, \mu \in \{0, 1\}$.[8] The upper face maps δ^1_A transform a cell x into one in which the events in A have terminated; the lower face maps δ^0_A transform

[7] A strict *partial* order is a relation which is irreflexive and transitive; a strict *total* order is a relation which is irreflexive, transitive, and total. We may omit the "strict".

[8] We will omit the extra subscript "U" in $\delta^\nu_{A,U}$ from here on.

x into a cell where the events in A have not yet started. (3) expresses the fact that these transformations commute for disjoint sets of events.

A *higher-dimensional automaton* (*HDA*) $A = (\Sigma, X, \bot, \top)$ consists of a finite alphabet Σ, a finite precubical set X on Σ, and subsets $\bot, \top \subseteq X$ of initial and accepting cells. The *dimension* of A is $\dim(A) = \max\{|\mathsf{ev}(x)| \mid x \in X\}$.

3.2 Pomsets with interfaces

The *language semantics* of HDAs is defined in terms of ipomsets which we define now; see again [26, 32] for more details. First, a *partially ordered multiset*, or *pomset*, over a finite alphabet Σ is a structure $P = (P, <, \dashrightarrow, \lambda)$ consisting of a finite set P, two strict partial orders $<, \dashrightarrow \subseteq P \times P$ (precedence and event order), and a labeling $\lambda : P \to \Sigma$, such that for each $x \neq y$ in P, at least one of $x < y$, $y < x$, $x \dashrightarrow y$, or $y \dashrightarrow x$ holds.[9]

A *pomset with interfaces*, or *ipomset*, over a finite alphabet Σ is a tuple $(P, <, \dashrightarrow, S, T, \lambda)$ consisting of a pomset $(P, <, \dashrightarrow, \lambda)$ and subsets $S, T \subseteq P$ of *source* and *target interfaces* such that the elements of S are $<$-minimal and those of T are $<$-maximal. Note that different events of ipomsets may carry the same label; in particular we do *not* exclude autoconcurrency. Source and target events are marked by "$\bullet$" at the left or right side, and if the event order is not shown, we assume that it goes downwards.

An ipomset P is *interval* if its precedence order $<_P$ is an interval order [33], that is, if it admits an interval representation given by functions $\sigma^-, \sigma^+ : P \to \mathbb{R}$ such that $\sigma^-(x) \leq \sigma^+(x)$ for all $x \in P$ and $x <_P y$ iff $\sigma^+(x) < \sigma^-(y)$ for all $x, y \in P$. We will only use interval ipomsets here and hence omit the qualification "interval". The set of (interval) ipomsets over Σ is denoted iPoms.

For ipomsets P and Q we say that Q *subsumes* P and write $P \sqsubseteq Q$ if there is a bijection $f : P \to Q$ for which

(1) $f(S_P) = S_Q$, $f(T_P) = T_Q$, and $\lambda_Q \circ f = \lambda_P$;
(2) $f(x) <_Q f(y)$ implies $x <_P y$;
(3) $x \not<_P y$, $y \not<_P x$ and $x \dashrightarrow_P y$ imply $f(x) \dashrightarrow_Q f(y)$.

That is, f respects interfaces and labels, reflects precedence, and preserves essential event order. (Event order is essential for concurrent events, but by transitivity, it also appears between non-concurrent events. Subsumptions ignore such non-essential event order.)

Isomorphisms of ipomsets are invertible subsumptions, *i.e.*, bijections f for which items (2) and (3) above are strengthened to

(2′) $f(x) <_Q f(y)$ iff $x <_P y$;
(3′) $x \not<_P y$ and $y \not<_P x$ imply that $x \dashrightarrow_P y$ iff $f(x) \dashrightarrow_Q f(y)$.

Due to the requirement that all elements are ordered by $<$ or $\dashrightarrow$, there is at most one isomorphism between any two ipomsets. Hence we may switch freely between ipomsets and their isomorphism classes. We will also call these equivalence classes ipomsets and often conflate equality and isomorphism.

[9] The event order is needed to identify concurrent events, see [26, 32] for details.

Serial composition of pomsets [37] generalises to a *gluing* composition for ipomsets which continues interface events across compositions and is defined as follows. Let P and Q be ipomsets such that $T_P = S_Q$, $x \dashrightarrow_P y$ iff $x \dashrightarrow_Q y$ for all $x, y \in T_P = S_Q$, and the restrictions $\lambda_{P|T_P} = \lambda_{Q|S_Q}$, then $P * Q = (P \cup Q, <, \dashrightarrow, S_P, T_Q, \lambda)$, where

- $x < y$ if $x <_P y$, $x <_Q y$, or $x \in P \setminus T_P$ and $y \in Q \setminus S_Q$;
- $\dashrightarrow$ is the transitive closure of $\dashrightarrow_P \cup \dashrightarrow_Q$;
- $\lambda(x) = \lambda_P(x)$ if $x \in P$ and $\lambda(x) = \lambda_Q(x)$ if $x \in Q$.

Gluing is, thus, only defined if the targets of P are equal to the sources of Q *as conclists*.

An ipomset P is a *word* (with interfaces) if $<_P$ is total. Conversely, P is *discrete* if $<_P$ is empty (hence $\dashrightarrow_P$ is total). Conclists are discrete ipomsets without interfaces. A *starter* is a discrete ipomset U with $T_U = U$, a *terminator* one with $S_U = U$. The intuition is that a starter does nothing but start the events in $A = U - S_U$, and a terminator terminates the events in $B = U - T_U$. These will be so important later that we introduce special notation, writing ${}_A{\uparrow}U$ and $U{\downarrow}_B$ for the above. Discrete ipomsets U with $S_U = T_U = U$ are identities for the gluing composition and written id_U. Note that $\mathrm{id}_U = {}_\emptyset{\uparrow}U = U{\downarrow}_\emptyset$. The empty ipomset is $\mathrm{id}_\emptyset$.

3.3 Step sequences

Any ipomset can be decomposed as a gluing of starters and terminators [28,39]. Such a presentation is called a *step decomposition*. If starters and terminators are alternating, the step decomposition is called *sparse*. [32, Prop. 4] shows that every ipomset has a unique sparse step decomposition.

We develop an algebra of step decompositions. Let $\mathsf{St}, \mathsf{Te}, \mathsf{Id} \subseteq \mathsf{iPoms}$ denote the sets of starters, terminators, and identities over Σ, then $\mathsf{Id} = \mathsf{St} \cap \mathsf{Te}$. Let $\mathsf{St}_+ = \mathsf{St} \setminus \mathsf{Id}$ and $\mathsf{Te}_+ = \mathsf{Te} \setminus \mathsf{Id}$. The following notion was introduced in [5].

Definition 1. *A word* $P_1 \dots P_n \in (\mathsf{St} \cup \mathsf{Te})^*$ *is* coherent *if the gluing* $P_1 * \dots * P_n$ *is defined.*

Let $\mathsf{Coh} \subseteq (\mathsf{St} \cup \mathsf{Te})^*$ denote the subset of coherent words and $\sim$ the congruence on Coh generated by the relations

$$\begin{gathered} I \sim \epsilon \quad (I \in \mathsf{Id}), \\ S_1 S_2 \sim S_1 * S_2 \quad (S_1, S_2 \in \mathsf{St}), \qquad T_1 T_2 \sim T_1 * T_2 \quad (T_1, T_2 \in \mathsf{Te}). \end{gathered} \tag{4}$$

Here, ϵ denotes the empty *word* in $(\mathsf{St} \cup \mathsf{Te})^*$, not the empty ipomset $\mathrm{id}_\emptyset \in \mathsf{Id}$.

Definition 2. *A* step sequence *is an element of the set* $\mathsf{SSeq} = \mathsf{Id}.\mathsf{Coh}_{/\sim}.\mathsf{Id}$.

The identities in the beginning and end of step sequences are used to "fix" events in the source and target interfaces, *i.e.*, which are already running in the beginning or continue beyond the end. For example, $\bullet a \bullet . \bullet a \bullet$ denotes an event labeled a which is neither started nor terminated. Technically we only need these

identities when the inner part (in $(\mathsf{St} \cup \mathsf{Te})^*{}_{/\sim}$) is empty (and even then we would only need one of them); but we prefer a more verbose notation that will be of interest when we introduce time, see Def. 12.

Lemma 3. *Every element of* SSeq *has a unique representative* $I_0 P_1 \dots P_n I_{n+1}$*, for* $n \geq 0$*, with the property that* $(P_i, P_{i+1}) \in \mathsf{St}_+ \times \mathsf{Te}_+ \cup \mathsf{Te}_+ \times \mathsf{St}_+$ *for all* $1 \leq i \leq n-1$*. Such a representative is called* sparse*.*

Proof. Directly from [32, Prop. 4].

Concatenation of step sequences is inherited from the monoid $(\mathsf{St} \cup \mathsf{Te})^*$ where $I_0.w.I_n \cdot I'_0.w'.I'_m$ is defined iff $I_n = I'_0$. Concatenations of sparse step sequences may not be sparse.

Remark 4. SSeq forms a *local partial monoid* [29] with left and right units $\mathrm{id}_U \mathrm{id}_U$, for $U \in \square$, and $(UV)W = U(VW)$ when the concatenation is defined. Local partial monoids may admit many units and are equivalent to categories: here, the objects are the conclists in $\square$ and the morphisms from $U \in \square$ to $V \in \square$ are the step sequences $\mathrm{id}_U w \,\mathrm{id}_V$. We refer to [29] for more details.

For a coherent word $P_1 \dots P_n \in \mathsf{Coh} \subseteq (\mathsf{St} \cup \mathsf{Te})^*$ we define $\mathsf{Glue}(P_1 \dots P_n) = P_1 * \dots * P_n$. It is clear that for $w_1, w_2 \neq \epsilon$, $w_1 \sim w_2$ implies $\mathsf{Glue}(w_1) = \mathsf{Glue}(w_2)$. That is, Glue induces a mapping $\mathsf{Glue} : \mathsf{SSeq} \to \mathsf{iPoms}$.

Lemma 5 ([32, Prop. 4]). $\mathsf{Glue} : \mathsf{SSeq} \to \mathsf{iPoms}$ *is a bijection.*

See below for examples of step sequences and ipomsets.

3.4 Languages of HDAs

Paths in an HDA X are sequences $\alpha = (x_0, \phi_1, x_1, \dots, x_{n-1}, \phi_n, x_n)$ consisting of cells x_i of X and symbols ϕ_i which indicate face map types: for every $i \in \{1, \dots, n\}$, (x_{i-1}, ϕ_i, x_i) is either

- $(\delta^0_A(x_i), \nearrow^A, x_i)$ for $A \subseteq \mathsf{ev}(x_i)$ (an *upstep*)
- or $(x_{i-1}, \searrow_A, \delta^1_A(x_{i-1}))$ for $A \subseteq \mathsf{ev}(x_{i-1})$ (a *downstep*).

Downsteps terminate events, following upper face maps, whereas upsteps start events by following inverses of lower face maps.

The *source* and *target* of α as above are $\mathsf{src}(\alpha) = x_0$ and $\mathsf{tgt}(\alpha) = x_n$. A path α is *accepting* if $\mathsf{src}(\alpha) \in \bot_X$ and $\mathsf{tgt}(\alpha) \in \top_X$. Paths α and β may be concatenated if $\mathsf{tgt}(\alpha) = \mathsf{src}(\beta)$. Their concatenation is written $\alpha * \beta$ or simply $\alpha\beta$.

The observable content or *event ipomset* $\mathsf{ev}(\alpha)$ of a path α is defined recursively as follows:

- if $\alpha = (x)$, then $\mathsf{ev}(\alpha) = \mathrm{id}_{\mathsf{ev}(x)}$;
- if $\alpha = (y \nearrow^A x)$, then $\mathsf{ev}(\alpha) = {}_A\!\uparrow \mathsf{ev}(x)$;
- if $\alpha = (x \searrow_A y)$, then $\mathsf{ev}(\alpha) = \mathsf{ev}(x) \downarrow_A$;
- if $\alpha = \alpha_1 * \dots * \alpha_n$ is a concatenation, then $\mathsf{ev}(\alpha) = \mathsf{ev}(\alpha_1) * \dots * \mathsf{ev}(\alpha_n)$.

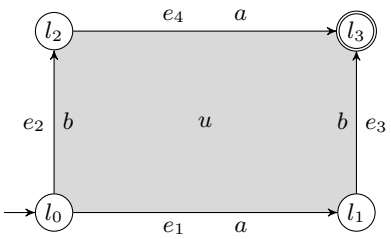

Fig. 3. HDA of Example 7. The grayed area indicates that a and b may occur concurrently, *i.e.*, there is a two-dimensional cell, u in this instance

Note that upsteps in α correspond to starters in $\mathsf{ev}(\alpha)$ and downsteps correspond to terminators.

For $A \subseteq \mathsf{iPoms}$, $A{\downarrow} = \{P \in \mathsf{iPoms} \mid \exists Q \in A\colon P \sqsubseteq Q\}$ denotes its subsumption closure. The language of an HDA X is $\mathcal{L}(X) = \{\mathsf{ev}(\alpha) \mid \alpha \text{ accepting path in } X\}$. A language is *regular* if it is the language of a finite HDA. It is *rational* if it is constructed from $\emptyset$, $\{\mathrm{id}_\emptyset\}$ and discrete ipomsets using $\cup$, $*$ and $^+$ (Kleene plus) [27]. Languages of HDAs are closed under subsumption, that is, if L is regular, then $L{\downarrow} = L$ [26, 27]. The rational operations above have to take this closure into account.

Theorem 6 ([27]). *A language is regular if and only if it is rational.*

Example 7. The HDA of Fig. 3 is two-dimensional and consists of nine cells: the corner cells $X_0 = \{l_0, l_1, l_2, l_3\}$ in which no event is active (for all $z \in X_0$, $\mathsf{ev}(z) = \emptyset$), the transition cells $X_1 = \{e_1, e_2, e_3, e_4\}$ in which one event is active ($\mathsf{ev}(e_1) = \mathsf{ev}(e_4) = a$ and $\mathsf{ev}(e_2) = \mathsf{ev}(e_3) = b$), and the square cell u where a and b are active: $\mathsf{ev}(u) = \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$. When we have two concurrent events a and b with $a \dashrightarrow b$, we will draw a horizontally and b vertically. Concerning face maps, we have for example $\delta^1_{ab}(u) = l_3$ and $\delta^0_{ab}(u) = l_0$.

This HDA admits several accepting paths, for example

$$\alpha_1 = l_0 \nearrow^{ab} u \searrow_{ab} l_3, \qquad \alpha_2 = l_0 \nearrow^{a} e_1 \nearrow^{b} u \searrow_{b} e_4 \searrow_{a} l_3,$$
$$\alpha_3 = l_0 \nearrow^{a} e_1 \searrow_{a} l_1 \nearrow^{b} e_3 \searrow_{b} l_3, \qquad \alpha_4 = l_0 \nearrow^{b} e_2 \searrow_{b} l_2 \nearrow^{a} e_4 \searrow_{b} l_3,$$

where $\mathsf{ev}(\alpha_1) = \left[\begin{smallmatrix} a\bullet \\ b\bullet \end{smallmatrix}\right] * \left[\begin{smallmatrix} \bullet a \\ \bullet b \end{smallmatrix}\right] = \mathsf{ev}(\alpha_2) = a\bullet * \left[\begin{smallmatrix} \bullet a\bullet \\ b\bullet \end{smallmatrix}\right] * \left[\begin{smallmatrix} \bullet a\bullet \\ \bullet b \end{smallmatrix}\right] * \bullet a = \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$, $\mathsf{ev}(\alpha_3) = a\bullet * \bullet a * b\bullet * \bullet b = ab$, and $\mathsf{ev}(\alpha_4) = b\bullet * \bullet b * a\bullet * \bullet a = ba$. Its language is $\{\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]\}{\downarrow} = \{\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right], ab, ba\}$.

Observe that α_1 and α_2 induce the coherent words $w_1 = \left[\begin{smallmatrix} a\bullet \\ b\bullet \end{smallmatrix}\right]\left[\begin{smallmatrix} \bullet a \\ \bullet b \end{smallmatrix}\right]$ and $w_2 = a\bullet \left[\begin{smallmatrix} \bullet a\bullet \\ b\bullet \end{smallmatrix}\right]\left[\begin{smallmatrix} \bullet a \\ \bullet b\bullet \end{smallmatrix}\right] \bullet b$ such that $w_1 \sim w_2$ and $s = \mathrm{id}_\emptyset w_1 \mathrm{id}_\emptyset$ is their corresponding sparse step sequence with $\mathsf{Glue}(s) = \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$.

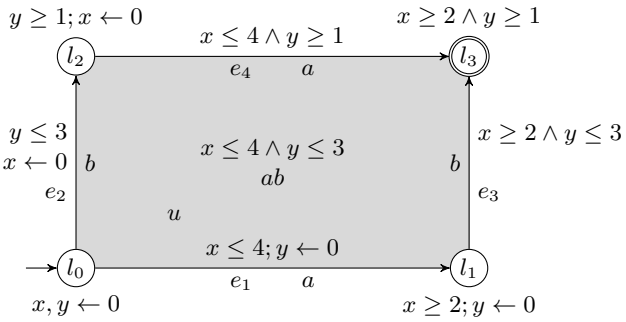

Fig. 4. HDTA of Example 9.

4 Higher-dimensional timed automata

Unlike timed automata, higher-dimensional automata make no formal distinction between states (0-cells), transitions (1-cells), and higher-dimensional cells. We transfer this intuition to higher-dimensional timed automata, so that each cell has an invariant which specifies when it is enabled and an exit condition giving the clocks to be reset when leaving. Semantically, this implies that time delays can occur in any n-cell, not only in states as in timed automata; hence actions are no longer instantaneous.

Definition 8. *A* higher-dimensional timed automaton (HDTA) *is a structure* $(\Sigma, C, Q, \bot, \top, \mathsf{inv}, \mathsf{exit})$*, where* $(\Sigma, Q, \bot, \top)$ *is a finite higher-dimensional automaton and* $\mathsf{inv} : Q \to \Phi(C)$*,* $\mathsf{exit} : Q \to 2^C$ *assign* invariant *and* exit *conditions to each cell of* Q*.*

As before, we will often omit Σ and C from the signature.

The *operational semantics* of an HDTA $A = (Q, \bot, \top, \mathsf{inv}, \mathsf{exit})$ is a (usually uncountably infinite) transition system $[\![A]\!] = (S, S^\bot, S^\top, \rightsquigarrow)$, with the set of transitions (moves) $\rightsquigarrow \subseteq S \times (\mathsf{St} \cup \mathsf{Te} \cup \mathbb{R}_{\geq 0}) \times S$ given as follows:

$$
\begin{aligned}
S &= \{(q, v) \in Q \times \mathbb{R}^C_{\geq 0} \mid v \models \mathsf{inv}(q)\} \\
S^\bot &= \{(q, v^0) \mid q \in \bot\} \qquad S^\top = S \cap \top \times \mathbb{R}^C_{\geq 0} \\
\rightsquigarrow &= \{((q, v), d, (q, v + d)) \mid \forall 0 \leq d' \leq d : v + d' \models \mathsf{inv}(q)\} \\
&\quad \cup \{((\delta^0_A(q), v), {}_A{\uparrow}\mathsf{ev}(q), (q, v')) \mid A \subseteq \mathsf{ev}(q), v' = v[\mathsf{exit}(\delta^0_A(q)) \leftarrow 0] \models \mathsf{inv}(q)\} \\
&\quad \cup \{((q, v), \mathsf{ev}(q){\downarrow_A}, (\delta^1_A(q), v')) \mid A \subseteq \mathsf{ev}(q), v' = v[\mathsf{exit}(q) \leftarrow 0] \models \mathsf{inv}(\delta^1_A(q))\}
\end{aligned}
$$

Note that in the first line of the definition of $\rightsquigarrow$ above, we allow time to evolve in any cell of Q. As before, these are called *delay moves* and denoted $\rightsquigarrow^d$ for some delay $d \in \mathbb{R}_{\geq 0}$. The second line in the definition of $\rightsquigarrow$ defines the start of concurrent events A (denoted $\rightsquigarrow^A$) and the third line describes what happens when finishing a set A of concurrent events (denoted $\rightsquigarrow_A$). These are again called *action moves*. Exit conditions specify which clocks to reset when leaving a cell.

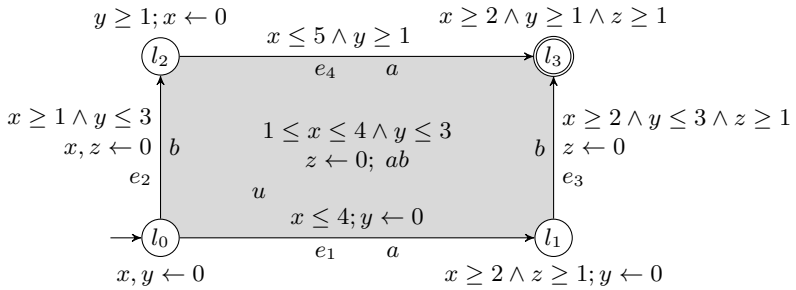

Fig. 5. HDTA of Example 10

Example 9. We give a few examples of two-dimensional timed automata. The first, in Fig. 4, is the HDA of Fig. 3 with time constraints. It models two actions, a and b, which can be performed concurrently. This HDTA models that performing a takes between two and four time units, whereas performing b takes between one and three time units. To this end, we use two clocks x and y which are reset when the respective actions are started and then keep track of how long they are running.

Hence $\mathsf{exit}(l_0) = \{x, y\}$, and the invariants $x \leq 4$ at the a-labeled transitions e_1, e_4 and at the square u ensure that a takes at most four time units. The invariants $x \geq 2$ at l_1, e_3 and l_3 take care that a cannot finish before two time units have passed. Note that x is also reset when exiting e_2 and l_2, ensuring that regardless when a is started, whether before b, while b is running, or after b is terminated, it must take between two and four time units.

Example 10. The HDTA in Fig. 5 models the following additional constraints:

- b may only start after a has been running for one time unit;
- once b has terminated, a may run one time unit longer;
- and b must finish one time unit before a.

To this end, an invariant $x \geq 1$ has been added to the two b-labeled transitions and to the ab-square (at the right-most b-transition $x \geq 1$ is already implied), and the condition on x at the top a-transition has been changed to $x \leq 5$. To enforce the last condition, an extra clock z is introduced which is reset when b terminates and must be at least 1 when a is terminating.

Note that the left edge is now unreachable: when entering it, x is reset to zero, but its edge invariant is $x \geq 1$. This is as expected, as b should not be able to start before a. Further, the right b-labeled edge is deadlocked: when leaving it, z is reset to zero but needs to be at least one when entering the accepting state. Again, this is expected, as a should not terminate before b. As both vertical edges are now permanently disabled, the accepting state can only be reached through the square.

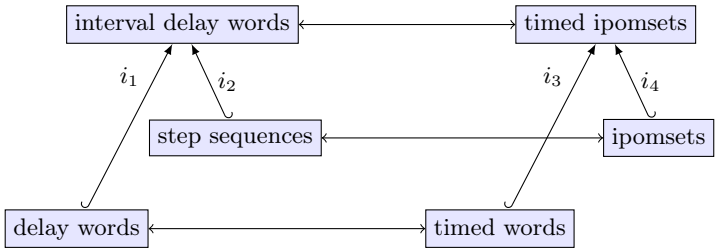

Fig. 6. Different types of language semantics: below, for languages of timed automata; middle, for languages of HDAs; top, for languages of HDTAs. Vertical arrows denote injections, horizontal arrows bijections.

5 Concurrent timed languages

We introduce two formalisms for concurrent timed words: interval delay words which generalize delay words and step sequences, and timed ipomsets which generalize timed words and ipomsets. Figure 6 shows the relations between the different language semantics used and introduced in this paper.

5.1 Interval delay words

Intuitively, an interval delay word is a step sequence interspersed with delays. These delays indicate how much time passes between starts and terminations of different events.

Definition 11. *A word* $x_1 \dots x_n \in (\mathsf{St} \cup \mathsf{Te} \cup \mathbb{R}_{\geq 0})^*$ *is* coherent *if, for all* $i < k$ *such that* $x_i, x_k \in \mathsf{St} \cup \mathsf{Te}$ *and* $\forall i < j < k : x_j \in \mathbb{R}_{\geq 0}$*, the gluing* $x_i * x_k$ *is defined.*

Let $\mathsf{tCoh} \subseteq (\mathsf{St} \cup \mathsf{Te} \cup \mathbb{R}_{\geq 0})^*$ denote the subset of coherent words. Let $\sim$ be the congruence on tCoh generated by the relations

$$dd' \sim d + d', \qquad 0 \sim \epsilon, \qquad I \sim \epsilon \quad (I \in \mathsf{Id}),$$
$$S_1 S_2 \sim S_1 * S_2 \quad (S_1, S_2 \in \mathsf{St}), \qquad T_1 T_2 \sim T_1 * T_2 \quad (T_1, T_2 \in \mathsf{Te}).$$

Again, ϵ denotes the empty word in $(\mathsf{St} \cup \mathsf{Te} \cup \mathbb{R}_{\geq 0})^*$ above. That is, successive delays may be added up and zero delays removed, as may identities, and successive starters or terminators may be composed. Note how this combines the identifications in (2) for delay words and the ones for step sequences in (4).

Definition 12. *An* interval delay word (idword) *is an element of the set*

$$\mathsf{IDW} = \mathsf{Id}.\mathsf{tCoh}_{/\sim}.\mathsf{Id}.$$

Lemma 13. *Every element of* IDW *has a unique representative* $I_0 d_0 P_1 d_1 \dots P_n d_n I_{n+1}$*, for* $n \geq 0$*, with the property that for all* $1 \leq i \leq n$*,* $P_i \notin \mathsf{Id}$ *and for all* $1 \leq i \leq n-1$*, if* $d_i = 0$*, then* $(P_i, P_{i+1}) \in \mathsf{St}_+ \times \mathsf{Te}_+ \cup \mathsf{Te}_+ \times \mathsf{St}_+$*. Such a representative is called* sparse*.*

Fig. 7. Tipomsets T_1 (left) and T_2 (right) of Ex. 15

This is analogous to Lem. 3, except that here, we must admit successive starters or terminators if they are separated by non-zero delays (see Ex. 21 below for an example).

With concatenation of idwords inherited from the monoid $(\mathsf{St} \cup \mathsf{Te} \cup \mathbb{R}_{\geq 0})^*$, IDW forms a partial monoid (successive identities are composed using $\sim$). Concatenations of sparse idwords are not generally sparse. The identities for concatenation are the words $\mathrm{id}_U \mathrm{id}_U \sim \mathrm{id}_U\, 0\, \mathrm{id}_U$ for $U \in \square$.

5.2 Timed ipomsets

Timed ipomsets are ipomsets with timestamps which mark beginnings and ends of events:

Definition 14. *Let P be a set, $\sigma^-, \sigma^+ : P \to \mathbb{R}_{\geq 0}$, $\sigma = (\sigma^-, \sigma^+)$, and $d \in \mathbb{R}_{\geq 0}$. Then $P = (P, <_P, \dashrightarrow, S, T, \lambda, \sigma, d)$ is a* timed ipomset (tipomset) *if*

- *$(P, <_P, \dashrightarrow, S, T, \lambda)$ is an ipomset,*
- *for all $x \in P$, $0 \leq \sigma^-(x) \leq \sigma^+(x) \leq d$,*
- *for all $x \in S$, $\sigma^-(x) = 0$,*
- *for all $x \in T$, $\sigma^+(x) = d$, and*
- *for all $x, y \in P$, $\sigma^+(x) < \sigma^-(y) \implies x <_P y \implies \sigma^+(x) \leq \sigma^-(y)$.*

The *activity interval* of event $x \in P$ is $\sigma(x) = [\sigma^-(x), \sigma^+(x)]$; we will always write $\sigma(x)$ using square brackets because of this. The *untiming* of P is its underlying ipomset, *i.e.*, $\mathrm{unt}(P) = (P, <_P, \dashrightarrow, S, T, \lambda)$. We will often write tipomsets as (P, σ, d) or just P.

Example 15. Figure 7 depicts the following tipomsets:

- $T_1 = (\{x_1, x_2, x_3\}, <_1, \dashrightarrow, \{x_1, x_3\}, \{x_1\}, \lambda_1, \sigma_1, 3)$ with
  - $<_1 = \{(x_3, x_2)\}$, $\dashrightarrow = \{(x_1, x_2), (x_1, x_3)\}$,
  - $\lambda_1(x_1) = a$, $\lambda_1(x_2) = d$, $\lambda_1(x_3) = c$, and
  - $\sigma_1(x_1) = [0, 3], \sigma_1(x_2) = [1.5, 3], \sigma_1(x_3) = [0, 1.5]$
- $T_2 = (\{x_4, x_5, x_6\}, <_2, x_4 \dashrightarrow x_5 \dashrightarrow x_6, \{x_4\}, \emptyset, \lambda_2, \sigma_2, 4)$ with
  - $<_2 = \emptyset$, $\lambda_2(x_4) = a$, $\lambda_2(x_5) = b$, $\lambda_2(x_6) = c$, and
  - $\sigma_2(x_4) = [0, 2], \sigma_2(x_5) = [0.5, 3.5], \sigma_2(x_6) = [1, 3]$.

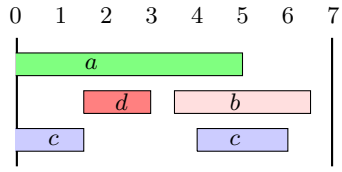

Fig. 8. Gluing $T_1 * T_2$, see Ex. 17

Note that in T_1, the d-labeled event x_2 is not in the terminating interface as it ends exactly at time 3. Further, the precedence order is *not* induced by the timestamps in T_1: we have $\sigma_1^+(x_3) = \sigma_1^-(x_2)$ but $x_3 <_1 x_2$; setting $<_1 = \emptyset$ instead would *also* be consistent with the timestamps. For the underlying ipomsets,

$$\mathrm{unt}(T_1) = \begin{bmatrix} \bullet a \bullet \\ \bullet c \to d \end{bmatrix}, \qquad \mathrm{unt}(T_2) = \begin{bmatrix} \bullet a \\ b \\ c \end{bmatrix}.$$

We generalize the gluing composition of ipomsets to tipomsets.

Definition 16. *Given two tipomsets (P, σ_P, d_P) and (Q, σ_Q, d_Q), the gluing composition $P * Q$ is defined if $\mathrm{unt}(P) * \mathrm{unt}(Q)$ is. Then, $P * Q = (U, \sigma_U, d_U)$, where*

- *$U = P * Q$ and $d_U = d_P + d_Q$,*
- *$\sigma_U^-(x) = \sigma_P^-(x)$ if $x \in P$ and $\sigma_U^-(x) = \sigma_Q^-(x) + d_P$ else,*
- *$\sigma_U^+(x) = \sigma_Q^+(x) + d_P$ if $x \in Q$ and $\sigma_U(x) = \sigma_P^+(x)$ else.*

The above definition is consistent for events $x \in T_P = S_Q$: here, $\sigma_U^-(x) = \sigma_P^-(x)$ and $\sigma_U^+(x) = \sigma_Q^+(x) + d_P$.

Example 17. Continuing Ex. 15, Fig. 8 depicts the gluing of T_1 and T_2, which is the tipomset $T = (\{x_1, x_2, x_3, x_5, x_6\}, <, \dashrightarrow, \{x_1, x_3\}, \emptyset, \lambda, \sigma, 7)$ with

- $< = \{(x_3, x_2), (x_2, x_5), (x_3, x_5), (x_2, x_6), (x_3, x_6)\}$,
- $\dashrightarrow = \{(x_1, x_2), (x_1, x_3), (x_1, x_5), (x_5, x_6), (x_1, x_6)\}$,
- $\lambda(x_1) = a$, $\lambda(x_2) = d$, $\lambda(x_3) = c$, $\lambda(x_5) = b$, $\lambda(x_6) = c$,
- $\sigma(x_1) = [0, 5]$, $\sigma(x_2) = [1.5, 3]$, $\sigma(x_3) = [0, 1.5]$, $\sigma(x_5) = [3.5, 6.5]$, and $\sigma(x_6) = [4, 6]$.

In this example, events x_1 and x_4 have been glued together. Thus

$$\mathrm{unt}(T) = \mathrm{unt}(T_1) * \mathrm{unt}(T_2) = \begin{bmatrix} \bullet a \\ \bullet c \longrightarrow d \longrightarrow b \\ \qquad\qquad\quad\; c \end{bmatrix}.$$

Here, dashed arrows indicate event order, and full arrows indicate precedence order.

The next lemma, whose proof is trivial, shows that untiming respects gluing composition.

Lemma 18. *For all tipomsets P and Q, $P * Q$ is defined iff* $\mathrm{unt}(P) * \mathrm{unt}(Q)$ *is, and in that case,* $\mathrm{unt}(P) * \mathrm{unt}(Q) = \mathrm{unt}(P * Q)$. □

Definition 19. *An* isomorphism *of tipomsets (P, σ_P, d_P) and (Q, σ_Q, d_Q) is an ipomset isomorphism $f : P \to Q$ for which $\sigma_P = \sigma_Q \circ f$ and $d_P = d_Q$.*

In other words, two tipomsets are isomorphic if they share the same activity intervals, durations, precedence order, interfaces, and essential event order. As for (untimed) ipomsets, isomorphisms between tipomsets are unique, hence we may switch freely between tipomsets and their isomorphism classes.

Remark 20. Analogously to ipomsets, one could define a notion of subsumption for tipomsets such that isomorphisms would be invertible subsumptions. We refrain from doing this here, mostly because we have not seen any need for it. Note that as per Ex. 27 below, untimings of HDTA languages are not closed under subsumption.

5.3 Translations

We now provide the translations shown in Fig. 6. First, the bijection between idwords and tipomsets. Let $I_0 d_0 P_1 d_1 \dots P_n d_n I_{n+1}$ be an interval delay word in sparse normal form. Define the ipomset $P = P_1 * \dots * P_n$ and let $d_P = \sum_{i=0}^{n} d_i$. In order to define the activity intervals, let $x \in P$ and denote

$$\mathsf{first}(x) = \min\{i \mid x \in P_i \vee x \in I_i\}, \qquad \mathsf{last}(x) = \max\{i \mid x \in P_i \vee x \in I_i\}.$$

If $\mathsf{first}(x) = 0$, then let $\sigma^-(x) = 0$, otherwise, $\sigma^-(x) = \sum_{i=0}^{\mathsf{first}(x)-1} d_i$. Similarly, if $\mathsf{last}(x) = n + 1$, then let $\sigma^+(x) = d_P$; otherwise, $\sigma^+(x) = \sum_{i=0}^{\mathsf{last}(x)-1} d_i$. Using Lem. 13, this defines a mapping tGlue from idwords to tipomsets.

Example 21. Tipomset T of Ex. 17 is the translation of the following sparse interval delay word:

$$\left[\begin{smallmatrix}\bullet a\bullet\\ \bullet c\bullet\end{smallmatrix}\right] 1.5 \left[\begin{smallmatrix}\bullet a\bullet\\ \bullet c\end{smallmatrix}\right] 0 \left[\begin{smallmatrix}\bullet a\bullet\\ d\bullet\end{smallmatrix}\right] 1.5 \left[\begin{smallmatrix}\bullet a\bullet\\ \bullet d\end{smallmatrix}\right] 0.5 \left[\begin{smallmatrix}\bullet a\bullet\\ \bullet b\bullet\end{smallmatrix}\right] 0.5 \begin{bmatrix}\bullet a\bullet\\ \bullet b\bullet\\ c\bullet\end{bmatrix} 1 \begin{bmatrix}\bullet a\\ \bullet b\bullet\\ \bullet c\bullet\end{bmatrix} 1 \left[\begin{smallmatrix}\bullet b\bullet\\ \bullet c\end{smallmatrix}\right] 0.5 \bullet b 0.5\ \mathrm{id}_\emptyset$$

Lemma 22. *The mapping* tGlue *is a bijection between idwords and tipomsets.*

Lemma 23. *The vertical mappings $i_1, \dots, i_4$ of Fig. 6 are injective and commute with the horizontal bijections.*

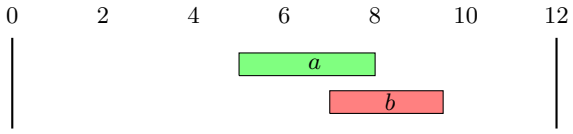

Fig. 9. Tipomset of accepting path in HDTA of Ex. 27

6 Languages of HDTAs

We are now ready to introduce languages of HDTAs as sets of timed ipomsets. Let $A = (\Sigma, C, L, \bot, \top, \mathsf{inv}, \mathsf{exit})$ be an HDTA and $[\![A]\!] = (S, S^\bot, S^\top, \rightsquigarrow)$. A *path* π in $[\![A]\!]$ is a finite sequence of consecutive moves $s_1 \rightsquigarrow s_2 \rightsquigarrow \cdots \rightsquigarrow s_n$, where each $s_i \rightsquigarrow s_{i+1}$ is either $s_i \rightsquigarrow^{d_i} s_{i+1}$, $s_i \rightsquigarrow^{U} s_{i+1}$ or $s_i \rightsquigarrow_U s_{i+1}$ for $d_i \in \mathbb{R}_{\geq 0}$ and $U \in \Box$. As usual, π is *accepting* if $s_1 \in S^\bot$ and $s_n \in S^\top$.

Definition 24. *The observable content* $\mathsf{ev}(\pi)$ *of a path* π *in* $[\![A]\!]$ *is the tipomset* $(P, <_P, \dashrightarrow_P, S_P, T_P, \lambda_P, \sigma_P, d_P)$ *defined recursively as follows:*

- *if* $\pi = (l, v)$*, then* $(P, <_P, \dashrightarrow_P, S_P, T_P, \lambda_P) = \mathrm{id}_{\mathsf{ev}(l)}$*,* $\sigma_P(x) = [0, 0]$ *for all* $x \in P$*, and* $d_P = 0$*;*
- *if* $\pi = (l, v) \rightsquigarrow^d (l, v + d)$*, then* $(P, <_P, \dashrightarrow_P, S_P, T_P, \lambda_P) = \mathrm{id}_{\mathsf{ev}(l)}$*,* $\sigma_P(x) = [0, d]$ *for all* $x \in P$*, and* $d_P = d$*;*
- *if* $\pi = (l_1, v_1) \rightsquigarrow^U (l_2, v_2)$*, then* $(P, <_P, \dashrightarrow_P, S_P, T_P, \lambda_P) = {}_U{\uparrow}\mathsf{ev}(l_2)$*,* $\sigma_P(x) = [0, 0]$ *for all* $x \in P$*, and* $d_P = 0$*;*
- *if* $\pi = (l_1, v_1) \rightsquigarrow_U (l_2, v_2)$*, then* $(P, <_P, \dashrightarrow_P, S_P, T_P, \lambda_P) = \mathsf{ev}(l_1){\downarrow_U}$*,* $\sigma_P(x) = [0, 0]$ *for all* $x \in P$*, and* $d_P = 0$*;*
- *if* $\pi = \pi_1 \pi_2$*, then* $\mathsf{ev}(\pi) = \mathsf{ev}(\pi_1) * \mathsf{ev}(\pi_2)$*.*

Observe that by definition of $[\![A]\!]$, the first item above is a special case of the second one with $d = 0$.

Definition 25. *The* language *of an HDTA* A *is*

$$\mathcal{L}(A) = \{\mathsf{ev}(\pi) \mid \pi \text{ accepting path of } A\}.$$

Remark 26. With a few simple changes to Def. 24 above, we can define the observable content of an HDTA path as an idword instead of a tipomset. (By Lem. 22 this is equivalent.) If we define $\mathsf{ev}((l, v) \rightsquigarrow^d (l, v + d)) = d$ in the second case above and use concatenation of idwords instead of gluing composition in the last case, then $\mathsf{ev}(\pi) \in \mathsf{IDW}$. Thus, the language of an HDTA can be seen as a set of tipomsets or as a set of idwords.

Example 27. We compute the language of the HDTA A of Fig. 5. As both vertical transitions are disabled, any accepting path must proceed along the location sequence (l_0, e_1, u, e_4, l_3). The general form of accepting paths is thus

$$\begin{aligned}\pi = (l_0, v^0) &\rightsquigarrow^{d_1} (l_0, v^0 + d_1) \rightsquigarrow^{a} (e_1, v_2) \rightsquigarrow^{d_2} (e_1, v_2 + d_2) \\ &\rightsquigarrow^{b} (u, v_3) \rightsquigarrow^{d_3} (u, v_3 + d_3) \rightsquigarrow_{b} (e_4, v_4) \\ &\rightsquigarrow^{d_4} (e_4, v_4 + d_4) \rightsquigarrow_{a} (l_3, v_5) \rightsquigarrow^{d_5} (l_3, v_5 + d_5).\end{aligned}$$

Fig. 10. Two HDTAs pertaining to Rem. 28

There are no conditions on d_1, as both clocks x and y are reset when leaving l_0. The conditions on x at the other four locations force $1 \le d_2 \le 4$, $1 \le d_2 + d_3 \le 4$, and $2 \le d_2 + d_3 + d_4 \le 5$. As y is reset when leaving e_1, we must have $1 \le d_3 \le 3$ and $1 \le d_3 + d_4$, and the condition on z at l_3 forces $1 \le d_4$. As there are no upper bounds on clocks in l_3, there are no constraints on d_5.

To sum up, $\mathcal{L}(A)$ is the set of tipomsets

$$(\{x_1, x_2\}, \emptyset, x_1 \dashrightarrow x_2, \emptyset, \emptyset, \lambda, \sigma, d_1 + \cdots + d_5)$$

with $\lambda(x_1) = a$, $\lambda(x_2) = b$, $\sigma(x_1) = [d_1, d_1 + \cdots + d_4]$ and $\sigma(x_2) = [d_1 + d_2, d_1 + d_2 + d_3]$, or equivalently the set of idwords

$$\mathrm{id}_\emptyset \, d_1 \, a\bullet \, d_2 \left[\begin{smallmatrix} \bullet a \bullet \\ b \bullet \end{smallmatrix}\right] d_3 \left[\begin{smallmatrix} \bullet a \\ \bullet b \bullet \end{smallmatrix}\right] d_4 \, \bullet b \, d_5 \, \mathrm{id}_\emptyset,$$

in which the delays satisfy the conditions above. As an example,

$$\begin{aligned} \pi = (l_0, (0,0,0)) &\rightsquigarrow^5 (l_0, (5,5,5)) \rightsquigarrow^a (e_1, (0,0,5)) \rightsquigarrow^2 (e_1, (2,2,7)) \\ &\rightsquigarrow^b (u, (2,0,7)) \rightsquigarrow^1 (u, (3,1,7)) \rightsquigarrow_b (e_4, (3,1,0)) \\ &\rightsquigarrow^{1.5} (e_4, (4.5, 2.5, 1.5)) \rightsquigarrow_a (l_3, (4.5, 2.5, 1.5)) \rightsquigarrow^{2.5} (l_3, (7,5,4)) \end{aligned}$$

is an accepting path whose associated tipomset is depicted in Fig. 9. Its interval delay word is

$$\mathrm{id}_\emptyset \, 5 \, a\bullet \, 2 \left[\begin{smallmatrix} \bullet a \bullet \\ b \bullet \end{smallmatrix}\right] 1 \left[\begin{smallmatrix} \bullet a \\ \bullet b \bullet \end{smallmatrix}\right] 1.5 \, \bullet b \, 4.5 \, \mathrm{id}_\emptyset$$

Note that $\mathrm{unt}(\mathcal{L}(A)) = \{\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]\}$ which is not closed under subsumption.

Remark 28. We can now show why the precedence order of a tipomset *cannot* generally be induced from the timestamps. Figure 10 shows two HDTAs in which events labeled a and b happen instantly. On the left, a precedes b, and the language consists of the tipomset ab with duration 0 and $\sigma(a) = \sigma(b) = [0,0]$. On the right, a and b are concurrent, and the language contains the tipomset $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ with the same duration and timestamps.

6.1 Language inclusion is undecidable

[25] introduces a translation from timed automata to HDTAs which we review below. We show that the translation preserves languages. It is not simply an

embedding of timed automata as one-dimensional HDTAs, as transitions in HDTAs are not instantaneous. We use an extra clock to force immediacy of transitions and write $\mathsf{idw}(w)$ for the idword induced by a delay word w below.

Let $A = (\Sigma, C, Q, \bot, \top, I, E)$ be a timed automaton and $C' = C \uplus \{c_T\}$, the disjoint union. In the following, we denote the components of a transition $e = (q_e, \phi_e, \ell_e, R_e, q'_e) \in E$. We define the HDTA $H(A) = (L, \bot, \top, \mathsf{inv}, \mathsf{exit})$ by $L = Q \uplus E$ and, for $q \in Q$ and $e \in E$,

$$\mathsf{ev}(q) = \emptyset, \qquad \mathsf{ev}(e) = \{\ell_e\}, \qquad \delta^0_{\ell_e}(e) = q_e\,, \qquad \delta^1_{\ell_e}(e) = q'_e\,,$$
$$\mathsf{inv}(q) = I(q), \qquad \mathsf{exit}(e) = R_e, \qquad \mathsf{inv}(e) = \phi_e \wedge c_T \leq 0, \qquad \mathsf{exit}(q) = \{c_T\}.$$

Example 29. The HDTA on the left of Fig. 10 is isomorphic to the translation of the timed automaton with the same depiction. (Because of the constraint $x \leq 0$ in the accepting location, the extra clock c_T may be removed.)

Lemma 30. *For any $q_1, q_2 \in Q$ and $v_1, v_2 : C \to \mathbb{R}_{\geq 0}$, $(q_1, v_1) \overset{a}{\rightsquigarrow} (q_2, v_2)$ is an action move of $[\![A]\!]$ if and only if $(q_1, v'_1) \rightsquigarrow^a (e, v')$ and $(e, v') \rightsquigarrow_a (q_2, v'_2)$ are moves of $[\![H(A)]\!]$ such that*

- *for all $c \in C$, $v'_1(c) = v'(c) = v_1(c)$ and $v'_2(c) = v_2(c)$;*
- *$v'_1(c_T) \in \mathbb{R}_{\geq 0}$ and $v'(c_T) = v'_2(c_T) = 0$.*

In addition, $\mathsf{idw}(\mathsf{ev}((q_1, v_1) \overset{a}{\rightsquigarrow} (q_2, v_2))) = \mathsf{ev}((q_1, v'_1) \rightsquigarrow^a (e, v') \rightsquigarrow_a (q_2, v'_2))$.

Lemma 31. *For any $a \in \Sigma$ and $d, d' \in \mathbb{R}_{\geq 0}$, $d\,a\,d'$ is the label of some path in $[\![A]\!]$ if and only if $d\,a\bullet\,0\,\bullet a\,d'$ is the label of some path in $[\![H(A)]\!]$.*

Theorem 32. *For any timed automaton A, $\mathcal{L}(H(A)) = \{\mathsf{idw}(w) \mid w \in \mathcal{L}(A)\}$.*

By the above theorem, we can reduce deciding inclusion of languages of timed automata to deciding inclusion of HDTA languages. It follows that inclusion of HDTA languages is undecidable:

Corollary 33. *For HDTAs A_1, A_2, it is undecidable whether $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$.*

6.2 Untimings of HDTA languages are (almost) regular

We revisit the notions of region equivalence and region automaton from [25] in order to study untimings of languages of HDTAs. For $d \in \mathbb{R}_{\geq 0}$ we write $\lfloor d \rfloor$ and $\langle d \rangle$ for the integral, respectively fractional, parts of d, so that $d = \lfloor d \rfloor + \langle d \rangle$.

Let $A = (\Sigma, C, L, \bot_L, \top_L, \mathsf{inv}, \mathsf{exit})$ be an HDTA. Denote by M the maximal constant which appears in the invariants of A and let $\cong$ denote the region equivalence on $\mathbb{R}^C_{\geq 0}$ induced by A. That is, valuations $v, v' : C \to \mathbb{R}_{\geq 0}$ are *region equivalent* if

- $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ or $v(x), v'(x) > M$, for all $x \in C$, and
- $\langle v(x) \rangle = 0$ iff $\langle v'(x) \rangle = 0$, for all $x \in C$, and
- $\langle v(x) \rangle \leq \langle v(y) \rangle$ iff $\langle v'(x) \rangle \leq \langle v'(y) \rangle$ for all $x, y \in C$.

[25] introduces a notion of untimed bisimulation for HDTA and shows that $\cong$ is an untimed bisimulation. An immediate consequence is the following.

Lemma 34. *Let $t = (l_1, v_1) \rightsquigarrow (l_2, v_2)$ be a transition in $[\![A]\!]$. For all $v_1' \cong v_1$ there exists a transition $t' = (l_1, v_1') \rightsquigarrow (l_2, v_2')$ such that $v_2' \cong v_2$.* □

As usual, a *region* is an equivalence class of $\mathbb{R}^C_{\geq 0}$ under $\cong$. Let $R = \mathbb{R}^C_{\geq 0/\cong}$ denote the set of regions, then R is finite [3].

Definition 35. *The* region automaton *of A is the transition system $R(A) = (S, S^\perp, S^\top, \twoheadrightarrow)$ given as follows:*

$$
\begin{aligned}
S &= \{(l, r) \in L \times R \mid r \subseteq [\![\mathsf{inv}(l)]\!]\} \cup \{(l^0_\perp, \{v^0\}) \mid l^0 \in \perp_L\} \\
S^\perp &= \{(l^0_\perp, \{v^0\}) \mid l^0 \in \perp_L\} \qquad S^\top = S \cap \top_L \times R \\
\twoheadrightarrow &= \{((l^0_\perp, \{v^0\}), \mathrm{id}_{\mathsf{ev}(l^0)}, (l^0, \{v^0\})) \mid l^0 \in \perp_L\} \\
&\quad \cup \{((l, r), \mathrm{id}_{\mathsf{ev}(l)}, (l, r')) \mid \exists v \in r, v' \in r', d \in \mathbb{R}_{\geq 0} : (l, v) \rightsquigarrow^d (l, v'), v' = v + d\} \\
&\quad \cup \{((l, r), {}_U{\uparrow}\mathsf{ev}(l'), (l', r')) \mid \exists v \in r, v' \in r' : (l, v) \rightsquigarrow^U (l', v')\} \\
&\quad \cup \{((l, r), \mathsf{ev}(l){\downarrow_U}, (l', r')) \mid \exists v \in r, v' \in r' : (l, v) \rightsquigarrow_U (l', v')\}
\end{aligned}
$$

Extra copies of start locations are added in order to avoid paths on the empty word ϵ. This construction is similar to the construction of an ST-automaton from an HDA [5]. The region automaton of A is a standard finite automaton whose transitions are labeled by elements of $\mathsf{St} \cup \mathsf{Te}$. Its language is a set of coherent words of $(\mathsf{St} \cup \mathsf{Te})^*$ [5].

Lemma 36. *A path $(l^0, v_0) \rightsquigarrow \cdots \rightsquigarrow (l_p, v_p)$ is accepting in $[\![A]\!]$ if and only if $(l^0_\perp, r_0) \twoheadrightarrow (l^0, r_0) \twoheadrightarrow \cdots \twoheadrightarrow (l_p, r_p)$ with $v_i \in r_i$ is accepting in $R(A)$.*

Theorem 37. *For any HDTA A, $\mathcal{L}(R(A)) = \mathrm{unt}(\mathcal{L}(A))$.*

By the Kleene theorem for finite automata, $\mathcal{L}(R(A))$ is represented by a regular expression over $\mathsf{St} \cup \mathsf{Te}$. Since $P_0 * P_1 * \cdots * P_n$ is accepted by A if and only if the coherent word $\mathrm{unt}(P_0)\,\mathrm{unt}(P_1) \ldots \mathrm{unt}(P_n)$ is accepted by $R(A)$, Theorems 6 and 37 now imply the following.

Corollary 38. *For any HDTA A, $\mathrm{unt}(\mathcal{L}(A))\downarrow$ is a regular ipomset language.*

In [5] it is shown that inclusion of regular ipomset languages is decidable. Now untimings of HDTA languages are not regular because they are not closed under subsumption, but the proof in [5], using ST-automata, immediately extends to a proof of the fact that also inclusion of untimings of HDTA languages is decidable:

Corollary 39. *For HDTAs A_1 and A_2, it is decidable whether $\mathrm{unt}(\mathcal{L}(A_1)) \subseteq \mathrm{unt}(\mathcal{L}(A_2))$.*

7 Conclusion and Perspectives

We have introduced a new language-based semantics for real-time concurrency, informed by recent work on higher-dimensional timed automata (HDTAs) and on languages of higher-dimensional automata. On one side we have combined the delay words of timed automata with the step sequences of higher-dimensional automata into interval delay words. On the other side we have generalized the timed words of timed automata and the ipomsets (*i.e.*, pomsets with interfaces) of higher-dimensional automata into timed ipomsets. We have further shown that both approaches are equivalent.

Higher-dimensional timed automata model concurrency with higher-dimensional cells and real time with clock constraints. Analogously, timed ipomsets express concurrency by partial orders and real time by interval timestamps on events. Compared to related work on languages of time Petri nets, what is new here are the interfaces and the fact that each event has two timestamps (instead of only one), the first marking its beginning and the second its termination. This permits to introduce a gluing operation for timed ipomsets which generalizes serial composition for pomsets. It further allows us to generalize step decompositions of ipomsets into a notion of interval delay words which resemble the delay words of timed automata.

As an application, we have shown that language inclusion of HDTAs is undecidable, but that the untimings of their languages have enough regularity to imply decidability of untimed language inclusion.

Perspectives. We have seen that unlike languages of higher-dimensional automata, untimings of HDTA languages are not closed under subsumption. This relates HDTAs to partial higher-dimensional automata [23, 31] and calls for the introduction of a proper language theory of these models.

Secondly, higher-dimensional automata admit Kleene and Myhill-Nerode theorems [27,32], but for timed automata this is more difficult [8]. We are wondering how such properties will play out for HDTAs.

Timed automata are very useful in real-time model checking, and our language-based semantics opens up first venues for real-time concurrent model checking using HDTAs and some linear-time logic akin to LTL. What would be needed now are notions of simulation and bisimulation—we conjecture that as for timed automata, these should be decidable for HDTAs—and a relation with CTL-type logics. One advantage of HDTAs is that they admit a partial-order semantics, so partial-order reduction (which is difficult for timed automata [15, 36]) should not be necessary.

Finally, a note on robustness. Adding information about durations and timings of events to HDTAs raises questions similar to those already existing in timed automata. Indeed, the model of timed automata supports unrealistic assumptions about clock precision and zero-delay actions, and adding concurrency makes the need for robustness in HDTAs even more crucial. It is thus pertinent to study the robustness of HDTAs and their languages under delay perturbations, similarly for example to the work done in [17, 21, 22].

Robustness may be formalized using notions of distances and topology, see for example [9,10,24,30,38]. Distances between timed words need to take permutations of symbols into account [9], and it seems promising to use partial orders and timed ipomsets to formalize this.

References

1. Luca Aceto, Anna Ingólfsdóttir, Kim G. Larsen, and Jiří Srba. *Reactive Systems*. Cambridge University Press, 2007.
2. Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.
3. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. Amazigh Amrane, Hugo Bazille, Emily Clement, and Uli Fahrenberg. Languages of higher-dimensional timed automata. *CoRR*, abs/2401.17444, 2024.
5. Amazigh Amrane, Hugo Bazille, Uli Fahrenberg, and Krzysztof Ziemiański. Closure and decision properties for higher-dimensional automata. In Erika Ábrahám, Clemens Dubslaff, and Silvia Lizeth Tapia Tarifa, editors, *ICTAC*, volume 14446 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2023.
6. Étienne André. IMITATOR: A tool for synthesizing constraints on timing bounds of timed automata. In Martin Leucker and Carroll Morgan, editors, *ICTAC*, volume 5684 of *Lecture Notes in Computer Science*, pages 336–342. Springer, August 2009.
7. Étienne André. IMITATOR 3: Synthesis of timing parameters beyond decidability. In Alexandra Silva and K. Rustan M. Leino, editors, *CAV*, volume 12759 of *Lecture Notes in Computer Science*, pages 552–565. Springer, July 2021.
8. Eugene Asarin. Challenges in timed languages: From applied theory to basic theory. *Bulletin of the EATCS*, 83:106–120, 2004.
9. Eugene Asarin, Nicolas Basset, and Aldric Degorre. Entropy of regular timed languages. *Information and Computation*, 241:142–176, 2015.
10. Eugene Asarin, Nicolas Basset, and Aldric Degorre. Distance on timed words and applications. In David N. Jansen and Pavithra Prabhakar, editors, *FORMATS*, volume 11022 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2018.
11. Tuomas Aura and Johan Lilius. A causal semantics for time Petri nets. *Theoretical Computer Science*, 243(1-2):409–447, 2000.
12. Sandie Balaguer, Thomas Chatain, and Stefan Haar. A concurrency-preserving translation from time Petri nets to networks of timed automata. *Formal Methods in System Design*, 40(3):330–355, 2012.
13. Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on Uppaal. In Marco Bernardo and Flavio Corradini, editors, *SFM-RT*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.
14. Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *QEST*, pages 125–126. IEEE Computer Society, September 2006.
15. Frederik M. Bønneland, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marco Muñiz, and Jirí Srba. Start pruning when time gets urgent: Partial order reduction for timed systems. In Hana Chockler and Georg Weissenbacher, editors, *CAV*, volume 10981 of *Lecture Notes in Computer Science*, pages 527–546. Springer, 2018.

16. Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, Joël Ouaknine, and James Worrell. Model checking real-time systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 1001–1046. Springer, 2018.
17. Patricia Bouyer, Erwin Fang, and Nicolas Markey. Permissive strategies in timed automata and games. *Electronic Communications of the EASST*, 72, 2015.
18. Thomas Chatain. *Concurrency in Real-Time Distributed Systems, from Unfoldings to Implementability*. PhD thesis, École normale supérieure de Cachan, 2013.
19. Thomas Chatain and Claude Jard. Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In Susanna Donatelli and P. S. Thiagarajan, editors, *PETRI NETS*, volume 4024 of *Lecture Notes in Computer Science*, pages 125–145. Springer, 2006.
20. Thomas Chatain and Claude Jard. Back in time Petri nets. In Víctor A. Braberman and Laurent Fribourg, editors, *FORMATS*, volume 8053 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 2013.
21. Emily Clement. *Robustness of Timed Automata: Computing the Maximally-Permissive Strategies*. PhD thesis, University of Rennes 1, France, 2022.
22. Emily Clement, Thierry Jéron, Nicolas Markey, and David Mentré. Computing maximally-permissive strategies in acyclic timed automata. In Nathalie Bertrand and Nils Jansen, editors, *FORMATS*, volume 12288 of *Lecture Notes in Computer Science*, pages 111–126. Springer, September 2020.
23. Jérémy Dubut. Trees in partial higher dimensional automata. In Mikołaj Bojańczyk and Alex Simpson, editors, *FOSSACS*, volume 11425 of *Lecture Notes in Computer Science*, pages 224–241. Springer, 2019.
24. Uli Fahrenberg. *A Generic Approach to Quantitative Verification*. Habilitation thesis, Paris-Saclay University, 2022.
25. Uli Fahrenberg. Higher-dimensional timed and hybrid automata. *Leibniz Transactions on Embedded Systems*, 8(2):03:1–03:16, 2022.
26. Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Languages of higher-dimensional automata. *Mathematical Structures in Computer Science*, page 1–39, 2021.
27. Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. A Kleene theorem for higher-dimensional automata. In Bartek Klin, Sławomir Lasota, and Anca Muscholl, editors, *CONCUR*, volume 243 of *LIPIcs*, pages 29:1–29:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
28. Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Posets with interfaces as a model for concurrency. *Information and Computation*, 285(B):104914, 2022.
29. Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Catoids and modal convolution algebras. *Algebra Universalis*, 84(10), 2023.
30. Uli Fahrenberg and Axel Legay. The quantitative linear-time–branching-time spectrum. *Theoretical Computer Science*, 538:54–69, 2014.
31. Uli Fahrenberg and Axel Legay. Partial higher-dimensional automata. In Lawrence S. Moss and Pawel Sobocinski, editors, *CALCO*, volume 35 of *LIPIcs*, pages 101–115, 2015.
32. Uli Fahrenberg and Krzysztof Ziemiański. A Myhill-Nerode theorem for higher-dimensional automata. In Luís Gomes and Robert Lorenz, editors, *PETRI NETS*, volume 13929 of *Lecture Notes in Computer Science*, pages 167–188. Springer, 2023.
33. Peter C. Fishburn. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*. Wiley, 1985.

34. Hans Fleischhack and Christian Stehno. Computing a finite prefix of a time Petri net. In Javier Esparza and Charles Lakos, editors, *PETRI NETS*, volume 2360 of *Lecture Notes in Computer Science*, pages 163–181. Springer, 2002.
35. Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier H. Roux. Romeo: A tool for analyzing time Petri nets. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423. Springer, July 2005.
36. R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Abstractions for the local-time semantics of timed automata: A foundation for partial-order methods. In Christel Baier and Dana Fisman, editors, *LICS*, pages 24:1–24:14. ACM, 2022.
37. Jan Grabowski. On partial languages. *Fundamenta Informaticae*, 4(2):427, 1981.
38. Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In Oded Maler, editor, *HART*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 1997.
39. Ryszard Janicki and Maciej Koutny. Operational semantics, interval orders and sequences of antichains. *Fundamenta Informaticae*, 169(1-2):31–55, 2019.
40. Kim G. Larsen, Uli Fahrenberg, and Axel Legay. From timed automata to stochastic hybrid games. In *Dependable Software Systems Engineering*, pages 60–103. IOS Press, 2017.
41. Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *Int. J. Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
42. Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou, editors, *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57. Springer, March 2009.
43. Philip M. Merlin and David J. Farber. Recoverability of communication protocols – implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.
44. Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
45. Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.
46. Carl A. Petri. *Kommunikation mit Automaten.* Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.
47. Vaughan R. Pratt. Modeling concurrency with geometry. In David S. Wise, editor, *POPL*, pages 311–322. ACM Press, 1991.
48. Rob J. van Glabbeek. Bisimulations for higher dimensional automata. Email message, June 1991. `http://theory.stanford.edu/~rvg/hda`.
49. Rob J. van Glabbeek. On the expressiveness of higher dimensional automata. *Theoretical Computer Science*, 356(3):265–290, 2006. See also [50].
50. Rob J. van Glabbeek. Erratum to "On the expressiveness of higher dimensional automata". *Theoretical Computer Science*, 368(1-2):168–194, 2006.
51. Rob J. van Glabbeek and Gordon D. Plotkin. Configuration structures. In *LICS*, pages 199–209. IEEE Computer Society, 1995.
52. Rob J. van Glabbeek and Gordon D. Plotkin. Configuration structures, event structures and Petri nets. *Theoretical Computer Science*, 410(41):4111–4159, 2009.

Appendix: Proofs

Proof (of Lem. 13). If two words of the form of the lemma are equivalent, then they are equal. This proves uniqueness. To show existence, first note that by using the equivalences

$$dd' \sim d + d', \qquad 0 \sim \epsilon,$$

any element of IDW may be rewritten to a word (not necessarily unique)

$$I_0 d_0 P_1 d_1 \cdots P_n d_n I_{n+1} \tag{5}$$

with $n \geq 0$ and $P_1, \ldots, P_n \in \mathsf{St} \cup \mathsf{Te}$.

Next we prove by induction on n that any word w of the form (5) is equivalent to one as in the lemma. The case $n = 0$ is trivial.

Now let $n \geq 1$ and suppose the property holds up to index $n-1$. If w is not of the form (5) at index n, then this may be for two reasons.

- There is $1 \leq k \leq n-1$ such that $d_k = 0$ and $(P_k, P_{k+1}) \notin \mathsf{St}_+ \times \mathsf{Te}_+ \cup \mathsf{Te}_+ \times \mathsf{St}_+$. In this case, $P_k d_k P_{k+1} \sim P_k * P_{k+1} \in \mathsf{St} \cup \mathsf{Te}$, and $w \sim w' = I_0 d_0 P_1 d_1 \cdots d_{k-1}(P_k * P_{k+1}) d_{k+1} \cdots P_n d_n I_{n+1}$.
- There is $1 \leq k \leq n$ such that $P_k \in \mathsf{Id}$. Then $d_{k-1} P_k d_k \sim d_{k-1} + d_k$ and $w \sim w' = I_0 d_0 P_1 d_1 \cdots P_{k-1}(d_{k-1} + d_k) P_{k+1} d_{k+1} \cdots P_n d_n I_{n+1}$.

In both cases, applying the induction hypothesis to w' finishes the proof.

Proof (of Lem. 22). To see injectivity, let $w = I_0 d_0 P_1 d_1 \ldots P_n d_n I_{n+1}$ and $w' = I'_0 d'_0 P'_1 d'_1 \ldots\ P'_n d'_m I'_{m+1}$ be sparse idwords such that $P = \mathsf{tGlue}(w) = \mathsf{tGlue}(w')$. By Lem. 3, $n = m$ and $(I_0, P_0, \ldots, P_n, I_{n+1}) = (I'_0, P'_0, \ldots, P'_m, I'_{m+1})$. Hence also $\mathsf{first}(x) = \mathsf{first}'(x)$ and $\mathsf{last}(x) = \mathsf{last}'(x)$ for all $x \in P$, and by induction, $d_i = d'_i$ for all i.

To show surjectivity, let P be a tipomset, $P = P_1 * \cdots * P_n$ its unique sparse decomposition given by Lem. 3, and $I_0 = \mathrm{id}_{S_{P_1}}$, $I_{n+1} = \mathrm{id}_{T_{P_n}}$. Again by induction, we can use first and last to define the delays $d_0, \ldots, d_n$, and then $P = \mathsf{tGlue}(I_0 d_0 P_1 d_1 \cdots P_n d_n I_{n+1})$.

Proof (of Lem. 23). For i_1, let $w = d_0 a_1 d_1 \ldots a_n d_n$ be a delay word, then w is translated to the idword $i_1(w) = \mathrm{id}_\emptyset\, d_0\, a_1 \bullet 0 \bullet a_1\, d_1 \ldots a_n \bullet 0 \bullet a_n\, d_n\, \mathrm{id}_\emptyset$, alternating starts and terminations of actions with 0 delays in-between (and the original delays between different actions).

Similarly for i_2, a sparse step sequence $I_0 P_1 \ldots P_n I_{n+1}$ is translated to the (sparse) idword $I_0\, 0\, P_1\, 0 \ldots 0\, P_n\, 0\, I_{n+1}$, inserting 0 delays between all elements.

For i_3, let $w = (a_0, t_0) \ldots (a_n, t_n)\, t_{n+1}$ be a timed word. This is translated to the tipomset $(\{x_1, \ldots, x_n\}, <, \sigma, \emptyset, \emptyset, \emptyset, \lambda, d)$ with $x_i < x_j \iff i < j$, $\sigma(x_i) = [t_i, t_i]$, $\lambda(x_i) = a_i$, and $d = t_{n+1}$.

Finally, an ipomset P is translated to a tipomset (P, σ, d) by setting $\sigma(x) = [0, 0]$ for all $x \in P$ and $d = 0$.

Injectivity of these four mappings (up to isomorphism) is clear, as is the fact that they commute with the horizontal bijections.

Proof (of Lem. 30). By construction, $(q_1, \phi, a, R, q_2) \in E$ if and only if there exist $q_1, q_2, e \in L$ such that $\delta^0_a(e) = q_1$, $\delta^1_a(e) = q_2$, $\mathsf{ev}(e) = a$, $\mathsf{inv}(q) = I(q)$, $\mathsf{inv}(e) = \phi \wedge c_T \leq 0$, $\mathsf{exit}(q) = \{c_T\}$, and $\mathsf{exit}(e) = R$. Let us denote by $\rightsquigarrow_A$ and $\rightsquigarrow_H$ the set of transitions of $[\![A]\!]$ and $[\![H(A)]\!]$, respectively. Then $\{((q_1, v_1), a, (q_2, v_2)) \mid v_1 \models \phi, v_2 = v_1[R \leftarrow 0]\} \subseteq \rightsquigarrow_A$ if and only if

$$\{((q_1, v_1'), (e, v')) \mid v' = v_1'[c_T \leftarrow 0] \models \phi \wedge c_T \leq 0\} \subseteq \rightsquigarrow_H$$

and

$$\{((e, v'), (q_2, v_2')) \mid v_2' = v'[R \leftarrow 0] \models I(q_2)\} \subseteq \rightsquigarrow_H.$$

This proves the first part of the lemma. The second part is obtained directly from the definitions.

Proof (of Lem. 31). From Lem. 30 we know that a is the label of a path in $[\![A]\!]$ if and only if $\mathsf{idw}(a) = a \bullet 0 \bullet a$ is the label of some path in $[\![H(A)]\!]$. We conclude by noting that by construction, time evolves in $H(A)$ only in 0-dimensional locations and exactly as it evolves in A.

Proof (of Thm. 32). From Lem. 31 we know that $d\,a\,d'$ is the label of some path in $[\![A]\!]$ if and only if $\mathsf{idw}(d\,a\,d')$ is the label of some path in $[\![H(A)]\!]$. Now for any two delay words w, w', $\mathsf{idw}(ww') = \mathsf{idw}(w)\,\mathsf{idw}(w')$, so the theorem follows by induction on paths and from $\bot_{H(A)} = \bot_A$ and $\top_{H(A)} = \top_A$.

Proof (of Lem. 36). The direction from left to right follows directly from the definition of $\twoheadrightarrow$ in Def. 35. For the other direction we proceed by induction on p. For the base case $p = 0$ we have $(l^0, \{v^0\})$ in $[\![A]\!]$.

Assume now that the lemma is true for $p-1$, that is we have a path $(l_0, v_0) \rightsquigarrow \cdots \rightsquigarrow (l_{p-1}, v_{p-1})$ in $[\![A]\!]$ satisfying the lemma. By construction the presence of a transition $(l_{p-1}, r_{p-1}) \twoheadrightarrow (l_p, r_p)$ in $R(A)$ is due to the existence of some transition $(l_{p-1}, v'_{p-1}) \twoheadrightarrow (l_p, v'_p)$ in $[\![A]\!]$ with $v'_{p-1} \in r_{p-1}$ and $v'_p \in r_p$. Since $v_{p-1} \cong v'_{p-1}$, we can conclude by using Lem. 34.

Proof (of Thm. 37). From Lem. 36 we know that for all $n \geq 0$, a path $\pi = (l^0, v_0) \rightsquigarrow \cdots \rightsquigarrow (l_n, v_n)$ in $[\![A]\!]$ is accepting if and only if $\pi' = (l^0_\bot, r_0) \twoheadrightarrow (l^0, r_0) \twoheadrightarrow \cdots \twoheadrightarrow (l_n, r_n)$ with $v_i \in r_i$ is accepting in $R(A)$. Let $P_0 = \mathsf{ev}((l^0, v_0))$ and $P_i = \mathsf{ev}((l_{i-1}, v_{i-1}) \rightsquigarrow (l_i, v_i))$ for all $1 \leq i \leq n$. Then $\mathsf{ev}(\pi) = P_1 * \cdots * P_n = P_0 * P_1 * \cdots * P_n$.

Notice that by construction of $R(A)$, $\mathrm{unt}(P_0)$ is the label of $(l^0_\bot, r_0) \twoheadrightarrow (l^0, r_0)$ and for all $1 \leq i \leq n$, $\mathrm{unt}(P_i)$ is the label of $(l_{i-1}, r_{i-1}) \twoheadrightarrow (l_i, r_i)$ regardless of the type of the move $\rightsquigarrow$. Thus the label of π' is $\mathrm{unt}(P_0)\,\mathrm{unt}(P_1) \ldots \mathrm{unt}(P_n)$. Hence $P_0 * P_1 * \cdots * P_n$ is accepted by A if and only if $\mathrm{unt}(P_0)\,\mathrm{unt}(P_1) \ldots \mathrm{unt}(P_n)$ is accepted by $R(A)$. We conclude:

$$\begin{aligned}
\mathcal{L}(R(A)) &= \{\mathrm{unt}(P_0) * \mathrm{unt}(P_1) * \cdots * \mathrm{unt}(P_n) \mid P_0 * P_1 * \cdots * P_n \in \mathcal{L}(A)\} \\
&= \{\mathrm{unt}(P_0 * P_1 * \cdots * P_n) \mid P_0 * P_1 * \cdots * P_n \in \mathcal{L}(A)\} \qquad \text{(Lem. 18)} \\
&= \mathrm{unt}(\mathcal{L}(A))
\end{aligned}$$