Petri Nets to Higher-Dimensional Automata

Timothée FRAGNAUD
(supervisor: Hugo BAZILLE)

Technical Report n^o202407-techrep-fragnaud, July 2024
revision 2308662

Petri nets (PN) and higher-dimensional automata (HDA) are both formalisms for modeling concurrent systems. Petri nets are represented by places, transitions, arcs, and a marking of places. The marking defines if we can activate a transition or not. On the other hand, higher-dimensional automata are a set of points, edges, squares, (hyper-)cubes, etc. where we can find a path through a subset of the given precubical set in order to join two cells. Even if they are very different, we know that a bounded Petri net can be converted to a higher-dimensional automata. However, the reciprocal is not true. The purpose of this project is to implement this algorithm to convert Petri nets to HDA. To achieve this goal, our primary focuses are the conversion algorithm itself and the creation of a formalism to represent and save HDAs, as no such standard exists.

Les réseaux de Petri (PN) et les automates en dimension supérieure (HDA) sont deux formalismes pour modéliser les systèmes concurrents. Les réseaux de Petri sont représentés par des places, des transitions, des arcs et un marquage des places. Le marquage définit si nous pouvons activer une transition ou non. D'autre part, les automates en dimension supérieure sont un ensemble de points, arêtes, carrés, (hyper-)cubes etc., où nous pouvons trouver un chemin à travers un sous-ensemble de l'ensemble précubique donné afin de relier deux cellules. Même s'ils sont très différents, nous savons qu'un réseau de Petri borné peut être converti en un automate en dimension supérieure. Cependant, la réciproque n'est pas vraie. L'objectif de ce projet est de mettre en œuvre cet algorithme pour convertir des réseaux de Petri en HDA. Pour atteindre cet objectif, nos principaux axes sont l'algorithme de conversion lui-même et la création d'un formalisme pour représenter et sauvegarder les HDA, car aucune norme de ce type n'existe.

Keywords
Petri Nets; Higher-Dimensional Automata; Concurrency theory; Conversion algorithm; Pomsets; ST-Traces

Laboratoire de Recherche de l'EPITA
14-16, rue Voltaire – FR-94276 Le Kremlin-Bicêtre CEDEX – France
Tél. +33 1 53 14 59 22 – Fax. +33 1 53 14 59 13
tfragnaud@lre.epita.fr – http://www.lre.epita.fr/

Copying this document

Contents

Introduction

The study of concurrent systems is pivotal in computer science, particularly for modeling and analyzing systems in which multiple processes occur concurrently. Two prominent formalisms for representing these systems are Petri nets (PN) and higher-dimensional automata (HDA). Petri nets, characterized by places, transitions, arcs and markings, provide a visual and straightforward method to depict distributed systems and manage resources. In a Petri net, the state of a system is represented by the marking of places, indicating whether transitions can be activated. This makes Petri nets a powerful tool for modeling system states and transitions in a clear and concise manner.

In contrast, higher-dimensional automata (HDA) extend the traditional automata model by incorporating higher-dimensional constructs, including points, edges, squares, and beyond. This allows HDAs to represent more complex relationships and interactions within concurrent processes, thereby providing a detailed and nuanced view of systems behavior. Notwithstanding the discrepancies in their structural and representational characteristics, it has been demonstrated that a bounded Petri net can be transformed into a higher-dimensional automaton. However, the inverse conversion has not yet been established as viable.

The motivation for converting Petri nets into higher-dimensional automata lies in harnessing the strengths of both formalisms. While Petri nets are appreciated for their simplicity and intuitive graphical representation, HDAs offer a richer and more detailed depiction of concurrent and parallel processes, which is particularly valuable in contexts mainly based on concurrency. The objective of this research is to establish a connection between these formalisms by developing and implementing an algorithm that converts Petri nets into higher-dimensional automata. Furthermore, due to the lack of a standard for representing and saving HDAs, this project will also focus on developing a formalism for HDAs.

Contributions This paper presents several key contributions:

- The development and implementation of a theoretical algorithm to convert Petri nets to higher-dimensional automata.
- The establishment of a formalism for representing and saving HDAs.

Overview of the Paper The remainder of this paper is organized as follows:

- Chapter 1 introduces the fundamental concepts that are essential for a comprehensive understanding of the two models used: Petri nets and higher-dimensional automata. It also defines the useful properties of these models and establishes a connection between them.

- Chapter 2 presents the conversion algorithm as originally proposed by Glabbeek [4].
- Chapter 3 presents my own implementation process and choices and the HDA standard format we want to introduce.
- Chapter 4 is a discussion about the limitations of the aforementioned algorithm.

Acknowledgements I would like to thank my supervisors, Hugo Bazille and Uli Fahrenberg, for their guidance and support throughout this project.

Chapter 1

Preliminaries

1.1 Petri Nets

A Petri net (PN) is a mathematical model used to represent distributed systems and concurrency theory by processing some resources (tokens) [4, 7]. A Petri net can be seen as a bipartite graph with two types of vertices: places and transitions. Places are used to store tokens while transitions represent actions and are indeed labelled. The tokens represent the resources available in the place they are stored. A marking is the distribution of these tokens. It thus represents the distribution of available resources within the different places. These resources will then be used by transitions representing events. The distribution of tokens through places is therefore a marking and the initial distribution is the initial marking. Some arcs make links from places to transitions or from transitions to places. The set of arcs is the flow of the Petri net.

Definition 1. *A Petri Net is a tuple* $N = (S, T, F, M_0, l)$ *where:*

- S *is a finite set of places,* T *is a finite set of transitions, and* $S \cap T = \emptyset$.
- $F : (S \times T \cup T \times S) \to \mathbb{N}$ *is the flow (a mapping defining directed (multi)arcs between transitions and places).*
- $M_0 : S \to \mathbb{N}$ *is the initial marking.*
- $l : T \to A$ *is a labelling function (with* A *a set of actions)*

Figure 1.1: Petri Net graphical representation

With the Fig. 1.1, we can show the graphical representation of the components of a Petri net. In this figure, places are represented with circles, transitions with rectangles and arcs with arrows. Tokens are represented with black filled circles in places. We can also denote the initial marking for this Petri net as a list of numbers representing the distribution of tokens for each place, indexing the places from left to right, such that the available tokens for the place at index i are the number at the same index in the marking list: $M_0 = (1, 1, 0)$. Finally, the labels are letters inside transitions. For that Petri net, we have the label a for the first transition and the label b for the last one.

The preset, and the postest of a transition t is the set of places that have an arc to, respectively from that transition. They represent the input, respectively output of the transition t.

Example:
In the Fig. 1.1, the preset of the transition labelled a is only the first place and the postset of that transition is only the second place.

Definition 2. *Let $N = (S, T, F, M_0, l)$ be a Petri net. For all $t \in T$:*

- $\bullet t = \{s \in S \mid F(s,t) > 0\}$ *is the* ***preset*** *of the transition t.*
- $t\bullet = \{s \in S \mid F(t,s) > 0\}$ *is the* ***postset*** *of the transition t.*

A transition t is enabled and can be activated if that transition has enough tokens in its input places. We consider that such transition has enough tokens in its input places, if for all of the places s in its preset, the number of tokens in that place is at least equal to the number of arcs from the place s to the transition t.

Definition 3. *Let $N = (S, T, F, M_0, l)$ be a Petri net. For all $t \in T$:*

$$t \textbf{ is enabled} \Leftrightarrow \forall s \in \bullet t,\ M(s) \geq F(s,t)$$

Example:
In the Fig. 1.1, the transition labelled a is enabled as well as the transition labelled b. In fact, for the transition labelled a, the number of tokens in its only one input place is 1 and there is exactly one arc from that place to the transition labelled a. The same reasoning can be used for the transition labelled b.

If a transition t is enabled, it means we can activate it. The activation process will process the tokens through the flow: we removed the one token per arc for each places in the preset and we add one token per arc for each places in the postset. The activation of a transition will produce a new marking representing the new distribution of tokens in places.

Definition 4. *Let $N = (S, T, F, M_0, l)$ be a Petri net and $M : S \to \mathbb{N}$ be the current marking. For all $t \in T$ such as t is enabled, the new marking resulting the* ***activation*** *of the transition t is:*

$$M'_t : S \to \mathbb{N}$$
$$\forall s \in S, \quad M'_t : s \mapsto M(s) - F(s,t) + F(t,s)$$

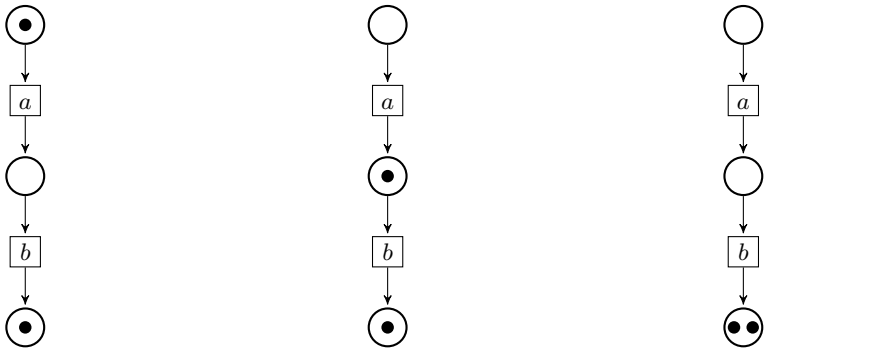
We say that the marking M'_t is reachable in one step from the marking M by activating the transition t and we denote it: $M \xrightarrow{t} M'_t$ such that $M < M'_t$.

Example:
In the Fig. 1.2a, the activation of the transition labelled b will result in a new distribution of tokens where the token in the second place was removed (because of the arc from it to the transition studied) and a new token was added in the last place (because the last place belong to the postset of the transition labelled b with one arc).

A run (or a fire sequence) is a succession of the activation of transitions until there is no more enabled transitions. For one Petri net, there may be several possible run.

Definition 5. *Let $N = (S, T, F, M_0, l)$ be a Petri net. A **run** (or firing sequence) of the Petri net is a sequence of transitions $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$ such that there exists a sequence of markings $M_0 \xrightarrow{t_1} M_1 \wedge M_1 \xrightarrow{t_2} M_2 \wedge \ldots \wedge M_{n-1} \xrightarrow{t_n} M_n$ where:*

- *M_0 is the initial marking,*
- *$\forall i \in \{1, 2, \ldots, n\}$, the transition t_i is enabled at M_{i-1} and activating t_i results in the marking M_i.*



(a) activation of transition b **(b)** activation of transition a **(c)** activation of transition b

Figure 1.2: A possible run for Fig. 1.1

Example:
The Fig. 1.2 illustrates one possible run for the Petri net described in Fig. 1.1. We start with the initial marking $M_0 = (1, 1, 0)$. Then in Fig. 1.2a, we activate the transition labelled b resulting in the new marking $M_1 = (1, 0, 1)$. After that, in Fig. 1.2b, we activate the transition labelled a to create the marking $M_2 = (0, 1, 1)$ and we can finally activate the transition labelled b, in Fig. 1.2c, to distribute the tokens according the marking $M_3 = (0, 0, 2)$. In that final marking, we can notice that there is no more transition enabled, we reached the end of the run.

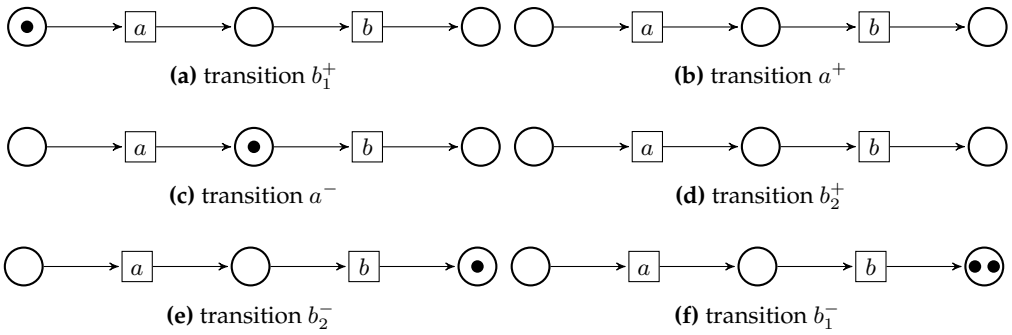
We want to consider that the activation of a transition is not instantaneous and need some time to be achieved. In addition, we must be able to activate more than one transition simultaneously, enabled the concurrency of action in our run. To do so, we firstly need to define the activation in two parts: the stating phase and the finishing phase. The starting phase, and the finishing phase, denoted with a plus sign, respectively a minus sign in exponent, is the part consisting of removing tokens from preset places, respectively adding tokens to postset places. To start the activation of a transition, that transition must be enabled. Moreover, to finish the activation of a transition, we need to start that transition first. With this activation definition, we can now start multiples transition while they does not finished yet. With this new activation process, a run is now a succession of starting and finishing phases such as all starting phases occur before their corresponding finishing phases and all started transition must be finished.

Definition 6. *Let $N = (S, T, F, M_0, l)$ be a Petri net and $M : S \to \mathbb{N}$ be the current marking. For all $t \in T$ such as t is enabled, the activation of the transition t can be split into two steps:*

- *$M_t^{+\prime}(s) = M(s) - F(s, t)$ is the starting phase.*
- *$M_t^{-\prime}(s) = M(s) + F(t, s)$ is the finishing phase.*

A run is now a sequence of start and end part of transitions $\sigma = \langle t_1^+, \ldots, t_i^+, \ldots, t_i^-, \ldots \rangle$ *for all* $1 \leq i \leq n$ *such that there exists a sequence of markings* $M_0 \xrightarrow{t_1^+} M_1^+ \wedge \ldots \wedge M_j \xrightarrow{t_i^+} M_i^+ \wedge \ldots \wedge M_k \xrightarrow{t_i^-} M_i^- \wedge \ldots$ *where:*

- M_0 *is the initial marking,*
- $\forall i \in \{1, 2, \ldots, n\}, \exists M_j < M_i^+, M_j \xrightarrow{t_i^+} M_i^+$, *the transition* t_i *is enabled at a state* M_j *and starting* t_i *results in the marking* M_i^+. *And* $\exists M_k < M_i^-, M_i^+ < M_i^-, M_k \xrightarrow{t_i^-} M_i^-$, *the transition* t_i *will be finished from the state* M_k *resulting in the marking* M_i^-.



(a) transition b_1^+ **(b)** transition a^+

(c) transition a^- **(d)** transition b_2^+

(e) transition b_2^- **(f)** transition b_1^-

Figure 1.3: A possible run with concurrency for figure 1.1

Example:
The Fig. 1.3 presents a possible run for the Petri net Fig. 1.1 with the concurrency definition. For example, when we start the transition labelled b in Fig. 1.3a, we only removed the token in the second place. Later on, that transition is finished in Fig. 1.3f resulting in the addition of a token in the last place (which is the only output place for that transition). In that example, we can notice that the transition labelled a is activated while the transition labelled b is still running. So we have a concurrence between the action b firstly activated and the action a.

1.2 Pomset

The Pomsets (partially ordered multi-set) is a labeled set with a finite partial order relation which is transitive, irreflexive and anti-symmetric [1, 2]. Those sets represent events, and the order relation is used to describe their precedence. Consequently, an event that has a predecessor (i.e., another event that precedes it according to the set's order relation) cannot begin until its predecessor has finished. It must therefore wait for the predecessor to finish. This relationship can therefore be used to represent sequential events. Furthermore, being partial, this order relation does not apply to certain pairs of events, which can then evolve at the same time (without restriction on waiting for the other). This enables the representation of concurrent events. Pomsets thus represent the behavior of each event as a function of the others.

Definition 7. *A* ***pomset*** *(partially ordered multi-set) over* Σ *is a triplet* $p = (E, \prec, \lambda)$ *where* E *is a finite set,* $\prec \subseteq E \times E$ *is a finite partial order and* $\lambda : E \to \Sigma$ *is a mapping from* E *to* Σ.

A pomset $p = (E, \prec, \lambda)$ is without auto-concurrency if $\lambda(x) = \lambda(y)$ implies $(x \prec y \vee y \prec x)$ for all $x, y \in E$.

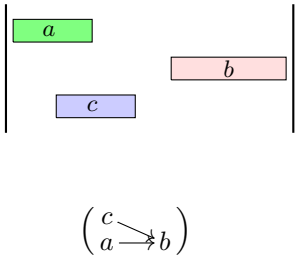


Figure 1.4: pomset representation

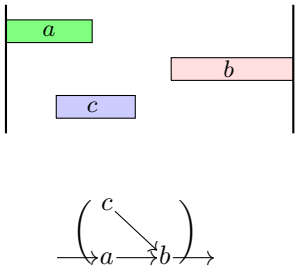


Figure 1.5: IPomset representation

Example:
In the Fig. 1.4, we represent the interval representation of three events denoted a, b, c and the corresponding Pomset. The interval representation allows us to easily display the time spent by each event for completion, and the pomset is mainly used to formally represent the order relation between those events. As we can notice, the events a and c occur before the event b and, in the pomset, we represent that precedence relation by the arrows such as $a \prec b$ and $c \prec b$. Moreover, there is no order relation between the events a and c. Even if, in the interval representation, the event a is started before the event c, we cannot order them because the event a is not finished before the event c starts. We have concurrency between a and c.

Pomsets can be equipped with interfaces [3]. There are two distinct types of interfaces: starting and terminating. This allows us to represent uncompleted events. In the context of event representation, starting interfaces are used to denote *unstarted* events, that is, events that have not been started. In contrast, terminating interfaces represent *unterminated* events, or events that are not completed.

Definition 8. *A pomset with interfaces (**ipomset**)* $(E, \prec, S, T, \lambda)$ *consists of a pomset* $(E, \prec, \lambda)$ *with 2 subsets* $S, T \subseteq E$ *(source and target interfaces) such that* $\forall x \in S, \forall y \in E, \neg(y \prec x)$ *and* $\forall x \in T, \forall y \in E, \neg(x \prec y)$.

Example:
In the Fig. 1.5, we represent the same three events as in the figure 1.4 but with two interfaces. In fact, in the interval representation, the event a has not been started during the studied time period and is therefore an unstarted event. On the other hand, the event b has not been finished at the end of the time period and is therefore an unterminated event. Both events are represented with an arrow to, respectively from the interface point event.

Some special pomsets and ipomsets have no elements in relation with each other. In other words, all the events described by these pomsets are non-ordered and take effect at the same time. These pomsets are concurrent lists (*conclist*). They are used to represent concurrent events only.

Definition 9. *A concurrency list (**conclist**) is a pomset $P = (E, \prec, \lambda)$ such that*

$$\forall x, y \in E, x \not\prec y \wedge y \not\prec x$$

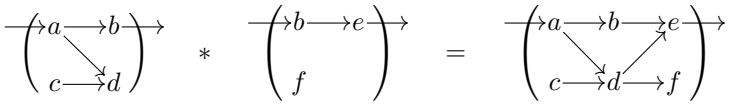
Example:
For example, $\begin{pmatrix} a \longrightarrow \\ b \end{pmatrix}$ is a conclist with an interface. On the other hand, $\begin{pmatrix} a \longrightarrow b \\ c \end{pmatrix}$ is not a conclist as we have $a \prec b$.

The gluing operation [3] is a fundamental concept in the study of ipomsets, enabling the sequential composition of complex processes from simpler sub-processes. The aim of the gluing operation is to merge two ipomsets in a way that respects the ordering constraints inherent to each ipomset and creates a coherent combined process. This operation is only defined if the terminated interfaces of the first ipomset, with the labels, are equal to the starting interfaces of the second ipomset, with the labels. The goal is to merge (or glue) those common interfaces, in order to create complete events by combining unterminated events with unstarted events. Moreover, the order relation is preserved within each ipomset. Finally, the terminated events of the first ipomset need to be achieved before the started events of the second one. In that sense, we say that the first ipomset is precedent to the second one.

Definition 10. *The **gluing composition** of two ipomsets $P = (E_P, \prec_P, S_P, T_P, \lambda_P)$, $Q = (E_Q, \prec_Q, S_Q, T_Q, \lambda_Q)$ such that $T_P = S_Q \wedge \lambda_{T_P} = \lambda_{S_Q}$ is $P * Q = (E_P \cup E_Q, \prec, S_P, T_Q, \lambda)$ where:*

- $x \prec y$ *if* $x \prec_P y$ *or* $x \prec_Q y$ *or* $(x \in P \cap \overline{T_P}) \wedge (y \in Q \cap \overline{S_Q})$
- $\lambda(x) = \begin{cases} \lambda_P(x) & \text{if} \quad x \in P \\ \lambda_Q(x) & \text{if} \quad x \in Q \end{cases}$

Example:



In fact, we are merging the interfaces labelled b to create a complete event. We are also conserving the order relation inherent to each ipomset. Finally, we are adding some relation to make the first ipomset before the second one. To achieve so, we need to add relations to make the events a, c, d precedent to the events e, f. As $a \prec d$ and $c \prec d$, and by the transitivity of the order relation, we just need to make the event d before the events e and f.

Definition 11. *Let $P_1, P_2, \ldots, P_n$ be a sequence of IPomsets. The $\circledast$ operation applied from $i = 1$ to n is defined as the gluing operation of those IPomsets in the given order. The notation $\circledast_{i=1}^{n} P_i$ represent the sequential gluing operation of the IPomsets $P_1, P_2, \ldots, P_n$. Explicitly, this can be written as:*

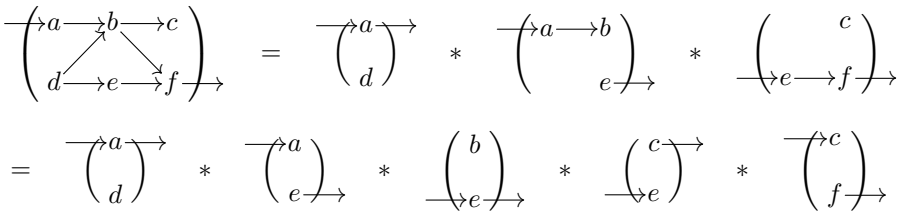
$$\circledast_{i=1}^{n} P_i = P_1 * P_2 * \ldots * P_n$$

The step decomposition of an ipomset is a method of breaking down a complex, partially ordered structure into a sequence of simpler, concurrent steps. Each step consists of a set of elements that can be executed in parallel, meaning there are no order dependencies between the elements within the same step. The overall process can be reconstructed by sequentially gluing these steps together.

Definition 12. *Let $P = (E, \prec, S, T, \lambda)$ be an IPomset. The **step decomposition** of the IPomset P is a sequence of IPomset $P_1, P_2, \ldots, P_n$ (with $P_i = (E_i, \prec_i, S_i, T_i, \lambda_i)$) such that:*

- $\bigcup_{i=1}^{n} E_i = E$
- *for each P_i, $x, y \in E_i \implies x \not\prec_i y \wedge y \not\prec_i x$ (P_i is a conclist)*
- *the IPomset P is the gluing composition of its decomposition: $P = \circledast_{i=1}^{n} P_i$*

Example:
Given the following ipomset, the step decomposition is obtain through this decomposition process:



1.3 Higher-Dimensional Automata

In the context of Higher-Dimensional Automata (HDA) [1, 2, 4], cells are fundamental components that represent various states and transitions. A 0-cell represents a state, a 1-cell represents a transition (like an edge in a graph), and higher-dimensional cells represent more complex interactions and concurrent transitions. The boundaries of these cells are lower-dimensional faces that form the structure of the cell. In an HDA, those cells represent, respectively to there dimension, the vertices, edges, faces, cubes, hypercubes, ...

Definition 13. *An n-**cell** is an element representing an n-dimensional hypercube. Let X_n be the set of all n-cell. Each n-cell is composed of a conclist of n elements and **boundaries** which are $3^n - 1$ faces of dimension $0 \leq i < n$.*

Example:
In the Fig. 1.6, there are four 0-cells: x, y, v, w representing states and the vertices of the square. We also have four edges (or 1-cells): e, f, g, h. The edge e for example is composed of two 0-cells boundaries: v, w. Finally, there is a 2-cell representing the square in Fig. 1.7 labelled q such as its boundaries are x, y, v, w, e, f, g, h ($3^d - 1 = 3^2 - 1 = 8 = |\text{boundaries}(q)|$).

An event in HDA corresponds to a transition (or an action). Each cell is composed of a list of concurrent events represented with the conclist thus there are as many concurrent events as the dimension of the cell. Therefore, there is no event for 0-cells (states), and elementary event for 1-cells (transitions / edges). An n-cell is composed of n concurrent events.

Definition 14. *Let $\square$ a set of conclists, the **events** of cells is defined with the mapping function $\mathcal{C} : X \to \square$ such as, for $x \in X_n$, $|\mathcal{C}(x)| = n$. For a conclist $U \in \square$, we write $X[U] = \{x \in X | \mathcal{C}(x) = U\}$.*

Example:
In the HDA Fig. 1.7, associated to the set of cell in Fig. 1.6, the 0-cells are not associated with events, the 1-cells represent the activation of one event, like the cell e witch is associated with the event a. Finally, the n-cells with $n \geq 2$ represent n concurrent events and are useful to represent multiple events. In our example, the cell q enables both a and b and therefore enable the conclist $\binom{a}{b}$.

In our set of cell, we have the following events and associated cells:

- $X[\emptyset] = \{x, y, v, w\}$
- $X[a] = \{e, f\}$
- $X[b] = \{g, h\}$
- $X[[\begin{smallmatrix} a \\ b \end{smallmatrix}]] = \{q\}$

Face maps are functions that describe how higher-dimensional cells are attached to lower-dimensional ones. We have the lower face maps (δ^0_A) and the upper face maps (δ^1_A) who transform the cell x into a cell where the events in A have not yet started, respectively have terminated.

Definition 15. *Let $x \in X_n$ an n-cell, for $A \subseteq \mathcal{C}(x)$ a set of events, the **face maps** of x are defined as follow:*

$$\delta^0_A, \delta^1_A : X_n \to X_{n-|A|}$$
$$\text{such as:} \quad \mathcal{C}(\delta^{\{0,1\}}_A(x)) = \mathcal{C}(x) \setminus A$$
$$\text{For a conclist} \quad U \in \square, A \subseteq U, \text{ we can write} \quad \delta^{\{0,1\}}_A : X[U] \to X[U \setminus A]$$

For $U \in \square$, $A, B \subseteq U$, such as $A \cap B = \emptyset$, for $\mu, \nu \in \{0, 1\}$, $\delta^\nu_A \delta^\mu_B = \delta^\mu_B \delta^\nu_A$

Example:
In the Fig. 1.6, each cell of dimension greater or equal to one, is linked to its face maps with labelled arrows from the studied cell to its face maps. For example, the 1-cell e representing the event a is linked to two 0-cells v, w such as v (which is the $\delta^0_a(e)$) is the cell where the event a has not yet started, and the 0-cell $w = \delta^1_a$ is the cell where the event a is finished.

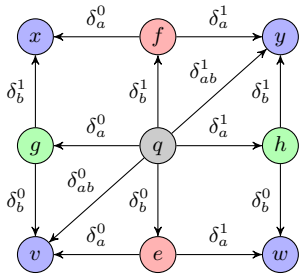


Figure 1.6: The precubical set representation of a two dimensional HDA

For the cell of dimension two (q), there is 8 boundaries but only 6 face maps: g, e, x unstart the events $a, b, [\begin{smallmatrix} a \\ b \end{smallmatrix}]$ and h, f, y finish those events (respectively).

A precubical set is a combinatorial structure that encodes higher-dimensional transitions using cells and face maps. It invokes the $\mathcal{C}$ function to map a conclist of events for each cell, indicating the events involved. Each cell also uses the face maps δ^0 and δ^1 providing a way to navigate between different dimensions of cells, by starting or finishing events, thus capturing the relationships between various states and transitions in a concurrent system. This union of all the face maps of a cell represent a part of its boundaries.

Definition 16. *Let X be the set of cells, $\square$ the set of conclist, the **precubical set** is:*

$$X = (X, \mathcal{C}, \{\delta_A^0, \delta_A^1 | U \in \square, A \subseteq U\})$$

with:

- $\mathcal{C} : X \to \square$ *mapping a conclist for all cells.*
- $\delta_A^{\{0,1\}}$ *for $U \in \square$, $A \subseteq U$, the face maps of cells $x \in X_{|U|}$ such as $\mathcal{C}(x) = U$.*

The Fig. 1.6 represent a precubical set, with all the elements already studied in previous examples.

An HDA extends the concept of traditional automata to higher dimensions by using a precubical set. The start cells represent initial states from which the computation begins, and the accept cells represent the final states where the computation can successfully terminate.

Definition 17. *Let X be the precubical set, the **HDA** H is defined as follow:*

$$H = (X, \perp_X, \top_X)$$

with: $\perp_X, \top_X \subseteq X$ the start and accept cells.

Example:
The Fig. 1.7 represents the HDA associated to the studied precubical set in figure 1.6. We can notice the different cells with q in the gray square representing the 2-cell. We can also notify the events of each edge, such as the 2-cell square execute events of its boundaries in concurrence. Boundaries of a n-cell with $n \geq 2$ can be paired such as there are opposite and identified by the same events. For the boundaries of the 2-cell q, we can pair (f, e) as well

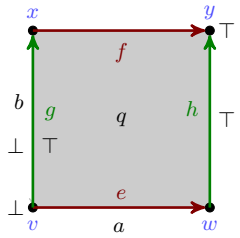


Figure 1.7: HDA associated to the precubical set in Fig. 1.6

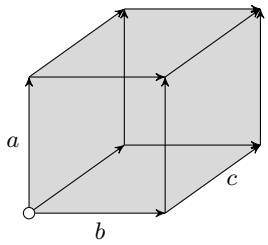


Figure 1.8: A three dimensional HDA

as (g, h) as they both activate the event a, respectively b. In this HDA, we represented the starting and accepting cells with the $\perp$ and $\top$ sign. We have the starting cells $\perp_X = \{v, g\}$ and the finishing cells $\top_X = \{g, h, y\}$.

The HDA in Fig. 1.8 is a cube. It therefore represents three concurrent events: a, b, c.

1.4 Languages

1.4.1 ST-Traces

ST-Traces are one of the formal languages of Petri nets [4, 6, 7]. It is a set of execution traces. Those execution traces represent the activation sequence of the transitions of the Petri net. In other words, those traces are a succession of transitions, indexed if multiple, with a plus or a minus sign to indicate the start or the end of the activation of the transition.

Definition 18. *Let N be a Petri net and* $\text{RUN} : N \to \Sigma^*$ *a non-deterministic function that associate to the Petri net N an activation sequence.*

$$ST(N) = \{\text{RUN}(N)\}$$

Example:
For the Petri net in Fig. 1.1, the ST-traces are:

$$ST(N) = \begin{cases} a^+a^-b_1^+b_1^-b_2^+b_2^- \\ a^+a^-b_1^+b_2^+b_1^-b_2^- \\ a^+a^-b_1^+b_2^+b_2^-b_1^- \\ a^+a^-b_2^+b_2^-b_1^+b_1^- \\ a^+a^-b_2^+b_1^+b_2^-b_1^- \\ a^+a^-b_2^+b_1^+b_1^-b_2^- \\ a^+b_1^+b_1^-a^-b_2^+b_2^- \\ a^+b_1^+a^-b_1^-b_2^+b_2^- \\ a^+b_1^+a^-b_2^+b_1^-b_2^- \\ a^+b_1^+a^-b_2^+b_2^-b_1^- \\ b_1^+b_1^-a^+a^-b_2^+b_2^- \\ b_1^+a^+b_1^-a^-b_2^+b_2^- \\ b_1^+a^+a^-b_1^-b_2^+b_2^- \\ b_1^+a^+a^-b_2^+b_1^-b_2^- \\ b_1^+a^+a^-b_2^+b_2^-b_1^- \end{cases}$$

Where each line represent a possible run of the Petri net. The indexing is very important, especially in the case of auto-concurrency, as we want to separate the cases where we end the first transition we activated or the second for example.

1.4.2 Language of HDA

In the context of HDA, a path is a continuous mapping from a time interval to the automaton's cells.

Definition 19. *Let H be an HDA and X the set of cells of H, a **path** in that HDA is defined with:*

$$\gamma : [0, 1] \to X$$

where $\gamma(0) \in \perp_X$ and $\gamma(1) \in \top_X$.

Example:
In the Fig. 1.9, the orange edges represent a possible path. This path starts in the only one initial cell in the upper left corner and finishes in the only one accepting cell in the lower right corner. It goes through the first 2-cell square associated to the conclist of events $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ and then go through a 1-cell edge identifying the event b.

These paths can be identified by pomsets such that the ipomset identifying a path is the result of the gluing composition of all the conclists of the cells traversed along the path. A pomset can identify multiple paths. In an HDA, a pomset denoting a possible path through it is a part of the language of that HDA. Therefore, the language of an HDA is the set of all possible pomset representing a valid path [1, 2].

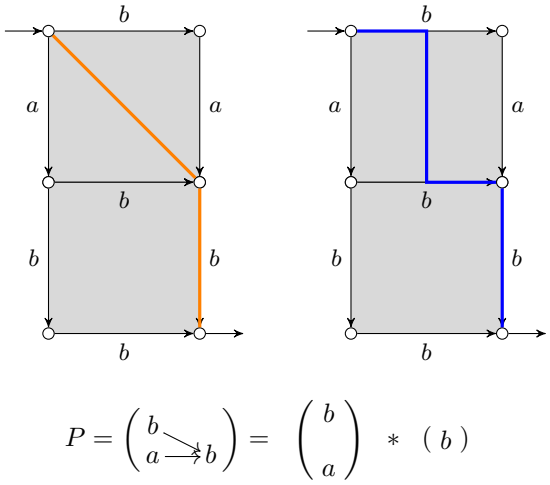


Figure 1.9: A possible path through an HDA

Definition 20. *Let $\gamma : [0,1] \to X$ one path over the HDA H. The IPomset associated to this path is:*

$$P = \bigstar_{i \in [0,1]} \mathcal{C}(\gamma(i))$$

Example:

In the Fig. 1.9, the orange path can be associated with the pomset $\begin{pmatrix} b \\ a \longrightarrow b \end{pmatrix}$ by applying the gluing composition to the conclists of the traversing cells: the conclist of the 2-cell $\begin{pmatrix} a \\ b \end{pmatrix}$ is glued to the conclist of the 1-cell $(\ b\ )$ traversed along the path. This same pomset can also be used to define a multitude of paths, such as the blue path on the neighboring HDA. Finally, a finite HDA is defined by a finite set of pomsets forming its language. The pomsets of the language of the HDA in Fig. 1.9 are all those (and only those) in figure 1.10.

Even though the last 2 cases in Fig. 1.10 are the same according to the transition labels, since they are not the same transition, we want to separate them (by indexing them, for example) so that we can differentiate between them. Thus, we have $(\ a \longrightarrow b_1 \longrightarrow b_2\ )$ and $(\ a \longrightarrow b_2 \longrightarrow b_1\ )$ which are 2 distinct cases.

Example:

If an n-cell, with $n \geq 1$ is either an initial or an accepting cell, then the pomset denoting a path starting, respectively finishing at this cell, is an ipomset with a starting interface, respectively a terminating interface. In the Fig. 1.11, the path terminates on an edge denoting the event b. In the corresponding ipomset, the event labelled b is not terminated ant is therefore a terminating interface.

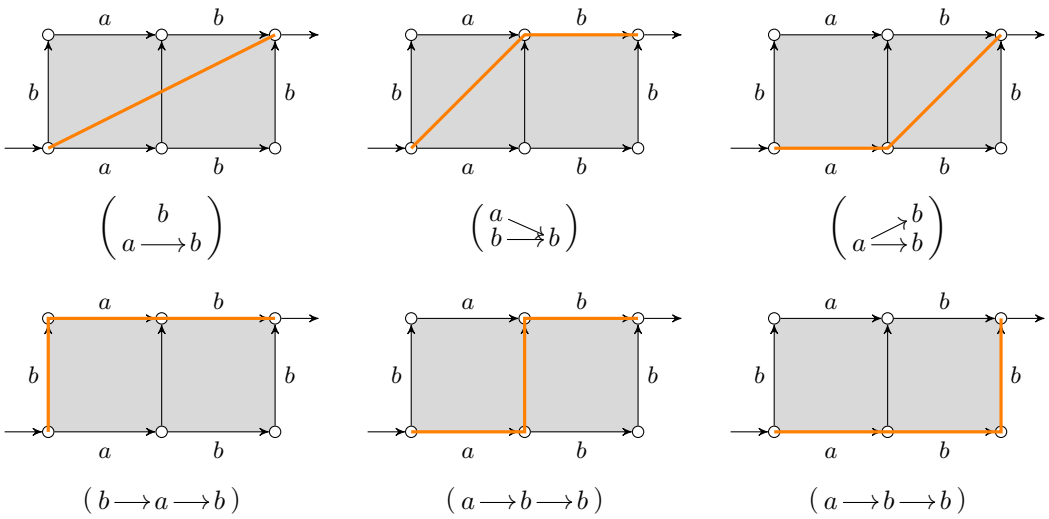
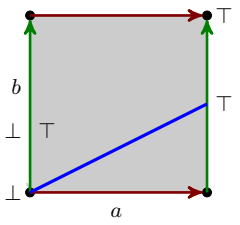


Figure 1.10: All pomset in the language of the HDA in Fig. 1.9



$$P = \begin{pmatrix} b \\ a \end{pmatrix}$$

Figure 1.11: A Path associated to an IPomset

This language exhibits a number of interesting properties for HDAs. It permits a decomposition whereby each ipomset is a conclist of one of the traversed cells.

1.4.3 Link between ST-traces and Pomsets

To ensure the coherent and similar behavior of the HDA resulting from the conversion of a Petri net, it is important to define a link between their languages (ST-traces and pomsets). We will therefore seek to link pomsets to execution traces by defining pomset traces as a set of all possible start and end orders of pomset events.

Definition 21. *Let $P = (E_P, \prec_P, \lambda_P, S_P, T_P)$ be a IPomset with $\lambda_P : E_P \to \Sigma_P$. The execution traces over that pomset are:*

$$ST(P) = \{w \in \Sigma_{ST}^{|w|}\}$$

with:

- $|w| = 2 \times |E_P| - |S_P| - |T_P|$
- $\Sigma_{ST} = (\Sigma_P^+) \cup (\Sigma_P^-)$
- $\forall x \in E_P, \forall w \in ST(P), x^- x^+ \notin \Pi_{\{x^+, x^-\}}(w)$
- $\forall x, y \in E_P, \forall w \in ST(P), x \prec_P y \iff \Pi_{\{x^-, y^+\}}(w) = x^- y^+$

with $\Pi_I(w)$ the projection of the elements in I from the ST-trace w conserving the order.

Example:

The execution traces of the pomset $P = \begin{pmatrix} a \searrow \\ b \to c \end{pmatrix}$ are the following:

$$ST(P) = \begin{cases} a^+b^+a^-b^-c^+c^- \\ a^+b^+b^-a^-c^+c^- \\ b^+a^+b^-a^-c^+c^- \\ b^+a^+a^-b^-c^+c^- \end{cases}$$

In fact, we want the order relation to be maintained as $a \prec c$ and $b \prec c$ but as there is no order relation between a and b. In that sense, we must finish a and b before starting c but but neither of the 2 events (a or b) can be completed before starting the other. In the latter case, there would be an order relationship between the 2.

In the context of HDA, the blue path of the HDA in the Fig. 1.9 can easily be decomposed such as we start a b transition, we make a complete a transition, we finish our firstly started b transition, and we finally achieve a b event. The ST-traces of that path, according to the given description, is $b^+a^+a^-b^-b^+b^-$.

An HDA therefore behaves in a similar way to the source Petri net if, for each pomset in the language of the HDA, its execution traces are included in the set of execution traces for the Petri net, and if each execution trace in the source Petri net finds a pomset in the resulting HDA language that generates the same trace.

Definition 22. *Let H be an HDA resulting from the conversion of the Petri net N:*

$$P \in \mathcal{L}(H) \iff ST(P) \subseteq ST(N)$$

similarly: $\bigcup ST(P_i \in \mathcal{L}(H)) = ST(N)$

Chapter 2

Algorithm

The conversion algorithm [4] aim to generate all possible reachable marking and execution traces from a Petri net and to map an HDA cell for each of them.

Definition 23. *Let $N = (S, T, F, M_0, l)$ be a Petri net. The HDA $H = (X = (X, \mathcal{C}, \{\delta^{\{0,1\}}\}), \bot_X, \top_X)$ is given by:*

- $X_n = \{x \in ST(N) | \, |x| - |\{y^- \in x\}| = n\}$
- $\delta^0_A(x \in X_n) = \{y \in X_{n-|A|} | \, \mathcal{C}(y) = \mathcal{C}(x) \setminus A, \mathcal{A}(\mathcal{M}(y), \{t^+_k | k \in A\}) = \mathcal{M}(x)\}$
- $\delta^1_A(x \in X_n) = \{y \in X_{n-|A|} | \, \mathcal{C}(y) = \mathcal{C}(x) \setminus A, \mathcal{A}(\mathcal{M}(x), \{t^-_k | k \in A\}) = \mathcal{M}(y)\}$
- $\bot_X = \{x \in X_0 | \, \mathcal{M}(x) = M_0\}$
- $\top_X = \emptyset$
- $\mathcal{C}(x \in X_n) = \{l(c) | \, x \in ST(N), c \in x \setminus \{y^- \in x\}\}$

with:

- $\mathcal{M} : X \to M$ *a mapping from cells of HDA to the corresponding marking M when the cell was created.*
- $\mathcal{A} : M \times T^n \to M$ *the successive activation of the given n transitions from the given marking.*

The algorithm outline is as follow:

- **Initialization**: Load the Petri net and its initial marking M_0 and initialize the HDA with the initial cell corresponding to the marking M_0.
- **Generate reachable markings**: For each transition in the Petri net, determine if it is enabled based on the current marking. If the transition is enabled, compute the new marking reached after the transition has been activated. Create a new HDA cell corresponding to this new marking.
- **Create HDA cells**: For each reachable marking, create an HDA cell (edge) associated with the transition activated. Define the boundaries for each newly created cell. Ensure that the boundaries respect the partial order of events, maintaining the causal relationships.
- **Construct higher-dimensions**: For concurrent events, construct higher-dimensional cells (2-cells, 3-cells, ...). Each higher-dimensional cell represents the simultaneous execution of

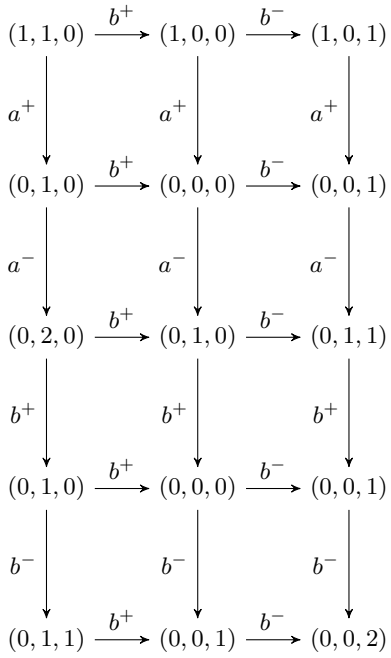


Figure 2.1: Generate cells from Petri net execution of Fig. 1.1

multiple transitions. Create boundaries for each of this newly created higher-dimensional cell.

- **Finalize HDA**: Define the initial and accepting cells of the HDA. Ensure all execution traces from the Petri net are captured in the HDA structure.

Example:
The Petri net in Fig. 1.1 can be converted in the HDA in Fig. 1.9 enumerating the tuples marking, execution traces as shown in figure 2.1. In the last figure, we can distinguish the 2 squares with each one cell (marking) in their center. Even if this marking is the same ($(0, 0, 0)$), the events are not the same ($\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ for the upper one and $\left[\begin{smallmatrix} b \\ b \end{smallmatrix}\right]$ for the lower one).

The theoretical complexity of the conversion algorithm can be analyzed in terms of the number of markings and the dimensions of the HDA cells.

Markings Let n be the number of places in the Petri net. The number of reachable markings can be exponential in the number of places, up to $\mathcal{O}(2^n)$ in the worst case.

Cells Let m be the number of transitions in the Petri net. For each reachable marking, a cell is created in the HDA. The creation of higher-dimensional cells involves combinatorial complexity. Specifically, for k concurrent events, the number of higher-dimensional cells is 3^k, leading to a total complexity of $\mathcal{O}(3^m)$.

Thus, the overall complexity is dominated by the exponential growth in the number of reachable marking and the creation of higher-dimensional cells, making the worst-case complexity $\mathcal{O}(2^n \dot{3}^m)$.

Chapter 3

Implementation

3.1 Conversion algorithm

The initial step is to delineate the various objects that will be employed in the algorithm. There are five distinct types. The five types of objects are as follows: HDA, cell, marking, transition, and PN.

transition A transition in a Petri net is represented by two sets, one associated with the preset and the other with the postset of that transition, as well as a label that represents the event name during its activation.

- **preset** (`vector<integer>`): the set of indexes of the preset places.
- **postset** (`vector<integer>`): the set of indexes of the postset places.

PN A Petri net (PN) is represented by two sets: the places and the transitions. Each element in each vector is indexed by its position, allowing for a unique description in subsequent operations.

- **places** (`vector<place>`): a set of indexed places in the Petri net.
- **transitions** (`vector<transition>`): a set of indexed transitions in the Petri net.

Marking A marking is defined as a vector of integers (`vector<integer>`). Each element of this vector, indexed by i, represents the number of tokens available at the place represented by the unique identifier i.

cell A cell is made up of a conclist of events active in that cell, and two sets of cells, which form the cell's face maps (and its boundaries). The cells in the face maps of the current one are those with dimensions exactly equal to the dimension of the current cell minus one.

- **dim** (`integer`): the dimension of the current cell.
- **conclist** (`vector<transition>`): the list of events active in that cell.

- **d0** (`vector<cell>`): the list of the lower face maps of that cell (representing δ^0) such as $\forall c \in d0$, dim of c $=$ (dim of current cell) - 1.
- **d1** (`vector<cell>`): the list of the upper face maps of that cell (representing δ^1) such as $\forall c \in d1$, dim of c $=$ (dim of current cell) - 1.

HDA HDAs are just a list of cells (`vector<cell>`) representing all the cells forming that HDA, such as all cells are accepting and only the first one cell, associated to the initial marking of the Petri net, is an initial cell.

I decided to implement the algorithm in C because that's the language I was most comfortable with, and also because I tried to base myself on Thomas Kahl's PG2HDA program for HDA representation (in memory).

Algorithm 1 PN2HDA

Require: out: HDA, PN: PN, save: Hashmap<Marking, vector<cell>>, currentConclist: vector<transition>, M0: Marking
Ensure: c : cell

```
1:  procedure PN2HDA(out, PN, save, currentConclist, M0)
2:      c ← cell(dim: length(currentConclist), conclist: currentConclist)
3:      add (M0, c) in save
4:      add c in out
5:      for all t ∈ transitions of PN do
6:          if t is enabled then
7:              M ← start_activate t from M0
8:              if M ∈ save  &  ∃ c' ∈ save[M], conclist of c' = (conclist of c) + t then
9:                                  ▷ If conclists are the same, their dimensions also match as the conclist
    contains dim elements
10:                 add c in d0 of c'
11:             else
12:                 add c in d0 of PN2HDA(out, PN, save, currentConclist + t, M)
13:             end if
14:         end if
15:     end for
16:     for all t ∈ currentConclist do
17:         M ← end_activate t from M0
18:         if M ∈ save  &  ∃ c' ∈ save[M], conclist of c' = (conclist of c) - t then
19:                                 ▷ We do not compare their dimensions for the same cause as above
20:             add c' in d1 of c
21:         else
22:             add PN2HDA(out, PN, save, currentConclist - t, M) in d1 of c
23:         end if
24:     end for
25:     return c
26: end procedure
```

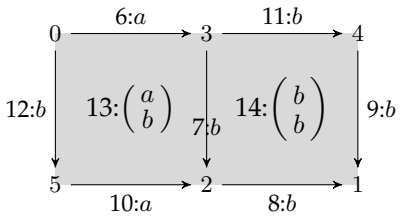


Figure 3.1: HDA representation of the software output in Fig. 2

3.2 Software input: PNML

In this software, we take as entry a standardized PNML (Petri Net Marking Language) file [5] describing a Petri net under an XML format. We can distinguish states (with the initial marking), transitions and arcs. An example is provided in the annexes Fig. 1 for the Petri net in Fig. 1.1.

3.3 Software output format

3.3.1 Current format

You can discover, in annexes Fig. 2, the output HDA from the software. It is corresponding to the HDA in Fig. 1.9. For a shake of readability, you can find its representation in Fig. 3.1.

For now, due to a lack of time, and because we want to make the algorithm working fine before going through new steps, the output of the software is pretty simple and assume the first cell is the only one inital and all cells are accepting.

3.3.2 Standardised format

The ideal format we want to introduce to represent HDAs is based on the YAML (Yet Another Marking Language) file format, as it was designed to be structured, human-readable and easily parsable and writable.

HDA Under that format, we want the HDA to be represented with:

- **name**: an optional string providing a name for the HDA.
- **description**: an optional string offering a brief description of the HDA.
- **cells**: a list of cell objects, representing the vertices, edges, and higher-dimensional cells of our HDA.
- **initial**: a list of integers representing the IDs of the initial cells of our HDA.
- **accepting**: a list of integers representing the IDs of the accepting cells of our HDA.

cell In that HDA, each cell is composed of the following fields:

- **id**: a unique integer identifier for the cell.

- **conclist**: a list of strings denoting the concurrent events. This is absent for vertices.
- **d0**: a list of integers containing the IDs of the lower face maps (δ^0) of the current cell. This is absent for vertices.
- **d1**: a list of integers containing the IDs of the upper face maps (δ^1) of the current cell. This is absent for vertices.

In the file 1, there is the detailed wanted format file to modelise an HDA. File 2 presents the HDA in Fig. 3.1, associated to the actual output in Fig. 2 in the wanted YAML format. It is really similar to the output we get in the Fig. 2 and the process to transform the output we currently have to the presented format we want will be very straightforward.

Chapter 4

Limits of the Algorithm

The conversion algorithm, while theoretically sound, has several limitations.

- **Scalability**: The exponential growth in the number of reachable markings and higher-dimensional cells limits the scalability of the algorithm. For large Petri nets with many places and transitions, the resulting HDA can become unmanageably large.
- **Computational resources**: The algorithm requires significant computational resources (memory and processing power) to store and process the large number of cells, especially for high-dimensional constructs.
- **Unbounded Petri nets**: The program execution time must be bounded, and the output HDA cannot, for obvious technical reasons, be infinite. As we want to work on finite sets and finite HDA, we cannot generate a potentially infinite HDA. This is mainly why we cannot work on unbounded Petri nets as they lead to infinite execution traces and configuration.

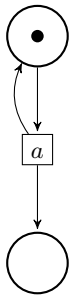


Figure 4.1: An unbounded Petri net

Example:
In the Fig. 4.1, there is an example of an unbounded Petri net. In that case, there is always a token in the first place. Thus, we can always activate a transition a to generate as many tokens as we want in the last place. This leads to an infinite number of executions.

Conclusion

In conclusion, this research presents an advance in the modilization of concurrency systems, offering an approach to bridge the gap between Petri nets and higher-dimensional automata. By developing and implementing a conversion algorithm, we have been able to leverage the strengths of both formalisms, thereby offering a more detailed representation of concurrent systems. The establishment of a formalism for HDAs addresses the absence of a standard in this field, providing a foundation for future research and development. Moving forward, further improvements to the algorithm and further exploration of additional applications will continue to enhance our understanding and utilization of these powerful modeling tools.

Chapter 5

Bibliography

[1] Amrane, A., Bazille, H., Fahrenberg, U., and Ziemiański, K. (2023). Closure and decision properties for higher-dimensional automata. (pages 9, 12, and 16)

[2] Fahrenberg, U., Johansen, C., Struth, G., and Ziemianski, K. (2021a). Languages of higher-dimensional automata. (pages 9, 12, and 16)

[3] Fahrenberg, U., Johansen, C., Struth, G., and Ziemianski, K. (2021b). Posets with interfaces for concurrent kleene algebra. (pages 10 and 11)

[4] Glabbeek, R. (2006). On the expressiveness of higher dimensional automata. (pages 5, 6, 12, 15, and 21)

[5] Hillah, L., Kindler, E., Kordon, F., Petrucci, L., and Trèves, N. (2009). A primer on the petri net markup language and iso/iec 15909-2. (page 26)

[6] Kuske, D. and Morin, R. (2002). Pomsets for local trace languages. (page 15)

[7] Peterson, J. L. (1981). Petri net theory and the modeling of systems. (pages 6 and 15)

Appendix PNML file

File: **examples/auto-concurrent-example.pnml**

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml xmlns="http://www.pnml.org/version-2009/grammar/pnml">
<net id="AutoConcurrency" type="http://www.pnml.org/version-2009/grammar/ptnet">
<name><text>AutoConcurrency</text></name>
<page id="page">
    <place id="p0"><initialMarking><text>1</text></initialMarking><name><text>p0</text></name></place>
    <place id="p1"><initialMarking><text>1</text></initialMarking><name><text>p1</text></name></place>
    <place id="p2"><name><text>p2</text></name></place>
    <transition id="t0"><name><text>a</text></name></transition>
    <transition id="t1"><name><text>b</text></name></transition>
    <arc id="a0" source="p0" target="t0"/>
    <arc id="a1" source="t0" target="p1"/>
    <arc id="a2" source="p1" target="t1"/>
    <arc id="a3" source="t1" target="p2"/>
</page>
</net>
</pnml>
```

Figure 1: PNML file for Petri net in Fig. 1.1

Appendix output HDA

```
[09:18] AyAztuB@PN2HDA$ ./build/pn2hda --logs NO --print_hda examples/auto-concurrent-example.pnml
cells:
0: dim=0,
1: dim=0,
2: dim=0,
3: dim=0,
4: dim=0,
5: dim=0,
6: dim=1:       [a]; d0: [0]; d1: [3],
7: dim=1:       [b]; d0: [3]; d1: [2],
8: dim=1:       [b]; d0: [2]; d1: [1],
9: dim=1:       [b]; d0: [4]; d1: [1],
10: dim=1:      [a]; d0: [5]; d1: [2],
11: dim=1:      [b]; d0: [3]; d1: [4],
12: dim=1:      [b]; d0: [0]; d1: [5],
13: dim=2:      [a, b]; d0: [6, 12]; d1: [7, 10],
14: dim=2:      [b, b]; d0: [7, 11]; d1: [8, 9]
```

Figure 2: Software output HDA

Appendix YAML format to represent HDA

Source Code 1: HDA standard format description

```yaml
HDA:
    name: <optional string>           # Name of the HDA
    description: <optional string>    # Description of the HDA

    cells:                            # List of cells in the HDA
        - id: <integer>               # Unique identifier for the cell
          conclist: <list of strings> # List of concurrent events, optional
          d0: <list of integers>      # Lower face maps of the current cell, optional
          d1: <list of integers>      # Upper face maps of the current cell, optional

    initial: <list of integers>       # List of initial cells IDs
    accepting: <list of integers>     # List of accepting cells IDs
```

Appendix YAML format example to represent the HDA in Fig. 2

Source Code 2: HDA standard format example

```yaml
HDA:
    cells:
        - id: 0
        - id: 1
        - id: 2
        - id: 3
        - id: 4
        - id: 5
        - id: 6
          conclist: ["a"]
          d0: [0]
          d1: [3]
        - id: 7
          conclist: ["b"]
          d0: [3]
          d1: [2]
        - id: 8
          conclist: ["b"]
          d0: [2]
          d1: [1]
        - id: 9
          conclist: ["b"]
          d0: [4]
          d1: [1]
        - id: 10
          conclist: ["a"]
          d0: [5]
          d1: [2]
        - id: 11
          conclist: ["b"]
          d0: [3]
```

```
          d1: [4]
        - id: 12
          conclist: ["b"]
          d0: [0]
          d1: [5]
        - id: 13
          conclist: ["a", "b"]
          d0: [6, 12]
          d1: [7, 10]
        - id: 14
          conclist: ["b", "b"]
          d0: [7, 11]
          d1: [8, 9]

    initial: [ 0 ]
    accepting: [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```