Towards an efficient conversion of Petri nets into Higher Dimensional Automata

Amazigh Amrane[1], Hugo Bazille[1],
Timothée Fragnaud[1], and Philipp Schlehuber-Caissier[2]*

[1] EPITA Research Lab (LRE), France
[2] SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France

Abstract

Higher-dimensional automata (HDAs) extend classical automata by explicitly modeling concurrency through n-dimensional cells, each representing n parallel events. Thanks to the cubical structure of HDAs, these cells implicitly determine their lower-dimensional faces via face maps, capturing all their interconnections.

In recent years, interest in HDAs has grown significantly, in particular as they generalize most concurrent formalisms. In a recent contribution, we proposed translations from Petri nets, including common features like weights, inhibitor arcs and even self-modifying nets to HDAs, along with a working implementation. However the representation and data structures retained in this work are directly drawn from the mathematical description of HDAs and do not take advantage of the HDA's structure.

In this paper, we take initial steps toward leveraging this implicit structure. We propose what we call the max-cell representation of an HDA: instead of storing all cells, we retain only those that are not induced by higher-dimensional ones. This can lead to an exponential reduction in the number of stored cells and face maps while preserving all the information needed to reconstruct the full HDA. To demonstrate its effectiveness, we present a new algorithm that directly translates Petri nets into max-cell HDAs.

1 Introduction

Petri nets and HDAs are two operational model that aim at expressing concurrency between events. In both formalisms, events may occur simultaneously. Besides, both make a distinction between parallel composition $a \parallel b$ and choice $a\cdot b+b\cdot a$. However, Petri nets are often considered through their interleaving semantics, thus reducing their expressive power and therefore effectively reducing $a \parallel b$ to $a\cdot b+b\cdot a$. As an example, Figure 1 shows two Petri nets and their HDA semantics: on the left, transitions a and b are mutually exclusive (as ensured by the token in p_5) and may be executed in any order but not concurrently; on the right, there is true concurrency between a and b, signified by the filled-in square of the HDA semantics. In interleaving semantics, no distinction is made between the two nets and both give rise to the transition system on the left.

To tackle this issue, [12] introduced concurrent step-semantics. In this semantics, a limited form of parallelism is introduced: two parallel events must start simultaneously and end simultaneously. While the expressivity is strictly greater than when only considering interleavings, it is shown that some possible behaviours were missed [5]. We illustrate this in Figure 2. In the Petri net, a and b can be fired concurrently. Besides, firing b enables a second firing of b. The behaviour of concurrent step-semantics is given by the graph on the right. However, it misses the possibility that the second a starts *while* the first a is running, as represented by the additional trajectory.

*Partially funded by the Academic and Research Chair “Architecture des Systèmes Complexes” Dassault Aviation, Naval Group, Dassault Systèmes, KNDS France, Agence de l’Innovation de Défense, Institut Polytechnique de Paris

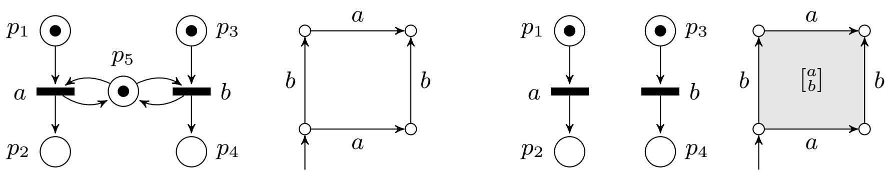


Figure 1: Petri nets and HDAs for interleaving (left) and true concurrency (right).

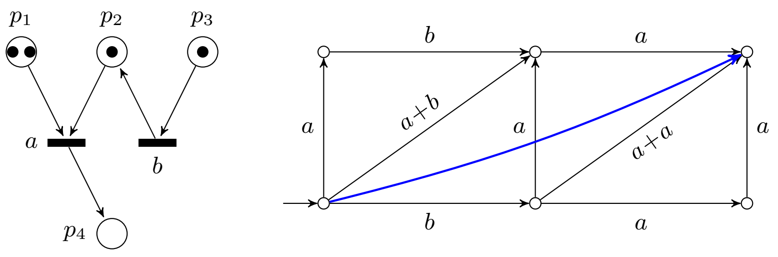


Figure 2: Concurrent step-semantics miss some possible behaviours

Van Glabbeek [16] first explored the relations between Petri nets and HDAs, where an HDA is defined as a labeled precubical set whose cells are hypercubes of different dimensions. More recently, [9] introduces an event-based setting for HDAs, defining their cells as totally ordered sets of labeled events. This framework has led to a number of new developments in the field of languages and logics for HDAs: a Kleene theorem [10] which relates HDAs and regular expressions; a Myhill-Nerode theorem [11] which constructs HDAs out of a prefix equivalence on regular languages; a proof of decidability of language inclusion [6], a notion of monadic second-order logic for HDAs and a corresponding Büchi-Elgot-Trakhtenbrot theorem, [2, 4], a notion of temporal logic for HDAs and corresponding Kamp's theorem [7] based on [3] relating HDAs to a subclass of standard automata, and an extension of HDAs and timed automata called HDTAs was explored in [1].

Concretely, HDAs are composed of cells, each associated with a list (called *conclist*) of events that are active in them. A 0-dimensional cell corresponds to a states with no active event, a 1-dimensional cell represents a transition (as in standard automata) with exactly one active event, and an n-dimensional cell captures concurrent behaviours involving n active events. These cells are interconnected through face maps. As an example, the HDA on the right in Figure 1 includes one two-dimensional cell, that we denote x, in which a and b (hence $[\begin{smallmatrix}a\\b\end{smallmatrix}]$) are active. It also contains two 1-dimensional cells where a is active, two where b is active and four 0-dimensional cells. Lower-dimensional cells (or faces) are connected to higher-dimensional ones via lower and upper face maps, which specify when individual events start or terminate. For instance, the lower face $\delta_b^0(x)$ of x corresponds to the lower a transition where b is not yet started, while the upper face $\delta_b^1(x)$ represents the upper a-transition in which b is terminated. Similarly, $\delta_a^0(x)$ is the left b-transition and $\delta_a^1(x)$ is the right b-transition. The composition $\delta_a^0\delta_b^0(x) = \delta_b^0\delta_a^0(x)$ identifies the bottom left 0-dimensional cell and $\delta_a^1\delta_b^1(x) = \delta_b^1\delta_a^1(x)$ corresponds to the upper right 0-dimensional cell.

In [5], Van Glabbeek's translation from Petri nets to HDAs [16] was adapted to the event-based setting and extended to cover additional Petri net semantics, such as nets with inhibitor arcs. The paper also provided an implementation of these translations. However, this implementation was carried out in a naive manner. For the first version of the `pn2HDA` tool, we followed the mathematical definitions as closely as possible. This means to explicitly define each cell of the HDA and all of the face maps between them, leading to an explosion of the memory required to store the HDA. Indeed, an n-dimensional cell (roughly corresponding to a hypercube of dimension n) has in total 3^n cell of dimension at most n. More generally, this base version performs poorly—even in comparison to standard explicit Petri net tools. As shown in [5], the standard reachability graph of a Petri net corresponds to the restriction of the resulting HDA to its 0-dimensional cells. This means that this base version is systematically at a disadvantage, both in terms of time and memory usage. Moreover, each cell in the HDA must store more information, not only the marking, but also the multiset of events that are active within it. In contrast, classical tools only store markings for places and source/target data for steps.

In this work, we extend the initial implementation by addressing its naive aspects. We adapt the translation algorithm from Petri nets to HDA to retain, on the fly, only the strictly necessary cells: those of maximal dimension, which in turn allow us to deduce all the *implied* lower-dimensional cells. For instance, storing a single 2-dimensional cell where a and b are active is sufficient to reconstruct the entire HDA depicted on the right in Figure 1. For each pair of maximal cells, we also store multisets of events that enable reaching a shared face, so that the full structure can be reconstructed. Indeed, each maximal cell induces an HDA on its own, and the multisets associated with shared faces allow to identify common cells between these local HDAs and reassemble the complete one. Figure 3 highlights the number of cells saved by our approach, compared to the naive construction.

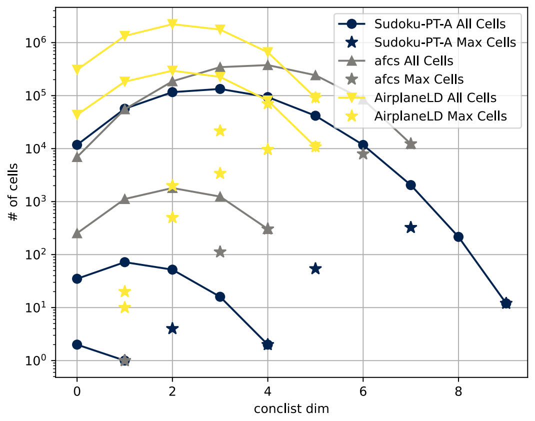


Figure 3: Comparison number of cells vs max-cells, examples taken from mcc.

This article is organised as follows. We begin in Sect. 2 and 3 by recalling HDAs and Petri nets, focusing on their concurrent semantics which allows several transitions to fire concurrently. The following sections present our proper contributions. We give a brief overview of the translation from [5] in Sect. 4, followed by the improvements we brought. These are summarized in Sect. 5, with an updated algorithm alongside examples. Finally, we evaluate this algorithm in Sect. 6 to illustrate the (often exponential) reduction in the size of the representation.

2 Higher-Dimensional Automata

Higher-Dimensional Automata (HDA) form a class of models which extend finite automata. They provide extra structure that allows one to specify independence or concurrency of events. They are constituted of cells connected by face maps. Each cell contains a (ordered) list of events representing the active events, and face maps terminate (upper maps) or “unstart” (lower maps) some events.

More formally, let Σ be an alphabet. A concurrency list (*conclist*) is a tuple $(U, \dashrightarrow, \lambda)$ with U a finite set of events, $\dashrightarrow \subset U \times U$ a total order called the *event order*, and a labelling function $\lambda : U \to \Sigma$. They represent labelled events running in parallel. For the sake of briefness, when no ambiguity arises, a conclist will be denoted by its underlying set, *i.e.*, denoting U instead of $(U, \dashrightarrow, \lambda)$ (and similarly for other structures defined along this paper). We denote by $\square(\Sigma)$ (or $\square$ when the context is clear) the set of conclists over Σ.

A *precubical set* $\big(X, \mathsf{ev}, \{\delta_{A,B;U} \mid U \in \square, A, B \subseteq U, A \cap B = \emptyset\}\big)$ consists of a set of *cells* X together with a function $\mathsf{ev} : X \to \square$. For a conclist U we write $X[U] = \{x \in X \mid \mathsf{ev}(x) = U\}$ for the cells of type U. Further, for every $U \in \square$ and $A, B \subseteq U$ with $A \cap B = \emptyset$ there are *face maps*

$\delta_{A,B;U}: X[U] \to X[U \setminus (A \cup B)]$ which satisfy the precubical identity

$$\delta_{C,D;U\setminus(A\cup B)}\delta_{A,B;U} = \delta_{A\cup C,B\cup D;U} \tag{1}$$

for every $U \in \square$, $A,B \subseteq U$, and $C,D \subseteq U \setminus (A \cup B)$.

We will often omit the extra subscript "U" in the face maps and often write δ^0_A for $\delta_{A,\emptyset}$ and δ^1_B for $\delta_{\emptyset,B}$. The *upper* face maps δ^1_B transform a cell x into one in which the events in B have terminated; the *lower* face maps δ^0_A transform x into a cell where the events in A have not yet started. These cells are sometimes called *subcells* of x. Every face map $\delta_{A,B}$ can be written as a composition $\delta_{A,B} = \delta^0_A \delta^1_B = \delta^1_B \delta^0_A$, and the *precubical identity* (1) expresses that these transformations commute.

As mentioned above, we associate to each cell of a precubical set a set events totally ordered by the event order: a conclist, instead of a multiset. Conclists are hence lists or words of Σ^*, but we often write them vertically to emphasize that the elements are running in parallel. This ordering is crucial in the presence of autoconcurrency, where multiple events may share the same label, as it allows us to determine which events are unstarted or terminated by the face maps. Formally, event order is needed to ensure uniqueness of conclist isomorphisms [9]: two conclists $(U_1, \dashrightarrow_1, \lambda_1)$ and $(U_2, \dashrightarrow_2, \lambda_2)$ are isomorphic if there exists a bijective mapping φ such that $a \dashrightarrow_1 b$ iff $\varphi(a) \dashrightarrow_1 \varphi(b)$ and $\lambda_2 \circ \varphi = \lambda_1$. Nevertheless, this technical detail is not central to the present work: since the event order has no computational significance, we often omit it and assume that it goes downwards.

We write $X_n = \{x \in X \mid |\mathsf{ev}(x)| = n\}$ for $n \in \mathbb{N}$ and call elements of X_n *n-cells*. The *dimension* of $x \in X$ is $\dim(x) = |\mathsf{ev}(x)| \in \mathbb{N}$; the dimension of X is $\dim(X) = \sup\{\dim(x) \mid x \in X\} \in \mathbb{N} \cup \{\infty\}$. For $k \in \mathbb{N}$, the *k-truncation* of X is the precubical set $X^{\leq k} = \{x \in X \mid \dim(x) \leq k\}$ with all cells of dimension higher than k removed.

A *higher-dimensional automaton* (*HDA*) $A = (\Sigma, X, \bot)$ consists of a finite alphabet Σ, a precubical set X on Σ, and a subset $\bot \subseteq X$ of initial cells. (We will not need accepting cells in this work.) An HDA may be finite or infinite, or even infinite-dimensional.

Computations of HDAs are *paths*, *i.e.*, sequences

$$\alpha = (x_0, \varphi_1, x_1, \ldots, x_{n-1}, \varphi_n, x_n) \tag{2}$$

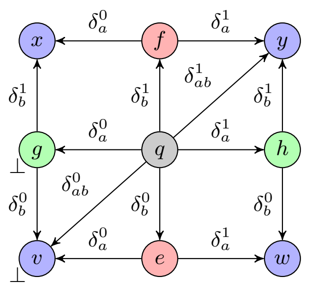
consisting of cells x_i of X and symbols φ_i which indicate which type of step is taken: for every $i \in \{1, \ldots, n\}$, $(x_{i-1}, \varphi_i, x_i)$ is either

- $(\delta^0_A(x_i), \nearrow^A, x_i)$ 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*).

Intuitively, a downstep terminates events in a cell, following an upper face map. This is why downsteps require that $A \subseteq \mathsf{ev}(x_{i-1})$, *i.e.*, events that are terminated belong to the cell. Similarly, an upstep starts events by following inverses of lower face maps. The constraints on upsteps require that $A \subseteq \mathsf{ev}(x_i)$, *i.e.*, the initiated events belong to the next cell after the step. Both types of steps may be empty.

A cell $x \in X$ is *reachable* if there exists a path α from an initial cell to x, *i.e.*, $x_0 \in \bot$ and $x_n = x$ in the notation (2) above. The *essential* part of X is the subset $\mathsf{ess}(X) \subseteq X$ containing only reachable cells. Note that, in general $\mathsf{ess}(X)$ is not necessarily an HDA, in particular when $\bot \cap X_0 = \emptyset$, in which case some 0-cells will be missing. However, this situation does not arise in our setting: we only consider HDAs X generated from Petri nets (see Section 4), whose initial cells are always of dimension 0 and for which $\mathsf{ess}(X)$ is an HDA. (This is a general principle: for any HDA X with $\bot_X \subseteq X_0$, $\mathsf{ess}(X)$ is also an HDA [11].)

Example 2.1. *Figure 4 shows a two-dimensional HDA as a combinatorial object (left) and in a geometric realisation (right). It consists of nine cells: the corner cells* $X_0 = \{x, y, v, w\}$



$X[\emptyset]=\{v,w,x,y\}$

$X[a]=\{e,f\}$

$X[b]=\{g,h\}$

$X[[\begin{smallmatrix}a\\b\end{smallmatrix}]]=\{q\}$

$\perp_X=\{v,g\}$

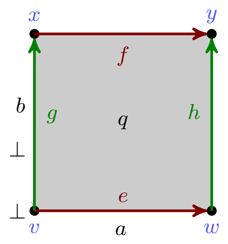


Figure 4: A two-dimensional HDA X on $\Sigma=\{a,b\}$, see Example 2.1.

(depicted in blue) in which no event is active (for all $z\in X_0$, $\mathsf{ev}(z)=\emptyset$), the transition cells $X_1=\{g,h,f,e\}$ (shown in red and green) in which precisely one event is active ($\mathsf{ev}(f)=\mathsf{ev}(e)=a$ and $\mathsf{ev}(g)=\mathsf{ev}(h)=b$), and the (gray) square cell q where $\mathsf{ev}(q)=[\begin{smallmatrix}a\\b\end{smallmatrix}]$, i.e. a and b are executed concurrently.

The arrows between the cells on the left representation correspond to the face maps connecting them. For example, the upper face map δ_{ab}^1 maps q to y because the latter is the cell in which the active events a and b of q have been terminated. On the right, face maps are used to glue cells together, so that for example $\delta_{ab}^1(q)$ is glued to the top right of q. In this and other geometric realisations, when we have two concurrent events a and b with $a \dashrightarrow b$, we will draw a horizontally and b vertically.

All the cells of the HDA X of Figure 4 are reachable from the initial cells $\{v,g\}$. For example h is reachable by $v\nearrow^{ab} q \searrow_a h$, $v\nearrow^{a} e\nearrow^{b} q\searrow_a h$ or $v\nearrow^{b} g\nearrow^{a} q\searrow_a h$. Consequently, h is also reachable from g, and the transition $h\searrow_b y$ may then be added to reach y.

3 Petri nets

A *Petri net* $N=(S,T,F)$ consists of a set of places S, a set of transitions T, with $S\cap T=\emptyset$, and a weighted flow relation $F: S\times T\cup T\times S\to\mathbb{N}$.

A *marking* of N is a function $m: S\to\mathbb{N}$, associating each place to the number of tokens present. A Petri net $N=(S,T,F)$ together with an initial marking $i: S\to\mathbb{N}$ is called a *marked Petri net*. A marked Petri net is *k-bounded* if $m(s)\le k$ for every place s. It is *bounded* if there is a value $k\in\mathbb{N}$ such that m is k-bounded.

Let E be any set. A function $f: E\to\mathbb{N}$ is a *multiset*, *i.e.*, an extension of sets allowing several instances of each element of E.

We introduce some notation for these. We write $x\in f$ if $f(x)\ge 1$. Given two multisets f_1,f_2 over E we will write $f_1\le f_2$ iff $f_1(x)\le f_2(x)$ for every element $x\in X$. If $f(x)\in\{0,1\}$ for all x, then f may be seen as a set, and the notation $x\in f$ agrees with the usual one for sets. The multisets we use will generally be finite in the sense that $\sum_{x\in E} f(x)<\infty$, and in that case we might use additive notation and write $f=\sum_{x\in E} f(x)x$. This notation easily applies to markings of Petri nets, and we will write for instance $m=2p_1+p_4$ for a marking such that $m(p_1)=2, m(p_4)=1$, and $m(p_i)=0$ for any other place $p_i\in S\setminus\{p_1,p_4\}$.

For a transition $t\in T$, the *preset* of t is the multiset ${}^\bullet t: S\to\mathbb{N}$ given by ${}^\bullet t(s)=F(s,t)$. This preset describes how many tokens are consumed in each place when t fires. The *postset* of t is the multiset $t^\bullet: S\to\mathbb{N}$ such that $t^\bullet(s)=F(t,s)$. It describes how many tokens are produced in each place of the net when firing t. For the sake of simplicity, all examples in this paper deal with Petri nets where $F\subseteq(S\times T)\cup(T\times S)$, that is, for every place s and transition t, we

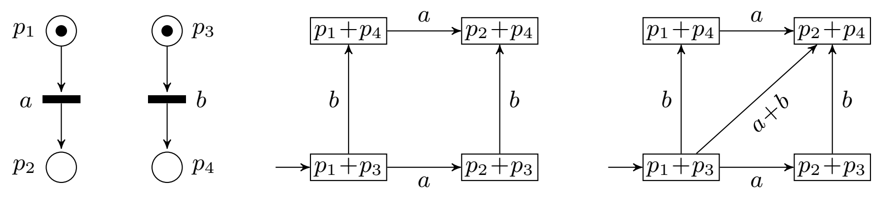


Figure 5: A Petri net N (left); the reachability graph $[\![N]\!]_1$ (middle); and its concurrent step reachability graph $[\![N]\!]_{\mathrm{CS}}$(right).

have ${}^\bullet t(s), t^\bullet(s) \in \{0,1\}$. However, the algorithms described here and their implementation are designed to handle the general case.

Petri nets compute by transforming markings. Their standard semantics is an interleaved semantics, where states are markings and a single transition can fire at each step. Let $m : S \to \mathbb{N}$ be a marking and $t \in T$, then t can *fire* in m if ${}^\bullet t \leq m$. Firing t produces a new marking $m' = m - {}^\bullet t + t^\bullet$.

The *reachability graph* (see for example [8]) of Petri net $N = (S,T,F)$ is the labeled graph $[\![N]\!]_1 = (V,E)$ given by $V = \mathbb{N}^S$ and

$$E = \{(m,t,m') \in V \times T \times V \mid {}^\bullet t \leq m, m' = m - {}^\bullet t + t^\bullet\}.$$

(The reason for the subscript 1 in $[\![N]\!]_1$ will become clear later.)

In a reachability graph vertices are markings and edges are labeled by the transition which fires. A computation of a Petri net is a path in its reachability graph. Note that we use *collective token semantics*, *i.e.*, tokens in ${}^\bullet t$ that are consumed by firing t are considered as blind resources. Petri nets also have an *individual token* semantics [13] where transitions distinguish tokens individually by considering their origin. This may be used to model realisation of independent processes; but we will not consider it here.

Let N_1, N_2 be two Petri nets. The reachability graphs $[\![N_1]\!]_1 = (V_1, E_1)$ and $[\![N_2]\!]_1 = (V_2, E_2)$ are *isomorphic*, denoted $[\![N_1]\!]_1 \cong [\![N_2]\!]_1$, if there exist bijections $f : V_1 \to V_2$ and $g : E_1 \to E_2$ such that for all $e_1 = (m_1, t_1, m_1') \in E_1$, $g(e_1) = (m_2, t_2, m_2')$ iff $f(m_1) = m_2$ and $f(m_1') = m_2'$.

Considering Petri nets via their interleaved semantics misses an important point of the model, namely concurrency. Indeed, it does not allow to distinguish between behaviours where a pair of transitions fire in sequence from behaviours where these transitions are independent and can fire concurrently. One way to cope with this issue is to consider executions of Petri nets as *processes* [13], that is, partial orders representing causal dependencies among transitions occurrences. Another possibility is the use of a *concurrent step semantics* [12], where several transitions are allowed to fire concurrently. The concurrent step semantics mimics that of the interleaved semantics, but fires multisets of transitions.

For a multiset $U : T \to \mathbb{N}$ of transitions we write ${}^\bullet U = \sum_{t \in T} {}^\bullet t U(t)$ and $U^\bullet = \sum_{t \in T} t^\bullet U(t)$. U is *firable* in marking m if ${}^\bullet U \leq m$. The *concurrent step reachability graph* [14] of Petri net $N = (S,T,F)$ is the labeled graph $[\![N]\!]_{\mathrm{CS}} = (V,E)$ given by $V = \mathbb{N}^S$ and

$$E = \{(m,U,m') \in V \times \mathbb{N}^T \times V \mid U \neq \emptyset, {}^\bullet U \leq m, m' = m - {}^\bullet U + U^\bullet\}. \tag{3}$$

Figure 5 shows a simple example of a Petri net and its two types of reachability graph. Note that transitions in $[\![N]\!]_{\mathrm{CS}}$ allow multisets of transition rather than only sets, thus several occurrences of a transition may fire in a concurrent step. This feature is called autoconcurrency, and it is well known that allowing autoconcurrency increases the expressive power of Petri nets [15]. Further, $[\![N]\!]_{\mathrm{CS}}$ is *closed under substeps* in the sense that for all multisets $V \leq U$, if $(m,U,m'') \in E$, then we also have $(m,V,m') \in E$ and $(m', U \setminus V, m'') \in E$ for some marking m'.

Notice that our definition of Petri nets allows preset-free transitions t with ${}^\bullet t = \emptyset$. When a

$$
\begin{aligned}
&X[\emptyset]=\{p_1+p_3,p_2+p_3,p_1+p_4,p_2+p_4\}\\
&X[a]=\{(p_3,a),(p_4,a)\}\\
&X[b]=\{(p_1,b),(p_2,b)\}\\
&X[\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]]=\{(0,\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right])\}\\
&X[\left[\begin{smallmatrix}b\\a\end{smallmatrix}\right]]=\{(0,\left[\begin{smallmatrix}b\\a\end{smallmatrix}\right])\}
\end{aligned}
$$

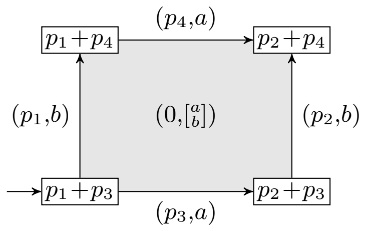


Figure 6: Higher-dimensional automaton (reachable part only) for the Petri net of Figure 5. Left: sets of cells; right: geometric realisation (not showing $X[\left[\begin{smallmatrix}b\\a\end{smallmatrix}\right]]$).

transition t has an empty preset, then t is firable from any marking. In an interleaved semantics, allowing preset-free transitions does not change expressive power, so one frequently assumes that ${}^\bullet t \neq \emptyset$ for every $t \in T$. In the setting of a concurrent semantics with autoconcurrency, an arbitrary number of occurrences of each preset-free transitions may fire from any marking. Because of this, every vertex in $[\![N]\!]_{\mathrm{CS}}$ would be of infinite degree. We will generally allow preset-free transitions in what follows, however as the resulting reachable markings is infinite, the algorithms we describe will not terminate. Notice that postset-free transitions (*i.e.*, transitions t with $t^\bullet = \emptyset$) are not problematic. They do not cause the reachability graph to become infinite.

4 From Petri nets to HDAs, revisited

A construction from a Petri net N to higher-dimensional automaton $[\![N]\!]$ was first explored in [16, Definition 9]. This construction was adapted in [5] to the event-based setting of HDAs introduced in [9] where the labels of the events in $[\![N]\!]$ are the transitions of N.

Let $N=(S,T,F)$ be a Petri net. Let $\square = \square(T)$ and define $X=\mathbb{N}^S \times \square$ and $\mathsf{ev}: X \to \square$ by $\mathsf{ev}(m,\tau)=\tau$. For $x=(m,\tau)\in X[\tau]$ with $\tau=(t_1,\dots,t_n)$ non-empty and $i\in\{1,\dots,n\}$, define

$$
\begin{aligned}
\delta^0_{t_i}(x)&=(m+{}^\bullet t_i,(t_1,\dots,t_{i-1},t_{i+1},\dots,t_n)),\\
\delta^1_{t_i}(x)&=(m+t_i^\bullet,(t_1,\dots,t_{i-1},t_{i+1},\dots,t_n)).
\end{aligned}
$$

Eq. (1) could be used to generate the face maps terminating or unstarting multisets of events, in order to define a precubical set $[\![N]\!]=X$. From a practical point of view this is however not necessary. In [5] and this work we are concerned with state-space exploration and reachability properties. In this context, there is no difference between using a face map with a multiset of events and its decomposition into a series of "elementary" face maps using a single event. Hence, a path in $[\![N]\!]$ is a computation in N in which steps start or terminate single events and concurrency of events, i.e. transition that fire concurrently in the net, can be retrieved by looking at the conclist of the cell in which the computation currently finds itself.

Note that the 0-cells of $[\![N]\!]$ correspond to the markings of N and in an n-cell of $[\![N]\!]$, n transitions of N are running concurrently. It was shown in [5] that $[\![N]\!]$ is closely related to both the reachability graph and the concurrent step reachability graph of N. Specifically, the graph whose vertices are the 0-cells of $[\![N]\!]$ and whose edges are triples of the form $(\delta^0_t(x),x=(m,t),\delta^1_t(x))$ is isomorphic to the reachability graph of N (see [5, Lemma 4]). Furthermore, consider the graph with the same vertex set (the 0-cells of $[\![N]\!]$), and edge set

$$E=\{(x,U,z)\mid \exists y\in X:\ \delta^0_{\mathsf{ev}(y)}(y)=x,\ \delta^1_{\mathsf{ev}(y)}(y)=z,\ \mathrm{pi}(\mathsf{ev}(y))=U\},$$

where, for a sequence $a=(a_1,\dots,a_n)\in\square(\Sigma)$ over some alphabet Σ, the *Parikh image* $\mathrm{pi}(a):\Sigma\to\mathbb{N}$ is defined by counting symbol occurrences $\mathrm{pi}(a)(x)=|\{i\mid a_i=x\}|$. This graph is isomorphic to the concurrent step reachability graph of N (see [5, Lemma 5]).

Note that for the initial marking i of the marked net, we have $i \in [\![N]\!][\emptyset]$, the construction above induces an HDA $[\![N]\!] = (T,X,\bot)$ with its initial cell set to $\bot = \{i\}$. In addition, as firable transitions only depend on the current marking, and as the effect of a firing is deterministic, when a marked net is bounded, the reachable part of $[\![N]\!]_1$ is finite. However, due to autoconcurrency, this property does not hold for the full $[\![N]\!]$, as shown in [5, Example 6]. Nevertheless, when marked Petri net N is bounded and has no preset-free transitions, then $[\![N]\!]$ is finite [5, Proposition 7].

The above definition of the HDA $[\![N]\!]$ is highly symmetric: for a given cell defined by its marking m and its conclist τ, $c = (m,\tau)$, there exists another cell $c = (m,\tau')$ with τ' being a permutation of τ. However, *in fine* we are only interested in the *multiset* of concurrently active transitions. In order to avoid the factorial blow-up in the number of cells, we fix an arbitrary (non-strict) total order $\preccurlyeq$ on the transitions in T and then instead of $\square(T) \cong T^*$ consider the set $T^*_{\preccurlyeq} = \{(t_1,\dots,t_n) \mid \forall i = 1,\dots,n-1 : t_i \preccurlyeq t_{i+1}\}$

The definition of the face maps of this reduced $X = [\![N]\!]$ stays the same, and X is now a (non-symmetric) precubical set with one cell for every marking m and every *multiset* of transitions τ.

Figure 6 shows the HDA $[\![N]\!]$ for the Petri net N of Figure 5 with initial marking $i = p_1 + p_3$.

While this transformation is effective, it is not efficient. By keeping in memory all cells, we waste computational resources and do not take advantage of the HDA's structure. One can notice that by knowing a cell, one can deduce its subcells. As 0-cells are uniquely defined by the corresponding marking in the Petri net, we can reconstruct the HDA knowing only the cells of highest dimensions and how to reach their shared faces. This is formalized in the following.

Let us first define for an HDA X and two cells $x,y \in X$ the set of faces that x and y share

$$\mathsf{sh}_X(x,y) = \{z \in X \mid \exists A_x, B_x, A_y, B_y : z = \delta_{A_x,B_x,\mathsf{ev}(x)}(x) = \delta_{A_y,B_y,\mathsf{ev}(y)}(y)\}$$

Definition 1 (Max-Cell HDA representation (MHDA))**.** *The max-cell HDA representation of an HDA A defined by its set of cells X and its face maps δ^0 and δ^1 is the pair $A_{\max} = (X_{\max}, \delta_{\max})$ where:*

- $X_{\max} = \{x \in X \mid \nexists y \in X, a \in \Sigma : x = \delta^0_a(y)$ *or* $x = \delta^1_a(y)\}$*: the cells of X that are neither lower nor upper face maps of any cell in X*
- *For each $x,y \in X_{\max}$, if*
  - *there exists $z \in \mathsf{sh}_X(x,y)$ such that $z = \delta_{A_x,B_x,\mathsf{ev}(x)}(x) = \delta_{A_y,B_y,\mathsf{ev}(y)}(y)$*
  - *and there is no $w \in \mathsf{sh}_X(x,y)$, $a \in \Sigma$ such that $z = \delta^0_a(w)$ or $z = \delta^1_a(w)$*

  then $((A_x,B_x),(A_y,B_y),(\mathsf{ev}(x),\mathsf{ev}(y))) \in \delta_{\max}$, for some $A_x,A_y,B_x,B_y \in \square$. This is the set of multisets one must unstart and terminate in respectively $\mathsf{ev}(x)$ and $\mathsf{ev}(y)$ to reach from both $x,y \in X_{\max}$ a shared face which is maximal in $\mathsf{sh}_X(x,y)$. We write $\delta_{(A_x,B_x),(A_y,B_y),(\mathsf{ev}(x),\mathsf{ev}(y))}$ when $(A_x,B_x),(A_y,B_y),(\mathsf{ev}(x),\mathsf{ev}(y)) \in \delta_{\max}$.

This reduced representation retains all the necessary information. Indeed, each maximal cell $x \in X_{\max}$ of dimension n determines an n-dimensional HDA: it contains a single n-cell, namely x, and all lower-dimensional faces can be uniquely reconstructed from it using the precubical structure. The relation $\delta_{\max}$ then serves to identify the shared cells between these HDAs: knowing how to reach a maximal shared face from two maximal cells x and y is sufficient to deduce the other shared faces and the ways to reach them from x and y (see Example 5.1). In the next section, we present an algorithm that constructs an MHDA from a Petri net, such that the corresponding HDA is the same as the one obtained via the translation from [5].

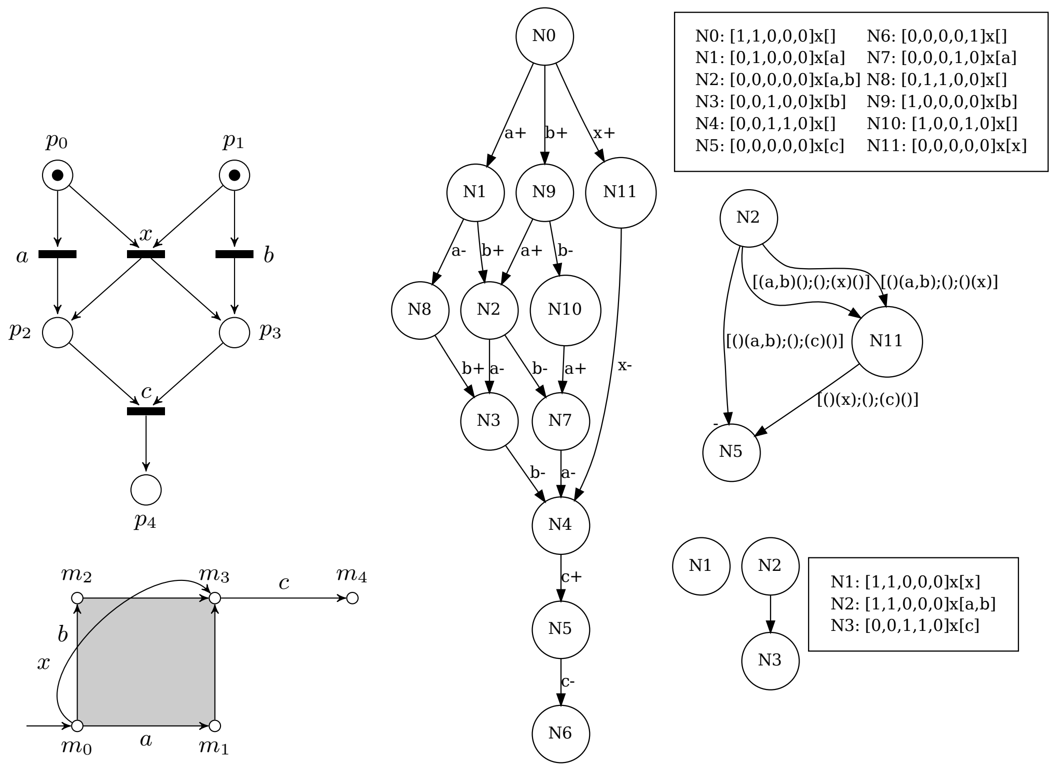


Figure 7: A Petri net, its corresponding HDA (geometrical on the bottom left and state view on the middle), with its MHDA. Top obtained from reducing an HDA, bottom from direct translation. Note that here both, $N1$ and $N2$ are intial cells.

5 Algorithm

5.1 Reduction from HDAs to MHDAs

As a first step, we implemented a translation from HDAs to MHDAs, and therefore, by using the translation from Petri nets to HDAs introduced in [5] also from Petri nets to MHDAs. The most expensive part of this translation is finding the maximal shared faces between max-cells, as this necessitates checking for each pair of max-cells if they have a common face. Once such a face is found we also need to determine its maximality, involving the inclusion check of conclists of the face itself, but also of the multisets allowing to attain it. That is a face map entry $\delta_{(A'_x,B'_x),(A'_y,B'_y),(\mathsf{ev}(x),\mathsf{ev}(y))}$ is considered larger than a face map entry $\delta_{(A_x,B_x),(A_y,B_y),(\mathsf{ev}(x),\mathsf{ev}(y))}$ if and only if the following conditions hold:

$$\text{(I) } \mathsf{ev}(x)\setminus(A_x\cup B_x)\subseteq\mathsf{ev}(x)\setminus(A'_x\cup B'_x) \quad \text{(II) } A'_x\subseteq A_x\wedge B'_x\subseteq B_x \quad \text{(III) } A'_y\subseteq A_y\wedge B'_y\subseteq B_y$$

The first condition ensures the maximality of the shared face itself, the second and third conditions ensure that both face maps use "comparable" events. This is particularly important when *parallel* transitions are present: set of transitions sharing the same pre- and postset.

Example 5.1 (Comparison of the standard and max-cell representation of an HDA)**.** *In Figure 7, a Petri net is depicted on the upper left alongside its resulting HDA, with the geometrical view bottom left and its representation as starter-terminator automaton in the middle: each transition* *corresponds to the start (such as a+) or the end (such as b−) of an event. The MHDA is represented on the bottom right. Note that the transition x is parallel to $\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]$. Therefore, firing x or the set $\{a,b\}$ in a concurrent step has the same effect on the marking.*

The fact that $\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]$ is parallel to x, i.e., causes the same marking change, becomes clear when looking at the paths from $N0$ to $N4$. On the MHDA, there are only three max-cells, with the conclists $\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]$ ($N2$), x ($N11$) and c ($N5$). Here transition need to be read as $(A_x,B_x);(\mathsf{ev}(x)\setminus A_x\cup B_x);(A_x,B_x)$, that is we list the events that need to unstarted / terminated from the source and target cell to get to the shared face with the given conclist. It can be clearly seen that the 2-cell $N2$ shares the all "all events unstarted" and "all events terminated" 0-cells with $N11$ corresponding to them being parallel. The first transitions unstarts all events, the second transitions terminates them. Note that in this example, we reduced the number of cells from 12 to 3, but even more importantly, the number of distinct markings (which has a significant impact on memory consumption) drops from 10 to 1.

All of this causes the reduction algorithm to be fairly expensive exemplified in Table 1. At the same time, the table also highlights the possible gains using the MHDA representation: The number of cells, face map entries and even more so the number of distinct markings and conclists is significantly reduced. Table 1 illustrates the spatial gain of the new MHDA representation compared to the previous naive HDA representation. For both, we reported the number of cells, conclists, and markings saved in the final data structure on multiple examples. In the table, we also highlight the time cost of the new conversion algorithm compared to the previous one.

	HDA			MHDA			time (ms)	
Name	cells	conclists	markings	cells	conclists	markings	PN→HDA	HDA→MHDA
abx_1	12	6	10	3	4	1	0.04	0.1
abx_2	69	20	46	6	18	1	0.2	7
abx_3	272	50	146	10	48	1	0.6	193
abx_4	846	105	371	15	103	1	2	4450
abx_5	2232	196	812	21	194	1	9	74.10^3
Sudoku-PT-A-N01	3	2	3	1	1	1	0.008	0.004
Sudoku-PT-A-N02	177	35	176	6	23	5	0.4	19
afcs_01_a	4727	941	1076	417	941	12	21	865.10^3

Table 1: Comparison of HDA and MHDA space complexity. The Sudoku and afcs instances are taken from the mcc, abx_n corresponds to our running example with n tokens in p_0 and p_1.

5.2 A first direct translation from Petri nets to MHDAs

As shown above, the reduction from HDAs to MHDAs is expensive, in particular due to the large number of inclusion checks between conclists. To remedy this issue we present a direct translation from Petri Nets to MHDAs in this section.

The basic idea is to iteratively construct the MHDA by exploring the reachable max-cells. We strive for a minimal memory footprint at the expense of runtime by never storing the reachable markings (corresponding to 0-cells) but only generating them on the fly when necessary.

In Algorithm 1 the definition of a max-cell is slightly different from the one used before. Instead of the marking of the cell itself, we add the preset of all currently executed transitions, as this allows to express certain operations more naturally and efficiently.

We write $(m,c)\sqsubseteq(m',c')$ to denote that a cell $lc=(m,c)$ defined by its initial marking m and conclist c is a subcell of $lc'=(m',c')$. This is expressed by splitting the conclist c' into two parts u and v, such that

Algorithm 1 Petri net to MHDA

```
Require: Input: Petri net $pn$
Require: Output: Corresponding max HDA
  $mhda \leftarrow empty\_mhda()$
  \* Stack of markings to visit
  $stack \leftarrow [pn.init\_mark]$
  while $stack$ do
      $nm \leftarrow stack.pop()$
      \* Generate all maximal concurrent steps from the current marking
      $all\_MCS = gen\_max\_conc\_step(pn,nm)$
      for $aMCS \in all\_MCS$ do \* Exploring them
          \* Check if it is a subcell of or equal to a currently existing max cell
          if $\exists lc \in mhda.max\_cells(): (nm,aMCS) \sqsubseteq (lc.m,lc.c)$ then
              $continue$
          \* We have found a new max cell
          $nc = mhda.add\_cell(nm,aMCS)$
          \* Check if the new cell subsumes existing ones
          for $sc \in mhda.max\_cells(): (sc.m,sc.c) \sqsubseteq (nc.m,nc.c)$ do
              $mhda.delete\_cells(sc)$
          \* Iterate over all possible concurrent steps
          for $cs \in 2^{aMCS}$ do
              \* Compute marking of the corner
              $co = nc.m + \texttt{fire}(cs)$
              stack.push(co)
```

- $c' = u \cup v$ with the conclists seen as multisets
- $m = m' - {}^\bullet u + u^\bullet$ reach the corner of lc from the corner of lc'
- $c \subseteq v$ the conclist of lc is subset by whatever is left in c' after firing the events in u.

These constraints are necessary and sufficient to ensure that the cell mc is a subcell of mc'. In order to avoid the explicit generation of subcells during inclusion checks, we reformulate the problem as an integer program.

$gen\_max\_conc\_step(pn,m)$ generates the list of all maximal concurrent steps that can be fired from the marking m. That is for any cs in the resulting list $gen\_max\_conc\_step(pn,m)$ (again, each concurrent step is represented by a multiset) we have ${}^\bullet cs \leq m$. It being maximal means that we can not add any transition to the multiset cs without violating the fireability constraint.

5.3 An example of the conversion

We illustrate Algorithm 1 with an example continuing Example 5.1 and represented in Figure 7. Intuitively, in this Petri net, there are mutually exclusive ways to enable transition c, either by firing (concurrently or sequentially) a and b, or by firing x. In the HDA, this is represented by the $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ square and x 1-cell. Notice that these two only share the lower and upper faces N_0 and N_4.

As the cells of a MHDA are represented by a couple (m,c), with m the marking of the 0-cell where all events are unstarted, and c its max-conclist, we represent the conclist labels on the 1-cells and the markings are named (from m_0 to m_4 included) on the 0-cells of the HDA figure. By doing so, we can iterate over the conversion process with the same names. The stack is represented as a bracketed list, with its first element being the latest inserted and first to be processed (even if the order does not matter to explore the whole structure).

First, we start with an empty HDA, an empty stack and the initial marking $m_0 = p_0 + p_1$ as the current marking. In this case, we will create all max-conclists. This results in the conclists set $\{[x], [\begin{smallmatrix}a\\b\end{smallmatrix}]\}$. By iterating over this set, we will first add the max 1-cell $(m_0, [x])$, and the marking $m_3 = p_2 + p_3$ is added to the stack. Then, the max 2-cell $(m_0, [\begin{smallmatrix}a\\b\end{smallmatrix}])$ is added to the HDA. Note that, depending on how the set is represented, it could have been explored first, but this does not change the result. With this new cell, the markings $m_1 = p_1 + p_2$, $m_2 = p_0 + p_3$, and $m_3 = p_2 + p_3$ of the 3 remaining corners of the 2-cell have to be added to the stack as well. Then, the marking m_3 is processed. This marking was generated by the two max-cells already created. However, after processing its max conclist (which is unique in this case), we obtain the conclist $[c]$, which is not a sub-conclist of any of the max-cells already generated. Therefore, the cell $(m_3, [c])$ is a new cell and is added to the HDA. The marking $m_4 = p_4$ is also added to the stack. While all maximum cells have been created, the algorithm still has to empty the stack to ensure that the exploration is complete. By processing m_4, we notice the max-conclist set is empty: there is no transition we can fire here. The marking is thus ignored. Then, the markings m_2 and m_1 have 1 max-conclist each ($[a]$ and $[b]$ respectively). However, for both cells $c \in \{(m_1, [b]), (m_2, [a])\}$, we have $c \sqsubseteq (m_0, [\begin{smallmatrix}a\\b\end{smallmatrix}])$. So, both cells found in markings m_1 and m_2 are in reality sub-cells of an already existing max 2-cell, and they are not added to the MHDA. Finally, the marking m_3 is the starting point of the cell $(m_3, [c])$ explored previously. Therefore, this marking was already explored and there is no interest in searching for its max-conclists. The stack is now empty and the conversion is successfully done. Note that currently, for direct MHDA construction, we only store a simplified version of the face maps, containing only the information about the source and destination cell. These transition form a spanning tree of the complete MHDA and this sufficient for reachability questions. If needed, the more detailed information about shared faces can be recovered.

Computing State Space properties from MHDA Using our reduced representation for reachability problems is fairly straight forward, however computing properties of the state space, *i.e.*, properties of the reachability graph of the Petri Net which is equivalent to the 1-truncation of the HDA, turns out to be fairly involved and we do not have a closed form solution yet.

Some of the arising problems are illustrated in Figure 8. This HDA has a single 3-cell with conclist $\begin{bmatrix}a\\a\\a\end{bmatrix}$. The standard calculation tells us that there are $3^3 = 27$ cells in the HDA. However this only holds when there is no autoconcurrency. Due to autoconcurrency, there are several cells that are identified with one another. That is they share the same marking and conclist, in the figure all identified cells share the same shape or line style. The actually geometric interpretation is therefore a cube "folded onto itself" on certain faces, edges and corners. In fact there is one 3-cell, 2 2-cells (the faces neighbouring the initial corner, and those neighbouring the diamond shaped top right corner), 3 1-cells identified by the line styles, and 3 0-cells identified via shape, for a total of 9 cells.

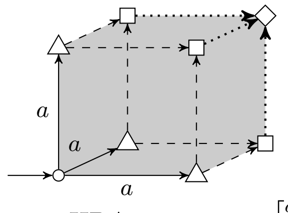


Figure 8: HDA recognizing $\begin{bmatrix}a\\a\\a\end{bmatrix}$

This phenomena does not only occur when there is autoconcurrency, but also when the conclist contains sets of parallel transitions. Additionally, to correctly compute state space properties, we would have to take into account the face maps and their shared faces, making the problem rather complex.

	HDA			MHDA			time (s)	
Name	cells	conclists	markings	cells	conclists	markings	PN→HDA	PN→MHDA
abx_3	272	50	146	10	10	4	0.003	0.7
abx_4	846	105	371	15	15	5	0.005	4
abx_5	2232	196	812	21	21	6	0.008	13
abx_6	5214	336	1596	28	28	7	0.01	44
Sudoku-PT-A-N02	177	35	176	6	23	5	0.4	19

Table 2: Comparison for direct MHDA construction. Note that the number of conclists necessary may be different here from the one reported in Table 1 as the face maps do not store detailed information. abx_n refers to our running example with n tokens in p_0 and p_1.

6 Evaluation

Here we present the results of concerning the number cells, conclists and markings needed to represent the MHDA, as well as the execution time for a selected set of benchmarks. We give the execution time for completeness, but it needs to be handled with care: currently the overall runtime is dominated by the cost of inclusion checks. These are very solver and formulation dependant and can in our opinion be greatly improved with further development. To reproduce these results and for further insights please see https://gitlabev.imtbs-tsp.eu/philipp.schlehuber-caissier/pn2hda/-/tree/sub/msr25.

7 Conclusion

To address the combinatorial explosion of the translation from Petri nets to HDAs, we proposed a reduced representation and a refined algorithm that, given a Petri net, does not construct the full HDA explicitly. Instead, it computes only an MHDA representation: a set of so-called maximal cells along with relations between them. The resulting compact representation drastically reduces the number of cells, conclists and markings that need to be stored, and the full HDA can be reconstructed when needed.

This work represents a first, yet promising, attempt at efficiently representing HDAs generated from Petri nets, while avoiding their full explicit construction. In practice, our construction tends to be slower than building the full HDA, but the memory savings make the approach beneficial overall. That said, both the algorithm and its implementation could be improved, especially to reduce runtime. This can be done by improving the exploration and computing the maximal concurrent steps from the faces of cells and not corners. This intertwines the generation of the maximal concurrent steps and their exploration reducing overall complexity.

Finally, our reduced representation of HDAs could also be used in logic-to-HDA translations for model checking purposes. The appeal of HDAs lies in their ability to accept all possible interleavings of a given execution. This property naturally supports partial-order reduction techniques and can significantly enhance state-space exploration in system modeling. Several recent works [2, 4, 7, 18] have explored connections between modal, first-order, and second-order logics and HDAs and we seek to adapt them to work directly on MHDAs.

References

[1] Amazigh Amrane, Hugo Bazille, Emily Clement, and Uli Fahrenberg. Languages of higher-dimensional timed automata. In Lars Michael Kristensen and Jan Martijn van der Werf, editors, *PETRI NETS*, volume 14628 of *Lecture Notes in Computer Science*, pages 197–219. Springer, 2024.

[2] Amazigh Amrane, Hugo Bazille, Emily Clement, Uli Fahrenberg, Marie Fortin, and Krzysztof Ziemianski. Büchi-elgot-trakhtenbrot theorem for higher-dimensional automata. *CoRR*, abs/2505.10461, 2025.

[3] Amazigh Amrane, Hugo Bazille, Emily Clement, Uli Fahrenberg, and Krzysztof Ziemiański. Presenting interval pomsets with interfaces. In Uli Fahrenberg, Wesley Fussner, and Roland Glück, editors, *RAMiCS*, volume 14787 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 2024.

[4] Amazigh Amrane, Hugo Bazille, Uli Fahrenberg, and Marie Fortin. Logic and languages of higher-dimensional automata. In Joel D. Day and Florin Manea, editors, *DLT*, volume 14791 of *Lecture Notes in Computer Science*, pages 51–67. Springer, 2024.

[5] Amazigh Amrane, Hugo Bazille, Uli Fahrenberg, Loïc Hélouët, and Philipp Schlehuber-Caissier. Petri nets and higher-dimensional automata. In Elvio Gilberto Amparore and Lukasz Mikulski, editors, *Application and Theory of Petri Nets and Concurrency - 46th International Conference, PETRI NETS 2025, Paris, France, June 22-27, 2025, Proceedings*, volume 15714 of *Lecture Notes in Computer Science*, pages 18–40. Springer, 2025.

[6] 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.

[7] Emily Clement, Enzo Erlich, and Jérémy Ledent. Expressivity of linear temporal logic for pomset languages of higher dimensional automata. *CoRR*, abs/2410.12493, 2024.

[8] Jörg Desel and Wolfgang Reisig. Place/transition Petri nets. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*, pages 122–173. Springer, 1998.

[9] Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Languages of higher-dimensional automata. *Mathematical Structures in Computer Science*, 31(5):575–613, 2021. `https://arxiv.org/abs/2103.07557`.

[10] Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Kleene theorem for higher-dimensional automata. *Logical Methods in Computer Science*, 20(4), 2024.

[11] Uli Fahrenberg and Krzysztof Ziemiański. Myhill-Nerode theorem for higher-dimensional automata. *Fundamentae Informatica*, 192(3-4):219–259, 2024.

[12] Hartmann J. Genrich, Kurt Lautenbach, and Pazhamaneri S. Thiagarajan. Elements of general net theory. In Wilfried Brauer, editor, *Net Theory and Applications*, pages 21–163. Springer, 1980.

[13] Ursula Goltz and Wolfgang Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57(2):125–147, 1983.

[14] Madhavan Mukund. Petri nets and step transition systems. *International Journal of Foundations of Computer Science*, 3(4):443–478, 1992.

[15] Rob J. van Glabbeek. The individual and collective token interpretations of Petri nets. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2005.

[16] Rob J. van Glabbeek. On the expressiveness of higher dimensional automata. *Theoretical Computer Science*, 356(3):265–290, 2006. See also [17].

[17] Rob J. van Glabbeek. Erratum to "On the expressiveness of higher dimensional automata". *Theoretical Computer Science*, 368(1-2):168–194, 2006.

[18] Safa Zouari, Krzysztof Ziemianski, and Uli Fahrenberg. Bisimulations and logics for higher-dimensional automata. In Chutiporn Anutariya and Marcello M. Bonsangue, editors, *Theoretical Aspects of Computing - ICTAC 2024 - 21st International Colloquium, Bangkok, Thailand, November 25-29, 2024, Proceedings*, volume 15373 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2024.

A Improvements to the algorithm

As reported in Table 2, the runtime of the new algorithm is significantly worse than the one of the original algorithm. This is partly due to the costly inclusion check, currently performed via a reduction to an integer program, which is then fed to z3 via its C++ API. However, z3 is capable of solving SMT problems, and using a more dedicated solver might improve performance. On the other hand, many optimizations concerning the exploration are possible. We present a first step in this direction in Algorithm 2. The idea is to check for existence of the marking before pushing it onto stack. That is, if there already exists a max-cell capable of producing this marking, it is either already on the stack or it has already been explored. This avoids the creation and inclusion checking of the, possibly many, maximal conclists that are fireable from the marking.

Algorithm 2 Improved Petri Net to MHDA

```
Require: Input: Petri net $pn$
Require: Output: Corresponding max HDA
  $mhda \leftarrow empty\_mhda()$
  \* Stack of markings to visit
  $stack \leftarrow [pn.init\_mark]$
  while $stack$ do
      $nm \leftarrow stack.pop()$
      \* Generate all maximal concurrent steps from the current marking
      $all\_MCS = gen\_max\_conc\_step(pn,nm)$
      for $aMCS \in all\_MCS$ do \* Exploring them
          \* Check if it is a subcells of or equal to a currently existing max cell
          if $\exists mc \in mhda.max\_cells() : (nm,aMCS) \sqsubseteq (mc.m,mc.c)$ then
              $continue$
          \* We have found a new max cell
          $nc = mhda.add\_cell(nm,aMCS)$
          \* Check if the new cell subsumes existing ones
          for $sc \in mhda.max\_cells() : (sc.m,sc.c) \sqsubseteq (nc.m,nc.c)$ do
              $mhda.delete\_cells(sc)$
          \* Iterate over all possible concurrent steps
          for $cs \in 2^{aMCS}$ do
              \* Compute marking of the corner
              $mo = nc.mi + \texttt{fire}(cs)$
              \* Verify that it is not already pushed or explored
              if $\exists s \notin stack : co = s \wedge \nexists m \in mhda.max\_cells() :$
                  $m \neq nc \wedge (mc,\{\}) \sqsubseteq (m.m,m.c)$ then
                  stack.push(co)
```

Note that, despite looking like a special case, $(m,\{\}) \sqsubseteq (m',c')$ and $(m,c) \sqsubseteq (m',c')$ are the same. For computation, the later is reduced to $(m,\{\}) \sqsubseteq (m',c''=c' \setminus c)$.

Due to timing constraints this version of the algorithm could not be tested properly for this work, but it will be available soon at https://gitlabev.imtbs-tsp.eu/philipp.schlehuber-caissier/pn2hda/-/tree/sub/msr25.