Backwards-Reachability for Cooperating Multi-Pushdown Systems

Chris Köcher[a,1], Dietrich Kuske[b]

[a] *Max Planck Institute for Software Systems, Paul-Ehrlich-Straße 26, 67663, Kaiserslautern, Germany*
[b] *Technische Universität Ilmenau, Ehrenbergstraße 29, 98693, Ilmenau, Germany*

Abstract

A cooperating multi-pushdown system consists of a tuple of pushdown systems that can delegate the execution of recursive procedures to sub-tuples; control returns to the calling tuple once all sub-tuples finished their task. This allows the concurrent execution since disjoint sub-tuples can perform their task independently. Because of the concrete form of recursive descent into sub-tuples, the content of the multi-pushdown does not form an arbitrary tuple of words, but can be understood as a Mazurkiewicz trace.

For such systems, we prove that the backwards reachability relation efficiently preserves recognizability, generalizing a result and proof technique by Bouajjani et al. for single-pushdown systems. It follows that the reachability relation is decidable for cooperating multi-pushdown systems in polynomial time and the same holds, e.g., for safety and liveness properties given by recognizable sets of configurations.

Keywords: Reachability, Formal Verification, Pushdown Automaton, Distributed System

---

Email addresses: ckoecher@mpi-sws.org (Chris Köcher), dietrich.kuske@tu-ilmenau.de (Dietrich Kuske)

[1] This work was done while Chris Köcher was affiliated with the Technische Universität Ilmenau.

1. Introduction

In this paper, we introduce the model of cooperating multi-pushdown systems[2] and study the reachability relation for such systems. To explain the idea of a cooperating multi-pushdown system, we first look at well-studied pushdown systems. They model the behavior of a sequential recursive program and possess a control state as well as a pushdown. The top symbol of the pushdown stores the execution context, e.g., parameters and local variables, the state can be used to return values from a subroutine to the calling routine. Such a system can, depending on the state and the top symbol, do three types of moves: it can call a subroutine (i.e., change state and top symbol and add a new symbol on top of the pushdown), it can do an internal action (i.e., change state and top symbol), and it can return from a subroutine (i.e., delete the top symbol and store the necessary information into the state). This leads to the unifying definition of a transition that, depending on state and top symbol, changes state and replaces the top symbol by a (possibly empty) word.

A cooperating multi-pushdown system consists of a finite family of pushdown systems (indexed by a set P). Cooperation is realized by the formation of temporary coalitions that perform a possibly recursive subroutine in a joint manner. Suppose the system is in a configuration where $C \subseteq P$ forms one of the coalitions. The execution context of the joint task is distributed between the top symbols of the pushdowns from the coalition and can only be changed in all these components at once. As above, there are three types of moves depending on the top symbols and the states of the systems from the coalition. First, a (further) subroutine can be called on a sub-coalition $C_0 \subseteq C$. Even more, several subroutines can be called in parallel on disjoint sub-coalitions of C. This is modeled as a change of states and top symbols of C and addition of some further symbols on the pushdowns from subsets of C. Internal actions of the coalition C can change the (common) top symbol as well as the states of the systems that form the coalition C. Similarly, a return move deletes the common top symbol and changes the states of the systems from C, in this moment, the coalition C is dissolved and the systems from C are free to be assigned to new coalitions and tasks by the calling routine. Since several, mutually disjoint coalitions can exist and operate at any particular moment, the cooperating multi-pushdown system is a non-sequential model. Note that the concrete coalitions of systems and the assignments to tasks are fixed by a given specification which we call distributed alphabet in this paper.

Since a cooperating multi-pushdown system consists of several pushdown systems, a configuration consists of a tuple of local states and a tuple of pushdown contents; the current division into coalitions is modeled by the top symbols of the pushdowns: any component forms a coalition with all components that have the same top symbol a on their stack. Since all these occurrences of the

[2] A more descriptive name would be "cooperating systems of pushdown systems", but we refrain from using this term.

letter a can only change at once, there is some dependency in the tuple of pushdown contents of a configuration. It turns out to be convenient and fruitful to understand such a "consistent" tuple of pushdown contents as a Mazurkiewicz trace. Since the set of all Mazurkiewicz traces forms a monoid, we can define recognizable and rational sets of traces and therefore of configurations: Both these classes of sets of traces enjoy finite representations (by asynchronous automata [1] and NFAs, resp.) that allow to decide membership, any recognizable set is rational but not *vice versa*, any singleton is both, recognizable and rational, and inclusion of a rational set (and therefore in particular of a recognizable set) in a recognizable set is efficiently decidable (but not *vice versa*).

As an example, we show that transformers, a computational model used for recent large language models (LLMs) [24], can be modeled with the help of cooperating multi-pushdown systems.

Then, as our main results, we obtain that backwards reachability (1) efficiently preserves the *recognizability* of sets of configurations while (2) it does not preserve *rationality*. We also show that asynchronous multi-pushdown systems (a slight generalization of our model) can model 2-pushdown systems and therefore have an undecidable reachability relation.

From our positive result, we infer that the reachability relation as well as certain safety and liveness properties are decidable in polynomial time. Furthermore, the first result implies that EF-model checking is decidable, although one only obtains a non-elementary complexity bound.

Related work. Multiple algorithms for computing the forwards or backwards reachable configurations in pushdown systems where rationality and recognizability coincide [2] can be found (e.g.) in [3, 4, 5, 6]. Our proof of (1) generalizes the one by Bouajjani et al.

Other forms of multi-pushdown systems have been considered by different groups of authors, e.g., [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. These alternative models may contain a central control or, similarly to our cooperating systems, local control states. The models can have a fixed number of processes and pushdowns or they are allowed to spawn or terminate other processes. Local processes can differ in their communication mechanism, e.g., by rendezvous or FIFO-channels. The decidability results concern logical formulas of some form or bounded model checking problems.

Mazurkiewicz traces as a form of storage mechanism have been considered by Hutagalung et al. in [18], where multi-buffer systems were studied.

The results of this paper were announced in the conference contribution [19].

2. Preliminaries

For a binary relation $R \subseteq S^2$ and $s, t \in S$ we define the sets $s\,R := \{t \in S \mid s\,R\,t\}$ and $R\,t := \{s \in S \mid s\,R\,t\}$.

For $n \in \mathbb{N}$, $[n] = \{1, \ldots, n\}$. Let $(S_i)_{i\in[n]}$ be a tuple of sets, $I, J \subseteq [n]$ be two disjoint sets, and $\overline{s} = (s_i)_{i\in[n]}$ and $\overline{t}$ be tuples from $\prod_{i=1}^{n} S_i$. We write

$\overline{s}\restriction_I = (s_i)_{i\in I} \in \prod_{i\in I} S_i$ for the restriction of $\overline{s}$ to the components in I and $(\overline{s}\restriction_I, \overline{t}\restriction_J)$ for the joint tuple $\overline{r} \in \prod_{i\in I\cup J} S_i$ with $\overline{r}\restriction_I = \overline{s}\restriction_I$ and $\overline{r}\restriction_J = \overline{t}\restriction_J$.

For a word $w \in A^*$, we write $\mathrm{Alph}(w)$ for the set of letters occurring in w.

A *non-deterministic finite automaton* or *NFA* is a tuple $\mathfrak{A} = (Q, A, I, \delta, F)$ where Q is a finite set of *states*, A is an alphabet, $I, F \subseteq Q$ are sets of *initial* and *accepting* states, respectively, and $\delta \subseteq Q \times A \times Q$ is a set of *transitions*; its size $\|\mathfrak{A}\|$ is $|Q| + |Q|^2 \cdot |A|$. For $Q_1, Q_2 \subseteq Q$ and $w \in A^*$, we write $Q_1 \xrightarrow{w}_{\mathfrak{A}} Q_2$ if there is a run from some state $p \in Q_1$ to some state $q \in Q_2$ labeled with w in $\mathfrak{A}$; $\{p\} \xrightarrow{w}_{\mathfrak{A}} \{q\}$ is abbreviated $p \xrightarrow{w}_{\mathfrak{A}} q$. The *language accepted by* $\mathfrak{A}$ is $L(\mathfrak{A}) := \{w \in A^* \mid I \xrightarrow{w}_{\mathfrak{A}} F\}$.

We will model the contents of our multi-pushdown systems with the help of Mazurkiewicz traces; for a comprehensive survey of this topic we refer to [20]. Traces were first studied in [21] as “heaps of pieces” and later introduced into computer science by Mazurkiewicz to model the behavior of a distributed system [22]. The fundamental idea is that any letter $a \in A$ is assigned a set of *locations* or *processes* $\mathrm{loc}(a) \subseteq P$ it operates on (where P is some set):

Definition 1. A *distributed alphabet* is a triple $\mathcal{D} = (A, P, \mathrm{loc})$ where A and P are two alphabets of *letters* and *processes*, respectively, and $\mathrm{loc}\colon A \to 2^P \setminus \{\emptyset\}$ maps to any letter $a \in A$ a nonempty set of processes. In this paper, $\mathcal{D}$ will always denote a distributed alphabet (A, P, loc).

For a word $w \in A^*$ we denote the set of processes associated with w by $\mathrm{loc}(w) := \bigcup_{a\in\mathrm{Alph}(w)} \mathrm{loc}(a) \subseteq P$. In particular, we set $\mathrm{loc}(\varepsilon) := \emptyset$. By $\pi_i\colon A^* \to A_i^*$ we denote the *projection* onto $A_i := \{a \in A \mid i \in \mathrm{loc}(a)\}$ (the alphabet of all letters associated to process i), i.e., the monoid morphism with $\pi_i(a) = a$ for $a \in A_i$ and $\pi_i(b) = \varepsilon$ for $b \in A \setminus A_i$.

Note that $\prod_{i\in P} A_i^*$ is a direct product of monoids and therefore a monoid itself (with componentwise concatenation). Since $\pi_i\colon A^* \to A_i^*$ is a monoid morphism for all $i \in P$, also the mapping

$$\overline{\pi}\colon A^* \to \prod_{i\in P} A_i^*\colon w \mapsto (\pi_i(w))_{i\in P}$$

is a monoid morphism. For $w \in A^*$, we call $\overline{\pi}(w)$ the *(Mazurkiewicz) trace* induced by w. The *trace monoid* is the submonoid of $\prod_{i\in P} A_i^*$ with universe $\mathbb{M}(\mathcal{D}) = \{\overline{\pi}(w) \mid w \in A^*\}$; its elements are *traces* and its subsets are *trace languages*.

We call two words $v, w \in A^*$ with $\mathrm{loc}(v) \cap \mathrm{loc}(w) = \emptyset$ *independent* and denote this fact by $v \parallel w$. We can see that $v \parallel w$ implies $\overline{\pi}(vw) = \overline{\pi}(wv)$.

Let $\mathfrak{A} = (Q, A, I, \delta, F)$ be an NFA. The *accepted trace language* of $\mathfrak{A}$ is $T(\mathfrak{A}) := \{\overline{\pi}(w) \mid I \xrightarrow{w}_{\mathfrak{A}} F\}$. In other words, $T(\mathfrak{A})$ is the image of the language $L(\mathfrak{A})$ under the morphism $\overline{\pi}$. A trace language $L \subseteq \mathbb{M}(\mathcal{D})$ is called *rational* if there is an NFA $\mathfrak{A}$ with $T(\mathfrak{A}) = L$, i.e., iff L is the image of some regular language in A^* under the morphism $\overline{\pi}$. A trace language L is *recognizable* iff its preimage under the morphism $\overline{\pi}$, i.e. $\{w \in A^* \mid \overline{\pi}(w) \in L\}$, is regular. Clearly, any recognizable trace language is rational. The converse implication holds only in case any two letters are dependent.

A finite automaton that reads letters of a distributed alphabet should consist of components for all $i \in P$ such that any letter $a \in A$ acts only on the components from $\mathrm{loc}(a)$. This idea leads to the following definition of an asynchronous automaton. But first, we fix a particular notation: For a tuple $(Q_i)_{i\in P}$ of finite sets Q_i, we write $\mathbf{Q}$ for the direct product $\prod_{i\in P} Q_i$.

Definition 2. Let $\mathcal{D} = (A, P, \mathrm{loc})$ be a distributed alphabet. An *asynchronous automaton* or *AA* is an NFA $\mathfrak{A} = (\mathbf{Q}, A, I, \delta, F)$ where $\mathbf{Q} = \prod_{i\in P} Q_i$ is the product of finite sets Q_i of *local states* — accordingly, the tuples from $\mathbf{Q}$ are called *global states* — and where, for every $(\overline{p}, a, \overline{q}) \in \delta$ and $\overline{r} \in \prod_{i\in P\setminus\mathrm{loc}(a)} Q_i$, we have

(i) $\overline{p}\restriction_{P\setminus\mathrm{loc}(a)} = \overline{q}\restriction_{P\setminus\mathrm{loc}(a)}$ and

(ii) $((\overline{p}\restriction_{\mathrm{loc}(a)}, \overline{r}), a, (\overline{q}\restriction_{\mathrm{loc}(a)}, \overline{r})) \in \delta$.

Here, (i) ensures that any a-transition of $\mathfrak{A}$ only modifies components from $\mathrm{loc}(a)$ while the other components are left untouched, and (ii) guarantees that a-transitions are insensitive to the local states of the components in $P \setminus \mathrm{loc}(a)$.

The transition relation δ defines, for each letter $a \in A$, a local transition relation $\delta_a \subseteq \prod_{i\in\mathrm{loc}(a)} Q_i \times \prod_{i\in\mathrm{loc}(a)} Q_i$ by $\delta_a = \{(\overline{p}\restriction_{\mathrm{loc}(a)}, \overline{q}\restriction_{\mathrm{loc}(a)}) \mid (\overline{p}, a, \overline{q}) \in \delta\}$. The above two conditions ensure that the collection of these local transition relations δ_a for $a \in A$ completely defines the transition relation δ: $(\overline{p}, a, \overline{q}) \in \delta$ if, and only if, $(\overline{p}\restriction_{\mathrm{loc}(a)}, \overline{q}\restriction_{\mathrm{loc}(a)}) \in \delta_a$ and $\overline{p}\restriction_{P\setminus\mathrm{loc}(a)} = \overline{q}\restriction_{P\setminus\mathrm{loc}(a)}$. Therefore, in the literature, asynchronous automata are often defined with the help of these local transition relations.

Every asynchronous automaton accepts a recognizable trace language. Conversely, Zielonka's celebrated result [1] states that, even more, every recognizable trace language $L \subseteq \mathbb{M}(\mathcal{D})$ is accepted by some deterministic asynchronous automaton.

Remark 3. Let $\mathcal{D} = (A, P, \mathrm{loc})$ be a distributed alphabet. While it is easy to check whether a given NFA $\mathfrak{A}$ is asynchronous, the question, whether $\mathfrak{A}$ is equivalent to an asynchronous automaton (i.e., whether $T(\mathfrak{A})$ is recognizable), is more manifold. Actually, this question depends on the underlying trace monoid $\mathbb{M}(\mathcal{D})$ resp. distributed alphabet $\mathcal{D}$. So, the recognizability problem is decidable if, and only if, $\mathbb{M}(\mathcal{D})$ is a free product of free commutative monoids if, and only if, the independence relation $\{(a, b) \in A^2 \mid a \parallel b\}$ is transitive [23].

3. Introducing Cooperating Multi-Pushdown Systems

An AA consists of several NFAs that synchronize by joint actions. In a similar manner, we will now consider several pushdown systems synchronizing by joint actions.

Recall that a pushdown system (or PDS) consists of a control unit (that can be in any of finitely many control states) and a pushdown (that can hold words over the pushdown alphabet A). Its transitions read the top letter a from the pushdown, write a word w onto it, and change the control state. In our model, we have a pushdown system $\mathfrak{P}_i$ for every $i \in P$ whose pushdown alphabet is A_i. These systems synchronize by the letters read and written onto their pushdown.

Definition 4. Let $\mathcal{D} = (A, P, \mathrm{loc})$ be a distributed alphabet. An *asynchronous multi-pushdown system* or *aPDS* is a tuple $\mathfrak{P} = (\mathbf{Q}, \Delta)$ where $\mathbf{Q} = \prod_{i\in P} Q_i$ holds for some finite sets Q_i of *local states* — accordingly, the tuples from $\mathbf{Q}$ are called *global states* — and $\Delta \subseteq \mathbf{Q} \times A \times A^* \times \mathbf{Q}$ is a finite set of *transitions* such that, for each transition $(\overline{p}, a, w, \overline{q}) \in \Delta$ and $\overline{r} \in \prod_{i\in P\setminus \mathrm{loc}(aw)} Q_i$, we have

(i) $\overline{p}\restriction_{P\setminus\mathrm{loc}(aw)} = \overline{q}\restriction_{P\setminus\mathrm{loc}(aw)}$ and

(ii) $((\overline{p}\restriction_{\mathrm{loc}(aw)}, \overline{r}), a, w, (q\restriction_{\mathrm{loc}(aw)}, \overline{r})) \in \Delta$.

Its size $\|\mathfrak{P}\|$ is $|\mathbf{Q}| + |\mathbf{Q}|^2 \cdot |A|^k$ where $k-1$ is the maximal length of a word written by any of the transitions (i.e., $\Delta \subseteq \mathbf{Q} \times A \times A^{<k} \times \mathbf{Q}$).

The set of configurations $\mathrm{Conf}_{\mathfrak{P}}$ of $\mathfrak{P}$ equals $\mathbf{Q} \times \mathbb{M}(\mathcal{D})$. We also define the one step relation $\vdash\, \subseteq \mathrm{Conf}_{\mathfrak{P}}^2$. It is the least relation on $\mathrm{Conf}_{\mathfrak{P}}$ such that for each transition $(\overline{p}, a, u, \overline{q}) \in \Delta$ and each word $x \in A^*$ we have $(\overline{p}, \overline{\pi}(ax)) \vdash (\overline{q}, \overline{\pi}(ux))$. The reflexive and transitive closure of $\vdash$ is the reachability relation $\vdash^*$.

Let C and D be sets of configurations.

- We write $C \vdash^* D$ if there are $c \in C$ and $d \in D$ with $c \vdash^* d$. If $C = \{c\}$ or $D = \{d\}$, resp., is a singleton, we also write $c \vdash^* D$ resp. $C \vdash^* d$. We use analogous notations for the relation $\vdash$.
- The set C is *rational* (*recognizable*, resp.) if, for all $\overline{q} \in Q$, the trace language $C_{\overline{q}} := \{\overline{\pi}(u) \mid (\overline{q}, \overline{\pi}(u)) \in C\}$ is rational (recognizable, resp.).

  Since it is undecidable whether a given rational trace language is recognizable (cf. Remark 3), the same undecidability transfers to rational sets of configurations.
- $\mathrm{pre}_{\mathfrak{P}}(C) := \{c \in \mathrm{Conf}_{\mathfrak{P}} \mid c \vdash C\}$ is the set of predecessors of configurations from C, and
$$\mathrm{pre}^*_{\mathfrak{P}}(C) := \bigcup_{k\in\mathbb{N}} \mathrm{pre}^k_{\mathfrak{P}}(C)$$
  is the set of configurations *backwards* reachable from some configuration in C.

The reachability relation for configurations of asynchronous multi-pushdown systems is, in general, undecidable:

Theorem 5. *There exists an aPDS with undecidable reachability relation $\vdash^*$.*

Proof. We start with a classical 2-pushdown system $\mathfrak{P}$ with an undecidable reachability relation (its set of states is Q and the two pushdowns use disjoint alphabets A_1 and A_2). Let $A = A_1 \cup A_2 \cup \{\top\}$ and $P = \{1, 2\}$. We consider the distributed alphabet $\mathcal{D}$ with $\mathrm{loc}(a) = \{i\}$ for $a \in A_i$ and $\mathrm{loc}(\top) = \{1, 2\}$.

We simulate $\mathfrak{P}$ by an aPDS $\mathfrak{P}'$ over $\mathcal{D}$ as follows. The first process of $\mathfrak{P}'$ stores the state of the simulated system $\mathfrak{P}$ together with a letter from A_1 or ε, i.e., $Q_1 = Q(A_1 \cup \{\varepsilon\})$, the second process can store a letter from A_2 or the empty word, i.e., $Q_2 = A_2 \cup \{\varepsilon\}$.

A transition $(p, (a, b), (u, v), q)$ of $\mathfrak{P}$ (that replaces a and b by u and v on the two pushdowns) is simulated by three transitions of the aPDS: $((p\varepsilon, .), a, \varepsilon, (pa, .))$ reads a from the first pushdown and stores it in the first local state; then $((., \varepsilon), b, \top, (., b))$ reads b from the second pushdown, stores it in the second local state, and puts $\top$ onto both pushdowns; finally, $((pa, b), \top, uv, (q\varepsilon, \varepsilon))$ replaces $\top$ by uv (i.e., $\pi_1(uv) = u$ is written onto the first pushdown and $\pi_2(uv) = v$ onto the second). $\square$

To obtain a model with a decidable reachability relation, we therefore have to restrict aPDS.[3] To this aim, we require that any transition can only write onto pushdowns it reads from.

Definition 6. Let $\mathcal{D} = (A, P, \mathrm{loc})$ be a distributed alphabet. A *cooperating multi-pushdown system* or *cPDS* is an aPDS $\mathfrak{P} = (\mathbf{Q}, \Delta)$ with $\mathrm{loc}(w) \subseteq \mathrm{loc}(a)$ for each transition $(\overline{p}, a, w, \overline{q}) \in \Delta$.

Let $\mathfrak{P} = (\mathbf{Q}, \Delta)$ be a cPDS and $(\overline{p}, a, w, \overline{q}) \in \Delta$ be a transition of $\mathfrak{P}$. Since we have $\mathrm{loc}(w) \subseteq \mathrm{loc}(a)$, the asynchronicity properties in cPDS can be simplified to

(i) $\overline{p}{\restriction}_{P\setminus\mathrm{loc}(a)} = \overline{q}{\restriction}_{P\setminus\mathrm{loc}(a)}$ and

(ii) $((\overline{p}{\restriction}_{\mathrm{loc}(a)}, \overline{r}), a, w, (q{\restriction}_{\mathrm{loc}(a)}, \overline{r})) \in \Delta$ for each $\overline{r} \in \prod_{i\in P\setminus\mathrm{loc}(a)} Q_i$.

This means, such transition does not touch the state of the processes not in $\mathrm{loc}(a)$ and is, additionally, independent of the actual state of the processes in $P \setminus \mathrm{loc}(a)$.

Similarly to the case of asynchronous automata, we can see the transition relation Δ as a family of *local* transition relations: for $a \in A$, let $\Delta_a \subseteq \prod_{i\in\mathrm{loc}(a)} Q_i \times A^* \times \prod_{i\in\mathrm{loc}(a)} Q_i$ be the collection of all tuples $(\overline{p}{\restriction}_{\mathrm{loc}(a)}, u, \overline{q}{\restriction}_{\mathrm{loc}(a)})$ with $(\overline{p}, a, u, \overline{q}) \in \Delta$. Again, these local relations completely determine the global relation. In the following we will use these local transition relations to emphasize the asynchronicity properties of $\mathfrak{P}$.

Example 7. Suppose $\mathcal{D} = (A, P, \mathrm{loc})$ with $A = \{a, b, c\}$, $P = \{1, 2\}$, $\mathrm{loc}(a) = P$, $\mathrm{loc}(b) = \{1\}$, and $\mathrm{loc}(c) = \{2\}$. We consider the cPDS $\mathfrak{P}$ from Fig. 1 where edges from global state $\overline{p}$ to global state $\overline{q}$ labeled $a \mid w$ visualize global transitions $(\overline{p}, a, w, \overline{q})$. The set of global states of $\mathfrak{P}$ is the product $\{p_1, q_1\} \times \{p_2, q_2\}$. Additionally, the transitions reading b and c only depend on process 1 and 2, resp. Since $\mathrm{loc}(b), \mathrm{loc}(c) \subseteq \mathrm{loc}(a)$, any global transition $(\overline{p}, x, w, \overline{q})$ satisfies $\mathrm{loc}(w) \subseteq \mathrm{loc}(x)$, i.e., $\mathfrak{P}$ is, indeed, a cPDS.

[3] The proof of Theorem 5 shows that requiring aw to be connected for any transition $(\overline{p}, a, w, \overline{q})$ does not yield decidability.

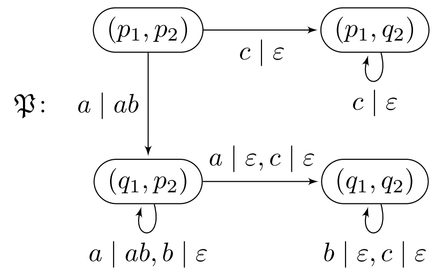

Figure 1: The cPDS $\mathfrak{P}$ from Example 7.

The following sequence is a run of $\mathfrak{P}$ from $((p_1, p_2), \overline{\pi}(ac))$ to $((q_1, q_2), \overline{\pi}(bb))$:

$$((p_1, p_2), \overline{\pi}(ac)) \vdash ((q_1, p_2), \overline{\pi}(abc)) \vdash ((q_1, p_2), \overline{\pi}(abbc)) \\ \vdash ((q_1, q_2), \overline{\pi}(bbc)) \vdash ((q_1, q_2), \overline{\pi}(bb)) \,.$$

3.1. Application: Transformers

Cooperating multi-pushdown systems can be used to model so-called *transformers*, a basic computation model in the area of machine learning used in recent large language models [24].

Let $\mathcal{A}$ be some finite set of *activation values* and $n \in \mathbb{N}$ be a natural number. There are two types of *simple transformers* called *layers*:

1. A *position-wise layer* L is a function $p \colon \mathcal{A} \to \mathcal{A}$. The position-wise layer L takes an input sequence $a_1 a_2 \cdots a_n \in \mathcal{A}^n$ and outputs the sequence $p(a_1)p(a_2) \cdots p(a_n) \in \mathcal{A}^n$. These position-wise layers are used to model the effect of neural networks.

2. An *attention layer* L (cf. Fig. 2) is a tuple $(\mathcal{S}, s, v)$ where $\mathcal{S}$ is a finite set of *score values*, $s \colon \mathcal{A} \times \mathcal{A} \to \mathcal{S}$ an *attention score function*, and $v \colon \mathcal{A} \times \mathcal{S}^n \to \mathcal{A}$ a *choice and valuation function*. The attention layer L takes an input sequence $a_1 a_2 \cdots a_n \in \mathcal{A}^n$. It then computes, for each pair $i, j \in [n]$, an attention score $s_{i,j} = s(a_i, a_j) \in \mathcal{S}$ combining the input values a_i and a_j using the function s. From a_i and the sequence of score values $s_{i,1} s_{i,2} \cdots s_{i,n}$, it then computes a new element $a'_i = v(a_i, s_{i,1} s_{i,2} \cdots s_{i,n})$ of $\mathcal{A}$. The word $a'_1 a'_2 \cdots a'_n$ is the output of the attention layer.

   For example, in so-called *unique hard attention transformers* (as defined in [25, 26]), $\mathcal{A} \subseteq \mathbb{R}^d$ is a finite set of real vectors, and $\mathcal{S} \subseteq \mathbb{R} \times \mathcal{A}$ is a finite set of reals with activation values. Then the function $s(a_i, a_j) = (x_j, a_j)$ outputs a_j and a product x_j of two affine functions to the vectors a_i and a_j. The valuation function $v(a_i, s_{i,1} s_{i,2} \cdots s_{i,n})$ first chooses the minimal position $j \in [n]$ such that x_j (where $s_{i,j} = (x_j, a_j)$) has the maximal attention score and then applies an affine function to a_i and a_j.

As another example, let $p\colon \mathcal{A} \to \mathcal{A}$ describe some position-wise layer. Set $\mathcal{S} = \{1\}$, $s(a, a') = 1$ for all $a, a' \in \mathcal{A}$ and $v(a, w) = p(a)$ for all $a \in \mathcal{A}$ and $w \in \mathcal{S}^n$. Then the attention layer $(\mathcal{S}, s, v)$ simulates the position-wise layer described by p. Hence, for our purposes, it suffices to consider attention layers.

A *transformer* is given by a finite sequence $L_1 L_2 \cdots L_k$ of (attention) layers $L_\ell = (\mathcal{S}_\ell, s_\ell, v_\ell)$ for $1 \leq \ell \leq k$, it computes the concatenation of the functions $\mathcal{A}^n \to \mathcal{A}^n$ given by these layers.

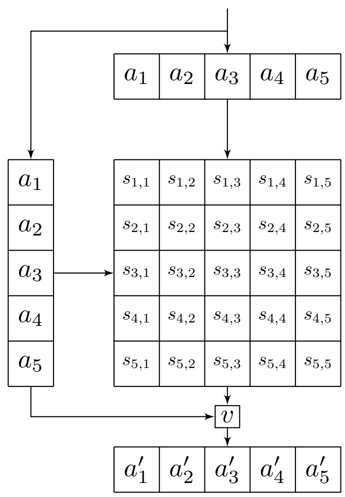

Figure 2: Visualization of an attention layer of a transformer.

We can model such a transformer as a cPDS as follows. The distributed alphabet $\mathcal{D} = (A, P, \mathrm{loc})$ uses the set of processes $P = [n]$, i.e., every position in the input word $a_1 \cdots a_n$ corresponds to some process. The letters from A correspond to the basic activities of the transformer, the association of processes to letters reflects the positions involved in the activity. More precisely, we have the following letters.

- $\mathtt{LAY}_\ell$ corresponds to the call of layer ℓ (for $1 \leq \ell \leq k$) and involves all processes, i.e., $\mathrm{loc}(\mathtt{LAY}_\ell) = P$.
- $\mathtt{S}_\ell^{i,j}$ corresponds to the computation of $s_\ell(a_i, a_j)$ (for $i, j \in [n]$ and $\ell \in [k]$) and involves the processes i and j, i.e., $\mathrm{loc}(\mathtt{S}_\ell^{i,j}) = \{i, j\}$.
- $\mathtt{V}_\ell^i$ corresponds to the computation of $v_\ell(a_i, s_{i,1} s_{i,2} \cdots s_{i,n})$ (for $i \in [n]$ and $\ell \in [k]$) and involves the process i, only, i.e., $\mathrm{loc}(\mathtt{V}_\ell^i) = \{i\}$.

We now describe a cPDS $\mathfrak{P} = (\mathbf{Q}, \Delta)$ over the above distributed alphabet $\mathcal{D}$ that simulates the transformer $L_1 L_2 \cdots L_k$ with $L_\ell = (\mathcal{S}_\ell, s_\ell, v_\ell)$ for $\ell \in [k]$. The aim here is to start with the letter a_i as state of process i and, at the end of the computation, to find the i-th letter in the state of process i. For notational simplicity, assume $\mathcal{S}_1 = \mathcal{S}_\ell$ for all $\ell \in [k]$ and denote this set with $\mathcal{S}$. Furthermore, let $\mathcal{S}_\perp = \mathcal{S} \cup \{\perp\}$ for some $\perp \notin \mathcal{S}$. Then elements of $\mathcal{S}_\perp^{[n]}$ describe a partial function from $[n]$ to $\mathcal{S}$.

Local states of process $p \in P = [n]$ consist of a letter a from $\mathcal{A}$ and a partial function w from $[n]$ to $\mathcal{S}$, i.e., $Q_p = (\mathcal{A}, \mathcal{S}_\perp^{[n]})$. Then we have the following transitions.

- In any global state $(a_p, w_p)_{p\in P}$, the letter $\mathtt{LAY}_\ell$ can be replaced by an arbitrary permutation of the letters $\mathtt{S}_\ell^{i,j}$ for $i, j \in [n]$ followed by an arbitrary permutation of the letters $\mathtt{V}_\ell^i$ for $i \in [n]$ (without changing the global state).

- Let $(a_p, w_p)_{p\in P}$ be a global state, $i, j \in [n]$, and $\ell \in [k]$. Then the letter $\mathtt{S}_\ell^{i,j}$ can be deleted from the pushdown while changing the global state to $(a_p, w'_p)_{p\in P}$ defined as follows:

$$w'_i(x) = \begin{cases} s_\ell(a_i, a_j) & \text{for } x = j \\ w_i(x) & \text{otherwise} \end{cases}$$
$$w'_p = w_p \text{ for } p \in [n] \setminus \{i\} = P \setminus \{i\}$$

- Let $(a_p, w_p)_{p\in P}$ be a global state, $i \in [n]$, and $\ell \in [k]$. Then the letter $\mathtt{V}_\ell^i$ can be deleted from the pushdown while changing the global state to $(a'_p, w'_p)_{p\in P}$ where $a'_i = v_\ell^i(a_i, w_i)$, $w'_i(x) = \perp$ for all $x \in [n]$, $a'_p = a_p$ and $w'_p = w_p$ for all $p \in [n] \setminus \{i\} = P \setminus \{i\}$.

Let $\overline{q} = (a_p, w_p)_{p\in P}$ and $\overline{q}' = (a'_p, w'_p)_{p\in P}$ be any global states. Then we have

$$(\overline{q}, \overline{\pi}(\mathtt{LAY}_\ell)) \vdash^* (\overline{q}', \overline{\pi}(\varepsilon))$$

if, and only if, $a'_1 a'_2 \cdots a'_n$ is the output of layer L_ℓ on input $a_1 a_2 \cdots a_n$, and $w'_p(i) = \perp$ for all $p \in P$ and $i \in [n]$. As a consequence,

$$(\overline{q}, \overline{\pi}(\mathtt{LAY}_1 \cdots \mathtt{LAY}_k)) \vdash^* (\overline{q}', \overline{\pi}(\varepsilon))$$

if, and only if, $a'_1 a'_2 \cdots a'_n$ is the output of the transformer $L = L_1 \cdots L_k$ on input $a_1 a_2 \cdots a_n$, and $w'_p(i) = \perp$ for all $p \in P$ and $i \in [n]$.

3.2. Recognizable sets of configurations

We return to the consideration of general cPDS. In order to decide the reachability relation, we will compute, from a set of configurations C, the set $\mathrm{pre}^*_{\mathfrak{P}}(C)$, i.e., the set of configurations that allow to reach some configuration from C or, put alternatively, the set of configurations backwards reachable from C. To represent possibly infinite sets of configurations, we use finite representations of sets of configurations. If the set of configurations C is rational, then (by definition) all the trace languages $C_{\overline{q}} = \{\overline{\pi}(w) \mid (\overline{q}, \overline{\pi}(w)) \in C\}$ are rational.

Hence we can represent C by a tuple of NFAs $\mathfrak{A}_{\overline{q}}$ accepting the trace language $C_{\overline{q}}$ (one for each global state $\overline{q}$ of $\mathfrak{P}$).

Alternatively, C can be recognizable such that, by definition, all the languages $C_{\overline{q}}$ are recognizable. Then we can represent each of the languages $C_{\overline{q}}$ by an asynchronous automaton $\mathfrak{A}_{\overline{q}}$. Since $\overline{q}$ is a P-tuple, we can assume, without loss of generality, that $\overline{q}$ is the only initial state of the AA $\mathfrak{A}_{\overline{q}}$ (in particular, local states of the cPDS are also local states of the AA, but the AA can have more local states). Following Bouajjani et al. [3], we can further assume that all these AAs differ in their initial state, only. — This idea leads to the concept of a $\mathfrak{P}$-AA given next.

Definition 8. Let $\mathcal{D} = (A, P, \mathrm{loc})$ be a distributed alphabet and $\mathfrak{P} = (\mathbf{Q}, \Delta)$ be a cPDS. A $\mathfrak{P}$-*asynchronous automaton* or $\mathfrak{P}$-*AA* is an AA $\mathfrak{A} = (\mathbf{S}, A, \emptyset, \delta, F)$ such that $Q_i \subseteq S_i$ for all $i \in P$.

The $\mathfrak{P}$-AA $\mathfrak{A}$ *accepts* the following set $C(\mathfrak{A})$ of configurations of $\mathfrak{P}$:

$$\{(\overline{q}, \overline{\pi}(w)) \in \mathrm{Conf}_{\mathfrak{P}} \mid \overline{q} \in \mathbf{Q}, \overline{q} \xrightarrow{w}_{\mathfrak{A}} F\}$$

In other words, the $\mathfrak{P}$-AA $\mathfrak{A}$ accepts a configuration $(\overline{q}, \overline{\pi}(w))$ if, from the state $\overline{q}$ of $\mathfrak{A}$, the AA $\mathfrak{A}$ can reach some accepting state.

The above arguments prove the following result.

Observation 9. *Let $\mathcal{D} = (A, P, \mathrm{loc})$ be a distributed alphabet and $\mathfrak{P} = (\mathbf{Q}, \Delta)$ be a cPDS. A set of configurations $C \subseteq \mathrm{Conf}_{\mathfrak{P}}$ is* recognizable *if, and only if, there is a $\mathfrak{P}$-AA $\mathfrak{A}$ with $C(\mathfrak{A}) = C$.*

In a $\mathfrak{P}$-AA $\mathfrak{A}$, any local state of the cPDS $\mathfrak{P}$ is also a local state of the $\mathfrak{P}$-AA $\mathfrak{A}$. In particular, the asynchronous automaton can move from a local state not belonging to $\mathfrak{P}$ into a local state from $\mathfrak{P}$. For later use, we now demonstrate that this behavior can be suppressed. So let $\mathfrak{A} = (\mathbf{S}, A, \emptyset, \delta, F)$ be a $\mathfrak{P}$-AA for some cPDS $\mathfrak{P} = (\mathbf{Q}, \Delta)$. For any process $i \in P$, let S_i' be a disjoint copy of S_i. Then $S_i \cup S_i'$ forms the set of local states of process i in the new $\mathfrak{P}$-AA $\mathfrak{A}^{\mathrm{new}}$.

For a letter $a \in A$ and tuples of (new) local states $\overline{s}, \overline{t} \in \prod_{i\in\mathrm{loc}(a)}(S_i \cup S_i')$, we set $(\overline{s}, \overline{t}) \in \delta_a^{\mathrm{new}}$ if, and only if, the undashed versions of $\overline{s}$ and $\overline{t}$ form a pair from δ_a and all entries of $\overline{t}$ are dashed, i.e., $\overline{t} \in \prod_{i\in\mathrm{loc}(a)} S_i'$. As a result, we get

$$\delta_a^{\mathrm{new}} \subseteq \prod_{i\in\mathrm{loc}(a)} (S_i \cup S_i') \times \prod_{i\in\mathrm{loc}(a)} S_i' \subseteq \prod_{i\in\mathrm{loc}(a)} (S_i \cup S_i') \times \prod_{i\in\mathrm{loc}(a)} (S_i \cup S_i') \setminus Q_i\,.$$

Furthermore, $\overline{s} \in \prod_{i\in P}(S_i \cup S_i')$ is accepting in $\mathfrak{A}^{\mathrm{new}}$ if, and only if, the undashed version of $\overline{s}$ is accepting in $\mathfrak{A}$. Since the $\mathfrak{P}$-AAs $\mathfrak{A}$ and $\mathfrak{A}^{\mathrm{new}}$ accept the same sets of configurations, we obtain

Lemma 10. *From a distributed alphabet $\mathcal{D} = (A, P, \mathrm{loc})$, a cPDS $\mathfrak{P} = (\mathbf{Q}, \Delta)$, and a $\mathfrak{P}$-AA $\mathfrak{A}$, we can construct in polynomial time an equivalent $\mathfrak{P}$-AA $(\mathbf{S}, A, \emptyset, \delta, F)$ such that $\overline{t} \in \prod_{i\in\mathrm{loc}(a)} S_i \setminus Q_i$ for any local transition $(\overline{s}, \overline{t}) \in \delta_a$ and $a \in A$.*

4. Computing the Backwards Reachable Configurations

In this section we want to compute the backwards reachable configurations in a cPDS $\mathfrak{P}$. The main result of this section states that the mapping $\mathrm{pre}^*_{\mathfrak{P}}$ effectively preserves the recognizability of sets of configurations.

Theorem 11. *Let $\mathcal{D} = (A, P, \mathrm{loc})$ be a distributed alphabet, $\mathfrak{P} = (\mathbf{Q}, \Delta)$ be a cPDS, and $C \subseteq \mathrm{Conf}_{\mathfrak{P}}$ be a recognizable set of configurations. Then the set $\mathrm{pre}^*_{\mathfrak{P}}(C)$ is recognizable.*

Even more, from $\mathcal{D}$, $\mathfrak{P}$, and a $\mathfrak{P}$-AA $\mathfrak{A}^{(0)} = (\mathbf{S}, A, \emptyset, \delta^{(0)}, F)$, one can construct in polynomial time a $\mathfrak{P}$-AA $\mathfrak{A}$ that accepts the set $\mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$.

The rest of this section is devoted to the proof of this result.

Adapting ideas by Bouajjani et al. [3] from NFAs to AA, we construct a $\mathfrak{P}$-AA $\mathfrak{A}$ that accepts the set $\mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$ of configurations backwards reachable from $C(\mathfrak{A}^{(0)})$. To this aim, we will inductively add new transitions to the $\mathfrak{P}$-AA $\mathfrak{A}^{(0)} = (\mathbf{S}, A, \emptyset, \delta^{(0)}, F)$, but leave the sets of states, initial states, and accepting states unchanged. By Lemma 10, we can assume (and this assumption is crucial for the correctness of the construction) that the automaton cannot enter a local state from the cPDS $\mathfrak{P}$, i.e., we have $\overline{q} \in \prod_{i \in \mathrm{loc}(a)} S_i \setminus Q_i$ for any local transition $(\overline{p}, \overline{q}) \in \delta^{(0)}_a$ and any letter $a \in A$.

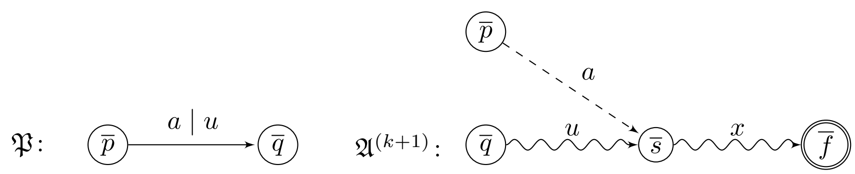

Figure 3: Visualization of the construction of $\mathfrak{A}^{(k+1)}$.

For a start, and to explain the idea, let $(\overline{p}, \overline{\pi}(v))$ and $(\overline{q}, \overline{\pi}(w))$ be configurations such that $(\overline{p}, \overline{\pi}(v)) \vdash (\overline{q}, \overline{\pi}(w))$ and $(\overline{q}, \overline{\pi}(w)) \in C(\mathfrak{A}^{(0)})$. Then the configuration $(\overline{p}, \overline{\pi}(v))$ is backwards reachable from $C(\mathfrak{A}^{(0)})$ and we will add, in a first step, a transition to the $\mathfrak{P}$-AA $\mathfrak{A}^{(0)}$ making sure that also this configuration $(\overline{p}, \overline{\pi}(v))$ is accepted (cf. Fig. 3). Since $(\overline{p}, \overline{\pi}(v)) \vdash (\overline{q}, \overline{\pi}(w))$, there is a local a-transition $(\overline{p}\restriction_{\mathrm{loc}(a)}, u, \overline{q}\restriction_{\mathrm{loc}(a)})$ in $\mathfrak{P}$ and a word $x \in A^*$ with $\overline{\pi}(v) = \overline{\pi}(ax)$ and $\overline{\pi}(w) = \overline{\pi}(ux)$. Since the configuration $(\overline{q}, \overline{\pi}(w)) = (\overline{q}, \overline{\pi}(ux))$ is accepted by the $\mathfrak{P}$-AA $\mathfrak{A}^{(0)}$, there is a state $\overline{s} \in \mathbf{S}$ such that

$$\overline{q} \xrightarrow{u}_{\mathfrak{A}^{(0)}} \overline{s} \xrightarrow{x}_{\mathfrak{A}^{(0)}} F\,.$$

We now add the local a-transition $(\overline{p}\restriction_{\mathrm{loc}(a)}, \overline{s}\restriction_{\mathrm{loc}(a)})$ to $\mathfrak{A}^{(0)}$, i.e., $\delta^{(1)}_a$ contains, in addition to all a-transitions from $\delta^{(0)}_a$, this local transition. Let $\mathfrak{A}^{(1)}$ denote the result of this addition. Then we get

$$\overline{p} \xrightarrow{a}_{\mathfrak{A}^{(1)}} \overline{s} \xrightarrow{x}_{\mathfrak{A}^{(1)}} F$$

implying that the configuration $(\overline{p}, \overline{\pi}(v)) = (\overline{p}, \overline{\pi}(ax))$ is accepted by the $\mathfrak{P}$-NFA $\mathfrak{A}^{(1)}$.

Since we added a local a-transition we can ensure that the $\mathfrak{P}$- NFA $\mathfrak{A}^{(1)}$ is also asynchronous.

Remark 12. The construction as described above requires $\mathfrak{P}$ to be cooperating. Assume that $(\overline{p}, a, u, \overline{q})$ is a transition in $\mathfrak{P}$ violating the cooperation property $\mathrm{loc}(u) \subseteq \mathrm{loc}(a)$ and that there is a process $i \in \mathrm{loc}(u) \setminus \mathrm{loc}(a)$ with $p_i \neq q_i$. If $\mathfrak{A}^{(0)}$ satisfies $\overline{q} \xrightarrow{u}_{\mathfrak{A}(0)} \overline{s}$, then the new transition $(\overline{p}, a, \overline{s})$ would depend also on process i. This implies that $\mathfrak{A}^{(1)}$ is not asynchronous anymore.

Formally, we construct $\mathfrak{P}$-asynchronous automata $\mathfrak{A}^{(k)} = (\mathbf{S}, A, \emptyset, \delta^{(k)}, F)$ for $k \geq 1$ as follows: for $k \in \mathbb{N}$ and $a \in A$ we define the local transition relation $\delta_a^{(k+1)}$ to be the set

$$\delta_a^{(k)} \cup \left\{ \left(\overline{p}\restriction_{\mathrm{loc}(a)}, \overline{s}\restriction_{\mathrm{loc}(a)}\right) \;\middle|\; \begin{array}{l} \overline{p} \in \mathbf{Q}, \overline{s} \in \mathbf{S}, \\ \exists\, \overline{q} \in \mathbf{Q}, u \in A^* \colon (\overline{p}\restriction_{\mathrm{loc}(a)}, u, \overline{q}\restriction_{\mathrm{loc}(a)}) \in \Delta_a,\ \overline{q} \xrightarrow{u}_{\mathfrak{A}^{(k)}} \overline{s} \end{array} \right\} .$$

Since we constructed $\mathfrak{A}^{(k+1)}$ from $\mathfrak{A}^{(k)}$ using local transitions it is clear that the properties of Definition 2 are satisfied. Hence, $\mathfrak{A}^{(k+1)}$ is also asynchronous.

The "limit" of this construction is the $\mathfrak{P}$-AA $\mathfrak{A}^{(\infty)} = (\mathbf{S}, A, \emptyset, \delta^{(\infty)}, F)$ with $\delta^{(\infty)} = \bigcup_{k \in \mathbb{N}} \delta^{(k)}$.

Example 13. Recall the cPDS $\mathfrak{P}$ from Example 7. In Fig. 4 we depict our algorithm on input $\mathfrak{P}$ and the set of configurations $C = \{((q_1, q_2), \varepsilon)\}$. A $\mathfrak{P}$-AA $\mathfrak{A}^{(0)} = (S_1 \times S_2, A, \emptyset, \delta, F)$ accepting this set is depicted in the left.

In $\mathfrak{A}^{(1)}$, we have $(q_1, p_2) \xrightarrow{ab}_{\mathfrak{A}^{(1)}} (q_1, q_2)$ (depicted in bold and red) and, in $\mathfrak{P}$, we have the transition $((p_1, p_2), a, ab, (q_1, p_2)) \in \Delta$. The definition of $\delta^{(2)}$ implies that $((p_1, p_2), a, (q_1, q_2))$ is a new local transition.

The construction terminates with $\mathfrak{A}^{(2)}$. This is a $\mathfrak{P}$-AA accepting the union of the sets of configurations $\left\{((p_1, p_2), \overline{\pi}(w)) \;\middle|\; w \in a\{b, c\}^*\right\}$, $\left\{((q_1, p_2), \overline{\pi}(w)) \;\middle|\; w \in b^*\{a, c\}\{b, c\}^*\right\}$, and $\left\{((q_1, q_2), \overline{\pi}(w)) \;\middle|\; w \in \{b, c\}^*\right\}$. But this is exactly the set of configurations backwards reachable from $C = \left\{((q_1, q_2), \varepsilon)\right\}$.

Remark 14. The inductive construction of $\mathfrak{A}^{(k)}$ is not possible if $\mathfrak{A}^{(0)}$ is not asynchronous. To this end, let $(\overline{q}, a, u, \overline{p}) \in \delta$ be a transition of $\mathfrak{P}$ and $b \in A$ with $a \parallel b$. Now, assume that $(\overline{p}, \overline{\pi}(ubx)) \in \mathrm{Conf}_{\mathfrak{P}}$ is accepted by a $\mathfrak{P}$-NFA $\mathfrak{A}^{(k)}$. Then we have $(\overline{q}, \overline{\pi}(abx)) \in \mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(k)}))$. Suppose that the only run of $\mathfrak{A}^{(k)}$ accepting $\overline{\pi}(ubx)$ is the following one:

$$\overline{p} \xrightarrow{b}_{\mathfrak{A}^{(k)}} s' \xrightarrow{u}_{\mathfrak{A}^{(k)}} s \xrightarrow{x}_{\mathfrak{A}^{(k)}} F\,.$$

Then we have to add a new path from $\overline{q}$ to $\overline{s}$ labeled with ab. To this end, we have to introduce one new state. Hence, the number of states of $\mathfrak{A}^{(k+1)}$ may increase in each iteration.

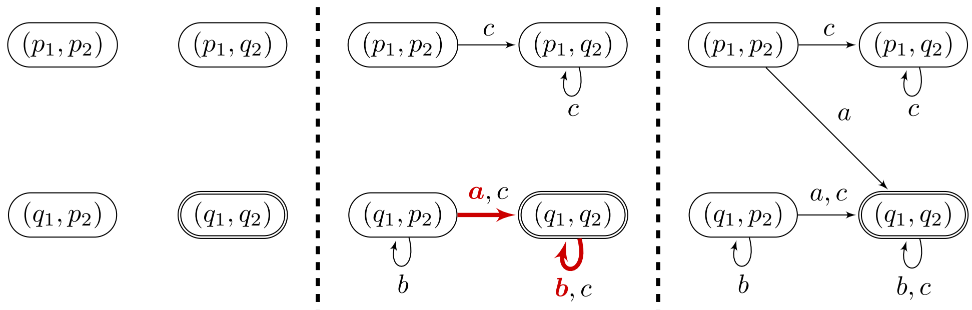

Figure 4: The $\mathfrak{P}$-AA $\mathfrak{A}^{(0)}$, $\mathfrak{A}^{(1)}$, and $\mathfrak{A}^{(2)}$ (from left to right) from Example 13.

In contrast, runs starting with some independent letters are not a problem if $\mathfrak{A}^{(k)}$ is asynchronous: since b-edges only modify the processes in $\mathrm{loc}(b)$, the u-labeled run only affects the processes in $\mathrm{loc}(u) \subseteq \mathrm{loc}(a)$, and since $\mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset$ holds due to $a \parallel b$, there would be another run

$$\overline{p} \xrightarrow{u}_{\mathfrak{A}^{(k)}} s'' \xrightarrow{b}_{\mathfrak{A}^{(k)}} s \xrightarrow{x}_{\mathfrak{A}^{(k)}} F$$

which starts with u.

Now, we show $C(\mathfrak{A}^{(\infty)}) = \mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$ with the help of the following three lemmas. First, by induction on $k \in \mathbb{N}$, one can easily prove $\mathrm{pre}^k_{\mathfrak{P}}(C(\mathfrak{A}^{(0)})) \subseteq C(\mathfrak{A}^{(k)})$ (which ensures the inclusion "$\supseteq$").

Lemma 15. *Let $k \in \mathbb{N}$. Then $\mathrm{pre}^k_{\mathfrak{P}}(C(\mathfrak{A}^{(0)})) \subseteq C(\mathfrak{A}^{(k)})$. In particular, we have $\mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)})) \subseteq C(\mathfrak{A}^{(\infty)})$.*

Proof. We prove the first statement by induction on $k \in \mathbb{N}$. The case $k = 0$ is obvious by $\mathrm{pre}^0_{\mathfrak{P}}(C(\mathfrak{A}^{(0)})) = C(\mathfrak{A}^{(0)})$. Now, let $k \geq 0$ and $(\overline{q}, \overline{\pi}(w)) \in \mathrm{pre}^{k+1}_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$. Then there is a configuration $(\overline{p}, \overline{\pi}(v)) \in \mathrm{pre}^k_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$ with $(\overline{q}, \overline{\pi}(w)) \vdash (\overline{p}, \overline{\pi}(v))$. By definition of $\vdash$ there is a transition $(\overline{p}, a, u, \overline{q}) \in \Delta$ and a word $x \in A^*$ with $\overline{\pi}(w) = \overline{\pi}(ax)$ and $\overline{\pi}(v) = \overline{\pi}(ux)$. By the induction hypothesis we know $(\overline{p}, \overline{\pi}(ux)) = (\overline{p}, \overline{\pi}(v)) \in C(\mathfrak{A}^{(k)})$. Hence, there is $\overline{s} \in \mathbf{S}$ with

$$\overline{p} \xrightarrow{u}_{\mathfrak{A}^{(k)}} \overline{s} \xrightarrow{x}_{\mathfrak{A}^{(k)}} F\,.$$

By $(\overline{p}, a, u, \overline{q}) \in \Delta$ and $\overline{p} \xrightarrow{u}_{\mathfrak{A}^{(k)}} \overline{s}$, we obtain a transition $(\overline{q}, a, \overline{s}) \in \delta^{(k+1)}$ and, hence,

$$\overline{q} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{s} \xrightarrow{x}_{\mathfrak{A}^{(k)}} F\,.$$

Since $\delta^{(k)} \subseteq \delta^{(k+1)}$ we finally obtain $(\overline{q}, \overline{\pi}(w)) = (\overline{q}, \overline{\pi}(ax)) \in C(\mathfrak{A}^{(k+1)})$.

Towards the second statement, recall that we have $\delta^{(0)} \subseteq \delta^{(1)} \subseteq \cdots \subseteq \delta^{(\infty)}$. From this fact we can infer $C(\mathfrak{A}^{(0)}) \subseteq C(\mathfrak{A}^{(1)}) \subseteq \cdots \subseteq C(\mathfrak{A}^{(\infty)})$. Then the first statement of this lemma implies the following inclusion:

$$\mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)})) = \bigcup_{k \in \mathbb{N}} \mathrm{pre}^k_{\mathfrak{P}}(C(\mathfrak{A}^{(0)})) \subseteq \bigcup_{k \in \mathbb{N}} C(\mathfrak{A}^{(k)}) = C(\mathfrak{A}^{(\infty)})\,. \qquad \square$$

Next, we want to show the converse inclusion $C(\mathfrak{A}^{(\infty)}) \subseteq \mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$. However, we could not just prove $C(\mathfrak{A}^{(k)}) \subseteq \mathrm{pre}^k_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$ inductively for each $k \in \mathbb{N}$. The $\mathfrak{P}$-AA $\mathfrak{A}^{(k)}$ can in particular accept more configurations than those that are backwards reachable from $C(\mathfrak{A}^{(0)})$ in at most k steps: consider Example 13. The configuration $c = ((p_1, p_2), \overline{\pi}(ac^5))$ is accepted by $\mathfrak{A}^{(2)}$ depicted in Fig. 4. On the other hand, any configuration from $C(\mathfrak{A}^{(0)})$ has an empty pushdown and any step in the cPDS $\mathfrak{P}$ decreases the size of the pushdowns by at most one. Hence, indeed, c is not backwards reachable from $C(\mathfrak{A}^{(0)})$ in two steps.

Therefore, to prove $C(\mathfrak{A}^{(\infty)}) \subseteq \mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$ we need the following, more technical lemma.

Lemma 16. *Let $k \in \mathbb{N}$, $v \in A^*$, $\overline{p} \in \mathbf{Q}$, and $\overline{s} \in \mathbf{S}$ with $\overline{p} \xrightarrow{v}_{\mathfrak{A}^{(k)}} \overline{s}$. Then there are a global state $\overline{r} \in \mathbf{Q}$ and a word $w \in A^*$ with the following properties:*

(a) $(\overline{p}, \overline{\pi}(v)) \vdash^* (\overline{r}, \overline{\pi}(w))$ *and*

(b) $\overline{r} \xrightarrow{w}_{\mathfrak{A}^{(0)}} \overline{s}$.

Proof. The proof of this lemma proceeds by double induction, the first one over k and the inductive step for this induction proceeds by induction on the length of the word v. To simplify bookkeeping, let $\mathrm{Cl}(k, n)$ (for natural numbers k and n) be the following claim:

"For all $v \in A^n$, $\overline{p} \in \mathbf{Q}$, and $\overline{s} \in \mathbf{S}$ with $\overline{p} \xrightarrow{v}_{\mathfrak{A}^{(k)}} \overline{s}$, there are $\overline{r} \in \mathbf{Q}$ and $w \in A^*$ satisfying (a) and (b)."

Then $\mathrm{Cl}(k)$ is the claim "$\mathrm{Cl}(k, n)$ holds for all $n \in \mathbb{N}$".

So we prove the lemma by showing $\mathrm{Cl}(k)$ for all $k \in \mathbb{N}$ by induction on k.

The claim $\mathrm{Cl}(0, n)$ is trivial for all $n \in \mathbb{N}$ since we can set $\overline{r} = \overline{p}$ and $w = v$. Hence $\mathrm{Cl}(0)$ holds.

Now let $k \in \mathbb{N}$ and suppose the claim $\mathrm{Cl}(k)$ holds. We prove $\mathrm{Cl}(k+1)$, i.e., validity of $\mathrm{Cl}(k+1, n)$ for all $n \in \mathbb{N}$, by induction on n.

For $n = 0$, we only have to consider the word $v = \varepsilon$. But then $\overline{p} = \overline{s}$. Hence setting $\overline{r} = \overline{p}$ and $w = v = \varepsilon$ yields (a) and (b).

Before we proceed inductively, we also prove $\mathrm{Cl}(k+1, 1)$ explicitly. So let $v = a \in A$, $\overline{p} \in \mathbf{Q}$, and $\overline{s} \in \mathbf{S}$ with $\overline{p} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{s}$.

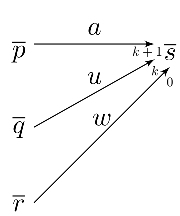

Figure 5: Proof of Lemma 16, validation of $\mathrm{Cl}(k+1, 1)$. The natural number ℓ at the tip of an arrow indicates a path in the $\mathfrak{P}$-AA $\mathfrak{A}^{(\ell)}$.

If even $\overline{p} \xrightarrow{a}_{\mathfrak{A}^{(k)}} \overline{s}$, claim $\mathrm{Cl}(k)$ yields $\overline{r}$ and w as desired. Otherwise, we have $(\overline{p}\upharpoonright_{\mathrm{loc}(a)}, \overline{s}\upharpoonright_{\mathrm{loc}(a)}) \in \delta_a^{(k+1)} \setminus \delta_a^{(k)}$ (see Fig. 5). By the definition of this local transition relation, there are global states $\overline{p'}, \overline{q'} \in \mathbf{Q}$ and $\overline{s'} \in \mathbf{S}$ and a word $u \in A^*$ such that

- $\overline{p}\upharpoonright_{\mathrm{loc}(a)} = \overline{p'}\upharpoonright_{\mathrm{loc}(a)}$ and $\overline{s}\upharpoonright_{\mathrm{loc}(a)} = \overline{s'}\upharpoonright_{\mathrm{loc}(a)}$,
- $(\overline{p'}\upharpoonright_{\mathrm{loc}(a)}, u, \overline{q'}\upharpoonright_{\mathrm{loc}(a)}) \in \Delta_a$, and
- $\overline{q'} \xrightarrow{u}_{\mathfrak{A}^{(k)}} \overline{s'}$.

Set $\overline{q} = (\overline{q'}\upharpoonright_{\mathrm{loc}(a)}, \overline{p}\upharpoonright_{P\setminus\mathrm{loc}(a)})$. Then $\overline{q}$ is a global state from $\mathbf{Q}$. Since $\mathfrak{P}$ is a cPDS and $(\overline{p'}\upharpoonright_{\mathrm{loc}(a)}, u, \overline{q'}\upharpoonright_{\mathrm{loc}(a)}) \in \Delta_a$, we can infer

$$(\overline{p}, \overline{\pi}(a)) = ((\overline{p'}\upharpoonright_{\mathrm{loc}(a)}, \overline{p}\upharpoonright_{P\setminus\mathrm{loc}(a)}), \overline{\pi}(a)) \vdash ((\overline{q'}\upharpoonright_{\mathrm{loc}(a)}, \overline{p}\upharpoonright_{P\setminus\mathrm{loc}(a)}), \overline{\pi}(u)) = (\overline{q}, \overline{\pi}(u))\,.$$

From $\overline{p} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{s}$, the asynchronicity of $\mathfrak{A}^{(k+1)}$ implies that the global states $\overline{p}$ and $\overline{s}$ agree on the components from $P \setminus \mathrm{loc}(a)$. Hence we get

$$\overline{q} = (\overline{q'}\upharpoonright_{\mathrm{loc}(a)}, \overline{p}\upharpoonright_{P\setminus\mathrm{loc}(a)}) = (\overline{q'}\upharpoonright_{\mathrm{loc}(a)}, \overline{s}\upharpoonright_{P\setminus\mathrm{loc}(a)})\,.$$

Since the local a-transition $(\overline{p'}\upharpoonright_{\mathrm{loc}(a)}, u, \overline{q'}\upharpoonright_{\mathrm{loc}(a)}) \in \Delta_a$ reads a and writes u and since $\mathfrak{P}$ is cooperating, we have $\mathrm{loc}(u) \subseteq \mathrm{loc}(a)$. Hence $\overline{q'} \xrightarrow{u}_{\mathfrak{A}^{(k)}} \overline{s'}$ and the asynchronicity of $\mathfrak{A}^{(k+1)}$ implies

$$\overline{q} = (\overline{q'}\upharpoonright_{\mathrm{loc}(a)}, \overline{s}\upharpoonright_{P\setminus\mathrm{loc}(a)}) \xrightarrow{u}_{\mathfrak{A}^{(k)}} (\overline{s'}\upharpoonright_{\mathrm{loc}(a)}, \overline{s}\upharpoonright_{P\setminus\mathrm{loc}(a)})\,.$$

Finally, $\overline{s}\upharpoonright_{\mathrm{loc}(a)} = \overline{s'}\upharpoonright_{\mathrm{loc}(a)}$ implies

$$(\overline{s'}\upharpoonright_{\mathrm{loc}(a)}, \overline{s}\upharpoonright_{P\setminus\mathrm{loc}(a)}) = \overline{s}\,.$$

In summary, we have

$$\overline{q} \xrightarrow{u}_{\mathfrak{A}^{(k)}} \overline{s}\,.$$

From $\mathrm{Cl}(k)$, we obtain a global state $\overline{r} \in \mathbf{Q}$ and a word $w \in A^*$ such that

$$(\overline{q}, \overline{\pi}(u)) \vdash^* (\overline{r}, \overline{\pi}(w)) \quad \text{and} \quad \overline{r} \xrightarrow{w}_{\mathfrak{A}^{(0)}} \overline{s}\,.$$

Putting everything together, we obtain

(a) $(\overline{p}, \overline{\pi}(v)) = (\overline{p}, \overline{\pi}(a)) \vdash (\overline{q}, \overline{\pi}(u)) \vdash^* (\overline{r}, \overline{\pi}(w))$ and

(b) $\overline{r} \xrightarrow{w}_{\mathfrak{A}^{(0)}} \overline{s}$

which completes the proof of $\mathrm{Cl}(k+1, 1)$.

From now on, assume that $\mathrm{Cl}(k+1, n)$ as well as $\mathrm{Cl}(k)$ hold. To verify $\mathrm{Cl}(k+1, n+1)$ for $n \geq 1$, let $\overline{p} \in \mathbf{Q}$, $\overline{s} \in \mathbf{S}$, and $v \in A^{n+1}$ such that $\overline{p} \xrightarrow{v}_{\mathfrak{A}^{(k+1)}} \overline{s}$.

Then we can write $v = v'a$ with $v' \in A^n$ and $a \in A$. Since $\overline{p} \xrightarrow{v'a}_{\mathfrak{A}^{(k+1)}} \overline{s}$, there is some global state $\overline{s'} \in \mathbf{S}$ with

$$\overline{p} \xrightarrow{v'}_{\mathfrak{A}^{(k+1)}} \overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{s}\,.$$

Since $|v'| = n$, claim $\mathrm{Cl}(k+1, n)$ provides a global state $\overline{q'} \in \mathbf{Q}$ and a word $w' \in A^*$ with

$$(\overline{p}, \overline{\pi}(v')) \vdash^* (\overline{q'}, \overline{\pi}(w')) \quad \text{and} \quad \overline{q'} \xrightarrow{w'}_{\mathfrak{A}^{(0)}} \overline{s'}\,.$$

Note that the former implies in particular $(\overline{p}, \overline{\pi}(v)) = (\overline{p}, \overline{\pi}(v'a)) \vdash^* (\overline{q'}, \overline{\pi}(w'a))$.

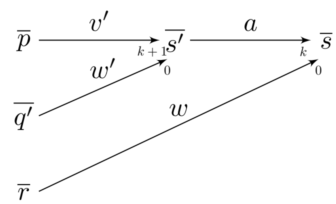

Figure 6: Proof of Lemma 16, validation of $\mathrm{Cl}(k+1, n+1)$ with $\overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k)}} \overline{s}$

Suppose that we do not only have $\overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{s}$, but even $\overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k)}} \overline{s}$ (see Fig. 6). Then $\mathfrak{A}^{(k)}$ has a $w'a$-labeled run from $\overline{q'}$ to $\overline{s}$. Hence, claim $\mathrm{Cl}(k)$ implies the existence of $\overline{r} \in \mathbf{Q}$ and $w \in A^*$ with

$$(\overline{q'}, \overline{\pi}(w'a)) \vdash^* (\overline{r}, \overline{\pi}(w)) \quad \text{and} \quad \overline{r} \xrightarrow{w}_{\mathfrak{A}^{(0)}} \overline{s}\,.$$

Note that the latter is (b). But also (a) holds since

$$(\overline{p}, \overline{\pi}(v)) \vdash^* (\overline{q'}, \overline{\pi}(w'a)) \vdash^* (\overline{r}, \overline{\pi}(w))$$

which completes the proof in case we even have an a-labeled in run $\mathfrak{A}^{(k)}$.

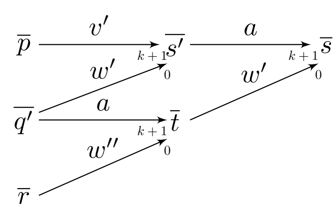

Figure 7: Proof of Lemma 16, validation of $\mathrm{Cl}(k+1, n+1)$ if $\overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k)}} \overline{s}$ does not hold

It remains to consider the case that no such run exists, i.e., we have $\overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{s}$, but not $\overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k)}} \overline{s}$ (see Fig. 7). This is equivalent to saying

$$(\overline{s'}\restriction_{\mathrm{loc}(a)}, \overline{s}\restriction_{\mathrm{loc}(a)}) \in \delta_a^{(k+1)} \setminus \delta_a^{(k)}\,.$$

The definition of the local transition relation $\delta_a^{(k+1)}$ yields in particular $\overline{s'}\restriction_{\mathrm{loc}(a)} \in \prod_{i\in\mathrm{loc}(a)} Q_i$. Recall that in $\mathfrak{P}$-AA $\mathfrak{A}^{(0)}$ the local states from Q_i have no in-edges, i.e., for each local a-transition $(\overline{x},\overline{y}) \in \delta_a^{(0)}$ we have $\overline{y} \in \prod_{i\in\mathrm{loc}(a)} S_i \setminus Q_i$ (this is the only use of this assumption in this proof). Hence the existence of some w'-labeled run in $\mathfrak{A}^{(0)}$ to $\overline{s'}$ implies $\overline{s'}\restriction_i \notin Q_i$ for all $i \in \mathrm{loc}(w')$. Consequently, $\mathrm{loc}(w') \cap \mathrm{loc}(a) = \emptyset$ implying $\overline{\pi}(w'a) = \overline{\pi}(aw')$.

Consider the global state

$$\overline{t} = (\overline{s}\restriction_{\mathrm{loc}(a)}, \overline{q'}\restriction_{\mathrm{loc}(w')}, \overline{s'}\restriction_{P\setminus\mathrm{loc}(w'a)}) .$$

- Since $\mathfrak{A}^{(k+1)}$ is asynchronous, $\overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{s}$ implies that the global states $\overline{s'}$ and $\overline{s}$ differ, at most, in the components of $\mathrm{loc}(a)$. Hence

  $$\overline{s} = (\overline{s}\restriction_{\mathrm{loc}(a)}, \overline{s'}\restriction_{\mathrm{loc}(w')}, \overline{s'}\restriction_{P\setminus\mathrm{loc}(w'a)}) .$$

  Since $\mathfrak{A}^{(0)}$ is asynchronous and $\overline{q'} \xrightarrow{w'}_{\mathfrak{A}^{(0)}} \overline{s'}$, this ensures $\overline{t} \xrightarrow{w'}_{\mathfrak{A}^{(0)}} \overline{s}$.

- Since $\mathfrak{A}^{(0)}$ is asynchronous, $\overline{q'} \xrightarrow{w'}_{\mathfrak{A}^{(0)}} \overline{s'}$ implies that the global states $\overline{q'}$ and $\overline{s'}$ differ, at most, in the components of $\mathrm{loc}(w')$. Hence

  $$\overline{q'} = (\overline{s'}\restriction_{\mathrm{loc}(a)}, \overline{q'}\restriction_{\mathrm{loc}(w')}, \overline{s'}\restriction_{P\setminus\mathrm{loc}(w'a)}) .$$

  Since $\mathfrak{A}^{(k+1)}$ is asynchronous and $\overline{s'} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{s}$, this ensures $\overline{q'} \xrightarrow{a}_{\mathfrak{A}^{(k+1)}} \overline{t}$.

  From $\mathrm{Cl}(k+1,1)$, we obtain a global state $\overline{r}$ and a word $w'' \in A^*$ such that

  $$(\overline{q'}, \overline{\pi}(a)) \vdash^* (\overline{r}, \overline{\pi}(w'')) \text{ and } \overline{r} \xrightarrow{w''}_{\mathfrak{A}^{(0)}} \overline{t} .$$

In summary, we have

(a) $(\overline{p}, \overline{\pi}(v)) \vdash^* (\overline{q'}, \overline{\pi}(w'a)) = (\overline{q'}, \overline{\pi}(aw'))$ and $(\overline{q'}, \overline{\pi}(a)) \vdash^* (\overline{r}, \overline{\pi}(w''))$ imply $(\overline{p}, \overline{\pi}(v)) \vdash^* (\overline{r}, \overline{\pi}(w''w'))$.

(b) $\overline{r} \xrightarrow{w''}_{\mathfrak{A}^{(0)}} \overline{t} \xrightarrow{w'}_{\mathfrak{A}^{(0)}} \overline{s}$.

This completes the proof of $\mathrm{Cl}(k+1,n+1)$ from $\mathrm{Cl}(k)$, $\mathrm{Cl}(k+1,1)$ and $\mathrm{Cl}(k+1,n)$.

Therefore, we completed the inductive proof of $\mathrm{Cl}(k+1)$ from $\mathrm{Cl}(k)$. But this means that $\mathrm{Cl}(k)$ holds for all $k \in \mathbb{N}$. □

Lemma 17. *Let $k \in \mathbb{N}$. Then we have $C(\mathfrak{A}^{(k)}) \subseteq \mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$.*

Proof. Now, let $(\overline{p}, \overline{\pi}(v)) \in C(\mathfrak{A}^{(k)})$. Then we have $\overline{p} \xrightarrow{v}_{\mathfrak{A}^{(k)}} \overline{f}$ for some final global state $\overline{f} \in F$. By Lemma 16 there are a global state $\overline{r} \in \mathbf{Q}$ and a word $w \in A^*$ with $(\overline{p}, \overline{\pi}(v)) \vdash^* (\overline{r}, \overline{\pi}(w))$ and $\overline{r} \xrightarrow{w}_{\mathfrak{A}^{(0)}} \overline{f}$ implying $(\overline{r}, \overline{\pi}(w)) \in C(\mathfrak{A}^{(0)})$. This finally implies $(\overline{p}, \overline{\pi}(v)) \in \mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$. □

All in all, from Lemmas 15 and 17 we obtain that $\mathfrak{A}^{(\infty)}$ accepts exactly the set of configurations of $\mathfrak{P}$ that are backwards reachable from $C(\mathfrak{A}^{(0)})$:

Proposition 18. *We have* $C(\mathfrak{A}^{(\infty)}) = \mathrm{pre}^*_{\mathfrak{P}}(C(\mathfrak{A}^{(0)}))$. □

This proves the first claim of Theorem 11, namely that the backwards reachability relation preserves recognizability. It remains to be shown that $\mathfrak{A}^{(\infty)}$ is efficiently constructible. To this aim, note that $\delta^{(0)} \subseteq \delta^{(1)} \subseteq \delta^{(2)} \subseteq \cdots \subseteq \prod_{i\in P} S_i \times A \times \prod_{i\in P} S_i$, i.e., the sequence of transition relations is increasing. Since $\ell := \left|\prod_{i\in P} S_i \times A \times \prod_{i\in P} S_i\right|$ is finite, we have $\delta^{(\ell)} = \delta^{(\ell+1)}$, i.e., $\delta^{(\infty)} = \delta^{(\ell)}$. Similar to the construction from [3] our construction takes time $\mathcal{O}(|\mathfrak{P}|^2 \cdot |\mathfrak{A}^{(0)}|^2 \cdot |A|)$ and results in a $\mathfrak{P}$-AA having the same set of states as $\mathfrak{A}^{(0)}$ (however, the number of transitions increases).

5. Backwards Reachability Does Not Preserve Rationality

Suppose we have a pushdown system (i.e., consider the case $|P| = 1$). Then a set of configurations is rational if, and only if, it is recognizable. Hence, the backwards reachability relation $\mathrm{pre}^*_{\mathfrak{P}}$ also preserves rationality.

Now, recall that there are rational trace languages that are not recognizable (e.g., the language of all traces $\overline{\pi}((ab)^n)$ with $n \in \mathbb{N}$ whenever $a \parallel b$). Then Theorem 11 does not imply that rationality is preserved under the backwards reachability relation. To the contrary, we will now prove that this preservation property does not hold. So, we will show now that in some special cases the set of backwards reachable configurations from a rational trace language is not even decidable (however, in any case $\mathrm{pre}^*_{\mathfrak{P}}(C)$ will be semi-decidable if C is rational).

Proposition 19. *There are a distributed alphabet $\mathcal{D}$, a cPDS $\mathfrak{P}$, and a rational set of configurations C such that $\mathrm{pre}^*_{\mathfrak{P}}(C)$ is not decidable.*

Proof. Consider a Turing-machine $\mathfrak{M}$ with an undecidable word problem. Let Q be the set of states and Σ be the tape alphabet of $\mathfrak{M}$. We construct the distributed alphabet $\mathcal{D} = (A, P, \mathrm{loc})$ as follows:

- $A = \{\$\} \cup (Q \cup \Sigma \cup \{\#\}) \cup (Q' \cup \Sigma' \cup \{\#'\})$ where $Q' = \{q' \mid q \in Q\}$ and $\Sigma' = \{a' \mid a \in \Sigma\}$ are disjoint copies of Q and Σ, respectively, and $\#, \#', \$$ are new symbols,
- $P = \{1, 2\}$, and
- $A_1 = Q \cup \Sigma \cup \{\#, \$\}$ and $A_2 = Q' \cup \Sigma' \cup \{\#', \$\}$ (note that $A_1 \cap A_2 = \{\$\}$).

In the following, for a word $w = a_1 \cdots a_n \in (Q \cup \Sigma \cup \{\#\})^*$, we write $w' = a'_1 \cdots a'_n$ for the copy of w.

Now, we want to construct a cPDS $\mathfrak{P} = (\mathbf{Q}, \Delta)$ writing sequences of configurations of $\mathfrak{M}$ into its stacks. Here, we use the letters $\#$ and $\#'$ as separators between two consecutive configurations and $\$$ for synchronization between the two processes. The states of $\mathfrak{P}$ are the following: $Q_1 = \{q_0, q'_0, q_1, q'_1, q_2, q'_2, q''_2\}$ and

$Q_2 = \{\top\}$. For a better readability we write $\overline{q}$ for the tuple $(q, \top)$ with $q \in Q_1$. Note that in the following $\mathfrak{P}$ will store the configuration sequences backwards due to the usage of the distributed stack. To this end, for $w = a_1 a_2 \cdots a_\ell \in A^*$ we write w^{R} for the word $a_\ell \cdots a_2 a_1$.

The cPDS $\mathfrak{P}$ computes as follows: first it guesses an initial configuration ιw of $\mathfrak{M}$ and writes $(\iota' w' \#')^{\mathrm{R}}$ onto its second stack. This can be done with the following transitions: $(\overline{q_0}, \$, \$\iota', \overline{q_0'}), (\overline{q_0'}, \$, \$a', \overline{q_0'}), (\overline{q_0'}, \$, \$\#', \overline{q_1}) \in \Delta$ where $\iota \in Q$ is the initial state of $\mathfrak{M}$ and $a \in \Sigma$ is any letter from the tape alphabet.

Next, $\mathfrak{P}$ simulates iteratively single computational steps of $\mathfrak{M}$. Let c and d be two configurations of $\mathfrak{M}$ with $c \vdash_{\mathfrak{M}} d$. Then $\mathfrak{P}$ writes $(c\#d'\#')^{\mathrm{R}}$ onto its stacks. We do this with help of the following transitions:

- for each transition of $\mathfrak{M}$ of the form (p, a, q, b, N) we have $(\overline{q_1}, \$, \$apb'q', \overline{q_1'}) \in \Delta$,
- for each transition of $\mathfrak{M}$ of the form (p, a, q, b, L) and each $c \in \Sigma$ we have $(\overline{q_1}, \$, \$apcb'c'q', \overline{q_1'}) \in \Delta$,
- for each transition of $\mathfrak{M}$ of the form (p, a, q, b, R) and each $c \in \Sigma$ we have $(\overline{q_1}, \$, \$capc'q'b', \overline{q_1'}) \in \Delta$,
- for each $a \in \Sigma$ we have $(\overline{q_1}, \$, \$aa', \overline{q_1}), (\overline{q_1'}, \$, \$aa', \overline{q_1'}) \in \Delta$, and
- $(\overline{q_1'}, \$, \$\#\#', \overline{q_1}), (\overline{q_1'}, \$, \$\#\#', \overline{q_2}) \in \Delta$.

Finally, $\mathfrak{P}$ guesses an accepting configuration fw of $\mathfrak{M}$ and pushes $(fw\#)^{\mathrm{R}}$ onto its stacks. To this end, we have the transitions $(\overline{q_2}, \$, \$\#f, \overline{q_2'}) \in \Delta$ for each accepting state f of $\mathfrak{M}$, $(\overline{q_2'}, \$, \$a, \overline{q_2'}) \in \Delta$ for each $a \in \Sigma$, and $(\overline{q_2'}, \$, \#, \overline{q_2''}) \in \Delta$.

Now, let $C = \{\overline{q_2''}\} \times \{aa' \mid a \in Q \cup \Sigma \cup \{\#\}\}^*$. This set of configurations clearly is rational. Then for any $w \in \Sigma^*$ we can see $(\overline{q_0'}, (\iota' w' \#' \$)^{\mathrm{R}}) \in \mathrm{pre}^*_{\mathfrak{P}}(C)$ holds if, and only if, there is a sequence of configurations $c_0, c_1, \ldots, c_k$ of $\mathfrak{M}$ with $(\overline{q_0'}, (\iota' w' \#' \$)^{\mathrm{R}}) \vdash^*_{\mathfrak{P}} (\overline{q_2''}, (c_0 c_0' \#\#' c_1 c_1' \#\#' \cdots c_k c_k' \#\#')^{\mathrm{R}}) \in C$ and $c_0 = \iota w$. But then, by construction of $\mathfrak{P}$, we learn c_0 is initial, $c_{i-1} \vdash_{\mathfrak{M}} c_i$ for each $1 \leq i \leq k$, and c_k is accepting, i.e., $c_0 \vdash_{\mathfrak{M}} c_1 \vdash_{\mathfrak{M}} \cdots \vdash_{\mathfrak{M}} c_k$ is an accepting run of $\mathfrak{M}$. In other words, we have $(\overline{q_0'}, (\iota' w' \#' \$)^{\mathrm{R}}) \in \mathrm{pre}^*_{\mathfrak{P}}(C)$ if, and only if, w is accepted by $\mathfrak{M}$. Since the latter problem is undecidable by assumption, the membership problem of $\mathrm{pre}^*_{\mathfrak{P}}(C)$ also is undecidable. □

6. Summary, Consequences, and Open Questions

We proved that the backwards reachability relation of cooperating multi-pushdown systems efficiently preserves the recognizability of a set of configurations. In addition, we demonstrated that the backwards reachability relation does not preserve rationality (i.e., there is a cPDS and a rational set C of configurations such that $\mathrm{pre}^*(C)$ is not rational anymore).

From the positive result, it follows that the reachability relation is decidable. It implies that it is decidable whether all predecessors of a recognizable set C_1

of configurations are contained in some recognizable set of configurations C_2. In particular, we can decide the control state reachability problem and the EF-model checking problem — although our result allows to bound the running time only non-elementary. However, our result can be understood as the first step towards the verification of cooperating multi-pushdown systems.

As next and obvious open question regarding the verification of cPDS, one would have to consider the recurrent reachability, i.e., the question whether, starting from some configuration, there is an infinite run that visits some global state infinitely often. This could then form the basis for algorithms deciding properties that are given by formulas from linear time temporal logics.

Since we can see cPDS as a natural extension of pushdown systems from word semantics to trace semantics, another open problem is to find some generalized context-free grammars accepting the class of languages of cPDS. Additionally, one could compare this new model with other known models for multi-pushdown systems.

Acknowledgment

We would like to thank all reviewers from FCT 2023 as well as from JCSS for their valuable feedback.

References

[1] W. Zielonka, Notes on finite asynchronous automata, RAIRO - Theoretical Informatics and Applications 21 (2) (1987) 99–135. doi:10.1051/ita/1987210200991.

[2] S. C. Kleene, Representation of events in nerve nets and finite automata, in: Automata Studies, Vol. 34 of Annals of Mathematics Studies, Princeton University Press, 1956, pp. 3–40.

[3] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: Application to model-checking, in: A. Mazurkiewicz, J. Winkowski (Eds.), 8th International Conference on Concurrency Theory, Vol. 1243 of Lecture Notes in Computer Science, Springer, 1997, pp. 135–150. doi:10.1007/3-540-63141-0_10.

[4] J. Esparza, D. Hansel, P. Rossmanith, S. Schwoon, Efficient Algorithms for Model Checking Pushdown Systems, in: E. A. Emerson, A. P. Sistla (Eds.), Computer Aided Verification, Vol. 1855 of Lecture Notes in Computer Science, Springer, 2000, pp. 232–247. doi:10.1007/10722167_20.

[5] A. Finkel, B. Willems, P. Wolper, A Direct Symbolic Approach to Model Checking Pushdown Systems, Electronic Notes in Theoretical Computer Science 9 (1997) 27–37. doi:10.1016/S1571-0661(05)80426-8.

[6] T. Schellmann, Model-Checking von Kellersystemen, Bachelor's Thesis, Technische Universität Ilmenau, Ilmenau (2019).

[7] A. Bouajjani, J. Esparza, T. Touili, A generic approach to the static analysis of concurrent programs with procedures, ACM SIGPLAN Notices 38 (1) (2003) 62–73. doi:10.1145/640128.604137.

[8] A. Bouajjani, M. Müller-Olm, T. Touili, Regular Symbolic Analysis of Dynamic Networks of Pushdown Systems, in: M. Abadi, L. de Alfaro (Eds.), CONCUR 2005 – Concurrency Theory, Vol. 3653 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 473–487. doi:10.1007/11539452_36.

[9] S. Qadeer, J. Rehof, Context-Bounded Model Checking of Concurrent Software, in: N. Halbwachs, L. D. Zuck (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 93–107. doi:10.1007/978-3-540-31980-1_7.

[10] S. La Torre, P. Madhusudan, G. Parlato, A Robust Class of Context-Sensitive Languages, in: 22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007), 2007, pp. 161–170. doi:10.1109/LICS.2007.9.

[11] A. Heußner, J. Leroux, A. Muscholl, G. Sutre, Reachability Analysis of Communicating Pushdown Systems, Logical Methods in Computer Science Volume 8, Issue 3 (2012). doi:10.2168/LMCS-8(3:23)2012.

[12] D. Babić, Z. Rakamarić, Asynchronously Communicating Visibly Pushdown Systems, in: D. Beyer, M. Boreale (Eds.), Formal Techniques for Distributed Systems, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2013, pp. 225–241. doi:10.1007/978-3-642-38592-6_16.

[13] C. Aiswarya, P. Gastin, K. Narayan Kumar, Verifying Communicating Multi-pushdown Systems via Split-Width, in: F. Cassez, J.-F. Raskin (Eds.), Automated Technology for Verification and Analysis, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2014, pp. 1–17. doi:10.1007/978-3-319-11936-6_1.

[14] C. Aiswarya, P. Gastin, K. Narayan Kumar, Controllers for the Verification of Communicating Multi-pushdown Systems, in: P. Baldan, D. Gorla (Eds.), CONCUR 2014 – Concurrency Theory, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2014, pp. 297–311. doi:10.1007/978-3-662-44584-6_21.

[15] M. F. Atig, B. Bollig, P. Habermehl, Emptiness of Ordered Multi-Pushdown Automata is 2ETIME-Complete, International Journal of Foundations of Computer Science 28 (08) (2017) 945–975. doi:10.1142/S0129054117500332.

[16] B. Bollig, D. Kuske, R. Mennicke, The Complexity of Model Checking Multi-Stack Systems, Theory of Computing Systems 60 (4) (2017) 695–736. doi:10.1007/s00224-016-9700-6.

[17] S. La Torre, M. Napoli, G. Parlato, Reachability of scope-bounded multi-stack pushdown systems, Information and Computation 275 (2020) 104588. doi:10.1016/j.ic.2020.104588.

[18] M. Hutagalung, N. Hundeshagen, D. Kuske, M. Lange, É. Lozes, Multi-buffer simulations: Decidability and complexity, Information and Computation 262 (2018) 280–310. doi:10.1016/j.ic.2018.09.008.

[19] C. Köcher, D. Kuske, Forwards- and Backwards-Reachability for Cooperating Multi-Pushdown Systems, in: FCT 2023, Vol. 14292 of Lecture Notes in Computer Science, Springer, 2023, pp. 318–332. doi:10.1007/978-3-031-43587-4_23.

[20] V. Diekert, G. Rozenberg, The Book of Traces, World scientific, 1995. doi:10.1142/9789814261456.

[21] P. Cartier, D. Foata, Problèmes Combinatoires de Commutation et Réarrangements, Vol. 85 of Lecture Notes in Mathematics, Springer, Berlin, Heidelberg, 1969. doi:10.1007/BFb0079468.

[22] A. Mazurkiewicz, Concurrent program schemes and their interpretations, DAIMI Report Series 6 (78) (1977).

[23] J. Sakarovitch, The "last" decision problem for rational trace languages, in: Latin American Symposium on Theoretical Informatics, Vol. 583, Springer, 1992, pp. 460–473.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: NeurIPS, 2017, pp. 5998–6008.

[25] Y. Hao, D. Angluin, R. Frank, Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity, Transactions of the Association for Computational Linguistics 10 (2022) 800–810. doi:10.1162/tacl_a_00490.

[26] P. Bergsträßer, C. Köcher, A. W. Lin, G. Zetzsche, The Power of Hard Attention Transformers on Data Sequences: A Formal Language Theoretic Perspective, 2024. doi:10.48550/arXiv.2405.16166.