Trimming Visibly Pushdown Automata

Mathieu Caralp, Pierre-Alain Reynier, Jean-Marc Talbot

Laboratoire d'Informatique Fondamentale de Marseille
UMR 7279, Aix-Marseille Université & CNRS, France

Abstract

We study the problem of trimming visibly pushdown automata (VPA). We first describe a polynomial time procedure which, given a visibly pushdown automaton that accepts only well-nested words, returns an equivalent visibly pushdown automaton that is trimmed. We then show how this procedure can be lifted to the setting of arbitrary VPA. Furthermore, we present a way of building, given a VPA, an equivalent VPA which is both deterministic and trimmed. Last, our trimming procedures can be applied to weighted VPA.

Keywords: Visibly Pushdown Automata, Trimming

1. Introduction

Visibly pushdown automata (VPA) are a particular class of pushdown automata defined over an alphabet split into call, internal and return symbols [2, 3] [1]. In VPA, the stack behavior is driven by the input word: when reading a call symbol, a symbol is pushed onto the stack, for a return symbol, the top symbol of the stack is popped, and for an internal symbol, the stack remains unchanged. VPA have been applied in research areas such as software verification (VPA allow one to model function calls and returns, thus avoiding the study of data flows along invalid paths) and XML documents processing (VPA can be used to model properties over words satisfying a matching property between opening and closing tags).

Languages defined by visibly pushdown automata enjoy many properties of regular languages such as (effective) closure by Boolean operations and these languages can always be defined by a deterministic visibly pushdown automaton. However, VPA do not have a unique minimal form [1]. Instead of minimization, one may consider trimming as a way to deal with smaller automata. Trimming a finite state automaton amounts to removing useless states, *i.e.* states that do not occur in some accepting computation of the automaton: every state of the automaton should be both reachable from an initial state, and co-reachable from a final state. This property is important from both a practical and a theoretical point of view. Indeed, most of the algorithmic operations performed on an automaton will only be relevant on the trimmed part of this automaton. Removing useless states may thus avoid the study of irrelevant paths in the automaton, and speed up the analysis. From a theoretical aspect, there are several results holding for automata provided they are trimmed. For instance, the boundedness of finite-state automata with multiplicities can be characterized by means of simple patterns for trimmed automata (see [14, 9]). Similarly, Choffrut introduced in [8] the twinning property to characterize sequentiality of (trimmed) finite-state transducers. This result was later extended to weighted finite-state automata in [6]. Both of these results have been extended to visibly pushdown automata and transducers in [7] and [10] respectively, requiring these objects to be trimmed.

While trimming finite state automata can be done easily in linear time by solving two reachability problems in the graph representing the automaton, the problem is much more involved for VPA (and for pushdown automata in general). Indeed, in this setting, the current state of a computation (called a configuration) is given by both a "control" state and a stack content. A procedure has been presented in [12] for pushdown automata. It consists in computing, for

Email addresses: `mathieu.caralp@lif.univ-mrs.fr` (Mathieu Caralp), `pierre-alain.reynier@lif.univ-mrs.fr` (Pierre-Alain Reynier), `jean-marc.talbot@lif.univ-mrs.fr` (Jean-Marc Talbot)

[1] These automata were first introduced in [5] as "input-driven automata".

each state, the regular language of stack contents that are both reachable and co-reachable, and using this information to constrain the behaviors of the pushdown automaton in order to trim it. This approach has however an exponential time complexity.

Contributions. In this work, we present a procedure for trimming visibly pushdown automata. The running time of this procedure is bounded by a polynomial in the size of the input VPA. We first tackle the case of VPA recognizing only so-called *well-nested* words, *i.e.* words which have no unmatched call or return symbols. This class of VPA is called well-nested VPA, and denoted by wnVPA. We actually present a construction for reducing wnVPA, *i.e* ensuring that every run starting from an initial configuration can be completed into an accepting run. Symmetrically, we define a construction for co-reduction acting in a dual fashion. Combination of both constructions yields a trimming procedure. In a second step, we address the general case. To do so, we present a construction which modifies a VPA in order to obtain a wnVPA. This construction has to be reversible, in order to recover the original language, and to be compatible with the trimming procedure. In addition, we also design this construction in such a way that it allows to prove the following result: given a VPA, we can effectively build an equivalent VPA which is both deterministic and trimmed. Moreover, when considering deterministic inputs, we prove that there is no trimming procedure running in polynomial time while preserving determinism. Finally, we show that our constructions can be applied to weighted VPA.

Organization of the paper. In Section 2 we introduce definitions. We address the case of well-nested VPA in Section 3 and the general case in Section 4. We consider the issue of determinization in Section 5 and the extension to weighted VPA in Section 6. Last, a summary of our results is presented in Section 7.

Related models. VPA are tightly connected to several models:

Context-free grammars: it is well-known that pushdown automata are equivalent to context-free grammars. This observation yields the following procedure for trimming pushdown automata [2]. One can first translate the automaton into an equivalent context-free grammar, then eliminate from this grammar variables generating the empty language or not reachable from the start symbol, and third convert the resulting grammar into the pushdown automaton performing its top-down analysis. This construction has a polynomial time complexity but, in this form, it does not apply to VPA. Indeed, the resulting pushdown automaton may not satisfy the condition of visibility as the third step may not always produce rules respecting the constraints on push and pop operations associated with call and return symbols.

Tree automata: by the standard interpretation of XML documents as unranked trees, VPA can be understood as acceptors of unranked tree languages. It is shown in [2] that they actually do recognize precisely the set of regular (ranked) tree languages, using the encoding of so-called *stack-trees*, which is similar to the first-child next-sibling encoding. Trimming ranked tree automata is standard (and can be performed in linear time), and one can wonder whether this approach could yield a polynomial time trimming procedure for VPA. Actually, going through tree automata would not ease the construction of a trimmed VPA. Indeed, trimming the first-child next-sibling encoding of a wnVPA, and then translating back the result into a wnVPA yields an automaton which is reduced but not trimmed (this is intuitively due to the fact that the first-child next-sibling encoding realizes a left-to-right traversal of the tree). Moreover, this construction does not ensure a bijection between accepting runs, a property that is useful when moving to weighted VPA.

Nested word automata [3]: this model is equivalent to that of VPA. One could thus rephrase our constructions in this context, and obtain the same results.

2. Definitions

2.1. Preliminaries

Words and well-nested words. A *structured alphabet* Σ is a finite set partitioned into three disjoint sets Σ_c, Σ_r and Σ_ι, denoting respectively the *call*, *return* and *internal* alphabets. We denote by Σ^* the set of words over Σ and by ε the empty word.

The set of *well-nested* words Σ^*_{wn} is the smallest subset of Σ^* such that $\varepsilon \in \Sigma^*_{\mathsf{wn}}$ and for all $a \in \Sigma_\iota$, $c \in \Sigma_c$, $r \in \Sigma_r$ and all $u, v \in \Sigma^*_{\mathsf{wn}}$, $au \in \Sigma^*_{\mathsf{wn}}$ and $curv \in \Sigma^*_{\mathsf{wn}}$.

[2]We thank Géraud Sénizergues for pointing us this construction.

Given a family of elements $e_1, e_2, \ldots, e_n$, we denote by $\Pi_{i=1}^n e_i$ the concatenation $e_1 e_2 \ldots e_n$. The length of a word u is denoted by $|u|$. For a tuple $e = (f_1, \ldots, f_l)$, we define for all $i \in \{1, \ldots, l\}$ the projection $\pi_i(e)$ as f_i. We extend this notion of projection to sequences as $\pi_i(\prod_{k=1}^n e_k) = \prod_{k=1}^n \pi_i(e_k)$

*Visibly pushdown automata (*VPA*).* Visibly pushdown automata are a restriction of pushdown automata in which the stack behavior is imposed by the input word. On a call symbol, the VPA pushes a symbol onto the stack, on a return symbol, it must pop the top symbol of the stack, and on an internal symbol, the stack remains unchanged. The only exception is that some return symbols may operate on the empty stack.

Definition 2.1 (Visibly pushdown automata). A *visibly pushdown automaton* (VPA) on finite words over Σ is a tuple $A = (Q, I, F, \Gamma, \delta)$ where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, Γ is a finite stack alphabet and $\delta = \delta_c \cup \delta_r \cup \delta_r^\perp \cup \delta_\iota$ is the (finite) transition relation with δ_c, δ_r, $\delta_r^\perp$ and δ_ι four disjoint sets, with $\delta_c \subseteq Q \times \Sigma_c \times \Gamma \times Q$, $\delta_r \subseteq Q \times \Sigma_r \times \Gamma \times Q$, $\delta_r^\perp \subseteq Q \times \Sigma_r \times \{\perp\} \times Q$, and $\delta_\iota \subseteq Q \times \Sigma_\iota \times Q$.

For a transition $t = (q, \alpha, \gamma, q')$ from δ_c, δ_r or $\delta_r^\perp$ or $t = (q, \alpha, q')$ in δ_ι, we denote by $\mathsf{source}(t)$ and $\mathsf{target}(t)$ the states q and q' respectively, and by $\mathsf{letter}(t)$ the symbol α.

A *stack* is a word from Γ^* and we denote by $\perp$ the empty word on Γ. A *configuration* κ of a VPA is a pair $(q, \sigma) \in Q \times \Gamma^*$.

Definition 2.2. *A run of A on a word $w = \alpha_1 \ldots \alpha_l \in \Sigma^*$ over a finite (and possibly empty) sequence of transitions $(t_k)_{1 \leqslant k \leqslant l}$ from a configuration (q, σ) to a configuration (q', σ') is a finite sequence of symbols and configurations $\rho = (q_0, \sigma_0)\Pi_{k=1}^l(\alpha_k(q_k, \sigma_k))$ such that $(q, \sigma) = (q_0, \sigma_0)$, $(q', \sigma') = (q_l, \sigma_l)$, and, for each $1 \leqslant k \leqslant l$, there exists $\gamma_k \in \Gamma$ such that either:*

- $t_k = (q_{k-1}, \alpha_k, \gamma_k, q_k) \in \delta_c$ *and* $\sigma_k = \sigma_{k-1}\gamma_k$, *or*
- $t_k = (q_{k-1}, \alpha_k, \gamma_k, q_k) \in \delta_r$ *and* $\sigma_{k-1} = \sigma_k\gamma_k$, *or*
- $t_k = (q_{k-1}, \alpha_k, \perp, q_k) \in \delta_r^\perp$ *and* $\sigma_{k-1} = \sigma_k = \perp$, *or*
- $t_k = (q_{k-1}, \alpha_k, q_k) \in \delta_\iota$ *and* $\sigma_k = \sigma_{k-1}$.

We denote by $\mathsf{Run}_w(A)$ the set of runs of A over the word w and by $\mathsf{Run}(A)$ the set of runs of A. Note that a run for the empty word is simply any configuration. We write $\rho : (q, \sigma) \xrightarrow{w} (q', \sigma')$ when there exists a run ρ over w from (q, σ) to (q', σ'). We may omit the superscript w when irrelevant. For this run ρ, we denote by $\mathsf{first}(\rho)$ the configuration (q, σ) and by $\mathsf{last}(\rho)$ the configuration (q', σ'). Given two runs $\rho_i = \kappa_0^i \Pi_{k=1}^{\ell_i}(\alpha_k^i(\kappa_k^i))$ for $i \in \{1, 2\}$, we define the concatenation $\rho_1\rho_2$ of these runs, provided that $\mathsf{last}(\rho_1) = \mathsf{first}(\rho_2)$, as $\rho_1\rho_2 = \kappa_0^1 \Pi_{k=1}^{\ell_1}(\alpha_k^1 \kappa_k^1) \Pi_{k=1}^{\ell_2}(\alpha_k^2 \kappa_k^2)$.

Initial and final configurations are configurations of the form $(q, \perp)$ with $q \in I$ and (q, σ) with $q \in F$ respectively. A run is *initialized* if it starts in an initial configuration and it is *accepting* if it is initialized and ends in a final configuration. We denote by $\mathsf{ARun}_w(A)$ the set of accepting runs of A over the word w. A word is accepted by A iff there exists an accepting run of A on it. The language of A, denoted by $L(A)$, is the set of words accepted by A.

A configuration κ is *reachable from a configuration* κ' if there exists a word u in Σ^* such that $\kappa' \xrightarrow{u} \kappa$; in this case we say also that κ' is *co-reachable from* κ. We say that a configuration κ' is *reachable* if there exists an initial configuration κ_i such that κ' is reachable from κ_i and *co-reachable* if there exists an final configuration κ_f such that κ' is co-reachable from κ_f.

Definition 2.3. *A* VPA $A = (Q, I, F, \Gamma, \delta)$ *is* deterministic *if I is a singleton and the following conditions hold:*

- *if* $(p, c, \gamma, q), (p, c, \gamma', q') \in \delta_c$ *then* $\gamma = \gamma'$ *and* $q = q'$,
- *if* $(p, a, q), (p, a, q') \in \delta_\iota$ *or* $(p, r, \perp, q), (p, r, \perp, q') \in \delta_r^\perp$ *or* $(p, r, \gamma, q), (p, r, \gamma, q') \in \delta_r t$ *then* $q = q'$.

A VPA A is said to be *well-nested* if $L(A) \subseteq \Sigma_{\mathsf{wn}}^*$. The class of well-nested VPA is denoted by wnVPA.

Remark 2.1. *If A is well-nested then its final configurations (f, σ) (with f a final state) are reachable only if $\sigma = \perp$.*

2.2. (Co)-reduced and trimmed VPA

Definition 2.4. *Let A be a* VPA. *Let us consider the three following conditions :*

(1) every reachable configuration is co-reachable.

(2) every configuration co-reachable from some reachable and final configuration is reachable.

(3) for every state q, there exists an accepting run going through a configuration (q, σ).

We say that the automaton A is reduced *if it fulfills conditions* (1) *and* (3)*, and* co-reduced *if it fulfills conditions* (2) *and* (3)*. It is* trimmed *if it is both reduced and co-reduced. It is* weakly reduced *if it fulfills condition* (1) *and* weakly co-reduced *if it fulfills condition* (2).

Observe in Figure 1 that condition (2) looks more complicated than the property stating that every co-reachable configuration is reachable. Indeed, unlike in finite state-automata, the presence of a stack requires one to focus on reachable final configurations, and not to consider arbitrary final configuration. However, we will see that for wnVPA, this condition is equivalent to the simpler one.

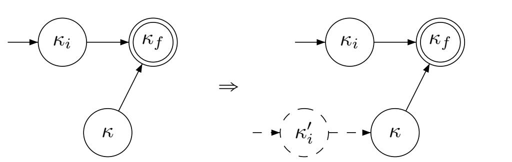

Figure 1: If κ is co-reachable from a final configuration κ_f, κ_f being reachable from an intial configuration κ_i, then κ is reachable from an initial configuration κ'_i.

Observe that condition (3) simply corresponds to the removal of states that are useless, which can easily be done in polynomial time:

Proposition 2.1. *([4]) Let $A = (Q, I, F, \Gamma, \delta)$ be a* VPA. *For any state $q \in Q$, one can build in polynomial time from A two finite state automata B_q and C_q over the alphabet Γ such that $L(B_q) = \{\sigma \in \Gamma^* \mid (q, \sigma)$ is reachable in $A\}$ and $L(C_q) = \{\sigma \in \Gamma^* \mid (q, \sigma)$ is co-reachable in $A\}$.*

The property of being co-reduced can be rephrased when considering wnVPA:

Proposition 2.2. *Let $A = (Q, I, F, \Gamma, \delta)$ be a* wnVPA *such that for all state $f \in F$ there exists an accepting run leading to the configuration $(f, \perp)$. Then A is* weakly co-reduced *if and only if every configuration co-reachable from some configuration $(f, \perp)$ with $f \in F$ is reachable.*

PROOF. By Remark 2.1 and the assumption on A in the proposition, the set of reachable final configurations of A is $\{(f, \perp) \mid f \in F\}$. $\square$

3. Trimming well-nested VPA

In this section, given a wnVPA A, we present the construction of a wnVPA $\mathsf{trim}_{\mathsf{wn}}(A)$, which recognizes the same language, and in addition is trimmed. First we define the reduced wnVPA $\mathsf{reduce}(A)$ which is equivalent to A. Next we will present $\mathsf{coreduce}(A)$ and lastly we will combine these two procedures in order to produce $\mathsf{trim}_{\mathsf{wn}}(A)$.

3.1. Construction of $\mathsf{wreduce}(A)$ *and* $\mathsf{reduce}(A)$

For a wnVPA A, we describe the construction of the weakly reduced VPA $\mathsf{wreduce}(A)$. The VPA $\mathsf{reduce}(A)$ is simply obtained by removing useless states of $\mathsf{wreduce}(A)$ using Proposition 2.1. In fact, if $L(B_q) \cap L(C_q) = \emptyset$ then q can be removed because there is no accepting run going through a configuration (q, σ) in A.

Consider a wnVPA $A = (Q, I, F, \Gamma, \delta)$. We define WN as the set $\{(p, q) \in Q \times Q \mid \exists (p, \perp) \rightarrow (q, \perp) \in \mathsf{Run}(A)\}$. This set can be computed in quadratic time as the least one satisfying

- $\{(p, p) \mid p \in Q\} \subseteq \mathsf{WN}$,
- if $(p, p') \in \mathsf{WN}$ and $(p', p'') \in \mathsf{WN}$, then $(p, p'') \in \mathsf{WN}$
- if $(p, q) \in \mathsf{WN}$, and $\exists (q, i, q') \in \delta_\iota$, then $(p, q') \in \mathsf{WN}$

- if $(p,q) \in \mathsf{WN}$ and $\exists (p', c, \gamma, p) \in \delta_c, (q, r, \gamma, q') \in \delta_r$, then $(p', q') \in \mathsf{WN}$

Definition 3.1. *Given a* wnVPA $A = (Q, I, F, \Gamma, \delta)$*, we define the* wnVPA $\mathsf{wreduce}(A) = (Q', I', F', \Gamma', \delta')$ *with* $Q' = \mathsf{WN}$*,* $I' = \mathsf{WN} \cap (I \times F)$*,* $F' = \{(f,f) \mid f \in F\}$*,* $\Gamma' = \Gamma \times Q$*, and* δ' *defined by its restrictions on call, return* [3] *and internal symbols respectively (namely* δ'_c*,* δ'_r *and* δ'_ι*):*

- $\delta'_c = \left\{ \quad ((p,q), c, (\gamma, q), (p', q')) \quad \middle| \quad \begin{array}{l} (p,q), (p',q') \in Q', (p, c, \gamma, p') \in \delta_c \\ \exists r \in \Sigma_r, \exists s \in Q, (q', r, \gamma, s) \in \delta_r \text{ and } (s,q) \in Q' \end{array} \right\}$
- $\delta'_r = \{\ ((q',q'), r, (\gamma, q), (p,q)) \mid (p,q) \in Q', (q', r, \gamma, p) \in \delta_r\ \}$
- $\delta'_\iota = \{\ ((p,q), a, (p',q)) \mid (p,q), (p',q) \in Q', (p, a, p') \in \delta_\iota\ \}$

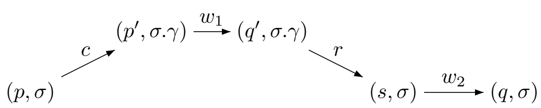

Figure 2: Construction of call transitions.

Intuitively, the states (and the stack) of $\mathsf{wreduce}(A)$ extend those of A with an additional state of A. This extra component is used by $\mathsf{wreduce}(A)$, when simulating a run of the VPA A, to store the target state that the run should reach to pop the symbol on top of the stack. The source states of return transitions are of the form (q,q) to ensure that the target state is reached when popping the top of the stack. To obtain a weakly reduced VPA, we require for the call transitions the existence of a matching return transition that allows one to reach the target state. This condition is depicted on Figure 2, and we give an example of the construction in Figure 3.

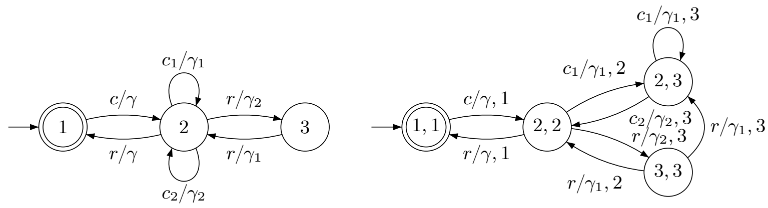

Figure 3: On the left a VPA A, on the right $\mathsf{reduce}(A)$. There exists an initialized run of A over cc_2 which cannot be completed into an accepting run. This run is no longer present in $\mathsf{reduce}(A)$.

For the rest of this subsection, we assume fixed some wnVPA $A = (Q, I, F, \Gamma, \delta)$ and denote A_{wred} the wnVPA $\mathsf{wreduce}(A)$.

The VPA A_{wred} satisfies that :

Lemma 3.1. *For all* $w \in \Sigma^*_{\mathsf{wn}}$ *and all* $\rho' : ((p,q), \perp) \xrightarrow{w} ((p',q'), \perp) \in \mathsf{Run}(A_{\mathsf{wred}})$*, we have* $q = q'$*.*

PROOF. By induction on the structure of w; we show the property for $w = \varepsilon$ for the base case. The induction step is done over $w = cw_1rw_2$ and $w = aw_1$ with $w_1, w_2 \in \Sigma^*_{\mathsf{wn}}$, $c \in \Sigma_c$, $r \in \Sigma_r$ and $a \in \Sigma_\iota$ assuming the property on w_1, w_2. □

We now relate accepting runs of A and of A_{wred}: we define a mapping Φ_{wred} from $\mathsf{Run}(A_{\mathsf{wred}})$ to $\mathsf{Run}(A)$. For any run $\rho' = ((p_0, q_0), \sigma_0)\Pi_{i=1}^k(\alpha_i((p_i, q_i), \sigma_i))$ of A_{wred}, we let $\Phi_{\mathsf{wred}}(\rho')$ be the run $(p_0, \pi_1(\sigma_0))\Pi_{i=1}^k(\alpha_i(p_i, \pi_1(\sigma_i)))$.

Lemma 3.2. *For all* $w \in \Sigma^*$ *and all runs* $\rho \in \mathsf{Run}_w(A_{\mathsf{wred}})$*, we have* $\Phi_{\mathsf{wred}}(\rho) \in \mathsf{Run}_w(A)$*.*

PROOF. By induction on the structure of w. □

[3] As the language is well-nested, we do not consider return transitions on the empty stack.

The next two lemmas prove the injectivity and surjectivity of Φ_{wred}.

Lemma 3.3. *For all $w \in \Sigma^*_{\mathsf{wn}}$, if there exists a run $\rho : (p, \bot) \xrightarrow{w} (q, \bot)$ in $\mathsf{Run}(A)$, then there exists a run ρ' : $((p, q), \bot) \xrightarrow{w} ((q, q), \bot) \in \mathsf{Run}(A_{\mathsf{wred}})$ such that $\Phi_{\mathsf{wred}}(\rho') = \rho$.*

PROOF. By induction on the structure of w using Lemma 3.2. □

Lemma 3.4. *For all $w \in \Sigma^*_{\mathsf{wn}}$ and all $\rho'_1, \rho'_2 \in \mathsf{Run}_w(A_{\mathsf{wred}})$, if $\Phi_{\mathsf{wred}}(\rho'_1) = \Phi_{\mathsf{wred}}(\rho'_2)$ and $\mathsf{last}(\rho'_1) = \mathsf{last}(\rho'_2)$, then $\rho'_1 = \rho'_2$.*

PROOF. By induction on the structure of w using Lemma 3.2 and Lemma 3.1. □

Theorem 3.1. *Let A be a wnVPA, and let $A_{\mathsf{wred}} = \mathsf{wreduce}(A)$ and $A_{\mathsf{red}} = \mathsf{reduce}(A)$. A_{wred} and A_{red} can be built in polynomial time, and satisfy:*

(1) *for all $w \in \Sigma^*_{\mathsf{wn}}$, there exist bijections between $\mathsf{ARun}_w(A_{\mathsf{wred}})$ and $\mathsf{ARun}_w(A)$, and $\mathsf{ARun}_w(A_{\mathsf{red}})$ and $\mathsf{ARun}_w(A)$ and thus in particular $L(A) = L(A_{\mathsf{wred}}) = L(A_{\mathsf{red}})$,*

(2) *A_{wred} is weakly reduced, and A_{red} is reduced.*

PROOF. We prove that when restricted to $\mathsf{ARun}_w(A_{\mathsf{wred}})$, the mapping Φ_{wred} is bijective.

- Φ_{wred} is surjective: it is easy to verify that applying Lemma 3.3 on an accepting run of A yields an accepting run of A_{wred}, which is in its inverse image by the mapping Φ_{wred}.

- Φ_{wred} is injective: this is an immediate corollary of Lemma 3.4 as last states of accepting runs of A_{wred} are of the form (f, f). Thus, if two accepting runs ρ'_1, ρ'_2 have the same image by Φ_{wred}, they must verify $\mathsf{last}(\rho'_1) = \mathsf{last}(\rho'_2)$.

We now prove that $\mathsf{wreduce}(A)$ is weakly reduced that is, that any reachable configuration of A_{wred} is also co-reachable.

Let $\kappa = ((p, q), \sigma)$ be a reachable configuration of A_{wred}. There exists a run ρ of A_{wred} of the form $((i, f), \bot) \to \kappa$ starting in an initial configuration $((i, f), \bot)$. We show by induction on the size of the stack σ that we can reach a final configuration from κ.

If $|\sigma| = 0$, by Lemma 3.1, we obtain $q = f$. In particular, this implies $q \in F$. Since $(p, q) \in \mathsf{WN}$, there is a run $(p, \bot) \to (q, \bot)$ in A, thus by Lemma 3.3 we have a run $((p, q), \bot) \to ((q, q), \bot)$ of A_{wred}. This concludes this case as by the above observation ($q \in F$), we have $(q, q) \in F'$.

We now assume for the induction that the property holds when $|\sigma| \leqslant n$ and we consider a stack σ such that $|\sigma| = n + 1$. Let denote by (γ, q') the top symbol of σ, write $\sigma = \sigma'.(\gamma, q')$, and consider the first position in the run ρ that pushes this symbol onto the stack. We denote by c the associated call. More precisely, there exists a unique decomposition of ρ as $\rho : ((i, f), \bot) \to ((p', q'), \sigma') \xrightarrow{c} ((p'', q''), \sigma) \to ((p, q), \sigma)$ such that the run from $((p'', q''), \sigma)$ to $((p, q), \sigma)$ is associated with a well-nested word. By Lemma 3.1, we obtain $q'' = q$. Considering the call transition associated with c, and by definition of δ'_c, there exists a return transition $((q, q), r, (\gamma, q'), (s, q')) \in \delta'_r$ for some letter $r \in \Sigma_r$. In addition, as $(p, q) \in \mathsf{WN}$, there is a run $(p, \bot) \to (q, \bot)$ in A, and thus by Lemma 3.3 we have a run $((p, q), \bot) \to ((q, q), \bot)$ in A_{wred}, and thus a run $((p, q), \sigma) \to ((q, q), \sigma)$ in A_{wred}.

As the top symbol of σ is (γ, q'), the above return transition can be used to reach configuration $((s, q'), \sigma')$ whose height is n. The result follows by induction hypothesis. □

An important feature of the construction reduce is that it preserves the co-reduction:

Proposition 3.1. *Let A be a wnVPA. If A is co-reduced, then $\mathsf{reduce}(A)$ is co-reduced as well.*

PROOF. We will prove that if A is co-reduced, then $A_{\mathsf{wred}} = \mathsf{wreduce}(A)$ is co-reduced as well. Since $\mathsf{reduce}(A)$ is obtained by removing all the useless states of A_{wred}, we can conclude that $\mathsf{reduce}(A)$ is also co-reduced.

Let $\kappa = ((p, q), \sigma)$ and $\kappa_i = ((p_i, q_i), \bot)$ be two configurations of A_{wred} such that (p_i, q_i) is an initial state of A_{wred} and there exists in A_{wred} two runs $\rho : \kappa \to ((f, f), \sigma')$ and $\kappa_i \to ((f, f), \sigma')$ where f is a final state of A. As

A_{wred} is a wnVPA and by Remark 2.1, we have $\sigma' = \perp$. We show by induction on the size of the stack σ that κ can be reached from an initial configuration.

If $|\sigma| = 0$, by Lemma 3.1, $q = f$. By construction of the set WN, $(p, \perp) \to (f, \perp)$ is a run of A. Since A is co-reduced, there exists a run $(i, \perp) \to (p, \perp)$ with $i \in I$. Thus by Lemmas 3.3 and 3.1 $((i, f), \perp) \to ((p, f), \perp) \to ((f, f), \perp)$ is a run of A_{wred}.

We now assume for the induction that the property holds when $|\sigma| \leqslant n$ and we consider a stack σ such that $|\sigma| = n + 1$. Let us consider the first position in the run ρ that pops the top symbol from the stack. We denote by r the associated return. More precisely, there exists a unique decomposition of ρ as follows:

$$((p, q), \sigma) \xrightarrow{w} ((q'', q''), \sigma) \xrightarrow{r} ((p', q'), \sigma') \xrightarrow{w'} ((f, f), \perp) \qquad \text{with } w \in \Sigma^*_{\mathsf{wn}} \text{ and } \sigma = \sigma'(\gamma, q') \text{ for } \gamma \in \Gamma$$

By Lemma 3.1, $q'' = q$. By Lemma 3.4, the projection of ρ is a run in A:

$$(p, \bar{\sigma}) \xrightarrow{w} (q, \bar{\sigma}) \xrightarrow{r} (p', \bar{\sigma}') \xrightarrow{w'} (f, \perp) \in \mathsf{Run}(A) \qquad \text{where } \bar{\sigma} = \pi_1(\sigma) \text{ and } \bar{\sigma}' = \pi_1(\sigma')$$

Since A is co-reduced, the configuration $(p, \bar{\sigma})$ is reachable from an initial configuration of A by some run ρ_1. We decompose ρ_1 so as to exhibit the call transition that pushed the top symbol of $\bar{\sigma}$:

$$\rho_1 : (i, \perp) \to (p'', \bar{\sigma}') \xrightarrow{c,\gamma} (s, \bar{\sigma}) \xrightarrow{w''} (p, \bar{\sigma}) \in \mathsf{Run}(A) \qquad \text{where } c \in \Sigma_c \text{ and } w'' \in \Sigma^*_{\mathsf{wn}}$$

By Lemmas 3.3 and 3.1, we deduce:

$$((s, q), \perp) \xrightarrow{w''} ((p, q), \perp) \xrightarrow{w} ((q, q), \perp) \in \mathsf{Run}(A_{\mathsf{wred}})$$

and thus

$$((s, q), \sigma) \xrightarrow{w''} ((p, q), \sigma) \xrightarrow{w} ((q, q), \sigma) \in \mathsf{Run}(A_{\mathsf{wred}})$$

Consider now the call transition on c used in the run ρ_1. It is of the form $t = (p'', c, \gamma, s)$. By the existence of the above run in A_{wred}, we can deduce the existence of the transition $((p'', q'), c, (\gamma, q'), (s, q))$ in δ'_c. We can thus extend the above run of A_{wred} as follows:

$$((p'', q'), \sigma') \xrightarrow{c} ((s, q), \sigma) \xrightarrow{w''} ((p, q), \sigma) \xrightarrow{w} ((q, q), \sigma) \xrightarrow{rw'} ((f, f), \perp) \in \mathsf{Run}(A_{\mathsf{wred}})$$

and the result follows by induction hypothesis. $\square$

3.2. *Construction of* $\mathsf{coreduce}(A)$ *and* $\mathsf{wcoreduce}(A)$

For a wnVPA A, we now describe the construction of the VPA $\mathsf{wcoreduce}(A)$, which is weakly co-reduced. This construction can be seen as the application of wreduce on the dual of A which is obtained by swapping call and return symbols and transitions, swapping initial and final states and swapping target and source of transitions. The VPA $\mathsf{coreduce}(A)$ is then obtained similarly as $\mathsf{reduce}(A)$ by removing useless states of $\mathsf{wcoreduce}(A)$.

Definition 3.2. *Given a* wnVPA $A = (Q, I, F, \Gamma, \delta)$*, we define the* wnVPA $\mathsf{wcoreduce}(A) = (Q', I', F', \Gamma', \delta')$ *with* $Q' = \mathsf{WN}$*,* $I' = \{(i, i) \mid i \in I\}$*,* $F' = \mathsf{WN} \cap (I \times F)$*,* $\Gamma' = \Gamma \times Q$*, and* δ' *defined by its restrictions on call, return and internal symbols respectively (namely* δ'_c*,* δ'_r *and* δ'_ι*):*

- $\delta'_c = \{((p, q), c, (\gamma, p), (q', q')) \mid (p, q) \in Q', (q, c, \gamma, q') \in \delta_c\}$
- $\delta'_r = \left\{ ((p, q), r, (\gamma, p'), (p', q')) \;\middle|\; \begin{array}{l} (p, q), (p', q') \in Q', (q, r, \gamma, q') \in \delta_r, \\ \exists c \in \Sigma_c, \exists s \in Q, (s, c, \gamma, p) \in \delta_c \text{ and } (p', s) \in Q' \end{array} \right\}$
- $\delta'_\iota = \{((p, q), a, (p, q')) \mid (p, q), (p, q') \in Q', (q, a, q') \in \delta_\iota\}$

As we did for wreduce, the states (and the stack) of $\mathsf{wreduce}(A)$ extend those of A with an additional state of A. Here this information is used to store the state q reached when the current top symbol of the stack was pushed. To obtain a weakly co-reduced VPA we require for the return transitions the existence of a matching call transition that allows one to go through the source state q. Similarly as wreduce and reduce, the constructions wcoreduce and coreduce have the following properties:

Theorem 3.2. *Let A be a* wnVPA*, and let $A_{\mathsf{wcor}} = \mathsf{wcoreduce}(A)$ and $A_{\mathsf{cor}} = \mathsf{coreduce}(A)$. A_{wcor} and A_{cor} can be built in polynomial time, and satisfy:*

(1) *for all $w \in \Sigma^*$ there exist bijections between $\mathsf{ARun}_w(A_{\mathsf{wcor}})$ and $\mathsf{ARun}_w(A)$, and $\mathsf{ARun}_w(A_{\mathsf{cor}})$ and $\mathsf{ARun}_w(A)$ and thus in particular $L(A) = L(A_{\mathsf{wcor}}) = L(A_{\mathsf{cor}})$,*

(2) *A_{wcor} is weakly co-reduced, and A_{cor} is co-reduced.*

To prove Theorem 3.2, we proceed the same way as in the proof of Theorem 3.1. Thus we define a mapping Φ_{wcor} from $\mathsf{Run}(A_{\mathsf{wcor}})$ to $\mathsf{Run}(A)$. For any run $\rho' = ((p_0, q_0), \sigma_0)\Pi_{i=1}^{k}(\alpha_i((p_i, q_i), \sigma_i))$ of A_{wcor}, we let $\Phi_{\mathsf{wcor}}(\rho')$ be the run $(q_0, \pi_1(\sigma_0))\Pi_{i=1}^{k}(\alpha_i(q_i, \pi_1(\sigma_i)))$.

PROOF. Since wcoreduce can be seen as the dual of wreduce, we can prove that Φ_{wcor} is a bijection between $\mathsf{ARun}_w(A_{\mathsf{wcor}})$ and $\mathsf{ARun}_w(A)$ by following the lines of the proof of property (1) of Theorem 3.1.

Due to Proposition 2.2, for wnVPA, being co-reduced is the exact dual of being reduced. Therefore, the proof of property (2) can be done by following the lines of the proof of property (2) of Theorem 3.1. □

The co-reduction construction preserves two importants properties:

Proposition 3.2. *Let A be a reduced* VPA*, then $\mathsf{coreduce}(A)$ is also reduced.*

PROOF. The proof of this proposition follows the lines of that of Proposition 3.1. □

Proposition 3.3. *Let A be a deterministic* VPA*, then $\mathsf{coreduce}(A)$ is also deterministic.*

PROOF. The proof of this proposition is done by inspecting the transitions of A_{wcor}. □

3.3. Trimming a wnVPA

We define the construction $\mathsf{trim}_{\mathsf{wn}}$ as $\mathsf{trim}_{\mathsf{wn}} = \mathsf{coreduce} \circ \mathsf{reduce}$. Proposition 3.2 entails that the construction coreduce preserves the reduction, and thus by Theorems 3.1 and 3.2 we obtain the following result:

Theorem 3.3. *Let A be a* wnVPA*, and $A_{\mathsf{trim}} = \mathsf{trim}_{\mathsf{wn}}(A)$. A_{trim} is trimmed and can be built in polynomial time. Furthermore, for all $w \in \Sigma^*$, there exists a bijection between $\mathsf{ARun}_w(A)$ and $\mathsf{ARun}_w(A_{\mathsf{trim}})$, and thus in particular $L(A) = L(A_{\mathsf{trim}})$.*

4. Trimming VPA

We now present a construction which turns a VPA A into a wnVPA over a special alphabet. In a second step, we trim the resulting VPA using the procedure described in Section 3 for wnVPA. Last, from the resulting wnVPA we construct a VPA over the original alphabet recognizing the language $L(A)$ and which is still trimmed. It is not obvious to propose procedures for transforming a VPA into wnVPA, and back, which are compatible with the notion of being trimmed.

4.1. From VPA *to* wnVPA...

Words from Σ^* differ from well-nested words as they admit unmatched returns and calls. We address these issues by extending the alphabet Σ and by modyfing words from Σ^*: some new symbols are added and used to complete the unmatched calls by adding new return symbols and to replace unmatched returns by new internal symbols.

Let $\Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_\iota$ be a structured alphabet. We introduce the structured alphabet $\Sigma^{\mathsf{ext}} = \Sigma_c^{\mathsf{ext}} \uplus \Sigma_r^{\mathsf{ext}} \uplus \Sigma_\iota^{\mathsf{ext}}$ defined by $\Sigma_c^{\mathsf{ext}} = \Sigma_c$, $\Sigma_r^{\mathsf{ext}} = \Sigma_r \uplus \{\bar{r}\}$, and $\Sigma_\iota^{\mathsf{ext}} = \Sigma_\iota \uplus \{a_r \mid r \in \Sigma_r\}$, where $\bar{r}$ and $\{a_r \mid r \in \Sigma_r\}$ are fresh symbols.

We define inductively the mapping Ω which transforms a word over Σ into a word over Σ^{ext} as follows, given $a \in \Sigma_\iota$, $r \in \Sigma_r$, $c \in \Sigma_c$ and $n \in \mathbb{N}$:

- $\Omega(\epsilon, n) = \epsilon, \qquad \Omega(aw, n) = a\,\Omega(w, n)$,
- $\Omega(rw, n) = r\,\Omega(w, n-1)$ if $n > 0$, $\qquad \Omega(rw, 0) = a_r\,\Omega(w, 0)$,
- $\Omega(cw, n) = \begin{cases} cw_1 r\,\Omega(w_2, n) & \text{if } \exists w_1 \in \Sigma^*_{\mathsf{wn}},\ w_2 \in \Sigma^* \text{ and } r \in \Sigma_r \text{ such that } w = w_1 r w_2, \\ c\,\Omega(w, n+1)\,\bar{r} & \text{otherwise.} \end{cases}$

For example, $\Omega(rrccar, 1) = ra_r ccar\bar{r}$. We denote by $\mathsf{ext}(w)$ the word $\Omega(w, 0)$. Intuitively, this mapping replaces every unmatched return r by the internal symbol a_r and adds a suffix of the form $\bar{r}^*$ in order to match every unmatched call. We can prove by induction on the structure of w that $\mathsf{ext}(w)$ is a well-nested word over the alphabet Σ^{ext}.

Given a word w and a natural n, $\Omega(w, n)$ is obtained from w by replacing every unmatched return r of height less than n by the internal symbol a_r and adds a suffix of the form $\bar{r}^*$ in order to match every unmatched call. We extend the functions Ω and ext to languages in the obvious way.

We define also a variant of Ω which does not add the suffix $\bar{r}^*$, we call this mapping Ω_ι:

$$\Omega_\iota(w, n) = w' \text{ such that } \Omega(w, n) = w'w'' \text{ with } w' \in (\Sigma^{\mathsf{ext}} \backslash \{\bar{r}\})^* \text{ and } w'' \in \bar{r}^*$$

We also define $\mathsf{ext}_\iota(w)$ as $\Omega_\iota(w, 0)$. We extend the function Ω_ι to runs in the following way:

$$\Omega_\iota(\kappa_0 \Pi_{k=1}^{l}(\alpha_k \kappa_k), n) = \kappa_0 \Pi_{k=1}^{l}(\alpha'_k \kappa_k) \qquad \text{with } \Omega_\iota(\alpha_1 \ldots \alpha_l, n) = \alpha'_1 \ldots \alpha'_l$$

Let A be a VPA, we now present the construction extend which turns a VPA over Σ into a wnVPA over Σ^{ext}. Intuitively, the construction adds a suffix to the VPA which reads words from $\bar{r}^+$ and replaces the returns on empty stack by internals, thus implementing the mapping ext. To achieve this, the VPA must have some specific properties, introduced in the next definition:

Definition 4.1. *Let $A = (Q, I, F, \Gamma, \delta)$ be a* VPA. *Then A is* stack-compliant *if there exist partitions of Q as $Q = Q_\perp \uplus Q_{\not\perp}$ and Γ as $\Gamma_\perp \uplus \Gamma_{\not\perp}$ such that:*

1. *For all initialized run of A leading to a configuration (p, σ), $\sigma = \perp$ iff $p \in Q_\perp$,*
2. *For all initialized run of A leading to a configuration (p, σ), $\sigma \in \{\perp\} \cup (\Gamma_\perp \cdot \Gamma^*_{\not\perp})$,*
3. *For all $t \in \delta_r$,* $\mathsf{source}(t) \notin F$.

We can now present the construction extend, which operates on stack-compliant VPA. We will present in Subsection 4.3 the construction scompliant that allows to build stack-compliant VPA.

Definition 4.2. *Let $A = (Q, I, F, \Gamma, \delta)$ be a stack-compliant* VPA *over the alphabet Σ with partitions $Q = Q_\perp \uplus Q_{\not\perp}$ and $\Gamma = \Gamma_\perp \uplus \Gamma_{\not\perp}$. We construct* $\mathsf{extend}(A) = (Q', I', F', \Gamma', \delta')$ *as the* wnVPA *over the alphabet Σ^{ext} where: $Q' = Q \cup \{\bar{f}_\perp, \bar{f}_{\not\perp}\}$, $I' = I$, $F' = (F \cap Q_\perp) \cup \{\bar{f}_\perp\}$, $\Gamma' = \Gamma$, and δ' is given by:*

- $\delta'_c = \delta_c$ *and* $\delta_r^{\perp\prime} = \varnothing$
- $\delta'_r = \delta_r \cup \{(p, \bar{r}, \gamma, \bar{f}_\tau) \mid \tau \in \{\perp, \not\perp\}, p \in F \cap Q_{\not\perp}, \gamma \in \Gamma_\tau\} \cup \{(\bar{f}_{\not\perp}, \bar{r}, \gamma, \bar{f}_\tau) \mid \tau \in \{\perp, \not\perp\}, \gamma \in \Gamma_\tau\}$
- $\delta'_\iota = \delta_\iota \cup \{(p, a_r, q) \mid (p, r, \perp, q) \in \delta_r^\perp, p \in Q_\perp\}$

Intuitively, $\mathsf{extend}(A)$ has two components: the first one, composed of the states of A, can read words over $(\Sigma^{\mathsf{ext}} \backslash \{\bar{r}\})^*$ and replaces every unmatched return by internal symbols, whereas the second one, composed of the two added states $\{\bar{f}_\perp, \bar{f}_{\not\perp}\}$, reads words of the form $\bar{r}^+$. This intuition is formalized for an arbitrary VPA over the alphabet Σ^{ext} in the next definition:

Definition 4.3. *Let Σ be an alphabet and $A = (Q, I, F, \Gamma, \delta)$ be a* VPA *over the alphabet Σ^{ext}. We define two subsets of Q as follows:*

$\mathsf{trap}(A) \quad = \{q \in Q \mid \exists p, (p, \bar{r}, \gamma, q) \in \delta_r\}$

$\mathsf{border}(A) \quad = \{p \notin \mathsf{trap}(A) \mid \exists t \in \delta$ *such that* $\mathsf{source}(t) = p$ *and* $\mathsf{target}(t) \in \mathsf{trap}(A)\}$

Elements of $\mathsf{border}(A)$ *are called* border states *of A.*

Border states are the only ones that allow to reach a state in $\mathsf{trap}(A)$ from a state not in $\mathsf{trap}(A)$. One can verify that in the case of the VPA $\mathsf{extend}(A)$, we have $\mathsf{trap}(\mathsf{extend}(A)) = \{\bar{f}_{\not\perp}, \bar{f}_\perp\}$ and $\mathsf{border}(\mathsf{extend}(A)) = F \cap Q_{\not\perp}$. We give an example of these properties in Figure 4.

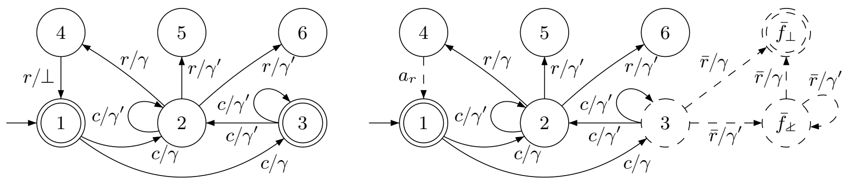

Figure 4: On the left, the VPA $A = (Q, I, F, \Gamma, \delta)$, on the right the wnVPA $A_{\mathsf{ext}} = \mathsf{extend}(A)$. The dashed parts are the components of A modified by extend. A is stack-compliant for $Q = Q_\perp \uplus Q_{\not\perp}$ and $\Gamma = \Gamma_\perp \uplus \Gamma_{\not\perp}$ where $Q_\perp = \{1, 4\}$, $Q_{\not\perp} = \{2, 3, 5, 6\}$, $\Gamma_\perp = \{\gamma\}$ and $\Gamma_{\not\perp} = \{\gamma'\}$. We will see later that A is produced by the construction $\mathsf{scompliant}$. Moreover we have $\mathsf{border}(A_{\mathsf{ext}}) = \{3\}$ and $\mathsf{trap}(A_{\mathsf{ext}}) = \{\bar{f}_{\not\perp}, \bar{f}_\perp\}$.

Let $A = (Q, I, F, \Gamma, \delta)$ be a stack-compliant VPA with partitions $Q = Q_\perp \uplus Q_{\not\perp}$ and $\Gamma = \Gamma_\perp \uplus \Gamma_{\not\perp}$. We define $A_{\mathsf{ext}} = \mathsf{extend}(A)$ and will prove that $L(A_{\mathsf{ext}}) = \mathsf{ext}(L(A))$.

Lemma 4.1. *Let $w = \alpha_1 \ldots \alpha_k \in \Sigma^*$ be a word and $\rho = \kappa_0 \Pi_{k=1}^{\ell}(\alpha_k(\kappa_k))$, with κ_i a configuration of A for all i, then ρ is an initialized run of A over w if and only if $\mathsf{ext}_\iota(\rho)$ is an initialized run of A_{ext} over $\mathsf{ext}_\iota(w)$.*

PROOF. By definition, $\mathsf{ext}_\iota(w)$ is obtained from w by replacing every unmatched return by an internal symbol. The construction extend implements correctly this transformation thanks to point (1) of definition of stack-compliance. □

Lemma 4.2. *Let ρ be an initialized run of A_{ext} over some word w leading to configuration (p, σ) such that p is a border state of A_{ext}, then there exists exactly one run ρ' over $\bar{r}^{|\sigma|}$ such that $\rho\rho'$ is an accepting run of A_{ext}.*

PROOF. By construction of A_{ext}, the word w does contain any occurrence of the letter $\bar{r}$. Thus, by Lemma 4.1, there exists a run ρ_1 of A such that $\rho = \mathsf{ext}_\iota(\rho_1)$. As observed above, we have $\mathsf{border}(A_{\mathsf{ext}}) = F \cap Q_{\not\perp}$. Thus by point (2) of definition of stack-compliance, $\sigma \in \Gamma_\perp \cdot \Gamma_{\not\perp}^*$, and by construction there exists a unique run of A_{ext} which leads from p to $\bar{f}_\perp$ over $\bar{r}^{|\sigma|}$ and pops all the symbols of the stack. □

Theorem 4.1. *Let $w \in \Sigma^*$ be a word, there exists a bijection between $\mathsf{ARun}_w(A)$ and $\mathsf{ARun}_{\mathsf{ext}(w)}(A_{\mathsf{ext}})$, and thus $L(A_{\mathsf{ext}}) = \mathsf{ext}(L(A))$.*

PROOF. First suppose that w is a word without unmatched calls, then $\mathsf{ext}_\iota(w) = \mathsf{ext}(w)$. Let ρ be an accepting run of A over w leading to configuration (p, σ). $\mathsf{ext}_\iota(w) = \mathsf{ext}(w)$ entails $\sigma = \perp$ and thus, by point (1) of the definition of stack-compliance, we have $p \in F \cap Q_\perp$. The result follows from Lemma 4.1.

Otherwise, w is a word with some unmatched calls. We define the set ERun_w as $\mathsf{ERun}_w = \{\rho \in \mathsf{Run}_{\mathsf{ext}_\iota(w)}(A_{\mathsf{ext}}) \mid$ ρ is initialized and ends in a border state$\}$.

As border states of A_{ext} arise from accepting states of A and by Lemma 4.1, for all run $\rho \in \mathsf{Run}_w(A)$, $\rho \in \mathsf{ARun}_w(A)$ if and only if $\mathsf{ext}_\iota(\rho) \in \mathsf{ERun}_w$. The result follows from Lemma 4.2. □

4.2. ... And back.

We will define the converse of the function extend, named retract, which performs the last step of the procedure. However, we have to identify the trap amongst the states of the wnVPA in order to remove it and recover the original language. This transformation is not always possible, and thus we introduce a sufficient condition, named retractability, allowing to apply the retract procedure:

Definition 4.4. *Let Σ be an alphabet and $A = (Q, I, F, \Gamma, \delta)$ be a* VPA *over the alphabet Σ^{ext}. Let us consider the following conditions:*

(1) *There exists a* VPA *B over Σ such that $L(A) = \mathsf{ext}(L(B))$,*

(2) *We have $\mathsf{trap}(A) \cap I = \varnothing$,*

(3) *For all transitions t in δ, $\mathsf{letter}(t) = \bar{r}$ if and only if $\mathsf{target}(t) \in \mathsf{trap}(A)$,*

(4) *For all transitions t in δ such that $\mathsf{source}(t) \in \mathsf{trap}(A)$, $\mathsf{letter}(t) = \bar{r}$,*

(5) *For each initialized run of A, which ends in a border state there exists exactly one run ρ' over $\bar{r}^+$ such that $\rho\rho'$ is an accepting run.*

(6) *For all transitions $t \in \delta_r$ such that $\mathsf{source}(t) \in \mathsf{border}(A)$, $\mathsf{letter}(t) = \bar{r}$.*

We say that A is retractable *if it fulfills conditions $(1), \ldots, (5)$ and* strongly retractable *if it is retractable and fulfills condition (6).*

Retractability properties warrant that retract can be applied. In the sequel we will prove that our different constructions preserve the retractability property and thus preserve the trap: for example the trap of the wnVPA produced by wreduce $\circ$ extend is of the form $\{(\bar{f}_\perp, \bar{f}_\perp), (\bar{f}_{\not\perp}, \bar{f}_{\not\perp})\}$. Note that we need the strong retractability property because the reduce construction does not preserve the retractability. We first show that the construction extend ensures the strong retractability:

Theorem 4.2. *Let A be a stack-compliant* VPA*, then* $\mathsf{extend}(A)$ *is strongly retractable.*

PROOF. Property (1) is a consequence of Theorem 4.1. Properties (2), (3), (4) are consequences of the construction, property (5) is a consequence of Lemma 4.2 and property (6) is a consequence of point (3) of definition of stack-compliance. □

We now define the construction retract:

Definition 4.5. *Let $A = (Q, I, F, \Gamma, \delta)$ be a retractable* wnVPA *over the alphabet Σ^{ext}, we define the* VPA $\mathsf{retract}(A) = (Q', I', F', \Gamma, \delta')$ *over the alphabet Σ with $Q' = Q \backslash \mathsf{trap}(A)$, $I' = I$, $F' = (F \backslash \mathsf{trap}(A)) \cup \mathsf{border}(A)$, and the set of transition rules $\delta' = \delta'_c \uplus \delta'_r \uplus \delta'^{\perp}_r \uplus \delta'_\iota$ is defined by $\delta'_c = \delta_c$, $\delta'_r = \{t \in \delta_r \mid \mathsf{letter}(t) \neq \bar{r}\}$, $\delta'^{\perp}_r = \{(p, r, \perp, q) \mid (p, a_r, q) \in \delta_\iota\}$, and $\delta'_\iota = \{t \in \delta_\iota \mid \mathsf{letter}(t) \in \Sigma_\iota\}$.*

This construction works as follows: it replaces all the internal transitions over a_r by return transitions on empty stack over symbol r, and removes the return transitions over $\bar{r}$. Note that the final states of $\mathsf{retract}(A)$ are the final states of A which are not in $\mathsf{trap}(A)$ and the border states of A.

Lemma 4.3. *Let $w = \alpha_1 \ldots \alpha_k \in \Sigma^*$ be a word and $\rho = \kappa_0 \Pi_{k=1}^{\ell}(\alpha_k(\kappa_k))$, with κ_i a configuration of* $\mathsf{retract}(A)$ *for all i, starting in configuration $\kappa_0 = (p, \sigma)$, then $\rho \in \mathsf{Run}_w(\mathsf{retract}(A))$ if and only if $\Omega_\iota(\rho, |\sigma|) \in \mathsf{Run}_{\Omega_\iota(w,|\sigma|)}(A)$.*

PROOF. First observe that $w \in \Sigma^*$ entails, by points (2) and (3) of the definition of retractability, that configurations $\kappa_i = (p_i, \sigma_i)$ verify the property $p_i \notin \mathsf{trap}(A)$ for all i. As Ω_ι is applied on ρ with $|\sigma|$ as second argument, it replaces the unmatched returns of w that are taken in ρ with an empty stack by internal symbols. The lemma is then proved by an induction on the length of w. □

Theorem 4.3. *Let A be a retractable* VPA *on Σ^{ext} then for any word $w \in \Sigma^*$, there exists a bijection between* $\mathsf{ARun}_w(\mathsf{retract}(A))$ *and* $\mathsf{ARun}_{\mathsf{ext}(w)}(A)$*, and thus in particular $L(A) = \mathsf{ext}(L(\mathsf{retract}(A)))$.*

PROOF. Let $w \in \Sigma^*$ be a word and distinguish two cases. If w is a word without unmatched calls, then $\mathsf{ext}(w) = \mathsf{ext}_\iota(w)$. By Lemma 4.3 and point (2) of the definition of retractability, ext induces a bijection between the set of initialized runs of $\mathsf{retract}(A)$ over w and the set of initialized runs of A over $\mathsf{ext}(w)$. In addition, an initialized run ρ of $\mathsf{retract}(A)$ over w is accepting iff $\mathsf{ext}(\rho)$ is accepting in A. Indeed, as w has no unmatched calls, the configuration (p, σ) reached by ρ verifies $\sigma = \perp$, and thus $p \notin \mathsf{border}(A)$ by point (5) of the definition of retractable. Then p is a final state of $\mathsf{retract}(A)$ iff p is a final state of A. Otherwise, w is a word with some unmatched calls. We define the set RRun_w as $\mathsf{RRun}_w = \{\rho \in \mathsf{Run}_{\mathsf{ext}_\iota(w)}(A) \mid \rho \text{ is initialized and ends in a border state}\}$.

As in the previous case, we can show by point (2) of the definition of retractability and by Lemma 4.3 that ext_ι induces a bijection between $\mathsf{ARun}_w(\mathsf{retract}(A))$ and RRun_w. Thus by point (5) of the definition of retractable, we have a bijection between $\mathsf{ARun}_w(\mathsf{retract}(A))$ and $\mathsf{ARun}_{\mathsf{ext}(w)}(A)$.

Last, by point (1) of the definition of retractability, any word accepted by A is of the form $\mathsf{ext}(w)$ for some $w \in \Sigma^*$, and thus the previous bijections imply $L(A) = \mathsf{ext}(L(\mathsf{retract}(A)))$. □

Our construction retract preserves the trim property:

Proposition 4.1. *Let A be a retractable* VPA *on Σ^{ext}, then if A is trimmed, so is* $\mathsf{retract}(A)$*.*

PROOF. Let $A_{\mathsf{ret}} = \mathsf{retract}(A)$. For the reduced part, let us consider an initialized run $\rho : (p, \perp) \xrightarrow{w} (q, \sigma)$ of A_{ret} over w. By Lemma 4.3 $\hat{\rho} = \mathsf{ext}_\iota(\rho)$ is a run of A over $\hat{w} = \mathsf{ext}_\iota(w)$. Observe that $\hat{\rho}$ is initialized. Since A is reduced, there exists a run $\hat{\rho}'$ over w' such that $\hat{\rho}\hat{\rho}'$ is an accepting run of A. By construction we can decompose w' as $w' = w_1 w_2$ with $w_1 \in \Sigma^*$ and $w_2 \in \bar{r}^*$ and $\hat{\rho}'$ as $\hat{\rho}_1 : (q, \sigma) \xrightarrow{w_1} (q', \sigma')$ over w_1 and $\hat{\rho}_2 : (q', \sigma') \xrightarrow{w_2} (p', \perp)$ over w_2. If $w_2 = \varepsilon$, then $\sigma' = \perp$ and by construction q' is a final state of A_{ret}. By Lemma 4.3, there exists a run ρ_1 such that $\hat{\rho}_1 = \Omega(\rho_1, |\sigma|)$ and $\rho\rho_1$ is an accepting run of A_{ret}. If $w_2 \neq \varepsilon$, then $q' \in \mathsf{border}A$ and so q' is a final state of A_{ret}. Again, by Lemma 4.3 there exists a run ρ_1 such that $\hat{\rho}_1 = \Omega(\rho_1, |\sigma|)$ and $\rho\rho_1$ is an accepting run of A_{ret}.

For the co-reduced part, let us consider two runs $\rho : (p, \sigma) \xrightarrow{w} (q, \sigma')$ and $\rho' : (i, \perp) \xrightarrow{w'} (q, \sigma')$ of A_{ret} with q a final state and i an initial state. We distinguish two cases:

- If $\sigma' = \perp$, then by point (5) of the definition of retractability, state q cannot be in $\mathsf{border}(A)$. By definition of A_{ret}, this implies that q is a final state of A. By Lemma 4.3, $\hat{\rho} = \Omega_\iota(\rho, |\sigma|)$ is a run of A over $\hat{w} = \Omega_\iota(w, |\sigma|)$, then since A as a wnVPA and is co-reduced, there exists a run $\hat{\rho}''$ of A over a word $\hat{w}''$ such that $\hat{\rho}''\hat{\rho}$ is an accepting run of A over $\hat{w}''\hat{w}$. Thus by Lemma 4.3 there exists a run ρ'' of A_{ret} such that $\hat{\rho}'' = \mathsf{ext}_\iota(\rho'')$ over w'' with $\hat{w}'' = \mathsf{ext}(w'')$ and $\rho''\rho$ is an accepting run of A_{ret} over $w''w$.

- If $\sigma' \neq \perp$, then by construction, $q \in \mathsf{border}(A)$. By Lemma 4.3, $\hat{\rho} = \Omega_\iota(\rho, |\sigma|)$ and $\hat{\rho}' = \mathsf{ext}_\iota(\rho')$ are runs of A over $\Omega_\iota(w, |\sigma|)$ and $\mathsf{ext}_\iota(w')$ respectively. Observe that $\hat{\rho}'$ is initialized. Then by point (5) of the definition of retractability there exists exactly one run $\bar{\rho}$ over $\bar{r}^+$ such that $\hat{\rho}'\bar{\rho}$ is an accepting run of A and so $\hat{\rho}\bar{\rho}$ is also a run of A. Since A is co-reduced there exists a run $\hat{\rho}''$ of A over a word $\hat{w}''$ such that $\hat{\rho}''\hat{\rho}\bar{\rho}$ is an accepting run of A. By Lemma 4.3, there exists a run ρ'' of A_{ret} such that $\hat{\rho}'' = \mathsf{ext}_\iota(\rho'')$ over w'' with $\hat{w}'' = \mathsf{ext}(w'')$. Thus by Lemma 4.3, $\rho''\rho$ is an accepting run of A_{ret} over $w''w$. □

We prove that the constructions reduce preserves the strong retractability and coreduce preserves the retractability:

Proposition 4.2. *Let A be a strongly retractable* VPA*, then $A_{\mathsf{red}} = \mathsf{reduce}(A)$ is strongly retractable.*

PROOF. We let $A = (Q, I, F, \Gamma, \delta)$ and $A_{\mathsf{red}} = (Q', I', F', \Gamma', \delta')$. First note that by construction:

$$\text{For all } (p, q) \in Q', \text{ if } (p, q) \in \mathsf{trap}(A_{\mathsf{red}}) \text{ then } p \in \mathsf{trap}(A) \tag{1a}$$

$$\text{For all } (p, q) \in Q', \text{ if } (p, q) \in \mathsf{border}(A_{\mathsf{red}}) \text{ then } p \in \mathsf{border}(A) \tag{1b}$$

Equation (1a) is a consequence of the construction reduce. Concerning Equation (1b), to prove that $p \in \mathsf{border}(A)$, the fact that $p \notin \mathsf{trap}(A)$ follows from point (3) of Definition 2.4 applied on (p, q), and the fact that there exists a transition on $\bar{r}$ from p to $\mathsf{trap}(A)$ follows from the construction reduce.

Condition (1) of the definition of retractibility is obvious since $L(A) = L(A_{\mathsf{red}})$. Conditions (2), (3) and (4) can be easily proved using Equation (1a), and condition (6) can be proved using Equation (1b).

To prove condition (5), we consider an initialized run ρ of A_{red} over some word w leading to configuration $((q, q), \sigma)$ such that $(q, q) \in \mathsf{border}(A_{\mathsf{red}})$. By Lemma 3.2, $\rho' = \Phi_{\mathsf{wred}}(\rho)$ is a run of A leading to configuration $(q, \pi_1(\sigma))$. As $q \in \mathsf{border}(A)$ (by Equation (1b)) and A is retractable, $\pi_1(\sigma)$ is not empty and so on for σ. Thus we can decompose ρ as follows:

$$\rho : ((i, f), \bot) \xrightarrow{w'} ((p', q'), \sigma') \xrightarrow{c} ((p, q), \sigma) \xrightarrow{w''} ((q, q), \sigma)$$

with $w' \in \Sigma^*$, $w'' \in \Sigma^*_{\mathsf{wn}}$, $c \in \Sigma_c$ and $\sigma = \sigma' \cdot (\gamma, q')$ for some $\gamma \in \Gamma$. In addition, by definition of reduced, there exists a transition $(q, r, \gamma, q'') \in \delta_r$. Since $q \in \mathsf{border}(A)$, by condition (6) of retractability we have $r = \bar{r}$, thus there exists a transition $((q, q), \bar{r}, (\gamma, q'), (q'', q')) \in \delta'_r$ and so $\rho_1 : ((q, q), \sigma) \xrightarrow{\bar{r}} ((q'', q'), \sigma')$ is a run of A_{red}. Since A_{red} is reduced, we can complete the initialized run $\rho\rho_1$ into an accepting run. By condition (1) of retractability, accepted words are of the form $\Sigma^*\bar{r}^*$. Thus there exists a run $\bar{\rho} = ((q'', q'), \sigma') \xrightarrow{\bar{r}^{|\sigma'|}} ((f, f), \bot)$ such that $\rho\rho_1\bar{\rho}$ is an accepting run of A_{red}. Last, unicity of the extension of ρ follows from the bijection Φ_{wred}. □

Proposition 4.3. *Let A be a retractable* VPA*, then $A_{\mathsf{cor}} = \mathsf{coreduce}(A)$ is retractable.*

PROOF. We let $A = (Q, I, F, \Gamma, \delta)$ and $A_{\mathsf{cor}} = (Q', I', F', \Gamma', \delta')$. First note that by construction:

For all $(p, q) \in Q'$, if $(p, q) \in \mathsf{trap}(A_{\mathsf{cor}})$ then $q \in \mathsf{trap}(A)$ (2a)

For all $(p, q) \in Q'$, if $(p, q) \in \mathsf{border}(A_{\mathsf{cor}})$ then $q \in \mathsf{border}(A)$ (2b)

Equations (2a) and (2b) can be proved similarly as Equations (1a) and (1b).

Condition (1) of the definition of retractability is obvious since $L(A) = L(A_{\mathsf{cor}})$. Conditions (2), (3) and (4) can be easily proved using Equation (2a).

To prove condition (5), we first state the following property which easily follows from the definition of coreduce: for all initialized run ρ of A_{cor} leading to some configuration $((p, q), \sigma)$, let ρ' be the run $\Phi_{\mathsf{wcor}}(\rho)$ of A leading to the configuration $(q, \bar{\sigma})$ with $\bar{\sigma} = \pi_1(\sigma)$, if there exists a transition $(q, r, \gamma, q') \in \delta_r$ such that the run $\rho' r(q', \bar{\sigma}')$ exists in A with $\bar{\sigma} = \bar{\sigma}'\gamma$, then there exists a transition $((p, q), r, (\gamma, p'), (p', q')) \in \delta'_r$ such that the run $\rho r\ ((p', q'), \sigma')$ exists in A_{cor}, with $\pi_1(\sigma') = \bar{\sigma}'$.

Then, condition (5) easily follows from this property, Equation (2b) and the fact that A is retractable. Unicity follows from the bijection Φ_{wcor}. □

4.3. The construction scompliant

We have seen in Subsection 4.1 how to transform a VPA into a wnVPA by means of the construction extend. However, extend requires as an input a stack-compliant VPA. In this section, we propose a construction scompliant that transforms any VPA into an equivalent stack-compliant one.

We first give an intuition on the construction and then show how it relates to the partitions of states and stack symbols from the definition of stack-compliance (Definition 4.1). Our construction first distinguishes three kinds of states: states for which the stack is empty, states stating that the top of the stack will be later popped and states stating that the top of the stack will remain in the stack. Stacks are used to retain correct information in the states and therefore will be of one of the following forms: the empty stack, or stacks composed of symbols that have to remain in the stack over which some symbols intended to be popped.

We now present the construction scompliant. We first define the set of special symbols $T = \{\bot, \top, \circ\}$.

Definition 4.6. *Given a* VPA $A = (Q, I, F, \Gamma, \delta)$*, we define the* VPA $\mathsf{scompliant}(A) = (Q', I', F', \Gamma', \delta')$*, with $Q' = (Q \times T)$, $I' = I \times \{\bot\}$, $F' = F \times \{\bot, \circ\}$, $\Gamma' = \Gamma \times T$, and δ' is given by:*

- $\delta'_c = \{((p, \tau), c, (\gamma, \tau), (q, \tau')) \mid (p, c, \gamma, q) \in \delta_c$ and either $\tau' {=} \top$ or $(\tau' {=} \circ \wedge \tau {\neq} \top)\}$
- $\delta'_r = \{((p, \top), r, (\gamma, \tau), (q, \tau)) \mid (p, r, \gamma, q) \in \delta_r\}$

- $\delta_r^{\perp\prime} = \{((p, \perp), r, \perp, (q, \perp))) \mid (p, r, \perp, q) \in \delta_r\}$
- $\delta_\iota' = \{((p, \tau), a, (q, \tau)) \mid (p, a, q) \in \delta_\iota\}$

The semantics of this construction is as follows: suppose that an initialized run reaches a state (p, τ), then the form of the stack depends on τ. If $\tau = \perp$ then the stack is empty. If $\tau = \circ$ then the symbols in the stack will never be popped. Finally, if $\tau = \top$, then the stack is composed at the bottom of some symbols which will never be popped and on top of them some symbols that will be popped before reaching a final state. We give an example of this construction in Figure 5.

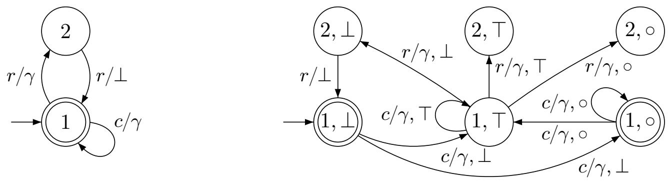

Figure 5: On the left, a VPA A, on the right the VPA $\mathsf{scompliant}(A)$.

Let us now give some basic properties of the construction scompliant:

Lemma 4.4. *Let A be a* VPA *and $\rho : ((p, \tau), \sigma) \xrightarrow{w} ((p', \tau'), \sigma)$ be a run of* $\mathsf{scompliant}(A)$ *over a well nested word w, then $\tau = \tau'$.*

PROOF. By induction on the structure of w. □

For each symbol $\tau \in T$ we define the set S_τ as follows:

$$S_\perp = \{\perp\} \qquad S_\top = (\Gamma \times \{\perp\}) \cdot (\Gamma \times \{\circ\})^* \cdot (\Gamma \times \{\top\})^* \qquad S_\circ = (\Gamma \times \{\perp\}) \cdot (\Gamma \times \{\circ\})^*$$

We now state how, in configurations of runs of $\mathsf{scompliant}(A)$, states are compliant with the stacks.

Lemma 4.5. *Let A be a* VPA *and $\rho = ((p_0, \tau_0), \sigma_0)\Pi_{i=1}^k(\alpha_i((p_i, \tau_i), \sigma_i))$ be a run of* $\mathsf{scompliant}(A)$*. If $\sigma_0 \in S_{\tau_0}$, then for all $i \in \{1, \ldots, k\}$, $\sigma_i \in S_{\tau_i}$.*

PROOF. By induction on the length of ρ. □

Definition 4.7. *Let A be a* VPA *and $\rho = ((q_0, \tau_0), \sigma_0)\Pi_{i=1}^k(\alpha_i((q_i, \tau_i), \sigma_i))$ be a run of* $\mathsf{scompliant}(A)$*, then ρ is a* stack-compliant run *if $\sigma_0 \in S_{\tau_0}$ and $\tau_k \neq \top$.*

Intuitively, in the previous definition, the constraint $\tau_k \neq \top$ entails every symbol of the stack will never be popped before reaching a final configuration. This is formalized in the next lemma.

Lemma 4.6. *Let A be a* VPA *and $\rho : ((p, \tau), \sigma) \xrightarrow{w} ((p', \tau'), \sigma')$ be a stack-compliant run of* $\mathsf{scompliant}(A)$ *with $w \in \Sigma^*$ and $\sigma \neq \perp$. Then we can decompose w as $w_1 r w_2$ with $w_1 \in \Sigma_{\mathsf{wn}}^*$ and $r \in \Sigma_r$ if and only if $\tau = \top$.*

PROOF. From left to right, if $\tau \neq \top$, suppose that we can decompose w as $w_1 r w_2$. By Lemma 4.4 we can also decompose ρ as follows:

$$\rho : ((p, \tau), \sigma) \xrightarrow{w_1} ((q, \tau), \sigma) \xrightarrow{r} ((q', \tau''), \sigma'') \xrightarrow{w_2} ((p', \tau'), \sigma')$$

τ cannot be equal to $\circ$ because there is no transition $t \in \delta_r'$ such that $\mathsf{source}(t) = (q, \circ)$ and $\mathsf{letter}(t) \in \Sigma_r$. τ is also different from $\perp$ because by Lemma 4.5, $\sigma \in S_\tau$ but $\sigma \neq \perp$. Contradiction.

From right to left, if $\tau = \top$, suppose we cannot decompose w as $w_1 r w_2$, then $w = w_0 c_1 w_1 \cdots c_n w_n$, for $n \in \mathbb{N}_{\geqslant 0}$, $c_i \in \Sigma_c$ and $w_i \in \Sigma_{\mathsf{wn}}^*$ for all $i \in \{0, \ldots n\}$. We can decompose ρ as follows:

$$\rho : ((p_0, \tau_0), \sigma_0) \xrightarrow{w_0} ((p_0', \tau_0'), \sigma_0') \xrightarrow{c_1} ((p_1, \tau_1), \sigma_1) \xrightarrow{w_1} \cdots \xrightarrow{c_n} ((p_n, \tau_n), \sigma_n) \xrightarrow{w_n} ((p_n', \tau_n'), \sigma_n')$$

with $((p,\tau),\sigma) = ((p_0,\tau_0),\sigma_0)$ and $((p',\tau'),\sigma') = ((p'_n,\tau'_n),\sigma'_n)$. By definition of transitions of $\mathsf{scompliant}(A)$, if $\tau'_i = \top$ then $\tau_{i+1} = \top$, and by Lemma 4.4 we have $\tau_i = \tau'_i$, thus since $\tau = \tau_0 = \top$ we can conclude that $\tau' = \tau'_n = \top$. This is a contradiction because ρ is a stack-compliant run and τ' must be different from $\top$. □

In the rest of this subsection, we let $A = (Q, I, F, \Gamma, \delta)$ be a VPA and $A_{\mathsf{sc}} = (Q', I', F', \Gamma', \delta')$ be the VPA $\mathsf{scompliant}(A)$. We are now going to prove the following theorem:

Theorem 4.4. *Let A be a* VPA, *then $A_{\mathsf{sc}} = \mathsf{scompliant}(A)$ can be built in polynomial time and:*

- *For all word $w \in \Sigma^*$ there exists a bijection between $\mathsf{ARun}_w(A)$ and $\mathsf{ARun}_w(A_{\mathsf{sc}})$, and so $L(A) = L(A_{\mathsf{sc}})$.*
- *A_{sc} is stack-compliant.*

We define a mapping Φ_{sc} from the runs of A_{sc} to the runs of A. For any run $\rho' = ((q_0,\tau_0),\sigma_0)\Pi_{i=1}^{k}(\alpha_i((q_i,\tau_i),\sigma_i))$ of A_{sc}, we let $\Phi_{\mathsf{sc}}(\rho')$ be the run $(q_0,\pi_1(\sigma_0))\Pi_{i=1}^{k}(a_i(q_i,\pi_1(\sigma_i)))$.

Proposition 4.4. *For any stack-compliant run ρ' of A_{sc}, $\rho = \Phi_{\mathsf{sc}}(\rho')$ is a run of A.*

PROOF. By induction on the length of ρ'. □

Lemma 4.7. *The mapping Φ_{sc} from the stack-compliant runs of* $\mathsf{scompliant}(A)$ *to the runs of A is bijective.*

PROOF. By induction on the length of the stack-compliant runs (the induction step is defined by adding a configuration at the beginning of the run of the hypothesis), using Proposition 4.4 and Lemmas 4.6 and 4.5. □

PROOF (THEOREM 4.4). For the first point note that any run $\rho \in \mathsf{ARun}(A_{\mathsf{sc}})$ is a stack-compliant run of A_{sc}. Thus by Proposition 4.4 and Lemma 4.7, ρ is an accepting run of A_{sc} over w if and only if $\Phi_{\mathsf{sc}}(\rho)$ is an accepting run of A over w. For the second point, conditions (1) and (2) of stack-compliance are a consequence of Lemma 4.5, with $Q'_{\not\perp} = (Q \times \{\circ\}) \cup (Q \times \{\top\})$ $Q'_{\perp} = (Q \times \{\perp\})$, $\Gamma'_{\not\perp} = (\Gamma \times \{\circ\}) \cup (\Gamma \times \{\top\})$ and $\Gamma'_{\perp} = (\Gamma \times \{\perp\})$. Condition (3) is a direct consequence of the construction. □

4.4. Trimming VPA

We consider the construction trim defined by $\mathsf{trim}(A) = \mathsf{retract} \circ \mathsf{trim}_{\mathsf{wn}} \circ \mathsf{extend} \circ \mathsf{scompliant}(A)$, and state its main properties:

Theorem 4.5. *Let A be a* VPA *on the alphabet Σ, and let $A_{\mathsf{trim}} = \mathsf{trim}(A)$. The* VPA *$A_{\mathsf{trim}}$ can be built in polynomial time, and satisfies:*

(1) *there is a bijection between $\mathsf{ARun}(A)$ and $\mathsf{ARun}(A_{\mathsf{trim}})$, and so $L(A) = L(A_{\mathsf{trim}})$,*

(2) *A_{trim} is trimmed.*

PROOF. We can prove this result using theorems and propositions summarized in the table of Section 7. □

5. Deterministic trimmed VPA

We have proven in the previous section that it is always possible, given a VPA, to build an equivalent VPA (*i.e.* recognizing the same language) which is trimmed. In addition, in the original paper of Alur and Madhusudan, it was proven that it is always possible to build an equivalent VPA that is deterministic. In this section, we prove that it is possible to build an equivalent VPA that is both trimmed and deterministic. This is not an immediate corollary of the two previous results, as the construction reduce introduces some non-determinism and the construction of a deterministic VPA given in [2] does not preserve the co-reduction.

We do not show the determinization procedure for VPA, we refer the reader to [2] for details. Given a VPA A as input, we denote by $\mathsf{det}(A)$ the result of this procedure. Note that its complexity is $O\left(2^{n^2}\right)$, where n denotes the number of states of the input VPA. This procedure enjoys the following two properties:

Lemma 5.1 ([2]). *Let A be a* VPA*. Let $w \in \Sigma^*$, and w' be the longest well-nested suffix of w. The (unique) run of* $\mathsf{det}(A)$ *on the word w reaches the state (R, S) defined by:*

- $R = \{p \in Q \mid \exists\ (i, \perp) \xrightarrow{w} (p, \sigma) \in \mathsf{Run}(A)$ *with i an initial state of A*$\}$
- $S = \{(p, q) \in Q \times Q \mid \exists\ (p, \perp) \xrightarrow{w'} (q, \perp) \in \mathsf{Run}(A)\}$

Lemma 5.2 ([2]). *Let $w \in \Sigma^*$. If there exists an initialized run ρ' of* $\mathsf{det}(A)$ *on w (not necessarily accepting), then there exists an initialized run ρ of A on w.*

We refine this construction by removing the useless states from $\mathsf{det}(A)$ (according to property (3) of Definition 2.4), this last phase can be done in polynomial time. We denote by determinize the whole process.

5.1. Determinization preserves reduction and retractability

It is obvious that Lemmas 5.1 and 5.2 are preserved by determinize and that this procedure is correct in the following sense [2]:

Theorem 5.1 ([2]). *Let A be a* VPA. $\mathsf{determinize}(A)$ *is a deterministic* VPA*, and $L(A) = L(\mathsf{determinize}(A))$.*

We prove now that the construction determinize preserves the properties of being weakly reduced and of being retractable. In the sequel, we let A be a VPA and denote by A_{det} the VPA $\mathsf{determinize}(A)$.

Proposition 5.1. *If A is weakly reduced, then* $\mathsf{determinize}(A)$ *is weakly reduced.*

PROOF. Let ρ' be an initialized run of A_{det}. We have to prove that ρ' can be completed into an accepting run of A_{det}. Let us denote by w_1 the word associated with the run ρ'.

By Lemma 5.2, there exists an initialized run ρ of A on the word w_1. As A is weakly reduced, the run ρ can be completed by a run ρ_2 into an accepting run of A, on some word w_2. As a consequence, we have $w_1 w_2 \in L(A)$. By the correction of the determinization procedure, we have $w_1 w_2 \in L(A_{\mathsf{det}})$. Thus there exists an accepting run ρ'' of A_{det} on this word. As A_{det} is deterministic, the prefix of ρ'' on w_1 must be exactly ρ'. This proves the result. □

Proposition 5.2. *If A is retractable, then* $\mathsf{determinize}(A)$ *is retractable.*

PROOF. We denote by Q' the states of A_{det}. First note that by Lemma 5.1:

$$\text{For all } (S, R) \in Q', \text{ if } (S, R) \in \mathsf{trap}(A_{\mathsf{det}}) \text{ then } \exists q \in R \text{ such that } q \in \mathsf{trap}(A) \tag{3a}$$
$$\text{For all } (S, R) \in Q', \text{ if } (S, R) \in \mathsf{border}(A_{\mathsf{det}}) \text{ then } \exists q \in R \text{ such that } q \in \mathsf{border}(A) \tag{3b}$$

For the properties of Definition 4.4: property (1) is obvious since $L(A_{\mathsf{det}}) = L(A)$. Properties $(2) - (4)$ are a consequence of Equation (3a). For the property (5), let us define the initialized run ρ of A_{det} over some word w with $\mathsf{last}(\rho) = ((S, R), \sigma)$ such that (S, R) is a border state of A_{det}. There exists a run ρ' of A over w with $\mathsf{last}(\rho) = (p, \sigma')$, $p \in R$. By Equation (3b), p is a border state. Thus by definition of retractable, there exists a run $\bar{\rho}$ of A over $\bar{r}^k$ such that $\rho'\bar{\rho}$ is an accepting run of A for $k \in \mathbb{N}$. As a consequence, we have $w\bar{r}^k \in L(A)$. By the correction of the determinization procedure, we have $w\bar{r}^k \in L(A_{\mathsf{det}})$. Thus there exists an accepting run ρ'' of A_{det} on this word. As A_{det} is deterministic, the prefix of ρ'' on w is exactly ρ. This proves the result. □

5.2. Construction of a deterministic trimmed VPA

We consider the following composition of the different constructions presented before:

$$\mathsf{det\text{-}trim} = \mathsf{retract} \circ \mathsf{coreduce} \circ \mathsf{determinize} \circ \mathsf{reduce} \circ \mathsf{extend} \circ \mathsf{scompliant}$$

We claim that this composition allows one to build an equivalent VPA that is both deterministic and trimmed:

Theorem 5.2. *Let A be a* VPA*. The* VPA $\mathsf{det\text{-}trim}(A)$ *is deterministic, trimmed, and satisfies $L(A) = L(\mathsf{det\text{-}trim}(A))$.*

PROOF. We can prove this result using theorems and propositions summarized in the table of Section 7. □

5.3. A lower bound for deterministic inputs

One can wonder whether a deterministic VPA can be trimmed with a polynomial time complexity, preserving its deterministic nature. The answer to this question is negative, and we now prove that there is no construction which allows to trim a deterministic VPA in polynomial time preserving determinism. First we recall this well-known result:

Theorem 5.3. *[13] Let k be an integer such that $k > 0$ and T_k be a deterministic finite state automaton over the alphabet $\Gamma = \{\gamma_1, \gamma_2\}$ such that $L(T_k) = (\gamma_1 + \gamma_2)^*\gamma_1(\gamma_1 + \gamma_2)^{k-1}$, then T_k have at least 2^k states.*

Let $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_\iota$ be a structured alphabet such that $\Sigma_c = \{c_1, c_2\}$, $\Sigma_r = \{r\}$, $\Sigma_\iota = \{a\}$ and $\Gamma = \{\gamma_1, \gamma_2\}$. We now present the family of deterministic VPA $A_k = (Q, I, F, \Gamma, \delta)$ where $Q = \{q\} \cup \{q_i \mid i \in \{0, \ldots, k\}\}$, $I = \{q\}$, $F = \{q_k\}$, and $\delta = \delta_c \cup \delta_r \cup \delta_\iota$ with:

- $\delta_c = \{(q, c_1, \gamma_1, q), (q, c_2, \gamma_2, q)\}$
- $\delta_r = \{(q_i, r, \gamma, q_{i+1}) \mid i \in \{0, \ldots, k-2\}, \gamma \in \Gamma\} \cup \{(q_{k-1}, r, \gamma_1, q_k)\} \cup \{(q_k, r, \gamma, q_k) \mid \gamma \in \Gamma\}$
- $\delta_\iota = \{(q, a, q_0)\}$

It is easy to see that A_k recognizes the language $L_k = \{(c_1 + c_2)^n a c_1 (c_1 + c_2)^k r^{k+n+1} \mid n \in \mathbb{N}\}$ with $k \in \mathbb{N}$. The VPA A_k is depicted on Figure 6. We now prove that there is no deterministic and trimmed VPA $\bar{A}_k$ such that $L(A_k) = L(\bar{A}_k)$ for arbitrary k and the size of $\bar{A}_k$ is polynomial in the size of A_k.

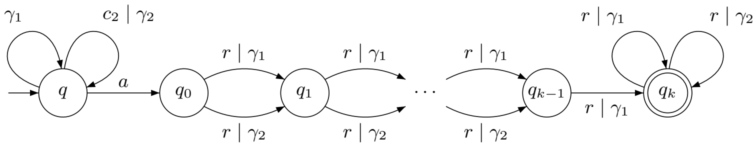

Figure 6: The VPA A_k.

Lemma 5.3. *Let k be an integer such that $k > 0$ and $\rho : (q, \bot) \xrightarrow{w} (q', \sigma)$ be an initialized run of A_k such that $w = w'a$ with $w' \in \Sigma^*$. Then there exists a run ρ' of A_k such that $\rho\rho'$ is an accepting run of A_k if and only if $w' \in (c_1 + c_2)^* c_1 (c_1 + c_2)^{k-1}$.*

PROOF. First note that by construction of A_k we have $q' = q_0$. The only way to reach a final state from (q_0, σ) is for $\sigma \in (\gamma_1 + \gamma_2)^*\gamma_1(\gamma_1 + \gamma_2)^{k-1}$. Since the only way to push a symbol γ_1 (resp. γ_2) is to go through a letter c_1 (resp. c_2) we can conclude that there exists a run ρ' of A_k such that $\rho\rho'$ is accepting if and only if ρ is over a word from $(c_1 + c_2)^* c_1 (c_1 + c_2)^{k-1}$. □

Let k be an integer such that $k > 0$. We define $\bar{A}_k$ as a reduced and deterministic VPA $\bar{A}_k = (\bar{Q}, \bar{I}, \bar{F}, \bar{\Gamma}, \bar{\delta})$ over Σ such that $L(\bar{A}_k) = L(A_k)$.

Lemma 5.4. *For all initialized run $\rho : (p, \bot) \xrightarrow{w'} (p', \sigma)$ of $\bar{A}_k$ such that there exists a transition $(p', a, p'') \in \bar{\delta}_i$, then $w' \in (c_1 + c_2)^* c_1 (c_1 + c_2)^{k-1}$.*

PROOF. Since a is an internal symbol, $(p, \bot) \xrightarrow{w'} (p', \sigma) \xrightarrow{a} (p'', \sigma)$ is a run of $\bar{A}_k$ over $w'a$. Since $\bar{A}_k$ is reduced, by definition there exists a run ρ' of $\bar{A}_k$ such that $\rho\rho'$ is an accepting run of $\bar{A}_k$. By Lemma 5.3 and since $L(A_k) = L(\bar{A}_k)$, $w' \in (c_1 + c_2)^* c_1 (c_1 + c_2)^{k-1}$. □

Lemma 5.5. *The VPA $\bar{A}_k$ has at least 2^k states.*

PROOF. To prove this result, from the VPA $\bar{A}_k = (\bar{Q}, \bar{I}, \bar{F}, \bar{\Gamma}, \bar{\delta})$ we define the finite state automaton $B_k = (Q', I', F', \delta')$ over the alphabet $\Sigma' = \{c_1, c_2\}$ with $Q' = \{p, p' \in \bar{Q} \mid \exists (p, c, \gamma, p') \in \bar{\delta_c}\} \cup \{p \in \bar{Q} \mid \exists (p, a, p') \in \bar{\delta_\iota}\}$, $I' = \bar{I}$, $F' = \{p \in \bar{Q} \mid (p, a, p') \in \bar{\delta_\iota}\}$, and $\delta' = \{(p, c, p') \mid (p, c, \gamma, p') \in \bar{\delta_c}\}$.

First note that B_k is defined as a restriction of $\bar{A}_k$, where the transitions of B_k are the call transitions of $\bar{A}_k$ with the stack symbol removed. Moreover as $\bar{A}_k$ is deterministic, so is B_k. The final states of B_k are the states of $\bar{A}_k$ which have an outgoing transition over a (and by assumption on the language of $\bar{A}_k$, have at least one incoming call transition). Thus by Lemma 5.4, we have $L(B_k) = (c_1 + c_2)^* c_1 (c_1 + c_2)^{k-1}$.

By Theorem 5.3, B_k have 2^k states. Since B_k is defined as a restriction of $\bar{A}_k$, A_k have at least 2^k states. □

Theorem 5.4. *There is no procedure which allows to trim a deterministic* VPA *in polynomial time preserving determinism.*

PROOF. Direct consequence of Lemma 5.5. □

6. Application to VPA with weights

We show in this section that our trimming procedure can be applied to VPA with weights, for instance visibly pushdown transducers (see [11]) where transitions are in addition labelled with output words, and VPA with multiplicities ($\mathbb{N}$-VPA for short, see [7]), where transitions are labelled by integers.

We consider a monoid $(M, \cdot)$ which will be used to represent weights associated with transitions (for instance Σ^* equipped with concatenation in the case of transducers and $\mathbb{N}$ equipped with addition for $\mathbb{N}$-VPA).

Definition 6.1. *A* weighted visibly pushdown automaton *on finite words over* Σ *with weights in* $(M, \cdot)$ *is a pair* $W = (A, \lambda)$ *composed of a* VPA $A = (Q, I, F, \Gamma, \delta)$ *and a mapping* $\lambda : \delta \to M$*, which assigns weights to transitions of* A*.*

The notions of configurations and runs are lifted from VPA to weighted VPA. Given a run ρ over a sequence of transitions $\eta = (t_i)_{1 \leqslant i \leqslant k}$, we define the weight of ρ, denoted $\langle \rho \rangle$, as $\langle \rho \rangle = \Pi_{i=1}^{i=k} \lambda(t_i)$.

Then, the behavior of the weighted VPA $W = (A, \lambda)$ is represented by the formal power serie $\langle\langle W \rangle\rangle$ from Σ^* to multisets over M, defined by $\langle\langle W \rangle\rangle(w) = \{\{\langle \rho \rangle \mid \rho \in \mathsf{ARun}_w(A)\}\}$.

Theorem 6.1. *Let* W *be a weighted* VPA*. We can build in polynomial time a weighted* VPA W' *that is trimmed and such that* $\langle\langle W \rangle\rangle = \langle\langle W' \rangle\rangle$*.*

PROOF (SKETCH). By definition, the weight of a run of a weighted VPA only depends on the run of the underlying VPA. In addition, one can verify that the bijections of runs proved for the different constructions of the paper do all "preserve" the transitions, *i.e.* every transition of the VPA we build is mapped on a single transition of the original VPA, and the bijection respects this mapping. This is clear for the construction reduce (and thus also coreduce) as the mapping is a projection. The case of extend and retract is a bit more involved. Indeed, the construction extend adds a unique suffix to each run, which is intuitively removed by the construction retract. Apart from that suffix, the bijection preserves the transitions of the original VPA. □

7. In a nutshell

We summarize in the table below the results presented in this paper:

Algorithm	Profile	Requires	Preserves	Ensures
reduce	wnVPA → wnVPA		Strong retractability (Proposition 4.2) Co-reduction (Proposition 3.1)	Reduction (Theorem 3.1)
coreduce	wnVPA → wnVPA		Retractability (Proposition 4.3) Reduction (Proposition 3.2) Determinism (Proposition 3.3)	Co-reduction (Theorem 3.2)
determinize	VPA → VPA		Reduction (Proposition 5.1) Retractability (Proposition 5.2)	Determinism (Theorem 5.1)
extend	VPA → wnVPA	Stack-compliance		Strong retractability (Theorem 4.2)
retract	wnVPA → VPA	Retractability	Reduction (Proposition 4.2) Co-reduction (Proposition 4.3)	
scompliant	VPA → VPA			Stack-compliance (Theorem 4.4)

Using these constructions we have defined the three following ones which respectively allow to trim wnVPA, to trim VPA, and to trim and determinize VPA:

$$\begin{aligned} \mathsf{trim}_{\mathsf{wn}} = \quad & \mathsf{coreduce} \circ \mathsf{reduce} \\ \mathsf{trim} = \quad & \mathsf{retract} \circ \mathsf{trim}_{\mathsf{wn}} \circ \mathsf{extend} \circ \mathsf{scompliant} \\ \mathsf{det\text{-}trim} = \quad & \mathsf{retract} \circ \mathsf{coreduce} \circ \mathsf{determinize} \circ \mathsf{reduce} \circ \mathsf{extend} \circ \mathsf{scompliant} \end{aligned}$$

References

[1] R. Alur, V. Kumar, P. Madhusudan, and M. Viswanathan. Congruences for Visibly Pushdown Languages. In *ICALP*, volume 3580 of *LNCS*, pages 1102–1114. Springer, 2005.

[2] R. Alur and P. Madhusudan. Visibly Pushdown Languages. In *STOC*, pages 202–211, 2004.

[3] R. Alur and P. Madhusudan. Adding Nesting Structure to Words. *JACM*, 56(3):1–43, 2009.

[4] J.-M. Autebert, J. Berstel, and L. Boasson. *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*, pages 111–174. Springer-Verlag New York, Inc., 1997.

[5] B. v. Braunmühl and R. Verbeek. Input-driven Languages are Recognized in log n Space. In *FCT*, volume 158 of *LNCS*, pages 40–51. Springer, 1983.

[6] A. L. Buchsbaum, R. Giancarlo, and J. Westbrook. On the Determinization of Weighted Finite Automata. *SIAM J. Comput.*, 30(5):1502–1531, 2000.

[7] M. Caralp, P.-A. Reynier, and J.-M. Talbot. Visibly Pushdown Automata with Multiplicities: Finiteness and K-Boundedness. In *DLT*, volume 7410 of *LNCS*, pages 226–238. Springer, 2012.

[8] C. Choffrut. Une Caractérisation des Fonctions Séquentielles et des Fonctions Sous-Séquentielles en tant que Relations Rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977.

[9] R. De Souza. *Étude Structurelle des Transducteurs de Norme Bornée*. PhD thesis, ENST, France, 2008.

[10] E. Filiot, O. Gauwin, P.-A. Reynier, and F. Servais. Streamability of Nested Word Transductions. In *FSTTCS*, volume 13 of *LIPIcs*, pages 312–324. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[11] E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. Properties of Visibly Pushdown Transducers. In *MFCS'10*, volume 6281 of *LNCS*, pages 355–367. Springer, 2010.

[12] D. Girault-Beauquier. Some Results About Finite and Infinite Behaviours of a Pushdown Automaton. In *Automata, Languages and Prog.*, volume 172 of *LNCS*, pages 187–195. Springer, 1984.

[13] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Adison-Wesley, 1979.

[14] A. Mandel and I. Simon. On Finite Semigroups of Matrices. *Theor. Comput. Sci.*, 5(2):101–111, 1977.