Trimming Visibly Pushdown Automata

Mathieu Caralp, Pierre-Alain Reynier, and 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.

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 that 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 [13,9]). 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 [5]. Both of these results have been extended to visibly

[1] These automata were first introduced in [4] as "input-driven automata".

pushdown automata and transducers in [6] and [11] 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 each state, the regular language of stack contents that are both reachable and co-reachable, and use 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 that do only recognize 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. As we consider well-nested VPA, one can consider the "dual" of the automaton (reads the word from right to left), and apply the reduction procedure on it, yielding 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.

Organization of the paper In Section 2 we introduce useful 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. Due to lack of space, some proofs are omitted but can be found in [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.

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

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 (fcns for short). 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 fcns 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 fcns 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

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 $\varepsilon \in \Sigma^*_{\mathsf{wn}}$, $\Sigma_\iota \subseteq \Sigma^*_{\mathsf{wn}}$ and for all $c \in \Sigma_c$, all $r \in \Sigma_r$, all $u, v \in \Sigma^*_{\mathsf{wn}}$, $cur \in \Sigma^*_{\mathsf{wn}}$ and $uv \in \Sigma^*_{\mathsf{wn}}$.

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 $|u|$.

*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 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, $F \subseteq Q$ the set of final states, Γ is a finite stack alphabet, $\delta = \delta_c \uplus \delta_r \uplus \delta_r^\perp \uplus \delta_\iota$ the (finite) transition relation, 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 $t = (q, a, x, q')$ from δ_c, δ_r or $\delta_r^\perp$ or $t = (q, a, q')$ 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 a.

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

Definition 2. *A run of A on a word $w = a_1 \ldots a_l \in \Sigma^*$ over a sequence of transitions $(t_k)_{1 \le k \le l}$ from a configuration (q, σ) to a configuration (q', σ') is a finite sequence of symbols and configurations $\rho = (q_0, \sigma_0)\Pi_{k=1}^l (a_k(q_k, \sigma_k))$ such that $(q, \sigma) = (q_0, \sigma_0)$, $(q', \sigma') = (q_l, \sigma_l)$, and, for each $1 \le k \le l$, there exists $\gamma_k \in \Gamma$ such that either:*

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

We denote by $\mathsf{Run}_w(A)$ the set of runs of A over the word w. Note that a run for the empty word is simply any configuration. We write $(q, \sigma) \xrightarrow{w} (q', \sigma')$ when there exists a run over w from (q, σ) to (q', σ'). We may omit the superscript w when irrelevant.

Given two runs $\rho_i = (q_0^i, \sigma_0^i)\Pi_{k=1}^{\ell_i}(a_k^i(q_k^i, \sigma_k^i))$ for $i \in \{1, 2\}$, we can consider the concatenation $\rho_1\rho_2$ of these runs, provided that $(q_{\ell_1}^1, \sigma_{\ell_1}^1) = (q_0^2, \sigma_0^2)$, defined as $\rho_1\rho_2 = (q_0^1, \sigma_0^1)\Pi_{k=1}^{\ell_1}(a_k^1(q_k^1, \sigma_k^1))\Pi_{k=1}^{\ell_2}(a_k^2(q_k^2, \sigma_k^2))$,

Initial (resp. final) configurations are configurations of the form $(q, \bot)$, with $q \in I$ (resp. (q, σ) with $q \in F$). 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. The set of all accepting runs of A is denoted $\mathsf{ARun}(A)$. 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.

Definition 3. *A* VPA $A = (Q, I, F, \Gamma, \delta)$ *is*

– deterministic *if I is a singleton and for all q in Q, for all c in Σ_c, for all i in Σ_ι, for all r in Σ_r,*
  - *there exists at most one rule of the form (q, c, γ, q') in δ_c, of the form (q, i, q') in δ_ι and of the form $(q, r, \bot, q')$ in $\delta_r^{\bot}$,*
  - *for all γ in Γ, there exists at most one rule of the form (q, r, γ, q') in δ_r*
– co-deterministic *if F is a singleton and for all q' in Q, for all c in Σ_c, for all i in Σ_ι, for all r in Σ_r,*
  - *for all γ in Γ, there exists at most one rule of the form (q, c, γ, q') in δ_c.*
  - *there exists at most one rule of the form (q, r, γ, q') in δ_r, of the form (q, i, q') in δ_ι and of the form $(q, c, \bot, q')$ in $\delta_\iota^{\bot}$.*

(Co)-reduced and trimmed VPA A configuration (q, σ) is *reachable from a configuration* (q', σ') if there exists a word u in Σ^* such that $(q', \sigma') \xrightarrow{u} (q, \sigma)$.

We say that a configuration (q, σ) is *reachable* (resp. *co-reachable*) if there exists an initial (resp. a final) configuration κ such that (q, σ) is reachable from κ (resp. κ is reachable from (q, σ)).

Definition 4. *Let A be a* VPA. *Let us consider the three following conditions :*

(i) every reachable configuration is co-reachable.
(ii) for every configuration (p, σ) and every reachable and final configuration κ' of A such that κ' is reachable from (p, σ), then (p, σ) is reachable.
(iii) for every state q, there exists an accepting run going through a configuration (q, σ).

We say that the automaton A is reduced *(resp.* co-reduced*) if it fulfills conditions (i) and (iii) (resp. (ii) and (iii)),*trimmed *if it is both reduced and co-reduced, and* weakly reduced *if it fulfills condition (i).*

Observe that condition (ii) 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 VPA accepting only well-nested words, this condition is equivalent to the simpler one.

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

2.1 Well-nested VPA (wnVPA)

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. Well-nested VPA enjoy some good properties; we describe some of them here.

Remark 1. If A is well-nested then its final configurations (f, σ) (with f a final state) are reachable only if $\sigma = \bot$.

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

Proposition 1. *Let A be a* wnVPA *satisfying condition (iii), then A is* co-reduced *iff every configuration that can reach some configuration $(f, \bot)$ with $f \in F$ is reachable.*

Proof. Let $f \in F$. By condition (iii), there exists $\sigma \in \Gamma^*$ such that configuration (f, σ) appears on some accepting run ρ, and by Remark 1, $\sigma = \bot$. Thus the set of reachable final configurations of A is equal to $\{(f, \bot) \mid f \in F\}$. □

Let $\Sigma = (\Sigma_c, \Sigma_r, \Sigma_\iota)$. The dual alphabet of Σ (denoted $\mathsf{dual}(\Sigma)$) is the alphabet $(\Sigma_r, \Sigma_c, \Sigma_\iota)$. Roughly speaking call and return symbols are switched.

Let w be a word in Σ^*_{wn}. We define $\mathsf{dual}(w)$ as the mirror image of w. We naturally extend this notion to languages L such that $L \subseteq \Sigma^*_{\mathsf{wn}}$.

Definition 5. *Let $A = (Q, I, F, \Gamma, \delta)$ be a* wnVPA *over some alphabet Σ. Its dual* VPA *denoted $\mathsf{dual}(A) = (Q', I', F', \Gamma', \delta')$ over the alphabet $\mathsf{dual}(\Sigma)$ is given by $Q' = Q$, $I' = F$, $F' = I$, $\Gamma' = \Gamma$, and $\delta'_c = \{(q, r, \gamma, q') \mid (q', r, \gamma, q) \in \delta_r\}$, $\delta'_r = \{(q, c, \gamma, q') \mid (q', c, \gamma, q) \in \delta_c\}$, $\delta'_\iota = \{(q, i, q') \mid (q', i, q) \in \delta_\iota\}$, and $\delta'^{\bot}_r = \emptyset$.*

It is easy to prove that :

Proposition 2. *Let A be a* wnVPA. *Then*

- *For all $w \in \Sigma^*_{\mathsf{wn}}$, there exists a bijection between $\mathsf{Run}_w(A)$ and $\mathsf{Run}_{\mathsf{dual}(w)}(\mathsf{dual}(A))$ which induces a bijection between $\mathsf{ARun}_w(A)$ and $\mathsf{ARun}_{\mathsf{dual}(w)}(\mathsf{dual}(A))$.*
- *A is reduced iff $\mathsf{dual}(A)$ is co-reduced.*
- *A is deterministic iff $\mathsf{dual}(A)$ is co-deterministic.*

[3] For any state q, one can build in polynomial time from A a word automaton over the alphabet Γ whose language is empty iff no configuration of the form (q, σ) is reachable.

3 Trimming well-nested VPA

Let A be a wnVPA on a structured alphabet Σ. We present the construction of the VPA $\mathsf{trim}_{\mathsf{wn}}(A)$, which recognizes the same language, and in addition is trimmed. First we define the reduced VPA $\mathsf{reduce}(A)$ which is equivalent to A.

3.1 Construction of wreduce(A) and reduce(A)

Consider a wnVPA $A = (Q, I, F, \Gamma, \delta)$. We describe the construction of the VPA $\mathsf{wreduce}(A)$, which is weakly reduced. The VPA $\mathsf{reduce}(A)$ is then obtained by removing useless states of $\mathsf{wreduce}(A)$, as explained in the previous section.

We first consider the set $\mathsf{WN} = \{(p,q) \in Q \times Q \mid \exists (p,\bot) \to (q,\bot) \in \mathsf{Run}(A)\}$. 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 6. *For any* wnVPA $A = (Q, I, F, \Gamma, \delta)$*, we define the* wnVPA $\mathsf{wreduce}(A)$ *as* $(Q', I', F', \Gamma', \delta')$ *where* $Q' = \mathsf{WN}$*,* $I' = \mathsf{WN} \cap (I \times F)$*,* $F' = \{(f,f) \mid f \in F\}$*,* $\Gamma' = \Gamma \times Q$*, and* δ' *is defined by its restrictions on call, return* [4] *and internal symbols respectively (namely* δ'_c*,* δ'_r *and* δ'_ι*):*

- $\delta'_c = \{((p,q),c,(\gamma,q),(p',q')) \mid (p,q),(p',q') \in Q', (p,c,\gamma,p') \in \delta_c,$ $\exists r \in \Sigma_r, \exists s \in Q, (q',r,\gamma,s) \in \delta_r \text{ and } (s,q) \in Q'\}$
- $\delta'_r = \{((q',q'),r,(\gamma,q),(p,q)) \mid (q',q'),(p,q) \in Q', (q',r,\gamma,p) \in \delta_r\}$
- $\delta'_\iota = \{((p,q),a,(p',q)) \mid (p,q),(p',q) \in Q', (p,a,p') \in \delta_\iota\}$

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 state that the run should reach to pop the symbol on 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 q. This condition is depicted on Figure 1, and we give an example of the construction in Figure 2.

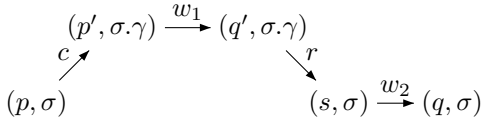

Fig. 1. Construction of call transitions.

3.2 Properties of wreduce(A) and reduce(A)

We consider the projection π from configurations of $A_{\mathsf{wred}} = \mathsf{wreduce}(A)$ to configurations of A obtained by considering the first component of states, as well as the first component of stack symbols. By definition of A_{wred}, each transition of A_{wred} is associated with a unique transition of A, we also denote by π this mapping. One can easily prove (see [7]), that π maps runs of A_{wred} onto runs of A.

The constructions wreduce and reduce have the following properties:

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

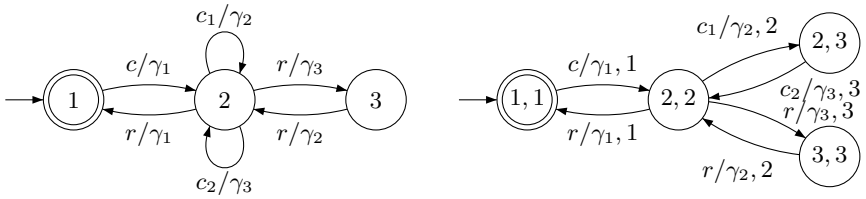

Fig. 2. On the left a VPA A, on the right $\mathsf{reduce}(A)$. There exists an initialized run of A over cc_1c_1 which cannot be completed into an accepting run. This run is no longer present in $\mathsf{reduce}(A)$.

Theorem 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) *there exist bijections between $\mathsf{ARun}(A_{\mathsf{wred}})$ and $\mathsf{ARun}(A)$, and $\mathsf{ARun}(A_{\mathsf{red}})$ and $\mathsf{ARun}(A)$ 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,*
(3) *if A is co-reduced, then A_{red} is co-reduced,*
(4) *if A is co-deterministic, then A_{wred} and A_{red} are co-deterministic.*

Proof (Sketch). As explained above, the mapping π induces a mapping from runs of A_{wred} to runs of A. It is easy to verify that this mapping preserves the property of being accepting. In addition, one can prove by induction on the structure of the underlying word that it is both injective and surjective when restricted to the set $\mathsf{ARun}(A_{\mathsf{wred}})$. This yields a bijection between $\mathsf{ARun}(A_{\mathsf{wred}})$ and $\mathsf{ARun}(A)$. The bijection between $\mathsf{ARun}(A_{\mathsf{wred}})$ and $\mathsf{ARun}(A_{\mathsf{red}})$ is trivial as by definition A_{red} only differs from A_{wred} by states that do not appear in accepting runs.

The proof that A_{wred} is weakly reduced proceeds by induction on the size of the stack of the reachable configuration (p, σ) under consideration. It is then an immediate consequence that A_{red} is reduced.

By Proposition 1, to prove Property (3), we only have to prove that every configuration of A_{wred} co-reachable from a final configuration with an empty stack is reachable. This is done by induction on the size of the stack of this configuration. Last, the proof of Property (4) is done by an inspection of the transitions of A_{wred}. □

3.3 From reduced to trimmed

A construction for the co-reduction Given a wnVPA, we can perform the following composition of constructions: $\mathsf{coreduce} = \mathsf{dual} \circ \mathsf{reduce} \circ \mathsf{dual}$. As a consequence of Proposition 2 and Theorem 1, the construction coreduce yields an equivalent wnVPA which is co-reduced.

Trimming We define the construction $\mathsf{trim}_{\mathsf{wn}}$ as $\mathsf{trim}_{\mathsf{wn}} = \mathsf{coreduce} \circ \mathsf{reduce}$. Property (3) of Theorem 1 entails that the construction coreduce preserves the reduction, and we thus obtain the following result:

Theorem 2. *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 there exists a bijection between $\mathsf{ARun}(A)$ and $\mathsf{ARun}(A_{\mathsf{trim}})$, and thus in particular $L(A) = L(A_{\mathsf{trim}})$.*

Remark 2. The construction of co-reduction could also be presented explicitely. It would consist in adding an extra component into states and stack symbols, as done for the construction reduce, but representing the state reached when the top symbol of the stack was pushed. The same approach allows to present explicitely the construction $\mathsf{trim}_{\mathsf{wn}}$.

4 General case

In order to trim a VPA A over some alphabet Σ, we will first build a wnVPA $\mathsf{extend}(A)$ over a new alphabet. In a second step, we trim the VPA $\mathsf{extend}(A)$ using the procedure described in the previous section for wnVPA. Last, we construct from the resulting wnVPA a VPA, which recognizes the language $L(A)$, and which is still trimmed. It is far from being trivial to propose procedures for transforming a VPA into wnVPA, and back, which are compatible with the notion of being trimmed. In addition, we will address the property of determinism in the next section, and our constructions should also be compatible with that issue.

4.1 Constructing well-nested words from arbitrary words

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 $\Sigma_\iota^{\mathsf{ext}} = \Sigma_\iota \uplus \{i_r \mid r \in \Sigma_r\}$, where $\bar{r}$ and $\{i_r \mid r \in \Sigma_r\}$ are fresh symbols.

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

$$\mathsf{ext}(aw) = a \cdot \mathsf{ext}(w), \qquad \mathsf{ext}(rw) = i_r \cdot \mathsf{ext}(w),$$

$$\mathsf{ext}(cw) = \begin{cases} cw_1 r \cdot \mathsf{ext}(w_2) & \text{if } \exists w_1 \in \Sigma_{\mathsf{wn}}^* \text{ such that } w = cw_1 r w_2, \\ c \cdot \mathsf{ext}(w) & \text{otherwise.} \end{cases}$$

For example, $\mathsf{ext}(rccar) = i_r ccar\bar{r}$ with $c \in \Sigma_c$, $r \in \Sigma_r$ and $a \in \Sigma_\iota$. The mapping ext replaces every return r on empty stack by the internal symbol i_r, and adds a suffix of the form $\bar{r}^*$ in order to match every unmatched call. As a consequence, $\mathsf{ext}(w)$ is a well-nested word over the alphabet Σ^{ext}. We extend the function ext to languages in the obvious way.

4.2 Reduction to wnVPA

From VPA *to* wnVPA ... We present the construction extend which turns a VPA over Σ into a wnVPA over Σ^{ext}. Intuitively, when firing a call transition, the automaton nondeterministically guesses whether this call will be matched or not. Then, if a call is considered as not matching, the automaton completes it by using a transition over $\bar{r}$ at the end of the run. This is done by adding a suffix to the VPA which reads words from $\bar{r}^+$. Moreover the construction replaces the returns on empty stack by internals, this is done by memorizing in the state the fact that the current stack is empty or not.

We let T denote the set of symbols $\{\bot, \top, \circ\}$. Intuitively, $\bot$ means that the stack is empty, $\circ$ means that the stack contains only useless symbols (*i.e.* symbols associated with calls that will be unmatched), and $\top$ means that the stack contains some useless symbols (possibly none) plus symbols which will be popped. Furthermore we consider a fresh final state denoted by $\bar{f}$. This state is used to pop all the useless symbols.

Definition 7. *Let $A = (Q, I, F, \Gamma, \delta)$ be a* VPA *over an alphabet Σ, we define the* VPA $\mathsf{extend}(A) = (Q', I', F', \Gamma', \delta')$ *over the alphabet Σ^{ext}, where $Q' = (Q \times T) \cup (\{\bar{f}\} \times \{\bot, \circ\})$, $I' = I \times \{\bot\}$, $F' = (F \cup \{\bar{f}\}) \times \{\bot\}$, $\Gamma' = \Gamma \times T$, and δ' is given by:*

$\delta'_c = \{((p,t), c, (\gamma,t), (q,x)) \mid (p,c,\gamma,q) \in \delta_c$ and either $x = \top$ or $(x = \circ \wedge t \neq \top)\}$

$\delta'_r = \{((p,\top), r, (\gamma,t), (q,t)) \mid (p,r,\gamma,q) \in \delta_r\} \cup$
$\{((p,\circ), \bar{r}, (\gamma,t), (\bar{f},t)) \mid p \in F \cup \{\bar{f}\}, \gamma \in \Gamma, t \neq \top\}$

$\delta'_\iota = \{((p,t), a, (q,t)) \mid (p,a,q) \in \delta_\iota\} \cup \{((p,\bot), i_r, (q,\bot)) \mid (p,r,\bot,q) \in \delta_r\}$

Theorem 3. *Let A be a* VPA*. Then for all words $w \in \Sigma^*$, there exists a bijection between* $\mathsf{ARun}_w(A)$ *and* $\mathsf{ARun}_{\mathsf{ext}(w)}(\mathsf{extend}(A))$.

... and back. We present now the construction allowing to go from a wnVPA on Σ^{ext} to the "original" VPA on Σ. It is not always possible to find such a VPA, we thus introduce the property of being retractable.

Definition 8. *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:*

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

$\mathsf{border}(A) = \{p \notin \mathsf{trap}(A) \mid \exists t \in \delta$ *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.*

Definition 9. *Let Σ be an alphabet and $A = (Q, I, F, \Gamma, \delta)$ be a* VPA *over the alphabet Σ^{ext}. Then A is said to be* retractable *if :*

- (i) *There exists a* VPA *B over Σ such that $L(A) = \mathsf{ext}(L(B))$,*
- (ii) *We have* $\mathsf{trap}(A) \cap I = \emptyset$
- (iii) *For all transitions t in δ such that* $\mathsf{source}(t) \notin \mathsf{trap}(A)$*, if* $\mathsf{letter}(t) = \bar{r}$*, then* $\mathsf{target}(t) \in \mathsf{trap}(A)$*, otherwise* $\mathsf{target}(t) \notin \mathsf{trap}(A)$.
- (iv) *For all transitions t in δ such that* $\mathsf{source}(t) \in \mathsf{trap}(A)$*, then* $\mathsf{letter}(t) = \bar{r}$*, and* $\mathsf{target}(t) \in \mathsf{trap}(A)$.
- (v) *For each initialized run of A which ends in a border state there exists a unique run ρ' over $\bar{r}^+$ such that $\rho\rho'$ is an accepting run.*

Intuitively, a VPA which is retractable has two components: the first (before entering $\mathsf{trap}(A)$) can read words over $(\Sigma^{\mathsf{ext}} \setminus \{\bar{r}\})^*$ whereas the second reads words of the form $\bar{r}^+$. Note that the only way to go from a state not in $\mathsf{trap}(A)$ to a state in $\mathsf{trap}(A)$ is to use a transition which leaves a border state by $\bar{r}$. We give an example of these properties in Figure 3. We naturally have:

Lemma 1. *Let A be a* VPA*, then* $\mathsf{extend}(A)$ *is retractable.*

We now define the converse of the function extend, named retract:

Definition 10. *Let $A = (Q, I, F, \Gamma, \delta)$ be a retractable* VPA *over the alphabet Σ^{ext}, we define the* VPA $\mathsf{retract}(A) = (Q', I', F', \Gamma, \delta')$ *over the alphabet Σ by $Q' = Q \setminus \mathsf{trap}(A)$, $I' = I$, $F' = (F \setminus \mathsf{trap}(A)) \cup \mathsf{border}(A)$, and the set of transition rules $\delta' = \delta'_c \uplus \delta'_r \uplus \delta'^{\bot}_r \uplus \delta'_\iota$ is defined by: $\delta'_c = \delta_c$, $\delta'_r = \{t \in \delta_r \mid \mathsf{letter}(t) \neq \bar{r}\}$, $\delta'^{\bot}_r = \{(p,r,\bot,q) \mid (p,i_r,q) \in \delta_\iota\}$, and $\delta'_\iota = \{t \in \delta_\iota \mid \mathsf{letter}(t) \in \Sigma_\iota\}$.*

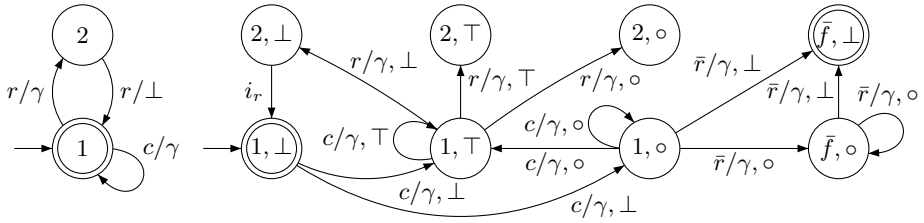

Fig. 3. At left, the VPA A with $L(A) = (crr)^*c^*$, at right the VPA $A_{\mathsf{ext}} = \mathsf{extend}(A)$ with $L(A_{\mathsf{ext}}) = \{(cri_r)^*c^k\bar{r}^k \mid k \in \mathbb{N}\}$. $\mathsf{border}(A_{\mathsf{ext}}) = \{(1,\circ)\}$, $\mathsf{trap}(A_{\mathsf{ext}}) = \{(\bar{f},\circ),(\bar{f},\bot)\}$.

This construction is very simple, it replaces all the internal transitions over i_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. We list below important properties of retract:

Theorem 4. *Let A be a retractable* VPA *on Σ^{ext}, we have:*

(i) *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 $L(A) = \mathsf{ext}(L(\mathsf{retract}(A)))$,*
(ii) *if A is trimmed, then so is $\mathsf{retract}(A)$.*

In addition, the retractibility is preserved by the trimming procedure described in the previous section for well-nested VPA:

Theorem 5. *Let A be a* VPA*, then the* VPA $\mathsf{trim}_{\mathsf{wn}}(\mathsf{extend}(A))$ *is retractable.*

4.3 Trimming VPA

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

Theorem 6. *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:*

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

Proof. First, by Theorem 5, the VPA $\mathsf{trim}_{\mathsf{wn}} \circ \mathsf{extend}(A)$ is retractable, and thus $\mathsf{trim}(A)$ is well-defined. Then, the first property follows from the fact that such bijections exist for the constructions extend, $\mathsf{trim}_{\mathsf{wn}}$ and retract. The second property is a consequence of Theorems 2 and 4.(ii). □

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.* that recognizes 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 a trivial corollary of the two previous results, as the different constructions can not be directly combined.

Due to lack of space, we do not show the determinization procedure for VPA, it can however be found in [2]. Note that its complexity is $O(2^{n^2})$, where n denotes the number of states of the input VPA. We denote by $\mathsf{determinize}$ the procedure obtained by applying the construction of [2], followed by the removal of useless states (according to property (iii) of Definition 4), which can be performed in polynomial time.

5.1 Determinization preserves reduction and retractability

We start by proving that the construction $\mathsf{determinize}$ preserves the properties of being weakly reduced and of being retractable. In the sequel, we let A be a VPA and we let $A_{\mathsf{det}} = \mathsf{determinize}(A)$. The following results are proved in [2]:

Theorem 7. *A_{det} is a deterministic* VPA*, and $L(A) = L(A_{\mathsf{det}})$.*

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

As a corollary we prove in [7]:

Proposition 3. *If A is weakly reduced (resp. retractable), then A_{det} is weakly reduced (resp. retractable).*

5.2 Construction of a deterministic trimmed VPA

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

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

We claim that this composition allows one to build an equivalent VPA that is both deterministic and trimmed, as stated in the following theorem:

Theorem 8. *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 first let $A_1 = \mathsf{reduce} \circ \mathsf{extend}(A)$. By Theorems 1 and 3, A_1 is weakly reduced, and recognizes the language $\mathsf{ext}(L(A))$. In addition, we prove in [7] that A_1 is retractable. Consider now $A_2 = \mathsf{determinize}(A_1)$. We have $L(A_2) = L(A_1)$ and, by Proposition 3 and as the construction $\mathsf{determinize}$ includes the removal of useless states, the VPA A_2 is deterministic, reduced and retractable. Consider now $A_3 = \mathsf{coreduce}(A_2)$. By Theorem 1, properties (3) and (4) and by Proposition 2, the construction $\mathsf{coreduce}$ preserves the properties of being reduced, and of being deterministic. In addition, we prove in [7] that this construction also preserves the property of being retractable. We thus conclude that A_3 is retractable, trimmed, deterministic, and satisfies $L(A_3) = L(A_1) = \mathsf{ext}(L(A))$. To conclude, it remains to observe that the construction $\mathsf{retract}$ preserves the determinism, and to use Theorem 4. □

6 Conclusion

We introduced a series of constructions to trim a VPA. For each of these constructions, there exist projections of transitions of the obtained VPA onto those of the original one, which yield bijections between the accepting runs of the two VPA's. As a corollary, our constructions can be lifted to weighted VPA (VPA equipped with a labelling function of transitions, such as visibly pushdown transducers).

Our trimming procedure doesn't preserve the deterministic nature of the input VPA. We have however presented an alternative method to simultaneously trim and determinize a VPA, the complexity of this method being exponential. One can wonder whether a deterministic VPA can be trimmed with a polynomial time complexity, preserving its determinsitic nature. The answer to this question is negative, and can be obtained using the family of languages $L_N = \{(c_1 + c_2)^k c_1 (c_1 + c_2)^N r^{N+k+1} \mid k \in \mathbb{N}\}$, with $N \in \mathbb{N}$, as a counter-example.

As future work, we plan to study the complexity of determining whether a VPA is trimmed. Another perspective is to implement our constructions in libraries for nested words such as [10].

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. 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.
5. A. L. Buchsbaum, R. Giancarlo, and J. Westbrook. On the Determinization of Weighted Finite Automata. *SIAM J. Comput.*, 30(5):1502–1531, 2000.
6. 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.
7. M. Caralp, P.-A. Reynier, and J.-M. Talbot. A Polynomial Procedure for Trimming Visibly Pushdown Automata. Technical Report hal-00606778, HAL, CNRS, France, 2013.
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. Driscoll, A. V. Thakur, and T. W. Reps. OpenNWA: A Nested-Word Automaton Library. In *CAV*, volume 7358 of *LNCS*, pages 665–671. Springer, 2012.
11. 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.
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. A. Mandel and I. Simon. On Finite Semigroups of Matrices. *Theor. Comput. Sci.*, 5(2):101–111, 1977.