Expressiveness of Visibly Pushdown Transducers

Mathieu Caralp$^3$, Emmanuel Filiot$^1$, Pierre-Alain Reynier$^3$, Frédéric Servais$^2$
Jean-Marc Talbot$^3$

$^1$Université Libre de Bruxelles

$^2$University of Hasselt

$^3$Université de Aix-Marseille
Visibly Pushdown Automata (VPAs) [Alur, Madhusudan, 04]

VPAs = Pushdown Automata on \textit{structured} alphabet \( \Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_i \):

- push \textbf{one} stack symbol on \textit{call} symbols \( \Sigma_c \)
- pop \textbf{one} stack symbol on \textit{return} symbols \( \Sigma_r \)
- don’t touch the stack on \textit{internal} symbols \( \Sigma_i \)
- in this talk, accept on empty stack and final state
Visibly Pushdown Automata (VPAs) [Alur, Madhusudan, 04]

VPAs = Pushdown Automata on \( \text{structured} \) alphabet \( \Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i \):

- push \textbf{one} stack symbol on \textbf{call} symbols \( \Sigma_c \)
- pop \textbf{one} stack symbol on \textbf{return} symbols \( \Sigma_r \)
- don’t touch the stack on \textbf{internal} symbols \( \Sigma_i \)
- in this talk, accept on empty stack and final state

\[ L(A) = \left\{ c^n \cdot r_1 \cdot (r_1 + r_2)^{n-1} \mid n > 0 \right\}. \]
Nested Words vs Trees

- acceptance on empty stack $\rightarrow$ VPA recognize well-nested words

\[
c c c r_1 r_2 c r_1 r_1
\]
Nested Words vs Trees

- acceptance on empty stack $\rightarrow$ VPA recognize well-nested words

$$c \ c \ c \ r_1 \ r_2 \ c \ r_1 \ r_1$$

- well-nested words over $\Sigma_c \uplus \Sigma_r \equiv$ linearization of unranked trees on $\Sigma_c \times \Sigma_r$:
Nested Words vs Trees

- acceptance on empty stack $\rightarrow$ VPA recognize well-nested words

$$c c c r_1 r_2 c r_1 r_1$$

- well-nested words over $\Sigma_c \uplus \Sigma_r \equiv$ linearization of unranked trees on $\Sigma_c \times \Sigma_r$:

![Diagram of unranked tree]

- and conversely ... $(\forall f, \text{ add } c_f \in \Sigma_c \text{ and } r_f \in \Sigma_r)$

![Diagram of word]

$$\rightarrow \quad c_f \ a \ c_f \ b \ a \ r_f \ r_f$$

- more generally, correspondence with hedges
VPA vs Tree Automata

- VPA $\equiv$ (unranked) tree automata
- Inherit all the good properties of tree automata
- Left-to-right evaluation: XML streaming applications (validation, queries)
- See http://www.cs.uiuc.edu/~madhu/vpa/
Visibly Pushdown Transducers (VPTs)

- VPTs = VPAs with output words on transitions
- no structure on the output $\rightarrow$ well-nested word to word transducers

$$T(cca_1r_1r_2) = baa \text{ (functional)}$$

- non-determinism $\rightarrow$ relations
Decision Problems for VPTs

- functionality is decidable in \( \text{PTime} \)
  - based on the morphism equivalence problem on context-free grammars, decidable in \( \text{PTime} \) [Plandowski 94]

\[
\forall u \in L(G), \ \phi_1(u) = \phi_2(u) ~?
\]
Decision Problems for VPTs

- functionality is decidable in $\text{PTime}$
  - based on the morphism equivalence problem on context-free grammars, decidable in $\text{PTime}$ [Plandowski 94]

\[
\forall u \in L(G), \phi_1(u) = \phi_2(u) ?
\]

- equivalence of functional VPTs is $\text{ExpTime}$-c ($\text{PTime}$ if total)
Decision Problems for VPTs

- functionality is decidable in \( \text{PTime} \)
  - based on the morphism equivalence problem on context-free grammars, decidable in \( \text{PTime} \) [Plandowski 94]
    \[
    \forall u \in L(G), \; \phi_1(u) = \phi_2(u) ?
    \]

- equivalence of functional VPTs is \( \text{ExpTime} \)-c (\( \text{PTime} \) if total)

- \( k \)-valuedness is decidable in \( \text{NP} \) (for a fixed \( k \))
  - based on the emptiness of bounded reversal pushdown counter machines [Ibarra 78]
  - emptiness is in \( \text{NP} \) [.,Raskin,Reynier,Servais,Talbot 10]
Decision Problems for VPTs

- functionality is decidable in $\mathsf{PTime}$
  - based on the morphism equivalence problem on context-free grammars, decidable in $\mathsf{PTime}$ [Plandowski 94]

$$\forall u \in L(G), \phi_1(u) = \phi_2(u) ?$$

- equivalence of functional VPTs is $\mathsf{ExpTime}$-c ($\mathsf{PTime}$ if total)
- $k$-valuedness is decidable in $\mathsf{NP}$ (for a fixed $k$)
  - based on the emptiness of bounded reversal pushdown counter machines [Ibarra 78]
  - emptiness is in $\mathsf{NP}$ [.,Raskin,Reynier,Servais,Talbot 10]

- (input) deterministic VPTs $< \text{ functional VPTs}$
- however functional VPTs $= \text{ deterministic VPTs + look-ahead} = \text{ unambiguous VPTs}$ [.,Servais,12]
Typechecking Problem

Definition

**Input:** $I, O$: VPAs, $T$:VPT

**Output:** $T(I) \subseteq O$?
Typechecking Problem

**Definition**

**Input:** $I, O$: VPAs, $T$: VPT

**Output:** $T(I) \subseteq O$?

**Result**

The typechecking problem is **undecidable**.

- stack of $O$ not synchronized with stack of $T$
- for the same reason, VPTs are not closed under composition
- restriction to ensure synchronization: **well-nested VPTs**
**Definition**

For all stack symbols $\gamma$ and all transitions $\textcolor{red}{c\mid w_1, +\gamma}$ and $\textcolor{red}{r\mid w_2, -\gamma}$, we must have $w_1 w_2$ is well-nested.
**Well-Nested VPTs**

### Definition

For all stack symbols $\gamma$ and all transitions $c|w_1, +\gamma \rightarrow$ and $r|w_2, -\gamma \rightarrow$, we must have $w_1w_2$ is well-nested.

- well-nested VPTs produce well-nested words
- closed under composition
- typechecking is $\text{ExpTime}$-c
Well-Nested VPTs

**Definition**

For all stack symbols $\gamma$ and all transitions $\frac{c|w_1, +\gamma}{\rightarrow}$ and $\frac{r|w_2, -\gamma}{\rightarrow}$, we must have $w_1 w_2$ is well-nested.

- well-nested VPTs produce well-nested words
- closed under composition
- typechecking is $\text{ExpTime}$-c
- well-nested VPTs are unranked tree to unranked tree transducers
- how do they compare to other tree transducer models?
Remarks and Example

Remarks

- They cannot copy subtrees: \( t \mapsto f(t, t) \)
- They cannot swap subtrees (order-preserving): \( f(t_1, t_2) \mapsto f(t_2, t_1) \)
- They can express all edit operations on trees (insert / delete / rename)
- Operations can be conditional (closure by regular look-ahead)
- E.g., \( h_1 g(h_2) h_3 \mapsto h_1 h_2 h_3 \)
Remarks and Example

Remarks

- they cannot copy subtrees $t \mapsto f(t, t)$
- they cannot swap subtrees (order-preserving) $f(t_1, t_2) \mapsto f(t_2, t_1)$
- they can express all edit operations on trees (insert / delete / rename)
- operations can be conditional (closure by regular look-ahead)
- e.g. $h_1 g(h_2) h_3 \mapsto h_1 h_2 h_3$

- assume $\Sigma_c = \{c, c_g\}$, $\Sigma_r = \{r, r_g\}$, $\Sigma_i = \{\#\}$

```
c | c, +γ  c | c, +γ  c | c, +γ
```
```
q0  c_g | ε  q1  r_g | ε  q2
```
```
r | r, −γ  r | r, −γ  r | r, −γ
```
First-Child Next-Sibling Encoding

- **ranked** tree transducers can define unranked tree transformations if they run on a binary encoding of trees
- first-child next-sibling encoding:

```
  a
  /|
 /  \
 c   d
|   /
|  /  
| /    
|/  
 b     b
```

```
  a
  / Convers
 /  \
/   
/    
/     
/      
#      
#      
#      
#      
#      
#      
```
Well-nested VPTs vs (Classical) Tree Transducers

- top-down tree transducers: $q(f(x_1, \ldots, x_n)) \rightarrow C[q_1(x_{i_1}), \ldots, q_p(x_{i_p})]$
Well-nested VPTs vs (Classical) Tree Transducers

- top-down tree transducers: $q(f(x_1, \ldots , x_n)) \rightarrow C[q_1(x_{i_1}), \ldots , q_p(x_{i_p})]$
- TT cannot express $h_1g(h_2)h_3 \mapsto h_1h_2h_3$ (on fncs encoding)
Well-nested VPTs vs (Classical) Tree Transducers

- top-down tree transducers: \( q(f(x_1, \ldots, x_n)) \rightarrow C[q_1(x_{i_1}), \ldots, q_p(x_{i_p})] \)
- TT cannot express \( h_1 g(h_2) h_3 \mapsto h_1 h_2 h_3 \) (on fncs encoding)

- VPTs are incomparable to TT ... need auxiliary memory!
Well-nested VPTs vs (Classical) Tree Transducers

- top-down tree transducers: \( q(f(x_1, \ldots, x_n)) \rightarrow C[q_1(x_{i_1}), \ldots, q_p(x_{i_p})] \)
- TT cannot express \( h_1 g(h_2) h_3 \mapsto h_1 h_2 h_3 \) (on fnct encoding)

VPTs are incomparable to TT ... need auxiliary memory!

well-nested VPTs can be expressed by macro tree transducers
Well-nested VPTs vs (Classical) Tree Transducers

- top-down tree transducers: $q(f(x_1, \ldots, x_n)) \rightarrow C[q_1(x_{i_1}), \ldots, q_p(x_{i_p})]
- TT cannot express $h_1g(h_2)h_3 \rightarrow h_1h_2h_3$ (on fncts encoding)

VPTs are incomparable to TT ... need auxiliary memory!

- well-nested VPTs can be expressed by macro tree transducers
- VPTs are linear-size increase → captured by MSO-transducers (based on [Engelfriet, Maneth, 03])
- more generally, this holds for VPTs (not necessarily well-nested)
**Simpler Top-Down Hedge-to-String Model**

**Hedge-to-String Transducers (H2S)**

Rules of the form $q(\epsilon) \rightarrow \epsilon$ and

$$q(f(x_1) \cdot x_2) \rightarrow w_1 q(x_1) w_2 q(x_2) w_3$$

where $w_1, w_2, w_3$ are strings (possibly empty).
Simpler Top-Down Hedge-to-String Model

Hedge-to-String Transducers (H2S)

Rules of the form $q(\epsilon) \rightarrow \epsilon$ and

$$q(f(x_1) \cdot x_2) \rightarrow w_1 q(x_1)w_2 q(x_2)w_3$$

where $w_1, w_2, w_3$ are strings (possibly empty).

- **Example:** $q(f(x_1).x_2) \rightarrow c_f q(x_1)r_f q(x_2)$, $q(g(x_1).x_2) \rightarrow q(x_1)q(x_2)$. 
Simpler Top-Down Hedge-to-String Model

Hedge-to-String Transducers (H2S)

Rules of the form $q(\epsilon) \rightarrow \epsilon$ and

$$q(f(x_1) \cdot x_2) \rightarrow w_1 q(x_1) w_2 q(x_2) w_3$$

where $w_1, w_2, w_3$ are strings (possibly empty).

- **Example:** $q(f(x_1).x_2) \rightarrow c_f q(x_1) r_f q(x_2)$, $q(g(x_1).x_2) \rightarrow q(x_1) q(x_2)$.
- **Remark:** if $w_1 w_2 w_3$ is well-nested, they define hedge to hedge transformations (H2H)
Simpler Top-Down Hedge-to-String Model

Hedge-to-String Transducers (H2S)

Rules of the form \( q(\epsilon) \rightarrow \epsilon \) and

\[ q(f(x_1) \cdot x_2) \rightarrow w_1 q(x_1) w_2 q(x_2) w_3 \]

where \( w_1, w_2, w_3 \) are strings (possibly empty).

- **Example:** \( q(f(x_1) . x_2) \rightarrow c_f q(x_1) r_f q(x_2), q(g(x_1) . x_2) \rightarrow q(x_1) q(x_2) \).
- **Remark:** if \( w_1 w_2 w_3 \) is well-nested, they define **hedge to hedge** transformations (H2H)

Results

- well-nested VPTs \( \equiv \) H2H with \( w_3 = \epsilon \)
- VPTs \( \equiv \) H2S with \( w_3 = \epsilon \)
Running VPTs on fncs encoding

- the following H2S example is not VPT-definable:

\[ q(0) \rightarrow \epsilon \quad q'(0) \rightarrow \epsilon \quad q(f(x_1)x_2) \rightarrow q'(x_1)q(x_2)f \]

(accepts only strings and mirror them)

- e.g. \( abc \mapsto cba \)
Running VPTs on fncts encoding

- the following H2S example is not VPT-definable:

\[
q(0) \rightarrow \epsilon \quad q'(0) \rightarrow \epsilon \quad q(f(x_1)x_2) \rightarrow q'(x_1)q(x_2)f
\]

(accepts only strings and mirror them)

- **e.g.** \(abc \mapsto cba\)
- \(\text{fcns}(abc) = a(\#, b(\#, c(\#, \#)))\)
- \(\text{linearization of fncts}(abc) = c_a\#c_b\#c_c\##r_c r_b r_a\)
- can be easily transformed into \(cba\) by a VPT
Running VPTs on fncts encoding

- the following H2S example is not VPT-definable:

  \[ q(0) \rightarrow \epsilon \quad q'(0) \rightarrow \epsilon \quad q(f(x_1)x_2) \rightarrow q'(x_1)q(x_2)f \]

  (accepts only strings and mirror them)

- e.g. \( abc \mapsto cba \)
- \( \text{fncts}(abc) = a(\#, b(\#, c(\#, \#))) \)
- linearization of \( \text{fncts}(abc) = c_a\#c_b\#c_c\#\#r_c r_b r_a \)
- can be easily transformed into \( cba \) by a VPT

Results

- \( \text{VPT} \circ \text{fncts} \equiv \text{H2S} \)
- well-nested \( \text{VPT} \circ \text{fncts} \equiv \text{H2H} \)
Is VPT a good model?

VPTs...

- have good decidability properties (equivalence, $k$-valuedness, typechecking if well-nested)
- have good closure properties (under regular look-ahead, composition if well-nested)
- strictly extend finite string transducers
- but still, low expressive power compared to tree transducers (no copy, no swap)
Is VPT a good model?

VPTs...

- have good decidability properties (equivalence, \( k \)-valuedness, typechecking if well-nested)
- have good closure properties (under regular look-ahead, composition if well-nested)
- strictly extend finite string transducers
- but still, low expressive power compared to tree transducers (no copy, no swap)
- so what are they good for?
- streaming tree transformations!
Focus on functions

- How to transform an input nested word efficiently (minimize memory usage)?
- Left-to-right evaluation
- Just as VPA are good for streaming validation, VPT are good for streaming transformations.
Streamability Problem [F, Gauwin, Reynier, Servais, 11]

- focus on **functions**
- how to transform an input nested word efficiently (minimize memory usage) ?
- left-to-right evaluation
- just as VPA are good for streaming validation, VPT are good for streaming transformations
- evaluation by a deterministic VPT is easy
- for non-deterministic (functional) VPT, you may need to store the whole input !

**Recap:** deterministic VPT $<$ functional VPT
Streamability Problem [F, Gauwin, Reynier, Servais, 11]

Streaming evaluation: avoid the storage of the whole input

Fix a VPT $T$.

How much memory is needed to compute $T(u)$ from an input stream $u$?
Streamability Problem [F, Gauwin, Reynier, Servais, 11]

Streaming evaluation: avoid the storage of the whole input

Fix a VPT $T$.

How much memory is needed to compute $T(u)$ from an input stream $u$?

- constant memory
- cannot check well-nestedness!

memory usage
Streamability Problem [F, Gauwin, Reynier, Servais, 11]

Streaming evaluation: avoid the storage of the whole input

Fix a VPT $T$.

How much memory is needed to compute $T(u)$ from an input stream $u$?

- constant memory
- dependent in $\text{length}(u)$

- cannot check well-nestedness!
- not streamable!
Streamability Problem [F, Gauwin, Reynier, Servais, 11]

Streaming evaluation: avoid the storage of the whole input

Fix a VPT $T$.

How much memory is needed to compute $T(u)$ from an input stream $u$?

constant memory

$\text{height}(u)$

dependent in $\text{length}(u)$

cannot check well-nestedness!

memory usage

not streamable!
Streamability Problem [F, Gauwin, Reynier, Servais, 11]

Streaming evaluation: avoid the storage of the whole input

Fix a VPT $T$.

How much memory is needed to compute $T(u)$ from an input stream $u$?

- **constant memory**
- **height($u$)**
- **dependent in length($u$)**

...cannot check well-nestedness!

...memory usage

...not streamable!

**Streamability Problem**

Given a VPT $T$, decide if $T$ defines a transformation that can be evaluated with memory $O(f(height(u)))$?

Decidable in NP for VPTs
still, they are MSO-transduction that are (height-bounded memory) streamable:

\[ f_1(f_2(\ldots f_n(a)\ldots )) \mapsto f_n(f_{n-1}(\ldots f_1(a)\ldots )) \]

**Problem:** decide height-bounded memory for MSO/MTT-transformations.
Three Open Problems

- equivalence of $k$-valued VPTs
- determinizability problem
- is the class of constant memory streamable VPT-transformations decidable (with the assumption that inputs are well-nested words)