Automata Theory

Nested Word Automata

Christian Schilling

June 4th, 2012
Overview

Motivation and background

Nested words and their acceptors

Determinization proof

Conclusion
Overview

Motivation and background
Common languages
Visibly pushdown languages

Nested words and their acceptors

Determinization proof

Conclusion
Regular language

\[
\mathcal{L}_1 = \{c \ r\} 
\]
Nested Word Automata
Motivation and background
Common languages

Regular language

```
procedure foo()
{
    return;
}
```

$L_1 = \{ c \ r \}$

Diagram showing states $q_0$, $q_1$, and $q_2$ with transitions on symbols $c$ and $r$. 
(det.) Context-free language

1 \textbf{procedure} bar ()
2 {
3 \hspace{1em} \textbf{if} (*)
4 \hspace{1em} \textbf{call} bar () ;
5 \hspace{1em} \textbf{return} ;
6 }

\[ \mathcal{L}_2 = \{ c^n r^n \mid n > 0 \} \]
(det.) Context-free language

```
procedure bar()
{
    if (*)
        call bar();
    return;
}
```

\[ \mathcal{L}_2 = \{ c^n r^n | n > 0 \} \]
## Comparison

<table>
<thead>
<tr>
<th>😊</th>
<th>regular</th>
<th>context-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison of numbers</td>
<td>constants</td>
<td>two variables</td>
</tr>
<tr>
<td>closure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decidability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>determinize</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison

<table>
<thead>
<tr>
<th></th>
<th>regular</th>
<th>context-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison of numbers</td>
<td>constants</td>
<td>two variables</td>
</tr>
<tr>
<td>closure</td>
<td>all standard properties</td>
<td>not under intersection and complementation</td>
</tr>
<tr>
<td>decidability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>determinize</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison

<table>
<thead>
<tr>
<th></th>
<th>regular</th>
<th>context-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison of numbers</td>
<td>constants</td>
<td>two variables</td>
</tr>
<tr>
<td>closure</td>
<td>all standard properties</td>
<td>not under intersection and complementation</td>
</tr>
<tr>
<td>decidability</td>
<td>all standard problems</td>
<td>intersection, inclusion, equivalence undecidable</td>
</tr>
<tr>
<td>determinize</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Nested Word Automata

### Motivation and background

### Common languages

## Comparison

<table>
<thead>
<tr>
<th></th>
<th>regular</th>
<th>context-free</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>comparison of numbers</strong></td>
<td>constants</td>
<td>two variables</td>
</tr>
<tr>
<td><strong>closure</strong></td>
<td>all standard properties</td>
<td>not under intersection and complementation</td>
</tr>
<tr>
<td><strong>decidability</strong></td>
<td>all standard problems</td>
<td>intersection, inclusion, equivalence undecidable</td>
</tr>
<tr>
<td><strong>determinize</strong></td>
<td>powerset construction</td>
<td>not possible</td>
</tr>
</tbody>
</table>
### Comparison

<table>
<thead>
<tr>
<th></th>
<th>Regular</th>
<th>Context-Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison of numbers</td>
<td>constants</td>
<td>two variables</td>
</tr>
<tr>
<td>closure</td>
<td>all standard properties</td>
<td>not under intersection and complementation</td>
</tr>
<tr>
<td>decidability</td>
<td>all standard problems</td>
<td>intersection, inclusion, equivalence undecidable</td>
</tr>
<tr>
<td>determinize</td>
<td>powerset construction</td>
<td>not possible</td>
</tr>
</tbody>
</table>

**Question:** Is there some class of languages in between that is more expressive than regular languages, but keeps their nice properties?
Comparison

<table>
<thead>
<tr>
<th></th>
<th>regular</th>
<th>context-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>comparison of numbers</td>
<td>constants</td>
<td>two variables</td>
</tr>
<tr>
<td>closure</td>
<td>all standard properties</td>
<td>not under intersection and complementation</td>
</tr>
<tr>
<td>decidability</td>
<td>all standard problems</td>
<td>intersection, inclusion, equivalence undecidable</td>
</tr>
<tr>
<td>determinize</td>
<td>powerset construction</td>
<td>not possible</td>
</tr>
</tbody>
</table>

**Question**: Is there some class of languages in between that is more expressive than regular languages, but keeps their nice properties?

**Answer** (Alur & Madhusudan 2004): yes, at least in some sense
Visibly pushdown languages (VPLs)

A visibly pushdown language (VPL) is the language accepted by a visibly pushdown automaton (VPA).

A VPA $\mathcal{A} = \langle Q, q_0, Q_f, \Sigma, \Gamma, \bot, \delta \rangle$ is a deterministic PDA with special rules: Determined by the input symbol, only one symbol per push is allowed and reading the stack implies a pop.
Visibly pushdown languages (VPLs)

A *visibly pushdown language* (VPL) is the language accepted by a *visibly pushdown automaton* (VPA).

A VPA $A = \langle Q, q_0, Q_f, \Sigma, \Gamma, \bot, \delta \rangle$ is a deterministic PDA with special rules: Determined by the input symbol, only one symbol per push is allowed and reading the stack implies a pop.

- states, initial state, final states, stack alphabet, bottom-of-stack symbol (no change here),
Visibly pushdown languages (VPLs)

A *visibly pushdown language* (VPL) is the language accepted by a *visibly pushdown automaton* (VPA).

A VPA $A = \langle Q, q_0, Q_f, \Sigma, \Gamma, \bot, \delta \rangle$ is a deterministic PDA with special rules: Determined by the input symbol, only one symbol per **push** is allowed and reading the stack implies a **pop**.

- states, initial state, final states, stack alphabet, bottom-of-stack symbol (no change here),
- partitioning of the input alphabet: $\Sigma = \Sigma_i \uplus \Sigma_c \uplus \Sigma_r$, 

Visibly pushdown languages (VPLs)

A \textit{visibly pushdown language} (VPL) is the language accepted by a \textit{visibly pushdown automaton} (VPA).

A VPA $\mathcal{A} = \langle Q, q_0, Q_f, \Sigma, \Gamma, \bot, \delta \rangle$ is a deterministic PDA with special rules: Determined by the input symbol, only one symbol per \textbf{push} is allowed and reading the stack implies a \textbf{pop}.

- states, initial state, final states, stack alphabet, bottom-of-stack symbol (no change here),
- partitioning of the input alphabet: $\Sigma = \Sigma_i \cup \Sigma_c \cup \Sigma_r$,
- $\delta = \delta_i \cup \delta_c \cup \delta_r$,
  - $\delta_i \subseteq Q \times \Sigma_i \rightarrow Q$
  - $\delta_c \subseteq Q \times \Sigma_c \rightarrow (\Gamma \setminus \{\bot\}) \times Q$
  - $\delta_r \subseteq Q \times \Sigma_r \times \Gamma \rightarrow Q$
Visibly pushdown languages (VPLs)

A *visibly pushdown language* (VPL) is the language accepted by a *visibly pushdown automaton* (VPA).

A VPA $A = \langle Q, q_0, Q_f, \Sigma, \Gamma, \bot, \delta \rangle$ is a deterministic PDA with special rules: Determined by the input symbol, only one symbol per push is allowed and reading the stack implies a pop.

- states, initial state, final states, stack alphabet, bottom-of-stack symbol (no change here),
- partitioning of the input alphabet: $\Sigma = \Sigma_i \cup \Sigma_c \cup \Sigma_r$,
- $\delta = \delta_i \cup \delta_c \cup \delta_r$,
  - $\delta_i \subseteq Q \times \Sigma_i \rightarrow Q$
  - $\delta_c \subseteq Q \times \Sigma_c \rightarrow (\Gamma \setminus \{\bot\}) \times Q$
  - $\delta_r \subseteq Q \times \Sigma_r \times \Gamma \rightarrow Q$

Note: pops occur implicitly, $\bot$ never popped, no $\varepsilon$
Consider again $L_2 = \{c^n r^n | n > 0\}$. 

\begin{center}
\begin{tikzpicture}
  \node[state,initial] (q0) at (0,0) {$q_0$};
  \node[state] (q1) at (1,-2) {$q_1$};
  \node[state] (q2) at (2,0) {$q_2$};
  \node[state,accepting] (q3) at (3,0) {$q_3$};

  \path[->, thick]
    (q0) edge node {$\perp, c, \perp A$} (q1)
    (q1) edge node {$A, r, \varepsilon$} (q2)
    (q1) edge node {$B, r, \varepsilon$} (q3)
    (q2) edge node {$A, c, AB$} (q1)
    (q2) edge node {$B, c, BB$} (q3)
    (q3) edge node {$A, r, \varepsilon$} (q2)
    (q3) edge node {$A, r, \varepsilon$} (q1);
\end{tikzpicture}
\end{center}
Consider again $L_2 = \{c^n r^n \mid n > 0\}$. We construct a VPA for $L_2$.

Partitioning:
$\Sigma_i = \emptyset$, $\Sigma_c = \{c\}$, $\Sigma_r = \{r\}$
Consider again $L_2 = \{c^n r^n \mid n > 0\}$. We construct a VPA for $L_2$.

Partitioning:
$\Sigma_i = \emptyset$, $\Sigma_c = \{c\}$, $\Sigma_r = \{r\}$

$\delta_c = \{(q_0, c, A, q_1), (q_1, c, B, q_1)\}$

$\delta_r = \{(q_1, r, A, q_3), (q_1, r, B, q_2), (q_2, r, A, q_3), (q_2, r, B, q_2)\}$
From VPAs to NWAs

- main differences between VPAs and PDAs:
  - closed under determinism
  - partitioning of the alphabet
  - very limited use of the stack
- Do we really need the stack?
From VPAs to NWAs

• main differences between VPAs and PDAs:
  • closed under determinism
  • partitioning of the alphabet
  • very limited use of the stack

• Do we really need the stack?
  (Alur & Madhusudan 2006): no, with some further treatment of the input → nested words (NWs)

• automaton model: nested word automata (NWAs)
**From VPAs to NWAs**

- main differences between VPAs and PDAs:
  - closed under determinism
  - partitioning of the alphabet
  - very limited use of the stack

- Do we really need the stack?
  (Alur & Madhusudan 2006): no, with some further treatment of the input $\rightarrow$ *nested words* (NWs)

- automaton model: *nested word automata* (NWAs)

- *nested word languages* (NWLs) and VPLs have same power
  $\rightarrow$ NWAs $\preceq$ deterministic PDAs
From VPAs to NWAs

- main differences between VPAs and PDAs:
  - closed under determinism
  - partitioning of the alphabet
  - very limited use of the stack

- Do we really need the stack?
  (Alur & Madhusudan 2006): no, with some further treatment of the input → *nested words* (NWs)

- automaton model: *nested word automata* (NWAs)

- *nested word languages* (NWLs) and VPLs have same power → NWAs \(\preceq\) deterministic PDAs

- main idea: call and return symbols are matched in the input
Overview

Motivation and background

Nested words and their acceptors
  Nested words
  Nested word automata

Determinization proof

Conclusion
Well nested sequences

A sequence of symbols is *well nested* if calls and returns are matched without crossing, i.e., for any different call-return-pairs $(c_i, r_i), (c_j, r_j), c_i < c_j < r_i < r_j$ is forbidden.
Well nested sequences

A sequence of symbols is **well nested** if calls and returns are matched without crossing, i.e., for any different call-return-pairs $(c_i, r_i), (c_j, r_j), c_i < c_j < r_i < r_j$ is forbidden.

Examples:

```
  i c i c i i r r i
```
Well nested sequences

A sequence of symbols is \textit{well nested} if calls and returns are matched without crossing, i.e., for any different call-return-pairs \((c_i, r_i), (c_j, r_j), c_i < c_j < r_i < r_j\) is forbidden.

Examples:

\[
i c i c i i r r i
\]
Well nested sequences

A sequence of symbols is **well nested** if calls and returns are matched without crossing, i.e., for any different call-return-pairs $(c_i, r_i), (c_j, r_j)$, $c_i < c_j < r_i < r_j$ is forbidden.

Examples:

```
  i c i c i i r r i
```

```
  r c r r c i c i
```
Well nested sequences

A sequence of symbols is *well nested* if calls and returns are matched without crossing, i.e., for any different call-return-pairs \((c_i, r_i), (c_j, r_j)\), \(c_i < c_j < r_i < r_j\) is forbidden.

Examples:

```
  i c i c i i r r i
  r c r r c i c i
```
Well nested sequences

A sequence of symbols is *well nested* if calls and returns are matched without crossing, i.e., for any different call-return-pairs \((c_i, r_i), (c_j, r_j)\), \(c_i < c_j < r_i < r_j\) is forbidden.

Examples:

```
  i c i c i i r r i
```

```
  r c r r c i c i
```

Note: Every sequence has a unique well nesting.
Nested words

A relation $\sim \subset \{-\infty, 1, 2, \ldots, \ell\} \times \{1, 2, \ldots, \ell, \infty\}$ of length $\ell \geq 0$ is a matching relation if the following holds:

I. if $i \sim j$, then $i < j$  
   (monotone)

II. if $i_1 \sim j$ and $i_2 \sim j$, then $i_1 = i_2$  
    if $i \sim j_1$ and $i \sim j_1$, then $j_1 = j_2$  
    (left-unique)  
    (right-unique)

III. if $i_1 \sim j_1$ and $i_2 \sim j_2$, then we have not $i_1 < i_2 < j_1 < j_2$  
     (well nested)

Explanation:

I. not $r c$, not reflexive  
II. not $c c r$, not $c r r$  
III. not $c c r r$  

ex post note: $(-\infty, \infty) \not\in \sim$,  
$\pm \infty$ excluded from uniqueness
Nested words

A relation $\sim \subset \{−\infty, 1, 2, \ldots, \ell\} \times \{1, 2, \ldots, \ell, \infty\}$ of length $\ell \geq 0$ is a matching relation if the following holds:

I. if $i \sim j$, then $i < j$  \hspace{1cm} (monotone)

II. if $i_1 \sim j$ and $i_2 \sim j$, then $i_1 = i_2$ \hspace{1cm} (left-unique)
   if $i \sim j_1$ and $i \sim j_1$, then $j_1 = j_2$ \hspace{1cm} (right-unique)

III. if $i_1 \sim j_1$ and $i_2 \sim j_2$, then we have not $i_1 < i_2 < j_1 < j_2$ \hspace{1cm} (well nested)

If $i \sim j$, $i$ is a call position and $j$ is a return position. All the rest is an internal position. If $i \neq −\infty$ and $j \neq \infty$, they are well-matched, otherwise pending. $e \in \sim$ is a nesting edge.
Nested words

A relation $\sim \subset \{-\infty, 1, 2, \ldots, \ell\} \times \{1, 2, \ldots, \ell, \infty\}$ of length $\ell \geq 0$ is a *matching relation* if the following holds:

- if $i \sim j$, then $i < j$ (monotone)
- if $i_1 \sim j$ and $i_2 \sim j$, then $i_1 = i_2$ (left-unique)
- if $i \sim j_1$ and $i \sim j_1$, then $j_1 = j_2$ (right-unique)
- if $i_1 \sim j_1$ and $i_2 \sim j_2$, then we have not $i_1 < i_2 < j_1 < j_2$ (well nested)

If $i \sim j$, $i$ is a *call position* and $j$ is a *return position*. All the rest is an *internal position*. If $i \neq -\infty$ and $j \neq \infty$, they are *well-matched*, otherwise *pending*. $e \in \sim$ is a *nesting edge*.

A *nested word* $n$ over $\Sigma$ is a pair $(a_1 \cdots a_{\ell}, \sim)$, where $a_i \in \Sigma$ and $\sim$ is a matching relation of length $\ell$. 
Example 1

Here: $2 \sim 8$, $4 \sim 7$ and the whole word is well-matched.
Example 2

Here: $-\infty \rightsquigarrow 1$, $2 \rightsquigarrow 3$, $-\infty \rightsquigarrow 4$, $5 \rightsquigarrow \infty$, $7 \rightsquigarrow \infty$ and only $2 \rightsquigarrow 3$ is well-matched.
Definition of NWAs

\[ A = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle \] over alphabet \( \Sigma \)
Definition of NWAs

\[ \mathcal{A} = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle \text{ over alphabet } \Sigma \]

- \( Q \) finite set of \textit{linear} states,
- \( q_0 \in Q \) initial \textit{linear} state,
- \( Q_f \subseteq Q \) set of \textit{linear} final states,
Definition of NWAs

\[ \mathcal{A} = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle \text{ over alphabet } \Sigma \]

- \( Q \) finite set of *linear* states,
- \( q_0 \in Q \) initial *linear* state,
- \( Q_f \subseteq Q \) set of *linear* final states,
- \( P \) finite set of *hierarchical* states,
- \( p_0 \in Q \) initial *hierarchical* state,
- \( P_f \subseteq P \) set of *hierarchical* final states,
Nested words and their acceptors

Nested word automata

Definition of NWAs

\[ A = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle \text{ over alphabet } \Sigma \]

- \( Q \) finite set of \textit{linear} states,
- \( q_0 \in Q \) initial \textit{linear} state,
- \( Q_f \subseteq Q \) set of \textit{linear} final states,
- \( P \) finite set of \textit{hierarchical} states,
- \( p_0 \in Q \) initial \textit{hierarchical} state,
- \( P_f \subseteq P \) set of \textit{hierarchical} final states,
- \( \delta_i \subseteq Q \times \Sigma \rightarrow Q \) internal transition function,
- \( \delta_c \subseteq Q \times \Sigma \rightarrow Q \times P \) call transition function,
- \( \delta_r \subseteq Q \times P \times \Sigma \rightarrow Q \) return transition function
Definition of NWAs

\[ \mathcal{A} = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle \text{ over alphabet } \Sigma \]

- \( Q \) finite set of \textit{linear} states,
- \( q_0 \in Q \) initial \textit{linear} state,
- \( Q_f \subseteq Q \) set of \textit{linear} final states,
- \( P \) finite set of \textit{hierarchical} states,
- \( p_0 \in Q \) initial \textit{hierarchical} state,
- \( P_f \subseteq P \) set of \textit{hierarchical} final states,
- \( \delta_i \subseteq Q \times \Sigma \rightarrow Q \) internal transition function,
- \( \delta_c \subseteq Q \times \Sigma \rightarrow Q \times P \) call transition function,
- \( \delta_r \subseteq Q \times P \times \Sigma \rightarrow Q \) return transition function

acceptance via both \( Q_f \) and \( P_f \)
as VPAs: at return implicitly go to hierarchical state before matching call
\[ \mathcal{L}_2 \] as NWA

Consider again \( \mathcal{L}_2 = \{c^n r^n \mid n > 0\} \).

We construct an NWA for \( \mathcal{L}_2' := \{(\langle c \rangle^n (r) \rangle)^n \mid n > 0\} \).
Consider again $L_2 = \{c^n r^n | n > 0\}$.

We construct an NWA for $L'_2 := \{((\langle c \rangle^n (r)\rangle)^n | n > 0\}$. 

$P = \{p_0, p_1\}, \ P_f \subseteq \{p_0\}$
Consider again $\mathcal{L}_2 = \{c^n r^n \mid n > 0\}$.

We construct an NWA for $\mathcal{L}_2' := \{((c)\langle (r)\rangle)^n \mid n > 0\}$.

We can also use hierarchical states for acceptance.

\[ P = \{p_0, p_1\}, \quad P_f = \{p_0\} \]
Remarks

- no stack anymore, but structure on the input word
Remarks

- no stack anymore, but structure on the input word
- nondeterministic NWAs: $Q_0 \subseteq Q$, $P_0 \subseteq P$, $\delta$
Remarks

- no stack anymore, but structure on the input word
- nondeterministic NWAs: $Q_0 \subseteq Q$, $P_0 \subseteq P$, $\delta$
  possibly exponentially more states for deterministic NWAs
Remarks

- no stack anymore, but structure on the input word
- nondeterministic NWAs: $Q_0 \subseteq Q$, $P_0 \subseteq P$, $\delta$
  possibly exponentially more states for deterministic NWAs
- not all sets of NWs acceptable by NWAs
  $\{(\langle a \rangle^n \langle b \rangle)^n \mid n > 0\}$ vs. $\{a^n b^n \mid n > 0\}$
### Comparison of properties

<table>
<thead>
<tr>
<th></th>
<th>DFA</th>
<th>DNWA</th>
<th>PDA</th>
<th>DPDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-/suffix</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>$\cup, \cdot, \ast$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complement</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\cap$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>emptiness</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>equivalence</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inclusion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Equivalence and inclusion problem are \textit{exptime}-complete for nondeterministic NWAs. Implication: determinization $\in \Omega(\text{exptime})$ if at all possible.
Comparison of properties

<table>
<thead>
<tr>
<th></th>
<th>DFA</th>
<th>DNWA</th>
<th>PDA</th>
<th>DPDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-/suffix</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>∪, ·, *</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>complement</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>∩</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>emptiness</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>equivalence</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inclusion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Equivalence and inclusion problem are exptime-complete for nondeterministic NWAs. Implication: determinization ∈ Ω(exptime) if at all possible.
Comparison of properties

<table>
<thead>
<tr>
<th></th>
<th>DFA</th>
<th>DNWA</th>
<th>PDA</th>
<th>DPDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-/suffix</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\cup, \cdot, \ast$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>$\times$</td>
</tr>
<tr>
<td>complement</td>
<td>✓</td>
<td>✓</td>
<td>$\times$</td>
<td>✓</td>
</tr>
<tr>
<td>$\cap$</td>
<td>✓</td>
<td>✓</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>emptiness</td>
<td>$\text{NLOGSPACE}$</td>
<td>$\text{PTIME}$</td>
<td>$\text{PTIME}$</td>
<td>$\text{PTIME}$</td>
</tr>
<tr>
<td>equivalence</td>
<td>$\text{NLOGSPACE}$</td>
<td>$\text{PTIME}$</td>
<td>undecidable</td>
<td>decidable</td>
</tr>
<tr>
<td>inclusion</td>
<td>$\text{NLOGSPACE}$</td>
<td>$\text{PTIME}$</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

Note: Equivalence and inclusion problem are $\text{EXPTIME}$-complete for nondeterministic NWAs. Implication: determinization $\in \Omega(\text{EXPTIME})$ if at all possible.
### Comparison of properties

<table>
<thead>
<tr>
<th></th>
<th>DFA</th>
<th>DNWA</th>
<th>PDA</th>
<th>DPDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-/suffix</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\cup, \cdot, *$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>$\times$</td>
</tr>
<tr>
<td>complement</td>
<td>✓</td>
<td>✓</td>
<td>$\times$</td>
<td>✓</td>
</tr>
<tr>
<td>$\cap$</td>
<td>✓</td>
<td>✓</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>emptiness</td>
<td>NLOGSPACE</td>
<td>pTIME</td>
<td>pTIME</td>
<td>pTIME</td>
</tr>
<tr>
<td>equivalence</td>
<td>NLOGSPACE</td>
<td>pTIME</td>
<td>undecidable</td>
<td>decidable</td>
</tr>
<tr>
<td>inclusion</td>
<td>NLOGSPACE</td>
<td>pTIME</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

Note: Equivalence and inclusion problem are \textsc{EXPTIME}-complete for nondeterministic NWAs.
Implication: determinization $\in \Omega(\textsc{EXPTIME})$ if at all possible.
Overview

Motivation and background

Nested words and their acceptors

Determinization proof
  Intuition
  Construction

Conclusion
Idea behind the proof

- goal: determinize a nondeterministic NWA (NNWA)
Idea behind the proof

- goal: determinize a nondeterministic NWA (NNWA)
- state of automaton $\mathcal{A}$ for nested word $n$ with position $k$:
  - deterministic NWA (DNWA): $(q_k, p_k)$
  - NNWA: one of $(q_{k_1}, p_{k_1}), \ldots, (q_{k_i}, p_{k_j})$
Idea behind the proof

- goal: determinize a nondeterministic NWA (NNWA)
- state of automaton \( A \) for nested word \( n \) with position \( k \):
  - deterministic NWA (DNWA): \( (q_k, p_k) \)
  - NNWA: one of \( (q_{k_1}, p_{k_1}), \ldots, (q_{k_i}, p_{k_j}) \)
- finite automata: call the states \( \{q_{k_1}, \ldots, q_{k_i}\} \)
Idea behind the proof

• goal: determinize a nondeterministic NWA (NNWA)
• state of automaton $\mathcal{A}$ for nested word $n$ with position $k$:
  deterministic NWA (DNWA): $(q_k, p_k)$
  NNWA: one of $(q_{k_1}, p_{k_1}), \ldots, (q_{k_i}, p_{k_j})$
• finite automata: call the states $\{q_{k_1}, \ldots, q_{k_i}\}$
• NWAs: also need information about hierarchical states
  $\rightarrow$ powerset construction over nesting edges
  hierarchical states $= \text{nesting edges} + \text{call symbol so far}$
Idea behind the proof

- goal: determinize a nondeterministic NWA (NNWA)
- state of automaton $A$ for nested word $n$ with position $k$:
  deterministic NWA (DNWA): $(q_k, p_k)$
  NNWA: one of $(q_{k_1}, p_{k_1}), \ldots, (q_{k_i}, p_{k_j})$
- finite automata: call the states $\{q_{k_1}, \ldots, q_{k_i}\}$
- NWAs: also need information about hierarchical states
  $\rightarrow$ powerset construction over nesting edges
  hierarchical states $= $ nesting edges + call symbol so far
- handle hierarchical proceeding when reading return symbols
The states: definition

Consider the NNWA $\mathcal{A} = \langle Q, Q_0, Q_f, P, P_0, P_f, \delta_i, \delta_c, \delta_r \rangle$.

We construct the DNWA $\mathcal{B} = \langle Q', q'_0, Q'_f, P', p'_0, P'_f, \delta'_i, \delta'_c, \delta'_r \rangle$: 
Consider the NNWA $\mathcal{A} = \langle Q, Q_0, Q_f, P, P_0, P_f, \delta_i, \delta_c, \delta_r \rangle$.

We construct the DNWA $\mathcal{B} = \langle Q', q'_0, Q'_f, P', p'_0, P'_f, \delta'_i, \delta'_c, \delta'_r \rangle$:

- $Q' := 2^{Q \times Q} = \{S_1, \ldots, S_i\}$
The states: definition

Consider the NNWA $\mathcal{A} = \langle Q, Q_0, Q_f, P, P_0, P_f, \delta_i, \delta_c, \delta_r \rangle$. We construct the DNWA $\mathcal{B} = \langle Q', q'_0, Q'_f, P', P'_0, P'_f, \delta'_i, \delta'_c, \delta'_r \rangle$:

- $Q' := 2^{Q \times Q} = \{S_1, \ldots, S_i\}$
- $q'_0 := Q_0 \times Q_0$
The states: definition

Consider the NNWA $A = \langle Q, Q_0, Q_f, P, P_0, P_f, \delta_i, \delta_c, \delta_r \rangle$.
We construct the DNWA $B = \langle Q', q'_0, Q'_f, P', p'_0, P'_f, \delta'_i, \delta'_c, \delta'_r \rangle$:

- $Q' := 2^{Q \times Q} = \{ S_1, \ldots, S_i \}$
- $q'_0 := Q_0 \times Q_0$
- $Q'_f := \{ S \mid \exists q, q'. (q, q') \in S \land q' \in Q_f \}$
  or: $S \in Q'_f :\iff S$ contains $(q, q')$ with $q' \in Q_f$
Consider the NNWA $A = \langle Q, Q_0, Q_f, P, P_0, P_f, \delta_i, \delta_c, \delta_r \rangle$.
We construct the DNWA $B = \langle Q', q'_0, Q'_f, P', p'_0, P'_f, \delta'_i, \delta'_c, \delta'_r \rangle$:

- $Q' := 2^{Q \times Q} = \{ S_1, \ldots, S_i \}$
- $q'_0 := Q_0 \times Q_0$
- $Q'_f := \{ S \mid \exists q, q'. (q, q') \in S \land q' \in Q_f \}$
  or: $S \in Q'_f :\iff S$ contains $(q, q')$ with $q' \in Q_f$
- $P' := \{ p'_0 \} \cup (Q' \times \Sigma)$
Consider the NNWA $A = \langle Q, Q_0, Q_f, P, P_0, P_f, \delta_i, \delta_c, \delta_r \rangle$.

We construct the DNWA $B = \langle Q', q'_0, Q'_f, P', p'_0, P'_f, \delta'_i, \delta'_c, \delta'_r \rangle$:

- $Q' := 2^{Q \times Q} = \{S_1, \ldots, S_i\}$
- $q'_0 := Q_0 \times Q_0$
- $Q'_f := \{S \mid \exists q, q'. (q, q') \in S \land q' \in Q_f\}$
  or: $S \in Q'_f :\Leftrightarrow S \text{ contains } (q, q') \text{ with } q' \in Q_f$
- $P' := \{p'_0\} \cup (Q' \times \Sigma)$
- $p'_0 := \text{fresh hierarchical state}$
The states: definition

Consider the NNWA $A = \langle Q, Q_0, Q_f, P, P_0, P_f, \delta_i, \delta_c, \delta_r \rangle$.
We construct the DNWA $B = \langle Q', q'_0, Q'_f, P', p'_0, P'_f, \delta'_i, \delta'_c, \delta'_r \rangle$:

- $Q' := 2^{Q \times Q} = \{ S_1, \ldots, S_i \}$
- $q'_0 := Q_0 \times Q_0$
- $Q'_f := \{ S \mid \exists q, q'.(q, q') \in S \land q' \in Q_f \}$
  or: $S \in Q'_f \iff S$ contains $(q, q')$ with $q' \in Q_f$
- $P' := \{ p'_0 \} \cup (Q' \times \Sigma)$
- $p'_0 :=$ fresh hierarchical state
- $P'_f := P'$
The states: semantics

Consider a nested word $n$ with $k$ pending calls. We can write this

$$n = n_1 \langle c_1 n_2 \langle c_2 \cdots n_k \langle c_k n_{k+1}$$

where the $n_i$ have no pending calls.
The states: semantics

Consider a nested word $n$ with $k$ pending calls. We can write this

$$n = n_1 \langle c_1 n_2 \langle c_2 \cdots n_k \langle c_k n_{k+1}$$

where the $n_i$ have no pending calls.

**Invariants**

1. After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $\langle c_i$. 
The states: semantics

Consider a nested word \( n \) with \( k \) pending calls. We can write this

\[
n = n_1 c_1 n_2 c_2 \cdots n_k c_k n_{k+1}
\]

where the \( n_i \) have no pending calls.

Invariants

I. After reading \( n \), \( B \) will be in state \( S_{k+1} \), where \( (S_i, c_i) \) will be the hierarchical state for each \( c_i \).

II. \( S_i \) contains the pair \( (q, q') \) iff \( q \xrightarrow{n_i} A q' \).
The states: semantics

Consider a nested word \( n \) with \( k \) pending calls. We can write this

\[
n = n_1 \langle c_1 n_2 \langle c_2 \cdots n_k \langle c_k n_{k+1}
\]

where the \( n_i \) have no pending calls.

Invariants

I  After reading \( n \), \( B \) will be in state \( S_{k+1} \), where \( (S_i, c_i) \) will be the hierarchical state for each \( \langle c_i \).

II  \( S_i \) contains the pair \( (q, q') \) iff \( q \xrightarrow{n_i} A q' \).

Question: acceptance condition of \( B \) for \( n \)?
The states: semantics

Consider a nested word \( n \) with \( k \) pending calls. We can write this

\[
n = n_1 \langle c_1 n_2 \langle c_2 \cdots n_k \langle c_k n_{k+1}
\]

where the \( n_i \) have no pending calls.

Invariants

1. After reading \( n \), \( B \) will be in state \( S_{k+1} \), where \( (S_i, c_i) \) will be the hierarchical state for each \( \langle c_i \).

2. \( S_i \) contains the pair \( (q, q') \) iff \( q \xrightarrow{n_i} A q' \).

Question: acceptance condition of \( B \) for \( n \)?
Answer: \( S_{k+1} \in Q'_f \)
The states: semantics

Consider a nested word $n$ with $k$ pending calls. We can write this

$$n = n_1\langle c_1 n_2\langle c_2 \cdots n_k\langle c_k n_{k+1}$$

where the $n_i$ have no pending calls.

Invariants

1. After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $\langle c_i$.

2. $S_i$ contains the pair $(q, q')$ iff $q \xrightarrow{n_i} \mathcal{A} q'$.

Question: acceptance condition of $B$ for $n$?

Answer: $S_{k+1} \in Q_f'$,

i.e., $\exists q, q'. (q, q') \in S_{k+1} \land q \xrightarrow{n_{k+1}} \mathcal{A} q' \land q' \in Q_f$
Internal transitions

1. After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $\langle c_i \rangle$.

2. $S_i$ contains the pair $(q, q')$ iff $q \xrightarrow{n_i} A q'$.

\[
n' = n \cdot i = n_1 \langle c_1 \rangle n_2 \langle c_2 \rangle \cdots n_k \langle c_k \rangle n_{k+1} i
\]

\[
\delta'_i(S_{k+1}, i) =
\]
Internal transitions

I After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $c_i$.

II $S_i$ contains the pair $(q, q')$ iff $q \xrightarrow{n_i} \mathcal{A} q'$. 

\[
q \xrightarrow{n_{k+1}} q' \xrightarrow{i} q''
\]

\[
n' = n \cdot i = n_1c_1n_2c_2 \cdots n_kc_kn_{k+1}i
\]

\[
\delta'_i(S_{k+1}, i) = \{(q, q'') \mid (q, q') \in S_{k+1} \land q'' \in \delta_i(q', i)\}
\]
Example
Example

\[
\begin{align*}
0 \xrightarrow{a} & \ 1 \xrightarrow{b} \ 3 \\
0 \xrightarrow{a} & \ 2 \xrightarrow{c} \ 4
\end{align*}
\]

\[
\Rightarrow
\begin{align*}
0 \xrightarrow{a} & \ (0, 0) \\
0 \xrightarrow{a} & \ (0, 1), \ (0, 2) \\
0 \xrightarrow{a} & \ (0, 3) \\
0 \xrightarrow{a} & \ (0, 4)
\end{align*}
\]
Call transitions

1. After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $\langle c_i \rangle$.

2. $S_i$ contains the pair $(q, q')$ iff $q \xrightarrow{n_i} A q'$.

$$n' = n \cdot \langle c_{k+1} \rangle = n_1 \langle c_1 \rangle n_2 \langle c_2 \rangle \cdots n_k \langle c_k \rangle n_{k+1} \langle c_{k+1} \rangle$$

$$\delta'_c(S_{k+1}, c_{k+1}) =$$

new hierarchical state that keeps track of the old state/symbol
Call transitions

1. After reading \( n \), \( B \) will be in state \( S_{k+1} \), where \( (S_i, c_i) \) will be the hierarchical state for each \( \langle c_i \rangle \).

2. \( S_i \) contains the pair \( (q, q') \) iff \( q \xrightarrow{n_i^j} \mathcal{A} q' \).

\[
n' = n \cdot \langle c_{k+1} \rangle = n_1 \langle c_1 \rangle n_2 \langle c_2 \rangle \cdots n_k \langle c_k \rangle n_{k+1} \langle c_{k+1} \rangle
\]

\[
\delta'_c(S_{k+1}, c_{k+1}) = (S', (S_{k+1}, c_{k+1}))
\]

new hierarchical state that keeps track of the old state/symbol...
Call transitions

I. After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $\langle c_i \rangle$.

II. $S_i$ contains the pair $(q, q')$ iff $q $\xrightarrow{n_i} A $ q'$.

$$n' = n \cdot \langle c_{k+1} = n_1 \langle c_1 n_2 \langle c_2 \cdots n_k \langle c_k n_{k+1} \langle c_{k+1}$$

$$\delta'(S_{k+1}, c_{k+1}) = (S', (S_{k+1}, c_{k+1})),$$

$$S' = \{ (q'', q'') \mid (q, q') \in S_{k+1} \land \exists p \in P. (q'', p) \in \delta_c (q', c_{k+1}) \}$$

new hierarchical state that keeps track of the old state/symbol
Example

\[
\langle \text{c/p}_1 \rangle \\
\langle \text{c/p}_1 \rangle
\]

\[
\begin{cases}
(0, 0) \\
(1, 1), (2, 2)
\end{cases}
\]
Example

\[\langle c/p_1 \rangle \]

\[\langle c/p_1 \rangle 1 \quad \Rightarrow \quad \langle c/(\{(0, 0)\}, c) \rangle \]

\[\{(0, 0)\} \quad \Rightarrow \quad \{(1, 1), (2, 2)\} \]
Return transitions

I. After reading \( n \), \( \mathcal{B} \) will be in state \( S_{k+1} \), where \( (S_i, c_i) \) will be the hierarchical state for each \( \langle c_i \rangle \).

II. \( S_i \) contains the pair \( (q, q') \) iff \( q \xrightarrow{n_i} \mathcal{A} q' \).

\[
n' = n \cdot r = n_1 \langle c_1 n_2 \langle c_2 \cdots n_k \langle c_k n_{k+1} r \rangle \rangle \rangle
\]

We have two cases here:

\( k = 0 \) no matching call, like internal transition

\[
\delta'_r(S_{k+1}, p'_0, r) =
\]
Return transitions

I. After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $\langle c_i \rangle$.

II. $S_i$ contains the pair $(q, q')$ iff $q \xrightarrow{n_i} \mathcal{A} q'$.

$$n' = n \cdot r = n_1 \langle c_1 n_2 \langle c_2 \ldots n_k \langle c_k n_{k+1} r \rangle \rangle$$

We have two cases here:

$k = 0$ no matching call, like internal transition

$$\delta'_r(S_{k+1}, p'_0, r) = \{(q, q'') \mid (q, q') \in S_{k+1} \land \exists p \in P_0.q'' \in \delta_r(q', p, r)\}$$
Return transitions

I After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $\langle c_i \rangle$.

II $S_i$ contains the pair $(q, q')$ iff $q \xrightarrow{n_i} A q'$.

$$n' = n \cdot r = n_1 \langle c_1 n_2 \langle c_2 \cdots n_k \langle c_k n_{k+1} r \rangle \rangle \rangle$$

We have two cases here:

$k = 0$ no matching call, like internal transition

$$\delta'_r(S_{k+1}, p'_0, r) = \{(q, q'') \mid (q, q') \in S_{k+1} \wedge \exists p \in P_0. q'' \in \delta_r(q', p, r)\}$$

$k > 0$ subword $n_k \langle c_k n_{k+1} r \rangle$, hierarchical state $= (S_k, c_k)$

$$\delta'_r(S_{k+1}, (S_k, c_k), r) = \ldots$$
Return transitions

I. After reading $n$, $B$ will be in state $S_{k+1}$, where $(S_i, c_i)$ will be the hierarchical state for each $\langle c_i \rangle$.

II. $S_i$ contains the pair $(q, q')$ iff $q \xrightarrow{n_i} A q'$.

\[
n' = n \cdot r = n_1 \langle c_1 \rangle n_2 \langle c_2 \cdots n_k \langle c_k \rangle n_{k+1} \rangle r
\]

We have two cases here:

$k = 0$ no matching call, like internal transition

\[
\delta_r'(S_{k+1}, p_0', r) = \{(q, q'') \mid (q, q') \in S_{k+1} \land \exists p \in P_0. q'' \in \delta_r(q', p, r)\}
\]

$k > 0$ subword $n_k \langle c_k \rangle n_{k+1} \rangle r$, hierarchical state $= (S_k, c_k)$

\[
\delta_r'(S_{k+1}, (S_k, c_k), r) = \{(q, q'') \mid (q, q') \in S_k \land (q_1, q_2) \in S_{k+1} \land \exists p \in P. (q_1, p) \in \delta_c(q', c_k) \land q'' \in \delta_r(q_2, p, r)\}
\]
Example
Nested Word Automata
Determinization proof
Construction

Example

\[ r \rangle / p_0 \]

\[ r \rangle / p_0' \]

\[ \{(0, 0)\} \]

\[ \{(0, 1), (0, 2)\} \]
Example
Example

\[
\langle c/p_1 \rangle / p_1 \\
0 \rightarrow 1 \rightarrow 2 \rightarrow 3
\]

\[
\Rightarrow \\
\langle c/(\{(0, 0)\}, c) \rangle //(\{(0, 0)\}, c) \\
\{(1, 1)\}
\]
Résumé

- now all components of $B$ defined
Résumé

- now all components of $B$ defined
- correctness results from invariants
Résumé

- now all components of $B$ defined
- correctness results from invariants
- complexity: if $|Q| = s$, then $|Q'| = 2^s^2$ and $|P'| \in \mathcal{O}(2^s^2)$

This is succinct, so there exists an example where the DNWA cannot have less states.
Overview

Motivation and background

Nested words and their acceptors

Determinization proof

Conclusion
nested word languages as a (proper) fragment of deterministic context-free languages strictly more expressive than regular languages
Conclusion

- nested word languages as a (proper) fragment of deterministic context-free languages strictly more expressive than regular languages
- visibly pushdown automata and nested word automata as suitable models for this class
Conclusion

- nested word languages as a (proper) fragment of deterministic context-free languages strictly more expressive than regular languages
- visibly pushdown automata and nested word automata as suitable models for this class
- no stack, but complexity shifted to the input word
Conclusion

- nested word languages as a (proper) fragment of deterministic context-free languages strictly more expressive than regular languages
- visibly pushdown automata and nested word automata as suitable models for this class
- no stack, but complexity shifted to the input word
- all relevant closure properties, all interesting problems decidable
Conclusion

- nested word languages as a (proper) fragment of deterministic context-free languages strictly more expressive than regular languages
- visibly pushdown automata and nested word automata as suitable models for this class
- no stack, but complexity shifted to the input word
- all relevant closure properties, all interesting problems decidable
- determinization always possible in $O(2^{s^2})$
Conclusion

- nested word languages as a (proper) fragment of deterministic context-free languages strictly more expressive than regular languages
- visibly pushdown automata and nested word automata as suitable models for this class
- no stack, but complexity shifted to the input word
- all relevant closure properties, all interesting problems decidable
- determinization always possible in $O(2^{s^2})$
- many practical problems describable as nested words
Nested Word Automata

Conclusion

• nested word languages as a (proper) fragment of deterministic context-free languages strictly more expressive than regular languages

• visibly pushdown automata and nested word automata as suitable models for this class

• no stack, but complexity shifted to the input word

• all relevant closure properties, all interesting problems decidable

• determinization always possible in $O(2^{s^2})$

• many practical problems describable as nested words

• recent concept, time will show the relevance