Automata Theory

Nested Word Automata

Christian Schilling

June 4th, 2012

Overview

Overview

Regular language

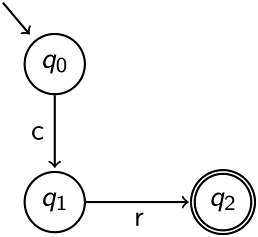
```
procedure foo()
{
    return;
}
```

$$\mathcal{L}_1 = \{\mathsf{c}\ \mathsf{r}\}$$

Regular language

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

$$\mathcal{L}_1 = \{\text{c r}\}$$



(det.) Context-free language

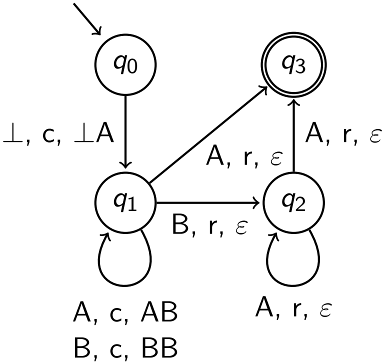
```
procedure bar()
{
    if (*)
        call bar();
    return;
}
```

$$\mathcal{L}_2 = \{\mathrm{c}^n\ \mathrm{r}^n \mid n > 0\}$$

(det.) Context-free language

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

$$\mathcal{L}_2 = \{\mathrm{c}^n\ \mathrm{r}^n \mid n > 0\}$$



Comparison

☺	**regular**	**context-free**
comparison of numbers	constants	two variables
closure		
decidability		
determinize		

Comparison

☺ ☹	**regular**	**context-free**
comparison of numbers	constants	two variables
closure	all standard properties	not under intersection and complementation
decidability		
determinize		

Comparison

☺ ☹ ☹	**regular**	**context-free**
comparison of numbers	constants	two variables
closure	all standard properties	not under intersection and complementation
decidability	all standard problems	intersection, inclusion, equivalence undecidable
determinize		

Comparison

☺ ☹ ☹ ☹	**regular**	**context-free**
comparison of numbers	constants	two variables
closure	all standard properties	not under intersection and complementation
decidability	all standard problems	intersection, inclusion, equivalence undecidable
determinize	powerset construction	not possible

Comparison

☺ ☹ ☹ ☹	**regular**	**context-free**
comparison of numbers	constants	two variables
closure	all standard properties	not under intersection and complementation
decidability	all standard problems	intersection, inclusion, equivalence undecidable
determinize	powerset construction	not possible

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

Comparison

☺ ☹ ☹ ☹	regular	context-free
comparison of numbers	constants	two variables
closure	all standard properties	not under intersection and complementation
decidability	all standard problems	intersection, inclusion, equivalence undecidable
determinize	powerset construction	not possible

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 $\mathcal{A} = \langle Q, q_0, Q_f, \Sigma, \Gamma, \perp, \delta \rangle$ 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 $\mathcal{A} = \langle Q, q_0, Q_f, \Sigma, \Gamma, \perp, \delta \rangle$ 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 $\mathcal{A} = \langle Q, q_0, Q_f, \Sigma, \Gamma, \perp, \delta \rangle$ 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 *visibly pushdown language* (VPL) is the language accepted by a *visibly pushdown automaton* (VPA).

A VPA $\mathcal{A} = \langle Q, q_0, Q_f, \Sigma, \Gamma, \perp, \delta \rangle$ 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: $\Sigma = \Sigma_i \uplus \Sigma_c \uplus \Sigma_r$,
- $\delta = \delta_i \uplus \delta_c \uplus \delta_r$,
    - $\delta_i \subseteq Q \times \Sigma_i \to Q$
    - $\delta_c \subseteq Q \times \Sigma_c \to (\Gamma \setminus \{\perp\}) \times Q$
    - $\delta_r \subseteq Q \times \Sigma_r \times \Gamma \to Q$

Visibly pushdown languages (VPLs)

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

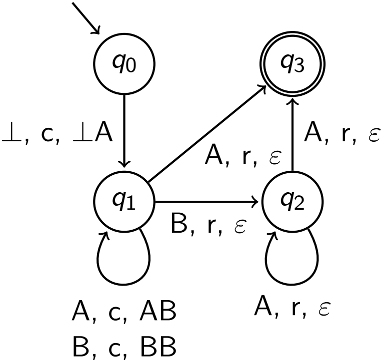
A VPA $\mathcal{A} = \langle Q, q_0, Q_f, \Sigma, \Gamma, \perp, \delta \rangle$ is a deterministic PDA with special rules: Determined by the input symbol, only one symbol per **push** is allowed and reading the stack implies a **pop**.

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

Note: pops occur implicitly, $\perp$ never popped, no ε

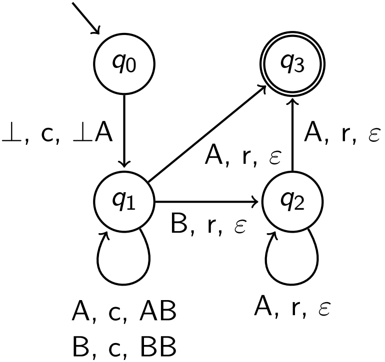
$\mathcal{L}_2$ as VPL

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



$\mathcal{L}_2$ as VPL

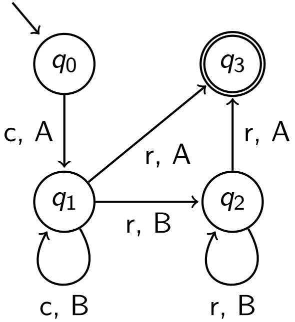
Consider again $\mathcal{L}_2 = \{\mathrm{c}^n\ \mathrm{r}^n \mid n > 0\}$. We construct a VPA for $\mathcal{L}_2$.



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

$\mathcal{L}_2$ as VPL

Consider again $\mathcal{L}_2 = \{c^n\ r^n \mid n > 0\}$. We construct a VPA for $\mathcal{L}_2$.



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

$$\begin{aligned}\delta_c = \{\ & (q_0, c, A, q_1),\\ & (q_1, c, B, q_1)\ \ \}\\ \delta_r = \{\ & (q_1, r, A, q_3),\\ & (q_1, r, B, q_2),\\ & (q_2, r, A, q_3),\\ & (q_2, r, B, q_2)\ \ \}\end{aligned}$$

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 $\rightarrow$ *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 $\rightarrow$ *nested words* (NWs)
- automaton model: *nested word automata* (NWAs)
- *nested word languages* (NWLs) and VPLs have same power
  $\rightarrow$ NWAs $\preceq$ deterministic PDAs
- main idea: call and return symbols are matched in the input

Overview

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

- I if $i \rightsquigarrow j$, then $i < j$ (monotone)
- II if $i_1 \rightsquigarrow j$ and $i_2 \rightsquigarrow j$, then $i_1 = i_2$ (left-unique)
  if $i \rightsquigarrow j_1$ and $i \rightsquigarrow j_1$, then $j_1 = j_2$ (right-unique)
- III if $i_1 \rightsquigarrow j_1$ and $i_2 \rightsquigarrow j_2$, 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**

$\boxed{\text{ex post note: } (-\infty, \infty) \notin \rightsquigarrow, \; \pm\infty \text{ excluded from uniqueness}}$

Nested words

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

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

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

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

If $i \rightsquigarrow j$, 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*. $e \in \rightsquigarrow$ is a *nesting edge*.

Nested words

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

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

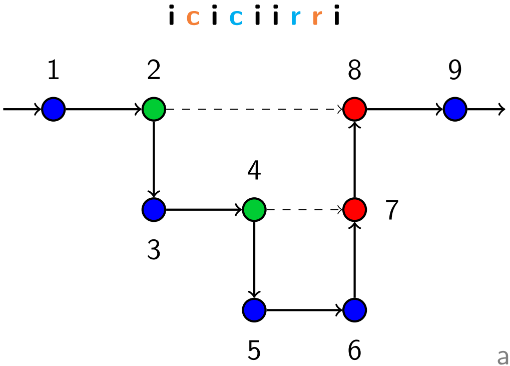
II if $i_1 \rightsquigarrow j$ and $i_2 \rightsquigarrow j$, then $i_1 = i_2$ (left-unique)
if $i \rightsquigarrow j_1$ and $i \rightsquigarrow j_1$, then $j_1 = j_2$ (right-unique)

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

If $i \rightsquigarrow j$, 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*. $e \in \rightsquigarrow$ is a *nesting edge*.

A *nested word* n over Σ is a pair $(a_1 \cdots a_\ell, \rightsquigarrow)$, where $a_i \in \Sigma$ and $\rightsquigarrow$ is a matching relation of length ℓ.

Example 1

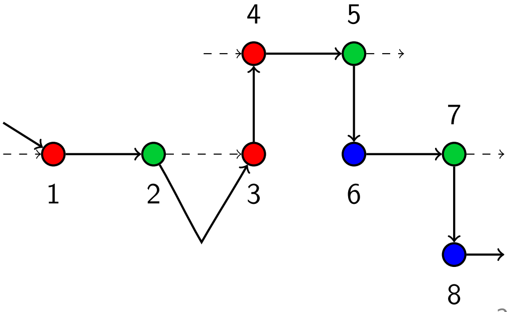


adapted from [1]

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

Example 2

r c r r c i c i



adapted from [1]

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

$\mathcal{A} = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle$ over alphabet Σ

Definition of NWAs

$\mathcal{A} = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle$ over alphabet Σ

- Q finite set of *linear* states,
- $q_0 \in Q$ initial *linear* state,
- $Q_f \subseteq Q$ set of *linear* final states,

Definition of NWAs

$\mathcal{A} = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle$ over alphabet Σ

- 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,

Definition of NWAs

$\mathcal{A} = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle$ over alphabet Σ

- 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,
- $\delta_i \subseteq Q \times \Sigma \to Q$ internal transition function,
- $\delta_c \subseteq Q \times \Sigma \to Q \times P$ call transition function,
- $\delta_r \subseteq Q \times P \times \Sigma \to Q$ return transition function

Definition of NWAs

$\mathcal{A} = \langle Q, q_0, Q_f, P, p_0, P_f, \delta_i, \delta_c, \delta_r \rangle$ over alphabet Σ

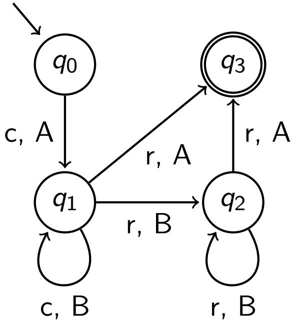
- Q finite set of *linear* states,
- $q_0 \in Q$ initial *linear* state,
- $Q_f \subseteq Q$ set of *linear* final states,
- P finite set of *hierarchical* states,
- $p_0 \in Q$ initial *hierarchical* state,
- $P_f \subseteq P$ set of *hierarchical* final states,
- $\delta_i \subseteq Q \times \Sigma \to Q$ internal transition function,
- $\delta_c \subseteq Q \times \Sigma \to Q \times P$ call transition function,
- $\delta_r \subseteq Q \times P \times \Sigma \to Q$ 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 $\mathcal{L}_2 = \{\mathrm{c}^n\ \mathrm{r}^n \mid n > 0\}$.

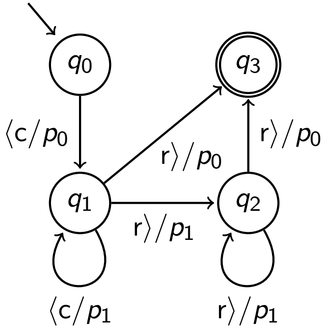
We construct an NWA for $\mathcal{L}_2' := \{(\langle\mathrm{c})^n\ (\mathrm{r}\rangle)^n \mid n > 0\}$.



$\mathcal{L}_2$ as NWA

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

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



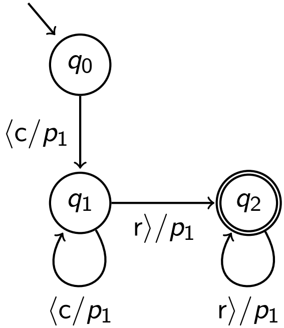
$P = \{p_0, p_1\}$, $P_f \subseteq \{p_0\}$

$\mathcal{L}_2$ as NWA

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

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

We can also use hierarchical states for acceptance.



$P = \{p_0, p_1\}$, $P_f = \{p_0\}$

Remarks

- no stack anymore, but structure on the input word

Remarks

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

Remarks

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

Remarks

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

Comparison of properties

	DFA	DNWA	PDA	DPDA
pre-/suffix	✓	✓	✓	✓
$\cup, \cdot, *$				
complement				
$\cap$				
emptiness				
equivalence				
inclusion				

Comparison of properties

	DFA	DNWA	PDA	DPDA
pre-/suffix	✓	✓	✓	✓
$\cup, \cdot, *$	✓	✓	✓	✗
complement	✓	✓	✗	✓
$\cap$	✓	✓	✗	✗
emptiness				
equivalence				
inclusion				

Comparison of properties

	DFA	DNWA	PDA	DPDA
pre-/suffix	✓	✓	✓	✓
$\cup, \cdot, *$	✓	✓	✓	✗
complement	✓	✓	✗	✓
$\cap$	✓	✓	✗	✗
emptiness	NLOGSPACE	PTIME	PTIME	PTIME
equivalence	NLOGSPACE	PTIME	undecidable	decidable
inclusion	NLOGSPACE	PTIME	undecidable	undecidable

Comparison of properties

	DFA	**DNWA**	**PDA**	**DPDA**
pre-/suffix	✓	✓	✓	✓
$\cup, \cdot, *$	✓	✓	✓	✗
complement	✓	✓	✗	✓
$\cap$	✓	✓	✗	✗
emptiness	NLOGSPACE	PTIME	PTIME	PTIME
equivalence	NLOGSPACE	PTIME	undecidable	decidable
inclusion	NLOGSPACE	PTIME	undecidable	undecidable

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

Overview

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 $\mathcal{A}$ for nested word n with position k:
  deterministic NWA (DNWA): (q_k, p_k)
  NNWA: one of (q_{k_1}, p_{k_1}), ..., (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}), ..., (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

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}), ..., (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 $\mathcal{A} = \langle Q, Q_0, Q_f, P, P_0, \overbrace{P_f}^{\stackrel{\text{wlog}}{=} P}, \delta_i, \delta_c, \delta_r \rangle$.
We construct the DNWA $\mathcal{B} = \langle Q', q'_0, Q'_f, P', p'_0, P'_f, \delta'_i, \delta'_c, \delta'_r \rangle$:

The states: definition

Consider the NNWA $\mathcal{A} = \langle Q, Q_0, Q_f, P, P_0, \overbrace{P_f}^{\overset{\text{wlog}}{=P}}, \delta_i, \delta_c, \delta_r \rangle$.
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 $\mathcal{A} = \langle Q, Q_0, Q_f, P, P_0, \overbrace{P_f}^{\overset{\text{wlog}}{=P}}, \delta_i, \delta_c, \delta_r \rangle$.
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 $\mathcal{A} = \langle Q, Q_0, Q_f, P, P_0, \overbrace{P_f}^{\overset{\text{wlog}}{=P}}, \delta_i, \delta_c, \delta_r \rangle$.
We construct the DNWA $\mathcal{B} = \langle Q', q_0', Q_f', P', p_0', P_f', \delta_i', \delta_c', \delta_r' \rangle$:

- $Q' := 2^{Q \times Q} = \{S_1, \ldots, S_i\}$
- $q_0' := Q_0 \times Q_0$
- $Q_f' := \{S \mid \exists q, q'.(q, q') \in S \wedge q' \in Q_f\}$
  or: $S \in Q_f' :\Leftrightarrow S$ contains (q, q') with $q' \in Q_f$

The states: definition

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

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

The states: definition

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

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

The states: definition

Consider the NNWA $\mathcal{A} = \langle Q, Q_0, Q_f, P, P_0, \overbrace{P_f}^{\stackrel{\text{wlog}}{=} P}, \delta_i, \delta_c, \delta_r \rangle$.
We construct the DNWA $\mathcal{B} = \langle Q', q_0', Q_f', P', p_0', P_f', \delta_i', \delta_c', \delta_r' \rangle$:

- $Q' := 2^{Q \times Q} = \{S_1, \dots, S_i\}$
- $q_0' := Q_0 \times Q_0$
- $Q_f' := \{S \mid \exists q, q'.(q, q') \in S \wedge q' \in Q_f\}$
  or: $S \in Q_f' :\Leftrightarrow S$ 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

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

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, $\mathcal{B}$ 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 $q \xrightarrow{n_i}_{\mathcal{A}} q'$.

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, $\mathcal{B}$ 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 $q \xrightarrow{n_i}_{\mathcal{A}} q'$.

Question: acceptance condition of $\mathcal{B}$ 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

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 $\langle c_i$.

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

Question: acceptance condition of $\mathcal{B}$ 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

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

Question: acceptance condition of $\mathcal{B}$ for n?
Answer: $S_{k+1} \in Q'_f$,
i.e., $\exists q, q'.(q, q') \in S_{k+1} \wedge q \xrightarrow{n_{k+1}}_{\mathcal{A}} q' \wedge q' \in Q_f$

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

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

$\delta_i'(S_{k+1}, i) =$

Internal transitions

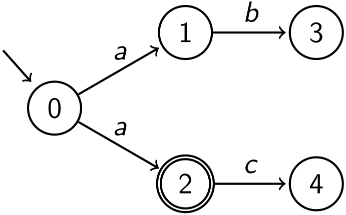
- I After reading n, $\mathcal{B}$ will be in state S_{k+1}, where (S_i, c_i) will be the hierarchical state for each $\langle c_i$.
- II S_i contains the pair (q, q') iff $q \xrightarrow{n_i}_{\mathcal{A}} q'$.

$$q \xrightarrow{n_{k+1}} q' \xrightarrow{i} q''$$

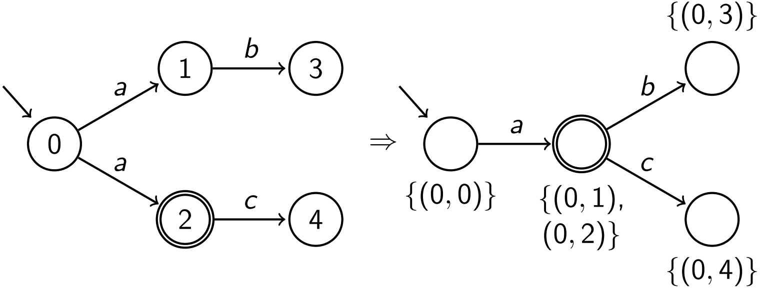
$$n' = n \cdot i = n_1\langle c_1 n_2\langle c_2 \cdots n_k\langle c_k n_{k+1} i$$

$$\delta_i'(S_{k+1}, i) = \{(q, q'') \mid (q, q') \in S_{k+1} \land q'' \in \delta_i(q', i)\}$$

Example



Example



Call 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 $\langle c_i$.

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

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

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

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

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

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

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

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

Call transitions

I. After reading n, $\mathcal{B}$ 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 $q \xrightarrow{n_i}_{\mathcal{A}} q'$.

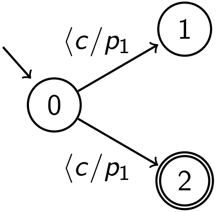
$$\begin{tikzcd} q \arrow[r, "n_{k+1}"] & q' \arrow[d, "c_{k+1}/p"'] \\ & q'' \end{tikzcd}$$

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

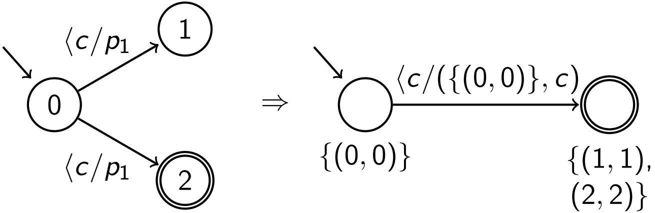
$$\delta'_c(S_{k+1}, c_{k+1}) = (S', (S_{k+1}, c_{k+1})),$$
$$S' = \{(q'', q'') \mid (q, q') \in S_{k+1} \wedge \exists p \in P.(q'', p) \in \delta_c(q', c_{k+1})\}$$

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

Example



Example



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 $\langle c_i$.

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

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

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, $\mathcal{B}$ 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 $q \xrightarrow{n_i}_{\mathcal{A}} q'$.

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

$$\begin{tikzcd} & q'' \\ q \arrow[r, "n_{k+1}"'] & q' \arrow[u, "r/p"] \end{tikzcd}$$

We have two cases here:

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

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

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 $\langle c_i$.

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

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

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 $= (S_k, c_k)$

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

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 $\langle c_i$.

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

$$\begin{tikzcd} q \arrow[r, "n_k"] & q' \arrow[r, dashed] \arrow[d, "c_k/p"'] & q'' \\ & q_1 \arrow[r, "n_{k+1}"] & q_2 \arrow[u, "r/p"'] \end{tikzcd}$$

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

We have two cases here:

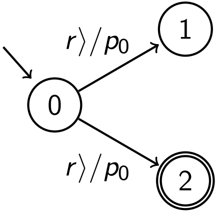
$k = 0$ no matching call, like internal transition

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

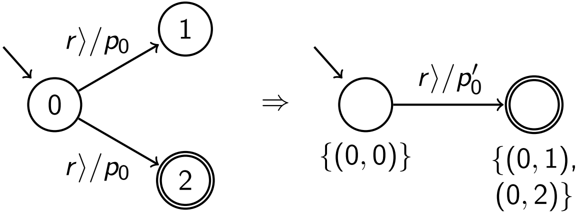
$k > 0$ subword $n_k\langle c_k n_{k+1} r\rangle$, hierarchical state $= (S_k, c_k)$

$$\delta'_r(S_{k+1}, (S_k, c_k), r) = \{(q, q'') \mid (q, q') \in S_k \wedge (q_1, q_2) \in S_{k+1} \wedge \exists p \in P . (q_1, p) \in \delta_c(q', c_k) \wedge q'' \in \delta_r(q_2, p, r)\}$$

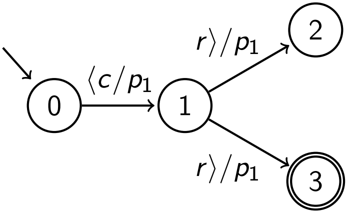
Example



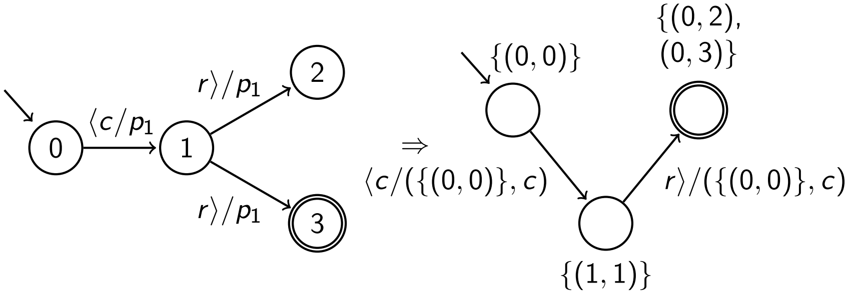
Example



Example



Example



Résumé

- now all components of $\mathcal{B}$ defined

Résumé

- now all components of $\mathcal{B}$ defined
- correctness results from invariants

Résumé

- now all components of $\mathcal{B}$ 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

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 $\mathcal{O}(2^{s^2})$

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 $\mathcal{O}(2^{s^2})$
- many practical problems describable as nested words

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 $\mathcal{O}(2^{s^2})$
- many practical problems describable as nested words
- recent concept, time will show the relevance

References

Rajeev Alur and Parthasarathy Madhusudan.
Adding Nested Structure to Words.
In *Journal of the ACM*, 2009.

Rajeev Alur and Parthasarathy Madhusudan.
Visibly Pushdown Languages.
In *STOC '04*, 2004.