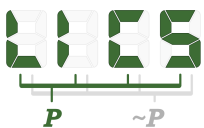
The complexity of downward closures of indexed languages

Richard Mandel **Corto Mascle** Georg Zetzsche

Max Planck Institute for Software Systems, Kaiserslautern



Downward-closures

u is a *subword* of v ($u \preceq v$) if we can remove letters of u to get v.

Downward-closures

u is a *subword* of v ($u \preceq v$) if we can remove letters of u to get v.
Ex: $bab \preceq a\underline{b}b\underline{ab}a$.

Downward-closures

u is a *subword* of v ($u \preceq v$) if we can remove letters of u to get v.
Ex: $bab \preceq a\underline{b}b\underline{ab}a$.
The *downward-closure* of a language L is the set of all subwords of words in L.

Downward-closures

u is a *subword* of v ($u \preceq v$) if we can remove letters of u to get v.
Ex: $bab \preceq a\underline{b}b\underline{a}ba$.
The *downward-closure* of a language L is the set of all subwords of words in L.

Theorem (Corollary of [Higman '52])

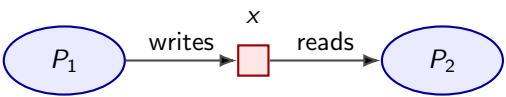
The downward-closure of a language $L \subseteq \Sigma^*$ is always regular.

Downward-closures

u is a *subword* of v ($u \preceq v$) if we can remove letters of u to get v.
Ex: $bab \preceq a\underline{b}b\underline{a}ba$.
The *downward-closure* of a language L is the set of all subwords of words in L.

Theorem (Corollary of [Higman '52])

The downward-closure of a language $L \subseteq \Sigma^*$ is always regular.

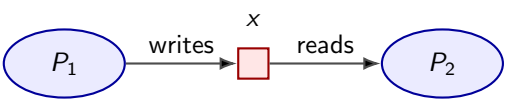


Downward-closures

u is a *subword* of v ($u \preceq v$) if we can remove letters of u to get v.
Ex: $bab \preceq a\underline{b}b\underline{a}ba$.
The *downward-closure* of a language L is the set of all subwords of words in L.

Theorem (Corollary of [Higman '52])

The downward-closure of a language $L \subseteq \Sigma^*$ is always regular.



- Verification of distributed systems with thread pools
- Separation by piecewise-testable languages
- Lossy FIFO machines

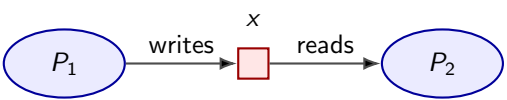
Downward-closures

u is a *subword* of v ($u \preceq v$) if we can remove letters of u to get v.
Ex: $bab \preceq a\underline{b}b\underline{a}ba$.
The *downward-closure* of a language L is the set of all subwords of words in L.

Theorem (Corollary of [Higman '52])

The downward-closure of a language $L \subseteq \Sigma^*$ is always regular.

Not effective!



- Verification of distributed systems with thread pools
- Separation by piecewise-testable languages
- Lossy FIFO machines

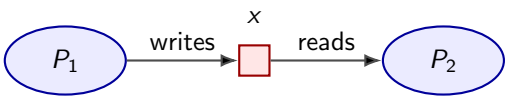
Downward-closures

u is a *subword* of v ($u \preceq v$) if we can remove letters of u to get v.
Ex: $bab \preceq a\underline{b}b\underline{a}ba$.
The *downward-closure* of a language L is the set of all subwords of words in L.

Theorem (Corollary of [Higman '52])

The downward-closure of a language $L \subseteq \Sigma^*$ is always regular.

Not effective!



- Verification of distributed systems with thread pools
- Separation by piecewise-testable languages
- Lossy FIFO machines

Given a class of languages $\mathcal{C}$, is the downward-closure *computable* for these languages? How large is the resulting automaton?

Computing downward closures

Downward-closures are computable for:

- **Context-free languages** [van Leeuwen '78]
- **Petri net languages** [Habermehl, Meyer, Wimmel '10]

Computing downward closures

Downward-closures are computable for:

- **Context-free languages** [van Leeuwen '78]
- **Petri net languages** [Habermehl, Meyer, Wimmel '10]

[Zetzsche '15]

New method to compute downward closures.

Computing downward closures

Downward-closures are computable for:

- **Context-free languages** [van Leeuwen '78]
- **Petri net languages** [Habermehl, Meyer, Wimmel '10]

[Zetzsche '15]

New method to compute downward closures.

- **Indexed languages** [Zetzsche '15]
- **Higher-order pushdown automata** [Hague, Kochems, Ong '16]
- **Higher-order recursion schemes** [Clemente, Parys, Salvati, Walukiewicz '16]

Computing downward closures

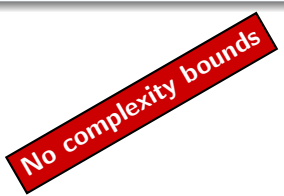
Downward-closures are computable for:

- **Context-free languages** [van Leeuwen '78]
- **Petri net languages** [Habermehl, Meyer, Wimmel '10]

[Zetzsche '15]

New method to compute downward closures.

- **Indexed languages** [Zetzsche '15]
- **Higher-order pushdown automata** [Hague, Kochems, Ong '16]
- **Higher-order recursion schemes** [Clemente, Parys, Salvati, Walukiewicz '16]



Computing downward closures

Downward-closures are computable for:

- **Context-free languages** [van Leeuwen '78]
- **Petri net languages** [Habermehl, Meyer, Wimmel '10]

[Zetzsche '15]

New method to compute downward closures.

- **Indexed languages** [Zetzsche '15]
- **Higher-order pushdown automata** [Hague, Kochems, Ong '16]
- **Higher-order recursion schemes** [Clemente, Parys, Salvati, Walukiewicz '16]

No complexity bounds

Problem

Given a system $\mathcal{S}$ of size k in class $\mathcal{C}$, how large can an automaton recognising $L(\mathcal{S})\downarrow$ be?

Indexed grammars [Aho '68]

Indexed grammar= Context-free grammar where each non-terminal carries a stack.
N set of *non-terminals*, T set of *terminals*, Γ set of *stack symbols*.

Indexed grammars [Aho '68]

Indexed grammar= Context-free grammar where each non-terminal carries a stack.
N set of *non-terminals*, T set of *terminals*, Γ set of *stack symbols*.

$$
\begin{aligned}
A &\to w \in T^* && \textit{terminal} \\
A &\to BC && \textit{copy} \\
A &\to B\gamma && \textit{push} \\
A\gamma &\to B && \textit{pop}
\end{aligned}
$$

Indexed grammars : Example

$$S \to S\gamma_\perp$$
$$S \to S\gamma_a$$
$$S \to S\gamma_b$$
$$S \to WW$$
$$W\gamma_a \to Wa$$
$$W\gamma_b \to Wb$$
$$W\gamma_\perp \to \varepsilon$$

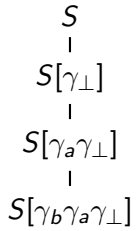
Indexed grammars : Example

S

$S \to S\gamma_\perp$
$S \to S\gamma_a$
$S \to S\gamma_b$
$S \to WW$
$W\gamma_a \to Wa$
$W\gamma_b \to Wb$
$W\gamma_\perp \to \varepsilon$

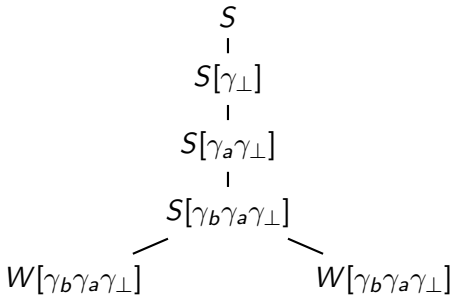
Indexed grammars : Example

$S \to S\gamma_\perp$
$S \to S\gamma_a$
$S \to S\gamma_b$
$S \to WW$
$W\gamma_a \to Wa$
$W\gamma_b \to Wb$
$W\gamma_\perp \to \varepsilon$



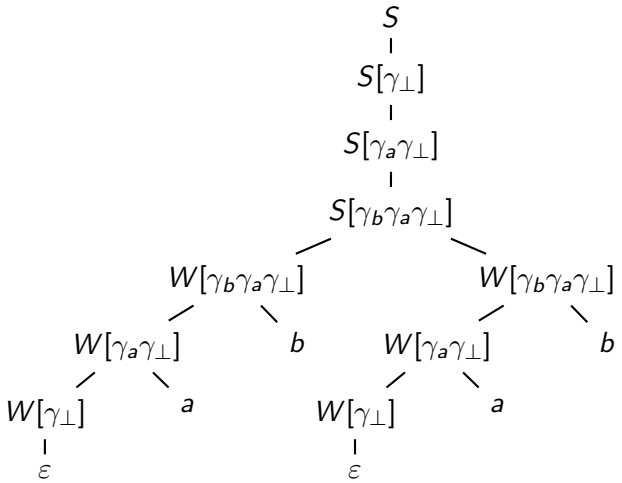
Indexed grammars : Example

$S \to S\gamma_\perp$
$S \to S\gamma_a$
$S \to S\gamma_b$
$S \to WW$
$W\gamma_a \to Wa$
$W\gamma_b \to Wb$
$W\gamma_\perp \to \varepsilon$



Indexed grammars : Example

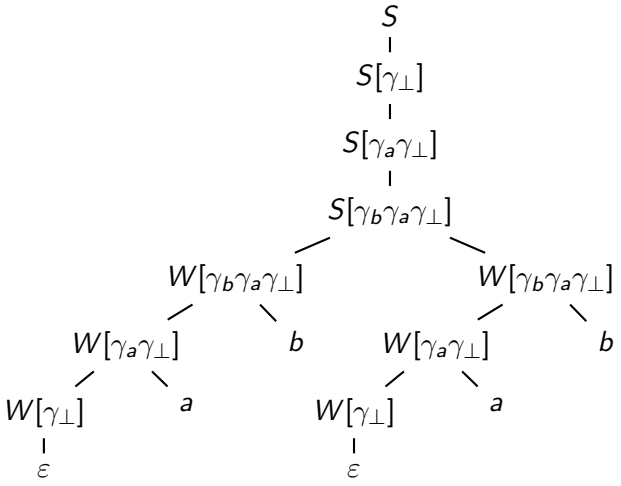
$S \to S\gamma_\perp$
$S \to S\gamma_a$
$S \to S\gamma_b$
$S \to WW$
$W\gamma_a \to Wa$
$W\gamma_b \to Wb$
$W\gamma_\perp \to \varepsilon$



Indexed grammars : Example

$$S \to S\gamma_\perp$$
$$S \to S\gamma_a$$
$$S \to S\gamma_b$$
$$S \to WW$$
$$W\gamma_a \to Wa$$
$$W\gamma_b \to Wb$$
$$W\gamma_\perp \to \varepsilon$$

$$\{ww \mid w \in \{a, b\}^*\}$$

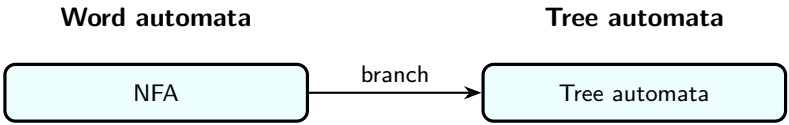


Higher-order pushdown automata

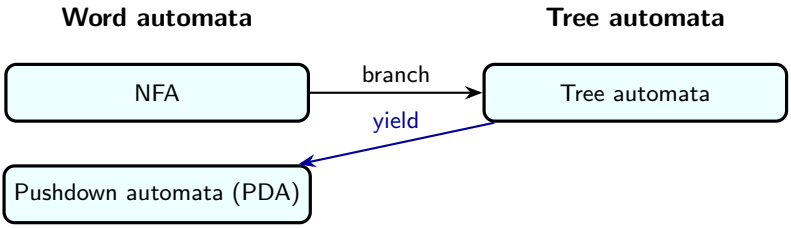
Word automata

Tree automata

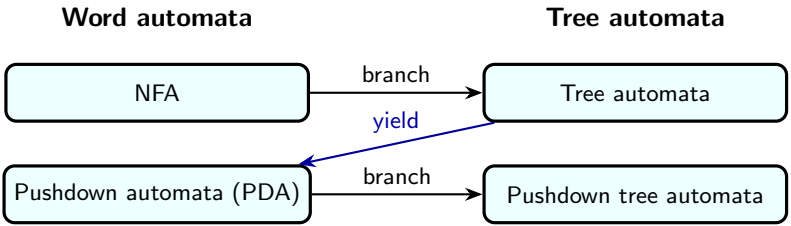
Higher-order pushdown automata



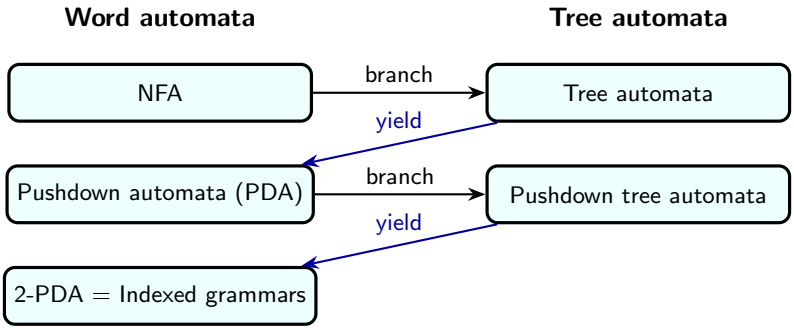
Higher-order pushdown automata



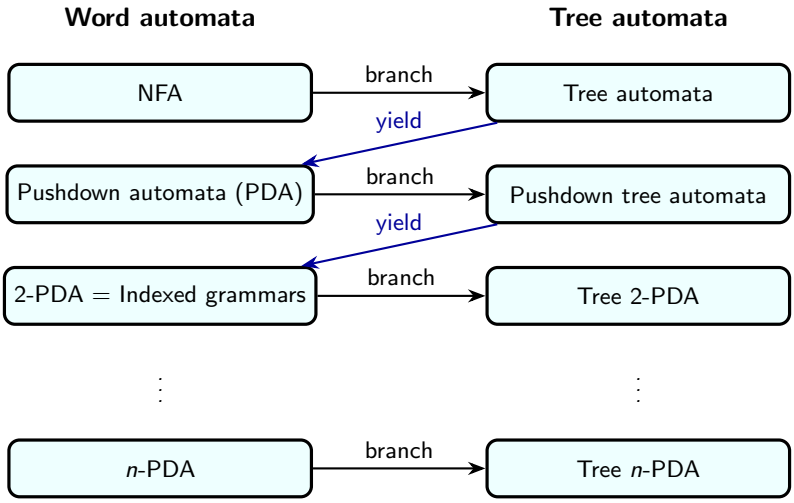
Higher-order pushdown automata



Higher-order pushdown automata



Higher-order pushdown automata



Our goal

Give tight bounds on the computation of downward-closures of indexed languages.

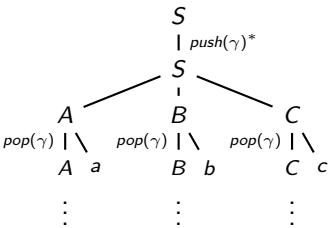
Our goal

Give tight bounds on the computation of downward-closures of indexed languages.

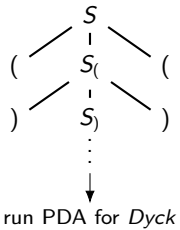
What can indexed grammars compute?

Examples

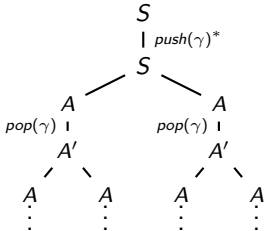
$\{a^n b^n c^n \mid n \in \mathbb{N}\}$



$\{uu^R \mid u \in Dyck_n\}$

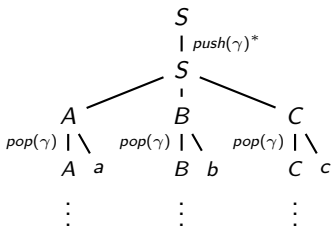


$\{a^{2^n} \mid n \in \mathbb{N}\}$

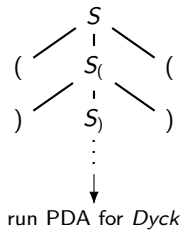


Examples

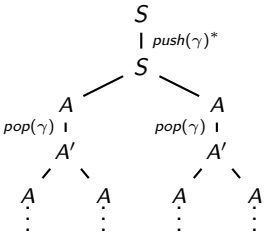
$\{a^n b^n c^n \mid n \in \mathbb{N}\}$



$\{uu^R \mid u \in Dyck_n\}$



$\{a^{2^n} \mid n \in \mathbb{N}\}$



$L \subseteq \{a, \wedge, \vee, \neg, (,)\}^*$ set of satisfiable SAT formulas, with variable x_k encoded as a^k

Indexed grammars : Counting

[Aho '68]

If $\mathcal{G}$ accepts a word then it accepts a word of at most 3-exponential size.

Indexed grammars : Counting

[Aho '68]

If $\mathcal{G}$ accepts a word then it accepts a word of at most 3-exponential size.

[Hayashi '73]

If $\mathcal{G}$ accepts a word of more than 3-exponential size then its language is infinite.

Indexed grammars : Counting

[Aho '68]

If $\mathcal{G}$ accepts a word then it accepts a word of at most 3-exponential size.

[Hayashi '73]

If $\mathcal{G}$ accepts a word of more than 3-exponential size then its language is infinite.

This work

There is $\mathcal{G}_k$ of size $\mathcal{O}(k)$ accepting the language $\{a^{2^{2^{2^k}}}\}$.

Indexed grammars : Counting

[Aho '68]

If $\mathcal{G}$ accepts a word then it accepts a word of at most 3-exponential size.

[Hayashi '73]

If $\mathcal{G}$ accepts a word of more than 3-exponential size then its language is infinite.

This work

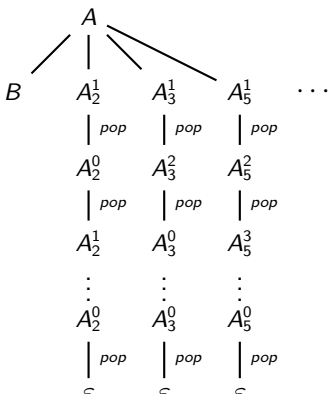
There is $\mathcal{G}_k$ of size $\mathcal{O}(k)$ accepting the language $\{a^{2^{2^{2^k}}}\}$.

Corollary

There is an indexed grammar whose downward closure requires a 3-exponential NFA.

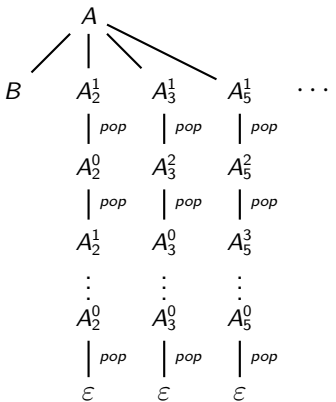
3-exponential construction

Check stack height



3-exponential construction

Check stack height

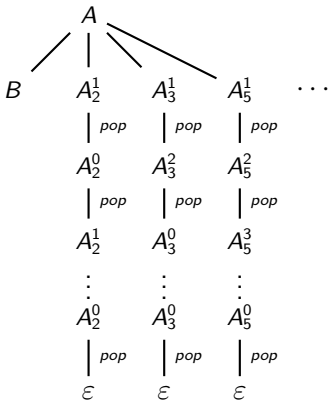


Binary counter

A
| $pop(1)^*$
B
| $pop(0)$
C
| $push(1)$
D
| $push(0)^*$
E
| check stack height 2^n
A
⋮

3-exponential construction

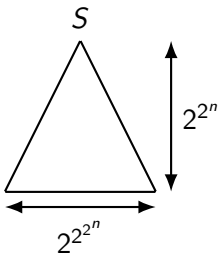
Check stack height



Binary counter

A
| $pop(1)^*$
B
| $pop(0)$
C
| $push(1)$
D
| $push(0)^*$
E
| check stack height 2^n
A
⋮

Branching



Upper bounds

Theorem [van Leeuwen '78, Courcelle '91]

The downward-closure of the language of a CFG is recognised by an NFA of exponential size.

Main result

The downward-closure of an indexed language is recognised by a CFG of 2-exponential size.

Corollary

The downward-closure of an indexed language is recognised by an NFA of 3-exponential size.

Upper bounds

Theorem [van Leeuwen '78, Courcelle '91]

The downward-closure of the language of a CFG is recognised by an NFA of exponential size.

Main result

The downward-closure of an indexed language is recognised by a CFG of 2-exponential size.

Corollary

The downward-closure of an indexed language is recognised by an NFA of 3-exponential size.

Upper bounds

Theorem [van Leeuwen '78, Courcelle '91]

The downward-closure of the language of a CFG is recognised by an NFA of exponential size.

Main result

The downward-closure of an indexed language is recognised by a CFG of 2-exponential size.

Corollary

The downward-closure of an indexed language is recognised by an NFA of 3-exponential size.

Upper bounds

Theorem [van Leeuwen '78, Courcelle '91]

The downward-closure of the language of a CFG is recognised by an NFA of exponential size.

Main result

The downward-closure of an indexed language is recognised by a CFG of 2-exponential size.

Corollary

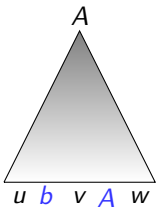
The downward-closure of an indexed language is recognised by an NFA of 3-exponential size.

A construction for context-free grammars

Step 1: Add new rules

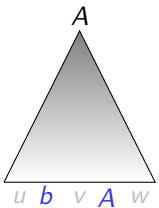
A construction for context-free grammars

Step 1: Add new rules



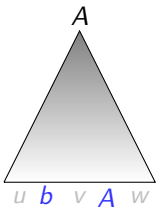
A construction for context-free grammars

Step 1: Add new rules



A construction for context-free grammars

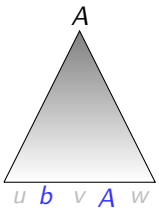
Step 1: Add new rules



$\Gamma_l(A) = \{b \mid A \text{ derives } w \in \Sigma^* b \Sigma^* A \Sigma^*\}$.
$\Gamma_r(A) = \{b \mid A \text{ derives } w \in \Sigma^* A \Sigma^* b \Sigma^*\}$.

A construction for context-free grammars

Step 1: Add new rules

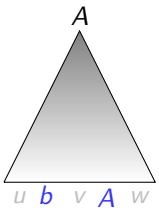


$\Gamma_l(A) = \{b \mid A \text{ derives } w \in \Sigma^* b \Sigma^* A \Sigma^*\}$.
$\Gamma_r(A) = \{b \mid A \text{ derives } w \in \Sigma^* A \Sigma^* b \Sigma^*\}$.

New rule: $A \to xAy$ for all $x \in \Gamma_l(A)^*, y \in \Gamma_r(A)^*$

A construction for context-free grammars

Step 1: Add new rules

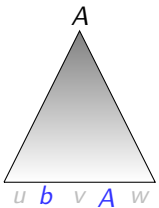


$\Gamma_l(A) = \{b \mid A \text{ derives } w \in \Sigma^* b \Sigma^* A \Sigma^*\}$.
$\Gamma_r(A) = \{b \mid A \text{ derives } w \in \Sigma^* A \Sigma^* b \Sigma^*\}$.

New rule: $A \to xAy$ for all $x \in \Gamma_l(A)^*, y \in \Gamma_r(A)^*$
Preserves the downward-closure.

A construction for context-free grammars

Step 1: Add new rules



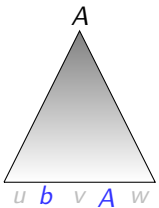
$\Gamma_l(A) = \{b \mid A \text{ derives } w \in \Sigma^* b \Sigma^* A \Sigma^*\}$.
$\Gamma_r(A) = \{b \mid A \text{ derives } w \in \Sigma^* A \Sigma^* b \Sigma^*\}$.

New rule: $A \to xAy$ for all $x \in \Gamma_l(A)^*, y \in \Gamma_r(A)^*$
Preserves the downward-closure.

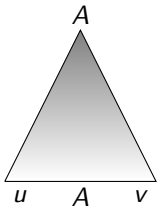
Step 2: Simulation with an NFA

A construction for context-free grammars

Step 1: Add new rules



Step 2: Simulation with an NFA

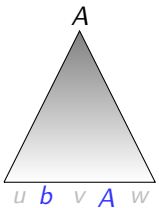


$\Gamma_l(A) = \{b \mid A \text{ derives } w \in \Sigma^* b \Sigma^* A \Sigma^*\}$.
$\Gamma_r(A) = \{b \mid A \text{ derives } w \in \Sigma^* A \Sigma^* b \Sigma^*\}$.

New rule: $A \to xAy$ for all $x \in \Gamma_l(A)^*, y \in \Gamma_r(A)^*$
Preserves the downward-closure.

A construction for context-free grammars

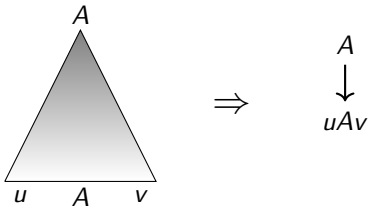
Step 1: Add new rules



$\Gamma_l(A) = \{b \mid A \text{ derives } w \in \Sigma^* b \Sigma^* A \Sigma^*\}$.
$\Gamma_r(A) = \{b \mid A \text{ derives } w \in \Sigma^* A \Sigma^* b \Sigma^*\}$.

New rule: $A \to xAy$ for all $x \in \Gamma_l(A)^*, y \in \Gamma_r(A)^*$
Preserves the downward-closure.

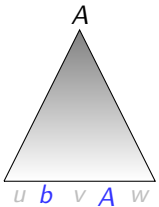
Step 2: Simulation with an NFA



We can restrict our attention to derivations *of bounded height*.

A construction for context-free grammars

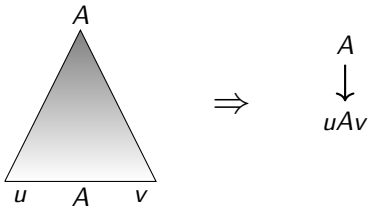
Step 1: Add new rules



$\Gamma_l(A) = \{b \mid A \text{ derives } w \in \Sigma^* b \Sigma^* A \Sigma^*\}$.
$\Gamma_r(A) = \{b \mid A \text{ derives } w \in \Sigma^* A \Sigma^* b \Sigma^*\}$.

New rule: $A \to xAy$ for all $x \in \Gamma_l(A)^*, y \in \Gamma_r(A)^*$
Preserves the downward-closure.

Step 2: Simulation with an NFA



We can restrict our attention to derivations *of bounded height*.

Those can be simulated by a finite-state automaton.

Upper bounds

Theorem [van Leeuwen '78, Courcelle '91]

The downward-closure of the language of a CFG is recognised by an NFA of exponential size.

Main result

The downward-closure of an indexed language is recognised by a CFG of 2-exponential size.

Upper bounds

Theorem [van Leeuwen '78, Courcelle '91]

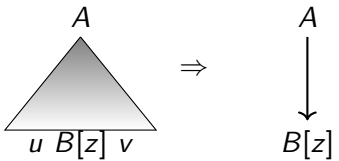
The downward-closure of the language of a CFG is recognised by an NFA of exponential size.

Main result

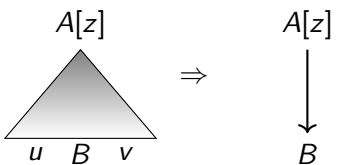
The downward-closure of an indexed language is recognised by a CFG of 2-exponential size.

Step 1: New rules for indexed grammars

If A can produce $uB[z]v$ then $A \to B[z]$ is a sound rule.



If $A[z]$ can produce uBv then $A[z] \to B$ is a sound rule.



Step 1: New rules for indexed grammars

A stack content $z \in \Gamma^*$ is mapped to Boolean matrices $M_{push}(z), M_{pop}(z) \in \{\top, \bot\}^{N \times N}$.

Step 1: New rules for indexed grammars

A stack content $z \in \Gamma^*$ is mapped to Boolean matrices $M_{push}(z), M_{pop}(z) \in \{\top, \bot\}^{N \times N}$.

$M_{push}(z)(A, B) = \top$ if and only if A derives $uB[z]v$ for some u, v.

Step 1: New rules for indexed grammars

A stack content $z \in \Gamma^*$ is mapped to Boolean matrices $M_{push}(z), M_{pop}(z) \in \{\top, \bot\}^{N \times N}$.

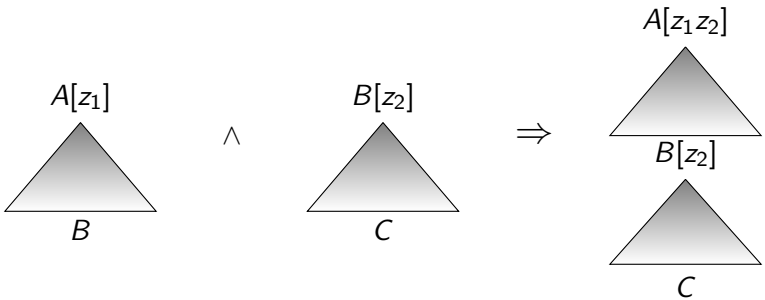
$M_{push}(z)(A, B) = \top$ if and only if A derives $uB[z]v$ for some u, v.
$M_{pop}(z)(A, B) = \top$ if and only if $A[z]$ derives uBv for some u, v.

Step 1: New rules for indexed grammars

A stack content $z \in \Gamma^*$ is mapped to Boolean matrices $M_{push}(z), M_{pop}(z) \in \{\top, \bot\}^{N \times N}$.

$M_{push}(z)(A, B) = \top$ if and only if A derives $uB[z]v$ for some u, v.
$M_{pop}(z)(A, B) = \top$ if and only if $A[z]$ derives uBv for some u, v.

M_{push} and M_{pop} are **morphisms**: $M_{pop}(z_1)M_{pop}(z_2) = M_{pop}(z_1 z_2)$.



Step 2: Simulation by a context-free grammar

Imagine we have z so that for all B,
$M_{push}(B, B) = \top$ and $M_{pop}(B, B) = \top$.

$A[z]$

Step 2: Simulation by a context-free grammar

Imagine we have z so that for all B,
$M_{push}(B, B) = \top$ and $M_{pop}(B, B) = \top$.

$A[z]$

Every non-terminal can push and pop z for free.

Step 2: Simulation by a context-free grammar

Imagine we have z so that for all B,
$M_{push}(B, B) = \top$ and $M_{pop}(B, B) = \top$.

$A[z]$

Every non-terminal can push and pop z for free.

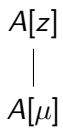
Let $\mu = (M_{push}(z), M_{pop}(z))$

Step 2: Simulation by a context-free grammar

Imagine we have z so that for all B,
$M_{push}(B, B) = \top$ and $M_{pop}(B, B) = \top$.

Every non-terminal can push and pop z for free.

Let $\mu = (M_{push}(z), M_{pop}(z))$
Then we can *fold* z: $A[z] \to A[\mu]$.

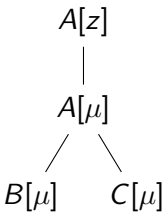


Step 2: Simulation by a context-free grammar

Imagine we have z so that for all B,
$M_{push}(B, B) = \top$ and $M_{pop}(B, B) = \top$.

Every non-terminal can push and pop z for free.

Let $\mu = (M_{push}(z), M_{pop}(z))$
Then we can *fold* z: $A[z] \to A[\mu]$.



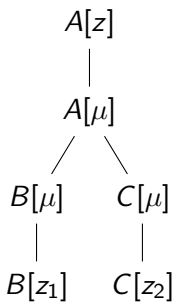
Step 2: Simulation by a context-free grammar

Imagine we have z so that for all B,
$M_{push}(B, B) = \top$ and $M_{pop}(B, B) = \top$.

Every non-terminal can push and pop z for free.

Let $\mu = (M_{push}(z), M_{pop}(z))$
Then we can *fold* z: $A[z] \to A[\mu]$.

We can also *unfold* μ: $B[\mu] \to B[z']$ whenever
$(M_{push}(z'), M_{pop}(z')) = \mu$



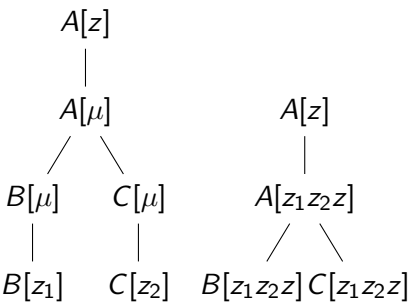
Step 2: Simulation by a context-free grammar

Imagine we have z so that for all B,
$M_{push}(B, B) = \top$ and $M_{pop}(B, B) = \top$.

Every non-terminal can push and pop z for free.

Let $\mu = (M_{push}(z), M_{pop}(z))$
Then we can *fold* z: $A[z] \to A[\mu]$.

We can also *unfold* μ: $B[\mu] \to B[z']$ whenever
$(M_{push}(z'), M_{pop}(z')) = \mu$



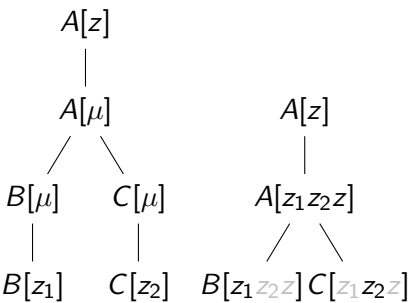
Step 2: Simulation by a context-free grammar

Imagine we have z so that for all B,
$M_{push}(B, B) = \top$ and $M_{pop}(B, B) = \top$.

Every non-terminal can push and pop z for free.

Let $\mu = (M_{push}(z), M_{pop}(z))$
Then we can *fold* z: $A[z] \to A[\mu]$.

We can also *unfold* μ: $B[\mu] \to B[z']$ whenever
$(M_{push}(z'), M_{pop}(z')) = \mu$



Step 2: Simulation by a context-free grammar

Adapted from [Gimbert, M., Totzke '26]

We can define *summaries* representing "folded" words so that:

- Summaries have at most exponential size
- Push and pop can be implemented on summaries

Step 2: Simulation by a context-free grammar

Adapted from [Gimbert, M., Totzke '26]

We can define *summaries* representing "folded" words so that:

- Summaries have at most exponential size
- Push and pop can be implemented on summaries

$$
\begin{aligned}
&A \to w \in T^* \\
&A \to BC \\
&A \to B\gamma \\
&A\gamma \to B
\end{aligned}
\qquad \Rightarrow \qquad
\begin{aligned}
&A[\sigma] \to w \in T^* \\
&A[\sigma] \to B[\sigma]C[\sigma] \\
&A[\sigma] \to B[\mathtt{push}_\gamma(\sigma)] \\
&A[\sigma] \to B[\mathtt{pop}_\gamma(\sigma)]
\end{aligned}
$$

Step 2: Simulation by a context-free grammar

Adapted from [Gimbert, M., Totzke '26]

We can define *summaries* representing "folded" words so that:

- Summaries have at most exponential size
- Push and pop can be implemented on summaries

$$
\begin{array}{ll}
A \to w \in T^* & A[\sigma] \to w \in T^* \\
A \to BC & \Rightarrow \quad A[\sigma] \to B[\sigma]C[\sigma] \\
A \to B\gamma & A[\sigma] \to B[\mathtt{push}_\gamma(\sigma)] \\
A\gamma \to B & A[\sigma] \to B[\mathtt{pop}_\gamma(\sigma)]
\end{array}
$$

Theorem

This CFG has the same downward closure as the indexed grammar.

Theorem

Given an indexed grammar $\mathcal{G}$, we can compute a context-free grammar of 2-exponential size with the same downward closure.

Theorem

Given an indexed grammar $\mathcal{G}$, we can compute a context-free grammar of 2-exponential size with the same downward closure.

Composed with the construction for context-free grammars:

Theorem

Given an indexed grammar $\mathcal{G}$, we can compute an NFA of 3-exponential size with the same downward closure.

Future work

Theorem

Given an indexed grammar $\mathcal{G}$, we can compute an NFA of 3-exponential size with the same downward closure, and this bound is tight.

Future work

Theorem

Given an indexed grammar $\mathcal{G}$, we can compute an NFA of 3-exponential size with the same downward closure, and this bound is tight.

- Extension to higher-order pushdown automata (work in progress)
- Higher-order recursion schemes!
- Upward-closures?

Future work

Theorem

Given an indexed grammar $\mathcal{G}$, we can compute an NFA of 3-exponential size with the same downward closure, and this bound is tight.

- Extension to higher-order pushdown automata (work in progress)
- Higher-order recursion schemes!
- Upward-closures?

An open problem

Parikh image

$$w = acbbabcb$$
$$\Downarrow$$
$$\psi(w) = 2a + 4b + 2c$$

Parikh's theorem

For all context-free grammar $\mathcal{G}$ there is a regular language with the same Parikh image.

What about indexed languages?

Thank you for your attention!

Theorem

Given an indexed grammar $\mathcal{G}$, we can compute an NFA of 3-exponential size with the same downward closure, and this bound is tight.

- Extension to higher-order pushdown automata (work in progress)
- Higher-order recursion schemes!
- Upward-closures?

An open problem

Parikh image

$$w = acbbabcb$$
$$\Downarrow$$
$$\psi(w) = 2a + 4b + 2c$$

Parikh's theorem

For all context-free grammar $\mathcal{G}$ there is a regular language with the same Parikh image.

What about indexed languages?