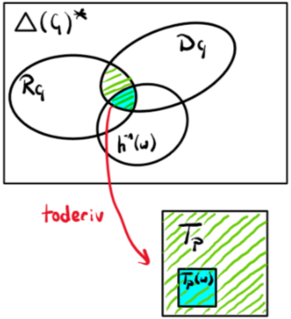
Master Thesis:
Implementation of k-best Chomksy-Schützenberger parsing for weighted multiple context-free grammars

Thomas Ruprecht

January 12, 2018

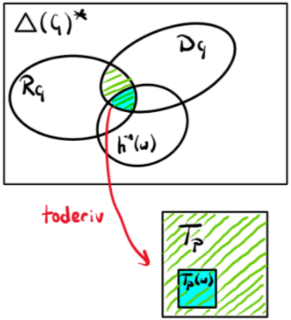
Overview

- MCFL L_G can be decomposed into $L_G = h(R \cap D)$ for
  - regular language R
  - mutliple Dyck language D
  - weighted homomorphism h



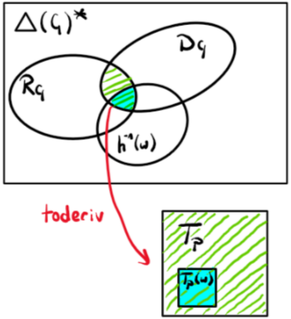
Overview

- MCFL L_G can be decomposed into $L_G = h(R \cap D)$ for
  - regular language R
  - mutliple Dyck language D
  - weighted homomorphism h
- k-best Chomksy-Schützenberger parsing, given w, G



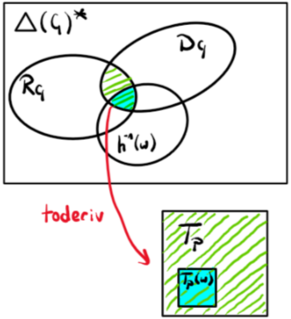
Overview

- MCFL L_G can be decomposed into $L_G = h(R \cap D)$ for
  - regular language R
  - mutliple Dyck language D
  - weighted homomorphism h
- k-best Chomksy-Schützenberger parsing, given w, G
  - weighted automaton $R(G)$ with $R = L_{R(G)}$



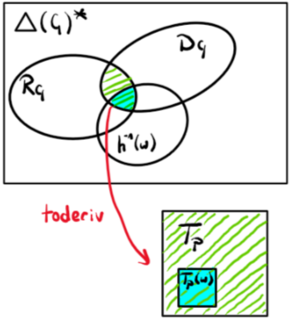
Overview

- MCFL L_G can be decomposed into $L_G = h(R \cap D)$ for
  - regular language R
  - mutliple Dyck language D
  - weighted homomorphism h
- k-best Chomksy-Schützenberger parsing, given w, G
  - weighted automaton $R(G)$ with $R = L_{R(G)}$
  - unweighted automaton $F(G, w)$ with $h^{-1}(w) = L_{F(G,w)}$



Overview

- MCFL L_G can be decomposed into $L_G = h(R \cap D)$ for
  - regular language R
  - mutliple Dyck language D
  - weighted homomorphism h
- k-best Chomksy-Schützenberger parsing, given w, G
  - weighted automaton $R(G)$ with $R = L_{R(G)}$
  - unweighted automaton $F(G, w)$ with $h^{-1}(w) = L_{F(G,w)}$
  - obtain k derivation trees:
    - enumerate best words in $L_{R(G) \circ F(G,w)}$
    - if a word is in D_G, yield $\mathrm{toderiv}_G(\delta)$



Weighted MCFG

- context-free grammar with tuples

$$
\begin{aligned}
G:\ \ & \rho_1 = S \to [x_{1,1}x_{1,2}](A) \\
& \rho_2 = A \to [a x_{1,1} b, c x_{1,2} d](A) \\
& \rho_3 = A \to [\varepsilon, \varepsilon]()
\end{aligned}
$$

Weighted MCFG

- context-free grammar with tuples
- linear composition, may delete components

$$
\begin{aligned}
G:\ \ \rho_1 &= S \to [x_{1,1}x_{1,2}](A) \\
\rho_2 &= A \to [a x_{1,1} b, c x_{1,2} d](A) \\
\rho_3 &= A \to [\varepsilon, \varepsilon]()
\end{aligned}
$$

Weighted MCFG

- context-free grammar with tuples
- linear composition, may delete components
- weight for each rule

$$\begin{aligned} G: \quad \rho_1 &= S \to [x_{1,1}x_{1,2}](A) & p(\rho_1) &= 1 \\ \rho_2 &= A \to [a x_{1,1} b, c x_{1,2} d](A) & p(\rho_2) &= 0.3 \\ \rho_3 &= A \to [\varepsilon, \varepsilon]() & p(\rho_3) &= 0.7 \end{aligned}$$

Weighted generator FSA $R(G)$

- use states as return point

$$\rho_2 = A \rightarrow \langle \quad a \qquad x_1^1 \quad b \qquad , \quad c \qquad x_1^2 \quad d \qquad \rangle(A)$$

Weighted generator FSA R(G)

- use states as return point

$$\rho_2 = A \to \langle \quad \langle^1_{\rho_2} a \langle^1_{\rho_2,1} \quad x^1_1 \quad \rangle^1_{\rho_2,1} b \rangle^1_{\rho_2} \quad , \quad \langle^2_{\rho_2} c \langle^2_{\rho_2,1} \quad x^2_1 \quad \rangle^2_{\rho_2,1} d \rangle^2_{\rho_2} \quad \rangle(A)$$

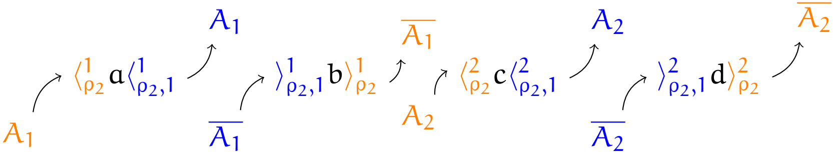
Weighted generator FSA R(G)

- use states as return point

$$\rho_2 = A \to \langle \underset{A_1}{\nearrow} \langle^1_{\rho_2} a \langle^1_{\rho_2,1} \overset{A_1}{\nearrow} x^1_1 \underset{\overline{A_1}}{\nearrow} \rangle^1_{\rho_2,1} b \rangle^1_{\rho_2} \overset{\overline{A_1}}{\nearrow} , \underset{A_2}{\nearrow} \langle^2_{\rho_2} c \langle^2_{\rho_2,1} \overset{A_2}{\nearrow} x^2_1 \underset{\overline{A_2}}{\nearrow} \rangle^2_{\rho_2,1} d \rangle^2_{\rho_2} \overset{\overline{A_2}}{\nearrow} \rangle(A)$$

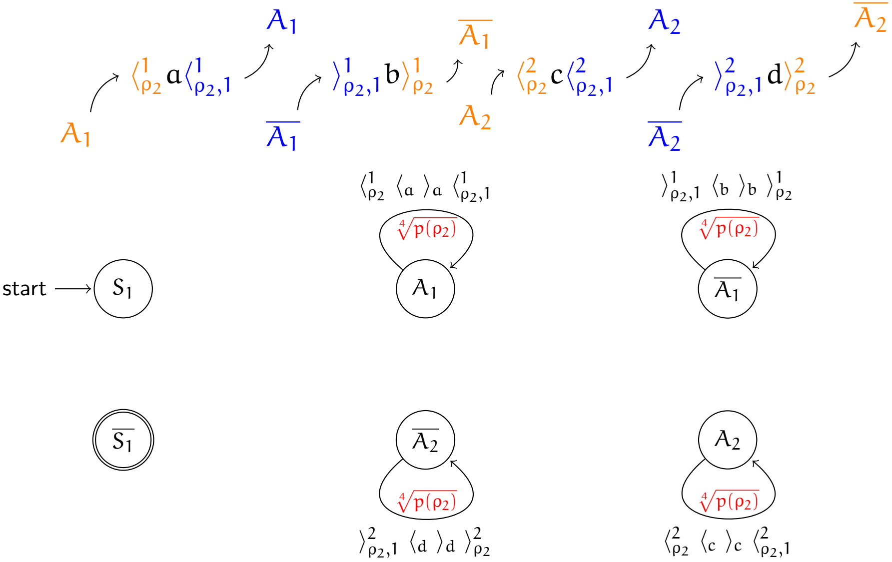
Weighted generator FSA R(G)

- use states as return point



Weighted generator FSA R(G)

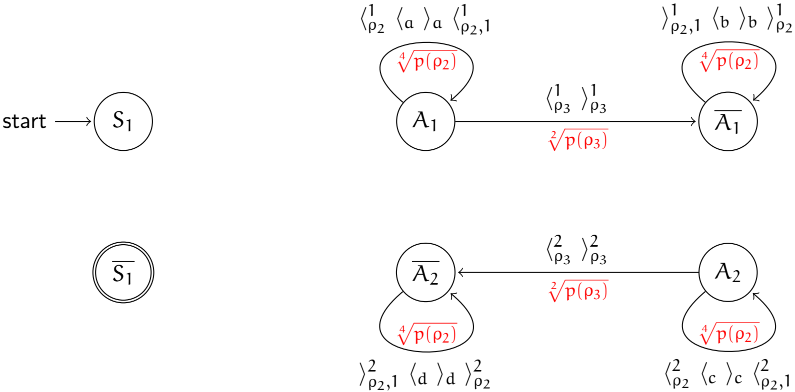
- use states as return point
- weight corresponds to rule application



Weighted generator FSA R(G)

- use states as return point
- weight corresponds to rule application

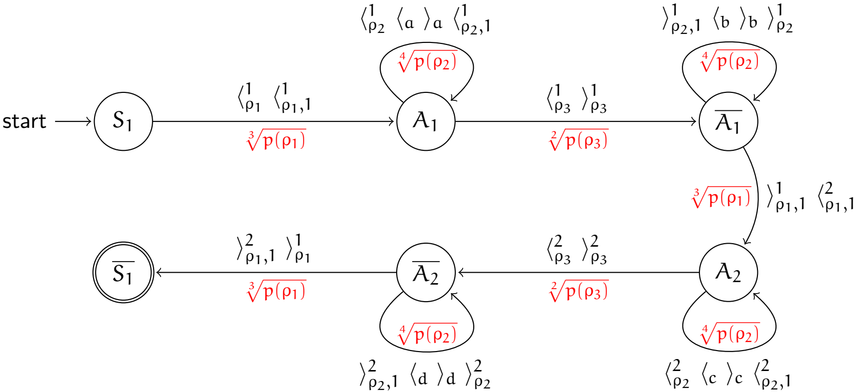
$$\rho_3 = A \to \langle \varepsilon, \varepsilon \rangle()$$



Weighted generator FSA R(G)

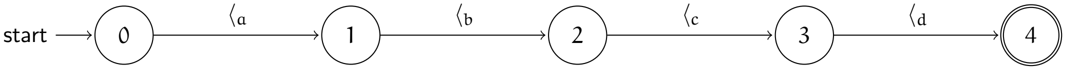
- use states as return point
- weight corresponds to rule application

$$\rho_1 = S \to \langle x_1^1 x_1^2 \rangle (A)$$



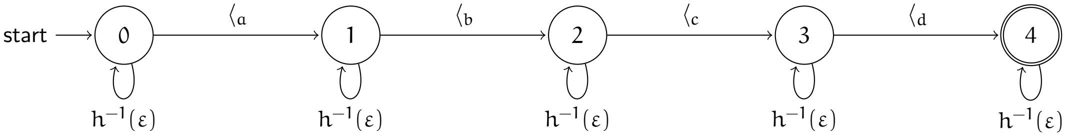
Filter FSA $F(G, abcd)$

- accept bracket words δ with $h(\delta)(abcd) \neq 0$



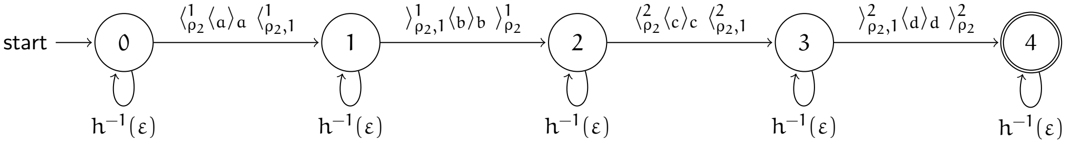
Filter FSA $F(G, abcd)$

- accept bracket words δ with $h(\delta)(abcd) \neq 0$
- $h^{-1}(\varepsilon)$ contains brackets w/o terminal symbols



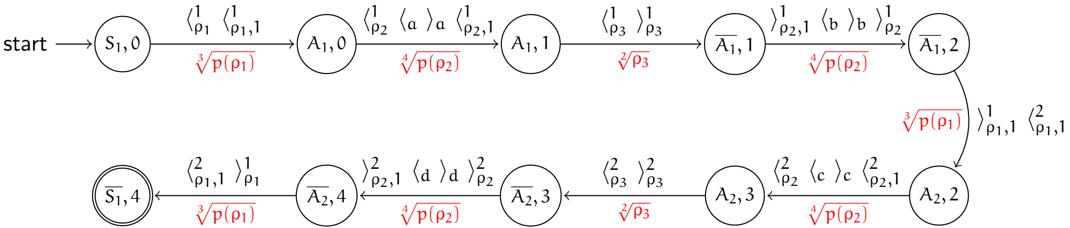
Filter FSA $F(G, abcd)$

- accept bracket words δ with $h(\delta)(abcd) \neq 0$
- $h^{-1}(\varepsilon)$ contains brackets w/o terminal symbols
- we know little context of brackets with terminals



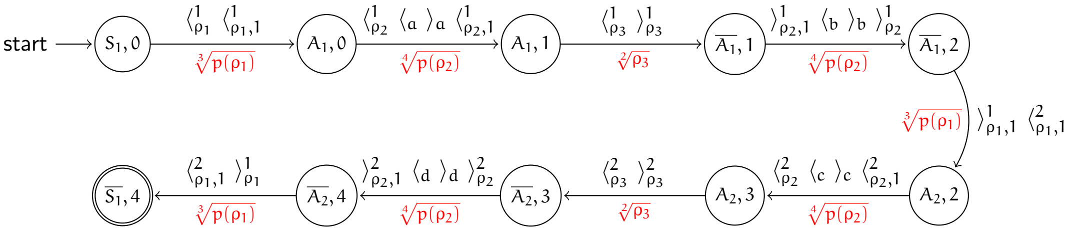
Hadamard product $R(G) \circ F(G, abcd)$

- accept words in $R \cap h^{-1}(w)$



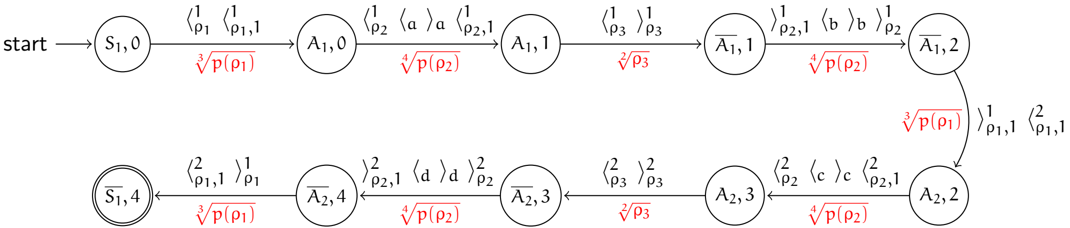
Hadamard product $R(G) \circ F(G, abcd)$

- accept words in $R \cap h^{-1}(w)$
- use weights of $R(G)$



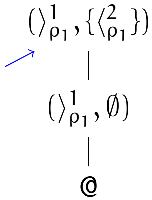
Hadamard product $R(G) \circ F(G, abcd)$

- accept words in $R \cap h^{-1}(w)$
- use weights of $R(G)$
- remove
  - unreachable transitions
  - non-productive states



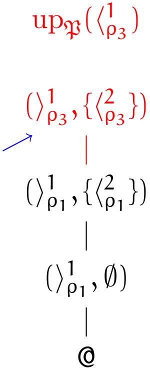
Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$
  - $\mathrm{down}(\delta)$



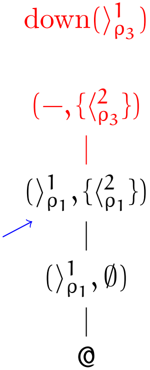
Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$
  - $\mathrm{down}(\delta)$



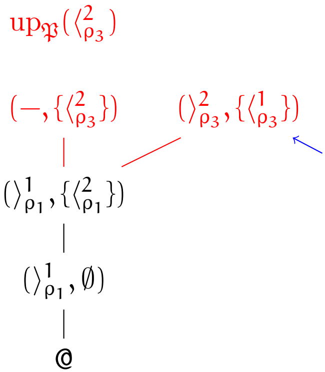
Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$
  - $\mathrm{down}(\delta)$



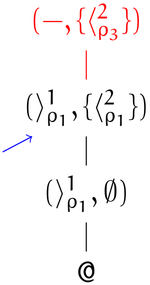
Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$
  - $\mathrm{down}(\delta)$



Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$ (nondeterministic)
  - $\mathrm{down}(\delta)$



Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$ (nondeterministic)
  - $\mathrm{down}(\delta)$

or $\mathrm{up}_{\mathfrak{P}}(\langle^{2}_{\rho_3})$

$(\rangle^{2}_{\rho_3}, \emptyset)$

$(\rangle^{1}_{\rho_1}, \{\langle^{2}_{\rho_1}\})$

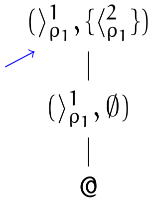
$(\rangle^{1}_{\rho_1}, \emptyset)$

@

Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$ (nondeterministic)
  - $\mathrm{down}(\delta)$

$\mathrm{down}(\rangle^2_{\rho_3})$

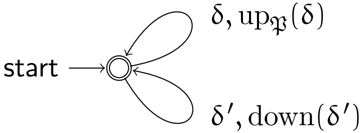


Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$ (nondeterministic)
  - $\mathrm{down}(\delta)$
- accepting configuration only contains root symbol

Recognizing D using TSA

- tree stack operations
  - $\mathrm{up}_{\mathfrak{P}}(\delta)$ (nondeterministic)
  - $\mathrm{down}(\delta)$
- accepting configuration only contains root symbol
- for each $\delta \in \Delta(G), \delta' \in \overline{\Delta(G)}$

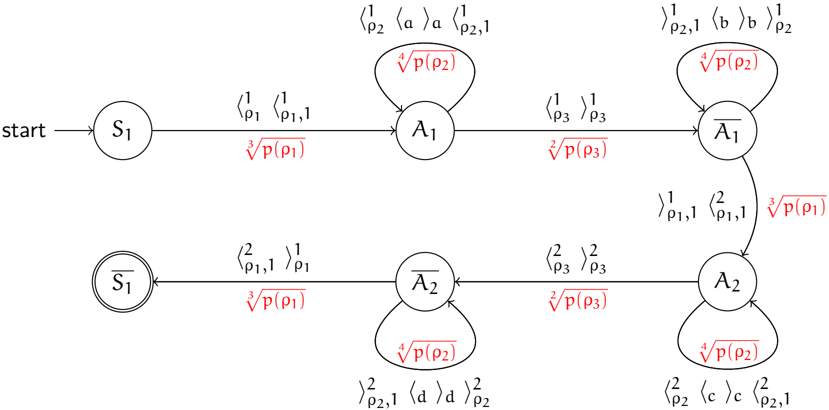


Push-down generator automaton

- there are *lots* of candidates, very few are even Dyck words

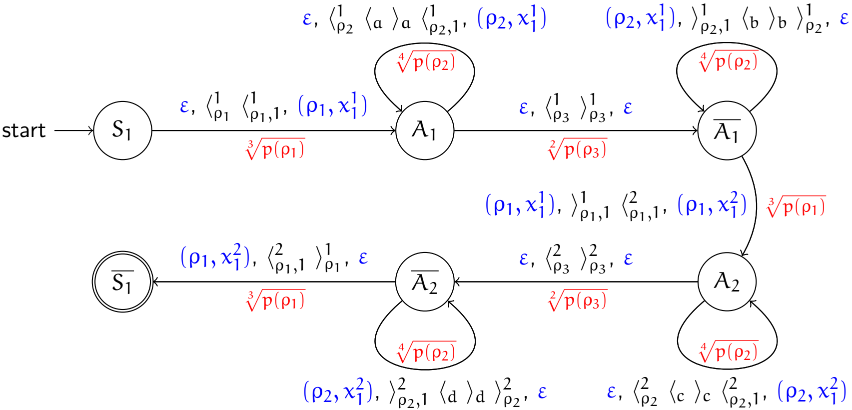
Push-down generator automaton

- there are *lots* of candidates, very few are even Dyck words
- limit R to Dyck words using PDA



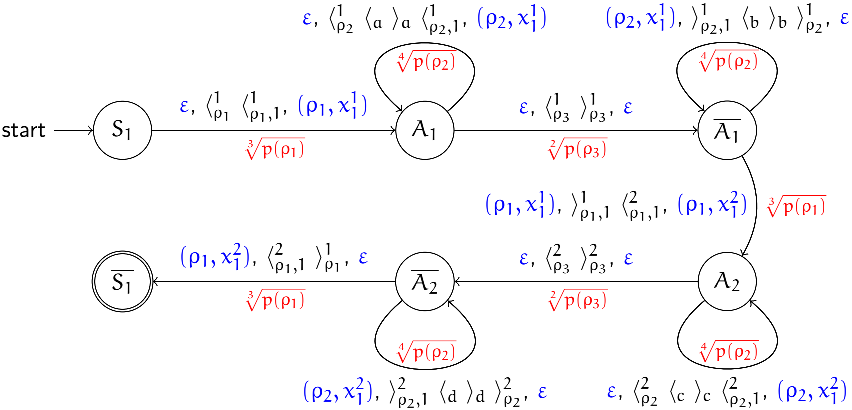
Push-down generator automaton

- there are *lots* of candidates, very few are even Dyck words
- limit R to Dyck words using PDA



Push-down generator automaton

- there are *lots* of candidates, very few are even Dyck words
- limit R to Dyck words using PDA
- superset approximation of generator PDA is generator FSA



Omit unnecessary brackets I

- each state of $\mathrm{F}(\mathrm{G})$ loops all brackets that don't contain terminals

Omit unnecessary brackets I

- each state of $F(G)$ loops all brackets that don't contain terminals
- omit all grammar rules that cannot be utilized in derivation

Omit unnecessary brackets I

- each state of $F(G)$ loops all brackets that don't contain terminals
- omit all grammar rules that cannot be utilized in derivation

$$
\begin{aligned}
P:\ & S \to [x_{1,1}x_{2,1}x_{1,2}x_{2,2}](A, B),\\
& A \to [x_{1,1}a, x_{1,2}c](A),\\
& A \to [\varepsilon, \varepsilon](),\\
& B \to [x_{1,1}b, x_{1,2}d](B),\\
& B \to [\varepsilon, \varepsilon]()
\end{aligned}
$$

P_ε :

Omit unnecessary brackets I

- each state of $F(G)$ loops all brackets that don't contain terminals
- omit all grammar rules that cannot be utilized in derivation

$$\begin{aligned} P:\ & S \to [x_{1,1}x_{2,1}x_{1,2}x_{2,2}](A, B), \\ & A \to [x_{1,1}a, x_{1,2}c](A), \\ & A \to [\varepsilon, \varepsilon](), \\ & B \to [x_{1,1}b, x_{1,2}d](B), \\ & B \to [\varepsilon, \varepsilon]() \end{aligned}$$

$$\begin{aligned} P_\varepsilon :\ & \\ & A \to [\varepsilon, \varepsilon](), \\ & B \to [\varepsilon, \varepsilon]() \end{aligned}$$

Omit unnecessary brackets I

- each state of $F(G)$ loops all brackets that don't contain terminals
- omit all grammar rules that cannot be utilized in derivation

$$
\begin{aligned}
P:\ & S \longrightarrow [x_{1,1}x_{2,1}x_{1,2}x_{2,2}](A, B),\\
& A \longrightarrow [x_{1,1}a, x_{1,2}c](A),\\
& A \longrightarrow [\varepsilon, \varepsilon](),\\
& B \longrightarrow [x_{1,1}b, x_{1,2}d](B),\\
& B \longrightarrow [\varepsilon, \varepsilon]()
\end{aligned}
$$

$$
\begin{aligned}
P_\varepsilon:\ & S \longrightarrow [x_{1,1}x_{2,1}x_{1,2}x_{2,2}](A, B),\\
& A \longrightarrow [\varepsilon, \varepsilon](),\\
& B \longrightarrow [\varepsilon, \varepsilon]()
\end{aligned}
$$

Omit unnecessary brackets I

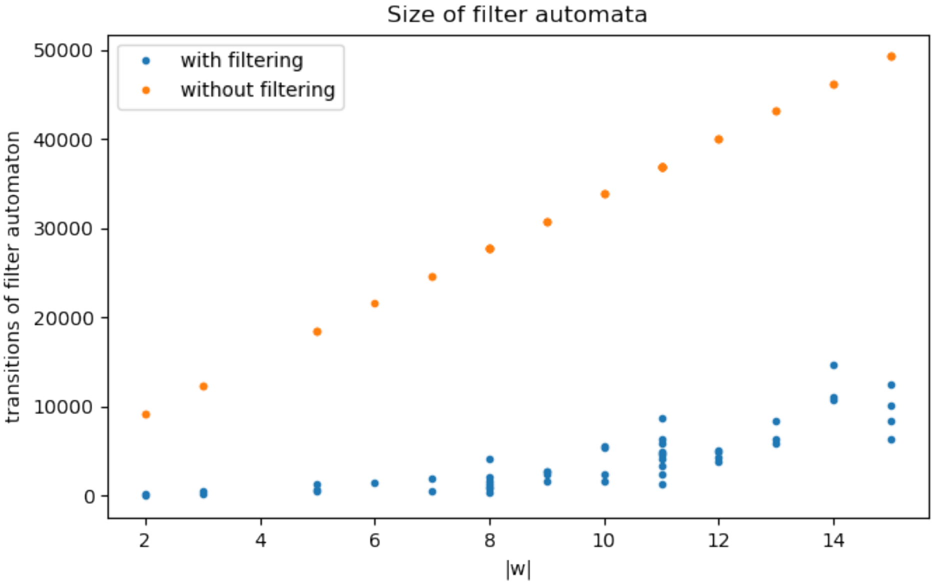
- each state of $F(G)$ loops all brackets that don't contain terminals
- omit all grammar rules that cannot be utilized in derivation

$$
\begin{aligned}
P:\ & S \to [x_{1,1}x_{2,1}x_{1,2}x_{2,2}](A, B),\\
& A \to [x_{1,1}a, x_{1,2}c](A),\\
& A \to [\varepsilon, \varepsilon](),\\
& B \to [x_{1,1}b, x_{1,2}d](B),\\
& B \to [\varepsilon, \varepsilon]()
\end{aligned}
$$

$$
\begin{aligned}
P_\varepsilon:\ & S \to [x_{1,1}x_{2,1}x_{1,2}x_{2,2}](A, B),\\
& A \to [\varepsilon, \varepsilon](),\\
& B \to [\varepsilon, \varepsilon]()
\end{aligned}
$$

- use $F((N, \Sigma, P_w, S), w)$ instead of $F((N, \Sigma, P, S), w)$

Omit unnecessary brackets II



Current status

- ✓ implementation of k-best parser for weighted MCFG
  - generator and filter automata
  - Hadamard product
  - recognizer for multiple Dyck languages
  - $\mathrm{toderiv}$
  - approximation via beam search
- ✗ OpenFst dropped
  - missing word iterator
  - switch to push-down automata
- - evaluation
  - beam search for generation of candidates
  - comparison to other parsers via parse time and Bleu-score

backup

Mutliple Dyck language D

- congruence relation for each $v_1, \ldots, v_k \in D(\Sigma)$ with $v_1 \ldots v_k \equiv_{\Sigma,\mathfrak{P}} \varepsilon$

$$\sigma_1 v_1 \overline{\sigma_1} u_1 \sigma_2 \ldots u_{k-1} \sigma_k v_k \overline{\sigma_k} \equiv_{\Sigma,\mathfrak{P}} u_1 \ldots u_{k-1}$$

  if $\{\sigma_1, \ldots, \sigma_k\} \in \mathfrak{P}$ and $u_1, \ldots, u_k$ are Dyck words over Σ
- $D = [\varepsilon]_{\equiv_{\Delta(G),\mathfrak{P}(G)}}$ where

$$\mathfrak{P}(G) = \{\ldots\}$$

Reachability analysis

- recursively enumerate $P_w \subseteq P$ using axioms
  - $A \to [u_1, \dots, u_k]() \in P_w$ if $u_1, \dots, u_k$ are subsequences in w
  - $A \to [u_{1,0} x_{i(1,1)}^{j(1,1)} u_{1,1} \dots u_{1,m_1}, \dots, \dots u_{l,m_l}](A_1, \dots, A_k) \in P_w$ if each $u_{1,0}, \dots, u_{l,m_l}$ is a subsequence of w and there are rules with lhs $A_1, \dots, A_k$ in P_w

Ordered multiple Dyck language

- encoded derivation trees in $\Delta(G)^*$ are simpler to recognize than multiple Dyck words
- remove the nonderminism introduced by $\mathrm{up}_{\mathfrak{P}}(\delta)$
  - we know which child node is visited, it is even annotated in each bracket
  - move to child i for brackets of form $\langle^{j}_{\rho,i}$
  - move to child 0 for all other brackets
- avoid expensive set operations by storing the associated rule and a list of visited components per node

OpenFst

was dropped

Enumeration of words recognized by an automaton

- heuristic: only considering states, weight of shortest path to final state