

Deterministic Pushdown Automata, Reductions and Normal Forms of CFGs

Martin Fränzle

Informatics and Mathematical Modelling
The Technical University of Denmark

What you'll learn

1. Deterministic pushdown automata:

- Importance
- Definition
- Language recognition capability (i.e. expressiveness)

2. Transformation/reductions of CFGs:

- Eliminating
 - useless symbols (terminal and non-terminal)
 - ϵ -productions
 - unit productions ($A \rightarrow B$)
- Chomsky normal form

Pushdown Automata

Deterministic language recognition

Motivation

Previous lecture detailed a direct construction of PDAs from CFGs.

- Highly nondeterministic

⇒ not practical, because of lack of adequate oracle.

Deterministic PDAs are a more practical alternative.

⇒ Their study sheds light on what constructs are suitable for use in practical

- programming languages
- markup languages
- database query languages
- ...

Deterministic PDAs

A PDA is deterministic iff it has no choice between moves when traversing any given word.

- No choice between two different
 - successor states or
 - stack updates.
- No choice between a move consuming an input letter and an ε move.

Deterministic PDAs

A PDA is deterministic iff it has no choice between moves when traversing any given word.

- No choice between two different
 - successor states or
 - stack updates.
- No choice between a move consuming an input letter and an ε move.

Def: A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is called **deterministic** or a **DPDA** iff

1. $|\delta(q, a, X)| \leq 1$ and
2. $\delta(q, \varepsilon, X) \neq \emptyset$ implies $\delta(q, a, X) = \emptyset$

for each $q \in Q$, each $a \in \Sigma \cup \{\varepsilon\}$ and each $X \in \Gamma$.

Regular Languages and DPDAs

Thm: If L is regular then $L = L(P)$ for some DPDA P .

Prf: Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ construct a corresponding DPDA $P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$ “ignoring” its stack by

$$\delta_P(q, a, Z_0) = \begin{cases} (\delta(q, a), Z_0) & \text{iff } a \in \Sigma \\ \emptyset & \text{otherwise.} \end{cases}$$

Regular Languages and DPDAs

Thm: If L is regular then $L = L(P)$ for some DPDA P .

Prf: Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ construct a corresponding DPDA $P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$ “ignoring” its stack by

$$\delta_P(q, a, Z_0) = \begin{cases} (\delta(q, a), Z_0) & \text{iff } a \in \Sigma \\ \emptyset & \text{otherwise.} \end{cases}$$

N.B. The same is not true with “ $N(P)$ ” instead of “ $L(P)$ ”, as a DPDA never accepts both a word w and a proper prefix v of w by empty stack. Hence, a DPDA cannot even accept $L = \{0\}^*$ by empty stack.

Cor: There are *regular* languages that cannot be defined by DPDAs by empty stack.

Regular vs. L(DPDA) vs. CFL

Thm: The languages accepted by DPDAs by final state properly includes the regular languages and are properly included in the CFLs.

- Prf:**
- DPDAs can simulate DFAs (see previous theorem), hence DPDA-recognizability covers regular languages;
 - DPDAs can recognize the non-regular language $\{0^n 1^n \mid n \in \mathbb{N}\}$;
 - DPDAs cannot recognize the CFL $\{w^w \mid w \in \{0, 1\}^*\}$.

Final state vs. empty stack

Def: A language L has the prefix property iff $w \in L$ implies $v \notin L$ for all proper prefixes v of w .

Thm: A language L is $N(P)$ for some DPDA P iff L has the prefix property and $L = L(P')$ for some DPDA P' .

Prf: Use the “from empty stack to final state” construction, and vice versa.

Final state vs. empty stack

Def: A language L has the prefix property iff $w \in L$ implies $v \notin L$ for all proper prefixes v of w .

Thm: A language L is $N(P)$ for some DPDA P iff L has the prefix property and $L = L(P')$ for some DPDA P' .

Prf: Use the “from empty stack to final state” construction, and vice versa.

Cor: If L has the prefix property then L is accepted by some DPDA by final state iff L is accepted by some DPDA by empty stack.

N.B. The prefix-property can always be enforced by adding a special end-marker (e.g., “EOF”).

PDPA vs. ambiguity

Thm: If $L = N(P)$ or $L = L(P)$ for some DPDA P then L has an unambiguous grammar.

Prf: The “PDA to grammar construction” yields an unambiguous grammar if P is deterministic.

PDPA vs. ambiguity

Thm: If $L = N(P)$ or $L = L(P)$ for some DPDA P then L has an unambiguous grammar.

Prf: The “PDA to grammar construction” yields an unambiguous grammar if P is deterministic.

N.B. There are nevertheless CFLs with unambiguous grammars that are not DPDA-definable, e.g. $\{ww^r \mid w \in \{0, 1\}^*\}$ which has the unambiguous grammar

$$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon .$$

PDPA vs. ambiguity

Thm: If $L = N(P)$ or $L = L(P)$ for some DPDA P then L has an unambiguous grammar.

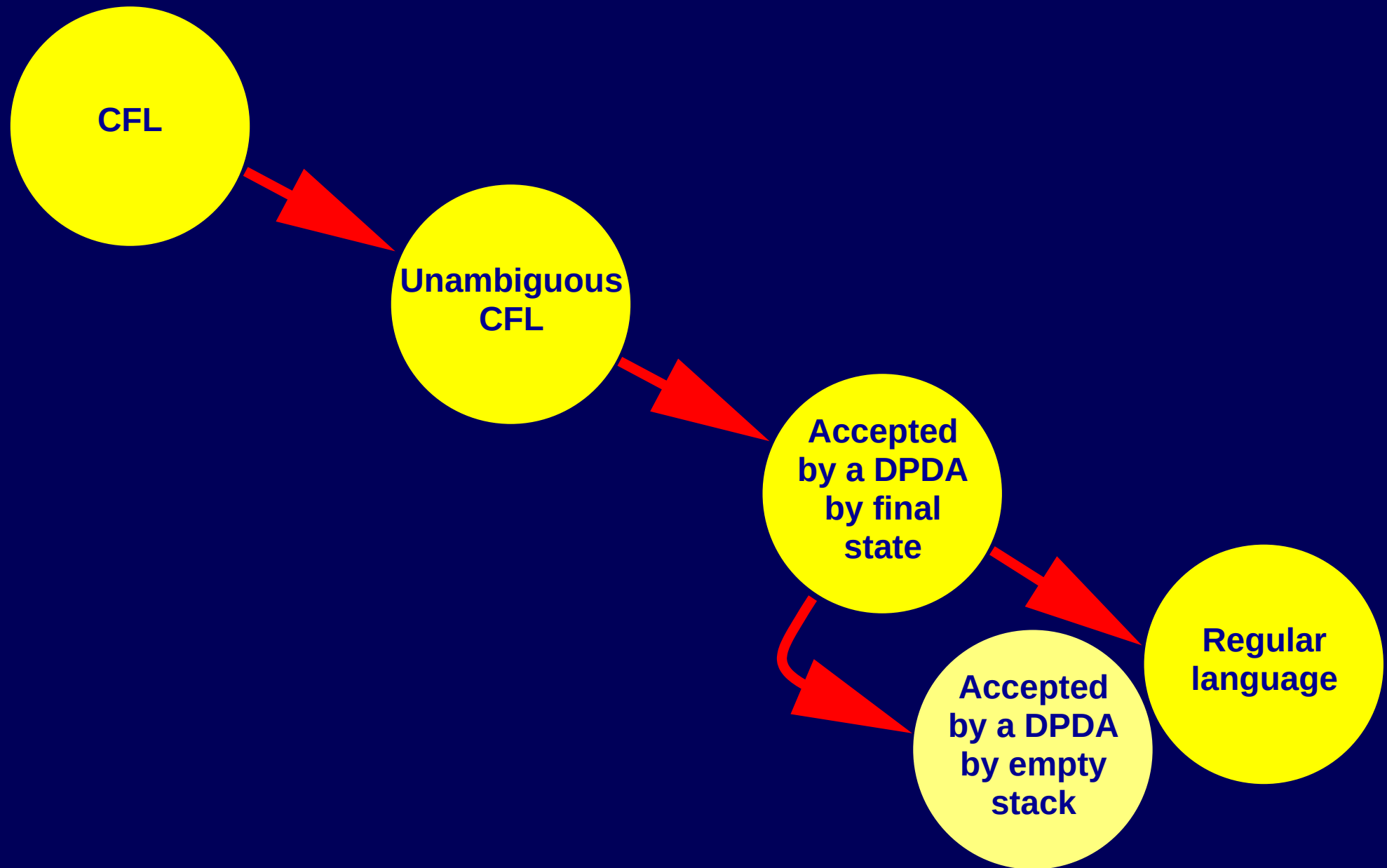
Prf: The “PDA to grammar construction” yields an unambiguous grammar if P is deterministic.

N.B. There are nevertheless CFLs with unambiguous grammars that are not DPDA-definable, e.g. $\{ww^r \mid w \in \{0,1\}^*\}$ which has the unambiguous grammar

$$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon .$$

Cor: The languages accepted by DPDAs by final state are properly included in the unambiguous (= not inherently ambiguous) CFLs.

A proper inclusion hierarchy



Context-free Grammars

Eliminations and normal forms

Chomsky normal form

Def: A context-free grammar $G = (V, T, P, S)$ is said to be in **Chomsky normal form** iff all its productions are of the two forms

- $A \rightarrow BC$, where $A, B, C \in V$, i.e. are all variables,
- $A \rightarrow a$, with $a \in T$

and G has no “useless” symbols.

A symbol (terminal or non-terminal) is considered to be useless if it occurs in no derivation from S to a string of the language.

Which CFLs can be expressed in Chomsky normal form?

Finding non-generating symbols

Def: A symbol $X \in V \cup T$ of a grammar $G = (V, T, P, S)$ is **generating** iff $X \Rightarrow_P^* w$ for some $w \in T^*$.

N.B. Each terminal symbol $X \in T$ is by definition generating.

Finding non-generating symbols

Def: A symbol $X \in V \cup T$ of a grammar $G = (V, T, P, S)$ is **generating** iff $X \Rightarrow_P^* w$ for some $w \in T^*$.

N.B. Each terminal symbol $X \in T$ is by definition generating.

Lem: The algorithm

Base: $Gen_0 := T$,

Recursion: Iterate

$$Gen_{i+1} := \begin{cases} Gen_i \cup \{A\} & \text{if there is } (A \rightarrow \alpha) \in P \text{ with } A \notin Gen_i \\ & \text{and every symbol of } \alpha \text{ being in } Gen_i \\ Gen_i & \text{otherwise} \end{cases}$$

until $Gen_{i+1} = Gen_i$

finds exactly the generating symbols of G .

Eliminating non-generating symbols

Thm: If $X \neq S$ is a non-generating symbol of $G = (V, T, P, S)$ then $L(G) = L(G_{-X})$ for $G_{-X} = (V \setminus \{X\}, T \setminus \{X\}, P_{-X}, S)$ with

$$P_{-X} = \{A \rightarrow \alpha \mid (A \rightarrow \alpha) \in P \text{ and all its symbols are } \neq X\} .$$

N.B. S itself can't be non-generating, unless $L(G) = \emptyset$. Hence, we can safely remove *all* non-generating symbols from any grammar G with $L(G) \neq \emptyset$.

Finding reachable symbols

Def: A symbol $X \in V \cup T$ of a grammar $G = (V, T, P, S)$ is **reachable** iff $S \Rightarrow_P^* \alpha X \beta$ for some $\alpha, \beta \in (V \cup T)^*$.

Finding reachable symbols

Def: A symbol $X \in V \cup T$ of a grammar $G = (V, T, P, S)$ is **reachable** iff $S \Rightarrow_P^* \alpha X \beta$ for some $\alpha, \beta \in (V \cup T)^*$.

Lem: The algorithm

Base: $Reach_0 := \{S\}$,

Recursion: Iterate

$$Reach_{i+1} := \begin{cases} Reach_i \cup \{\alpha_1, \dots, \alpha_n\} & \text{if there is} \\ & (A \rightarrow \alpha_1 \dots \alpha_n) \in P \\ & \text{with } A \in Reach_i \text{ and} \\ & \alpha_j \notin Reach_i \text{ for some } j \\ Reach_i & \text{otherwise} \end{cases}$$

until $Reach_{i+1} = Reach_i$

finds exactly the reachable symbols of G .

Eliminating unreachable symbols

Thm: If $X \neq S$ is an unreachable symbol of $G = (V, T, P, S)$ then
 $L(G) = L(G_{-X})$ for $G_{-X} = (V \setminus \{X\}, T \setminus \{X\}, P_{-X}, S)$ with

$$P_{-X} = \{A \rightarrow \alpha \mid (A \rightarrow \alpha) \in P \text{ and all its symbols are } \neq X\} .$$

N.B. S itself is reachable. Hence, we can safely remove *all* unreachable symbols from any grammar G .

Finding nullable symbols

Def: A symbol $X \in V$ of a grammar $G = (V, T, P, S)$ is **nullable** iff $X \Rightarrow_P^* \varepsilon$.

N.B. Nullability of X does *not* imply that there must be a production $X \rightarrow \varepsilon$ in P .

Finding nullable symbols

Def: A symbol $X \in V$ of a grammar $G = (V, T, P, S)$ is **nullable** iff $X \Rightarrow_P^* \varepsilon$.

N.B. Nullability of X does *not* imply that there must be a production $X \rightarrow \varepsilon$ in P .

Lem: The algorithm

Base: $Null_0 := \{A \mid (A \rightarrow \varepsilon) \in P\},$

Recursion: Iterate

$$Null_{i+1} := \begin{cases} Null_i \cup \{A\} & \text{if there is } (A \rightarrow \alpha) \in P \text{ with} \\ & \text{all symbols of } \alpha \text{ being in } Null_i \text{ and} \\ & A \notin Null_i \\ Null_i & \text{otherwise} \end{cases}$$

until $Null_{i+1} = Null_i$

finds exactly the nullable symbols of G .

Eliminating ε productions

Lem: If $G = (V, T, P, S)$ then $L(G) \setminus \{\varepsilon\} = L(G_{-\varepsilon})$ for
 $G_{-\varepsilon} = (V, T, P_{-\varepsilon}, S)$ with

$$P_{-\varepsilon} = \left\{ A \rightarrow w_1 w_2 \dots w_n \left| \begin{array}{l} (A \rightarrow w_1 \alpha_1 w_2 \alpha_2 \dots \alpha_n w_n) \in P, \\ w_1 w_2 \dots w_n \neq \varepsilon, \text{ and} \\ \alpha_i \text{ is nullable for each } i \end{array} \right. \right\} .$$

I.e., the new productions are obtained by removing an (almost) arbitrary number of nullable symbols from the production body. Resulting productions of the form $A \rightarrow \varepsilon$ are, however, not permitted.

Finding unit pairs

Def: A pair $(X, Y) \in V^2$ of symbols in a grammar $G = (V, T, P, S)$ is a **unit pair** iff $X \Rightarrow_P^* Y$.

Finding unit pairs

Def: A pair $(X, Y) \in V^2$ of symbols in a grammar $G = (V, T, P, S)$ is a unit pair iff $X \Rightarrow_P^* Y$.

Lem: The algorithm

Base: $UPair_0 := \{(A, A) \mid A \in V\}$,

Recursion: Iterate

$$UPair_{i+1} := \begin{cases} UPair_i \cup \{(A, C)\} & \text{if there is } B \in V \text{ with} \\ & (B, C) \in UPair_i \text{ and} \\ & (A \rightarrow \alpha B \beta) \in P \text{ and all symbols} \\ & \text{in } \alpha \text{ and } \beta \text{ are nullable and} \\ & (A, C) \notin UPair_i, \\ UPair_i & \text{otherwise} \end{cases}$$

until $UPair_{i+1} = UPair_i$

finds exactly the unit pairs of G .

Eliminating unit productions

Lem: If $G = (V, T, P, S)$ then $L(G) = L(G_{\text{unit}})$ for
 $G_{\text{unit}} = (V, T, P_{\text{unit}}, S)$ with

$$P_{\text{unit}} = \left\{ A \rightarrow \beta \mid \begin{array}{l} (A, B) \text{ is a unit pair and} \\ (B \rightarrow \beta) \in P \text{ and} \\ \beta \text{ is not a single variable} \end{array} \right\} .$$

Converting to Chomsky normal form

Thm: Any nonempty CFL not containing ε has a Chomsky normal form grammar.

Prf: Take an arbitrary grammar for the language and

1. eliminate ε -productions,
2. eliminate unit productions,
3. eliminate non-generating symbols,
4. eliminate unreachable symbols,
5. chain productions $A \rightarrow \alpha$ with $|\alpha| > 2$ by introducing helper variables:

$$A \rightarrow BCD \rightsquigarrow \left(\begin{array}{l} A \rightarrow BX \\ X \rightarrow CD \end{array} \right)$$

with a fresh variable X .