Ambiguity in Context-Free Grammars, Introduction to Pushdown Automata

Martin Fränzle

Informatics and Mathematical Modelling

The Technical University of Denmark

What you'll learn

1. **Context-free grammars:**
   - What's ambiguity
   - Why it is a problem
   - How to avoid it

   $\rightsquigarrow$ How to build useful CFGs

2. **Pushdown automata:**
   - Machines recognizing CFLs

   $\rightsquigarrow$ Operational approach towards CFLs

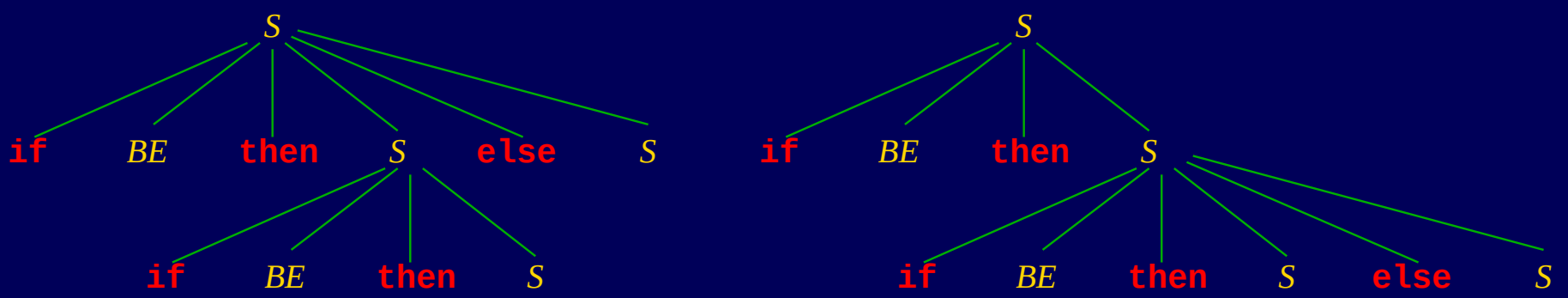
   $\rightsquigarrow$ Tools for CFLs

Ambiguity in CFGs

Pragmatics

The use cases of regular languages and context-free languages are subtly different:

- For RLs, one is primarily interested in language membership:
  - finding strings in a text,
  - classifying words/phrases in a text,
  - ...
- With CFLs, one is often also interested in the structural information conveyed by a parse tree:



A classical ambiguity

Given

$$\begin{aligned} S &\rightarrow \texttt{if}\ BE\ \texttt{then}\ S\ \texttt{else}\ S \mid \texttt{if}\ BE\ \texttt{then}\ S \mid \texttt{print}\ BE \mid \ldots \\ BE &\rightarrow \texttt{true} \mid \texttt{false} \mid \ldots \end{aligned}$$

what shall

```
if false then if false then print true else print false
```

output?

A classical ambiguity

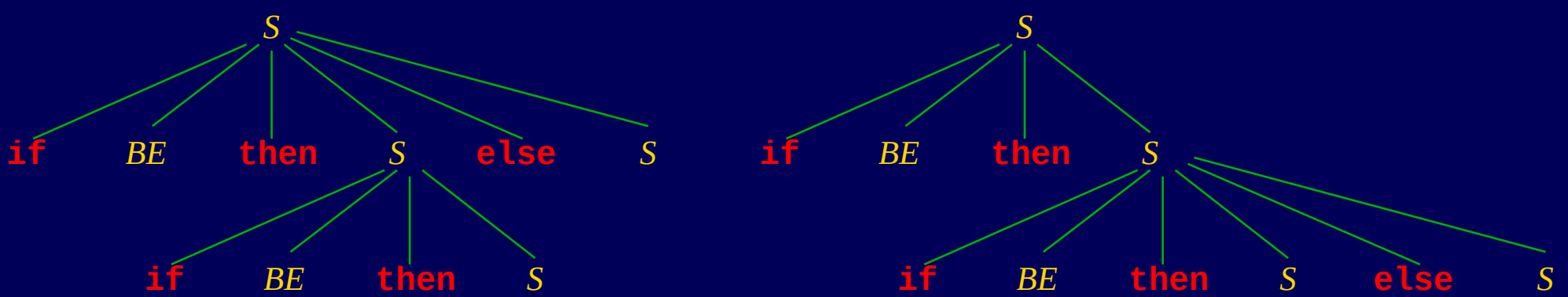
Given

$$S \rightarrow \texttt{if}\ BE\ \texttt{then}\ S\ \texttt{else}\ S \mid \texttt{if}\ BE\ \texttt{then}\ S \mid \texttt{print}\ BE \mid \ldots$$
$$BE \rightarrow \texttt{true} \mid \texttt{false} \mid \ldots$$

what shall

```
if false then if false then print true else print false
```

output?



A classical ambiguity — resolved

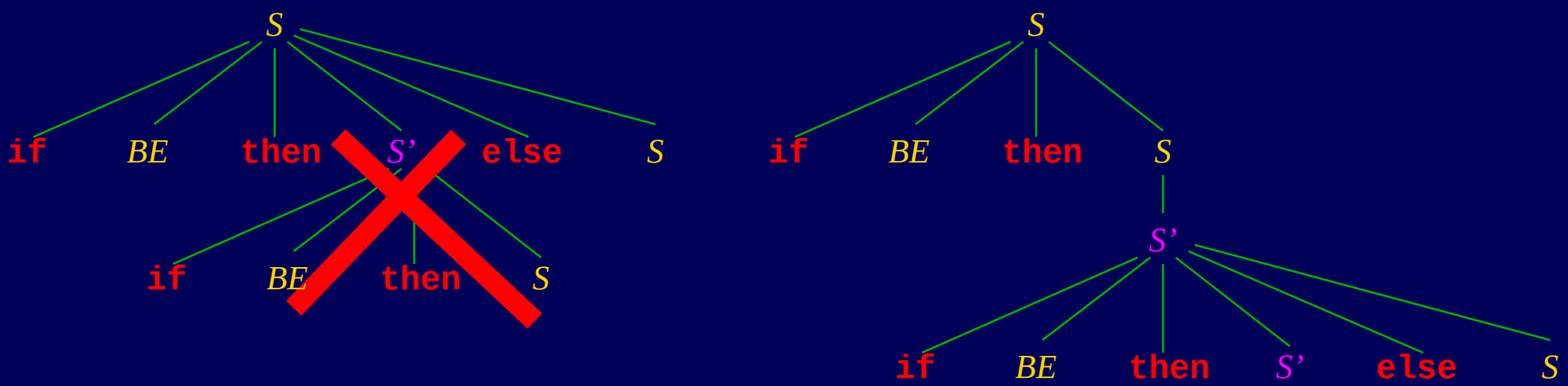
Given

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{S}' \mid \texttt{if}\ \mathrm{BE}\ \texttt{then}\ \mathrm{S} \mid \texttt{print}\ \mathrm{BE} \mid \ldots \\ \mathrm{S}' &\rightarrow \texttt{if}\ \mathrm{BE}\ \texttt{then}\ \mathrm{S}'\ \texttt{else}\ \mathrm{S} \mid \texttt{print}\ \mathrm{BE} \mid \ldots \\ \mathrm{BE} &\rightarrow \texttt{true} \mid \texttt{false} \mid \ldots \end{aligned}$$

what shall

```
if false then if false then print true else print false
```

output?



Ambiguity

Def: A CFG G is called ambiguous iff there is $w \in L(G)$ s.t. w has (at least) two different *parse trees* wrt. G.

N.B. It is *not* the existence of multiple *derivations* for some string $w \in L(G)$ that renders G ambiguous!

Removing ambiguity

There is no algorithm for resolving ambiguity (in the sense of automatically deriving an unambiguous grammar from a given grammar).

Removing ambiguity

There is no algorithm for resolving ambiguity (in the sense of automatically deriving an unambiguous grammar from a given grammar).

There is not even an algorithm for finding out whether a given CFG is ambiguous.

Removing ambiguity

There is no algorithm for resolving ambiguity (in the sense of automatically deriving an unambiguous grammar from a given grammar).

There is not even an algorithm for finding out whether a given CFG is ambiguous.

However, there are standard techniques for writing an unambiguous grammar that help in most cases.

Expression grammars

Ambiguous

$$E \rightarrow I \mid (E)$$
$$E \rightarrow E * E \mid E/E$$
$$E \rightarrow E + E \mid E - E$$
$$I \rightarrow a \mid b \mid Ia \mid Ib$$

Expression grammars

Ambiguous

$$E \rightarrow I \mid (E)$$
$$E \rightarrow E * E \mid E/E$$
$$E \rightarrow E + E \mid E - E$$
$$I \rightarrow a \mid b \mid Ia \mid Ib$$

Unambiguous

$$F \rightarrow I \mid (E)$$
$$T \rightarrow T * F \mid T/F \mid F$$
$$E \rightarrow E + T \mid E - T \mid T$$
$$I \rightarrow a \mid b \mid Ia \mid Ib$$

Derivations vs. ambiguity

Let $G = (V, T, P, S)$ be a grammar and $w \in T^*$.

Thm: The following statements are equivalent:

- w has more than one parse tree
- w has more than one *leftmost* derivation
- w has more than one *rightmost* derivation.

Prf. Leftmost (rightmost, resp.) derivations can be understood as a canonical way of traversing a parse tree and are thus in one-to-one correspondence to parse trees.

Inherent ambiguity

Def: A CFL L is called inherently ambiguous iff *all* CFGs G with $\mathrm{L}(\mathrm{G}) = \mathrm{L}$ are ambiguous.

N.B.

- The mere existence of *one* ambiguous CFG for L is not sufficient to render L inherently ambiguous.
  (In fact, any non-empty CFL has an ambiguous CFG.)

- All the previous examples had an unambiguous CFG and are thus not inherently ambiguous.

An inherently ambiguous language

The language

$$\begin{aligned} L = & \quad \{a^n b^n c^m d^m \mid n \geq 0, m \geq 0\} \\ & \cup \{a^n b^m c^m d^n \mid n \geq 0, m \geq 0\} \end{aligned}$$

is inherently ambiguous.

An inherently ambiguous language

The language

$$\begin{aligned} L = & \quad \{a^n b^n c^m d^m \mid n \geq 0, m \geq 0\} \\ & \cup \{a^n b^m c^m d^n \mid n \geq 0, m \geq 0\} \end{aligned}$$

is inherently ambiguous.

An example grammar:

$$\begin{aligned} S &\rightarrow AB \mid C \\ A &\rightarrow aAb \mid \varepsilon \\ B &\rightarrow cBd \mid \varepsilon \\ C &\rightarrow aCd \mid D \\ D &\rightarrow bDc \mid \varepsilon \end{aligned}$$

Pushdown Automata

Idea of pushdown automaton

Take an ε-NFA and

1. add a stack for unbounded storage, yet with access limited to "last-in-first-out";
2. make transitions dependent on input symbol *and* stack top;
3. add side effect on stack to transitions:
   transitions replace stack top by an arbitrary string, thus being able to
   - "pop" stack (replacement by ε),
   - preserve stack (replace stack top X by X),
   - "push" onto stack (replace stack top X by $w_1 \dots w_n X$),
   - "pop" and then "push" (replace stack top X by $w_1 \dots w_n$).

Formal definition

A pushdown automaton (PDA) is a seven-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ with

- Q being a *finite* set of states,
- Σ being the input alphabet, i.e. the *finite* set of input symbols,
- Γ being the stack alphabet,
- $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \to \mathcal{P}_{\text{fin}}(\Sigma \times \Gamma^*)$ being the transition function,
  - $(q', w) \in \delta(q, a, X)$ implies that P can move from q to q' when the input symbol is a (or arbitrary in case $a = \varepsilon$) and the stack top is X. X is then replaced by w.
- q_0 being the start state,
- Z_0 being the start symbol on the stack (i.e., execution starts with the stack containing exactely one item, namely Z_0),
- F being the set of accepting states.

Instantaneous descriptions (IDs)

- An instantaneous description (ID) is a triple $(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$, where
  - q denotes the current state,
  - w denotes the *remaining* input,
  - γ denotes the stack contents (top of the stack $\hat{=}$ *first* letter of γ).
- IDs describe *configurations of PDA computations*.

Instantaneous descriptions (IDs)

- An instantaneous description (ID) is a triple $(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$, where
  - q denotes the current state,
  - w denotes the *remaining* input,
  - γ denotes the stack contents (top of the stack $\triangleq$ *first* letter of γ).
- IDs describe *configurations of PDA computations*.
- $\vdash_P$ is a relation between IDs of PDA P formalizing "moves" of P:

$$(q, w, \gamma) \vdash_P (q', w', \gamma') \qquad \text{iff}$$

$$\exists\, a \in \Sigma \cup \{\varepsilon\}, X \in \Gamma, \alpha \in \Gamma^*, u \in \Gamma^* \bullet \begin{pmatrix} w = aw' & \wedge \\ \gamma = X\alpha & \wedge \\ \gamma' = u\alpha & \wedge \\ (q', u) \in \delta(q, a, X) & \end{pmatrix}$$

Translation invariance

Thm: If $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA and $w \in \Sigma^*$ and $\gamma \in \Gamma^*$ then

$$(q, x, \alpha) \vdash_P^* (p, y, \beta) \quad \text{implies} \quad (q, xw, \alpha\gamma) \vdash_P^* (p, yw, \beta\gamma)$$

I.e., PDA behaviours are preserved under stack extension and/or input extension.

Prf: It is easy to see from the definition that

$$(q, x, \alpha) \vdash_P (p, y, \beta) \quad \text{implies} \quad (q, xw, \alpha\gamma) \vdash_P (p, yw, \beta\gamma) \ .$$

The theorem follows by induction on the number of steps.

Translation invariance

Thm: If $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA and $x, y \in \Sigma^*$ then

$$(q, xw, \alpha) \vdash_P^* (p, yw, \beta) \quad \text{implies} \quad (q, x, \alpha) \vdash_P^* (p, y, \beta)$$

I.e., PDA behaviour is independent from trailing input.

N.B. The same is *not* true wrt. trailing stack contents, as the PDA might well pop off, inspect, and then restore some part of the stack contents.

Prf: It is easy to see from the definition that

$$(q, xw, \alpha) \vdash_P (p, yw, \beta) \quad \text{implies} \quad (q, x, \alpha) \vdash_P (p, y, \beta) \ .$$

The theorem follows by induction on the number of steps.

Language of a PDA

Def: The language $L(P)$ of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is

$$L(P) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_P^* (q, \varepsilon, \alpha), q \in F, \alpha \in \Gamma^*\} \ .$$

This form of language acceptance is often called "acceptance by final state".

Language of a PDA

Def: The language $L(P)$ of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is

$$L(P) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_P^* (q, \varepsilon, \alpha), q \in F, \alpha \in \Gamma^*\} \ .$$

This form of language acceptance is often called "acceptance by final state".

Def: The language $N(P)$ of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is

$$N(P) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_P^* (q, \varepsilon, \varepsilon), q \in \Sigma\} \ .$$

This form of language acceptance is often called "acceptance by empty stack".

N.B. In general, $L(P) \neq N(P)$.

Expressiveness of the acceptance criteria

Thm: If $L = N(P)$ for some pushdown automaton P then there is a pushdown automaton P' with $L = L(P')$.
(Given P, the construction of P' is completely mechanic.)

Both forms of acceptance are thus equally “expressive”.

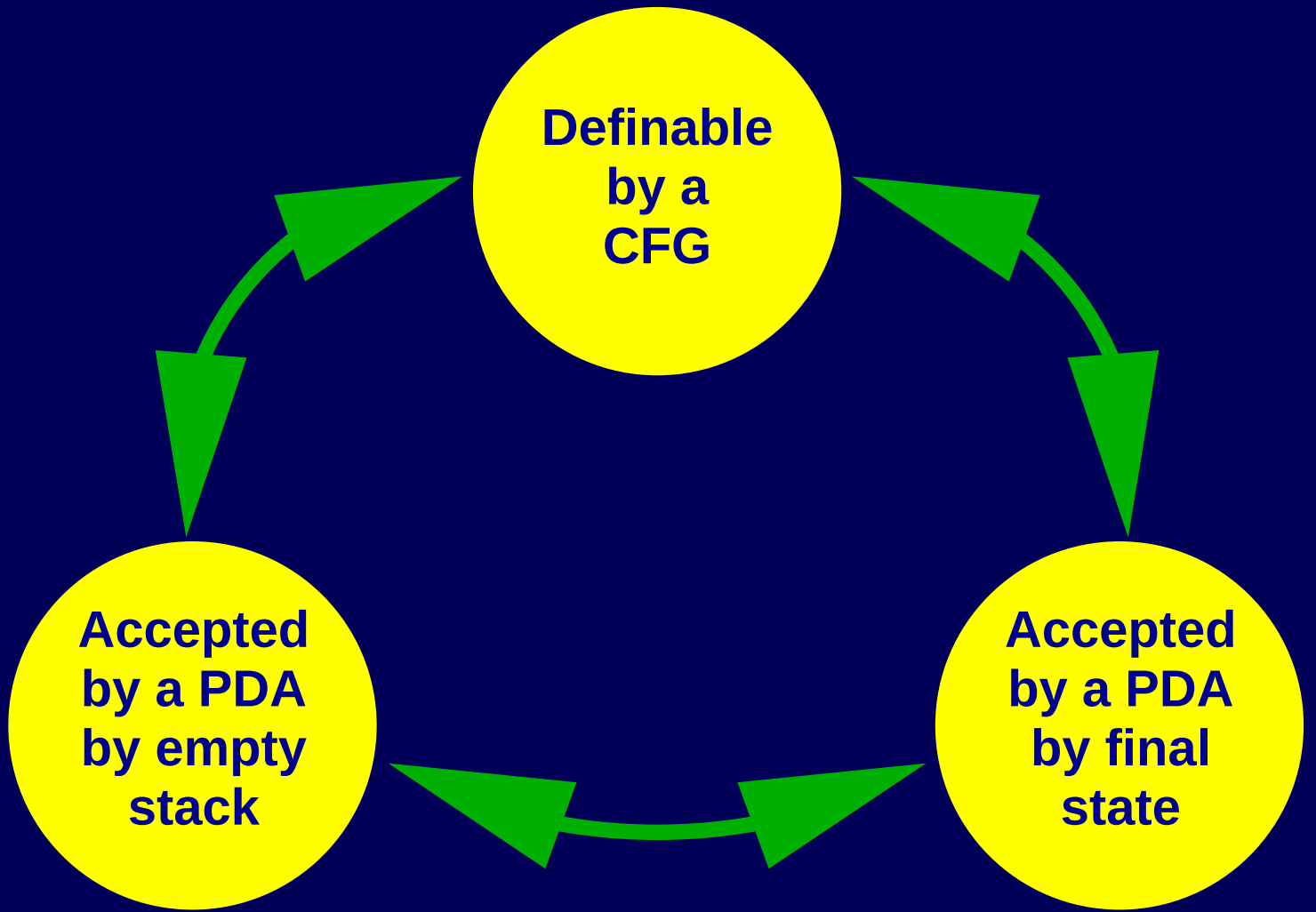
Expressiveness of the acceptance criteria

Thm: If $L = N(P)$ for some pushdown automaton P then there is a pushdown automaton P' with $L = L(P')$.
(Given P, the construction of P' is completely mechanic.)

Thm: If $L = L(P)$ for some pushdown automaton P then there is a pushdown automaton P' with $L = N(P')$.
(Given P, the construction of P' is completely mechanic.)

Both forms of acceptance are thus equally “expressive”.

PDAs vs. CFGs



Claim: PDAs and CFGs define the same languages.

Proof of the claim

- Have already shown equivalence between
  - being accepted by some PDA by empty stack
  - being accepted by some PDA by final state.

$\Rightarrow$ Suffices to show equivalence of
  - being definable by a CFG

to *one* of the two forms of PDA definability.

- We will show equivalence of CFG-definabilty to being accepted by some PDA by *empty stack*.

From CFG to PDA

Thm: If $\mathrm{L} = \mathrm{L}(\mathrm{G})$ for some contextfree grammar G then there is a pushdown automaton P with $\mathrm{L} = \mathrm{N}(\mathrm{P})$.
(Given G, the construction of P is completely mechanic.)

From CFG to PDA

Thm: If $L = L(G)$ for some contextfree grammar G then there is a pushdown automaton P with $L = N(P)$.
(Given G, the construction of P is completely mechanic.)

Prf: (Essence of) Given $G = (V, T, Q, S)$, a corresponding 1-state PDA P can be defined by

$$
\begin{array}{rcll}
P & = & (\{q\}, T, V \cup T, \delta, q, S) & \\
\delta(q, \varepsilon, A) & = & \{(q, \beta) \mid A \to \beta \in Q\} & \text{for } A \in V \\
\delta(q, a, a) & = & \{(q, \varepsilon)\} & \text{for } a \in T \\
\delta(q, x, \gamma) & = & \emptyset & \text{otherwise.}
\end{array}
$$

Then show that for any $w \in T^*$ and any $\alpha \in (T \cup V)^*$

$$(q, w, S) \vdash_P^* (q, \varepsilon, \alpha) \quad \text{iff} \quad S \overset{lm}{\Rightarrow}{}_G^* w\alpha \ .$$

From PDA to CFG

Thm: If $L = N(P)$ for some pushdown automaton P then there is a contextfree grammar G with $L = L(G)$.
(Given G, the construction of P is completely mechanic.)

From PDA to CFG

Thm: If $L = N(P)$ for some pushdown automaton P then there is a contextfree grammar G with $L = L(G)$.
(Given G, the construction of P is completely mechanic.)

Prf: Given $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, a corresponding CFG G can be defined by

$$
\begin{aligned}
G &= (V, \Sigma, R, S) \\
V &= \{S\} \cup \{[p\gamma q] \mid p, q \in Q, \gamma \in \Gamma\} \\
R &= \{S \to [q_0 Z_0 q] \mid q \in Q\} \cup \\
&\quad \left\{ [q\gamma r_k] \to a[r\alpha_1 r_1][r_1\alpha_2 r_2]\ldots[r_{k-1}\alpha_k r_k] \;\middle|\; \begin{array}{l} q \in Q,\ r_1, \ldots, r_k \in Q, \\ \alpha \in \Sigma \cup \{\varepsilon\},\ \gamma \in \Gamma, \\ (r, \alpha_1 \ldots \alpha_k) \in \delta(q, a, \gamma) \end{array} \right\}
\end{aligned}
$$

Then show that for any $p, q \in Q$, any $w \in \Sigma^*$, and any $\gamma \in \Gamma$

$$[p\gamma q] \Rightarrow_G^* w \quad \text{iff} \quad (p, w, \gamma) \vdash_P^* (q, \varepsilon, \varepsilon) \ .$$

Stack symbols can simulate states

Cor: For each PDA P there is a PDA $P' = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ with $|Q| = 1$ such that $N(P) = N(P')$.

Prf:
1. Convert P to an equivalent CFG G using the construction of the previous theorem,
2. convert G to a one-state PDA P' using the construction of the second-last theorem.

Stack symbols can simulate states

Cor: For each PDA P there is a PDA $P' = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ with $|Q| = 1$ such that $N(P) = N(P')$.

Prf:
1. Convert P to an equivalent CFG G using the construction of the previous theorem,
2. convert G to a one-state PDA P' using the construction of the second-last theorem.

Prize is blowup in the size of the stack alphabet!