Logic and Theoretical Foundation of Computer Science

LaTFoCS

Pamela Fleischmann

fpa@informatik.uni-kiel.de

Winter Semester 2019

Kiel University
Dependable Systems Group

Context-Free Grammars and Languages

New Approach

- Goal: Automaton that can recognize $\{a^n b^n \mid n \in \mathbb{N}\}$

New Approach

- Goal: Automaton that can recognize $\{a^n b^n \mid n \in \mathbb{N}\}$
- Ideas:

New Approach

- Goal: Automaton that can recognize $\{a^n b^n \mid n \in \mathbb{N}\}$
- Ideas:
  - Each time we read an a, we read simultaneously an b

New Approach

- Goal: Automaton that can recognize $\{a^n b^n | n \in \mathbb{N}\}$
- Ideas:
  - Each time we read an a, we read simultaneously an b
  - We have to ensure that all as are before the bs

New Approach

- Goal: Automaton that can recognize $\{a^n b^n | n \in \mathbb{N}\}$
- Ideas:
  - Each time we read an a, we read simultaneously an b
  - We have to ensure that all as are before the bs
- Constructive Approach with rules: $S \longrightarrow aSb|\varepsilon$

(Context-Free) Grammars

Definition

- $G = (V, \Sigma, S, P)$ grammar iff

(Context-Free) Grammars

Definition

- $G = (V, \Sigma, S, P)$ grammar iff
  - V finite set of variables

(Context-Free) Grammars

Definition

- $G = (V, \Sigma, S, P)$ grammar iff
  - V finite set of variables
  - Σ alphabet with $V \cap \Sigma = \emptyset$

(Context-Free) Grammars

Definition

- $G = (V, \Sigma, S, P)$ grammar iff
  - V finite set of variables
  - Σ alphabet with $V \cap \Sigma = \emptyset$
  - $S \in V$ start symbol

(Context-Free) Grammars

Definition

- $G = (V, \Sigma, S, P)$ grammar iff
  - V finite set of variables
  - Σ alphabet with $V \cap \Sigma = \emptyset$
  - $S \in V$ start symbol
  - $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma \cup \{\varepsilon\})^*$ production rules

(Context-Free) Grammars

Definition

- $G = (V, \Sigma, S, P)$ grammar iff
  - V finite set of variables
  - Σ alphabet with $V \cap \Sigma = \emptyset$
  - $S \in V$ start symbol
  - $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma \cup \{\varepsilon\})^*$ production rules
- G context-free: $P \subseteq V \times (V \cup \Sigma \cup \{\varepsilon\})^*$

A Grammar's Language

Definition

G context-free grammar, $\alpha, \beta \in (V \cup \Sigma)^*$

- $\alpha \vdash \beta$ (β is derivable from α in one step) iff

$$\exists (A, \gamma) \in P : \alpha = \alpha_1 A \alpha_2 \wedge \beta = \alpha_1 \gamma \alpha_2$$

A Grammar's Language

Definition

G context-free grammar, $\alpha, \beta \in (V \cup \Sigma)^*$

- $\alpha \vdash \beta$ (β is derivable from α in one step) iff

$$\exists (A, \gamma) \in P : \alpha = \alpha_1 A \alpha_2 \wedge \beta = \alpha_1 \gamma \alpha_2$$

- $\vdash^*$ reflexive-transitive closure

A Grammar's Language

Definition

G context-free grammar, $\alpha, \beta \in (V \cup \Sigma)^*$

- $\alpha \vdash \beta$ (β is derivable from α in one step) iff

$$\exists (A, \gamma) \in P : \alpha = \alpha_1 A \alpha_2 \wedge \beta = \alpha_1 \gamma \alpha_2$$

- $\vdash^*$ reflexive-transitive closure
- $L(G) = \{w \in \Sigma^* \mid S \vdash^* w\}$

Balanced Paranthesis

Motivation

- arithmetics: $(5+3)\cdot 2$, $(7+(8\div 2))\cdot 4$

Motivation

- arithmetics: $(5 + 3) \cdot 2$, $(7 + (8 \div 2)) \cdot 4$
- programming languages:
  `if` (everything alright) { do a lot of stuff}

Motivation

- arithmetics: $(5+3)\cdot 2$, $(7+(8\div 2))\cdot 4$
- programming languages:
  `if` (everything alright) { do a lot of stuff}
- can we detect if something is not correct?

Balanced Parenthesis

informal:

- number of left parenthesis = number of right parenthesis

Balanced Parenthesis

informal:

- number of left parenthesis = number of right parenthesis
- the number of right parenthesis in every prefix is at most the number of left parenthesis

Balanced Parenthesis

informal:

- number of left parenthesis = number of right parenthesis
- the number of right parenthesis in every prefix is at most the number of left parenthesis

formal:

Definition (Balanced Parenthesis)

$\Sigma_P = \Sigma \cup \{(,)\}$

$x \in \Sigma_P^*$ balanced iff

1. $|x|_(= |x|_)$
2. $\forall y \in \mathrm{Pref}(x) : \; |y|_(\geq |y|_)$

Checking the Balanced Property

Definition

$G_P = (\{S\}, \Sigma_P, S, P)$ with the productions P

$$S \longrightarrow (S)|SS|\varepsilon$$
$$\forall a \in \Sigma: \; S \longrightarrow a$$

Checking the Balanced Property

Definition

$G_P = (\{S\}, \Sigma_P, S, P)$ with the productions P

$$S \longrightarrow (S)|SS|\varepsilon$$
$$\forall a \in \Sigma: \; S \longrightarrow a$$

Theorem

$L(G_P) = \{x \in \Sigma_P |\, x \text{ balanced}\}$

$\subseteq$-Proof

Induction on length of derivation:

- Start: $S \vdash_G^0 x \Rightarrow$ no step $\Rightarrow$ no parenthesis $\Rightarrow \surd$

$\subseteq$-Proof

Induction on length of derivation:

- Start: $S \vdash_G^0 x \Rightarrow$ no step $\Rightarrow$ no parenthesis $\Rightarrow \surd$
- Hypothesis: if $S \vdash_G^n x$ for an arbitrary but fixed $n \in \mathbb{N}$ then x is balanced

$\subseteq$-Proof

Induction on length of derivation:

- Start: $S \vdash_G^0 x \Rightarrow$ no step $\Rightarrow$ no parenthesis $\Rightarrow \surd$
- Hypothesis: if $S \vdash_G^n x$ for an arbitrary but fixed $n \in \mathbb{N}$ then x is balanced
- Step: Consider $S \vdash_G^{n+1} x$

$\subseteq$-Proof

Induction on length of derivation:

- Start: $S \vdash_G^0 x \Rightarrow$ no step $\Rightarrow$ no parenthesis $\Rightarrow \surd$
- Hypothesis: if $S \vdash_G^n x$ for an arbitrary but fixed $n \in \mathbb{N}$ then x is balanced
- Step: Consider $S \vdash_G^{n+1} x$
  - definition of derivation $\Rightarrow \exists z \in \Sigma_P^* : S \vdash_G^n z \vdash_G^1 x$

$\subseteq$-Proof

Induction on length of derivation:

- Start: $S \vdash_G^0 x \Rightarrow$ no step $\Rightarrow$ no parenthesis $\Rightarrow \surd$
- Hypothesis: if $S \vdash_G^n x$ for an arbitrary but fixed $n \in \mathbb{N}$ then x is balanced
- Step: Consider $S \vdash_G^{n+1} x$
  - definition of derivation $\Rightarrow \exists z \in \Sigma_P^* : S \vdash_G^n z \vdash_G^1 x$
  - hypothesis $\Rightarrow z$ balanced

Cont.

- case 1: second, third or fourth rule applied

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1 (S) z_2$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$
  - case a: $|y| \leq |z_1|$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$
  - case a: $|y| \leq |z_1|$
    - hypothesis $\Rightarrow \surd$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$
  - case a: $|y| \leq |z_1|$
    - hypothesis $\Rightarrow \surd$
  - case b: $|y| \leq |z_1| + 2$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$
  - case a: $|y| \leq |z_1|$
    - hypothesis $\Rightarrow \surd$
  - case b: $|y| \leq |z_1| + 2$
    - $|y|_{(} = |z_1|_{(} + 1$ and $|y|_{)} = |z_1|_{)}$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$
  - case a: $|y| \leq |z_1|$
    - hypothesis $\Rightarrow \surd$
  - case b: $|y| \leq |z_1| + 2$
    - $|y|_(= |z_1|_(+ 1$ and $|y|_) = |z_1|_)$
    - $\Rightarrow |y|_(= |z_1|_(+ 1 = |z_1|_) + 1 = |y|_) + 1 \Rightarrow \surd$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$
  - case a: $|y| \leq |z_1|$
    - hypothesis $\Rightarrow \surd$
  - case b: $|y| \leq |z_1| + 2$
    - $|y|_(= |z_1|_(+ 1$ and $|y|_) = |z_1|_)$
    - $\Rightarrow |y|_(= |z_1|_(+ 1 = |z_1|_) + 1 = |y|_) + 1 \Rightarrow \surd$
  - case c: $|y| > |z_1| + 2$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$
  - case a: $|y| \leq |z_1|$
    - hypothesis $\Rightarrow \surd$
  - case b: $|y| \leq |z_1| + 2$
    - $|y|_(= |z_1|_(+ 1$ and $|y|_) = |z_1|_)$
    - $\Rightarrow |y|_(= |z_1|_(+ 1 = |z_1|_) + 1 = |y|_) + 1 \Rightarrow \surd$
  - case c: $|y| > |z_1| + 2$
    - $|y|_(= |z_1|_(+ 1$ and $|y|_) = |z_1|_) + 1$

Cont.

- case 1: second, third or fourth rule applied
  - no change in number or order of parenthesis $\Rightarrow x$ balanced
- case 2: first rule applied
  - $z = z_1 S z_2$, $x = z_1(S)z_2$
  - obviously first property is satifies
  - for the second property: $y \in \mathrm{Pref}(x)$
  - case a: $|y| \leq |z_1|$
    - hypothesis $\Rightarrow \surd$
  - case b: $|y| \leq |z_1| + 2$
    - $|y|_(= |z_1|_(+ 1$ and $|y|_) = |z_1|_)$
    - $\Rightarrow |y|_(= |z_1|_(+ 1 = |z_1|_) + 1 = |y|_) + 1 \Rightarrow \surd$
  - case c: $|y| > |z_1| + 2$
    - $|y|_(= |z_1|_(+ 1$ and $|y|_) = |z_1|_) + 1$
    - $\Rightarrow |y|_(= |z_1|_(+ 1 = |z_1|_) + 1 = |y|_) \Rightarrow \surd$

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \to \varepsilon$

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \rightarrow \varepsilon$
- Hypothesis: if x balanced with $|x| = n \in \mathbb{N}$ for n arbitrary but fixed then x is producible by G_P

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \to \varepsilon$
- Hypothesis: if x balanced with $|x| = n \in \mathbb{N}$ for n arbitrary but fixed then x is producible by G_P
- Step: 2 cases

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \longrightarrow \varepsilon$
- Hypothesis: if x balanced with $|x| = n \in \mathbb{N}$ for n arbitrary but fixed then x is producible by G_P
- Step: 2 cases
- case 1: $\exists y, z \in \Sigma^+ : \ y, z$ balanced and $x = yz$

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \rightarrow \varepsilon$
- Hypothesis: if x balanced with $|x| = n \in \mathbb{N}$ for n arbitrary but fixed then x is producible by G_P
- Step: 2 cases
- case 1: $\exists y, z \in \Sigma^+ : \; y, z$ balanced and $x = yz$
  - $|y|, |z| < |x| \Rightarrow S \vdash_G^* y$ and $S \vdash_G^* z$

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \to \varepsilon$
- Hypothesis: if x balanced with $|x| = n \in \mathbb{N}$ for n arbitrary but fixed then x is producible by G_P
- Step: 2 cases
- case 1: $\exists y, z \in \Sigma^+ : \; y, z$ balanced and $x = yz$
  - $|y|, |z| < |x| \Rightarrow S \vdash_G^* y$ and $S \vdash_G^* z$
  - derivation for x: $S \vdash_G^1 SS \vdash_G^* yS \vdash_G^* yz = x$

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \to \varepsilon$
- Hypothesis: if x balanced with $|x| = n \in \mathbb{N}$ for n arbitrary but fixed then x is producible by G_P
- Step: 2 cases
- case 1: $\exists y, z \in \Sigma^+ :\ y, z$ balanced and $x = yz$
  - $|y|, |z| < |x| \Rightarrow S \vdash_G^* y$ and $S \vdash_G^* z$
  - derivation for x: $S \vdash_G^1 SS \vdash_G^* yS \vdash_G^* yz = x$
- case 2: x not splittable into two balanced parts

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \rightarrow \varepsilon$
- Hypothesis: if x balanced with $|x| = n \in \mathbb{N}$ for n arbitrary but fixed then x is producible by G_P
- Step: 2 cases
- case 1: $\exists y, z \in \Sigma^+ :\ y, z$ balanced and $x = yz$
  - $|y|, |z| < |x| \Rightarrow S \vdash_G^* y$ and $S \vdash_G^* z$
  - derivation for x: $S \vdash_G^1 SS \vdash_G^* yS \vdash_G^* yz = x$
- case 2: x not splittable into two balanced parts
  - first rule is the first $\Rightarrow x = (z) \Rightarrow |z| < |x|$

$\supseteq$-Proof

Induction on $|x|$

- Start: $x = \varepsilon \Rightarrow S \vdash x$ by production $S \to \varepsilon$
- Hypothesis: if x balanced with $|x| = n \in \mathbb{N}$ for n arbitrary but fixed then x is producible by G_P
- Step: 2 cases
- case 1: $\exists y, z \in \Sigma^+ :\ y, z$ balanced and $x = yz$
  - $|y|, |z| < |x| \Rightarrow S \vdash_G^* y$ and $S \vdash_G^* z$
  - derivation for x: $S \vdash_G^1 SS \vdash_G^* yS \vdash_G^* yz = x$
- case 2: x not splittable into two balanced parts
  - first rule is the first $\Rightarrow x = (z) \Rightarrow |z| < |x|$
  - is z balanced?

⊆-Proof cont.

- Ind. Step, case 2: x not splittable into two balanced parts
  - first rule is the first $\Rightarrow x = (z) \Rightarrow |z| < |x|$
  - is z balanced?

$\subseteq$-Proof cont.

- Ind. Step, case 2: x not splittable into two balanced parts
  - first rule is the first $\Rightarrow x = (z) \Rightarrow |z| < |x|$
  - is z balanced?
    - $|z|_(= |x|_(- 1 = |x|_) - 1 = |z|_) \Rightarrow$ 1.

$\subseteq$-Proof cont.

- Ind. Step, case 2: x not splittable into two balanced parts
  - first rule is the first $\Rightarrow x = (z) \Rightarrow |z| < |x|$
  - is z balanced?
    - $|z|_(= |x|_(- 1 = |x|_) - 1 = |z|_) \Rightarrow$ 1.
    - $u \in \mathrm{Pref}(z)$, x balanced $\Rightarrow |u|_(- |u|_) = |(u|_(- 1 - |(u|_) \geq 0$

$\subseteq$-Proof cont.

- Ind. Step, case 2: x not splittable into two balanced parts
  - first rule is the first $\Rightarrow x = (z) \Rightarrow |z| < |x|$
  - is z balanced?
    - $|z|_(= |x|_(- 1 = |x|_) - 1 = |z|_) \Rightarrow$ 1.
    - $u \in \mathrm{Pref}(z)$, x balanced $\Rightarrow |u|_(- |u|_) = |(u|_(- 1 - |(u|_) \geq 0$
    - $\Rightarrow z$ balanced

⊆-Proof cont.

- Ind. Step, case 2: x not splittable into two balanced parts
  - first rule is the first $\Rightarrow x = (z) \Rightarrow |z| < |x|$
  - is z balanced?
    - $|z|_(= |x|_(- 1 = |x|_) - 1 = |z|_) \Rightarrow$ 1.
    - $u \in \mathrm{Pref}(z)$, x balanced $\Rightarrow |u|_(- |u|_) = |(u|_(- 1 - |(u|_) \geq 0$
    - $\Rightarrow z$ balanced
  - $\Rightarrow S \vdash_G^* z$

$\subseteq$-Proof cont.

- Ind. Step, case 2: x not splittable into two balanced parts
  - first rule is the first $\Rightarrow x = (z) \Rightarrow |z| < |x|$
  - is z balanced?
    - $|z|_(= |x|_(- 1 = |x|_) - 1 = |z|_) \Rightarrow$ 1.
    - $u \in \mathrm{Pref}(z)$, x balanced $\Rightarrow |u|_(- |u|_) = |(u|_(- 1 - |(u|_) \geq 0$
    - $\Rightarrow z$ balanced
  - $\Rightarrow S \vdash_G^* z$
  - $\Rightarrow S \vdash_G^1 (S) \vdash_G^* (z) = x$ $\square$

Normal Forms

Why normal forms?

- general grammar: everything is allowed
- $\Rightarrow$ complicated to reason with it
- first restriction: context-freedom
  - rule depends on one variable
  - rule does not depend on the variable's *neighbours*
- the right-hand side may be as *insane* as we can imagine
  - $A \longrightarrow B$ and $B \longrightarrow A$
  - $A \longrightarrow \varepsilon$

Chomsky and Greibach Normalform

Definition (Chomsky Normalform)

G context-free grammar; G in CNF iff $P \subseteq (V \times V^2) \cup (V \times \Sigma)$.

Chomsky and Greibach Normalform

Definition (Chomsky Normalform)

G context-free grammar; G in CNF iff $P \subseteq (V \times V^2) \cup (V \times \Sigma)$.

Definition (Greibach Normalform)

G context-free grammar; G in GNF iff $P \subseteq V \times \Sigma V^*$

Chomsky and Greibach Normalform

Definition (Chomsky Normalform)

G context-free grammar; G in CNF iff $P \subseteq (V \times V^2) \cup (V \times \Sigma)$.

Definition (Greibach Normalform)

G context-free grammar; G in GNF iff $P \subseteq V \times \Sigma V^*$

Notice: ε is not producible in CNF or GNF

What do we want?

we would like to work and prove with CFG in CNF or GNF, i.e. we'd like to have

Theorem

For all CFG G exists CNF G' and GNF G'' with

$$L(G'') = L(G') = L(G) \backslash \{\varepsilon\}.$$

What do we want?

we would like to work and prove with CFG in CNF or GNF, i.e. we'd like to have

Theorem

For all CFG G exists CNF G' and GNF G'' with

$$L(G'') = L(G') = L(G) \setminus \{\varepsilon\}.$$

- can we prove that?

What do we want?

we would like to work and prove with CFG in CNF or GNF, i.e. we'd like to have

Theorem

For all CFG G exists CNF G' and GNF G'' with

$$L(G'') = L(G') = L(G) \setminus \{\varepsilon\}.$$

- can we prove that?
- hope: in NFA the ε-transitions were not needed

Getting rid of ε

Lemma

For all CFG G exists CFG G' without ε-production or unit-production such that $L(G') = L(G) \setminus \{\varepsilon\}$.

Getting rid of ε

Lemma

For all CFG G exists CFG G' without ε-production or unit-production such that $L(G') = L(G) \setminus \{\varepsilon\}$.

Construction of G':

Getting rid of ε

Lemma

For all CFG G exists CFG G' without ε-production or unit-production such that $L(G') = L(G) \setminus \{\varepsilon\}$.

Construction of G':

- $\hat{P} := P$

Getting rid of ε

Lemma

For all CFG G exists CFG G' without ε-production or unit-production such that $L(G') = L(G) \backslash \{\varepsilon\}$.

Construction of G':

- $\hat{P} := P$
- `while` changes

Getting rid of ε

Lemma

For all CFG G exists CFG G' without ε-production or unit-production such that $L(G') = L(G) \setminus \{\varepsilon\}$.

Construction of G':

- $\hat{P} := P$
- `while` changes
  - $A \longrightarrow \alpha B \beta \in \hat{P} \wedge B \longrightarrow \varepsilon \in \hat{P} \Rightarrow A \longrightarrow \alpha\beta \in \hat{P}$

Getting rid of ε

Lemma

For all CFG G exists CFG G' without ε-production or unit-production such that $L(G') = L(G) \setminus \{\varepsilon\}$.

Construction of G':

- $\hat{P} := P$
- `while` changes
  - $A \longrightarrow \alpha B \beta \in \hat{P} \wedge B \longrightarrow \varepsilon \in \hat{P} \Rightarrow A \longrightarrow \alpha\beta \in \hat{P}$
  - $A \longrightarrow B \in \hat{P} \wedge B \longrightarrow \gamma \in \hat{P} \Rightarrow A \longrightarrow \gamma \in \hat{P}$

Getting rid of ε

Lemma

For all CFG G exists CFG G' without ε-production or unit-production such that $L(G') = L(G) \setminus \{\varepsilon\}$.

Construction of G':

- $\hat{P} := P$
- `while` changes
  - $A \to \alpha B \beta \in \hat{P} \wedge B \to \varepsilon \in \hat{P} \Rightarrow A \to \alpha\beta \in \hat{P}$
  - $A \to B \in \hat{P} \wedge B \to \gamma \in \hat{P} \Rightarrow A \to \gamma \in \hat{P}$
- for P' delete all ε-productions and unit-productions from $\hat{P}^*$

Constructing a CNF

G context-free grammar without ε-productions or unit-productions

1. replace all $a \in \Sigma$ on right-hand sides by new variable A_a and introduce $A_a \longrightarrow a$
2. for all $A \longrightarrow B_1 \dots B_k$ introduce $A \longrightarrow B_1 C$ and $C \longrightarrow B_2 \dots B_k$ for fresh variable C

Constructing a GNF: Step 1

G grammar in CNF (for convenience)

G grammar in CNF (for convenience)

Definition

- $\alpha \rightarrow^L_G \beta$ leftmost derivation: derive β from α by always replacing the left-most variable

Constructing a GNF: Step 1

G grammar in CNF (for convenience)

Definition

- $\alpha \to_G^L \beta$ leftmost derivation: derive β from α by always replacing the left-most variable
- $R_{A,a} = \{\beta \in N^* | \, A \to_G^L a\beta\}$ (regular over N)

Constructing a GNF: Step 1

G grammar in CNF (for convenience)

Definition

- $\alpha \to_G^L \beta$ leftmost derivation: derive β from α by always replacing the left-most variable
- $R_{A,a} = \{\beta \in N^* \mid A \to_G^L a\beta\}$ (regular over N)
- $G_{A,a}$ grammar with $L(G_{A,a}) = R_{A,a}$

Constructing a GNF: Step 1

G grammar in CNF (for convenience)

Definition

- $\alpha \rightarrow_G^L \beta$ leftmost derivation: derive β from α by always replacing the left-most variable
- $R_{A,a} = \{\beta \in N^* \mid A \rightarrow_G^L a\beta\}$ (regular over N)
- $G_{A,a}$ grammar with $L(G_{A,a}) = R_{A,a}$
- w.l.o.g. $T_{A,a}$ start symbol of $G_{A,a}$

Constructing a GNF: Step 1

G grammar in CNF (for convenience)

Definition

- $\alpha \rightarrow^L_G \beta$ leftmost derivation: derive β from α by always replacing the left-most variable
- $R_{A,a} = \{\beta \in N^* | A \rightarrow^L_G a\beta\}$ (regular over N)
- $G_{A,a}$ grammar with $L(G_{A,a}) = R_{A,a}$
- w.l.o.g. $T_{A,a}$ start symbol of $G_{A,a}$
- w.l.o.g. variables of $G_{A,a}$ and G disjoint (renaming)

Properties

- $G_{A,a}$ strongly right-linear, i.e. all productions of form
  - $X \longrightarrow BY$ for X, Y non-terminals of $G_{A,a}$, $B \in N$
  - $X \longrightarrow \varepsilon$

Constructing a GNF: Step 2

Constructing G_1

- take all non-terminals and productions from $G_{A,a}$ and G
- S start symbol of G_1
- $\Rightarrow$ production in G_1 of form
  - $X \to b$
  - $X \to \varepsilon$
  - $X \to BY$

Constructing a GNF: Step 3

Constructing G_2

- replace every $X \to BY$ by $X \to bT_{B,b}Y$

Constructing a GNF: Step 3

Constructing G_2

- replace every $X \to BY$ by $X \to bT_{B,b}Y$

Constructing G_3

- get rid of ε-transitions by known mechanism

Correctness of Construction

Lemma

$\forall X \in N \forall x \in \Sigma^* : (X \rightarrow^*_{G_1} x \Leftrightarrow X \rightarrow^*_{G_2} x)$

Correctness of Construction

Lemma

$\forall X \in N \forall x \in \Sigma^* : (X \rightarrow^*_{G_1} x \Leftrightarrow X \rightarrow^*_{G_2} x)$

Theorem

G_3 is in GNF and $L(G_3) = L(G)$ holds.

Correctness of Construction

Lemma

$\forall X \in N \forall x \in \Sigma^* : (X \rightarrow^*_{G_1} x \Leftrightarrow X \rightarrow^*_{G_2} x)$

Theorem

G_3 is in GNF and $L(G_3) = L(G)$ holds.

having in mind that ε is not producible we are able to use CNF in GNF or CNF whenever we want

Pushdown Automata

How to transform Grammars in Automata?

- let's try it with the idea of a memory

How to transform Grammars in Automata?

- let's try it with the idea of a memory
- everything as simple as possible:
  - Automaton: only reading from left to right
  - Memory: stack (last-in-first-out)

How to transform Grammars in Automata?

- let's try it with the idea of a memory
- everything as simple as possible:
  - Automaton: only reading from left to right
  - Memory: stack (last-in-first-out)
- we have access to:
  - 1 letter of the word
  - top element of the stack
  - state the machine is in

Pushdown Automata

Definition

$\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ PDA iff

Pushdown Automata

Definition

$\mathscr{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ PDA iff

- Q finite set of states, q_0 initial state

Pushdown Automata

Definition

$\mathscr{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \bot, F)$ PDA iff

- Q finite set of states, q_0 initial state
- Σ input alphabet, Γ stack alphabet

Pushdown Automata

Definition

$\mathscr{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ PDA iff

- Q finite set of states, q_0 initial state
- Σ input alphabet, Γ stack alphabet
- $\perp$ inital stack symbol

Pushdown Automata

Definition

$\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ PDA iff

- Q finite set of states, q_0 initial state
- Σ input alphabet, Γ stack alphabet
- $\perp$ inital stack symbol
- $F \subseteq Q$ final states

Pushdown Automata

Definition

$\mathscr{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ PDA iff

- Q finite set of states, q_0 initial state
- Σ input alphabet, Γ stack alphabet
- $\perp$ inital stack symbol
- $F \subseteq Q$ final states
- $\Delta \subseteq (Q \times \Sigma \cup \{\varepsilon\} \cup \Gamma) \times (Q \times \Gamma^*)$

Pushdown Automata

Definition

$\mathscr{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ PDA iff

- Q finite set of states, q_0 initial state
- Σ input alphabet, Γ stack alphabet
- $\perp$ inital stack symbol
- $F \subseteq Q$ final states
- $\Delta \subseteq (Q \times \Sigma \cup \{\varepsilon\} \cup \Gamma) \times (Q \times \Gamma^*)$

Notice: PDAs are non-deterministic!

Explanations

- $((p, a, \alpha), (q, \beta)) \in \Delta$:
  - I am in p,
  - I read a,
  - α is on the stack,
  - I go to q, and
  - I write β on the stack

Configurations of a PDA

informal: what is the current state of the machine?

Configurations of a PDA

informal: what is the current state of the machine?

Definition

- start configuration $(q_0, w, \perp)$
- next configuration relation
  - 1 step: $\xrightarrow[\mathcal{A}]{1}$ defined by

$$((p,a,\alpha),(q,\gamma)) \in \Delta \Rightarrow$$

$$\forall y \in \Sigma^* \forall \beta \in \Gamma^* : (p, ay, \alpha\beta) \xrightarrow[\mathcal{A}]{1} (q, y, \gamma\beta)$$

- $C \xrightarrow[\mathcal{A}]{n} D$, $C \xrightarrow[\mathcal{A}]{*} D$ as usual

Acceptances

What could it mean that PDA *accepts* word w?

Acceptances

What could it mean that PDA *accepts* word w?

- PDA is in final state

Acceptances

What could it mean that PDA *accepts* word w?

- PDA is in final state
- stack is empty

Acceptances

Definition

$\mathscr{A}$ accepts w by final state: $\exists q \in F \exists \gamma \in \Gamma^* : (q_0, w, \perp) \xrightarrow[\mathscr{A}]{*} (q, \varepsilon, \gamma)$

Acceptances

Definition

$\mathcal{A}$ accepts w by final state: $\exists q \in F \exists \gamma \in \Gamma^* : (q_0, w, \perp) \xrightarrow[\mathcal{A}]{*} (q, \varepsilon, \gamma)$

Definition

$\mathcal{A}$ accepts w by empty stack: $\exists q \in Q : (q_0, w, \perp) \xrightarrow[\mathcal{A}]{*} (q, \varepsilon, \varepsilon)$

Technical Remarks about PDAs

- determistic variant possible (see later)

Technical Remarks about PDAs

- determistic variant possible (see later)
- $\perp$ only for defining start configuration

Technical Remarks about PDAs

- determistic variant possible (see later)
- $\perp$ only for defining start configuration
- PDA can be stuck, if stack symbol does not match any transition

Technical Remarks about PDAs

- determistic variant possible (see later)
- $\perp$ only for defining start configuration
- PDA can be stuck, if stack symbol does not match any transition
- the infinitly short time between popping and pushing **does not** count as empty stack

PDAs and CFGs

PDAs and CFGs

Did we do the right thing, i.e. do we have an automata-model being equivalent to context-free grammars?

PDAs and CFGs

Did we do the right thing, i.e. do we have an automata-model being equivalent to context-free grammars?

Let's try to prove it.

CFG $\longrightarrow$ PDA

Given: CFG $G = (V, \Sigma, P, S)$ w.l.o.g. in GNF

- PDA $\mathcal{A} = (\{q\}, \Sigma, V, \Delta, q, S, \emptyset)$ with acceptance by empty stack and
- $((q, c, A), (q, B_1 B_2 \ldots B_k)) \in \Delta$ iff $A \longrightarrow c B_1 B_2 \ldots B_k$ in P

CFG $\longrightarrow$ PDA

Given: CFG $G = (V, \Sigma, P, S)$ w.l.o.g. in GNF

- PDA $\mathcal{A} = (\{q\}, \Sigma, V, \Delta, q, S, \emptyset)$ with acceptance by empty stack and
- $((q, c, A), (q, B_1 B_2 \ldots B_k)) \in \Delta$ iff $A \longrightarrow c B_1 B_2 \ldots B_k$ in P

Plan: Prove that leftmost derivation corresponds to accepting computation in $\mathcal{A}$

Proof

Lemma

$\forall z, y \in \Sigma^* \forall \gamma \in N^* \forall A \in N$:

$$A \xrightarrow[G,L]{n} z\gamma \Leftrightarrow (q, zy, A) \xrightarrow[\mathcal{A}]{n} (q, y, \gamma)$$

Proof

Lemma

$\forall z, y \in \Sigma^* \forall \gamma \in N^* \forall A \in N$:

$$A \xrightarrow[G,L]{n} z\gamma \Leftrightarrow (q, zy, A) \xrightarrow[\mathcal{A}]{n} (q, y, \gamma)$$

Proof (induction on n):

- $n = 0 \Rightarrow$

$$A \xrightarrow[G]{0} z\gamma \Leftrightarrow A = z\gamma \Leftrightarrow z = \varepsilon \wedge \gamma = A$$

$$\Leftrightarrow (q, zy, A) = (q, y, \gamma)$$

$$\Leftrightarrow (q, zy, A) \xrightarrow[\mathcal{A}]{0} (q, y, \gamma)$$

Proof: Induction Step, $\Rightarrow$

- $A \xrightarrow[G,L]{n+1} z\gamma$

Proof: Induction Step, $\Rightarrow$

- $A \xrightarrow[G,L]{n+1} z\gamma$
- assume: $B \to c\beta$ was last production applied
  ($c \in \Sigma \cup \{\varepsilon\}, \beta \in V^*$)

Proof: Induction Step, $\Rightarrow$

- $A \xrightarrow[G,L]{n+1} z\gamma$
- assume: $B \to c\beta$ was last production applied ($c \in \Sigma \cup \{\varepsilon\}, \beta \in V^*$)
- left-most derivation $\Rightarrow$ before B only $u \in \Sigma^*$

Proof: Induction Step, $\Rightarrow$

- $A \xrightarrow[G,L]{n+1} z\gamma$
- assume: $B \rightarrow c\beta$ was last production applied ($c \in \Sigma \cup \{\varepsilon\}, \beta \in V^*$)
- left-most derivation $\Rightarrow$ before B only $u \in \Sigma^*$
- $\Rightarrow A \xrightarrow[G,L]{n} uB\alpha \xrightarrow[G,L]{1} uc\beta\alpha = z\gamma \Rightarrow z = uc, \gamma = \beta\alpha$

Proof: Induction Step, $\Longrightarrow$

- $A \xrightarrow[G,L]{n+1} z\gamma$
- assume: $B \longrightarrow c\beta$ was last production applied ($c \in \Sigma \cup \{\varepsilon\}, \beta \in V^*$)
- left-most derivation $\Rightarrow$ before B only $u \in \Sigma^*$
- $\Rightarrow A \xrightarrow[G,L]{n} uB\alpha \xrightarrow[G,L]{1} uc\beta\alpha = z\gamma \Rightarrow z = uc, \gamma = \beta\alpha$
- IH $\Rightarrow (q, ucy, A) \xrightarrow[\mathcal{A}]{n} (q, cy, B\alpha)$

Proof: Induction Step, $\Rightarrow$

- $A \xrightarrow[G,L]{n+1} z\gamma$
- assume: $B \longrightarrow c\beta$ was last production applied ($c \in \Sigma \cup \{\varepsilon\}, \beta \in V^*$)
- left-most derivation $\Rightarrow$ before B only $u \in \Sigma^*$
- $\Rightarrow A \xrightarrow[G,L]{n} uB\alpha \xrightarrow[G,L]{1} uc\beta\alpha = z\gamma \Rightarrow z = uc, \gamma = \beta\alpha$
- IH $\Rightarrow (q, ucy, A) \xrightarrow[\mathscr{A}]{n} (q, cy, B\alpha)$
- definition of $\mathscr{A} \Rightarrow ((q, c, B), (q, \beta)) \in \Delta$

Proof: Induction Step, $\Rightarrow$

- $A \xrightarrow[G,L]{n+1} z\gamma$
- assume: $B \longrightarrow c\beta$ was last production applied ($c \in \Sigma \cup \{\varepsilon\}, \beta \in V^*$)
- left-most derivation $\Rightarrow$ before B only $u \in \Sigma^*$
- $\Rightarrow A \xrightarrow[G,L]{n} uB\alpha \xrightarrow[G,L]{1} uc\beta\alpha = z\gamma \Rightarrow z = uc, \gamma = \beta\alpha$
- IH $\Rightarrow (q, ucy, A) \xrightarrow[\mathcal{A}]{n} (q, cy, B\alpha)$
- definition of $\mathcal{A} \Rightarrow ((q, c, B), (q, \beta)) \in \Delta$
- $\Rightarrow (q, cy, B\alpha) \xrightarrow[\mathcal{A}]{1} (q, y, \beta\alpha)$

Proof: Induction Step, $\Rightarrow$

- $A \xrightarrow[G,L]{n+1} z\gamma$
- assume: $B \longrightarrow c\beta$ was last production applied ($c \in \Sigma \cup \{\varepsilon\}, \beta \in V^*$)
- left-most derivation $\Rightarrow$ before B only $u \in \Sigma^*$
- $\Rightarrow A \xrightarrow[G,L]{n} uB\alpha \xrightarrow[G,L]{1} uc\beta\alpha = z\gamma \Rightarrow z = uc, \gamma = \beta\alpha$
- IH $\Rightarrow (q, ucy, A) \xrightarrow[\mathcal{A}]{n} (q, cy, B\alpha)$
- definition of $\mathcal{A} \Rightarrow ((q, c, B), (q, \beta)) \in \Delta$
- $\Rightarrow (q, cy, B\alpha) \xrightarrow[\mathcal{A}]{1} (q, y, \beta\alpha)$
- $\Rightarrow (q, zy, A) = (q, ucy, A) \xrightarrow[\mathcal{A}]{n+1} (q, y, \beta\alpha) = (q, y, \gamma)$

Proof: Induction Step, $\Leftarrow$

- $(q, zy, A) \xrightarrow[\mathcal{A}]{n+1} (q, y, \gamma)$

Proof: Induction Step, $\Leftarrow$

- $(q, zy, A) \xrightarrow[\mathcal{A}]{n+1} (q, y, \gamma)$
- assume $((q, c, B), (q, \beta)) \in \Delta$ last transition taken

Proof: Induction Step, $\Leftarrow$

- $(q, zy, A) \xrightarrow[\mathcal{A}]{n+1} (q, y, \gamma)$
- assume $((q, c, B), (q, \beta)) \in \Delta$ last transition taken
- $\Rightarrow z = uc, u \in \Sigma^*, \gamma = \beta\alpha, \alpha \in \Gamma^*$

Proof: Induction Step, $\Leftarrow$

- $(q, zy, A) \xrightarrow[\mathcal{A}]{n+1} (q, y, \gamma)$
- assume $((q, c, B), (q, \beta)) \in \Delta$ last transition taken
- $\Rightarrow z = uc,\ u \in \Sigma^*,\ \gamma = \beta\alpha,\ \alpha \in \Gamma^*$
- $\Rightarrow (q, ucy, A) \xrightarrow[\mathcal{A}]{n} (q, cy, B\alpha) \xrightarrow[\mathcal{A}]{1} (q, y, \beta\alpha)$

Proof: Induction Step, $\Leftarrow$

- $(q, zy, A) \xrightarrow[\mathcal{A}]{n+1} (q, y, \gamma)$
- assume $((q, c, B), (q, \beta)) \in \Delta$ last transition taken
- $\Rightarrow z = uc, u \in \Sigma^*, \gamma = \beta\alpha, \alpha \in \Gamma^*$
- $\Rightarrow (q, ucy, A) \xrightarrow[\mathcal{A}]{n} (q, cy, B\alpha) \xrightarrow[\mathcal{A}]{1} (q, y, \beta\alpha)$
- IH $\Rightarrow A \xrightarrow[G,L]{n} uB\alpha$

Proof: Induction Step, $\Leftarrow$

- $(q, zy, A) \xrightarrow[\mathcal{A}]{n+1} (q, y, \gamma)$
- assume $((q, c, B), (q, \beta)) \in \Delta$ last transition taken
- $\Rightarrow z = uc, u \in \Sigma^*, \gamma = \beta\alpha, \alpha \in \Gamma^*$
- $\Rightarrow (q, ucy, A) \xrightarrow[\mathcal{A}]{n} (q, cy, B\alpha) \xrightarrow[\mathcal{A}]{1} (q, y, \beta\alpha)$
- IH $\Rightarrow A \xrightarrow[G,L]{n} uB\alpha$
- $B \to c\beta$ production in G

Proof: Induction Step, $\Leftarrow$

- $(q, zy, A) \xrightarrow[\mathcal{A}]{n+1} (q, y, \gamma)$
- assume $((q, c, B), (q, \beta)) \in \Delta$ last transition taken
- $\Rightarrow z = uc, u \in \Sigma^*, \gamma = \beta\alpha, \alpha \in \Gamma^*$
- $\Rightarrow (q, ucy, A) \xrightarrow[\mathcal{A}]{n} (q, cy, B\alpha) \xrightarrow[\mathcal{A}]{1} (q, y, \beta\alpha)$
- IH $\Rightarrow A \xrightarrow[G,L]{n} uB\alpha$
- $B \to c\beta$ production in G
- $\Rightarrow A \xrightarrow[G]{n} uB\alpha \xrightarrow[G]{1} uc\beta\alpha = z\gamma$

Proof: last step

Thus we proved

Theorem

$L(\mathscr{A}) = L(G)$

Simulating PDAs by CFGs

Idea of the Construction

Two steps:

Idea of the Construction

Two steps:

1. every PDA can be simutated by PDA with one state

Idea of the Construction

Two steps:

1. every PDA can be simutated by PDA with one state
2. every PDA with one state is equivalent to CFG

ad Step (2)

- Construction from CFG→PDA is invertible:

ad Step (2)

- Construction from CFG→PDA is invertible:
  - given $\mathscr{A} = (\{q\}, \Sigma, \Gamma, \Delta, q, \perp, \emptyset)$

ad Step (2)

- Construction from CFG→PDA is invertible:
  - given $\mathcal{A} = (\{q\}, \Sigma, \Gamma, \Delta, q, \perp, \emptyset)$
  - set $G = (\Gamma, \Sigma, P, \perp)$ where P contains production

ad Step (2)

- Construction from CFG→PDA is invertible:
  - given $\mathscr{A} = (\{q\}, \Sigma, \Gamma, \Delta, q, \bot, \emptyset)$
  - set $G = (\Gamma, \Sigma, P, \bot)$ where P contains production
  - $A \longrightarrow cB_1 \ldots B_k$ for all $((q, c, A), (q, B_1 \ldots B_k)) \in \Delta$

ad Step (2)

- Construction from CFG→PDA is invertible:
  - given $\mathscr{A} = (\{q\}, \Sigma, \Gamma, \Delta, q, \bot, \emptyset)$
  - set $G = (\Gamma, \Sigma, P, \bot)$ where P contains production
  - $A \longrightarrow cB_1 \ldots B_k$ for all $((q, c, A), (q, B_1 \ldots B_k)) \in \Delta$
- Proof is analogous

ad Step (1)

Idea: keep some state-information on the stack

ad Step (1)

Idea: keep some state-information on the stack

- w.l.o.g. $\mathscr{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, \{q_f\})$

ad Step (1)

Idea: keep some state-information on the stack

- w.l.o.g. $\mathscr{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, \{q_f\})$
- $\Gamma' := Q \times \Gamma \times Q$

Idea: keep some state-information on the stack

- w.l.o.g. $\mathscr{A} = (Q, \Sigma, \Gamma, \Delta, q_0, \bot, \{q_f\})$
- $\Gamma' := Q \times \Gamma \times Q$
- $\mathscr{A}' = (\{q\}, \Sigma, \Gamma', \Delta', q, (q_0, \bot, t), \emptyset)$

ad Step (1): Defining Δ'

- $((p_1, c, A), (p_2, \varepsilon)) \in \Delta \Rightarrow$

$$((q, c, (p_1, A, p_2)), (q, \varepsilon)) \in \Delta'.$$

ad Step (1): Defining Δ'

- $((p_1, c, A), (p_2, \varepsilon)) \in \Delta \Rightarrow$

$$((q, c, (p_1, A, p_2)), (q, \varepsilon)) \in \Delta'.$$

- $((p_1, c, A), (p_2, B_1 \ldots B_k)) \in \Delta \Rightarrow$

$$((q, c, (p_1, A, q_{k+1})), (q, (q_1, B_1, q_2) \ldots (q_k B_k q_{k+1}))) \in \Delta'$$

ad Step (1): Defining Δ'

- $((p_1, c, A), (p_2, \varepsilon)) \in \Delta \Longrightarrow$

$$((q, c, (p_1, A, p_2)), (q, \varepsilon)) \in \Delta'.$$

- $((p_1, c, A), (p_2, B_1 \ldots B_k)) \in \Delta \Longrightarrow$

$$((q, c, (p_1, A, q_{k+1})), (q, (q_1, B_1, q_2) \ldots (q_k B_k q_{k+1}))) \in \Delta'$$

Intuition: $\mathscr{A}'$ simulates $\mathscr{A}$ by guessing in what state $\mathscr{A}$ will be and saving those guesses on the stack

ad Step (1)

Lemma

$$(p_1, x, B_1 \dots B_k) \xrightarrow[\mathcal{A}]{n} (p_2, \varepsilon, \varepsilon) \Leftrightarrow$$

$$\exists q_1, \dots, q_k : p_1 = q_1, p_2 = q_k \wedge$$

$$(q, x, (q_1, B_1, q_2) \dots (q_k, B_k, q_k)) \xrightarrow[\mathcal{A}']{n} (q, \varepsilon, \varepsilon)$$

ad Step (1)

Theorem

$L(\mathscr{A}') = L(\mathscr{A})$

ad Step (1)

Theorem

$L(\mathscr{A}') = L(\mathscr{A})$

Proof: $\forall x \in \Sigma^* :$

$$
\begin{aligned}
x \in L(\mathscr{A}') &\Leftrightarrow (q, x, (q_0, \bot, q_f)) \xrightarrow[\mathscr{A}']{n} (q, \varepsilon, \varepsilon) \\
&\Leftrightarrow (q_0, x, \bot) \xrightarrow[\mathscr{A}]{*} (q_f, \varepsilon, \varepsilon) \\
&\Leftrightarrow x \in L(\mathscr{A})
\end{aligned}
$$

Deterministic Pushdown Automata

Deterministic Pushdown Automata

Definition

$\mathscr{A} = (Q, \Sigma, \Gamma, \delta, \perp, \dashv, q_0, F)$ DPDA iff

- $Q, \Sigma, \Gamma, \perp, q_0, F$ as in PDA
- $\dashv$ right endmarker (end of the word)
- $\delta : Q \times (\Sigma \cup \{\dashv, \varepsilon\}) \times \Gamma \to Q \times \Gamma^*$
- $\perp$ has to remain at the bottom of the stack (deadlock prevention)
- acceptance only by final state

Configuration, Acceptance by DPDA

Definition

- start configuration: $(q_0, x \dashv, \perp)$
- $\mathscr{A}$ accepts x: $(q_0, x \dashv, \perp) \xrightarrow[\mathscr{A}]{*} (q_f, \varepsilon, \beta)$
- language deterministic context-free: accepted by DPDA

DPDAs are closed under complement

Lemma

If L is a deterministic context-free language, then $\Sigma^* \backslash L$ is as well.

DPDAs are closed under complement

Lemma

If L is a deterministic context-free language, then $\Sigma^* \backslash L$ is as well.

Proof:

DPDAs are closed under complement

Lemma

If L is a deterministic context-free language, then $\Sigma^* \backslash L$ is as well.

Proof:

- $\mathscr{A}$ DPDA for L

DPDAs are closed under complement

Lemma

If L is a deterministic context-free language, then $\Sigma^*\backslash L$ is as well.

Proof:

- $\mathscr{A}$ DPDA for L
- we have to construct $\mathscr{A}'$ for $\Sigma^*\backslash L$

DPDAs are closed under complement

Lemma

If L is a deterministic context-free language, then $\Sigma^*\backslash L$ is as well.

Proof:

- $\mathscr{A}$ DPDA for L
- we have to construct $\mathscr{A}'$ for $\Sigma^*\backslash L$
- problem: switching F and $Q\backslash F$ is not possible

DPDAs are closed under complement

Lemma

If L is a deterministic context-free language, then $\Sigma^* \backslash L$ is as well.

Proof:

- $\mathscr{A}$ DPDA for L
- we have to construct $\mathscr{A}'$ for $\Sigma^* \backslash L$
- problem: switching F and $Q \backslash F$ is not possible
  - DPDAs have to scan the complete input

DPDAs are closed under complement

Lemma

If L is a deterministic context-free language, then $\Sigma^*\backslash L$ is as well.

Proof:

- $\mathscr{A}$ DPDA for L
- we have to construct $\mathscr{A}'$ for $\Sigma^*\backslash L$
- problem: switching F and $Q\backslash F$ is not possible
  - DPDAs have to scan the complete input
  - $\Rightarrow$ may loop infinitely on not accepted inputs

Proof Cont.

Trick: modify $\mathscr{A}$ such that we know exactly if we have already seen ⊣

Proof Cont.

Trick: modify $\mathscr{A}$ such that we know exactly if we have already seen $\dashv$

- $Q' = \{q' \mid q \in Q\}$ (disjoint duplication of Q)

Proof Cont.

Trick: modify $\mathscr{A}$ such that we know exactly if we have already seen $\dashv$

- $Q' = \{q' \mid q \in Q\}$ (disjoint duplication of Q)
- new transitions:
  - $\delta(p', a, A) = (q', \beta)$ for $\delta(p, a, A) = (q, \beta)$

Proof Cont.

Trick: modify $\mathscr{A}$ such that we know exactly if we have already seen $\dashv$

- $Q' = \{q' \mid q \in Q\}$ (disjoint duplication of Q)
- new transitions:
  - $\delta(p', a, A) = (q', \beta)$ for $\delta(p, a, A) = (q, \beta)$
  - replace $\delta(p, \dashv, A) = (q, \beta)$ by $\delta(p, \dashv, A) = (q', \beta)$

Proof Cont.

Trick: modify $\mathscr{A}$ such that we know exactly if we have already seen $\dashv$

- $Q' = \{q' \mid q \in Q\}$ (disjoint duplication of Q)
- new transitions:
  - $\delta(p', a, A) = (q', \beta)$ for $\delta(p, a, A) = (q, \beta)$
  - replace $\delta(p, \dashv, A) = (q, \beta)$ by $\delta(p, \dashv, A) = (q', \beta)$
- we switch to the primed version if we saw the endmarker

Proof Cont.

Trick: modify $\mathscr{A}$ such that we know exactly if we have already seen $\dashv$

- $Q' = \{q' \mid q \in Q\}$ (disjoint duplication of Q)
- new transitions:
  - $\delta(p', a, A) = (q', \beta)$ for $\delta(p, a, A) = (q, \beta)$
  - replace $\delta(p, \dashv, A) = (q, \beta)$ by $\delta(p, \dashv, A) = (q', \beta)$
- we switch to the primed version if we saw the endmarker
- $F' = \{q' \mid q \in F\}$

Proof Cont.

Trick: modify $\mathscr{A}$ such that we know exactly if we have already seen $\dashv$

- $Q' = \{q' \mid q \in Q\}$ (disjoint duplication of Q)
- new transitions:
  - $\delta(p', a, A) = (q', \beta)$ for $\delta(p, a, A) = (q, \beta)$
  - replace $\delta(p, \dashv, A) = (q, \beta)$ by $\delta(p, \dashv, A) = (q', \beta)$
- we switch to the primed version if we saw the endmarker
- $F' = \{q' \mid q \in F\}$
- $\mathscr{A}' = (Q \cup Q', \Sigma, \Gamma, \delta', \bot, \dashv, q_0, F')$

Proof Cont.

Trick: modify $\mathscr{A}$ such that we know exactly if we have already seen $\dashv$

- $Q' = \{q' \mid q \in Q\}$ (disjoint duplication of Q)
- new transitions:
  - $\delta(p', a, A) = (q', \beta)$ for $\delta(p, a, A) = (q, \beta)$
  - replace $\delta(p, \dashv, A) = (q, \beta)$ by $\delta(p, \dashv, A) = (q', \beta)$
- we switch to the primed version if we saw the endmarker
- $F' = \{q' \mid q \in F\}$
- $\mathscr{A}' = (Q \cup Q', \Sigma, \Gamma, \delta', \bot, \dashv, q_0, F')$
- $L(\mathscr{A}') = \Sigma^* \setminus L$ and $\mathscr{A}'$ is DPDA

Proof Cont

Stopping the machine:

Proof Cont

Stopping the machine:

- we are only in primed states if we saw ⊣

Proof Cont

Stopping the machine:

- we are only in primed states if we saw $\dashv$
- $\Rightarrow$ we can rest in a final state and stop doing funny things

Proof Cont

Stopping the machine:

- we are only in primed states if we saw $\dashv$
- $\Rightarrow$ we can rest in a final state and stop doing funny things
- redefine $\delta(p', \varepsilon, A) = (p', A)$ if the image was (q', β) for $p' \in F'$

Proof: Getting rid of Spurious Loops

- new states
  - $r \in Q' \backslash F'$: move to the end, don't change the stack

Proof: Getting rid of Spurious Loops

- new states
  - $r \in Q' \backslash F'$: move to the end, don't change the stack
  - $r' \in Q' \backslash F'$: reject

Proof: Getting rid of Spurious Loops

- new states
  - $r \in Q' \setminus F'$: move to the end, don't change the stack
  - $r' \in Q' \setminus F'$: reject
  - $\delta(r, a, A) = (r, A)$ for $a \in \Sigma, A \in \Gamma$

Proof: Getting rid of Spurious Loops

- new states
  - $r \in Q' \setminus F'$: move to the end, don't change the stack
  - $r' \in Q' \setminus F'$: reject
  - $\delta(r, a, A) = (r, A)$ for $a \in \Sigma, A \in \Gamma$
  - $\delta(r, \vdash, A) = (r', A)$ for $A \in \Gamma$

Proof: Getting rid of Spurious Loops

- new states
  - $r \in Q' \backslash F'$: move to the end, don't change the stack
  - $r' \in Q' \backslash F'$: reject
  - $\delta(r, a, A) = (r, A)$ for $a \in \Sigma, A \in \Gamma$
  - $\delta(r, \vdash, A) = (r', A)$ for $A \in \Gamma$
  - $\delta(r', \varepsilon, A) = (r', A)$ for $A \in \Gamma$

Proof: Getting rid of Spurious Loops

- new states
  - $r \in Q' \backslash F'$: move to the end, don't change the stack
  - $r' \in Q' \backslash F'$: reject
  - $\delta(r, a, A) = (r, A)$ for $a \in \Sigma, A \in \Gamma$
  - $\delta(r, \vdash, A) = (r', A)$ for $A \in \Gamma$
  - $\delta(r', \varepsilon, A) = (r', A)$ for $A \in \Gamma$
- replace $\delta(p, \varepsilon, A) = (q, \beta)$ by $\delta(p, \varepsilon, A) = (r, A)$

Proof: Getting rid of Spurious Loops

- new states
  - $r \in Q' \backslash F'$: move to the end, don't change the stack
  - $r' \in Q' \backslash F'$: reject
  - $\delta(r, a, A) = (r, A)$ for $a \in \Sigma, A \in \Gamma$
  - $\delta(r, \vdash, A) = (r', A)$ for $A \in \Gamma$
  - $\delta(r', \varepsilon, A) = (r', A)$ for $A \in \Gamma$
- replace $\delta(p, \varepsilon, A) = (q, \beta)$ by $\delta(p, \varepsilon, A) = (r, A)$
- replace $\delta(p', \varepsilon, A) = (q, \beta)$ by $\delta(p, \varepsilon, A) = (r', A)$

DPDA - (N)PDA

- DFAs and NFAs are equivalent, have the same power

DPDA - (N)PDA

- DFAs and NFAs are equivalent, have the same power
- does this hold for pushdown automata as well?

PDAs are more powerful than DPDAs

Theorem

There exists languages recognizable by a PDA but not by a DPDA.

PDAs are more powerful than DPDAs

Theorem

There exists languages recognizable by a PDA but not by a DPDA.

Proof:

PDAs are more powerful than DPDAs

Theorem

There exists languages recognizable by a PDA but not by a DPDA.

Proof:

- $L = \{ww \mid w \in \Sigma^*\}$ not context-free.

PDAs are more powerful than DPDAs

Theorem

There exists languages recognizable by a PDA but not by a DPDA.

Proof:

- $L = \{ww \mid w \in \Sigma^*\}$ not context-free.
- $\overline{L} = \Sigma^* \backslash L$ context-free

PDAs are more powerful than DPDAs

Theorem

There exists languages recognizable by a PDA but not by a DPDA.

Proof:

- $L = \{ww \mid w \in \Sigma^*\}$ not context-free.
- $\overline{L} = \Sigma^* \backslash L$ context-free
- DPDA closed under complement $\Rightarrow$ L recognizable by DPDA $\Rightarrow$ Contradiction

The Cocke-Younger-Kasami Algorithm

Membership-Problem

Definition

Given a language L over Σ^* and a word $w \in \Sigma^*$, decide whether $w \in L$ or not.

Membership-Problem

Definition

Given a language L over Σ^* and a word $w \in \Sigma^*$, decide whether $w \in L$ or not.

- problem is in general hard to solve
- we can't build all paths in an automaton or all derivations in a grammar

CYK-Algorithm

Given context-free grammar G (w.l.o.g. in CNF) the CYK-algorithm decides *efficiently* whether a word is producible or not.

CYK-Algorithm

Given context-free grammar G (w.l.o.g. in CNF) the CYK-algorithm decides *efficiently* whether a word is producible or not.

Some Preliminaries:

CYK-Algorithm

Given context-free grammar G (w.l.o.g. in CNF) the CYK-algorithm decides *efficiently* whether a word is producible or not.

Some Preliminaries:

- $w\left\langle i, j\right\rangle = w[i+1\ldots j]$
  $(aabbab\left\langle 1,4\right\rangle = aabbab[2,\ldots,4] = abb)$

CYK-Algorithm

Given context-free grammar G (w.l.o.g. in CNF) the CYK-algorithm decides *efficiently* whether a word is producible or not.

Some Preliminaries:

- $w\left\langle i,j\right\rangle = w[i+1\dots j]$
  $(aabbab\left\langle 1,4\right\rangle = aabbab[2,\dots,4] = abb)$
- $T_{ij} \subseteq V$ generating $w\left\langle i,j\right\rangle$

CYK-Algorithm

Given context-free grammar G (w.l.o.g. in CNF) the CYK-algorithm decides *efficiently* whether a word is producible or not.

Some Preliminaries:

- $w\left\langle i,j\right\rangle = w[i+1\ldots j]$
  $(aabbab\,\langle 1,4\rangle = aabbab[2,\ldots,4] = abb)$
- $T_{ij} \subseteq V$ generating $w\left\langle i,j\right\rangle$
  - $A \to a \in P \Rightarrow aabbab\,\langle 0,1\rangle\,, aabbab\,\langle 1,2\rangle\,, aabbab\,\langle 4,5\rangle$ producible
    $\Rightarrow T_{01} = T_{12} = T_{45} = \{A\}$

CYK-Algorithm

Given context-free grammar G (w.l.o.g. in CNF) the CYK-algorithm decides *efficiently* whether a word is producible or not.

Some Preliminaries:

- $w\left\langle i,j\right\rangle = w[i+1\ldots j]$
  $(aabbab\left\langle 1,4\right\rangle = aabbab[2,\ldots,4] = abb)$
- $T_{ij} \subseteq V$ generating $w\left\langle i,j\right\rangle$
  - $A \rightarrow a \in P \Rightarrow aabbab\left\langle 0,1\right\rangle, aabbab\left\langle 1,2\right\rangle, aabbab\left\langle 4,5\right\rangle$ producible
    $\Rightarrow T_{01} = T_{12} = T_{45} = \{A\}$
- by this we get easily all factors of length 1

CYK-Algo Cont.

- consider w's factors of length 2

CYK-Algo Cont.

- consider w's factors of length 2
- $w[i \ldots i+1] = w[i]w[i+1] = w\langle i-1, i\rangle\, w\langle i, i+1\rangle$

CYK-Algo Cont.

- consider w's factors of length 2
- $w[i \dots i+1] = w[i]w[i+1] = w\langle i-1, i\rangle\, w\langle i, i+1\rangle$
- look-up productions for $w\langle i-1, i\rangle$ and $w\langle i, i+1\rangle$ in $T_{i-1,i}$ resp. $T_{i,i+1}$

CYK-Algo Cont.

- consider w's factors of length 2
- $w[i \dots i+1] = w[i]w[i+1] = w\langle i-1, i\rangle\, w\langle i, i+1\rangle$
- look-up productions for $w\langle i-1, i\rangle$ and $w\langle i, i+1\rangle$ in $T_{i-1,i}$ resp. $T_{i,i+1}$
- for all $X \in T_{i-1,i}$ and for all $Y \in T_{i,i+1}$ check if there is a production with right-hand side XY

CYK-Algo Cont.

- consider w's factors of length 2
- $w[i \dots i+1] = w[i]w[i+1] = w\langle i-1, i\rangle\, w\langle i, i+1\rangle$
- look-up productions for $w\langle i-1, i\rangle$ and $w\langle i, i+1\rangle$ in $T_{i-1,i}$ resp. $T_{i,i+1}$
- for all $X \in T_{i-1,i}$ and for all $Y \in T_{i,i+1}$ check if there is a production with right-hand side XY
- update $T_{i-1,i+1}$ by the corresponding left-hand side

CYK-Algo Cont.

in general:

- given a factor x of w of length k

CYK-Algo Cont.

in general:

- given a factor x of w of length k
- decompose x into two parts in all possible ways

CYK-Algo Cont.

in general:

- given a factor x of w of length k
- decompose x into two parts in all possible ways
- lookup the corresponding sets and see if a combination gives the right-hand side of a production

CYK-Algo Cont.

in general:

- given a factor x of w of length k
- decompose x into two parts in all possible ways
- lookup the corresponding sets and see if a combination gives the right-hand side of a production
- if so, take the left-hand side into the set corresponding to x

CYK-Algo Cont.

in general:

- given a factor x of w of length k
- decompose x into two parts in all possible ways
- lookup the corresponding sets and see if a combination gives the right-hand side of a production
- if so, take the left-hand side into the set corresponding to x
- if S is in $T_{0,n}$ for $|w| = n$ then $w \in L$

Closure-Properties for CFLs

CFLs are closed under

- union

Closure-Properties for CFLs

CFLs are closed under

- union
- concatenation

Closure-Properties for CFLs

CFLs are closed under

- union
- concatenation
- star

Closure-Properties for CFLs

CFLs are closed under

- union
- concatenation
- star

CFLs are not closed under intersection!

- But... the intersection of a regular language and a contex-free one is context-free

Closure-Properties for DCFLs

DCFLs are closed under intersection.

Closure-Properties for DCFLs

DCFLs are closed under intersection.

DCFLs are not closed under

- union

Closure-Properties for DCFLs

DCFLs are closed under intersection.

DCFLs are not closed under

- union
- reversal

The Chomsky-Schützenberger Theorem

The Language PAREN_n

Definition

PAREN_n (Dyck language) is generated by the grammar

$$S \longrightarrow [_1 S]_1 | \ldots | [_n S]_n | SS | \varepsilon$$

(n different kinds of parenthesis)

The Chomsky-Schützenberger-Theorem

Theorem (Chomsky-Schützenberger)

For every CFL $\mathscr{A}$ there exists an $n \in \mathbb{N}_0$, a regular language R, and a homomorphism h with

$$\mathscr{A} = h(\mathrm{PAREN}_n \cap R).$$