Foundations of Informatics: a Bridging Course

Week 3: Formal Languages and Semantics

Thomas Noll

Lehrstuhl für Informatik 2
RWTH Aachen University
noll@cs.rwth-aachen.de

http://www.b-it-center.de/Wob/en/view/class211_id569.html

B–IT, Bonn, Winter term 2006/07

Part II

Context–Free Languages

Outline

1. Context–Free Grammars and Languages

2. Context–Free and Regular Languages

3. The Word Problem for Context–Free Languages

4. The Emptiness Problem for Context–Free Languages

5. Pushdown Automata

6. Outlook

Introductory Example I

Example II.1

Syntax definition of programming languages by “Backus Naur” rules
Here: simple arithmetic expressions

$$
\begin{array}{lcl}
\langle \mathit{Expression} \rangle & ::= & \mathsf{0} \\
 & \mid & \mathsf{1} \\
 & \mid & \langle \mathit{Expression} \rangle + \langle \mathit{Expression} \rangle \\
 & \mid & \langle \mathit{Expression} \rangle * \langle \mathit{Expression} \rangle \\
 & \mid & (\langle \mathit{Expression} \rangle)
\end{array}
$$

Meaning:

> *An expression is either 0 or 1, or it is of the form $u + v$, $u * v$, or (u) where u, v are again expressions*

Introductory Example II

Example II.2 (continued)

Here we abbreviate $\langle \textit{Expression} \rangle$ as E, and use → instead of ::=. Thus:

$$E \quad \rightarrow \quad \mathsf{0} \mid \mathsf{1} \mid E + E \mid E * E \mid (E)$$

Introductory Example II

Example II.2 (continued)

Here we abbreviate $\langle \mathit{Expression} \rangle$ as E, and use → instead of ::=. Thus:

$$E \quad \to \quad \mathsf{0} \mid \mathsf{1} \mid E + E \mid E * E \mid (E)$$

Now expressions can be generated by applying rules to the start symbol E:

$$\begin{aligned} E &\Rightarrow E * E \\ &\Rightarrow (E) * E \\ &\Rightarrow (E) * \mathsf{1} \\ &\Rightarrow (E + E) * \mathsf{1} \\ &\Rightarrow (\mathsf{0} + E) * \mathsf{1} \\ &\Rightarrow (\mathsf{0} + \mathsf{1}) * \mathsf{1} \end{aligned}$$

Context–Free Grammars I

Definition II.3

A context–free grammar (CFG) is a quadruple

$$G = \langle N, \Sigma, P, S \rangle$$

where

- N is a finite set of nonterminal symbols
- Σ is the (finite) alphabet of terminal symbols (disjoint from N)
- P is a finite set of production rules of the form $A \rightarrow \alpha$ where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$
- $S \in N$ is a start symbol

Context–Free Grammars II

Example II.4

For the above example, we have:

- $N = \{E\}$
- $\Sigma = \{\mathsf{0}, \mathsf{1}, +, *, (,)\}$
- $P = \{E \to \mathsf{0}, E \to \mathsf{1}, E \to E + E, E \to E * E, E \to (E)\}$
- $S = E$

Context–Free Grammars II

Example II.4

For the above example, we have:

- $N = \{E\}$
- $\Sigma = \{\mathsf{0}, \mathsf{1}, +, *, (,)\}$
- $P = \{E \to \mathsf{0}, E \to \mathsf{1}, E \to E + E, E \to E * E, E \to (E)\}$
- $S = E$

Naming conventions:

- nonterminals start with uppercase letters
- terminals start with lowercase letters
- start symbol = symbol on LHS of first production

$\Longrightarrow$ grammar completely defined by productions

Context–Free Languages I

Definition II.5

Let $G = \langle N, \Sigma, P, S\rangle$ be a CFG.

- A sentence $\gamma \in (N \cup \Sigma)^*$ is directly derivable from $\beta \in (N \cup \Sigma)^*$ if there exist $\pi = A \to \alpha \in P$ and $\delta_1, \delta_2 \in (N \cup \Sigma)^*$ such that $\beta = \delta_1 A \delta_2$ and $\gamma = \delta_1 \alpha \delta_2$ (notation: $\beta \overset{\pi}{\Rightarrow} \gamma$ or just $\beta \Rightarrow \gamma$) .
- A derivation (of length n) of γ from β is a sequence of direct derivations of the form $\delta_0 \Rightarrow \delta_1 \Rightarrow \ldots \Rightarrow \delta_n$ where $\delta_0 = \beta$, $\delta_n = \gamma$, and $\delta_{i-1} \Rightarrow \delta_i$ for every $1 \leq i \leq n$ (notation: $\beta \Rightarrow^* \gamma$).
- A word $w \in \Sigma^*$ is called derivable in G if $S \Rightarrow^* w$.
- The language generated by G is $L(G) := \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.
- A language $L \subseteq \Sigma^*$ is called context–free (CFL) if it is generated by some CFG.
- Two grammars G_1, G_2 are equivalent if $L(G_1) = L(G_2)$.

Context–Free Languages II

Example II.6

The language $\{a^n b^n \mid n \in \mathbb{N}\}$ is context–free (but not regular—see previous part). It is generated by the grammar $G = \langle N, \Sigma, P, S\rangle$ with

- $N = \{S\}$
- $\Sigma = \{a, b\}$
- $P = \{S \to aSb \mid \varepsilon\}$

(proof: on the board)

Context–Free Languages II

Example II.6

The language $\{a^n b^n \mid n \in \mathbb{N}\}$ is context–free (but not regular—see previous part). It is generated by the grammar $G = \langle N, \Sigma, P, S \rangle$ with

- $N = \{S\}$
- $\Sigma = \{a, b\}$
- $P = \{S \rightarrow aSb \mid \varepsilon\}$

(proof: on the board)

Remark: illustration of derivations by derivation trees

- root labeled by start symbol
- leafs labeled by terminal symbols
- successors of node labeled according to right–hand side of production rule

(example on the board)

Context–Free Grammars and Languages

Seen:

- Context–free grammars
- Derivations
- Context–free languages

Context–Free Grammars and Languages

Seen:

- Context–free grammars
- Derivations
- Context–free languages

Open:

- Relation between context–free and regular languages

Outline

1. Context–Free Grammars and Languages
2. Context–Free and Regular Languages
3. The Word Problem for Context–Free Languages
4. The Emptiness Problem for Context–Free Languages
5. Pushdown Automata
6. Outlook

Context–Free and Regular Languages

Theorem II.7

1. *Every regular language is context–free.*
2. *There exist CFLs which are not regular.*

(In other words: the class of regular languages is a proper subset of the class of CFLs.)

Context–Free and Regular Languages

Theorem II.7

1. *Every regular language is context–free.*
2. *There exist CFLs which are not regular.*

(In other words: the class of regular languages is a proper subset of the class of CFLs.)

Proof.

1. Let L be a regular language, and let $\mathfrak{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ be a DFA which recognizes L. $G := \langle N, \Sigma, P, S\rangle$ is defined as follows:
   - $N := Q$, $S := q_0$
   - if $\delta(q, a) = q'$, then $q \to aq' \in P$
   - if $q \in F$, then $q \to \varepsilon \in P$

   Obviously a w–labeled run in $\mathfrak{A}$ from q_0 to F corresponds to a derivation of w in G, and vice versa. Thus $L(\mathfrak{A}) = L(G)$ (example on the board).
2. An example is $\{a^n b^n \mid n \in \mathbb{N}\}$.

Context–Free Grammars and Languages

Seen:

- CFLs are more expressive than regular languages

Context–Free Grammars and Languages

Seen:

- CFLs are more expressive than regular languages

Open:

- Decidability of word problem

Outline

1. Context–Free Grammars and Languages

2. Context–Free and Regular Languages

3. The Word Problem for Context–Free Languages

4. The Emptiness Problem for Context–Free Languages

5. Pushdown Automata

6. Outlook

The Word Problem

- **Goal:** given $G = \langle N, \Sigma, P, S \rangle$ and $w \in \Sigma^*$, decide whether $w \in L(G)$ or not
- For regular languages this was easy: just let the corresponding DFA run on w.
- But here: how to decide when to stop a derivation?
- **Solution:** establish normal form for grammars which guarantees that each nonterminal produces at least one terminal symbol

$\Longrightarrow$ only finitely many combinations

Chomsky Normal Form I

Definition II.8

A CFG is in Chomsky normal form (Chomsky NF) if every of its productions is of the form

$$A \to BC \quad \text{or} \quad A \to a$$

(and maybe $S \to \varepsilon$, in which case S does not occur on the right–hand side of any production).

Chomsky Normal Form I

Definition II.8

A CFG is in Chomsky normal form (Chomsky NF) if every of its productions is of the form

$$A \to BC \quad \text{or} \quad A \to a$$

(and maybe $S \to \varepsilon$, in which case S does not occur on the right–hand side of any production).

Example II.9

Let $S \to aSb \mid \varepsilon$ be the known grammar which generates $L := \{a^n b^n \mid n \in \mathbb{N}\}$. An equivalent grammar in Chomsky NF is

$$\begin{array}{ll} S \to \varepsilon \mid AC & \text{(generates } L\text{)} \\ A \to a & \text{(generates } \{a\}\text{)} \\ B \to b & \text{(generates } \{b\}\text{)} \\ C \to SB & \text{(generates } \{a^n b^{n+1} \mid n \in \mathbb{N}\}\text{)} \end{array}$$

Chomsky Normal Form II

Theorem II.10

Every CFL is generatable by a CFG in Chomsky NF.

Chomsky Normal Form II

Theorem II.10

Every CFL is generatable by a CFG in Chomsky NF.

Proof.

Let L be a CFL, and let $G = \langle N, \Sigma, P, S \rangle$ be some CFG which generates L. The transformation of P into rules of the form $A \longrightarrow BC$ and $A \longrightarrow a$ proceeds in three steps:

1. terminal symbols only in rules of the form $A \longrightarrow a$ (thus all other rules have the shape $A \longrightarrow A_1 \ldots A_n$)
2. elimination of rules of the form $A \longrightarrow B$
3. elimination of rules of the form $A \longrightarrow A_1 \ldots A_n$ where $n > 2$

□

Chomsky Normal Form III

Proof of Theorem II.10 (continued).

Step 1: (only $A \to a$)

1. let $N' := \{B_a \mid a \in \Sigma\}$
2. let $P' := \{A \to \alpha' \mid A \to \alpha \in P\} \cup \{B_a \to a \mid a \in \Sigma\}$ where $\alpha' := \alpha[a \mapsto B_a \mid a \in \Sigma]$

This yields G' (example: on the board)

Chomsky Normal Form III

Proof of Theorem II.10 (continued).

Step 1: (only $A \to a$)

1. let $N' := \{B_a \mid a \in \Sigma\}$
2. let $P' := \{A \to \alpha' \mid A \to \alpha \in P\} \cup \{B_a \to a \mid a \in \Sigma\}$ where $\alpha' := \alpha[a \mapsto B_a \mid a \in \Sigma]$

This yields G' (example: on the board)

Step 2: (elimination of $A \to B$)

1. determine all derivations $A_1 \Rightarrow \ldots \Rightarrow A_n$ with rules of the form $A \to B$ without repetition of nonterminals ($\implies$ only finitely many!)
2. let
$$\begin{aligned} P'' := (P \cup \{A_1 \to \alpha \mid A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow \alpha,\\ \alpha \notin N\})\\ \setminus \{A \to B \mid A \to B \in P'\}\end{aligned}$$

This yields G'' (example: on the board)

□

Chomsky Normal Form IV

Proof of Theorem II.10 (continued).

Step 3: for every $A \to A_1 \dots A_n$ with $n > 2$:

1. add new symbols $B_1, \dots, B_{n-2}$ to N''
2. replace $A \to A_1 \dots A_n$ by

$$\begin{array}{rcl} A & \to & A_1 B_1 \\ B_1 & \to & A_2 B_2 \\ & \vdots & \\ B_{n-3} & \to & A_{n-2} B_{n-2} \\ B_{n-2} & \to & A_{n-1} A_n \end{array}$$

This yields G''' (example: on the board)

One can show: G, G', G'', G''' are equivalent □

The Word Problem Revisited

Goal: given $G = \langle N, \Sigma, P, S \rangle$ and $w \in \Sigma^*$, decide if $w \in L(G)$ or not

Approach by Cocke, Younger, Kasami (CYK algorithm):

1. assume G in Chomsky NF
2. let $w = a_1 \dots a_n$
3. if $n = 0$, then the word problem is trivial (since G in Chomsky NF)
4. otherwise let $w[i,j] := a_i \dots a_j$ for every $1 \leq i \leq j \leq n$
5. consider segments $w[i,j]$ in order of increasing length, starting with $w[i,i]$ (i.e., single letters)
6. in each case, determine $N_{i,j} := \{A \in N \mid A \Rightarrow^* w[i,j]\}$
7. test whether $S \in N_{1,n}$ (and thus, whether $S \Rightarrow^* w[1,n] = w$)

The CYK Algorithm I

Algorithm II.11 (CYK Algorithm)

Input: $G = \langle N, \Sigma, P, S \rangle$, $w = a_1 \dots a_n \in \Sigma^*$

Question: $w \in L(G)$?

Procedure:

```
for $i := 1$ to $n$ do
    $N_{i,i} := \{A \in N \mid A \to a_i \in P\}$
next $i$
for $d := 1$ to $n-1$ do    % compute $N_{i,i+d}$
    for $i := 1$ to $n-d$ do
        $j := i+d$; $N_{i,j} := \emptyset$;
        for $k := i$ to $j-1$ do
            $N_{i,j} := N_{i,j} \cup \{A \in N \mid$ there is $A \to BC \in P$
                                        with $B \in N_{i,k}, C \in N_{k+1,j}\}$
        next $k$
    next $i$
next $d$
```

Output: *"yes" if $S \in N_{1,n}$, otherwise "no"*

The CYK Algorithm II

Example II.12

- $$\begin{aligned} G:\quad S &\to SA \mid a \\ A &\to BS \\ B &\to BB \mid BS \mid b \mid c \end{aligned}$$
- $w = abaaba$
- Matrix representation of $N_{i,j}$

(on the board)

The Word Problem for Context–Free Languages

Seen:

- Word problem decidable using CYK algorithm

The Word Problem for Context–Free Languages

Seen:

- Word problem decidable using CYK algorithm

Open:

- Emptiness problem

Outline

1. Context–Free Grammars and Languages

2. Context–Free and Regular Languages

3. The Word Problem for Context–Free Languages

4. The Emptiness Problem for Context–Free Languages

5. Pushdown Automata

6. Outlook

The Emptiness Problem

- **Goal:** given $G = \langle N, \Sigma, P, S \rangle$, decide whether $L(G) = \emptyset$ or not
- For regular languages this was easy: check whether some final state is reachable from the initial state.
- Here: test whether start symbol is productive, i.e., whether it generates a terminal word

The Productivity Test

Algorithm II.13 (Productivity Test)

Input: $G = \langle N, \Sigma, P, S \rangle$

Question: $L(G) = \emptyset$*?*

Procedure:

```
let $i := 0, X_0 := \emptyset, X_1 := \Sigma$;    (* productive symbols *)
while $X_{i+1} \neq X_i$ do
   let $i := i + 1$;
   let $X_{i+1} := X_i \cup \{A \in N \mid A \to \alpha \in P, \alpha \in X_i^*\}$
od
```

Output: *"yes" if $S \notin X_i$, otherwise "no"*

The Productivity Test

Algorithm II.13 (Productivity Test)

Input: $G = \langle N, \Sigma, P, S \rangle$

Question: $L(G) = \emptyset$?

Procedure:

```
let $i := 0, X_0 := \emptyset, X_1 := \Sigma$;    (* productive symbols *)
while $X_{i+1} \neq X_i$ do
   let $i := i + 1$;
   let $X_{i+1} := X_i \cup \{A \in N \mid A \to \alpha \in P, \alpha \in X_i^*\}$
od
```

Output: "yes" if $S \notin X_i$, otherwise "no"

Example II.14

$$\begin{aligned} G: \quad S &\to AB \mid CA \\ A &\to a \\ B &\to BC \mid AB \\ C &\to aB \mid b \end{aligned}$$

(on the board)

The Emptiness Problem for Context–Free Languages

Seen:

- Emptiness problem decidable using productivity test

The Emptiness Problem for Context–Free Languages

Seen:

- Emptiness problem decidable using productivity test

Open:

- Characterizing automata model

Outline

1. Context–Free Grammars and Languages

2. Context–Free and Regular Languages

3. The Word Problem for Context–Free Languages

4. The Emptiness Problem for Context–Free Languages

5. Pushdown Automata

6. Outlook

Pushdown Automata I

- **Goal:** introduce an automata model which exactly accepts CFLs
- **Clear:** DFA not sufficient
  (missing "counting capability", e.g. for $\{a^n b^n \mid n \in \mathbb{N}\}$)
- DFA will be extended to pushdown automata by
  - adding a pushdown store which stores symbols from a pushdown alphabet and uses a specific bottom symbol
  - adding push and pop operations to transitions

Pushdown Automata II

Definition II.15

A pushdown automaton (PDA) is of the form
$\mathfrak{A} = \langle Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F \rangle$ where

- Q is a finite set of states
- Σ is the (finite) input alphabet
- Γ is the (finite) pushdown alphabet
- $\Delta \subseteq (Q \times \Gamma \times \Sigma_\varepsilon) \times (Q \times \Gamma^*)$ is a finite set of transitions
- $q_0 \in Q$ is the initial state
- Z_0 is the (pushdown) bottom symbol
- $F \subseteq Q$ is a set of final states

Interpretation of $((q, Z, x), (q', \delta)) \in \Delta$: if the PDA $\mathfrak{A}$ is in state q where Z is on top of the stack and x is the next input symbol (or empty), then $\mathfrak{A}$ reads x, replaces Z by δ, and changes into the state q'.

Configurations, Runs, Acceptance

Definition II.16

Let $\mathfrak{A} = \langle Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F \rangle$ be a PDA.

- An element of $Q \times \Gamma^* \times \Sigma^*$ is called a configuration of $\mathfrak{A}$.
- The initial configuration for input $w \in \Sigma^*$ is given by (q_0, Z_0, w).
- The set of final configurations is given by $F \times \Gamma^* \times \{\varepsilon\}$.
- If $((q, Z, x), (q', \delta)) \in \Delta$, then $(q, Z\gamma, xw) \vdash (q', \delta\gamma, w)$ for every $\gamma \in \Gamma^*$, $w \in \Sigma^*$.
- $\mathfrak{A}$ accepts $w \in \Sigma^*$ if $(q_0, Z_0, w) \vdash^* (q, \gamma, \varepsilon)$ for some $q \in F$, $\gamma \in \Gamma^*$.
- The language accepted by $\mathfrak{A}$ is $L(\mathfrak{A}) := \{w \in \Sigma^* \mid \mathfrak{A} \text{ accepts } w\}$.
- A language L is called PDA–recognizable if $L = L(\mathfrak{A})$ for some PDA $\mathfrak{A}$.
- Two PDA $\mathfrak{A}_1, \mathfrak{A}_2$ are called equivalent if $L(\mathfrak{A}_1) = L(\mathfrak{A}_2)$.

Examples

Example II.17

1. PDA which recognizes $L = \{a^n b^n \mid n \in \mathbb{N}\}$
   (on the board)

Examples

Example II.17

1. PDA which recognizes $L = \{a^n b^n \mid n \in \mathbb{N}\}$
   (on the board)
2. PDA which recognizes $L = \{ww^R \mid w \in \{a, b\}^*\}$
   (palindromes of even length; on the board)

Deterministic PDA

Observation: $\mathfrak{A}_2$ is nondeterministic: in every construction step, the pushdown could also be deconstructed

Deterministic PDA

Observation: $\mathfrak{A}_2$ is nondeterministic: in every construction step, the pushdown could also be deconstructed

Definition II.18

A PDA $\mathfrak{A} = \langle Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F \rangle$ is called deterministic (DPDA) if for every $q \in Q, Z \in \Gamma$,

- for every $x \in \Sigma_\varepsilon$, at most one (q, Z, x)–step in Δ and
- if there is a (q, Z, a)–step in Δ for some $a \in \Sigma$, then no (q, Z, ε)–step is possible.

Deterministic PDA

Observation: $\mathfrak{A}_2$ is nondeterministic: in every construction step, the pushdown could also be deconstructed

Definition II.18

A PDA $\mathfrak{A} = \langle Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F \rangle$ is called deterministic (DPDA) if for every $q \in Q, Z \in \Gamma$,

- for every $x \in \Sigma_\varepsilon$, at most one (q, Z, x)–step in Δ and
- if there is a (q, Z, a)–step in Δ for some $a \in \Sigma$, then no (q, Z, ε)–step is possible.

One can show: determinism restricts the set of acceptable languages (DPDA–recognizable languages are closed under complement, which is generally not true for PDA–recognizable languages)

Example II.19

The set of palindromes of even length is PDA–recognizable, but not DPDA–recognizable.

PDA and Context–Free Languages I

Theorem II.20

A language is context–free iff it is PDA–recognizable.

PDA and Context–Free Languages I

Theorem II.20

A language is context–free iff it is PDA–recognizable.

Proof.

$\Longleftarrow$ omitted

$\Longrightarrow$ let $G = \langle N, \Sigma, P, S \rangle$ be a CFG. Construction of PDA $\mathfrak{A}_G$ recognizing $L(G)$:

- $\mathfrak{A}_G$ simulates a derivation of G where the leftmost nonterminal of a sentence form is replaced ("leftmost derivation")
- begin with S on pushdown
- if nonterminal on top: apply corresponding production rule
- if terminal on top: match with next input symbol

□

PDA and Context–Free Languages II

Proof of Theorem II.20 (continued).

$\Longrightarrow$ Formally: $\mathfrak{A}_G := \langle Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F \rangle$ is given by

- $Q := \{q_0\}$
- $\Gamma := N \cup \Sigma$
- if $A \to \alpha \in P$, then $((q_0, A, \varepsilon), (q_0, \alpha)) \in \Delta$
- for every $a \in \Sigma$, $((q_0, a, a), (q_0, \varepsilon)) \in \Delta$
- $Z_0 := S$
- $F := Q$

□

PDA and Context–Free Languages II

Proof of Theorem II.20 (continued).

$\Longrightarrow$ Formally: $\mathfrak{A}_G := \langle Q, \Sigma, \Gamma, \Delta, q_0, Z_0, F\rangle$ is given by

- $Q := \{q_0\}$
- $\Gamma := N \cup \Sigma$
- if $A \to \alpha \in P$, then $((q_0, A, \varepsilon), (q_0, \alpha)) \in \Delta$
- for every $a \in \Sigma$, $((q_0, a, a), (q_0, \varepsilon)) \in \Delta$
- $Z_0 := S$
- $F := Q$

□

Example II.21

"Bracket language", given by G:

$$S \to \langle\rangle \mid \langle S\rangle \mid SS$$

(on the board)

Pushdown Automata

Seen:

- Definition of PDA
- Equivalence of PDA–recognizable and context–free languages

Pushdown Automata

Seen:

- Definition of PDA
- Equivalence of PDA–recognizable and context–free languages

Open:

- Description of concurrent systems

Outline

1. Context–Free Grammars and Languages
2. Context–Free and Regular Languages
3. The Word Problem for Context–Free Languages
4. The Emptiness Problem for Context–Free Languages
5. Pushdown Automata
6. Outlook

Outlook

- Equivalence problem for CFG and PDA ("$L(X_1) = L(X_2)$?") (generally undecidable, decidable for DPDA)
- Pumping Lemma for CFL
- Construction of parsers for compilers
- Non–context–free grammars and languages (context–sensitive and recursively enumerable languages, Turing machines—see Week 4)