COMP4141 Theory of Computation

Lecture 5 Context-Free Languages

Ron van der Meyden

CSE, UNSW

Revision: Date: 2014/03/17

(Credits: David Dill, Thomas Wilke, Kai Engelhardt, Peter Höfner, Rob van Glabbeek)

Context-Free Languages

Regular languages have many wonderful properties, but not all languages are regular. (E.g. $\{\ \mathtt{a}^i\mathtt{b}^i \mid i \in \mathbb{N}\ \}$, arithmetic expressions)

Next, we'll study a more powerful class of languages, the *context-free languages* (*CFL*s).

CFLs were identified in the 1950's by linguist Noam Chomsky, as a natural place in a hierarchy of languages, which included the regular languages.

Formal Definition of Context-Free Grammars

Definition

A *context-free grammar* (*CFG*) is a 4-tuple (N, Σ, P, S), where

1. N is a finite set of *variables*,
2. Σ is a finite set, disjoint from N, of *terminals*,
3. $P \subseteq N \times (N \cup \Sigma)^*$ is a finite set of *rules*, and
4. $S \in N$ is the *start variable*.

Variables are often called *non-terminal symbols*, terminals are often called *terminal symbols*, rules also go under the name *productions*, and the start variable is also known as the *sentence symbol*.

Notational Conventions for CFGs

Typically,

- upper case letters $A, B, S, \ldots$ are used for variables,
- $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{0}, \mathtt{1} \ldots$ for terminals,
- w, x, y, z for strings of terminals (Σ^*), and
- $\alpha, \beta, \gamma, \ldots$ for strings of terminals and/or variables ($(N \cup \Sigma)^*$).

Productions are written as in

$$A \to \mathtt{a}B\mathtt{c}$$

Here

- A is the left-hand side (LHS), also called the *head*, and
- $\mathtt{a}B\mathtt{c}$ is the right-hand side (RHS), also called the *body*.

Several productions with common heads can be combined:

$$A \rightarrow \mathtt{a} \mid A\mathtt{a} \mid \mathtt{b}A\mathtt{b}$$

Example

$$G = (\{S\}, \{0, 1\}, \{S \rightarrow \epsilon \mid 0S1\}, S)$$

Derivations

The language of a given CFG, $G = (N, \Sigma, P, S)$, can be characterized using the concept of a *derivation*.

Definition

Derivation step: $\alpha A \beta \Rightarrow_G \alpha\gamma\beta$ whenever $A \to \gamma \in P$.

Define $\Rightarrow_G^*$ to be the *reflexive transitive closure* of $\Rightarrow_G$. That is, $\alpha \Rightarrow_G^* \beta$ if we can get from α to β in zero or more steps.

The *language of G* is

$$L(G) = \{\ w \in \Sigma^* \mid S \Rightarrow_G^* w\ \}$$

Example

If

$$G = (\{S\}, \{0, 1\}, \{S \to \epsilon \mid 0S1\}, S)$$

then

$S \Rightarrow_G^* 0011$ because $S \Rightarrow_G 0S1 \Rightarrow_G 00S11 \Rightarrow_G 0011$.

Apparently, $L(G) = \{\ 0^i 1^i \mid i \in \mathbb{N}\ \}$

Example: Grammar for Regular Expressions

Suppose $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$$S \to \emptyset \mid \epsilon \mid \mathtt{a} \mid \mathtt{b} \mid S \cup S \mid S \circ S \mid S^* \mid (S)$$

$(\mathtt{a} \cup \mathtt{b} \circ \mathtt{a})^*$ is a regular expression because
$S \Rightarrow_G S^* \Rightarrow_G (S)^* \Rightarrow_G (S \cup S)^* \Rightarrow_G (S \cup S \circ S)^* \Rightarrow_G$
$(\mathtt{a} \cup S \circ S)^* \Rightarrow_G (\mathtt{a} \cup S \circ \mathtt{a})^* \Rightarrow_G (\mathtt{a} \cup \mathtt{b} \circ \mathtt{a})^*$.

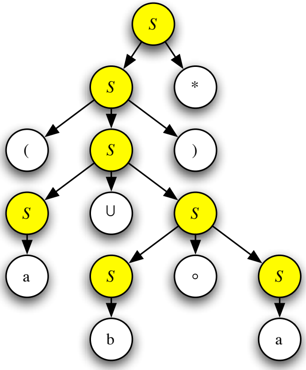
Parse Trees

A *parse tree* is a tree that shows how to derive a string from a non-terminal.

The children of a node in the tree correspond to the body of a production that has the node as head.

For $A \to \epsilon$, there is a single child, ϵ.

Parse tree for $(\mathtt{a} \cup \mathtt{b} \circ \mathtt{a})^*$:



Yield of a Parse Tree

The concatenation of the symbols at the leaves of a parse tree is called the *yield* of the parse tree.

The yield can always be derived from the symbol at the root of the tree. If the root is S and the yield is $x \in \Sigma^*$, then $x \in L(G)$.

Leftmost Derivations

There are many ways to extract a derivation from a parse tree. If we put a restriction on how the derivation is done, we can get the derivation uniquely.

Definition

A derivation of a string w in a grammar G is a *leftmost derivation* if at every step the leftmost remaining variable is the one replaced.

Example

$$\underline{S} \Rightarrow_G \underline{S}^* \Rightarrow_G (\underline{S})^* \Rightarrow_G (\underline{S} \cup S)^* \Rightarrow_G (\mathtt{a} \cup \underline{S})^* \Rightarrow_G (\mathtt{a} \cup \underline{S} \circ S)^* \Rightarrow_G (\mathtt{a} \cup \mathtt{b} \circ \underline{S})^* \Rightarrow_G (\mathtt{a} \cup \mathtt{b} \circ \mathtt{a})^*$$

where we have underlined the leftmost variable at each step

Ambiguity

A CFG is *ambiguous* if there is more than one leftmost derivation for the same string.

Equivalently: more than one parse tree for the same string.

Ambiguity often causes problems:

- With interpretation.
- With parsing.

Is our grammar for regular expressions ambiguous?
Yes, but it needn't be.

Example: Unambiguous Grammar for RE_Σ

$G_{\mathrm{RE}_\Sigma} = (\{U, C, K, T\}, \Sigma \cup \{\epsilon, \emptyset, \cup, \circ, {}^*, (,)\}, P, U)$ where the rules P are:

$$
\begin{aligned}
U &\to U \cup C \mid C \\
C &\to C \circ K \mid K \\
K &\to T^* \mid T \\
T &\to (U) \mid \emptyset \mid \epsilon \mid \mathtt{a} \mid \mathtt{b} \mid \ \ldots
\end{aligned}
$$

$(\mathtt{a} \cup \mathtt{b} \circ \mathtt{a})^*$ is a regular expression according to $G_{\mathrm{RE}_{\{\mathtt{a},\mathtt{b}\}}}$ because $U \Rightarrow_G C \Rightarrow_G K \Rightarrow_G T^* \Rightarrow_G (U)^* \Rightarrow_G (U \cup C)^* \Rightarrow_G (C \cup C)^* \Rightarrow_G (K \cup C)^* \Rightarrow_G (T \cup C)^* \Rightarrow_G (\mathtt{a} \cup C)^* \Rightarrow_G (\mathtt{a} \cup C \circ K)^* \Rightarrow_G (\mathtt{a} \cup K \circ K)^* \Rightarrow_G (\mathtt{a} \cup T \circ K)^* \Rightarrow_G (\mathtt{a} \cup \mathtt{b} \circ K)^* \Rightarrow_G (\mathtt{a} \cup \mathtt{b} \circ T)^* \Rightarrow_G (\mathtt{a} \cup \mathtt{b} \circ \mathtt{a})^*$.

Inherently Ambiguous CFLs

Some languages are context-free but don't have unambiguous CFGs.

$$\left\{ \mathtt{a}^i \mathtt{b}^j \mathtt{c}^k \mid i = j \vee j = k \right\}$$

Intuition: This is the union of two unambiguous grammars, but they have to overlap when $i = j = k$.

$$S \to AC \mid BD$$
$$A \to \mathtt{a}A\mathtt{b} \mid \epsilon$$
$$C \to C\mathtt{c} \mid \epsilon$$
$$D \to \mathtt{b}D\mathtt{c} \mid \epsilon$$
$$B \to B\mathtt{a} \mid \epsilon$$

Push-Down Automata

Pushdown Automata are to CFGs what Finite Automata are to Regular Expressions.

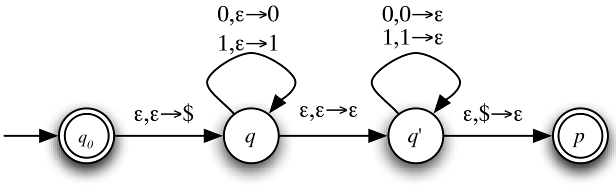
A PDA is an ϵ-NFA with an additional *stack*.

The stack makes it more powerful than an NFA because states can only "store" a fixed amount of information, while the stack is unbounded.

But the stack can only be used in a limited way, by pushing and popping symbols.

PDA

Here is an example PDA that accepts the language $\{\ ww^{\mathcal{R}}\ |\ w \in \Sigma^*\ \}$ (even-length palindromes):



In this case, the input alphabet is $\{0, 1\}$ and the stack alphabet is $\{0, 1, \$\}$.

Rule $x, y \to z$ intuitively means "read input x, replace the y at the top of the stack by z". It applies only if there *is* a y at the top of the stack!

PDA Formalities

Definition

A *pushdown automaton* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, and Γ are all finite sets, and

1. Q is the set of *states*,
2. Σ is the *input alphabet*,
3. Γ is the *stack alphabet*,
4. $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \longrightarrow 2^{Q \times \Gamma_\epsilon}$ is a *transition function*,
5. $q_0 \in Q$ is the *start state*, and
6. $F \subseteq Q$ is the set of *accept states*.

Acceptance by a PDA

Definition

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA. An *instantaneous description* (or *ID*) is a snapshot $\iota = (q, w, \alpha)$ of the PDA recording

- the current state $q \in Q$,
- the input that has not yet been read $w \in \Sigma^*$, and
- the complete contents of the stack $\alpha \in \Gamma^*$.

An ID has everything necessary to predict the possible future IDs of the PDA.

Acceptance

IDs evolve over time. $\iota_i \rightsquigarrow \iota_{i+1}$ if the PDA can transform ι_i to ι_{i+1}:

Definition

Define $\rightsquigarrow$ by

$$(q, aw, X\beta) \rightsquigarrow (p, w, \alpha\beta)$$

if $\delta(q, a, X)$ contains (p, α).

Define $\overset{*}{\rightsquigarrow}$ to be the reflexive transitive closure of $\rightsquigarrow$.

Definition

A string w is accepted if there exists a γ such that $(q_0, w, \epsilon) \overset{*}{\rightsquigarrow} (p, \epsilon, \gamma)$ and $p \in F$.

$$L(P) = \left\{ w \in \Sigma^* \;\middle|\; \exists p \in F, \gamma \in \Gamma^* \left((q_0, w, \epsilon) \overset{*}{\rightsquigarrow} (p, \epsilon, \gamma) \right) \right\}$$

PDA Acceptance Example

Running the previous PDA on input 0110 gives the following computation

$(q_0, 0110, \epsilon) \rightsquigarrow (q, 0110, \$) \rightsquigarrow (q, 110, 0\$) \rightsquigarrow (q, 10, 10\$) \rightsquigarrow (q', 10, 10\$) \rightsquigarrow (q', 0, 0\$) \rightsquigarrow (q', \epsilon, \$) \rightsquigarrow (p, \epsilon, \epsilon)$

Theorem

$L \subseteq \Sigma^*$ is context-free iff some PDA recognises L.

—THE END—