Introduction to the Theory of Computation

XU Ming

School of Software Engineering, East China Normal University

May 8, 2024

OUTLINE

- Context-Free Grammars and Context-Free Languages
- Parse Trees for Context-Free Grammars

Context-Free Grammars and Context-Free Languages

- Context-Free Grammars (CFG's) was first proposed as a description method for natural languages by Chomsky in 1956. A similar idea was used to describe computer languages: **Fortran** and **Algol** since 1960's. Sometimes CFG's are referred to as "Backus–Naur form grammars".
- Context-Free Languages (CFL's) played a central role in compilers technology. Today CFL's are increasingly important for XML and their DTD (Document-Type Definition).

Formal Definition of CFG's

A context-free grammar is a 4-tuple $G = (V, T, P, S)$, in which

- V is a finite set of *variables*,
- T is a finite set of *terminals*,
- P is a finite set of *productions* of the form $A \to \alpha$, where A is a variable and $\alpha \in (V \cup T)^*$, and
- S is a designated variable called the *start symbol*.

Example

Consider grammar $G_{pal} = (\{S\}, \{0, 1\}, P, S)$, where

$$P = \{S \to \epsilon, S \to 0, S \to 1, S \to 0S0, S \to 1S1\}.$$

G_{pal} is a context-free grammar. Later we will prove that it describes the language of palindromes L_{pal} on alphabet $\{0, 1\}$. It is easy to verify L_{pal} is not regular language. Sometimes, we group productions with the same head, e.g.

$$P = \{S \to \epsilon \,|\, 0 \,|\, 1 \,|\, 0S0 \,|\, 1S1\}.$$

Example

Regular expressions over $\{0, 1\}$ can be defined by the grammar

$$G_{regex} = (\{E\}, \{0, 1, (,), ., +, *, \epsilon, \emptyset\}, P, E)$$

where

$$P = \{E \to \epsilon, E \to 0, E \to 1, E \to \emptyset, E \to E.E, E \to E + E, E \to E^*, E \to (E)\}$$

or

$$P = \{E \to \epsilon \,|\, 0 \,|\, 1 \,|\, \emptyset \,|\, E.E \,|\, E + E \,|\, E^* \,|\, (E)\}.$$

Example

Consider a CFG that represent simple expressions in a typical programming language. Operators are $+$ and $\times$, and arguments are identifiers, i.e. strings in

$$L((\mathbf{a}+\mathbf{b})(\mathbf{a}+\mathbf{b}+\mathbf{0}+\mathbf{1})^*).$$

The expressions are defined by the grammar $G = (\{E, I\}, T, P, E)$ where $T = \{+, \times, (,), a, b, 0, 1\}$ and P is the following set of productions:

1. $E \to I$,
2. $E \to E + E$,
3. $E \to E \times E$,
4. $E \to (E)$,
5. $I \to a$,
6. $I \to b$,
7. $I \to Ia$,
8. $I \to Ib$,
9. $I \to I0$,
10. $I \to I1$.

Derivations Using Grammars

We apply the productions of a CFG to infer that certain strings are in the language of a certain variable. There are two approaches to this inference.

- Recursive inference, using productions from body to head.
- Derivations, using productions from head to body.

It is often more natural to think of grammar as used in derivations.

Example

Let us consider some of the recursive inferences we can make using the grammar for simple expressions.

	String	Language	Production	String(s) used
(i)	a	I	$I \to a$	–
(ii)	b	I	$I \to b$	–
(iii)	$b0$	I	$I \to I0$	(ii)
(iv)	$b00$	I	$I \to I0$	(iii)
(v)	a	E	$E \to I$	(i)
(vi)	$b00$	E	$E \to I$	(iv)
(vii)	$a + b00$	E	$E \to E + E$	$(v), (vi)$
$(viii)$	$(a + b00)$	E	$E \to (E)$	(vii)
(ix)	$a \times (a + b00)$	E	$E \to E \times E$	$(v), (viii)$

Now we develop the notation for describing the derivations.

Let $G = (V, T, P, S)$ be a CFG, $A \in V$, $\{\alpha, \beta\} \subset (V \cup T)^*$, and $A \to \gamma \in P$. Then we write

$$\alpha A \beta \underset{G}{\Longrightarrow} \alpha \gamma \beta$$

or, if G is understood

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

and say that $\alpha A \beta$ derives $\alpha \gamma \beta$.

Specially, we have $A \Rightarrow \gamma$.

We may extend the $\Rightarrow$ relationship to present zero, one, or many derivation steps. In other words, we define $\overset{*}{\Rightarrow}$ to be the reflexive and transitive closure of $\Rightarrow$, as follows:

Basis step: Let $\alpha \in (V \cup T)^*$. Then $\alpha \overset{*}{\Rightarrow} \alpha$.

Inductive step: If $\alpha \overset{*}{\Rightarrow} \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \overset{*}{\Rightarrow} \gamma$.

Use induction we can prove that if $\alpha \overset{*}{\Rightarrow} \beta$, and $\beta \overset{*}{\Rightarrow} \gamma$, then $\alpha \overset{*}{\Rightarrow} \gamma$.

Example

Derivation of $a \times (a + b00)$ from E in the productions:

$$E \to I \,|\, E + E \,|\, E \times E \,|\, (E), \quad I \to a \,|\, b \,|\, Ia \,|\, Ib \,|\, I0 \,|\, I1.$$

Here is one such derivation:

$$E \Rightarrow E \times E \Rightarrow I \times E \Rightarrow a \times E \Rightarrow a \times (E) \Rightarrow a \times (E + E) \Rightarrow a \times (I + E) \Rightarrow$$

$$a \times (a + E) \Rightarrow a \times (a + I) \Rightarrow a \times (a + I0) \Rightarrow a \times (a + I00) \Rightarrow a \times (a + b00)$$

Finally, $E \overset{*}{\Rightarrow} a \times (a + b00)$. The two viewpoints — recursive inference and derivation — are equivalent.

At each step we might have several rules to choose from, e.g.

$$I \times E \Rightarrow a \times E \Rightarrow a \times (E) \text{ vs } I \times E \Rightarrow I \times (E) \Rightarrow a \times (E).$$

Not all choices lead to successful derivation of a particular string, for instance

$$E \Rightarrow E + E$$

won't lead to a derivation of $a \times (a + b00)$.

Leftmost and Rightmost Derivations

In order to restrict the number of choices we have in deriving a string, it is often useful to require

- Leftmost derivation $\underset{lm}{\Longrightarrow}$: Always replace the leftmost variable by one of its rule-bodies.
- Rightmost derivation $\underset{rm}{\Longrightarrow}$: Always replace the rightmost variable by one of its rule-bodies.

Example

Derivation of $a \times (a + b00)$ from E in the productions:

$$E \to I \,|\, E + E \,|\, E \times E \,|\, (E), \quad I \to a \,|\, b \,|\, Ia \,|\, Ib \,|\, I0 \,|\, I1.$$

Here is the leftmost derivation $E \underset{lm}{\overset{*}{\Longrightarrow}} a \times (a + b00)$ as

$$E \Rightarrow E \times E \Rightarrow I \times E \Rightarrow a \times E \Rightarrow a \times (E) \Rightarrow a \times (E + E) \Rightarrow a \times (I + E) \Rightarrow$$

$$a \times (a + E) \Rightarrow a \times (a + I) \Rightarrow a \times (a + I0) \Rightarrow a \times (a + I00) \Rightarrow a \times (a + b00).$$

We can also derive $a \times (a + b00)$ from E in the productions:

$$E \to I \,|\, E + E \,|\, E \times E \,|\, (E), \quad I \to a \,|\, b \,|\, Ia \,|\, Ib \,|\, I0 \,|\, I1.$$

by the rightmost derivation

$$E \Rightarrow E \times E \Rightarrow E \times (E) \Rightarrow E \times (E + E) \Rightarrow E \times (E + I) \Rightarrow E \times (E + I0) \Rightarrow$$
$$E \times (E + I00) \Rightarrow E \times (E + b00) \Rightarrow E \times (I + b00) \Rightarrow E \times (a + b00) \Rightarrow$$
$$I \times (a + b00) \Rightarrow a \times (a + b00).$$

That is $E \overset{*}{\underset{rm}{\Longrightarrow}} a \times (a + b00)$.

The Language of a CFG

If $G(V, T, P, S)$ is a CFG, then the language of G is

$$L(G) = \{w \in T^* \mid S \overset{*}{\underset{G}{\Longrightarrow}} w\}.$$

That is, the set of strings over T^* derivable from the start symbol S. We call $L(G)$ a context-free language.

In general, we call $L = \{w \in T^* \mid A \overset{*}{\underset{G}{\Longrightarrow}} w\}$ the language of variable A if $A \in V$.

Example

Recall the grammar $G_{pal} = (\{S\}, \{0, 1\}, P, S)$, where

$$P = \{S \to \epsilon \,|\, 0 \,|\, 1 \,|\, 0S0 \,|\, 1S1\}.$$

We have known $L(G_{pal})$ is a context-free language. Now we will show

$$L(G_{pal}) = \{w \in \{0, 1\}^* \,|\, w = w^R\}.$$

Proof ($\supseteq$-direction) Suppose $w = w^R$. We show by an induction on $|w|$ that $w \in L(G_{pal})$.

Basis step: $|w| = 0$, or $|w| = 1$. Then w is ϵ, 0, or 1. Since $S \to \epsilon$, $S \to 0$ and $S \to 1$ are productions, we conclude that $S \overset{*}{\Rightarrow} w$ in all base cases.

Inductive step: Suppose $|w| \geq 2$. since $w = w^R$. we have $w = 0x0$, or $w = 1x1$, and $x = x^R$.

If $w = 0x0$ we know from the induction hypothesis that $S \overset{*}{\Rightarrow} x$. Then

$$S \Rightarrow 0S0 \overset{*}{\Rightarrow} 0x0 = w.$$

Thus $w \in L(G_{pal})$. The case for $w = 1x1$ is similar.

($\subseteq$-direction) We assume $w \in L(G_{pal})$ and must prove $w = w^R$. Since $w \in L(G_{pal})$, we have $S \overset{*}{\Rightarrow} w$. We do an induction on the length of $\overset{*}{\Rightarrow}$.

Basis step: The derivation $S \overset{*}{\Rightarrow} w$ is done in one step. Then w must be ϵ, 0, or 1, all palindromes.

Inductive step: Let $n \geq 1$, and suppose the derivation takes $n + 1$ step. Then we must have

$$w = 0x0 \overset{*}{\Leftarrow} 0S0 \Leftarrow S, \text{or } w = 1x1 \overset{*}{\Leftarrow} 1S1 \Leftarrow S$$

where the second derivation is done in n steps.

By the induction hypothesis x is a palindrome, so is w. The induction proof is complete. □

Sentential Forms

Let $G = (V, T, P, S)$ be a CFG, and $\alpha \in (V \cup T)^*$. If

$$S \overset{*}{\Rightarrow} \alpha,$$

we say that α is a sentential form.

If $S \underset{lm}{\Longrightarrow} \alpha$ we say that α is a left-sentential form, and $S \underset{rm}{\Longrightarrow} \alpha$ we say that α is a right-sentential form.

$L(G)$ is those sentential forms that are in T^*.

Example

Take G as $G = (\{E, I\}, T, P, E)$ where $T = \{+, \times, (,), a, b, 0, 1\}$ and P is

$$E \to I \,|\, E + E \,|\, E \times E \,|\, (E), \quad I \to a \,|\, b \,|\, Ia \,|\, Ib \,|\, I0 \,|\, I1.$$

Then $E \times (I + E)$ is a sentential form since

$$E \Rightarrow E \times E \Rightarrow E \times (E) \Rightarrow E \times (E + E) \Rightarrow E \times (I + E).$$

This derivation is neither leftmost, nor rightmost.

However, $a \times (E)$ is a left-sentential form, since

$$E \underset{lm}{\Longrightarrow} E \times E \underset{lm}{\Longrightarrow} I \times (E) \underset{lm}{\Longrightarrow} a \times (E).$$

Applications of CFG's

Many aspects of a programming language have a structure that may be described by regular expressions, but not all.

Example

Typical languages use parentheses in a nested and balanced fashion. A grammar $G_{bal} = (\{B\}, \{(,)\}, P, B)$ generates all and only the strings of balanced parentheses, where P consists of the productions

$$B \to BB \,|\, (B) \,|\, \epsilon.$$

Note that $L(G_{bal})$ is not a regular language.

Example

Consider **if** and **else** appear in a C programm. An if-clause can appear unbalanced by any else-clause, or it may be balanced by a matching else-clause.

A grammar generating the possible sequences of **if** and **else** (represented by i and e, respectively) is: $G_{ie} = (\{S\}, \{i, e\}, P, S)$ where P consists of the productions

$$S \to \epsilon \,|\, SS \,|\, iS \,|\, iSe.$$

For instance, $ieieie, iiie \in L(G_{ie})$.

We could test for membership in $L = L(G_{ie})$ by repeatedly doing the following, starting with a string w. The string w changes during repetition.

1. If the current string begins with e, fail; w is not in L.
2. If the string currently has no e's, succeed; w is in L.
3. Otherwise, delete the first e and the i immediately to its left. Then repeat these three steps on the resulting string.

It is easy to test $ieeii \notin L$, but $iieie \in L$.

Parse Trees for Context-Free Grammars

- If $w \in L(G)$ for some CFG G, then w has a parse tree, which tells us the (syntactic) structure of w. Note that w could be a program, a SQL-query, an XML-document, etc.
- Parse trees are an alternative representation to derivation and recursive inferences.
- There can be several parse trees for the same string. Ideally there should be one and only one parse tree for each string, i.e. the language should be unambiguous. Unfortunately, we cannot always remove the ambiguity.

Constructing Parse Trees

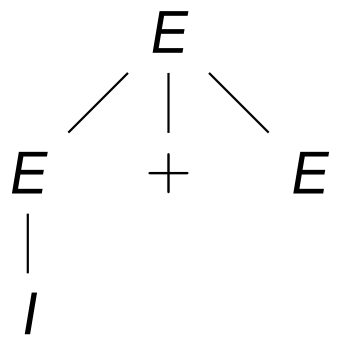
Let $G = (V, T, P, S)$ be a CFG. A tree is a parse tree for G, if:

- Each interior nodes is labeled by a variable in V.
- Each leaf is labeled by a symbol in $V \cup T \cup \{\epsilon\}$. Any ϵ-labeled leaf is the only child of its parent.
- If an interior node is labeled A, and its children (from left to right) labeled $X_1, X_2, \ldots, X_k$, then $A \to X_1 X_2 \cdots X_k \in P$.

Example

In the grammar $G = (\{E, I\}, T, P, E)$, where $T = \{+, \times, (,), a, b, 0, 1\}$ and $P = \{E \to I \mid E + E \mid E \times E \mid (E), \quad I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1\}$.

The parse tree

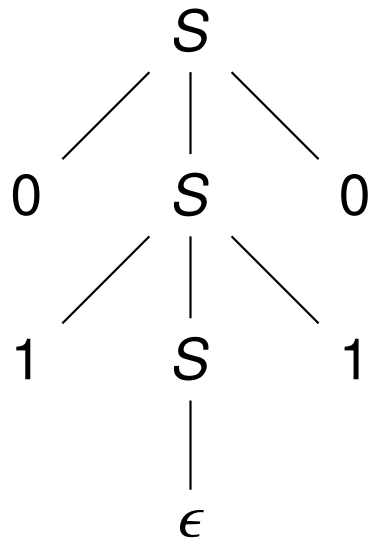


shows the derivation $E \overset{*}{\Rightarrow} I + E$.

In the grammar $G_{pal} = (\{S\}, \{0, 1\}, P, S)$, where

$$P = \{S \to \epsilon, S \to 0, S \to 1, S \to 0S0, S \to 1S1\}.$$

The parse tree



shows the derivation $S \overset{*}{\Rightarrow} 0110$.

The Yields of Parse Trees

The yield of a parse tree is the string of leaves from left to right.

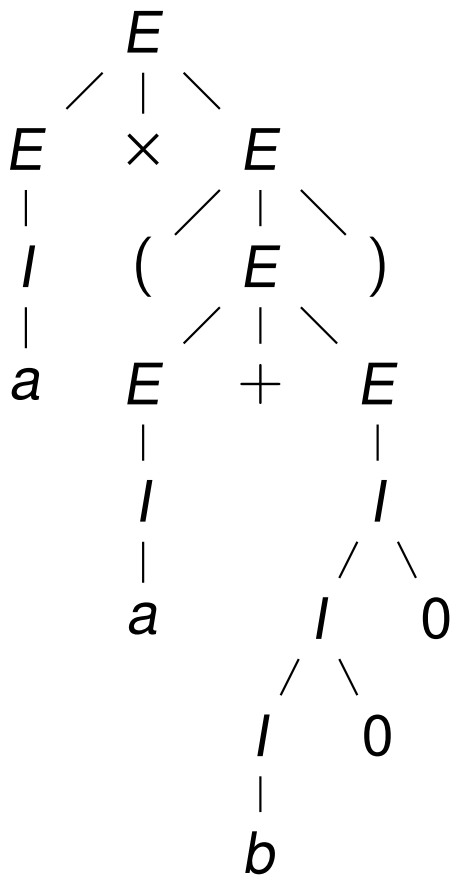
Important are those parse trees where:

- the root is labeled by the start symbol, and
- the yield is a terminal string.

We shall see the set of yields of these important parse trees is the language of the grammar.

Example

Below is an important parse tree.



The yield is $a \times (a + b00)$.

Inferences, Derivations, and Parse Trees

Let $G = (V, T, P, S)$ be a CFG, and $A \in V$. We are going to show that the following are equivalent:

- w is recursively infered to be in the language of A.
- $A \overset{*}{\Rightarrow} w$.
- $A \underset{lm}{\overset{*}{\Longrightarrow}} w$, and $A \underset{rm}{\overset{*}{\Longrightarrow}} w$.
- There is a parse tree of G with root A and yield w.

Homework

Exercises 5.1.1(b), 5.1.2(a), 5.1.5