CS 374: Algorithms & Models of Computation

Chandra Chekuri

University of Illinois, Urbana-Champaign

Spring 2017

Context Free Languages and Grammars

Lecture 7

February 7, 2017

Context Free Languages and Grammars

- Programming Language Specification
- Parsing
- Natural language understanding
- Generative model giving structure
- ...

$x = (3 * y) * z + 2 * y$

Programming Languages

```
<relational-expression> ::= <shift-expression>
                          | <relational-expression> < <shift-expression>
                          | <relational-expression> > <shift-expression>
                          | <relational-expression> <= <shift-expression>
                          | <relational-expression> >= <shift-expression>

<shift-expression> ::= <additive-expression>
                     | <shift-expression> << <additive-expression>
                     | <shift-expression> >> <additive-expression>

<additive-expression> ::= <multiplicative-expression>
                        | <additive-expression> + <multiplicative-expression>
                        | <additive-expression> - <multiplicative-expression>

<multiplicative-expression> ::= <cast-expression>
                              | <multiplicative-expression> * <cast-expression>
                              | <multiplicative-expression> / <cast-expression>
                              | <multiplicative-expression> % <cast-expression>

<cast-expression> ::= <unary-expression>
                    | ( <type-name> ) <cast-expression>

<unary-expression> ::= <postfix-expression>
                     | ++ <unary-expression>
                     | -- <unary-expression>
                     | <unary-operator> <cast-expression>
                     | sizeof <unary-expression>
                     | sizeof <type-name>

<postfix-expression> ::= <primary-expression>
                       | <postfix-expression> [ <expression> ]
                       | <postfix-expression> ( {<assignment-expression>}* )
                       | <postfix-expression> . <identifier>
                       | <postfix-expression> -> <identifier>
                       | <postfix-expression> ++
                       | <postfix-expression> --
```

Natural Language Processing

English sentences can be described as

$$\begin{aligned}
&\langle S\rangle \to \langle NP\rangle\langle VP\rangle\\
&\langle NP\rangle \to \langle CN\rangle \mid \langle CN\rangle\langle PP\rangle\\
&\langle VP\rangle \to \langle CV\rangle \mid \langle CV\rangle\langle PP\rangle\\
&\langle PP\rangle \to \langle P\rangle\langle CN\rangle\\
&\langle CN\rangle \to \langle A\rangle\langle N\rangle\\
&\langle CV\rangle \to \langle V\rangle \mid \langle V\rangle\langle NP\rangle\\
&\langle A\rangle \to \text{a} \mid \text{the}\\
&\langle N\rangle \to \text{boy} \mid \text{girl} \mid \text{flower}\\
&\langle V\rangle \to \text{touches} \mid \text{likes} \mid \text{sees}\\
&\langle P\rangle \to \text{with}
\end{aligned}$$

English Sentences

Examples

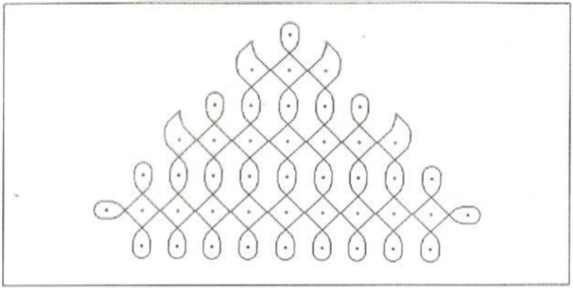
$$\overbrace{\underbrace{\text{a}}_{\text{article}}\ \underbrace{\text{boy}}_{\text{noun}}}^{\text{noun-phrs}}\ \overbrace{\underbrace{\text{sees}}_{\text{verb}}}^{\text{verb-phrs}}$$

$$\overbrace{\underbrace{\text{the}}_{\text{article}}\ \underbrace{\text{boy}}_{\text{noun}}}^{\text{noun-phrs}}\ \overbrace{\underbrace{\text{sees}}_{\text{verb}}\ \underbrace{\text{a flower}}_{\text{noun-phrs}}}^{\text{verb-phrs}}$$

Models of Growth

- *L*-systems
- http://www.kevs3d.co.uk/dev/lsystems/

Kolam drawing generated by grammar



Context Free Grammar (CFG) Definition

Definition

A CFG is is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal symbols

Context Free Grammar (CFG) Definition

Definition

A CFG is is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal symbols
- T is a finite set of terminal symbols (alphabet)

Context Free Grammar (CFG) Definition

Definition

A CFG is is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal symbols
- T is a finite set of terminal symbols (alphabet)
- P is a finite set of productions, each of the form
  $A \to \alpha$
  where $A \in V$ and α is a string in $(V \cup T)^*$.
  Formally, $P \subset V \times (V \cup T)^*$.

Context Free Grammar (CFG) Definition

Definition

A CFG is is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal symbols
- T is a finite set of terminal symbols (alphabet)
- P is a finite set of productions, each of the form
  $A \rightarrow \alpha$
  where $A \in V$ and α is a string in $(V \cup T)^*$.
  Formally, $P \subset V \times (V \cup T)^*$.
- $S \in V$ is a start symbol

Example

- $V = \{S\}$
- $T = \{a, b\}$
- $P = \{S \to \epsilon \mid a \mid b \mid aSa \mid bSb\}$
  (abbrev. for $S \to \epsilon, S \to a, S \to b, S \to aSa, S \to bSb$)

Example

- $V = \{S\}$
- $T = \{a, b\}$
- $P = \{S \to \epsilon \mid a \mid b \mid aSa \mid bSb\}$
  (abbrev. for $S \to \epsilon, S \to a, S \to b, S \to aSa, S \to bSb$)

$$S \rightsquigarrow aS\underset{a}{\cancel{A}} \rightsquigarrow abSba \rightsquigarrow abbS\underset{b}{\cancel{B}}ba \rightsquigarrow abbba$$

Example

- $V = \{S\}$
- $T = \{a, b\}$
- $P = \{S \to \epsilon \mid a \mid b \mid aSa \mid bSb\}$
  (abbrev. for $S \to \epsilon, S \to a, S \to b, S \to aSa, S \to bSb$)

$$S \rightsquigarrow aSA \rightsquigarrow abSba \rightsquigarrow abbSBba \rightsquigarrow abbba$$

What strings can S generate like this?

Palindromes

- Madam in Eden I'm Adam
- Dog doo? Good God!
- Dogma: I am God.
- A man, a plan, a canal, Panama
- Are we not drawn onward, we few, drawn onward to new era?
- Doc, note: I dissent. A fast never prevents a fatness. I diet on cod.
- http://www.palindromelist.net

Examples

$L = \{0^n 1^n \mid n \geq 0\}$

$S \rightarrow 0S1 \mid \varepsilon$

Examples

$L = \{0^n 1^n \mid n \geq 0\}$

$S \to \epsilon \mid 0S1$

Notation and Convention

Let $G = (V, T, P, S)$ then

- $a, b, c, d, \ldots,$ in T (terminals)
- $A, B, C, D, \ldots,$ in V (non-terminals)
- $u, v, w, x, y, \ldots$ in T^* for strings of terminals
- $\alpha, \beta, \gamma, \ldots$ in $(V \cup T)^*$
- $X, Y, \cancel{X}$ in $V \cup T$

  Z

"Derives" relation

Formalism for how strings are derived/generated

Definition

Let $G = (V, T, P, S)$ be a CFG. For strings $\alpha_1, \alpha_2 \in (V \cup T)^*$ we say α_1 derives α_2 denoted by $\alpha_1 \rightsquigarrow_G \alpha_2$ if there exist strings β, γ, δ in $(V \cup T)^*$ such that

- $\alpha_1 = \beta A \delta$
- $\alpha_2 = \beta \gamma \delta$
- $A \rightarrow \gamma$ is in P.

Examples: $S \rightsquigarrow \epsilon$, $S \rightsquigarrow 0S1$, $0S1 \rightsquigarrow 00S11$, $0S1 \rightsquigarrow 01$.

"Derives" relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \leadsto^k \alpha_2$ inductive defined:

- $\alpha_1 \leadsto^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \leadsto^k \alpha_2$ if $\alpha_1 \leadsto \beta_1$ and $\beta_1 \leadsto^{k-1} \alpha_2$.

"Derives" relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \rightsquigarrow^k \alpha_2$ inductive defined:

- $\alpha_1 \rightsquigarrow^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow \beta_1$ and $\beta_1 \rightsquigarrow^{k-1} \alpha_2$.
- Alternative defn: $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow^{k-1} \beta_1$ and $\beta_1 \rightsquigarrow \alpha_2$

"Derives" relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \rightsquigarrow^k \alpha_2$ inductive defined:

- $\alpha_1 \rightsquigarrow^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow \beta_1$ and $\beta_1 \rightsquigarrow^{k-1} \alpha_2$.
- Alternative defn: $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow^{k-1} \beta_1$ and $\beta_1 \rightsquigarrow \alpha_2$

$\rightsquigarrow^*$ is the reflexive and transitive closure of $\rightsquigarrow$.

$\alpha_1 \rightsquigarrow^* \alpha_2$ if $\alpha_1 \rightsquigarrow^k \alpha_2$ for some k.

Examples: $S \rightsquigarrow^* \epsilon$, $0S1 \rightsquigarrow^* 0000011111$.

Context Free Languages

Definition

The language generated by CFG $G = (V, T, P, S)$ is denoted by $L(G)$ where $L(G) = \{w \in T^* \mid S \rightsquigarrow^* w\}$.

$\alpha A \beta \rightarrow \quad \gamma$

Context Free Languages

Definition

The language generated by CFG $G = (V, T, P, S)$ is denoted by $L(G)$ where $L(G) = \{w \in T^* \mid S \rightsquigarrow^* w\}$.

Definition

A language L is context free (CFL) if it is generated by a context free grammar. That is, there is a CFG G such that $L = L(G)$.

Example

$L = \{0^n 1^n \mid n \geq 0\}$

$S \to \epsilon \mid 0S1$

$(\,) (\,) ((\,))$

$S \to \varepsilon \mid (S) \mid SS$

$L = \{0^n 1^m \mid m > n\}$

$L = \{w \in \{(,)\}^* \mid w$ is $\underbrace{\text{properly nested}}_{}$ string of parenthesis$\}$

valid

$A \to 0A1 \mid \varepsilon$

$B \to B1 \mid 1$

$S \to AB$

011

$S \rightsquigarrow AB \rightsquigarrow 0A1B \rightsquigarrow 01B \rightsquigarrow 011$

Closure Properties of CFLs

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$
Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Closure Properties of CFLs

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$
Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Theorem

CFLs are closed under union. L_1, L_2 *CFLs implies* $L_1 \cup L_2$ *is a CFL.*

$S \rightarrow S_1 \mid S_2$

P_1, P_2

Closure Properties of CFLs

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$
Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Theorem

CFLs are closed under union. L_1, L_2 *CFLs implies* $L_1 \cup L_2$ *is a CFL.*

Theorem

CFLs are closed under concatenation. L_1, L_2 *CFLs implies* $L_1 \bullet L_2$ *is a CFL.*

$S \rightarrow S_1 S_2$ $\quad P_1, P_2$

Closure Properties of CFLs

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$

Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Theorem

CFLs are closed under union. L_1, L_2 *CFLs implies* $L_1 \cup L_2$ *is a CFL.*

Theorem

CFLs are closed under concatenation. L_1, L_2 *CFLs implies* $L_1 \bullet L_2$ *is a CFL.*

Theorem

CFLs are closed under Kleene star. L *CFL implies* L^* *is a CFL.*

$S \rightarrow \varepsilon \mid S_1 S$ $\quad P_1$

Closure Properties of CFLs

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$
Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Theorem

CFLs are closed under union. L_1, L_2 *CFLs implies* $L_1 \cup L_2$ *is a CFL.*

Theorem

CFLs are closed under concatenation. L_1, L_2 *CFLs implies* $L_1 \bullet L_2$ *is a CFL.*

Theorem

CFLs are closed under Kleene star. L *CFL implies* L^* *is a CFL.*

Exercise

- Prove that every regular language is context-free using previous closure properties.
- Prove the set of regular expressions over an alphabet $\boldsymbol{\Sigma}$ forms a non-regular language which is context-free.

Closure Properties of CFLs continued

Theorem

CFLs are not closed under complement or intersection.

Theorem

If L_1 is a CFL and L_2 is regular then $L_1 \cap L_2$ is a CFL.

Canonical non-CFL

Theorem

$L = \{a^n b^n c^n \mid n \geq 0\}$ *is not context-free.*

Proof based on pumping lemma for CFLs. Technical and outside the scope of this class.

Parse Trees or Derivation Trees

A tree to represent the derivation $S \rightsquigarrow^* w$.

- Rooted tree with root labeled S
- Non-terminals at each internal node of tree
- Terminals at leaves
- Children of internal node indicate how non-terminal was expanded using a production rule

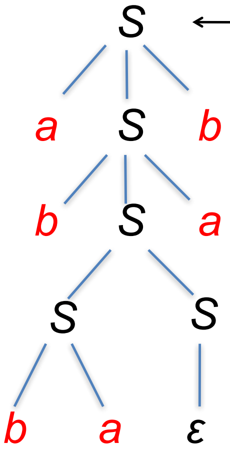
Parse Trees or Derivation Trees

A tree to represent the derivation $S \rightsquigarrow^* w$.

- Rooted tree with root labeled S
- Non-terminals at each internal node of tree
- Terminals at leaves
- Children of internal node indicate how non-terminal was expanded using a production rule

A picture is worth a thousand words

Example



A derivation tree for *abbaab*

(also called "parse tree")

$S \rightarrow aSb \mid bSa \mid SS \mid ab \mid ba \mid \varepsilon$

A corresponding derivation of *abbaab*

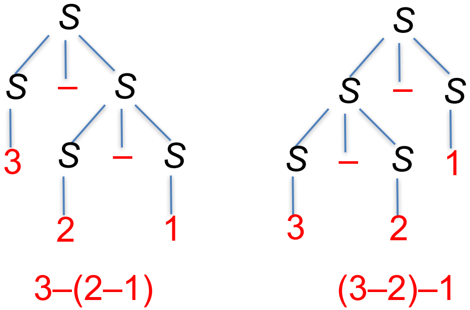
$S \rightarrow aSb \rightarrow abSab \rightarrow abSSab \rightarrow abbaSab \rightarrow abbaab$

Ambiguity in CFLs

Definition

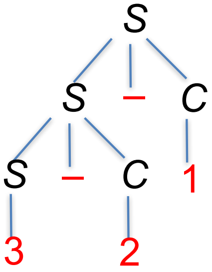
A CFG G is ambiguous if there is a string $w \in L(G)$ with two different parse trees. If there is no such string then G is unambiguous.

Example: $S \rightarrow S - S \mid 1 \mid 2 \mid 3$



Ambiguity in CFLs

- Original grammar: $S \to S - S \mid 1 \mid 2 \mid 3$
- Unambiguous grammar:
  $S \to S - C \mid 1 \mid 2 \mid 3$
  $C \to 1 \mid 2 \mid 3$



(3–2)–1

The grammar forces a parse corresponding to left-to-right evaluation.

Inherently ambiguous languages

Definition

A CFL L is inherently ambiguous if there is no unambiguous CFG G such that $L = L(G)$.

Inherently ambiguous languages

Definition

A CFL L is inherently ambiguous if there is no unambiguous CFG G such that $L = L(G)$.

- There exist inherently ambiguous CFLs.
  Example: $L = \{a^n b^m c^k \mid n = m \text{ or } m = k\}$

Inherently ambiguous languages

Definition

A CFL L is inherently ambiguous if there is no unambiguous CFG G such that $L = L(G)$.

- There exist inherently ambiguous CFLs.
  Example: $L = \{a^n b^m c^k \mid n = m \text{ or } m = k\}$
- Given a grammar G it is undecidable to check whether $L(G)$ is inherently ambiguous. No algorithm!

Inductive proofs for CFGs

Question: How do we formally prove that a CFG $L(G) = L$?

Example: $S \rightarrow \epsilon \mid a \mid b \mid aSa \mid bSb$

Theorem

$L(G) = \{palindromes\} = \{w \mid w = w^R\}$

Inductive proofs for CFGs

Question: How do we formally prove that a CFG $L(G) = L$?

Example: $S \to \epsilon \mid a \mid b \mid aSa \mid bSb$

Theorem

$L(G) = \{palindromes\} = \{w \mid w = w^R\}$

Two directions:

- $L(G) \subseteq L$, that is, $S \rightsquigarrow^* w$ then $w = w^R$
- $L \subseteq L(G)$, that is, $w = w^R$ then $S \rightsquigarrow^* w$

$L(G) \subseteq L$

Show that if $S \rightsquigarrow^* w$ then $w = w^R$

By induction on length of derivation, meaning
For all $k \geq 1$, $S \rightsquigarrow^{*k} w$ implies $w = w^R$.

$L(G) \subseteq L$

Show that if $S \rightsquigarrow^* w$ then $w = w^R$

By induction on length of derivation, meaning
For all $k \geq 1$, $S \rightsquigarrow^{*k} w$ implies $w = w^R$.

- If $S \rightsquigarrow^1 w$ then $w = \epsilon$ or $w = a$ or $w = b$. Each case $w = w^R$.
- Assume that for all $k < n$, that if $S \rightsquigarrow^k w$ then $w = w^R$
- Let $S \rightsquigarrow^n w$ (with $n > 1$). Wlog w begin with a.
    - Then $S \rightarrow aSa \rightsquigarrow^{k-1} aua$ where $w = aua$.
    - And $S \rightsquigarrow^{n-1} u$ and hence IH, $u = u^R$.
    - Therefore $w^r = (aua)^R = (ua)^R a = au^R a = aua = w$.

$L \subseteq L(G)$

Show that if $w = w^R$ then $S \rightsquigarrow^* w$.

By induction on $|w|$
That is, for all $k \geq 0$, $|w| = k$ and $w = w^R$ implies $S \rightsquigarrow^* w$.

Exercise: Fill in proof.

Mutual Induction

Situation is more complicated with grammars that have multiple non-terminals.

See Section 5.3.2 of the notes for an example proof.

Normal Forms

Normal forms are a way to restrict form of production rules

Advantage: Simpler/more convenient algorithms and proofs

Normal Forms

Normal forms are a way to restrict form of production rules

Advantage: Simpler/more convenient algorithms and proofs

Two standard normal forms for CFGs

- Chomsky normal form
- Greibach normal form

Normal Forms

Chomsky Normal Form:

- Productions are all of the form $A \rightarrow BC$ or $A \rightarrow a$.
  If $\epsilon \in L$ then $S \rightarrow \epsilon$ is also allowed.
- Every CFG G can be converted into CNF form via an efficient algorithm
- Advantage: parse tree of constant degree.

Normal Forms

Chomsky Normal Form:

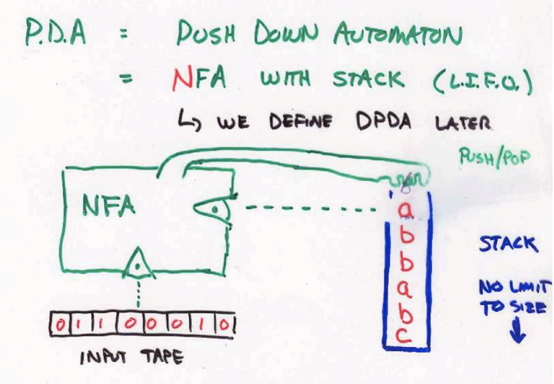
- Productions are all of the form $A \to BC$ or $A \to a$. If $\epsilon \in L$ then $S \to \epsilon$ is also allowed.
- Every CFG G can be converted into CNF form via an efficient algorithm
- Advantage: parse tree of constant degree.

Greiback Normal Form:

- Only productions of the form $A \to a\beta$ are allowed.
- All CFLs without ϵ have a grammar in GNF. Efficient algorithm.
- Advantage: Every derivation adds exactly one terminal.

Things to know: Pushdown Automata

PDA: a NFA coupled with a stack



PDAs and CFGs are equivalent: both generate exactly CFLs.
PDA is a machine-centric view of CFLs.