Grammars: Raw lecture notes

Terence Parr
University of San Francisco
parrt@cs.usfca.edu

Abstract

1. CFGs

A *grammar* is a set of rules that describe the set of valid sentences in a language, L. The rules have a very specific format and therefore follow a language, a metalanguage. The rules are called *production rules* and say how to generate strings in the language. There are rule names, *non-terminals*, and vocabulary symbols (*terminals* or *tokens*). The rules are of the form

$leftside \rightarrow rightside$

Context free grammar: *CFG* is a grammar where $leftside$ has to be a single non-terminal:

$expression \rightarrow \mathbf{id}$

there can be multiple rules

$expression \rightarrow \mathbf{integer}$

For formal grammars we tend to use single single capital letters for non-terminals CFG notation:

$E \rightarrow \mathbf{id}$
$E \rightarrow \mathbf{integer}$

examples

$A \rightarrow \mathbf{a}$
$A \rightarrow \mathbf{b}$

Or, $A \rightarrow \mathbf{a}|\mathbf{b}$

Language $L = \{a, b\}$.

Infinite languages

$A \rightarrow aA$
$A \rightarrow \epsilon$

or

$A \rightarrow aA$
$A \rightarrow$

$L = a^*$

$A \rightarrow aA$
$A \rightarrow a$

$L = a^+$

Many grammars for one L.

A CFG grammar $G = (N, T, P, S)$ has elements:

- N is the set of nonterminals (rule names)
- T is the set of terminals (tokens)
- P is the set of productions
- $S \in N$ is the start symbol

$A \in N$	Nonterminal
$a, b, c, d \in T$	Terminal
$X \in (N \cup T)$	Production element
$\alpha, \beta, \delta \in X^*$	Sequence of grammar symbols
$u, v, w, x, y \in T^*$	Sequence of terminals
ϵ	Empty string
$	End of file “symbol”

$L = \{a^n b^n | n \geq 1\}$ is CF but $L = \{a^n b^n c^n | n \geq 1\}$ non CF.

$A \rightarrow aAb$
$A \rightarrow ab$

The set of all context-free languages is identical to the set of languages accepted by pushdown automata (*PDA*) and all contexts we languages can be parsed in $O(n^3)$ time.

2. Regular grammars

. A single non-terminal on the left like CFG, but the right-hand side can only be: empty, a sequence of terminals, a sequence of terminals followed by a non-terminal, but that's it. All regular languages can be recognized by a finite state machine linear time. NFA/DFA.

Non-regular $\{a^n b^n | n \geq 1\}$ because we have no memory but $\{a^n b^m | n \geq 1\}$ is regular.
$A \to a^* b^*$

3. EBNF

which introduces us to extended BNF (EBNF). we typically use a variation on yacc notation, which flips things so that non-terminals are lowercase and terminals are uppercase like constants:

```
a : A* B* ; // extended; yacc doesn't allow *
```

or

```
a : 'a'* 'b'* ; // ANTLR notation
```

```
grammar Ex; // generates class ExParser
// action defines ExParser member: enum_is_keyword
@members {boolean enum_is_keyword = true;}
stat: expr '=' expr ';' // production 1
    | expr ';'          // production 2
    ;
expr: expr '*' expr
    | expr '+' expr
    | expr '(' expr ')' // f(x)
    | id
    ;
id  : ID | {!enum_is_keyword}? 'enum' ;
ID  : [A-Za-z]+ ; // match id with upper, lowercase
WS  : [ \t\r\n]+ -> skip ; // ignore whitespace
```

4. Derivations

$\alpha \Rightarrow \beta$ $\alpha \Rightarrow^* \beta$ $\alpha \Rightarrow^+ \beta$

how to regenerate a string starting from the start symbol?

$A \to aAb$
$A \to ab$

generation of ab:
$A \Rightarrow ab$
generation of $aabb$:
$A \Rightarrow aAb \Rightarrow aaba$
leftmost and rightmost derivations

$S \to \mathbf{if}\, E\, \mathbf{then}\, S$
$S \to \mathbf{return}\, E$
$E \to \mathbf{id}$

$S \Rightarrow_{lm} \mathbf{if}\, E\, \mathbf{then}\, S$
$\Rightarrow_{lm} \mathbf{ifxthen}\, S$
$\Rightarrow_{lm} \mathbf{ifxthenreturn}\, E$
$\Rightarrow_{lm} \mathbf{ifxthenreturny}$

or

$S \Rightarrow_{rm} \mathbf{if}\, E\, \mathbf{then}\, S$
$\Rightarrow_{rm} \mathbf{if}\, E\, \mathbf{thenreturn}\, E$
$\Rightarrow_{rm} \mathbf{if}\, E\, \mathbf{thenreturny}$
$\Rightarrow_{rm} \mathbf{ifxthenreturny}$

diff deriv but same tree. So parse tree of A then if-then-else.

Formally, the language generated by grammar sequence α in user state $\mathbb{S}$ is $L(\alpha) = \{w \,|\, (\alpha) \Rightarrow^* (w)\}$ and the language of grammar G is $L(G) = \{w \,|\, (S) \Rightarrow^* (w)\}$. Language L is CF *iff* there exists a CFG for L.

α is *sentential form* if S derives to it. If $\alpha \in T \cup N$ it's a *sentence*.

Proof that G gens L means show every string gen'd by G is in L and every string in L can be gen'd by G. and

5. Parse trees

Show parse tree of $A \to aA|\epsilon$ and if-then-else above.

6. Ambiguity

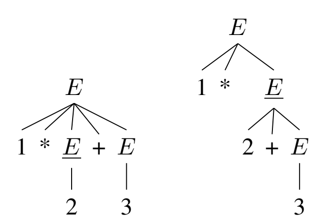
See section 5.4 in ANTLR 4 book. p69 printed.

more than one lm or rm deriv for same input. normally an error. L can be ambig too syntactically; e.g., expressions. Ambig L gives ambig G. Need disambiguating rules from semantics or otherwise.

$A \to aA|\epsilon$
classic:
$E \to E * E$
$E \to E + E$
$E \to \mathbf{id}$

1+2*3 has two interps. show trees.

Parse trees for `a*b+c` and $E \to E * E \,|\, E{+}E \,|\, \mathbf{id}$. Converted to non-Left-recur: $E \to \mathbf{id}\ (* \ E | + \ E)^*$. The parser must recognize `a*b+c` as `(a*b)+c` not `a*(b+c)`.



(a) `(a*b)+c` **(b)** `a*(b+c)`

A predicated grammar $G = (N, T, P, S, \Pi, \mathcal{M})$ has elements:

- N is the set of nonterminals (rule names)
- T is the set of terminals (tokens)
- P is the set of productions
- $S \in N$ is the start symbol
- Π is a set of side-effect-free semantic predicates
- $\mathcal{M}$ is a set of actions (mutators)

7. Left recur

Indirectly left-recursive rules call themselves through another rule; e.g., $A \rightarrow B$, $B \rightarrow A$. Hidden left-recursion occurs when an empty production exposes left recursion; e.g., $A \rightarrow BA$, $B \rightarrow \epsilon$.
$A \rightarrow Aa$
$A \rightarrow b$

right recur
$A \rightarrow aA$
$A \rightarrow b$

Ex: elim in direct left recur
$E \rightarrow E * E$
$E \rightarrow E/E$
$E \rightarrow E + E$
$E \rightarrow \mathbf{id}$

Show typical non-left recur arith expr rules.

8. Left factoring

$S \rightarrow \mathbf{if} E \mathbf{then} S \mathbf{else} S$
$S \rightarrow \mathbf{if} E \mathbf{then} S$

$S \rightarrow \mathbf{if} E \mathbf{then} S S'$
$S' \rightarrow \mathbf{else} S$
$S' \rightarrow$
or EBNF
$S \rightarrow \mathbf{if} E \mathbf{then} S (\mathbf{else} S)^?$