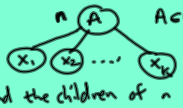
Parse trees

Given some fixed grammar G, a parse tree of G is a rooted, ordered tree whose nodes are labeled with grammar symbols, such that, for every internal node n, n's label is a nonterminal:



and the children of n (A), read from left to right, form the body of some production whose head is A [$A \to X_1 \cdots X_k$ is a prod. of G].

One exception: if $A \to \varepsilon$ is a production, then using this would give

All leaves are labeled with terminals or ε.

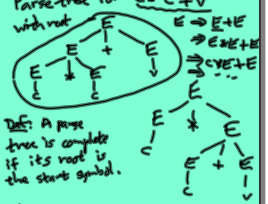
The yield of a parse tree is the string of terminals obtained by concatenating all the leaves from left to right.

[occurrences of ε "disappear"]

$E \to c \mid v \mid E+E \mid E*E \mid (E)$

[arith exprs over 2 ops: +, *]

Parse tree for $c*c+v$ with root



$E \Rightarrow E+E$
$\Rightarrow E*E+E$
$\Rightarrow c*E+E$
$\Rightarrow \cdots$

Def: A parse tree is complete if its root is the start symbol.

Thm: For any string of terminals w, w is derivable (from start symbol) in G if and only if there is a complete parse tree yielding w.
Further, complete parse trees and complete leftmost derivations are in one-to-one correspondence.

So

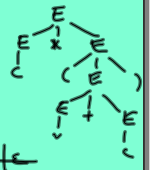
Corollary: A grammar is ambiguous if there is some string that is the yield of two distinct complete parse trees.
See above

Job of a parser: to "construct" a parse tree for the input, if there is one.

[complete parse trees $\longleftrightarrow$ rightmost derivations]

$L(G)$:= the set of strings of terminals that are yielded by some complete parse tree

$c*(v+c)$

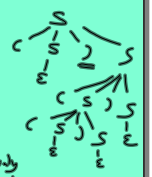


~~$S \to (S)S \mid \varepsilon$~~

Feb 7-11:02 AM

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

$()(())$



only one parse tree

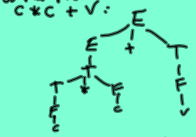
This grammar is unambiguous.

exps over +, * only

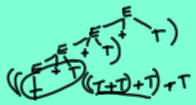
$E \to E+T \mid T$

$T \to T*F \mid F$

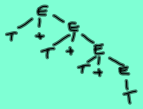
$F \to c \mid v \mid (E)$



Note: $E \to E+T \mid T$



$E \to T+E \mid T$



Best when parse tree structure closely matches intended grouping within an expr.
[translation is easiest in this case]

Alternate notation conventions for specifying grammars (ASCII-friendly)

Backus-Naur Form (BNF)

```
<expr> ::= <expr> '+' <term>
         | <term>
<term> ::= <term> '*' <factor>
         | <factor>
<factor> ::= CONST | VAR
         | '(' <expr> ')'
```

yacc/bison form:

```
expr :
     expr '+' term
   | term
   ;
term :
     term '*' factor
   | factor
   ;
factor :
     CONST | VAR | '(' expr ')'
   ;
```

named tokens declared in the preamble.

Extended BNF (EBNF)

Productions are of the form

$A \to R$

where R is a regular expression.

A can be substituted with any string matching R (in a derivation or parse tree)

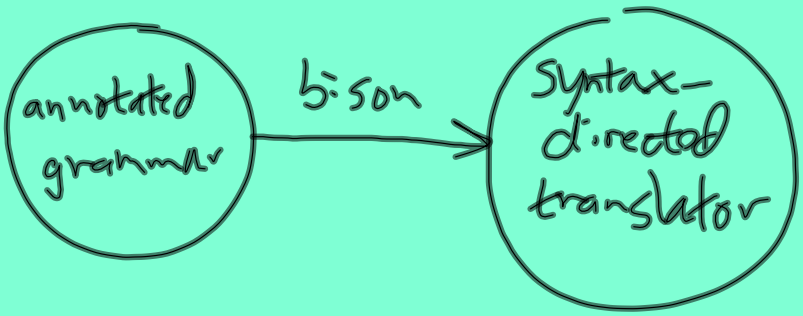
$\langle expr \rangle ::= (\langle term \rangle \text{ '+'})^* \langle term \rangle$

$\langle term \rangle ::= (\langle factor \rangle \text{ '*'})^* \langle factor \rangle$

$\langle factor \rangle ::= CONST \mid VAR \mid \text{'('} \langle expr \rangle \text{')'}$

Fact: EBNF is no more expressive than normal CFGs
(can convert any EBNF grammar into an equivalent CFG)
EBNF can be more succinct.

Feb 7-11:32 AM



Syntax-directed translation:
produce output translation as input is parsed

2 types of annotating a grammar:
- syntax-directed definition (SDD)
- syntax-directed translation scheme. (STTS)

c*c+c

3*17+9