Formal Languages - 2

- Context-free PLs
- Grammars
  - Derivation
  - Ambiguity
  - Precedence and Associativity
- Parse trees

Grammar

- <set of terminals, set of nonterminals, productions (rules), special symbol>
  - terminals are alphabet symbols
  - nonterminals represent PL constructs (e.g., Stmt)
  - productions are rules for forming syntactically correct constructs
  - special symbol tells where to start applying the rules

Example

<letter>::=
a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<digit>::= 0|1|2|3|4|5|6|7|8|9

<identifier> ::= <letter> | <identifier> <letter> | <identifier> <digit>

<assign-stmt> ::= <identifier> = 0 //terminals;

//nonterminals are {<letter><digit><assign_stmt><identifier>}

//special symbol is <assign-stmt>

Derivation

1 <letter>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

2 <digit>::= 0|1|2|3|4|5|6|7|8|9

3 <identifier> ::= <letter> | <identifier> <letter> | <identifier> <digit>

4 <assign-stmt> ::= <identifier> = 0

Can we generate x2 = 0 from these rules?

<assign-stmt> →4 <identifier> = 0
→3c <identifier> <digit> = 0
→3a <letter> <digit> = 0
→1 x <digit> = 0
→2 x 2 = 0

YES!

This is a *derivation* of a *sentence* in the language described by the grammar above.

Each sequence in this derivation is a *sentential form.* This is a *leftmost* or c*anonical derivation.* At each step, the rule indicated is used to substitute the rhs of the rule for the leftmost nonterminal in the sentential form.

Parse

1 <letter>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
2 <digit>::= 0|1|2|3|4|5|6|7|8|9
3 <identifier> ::= <letter> | <identifier> <letter> | <identifier> <digit>
4 <assign-stmt> ::= <identifier> = 0

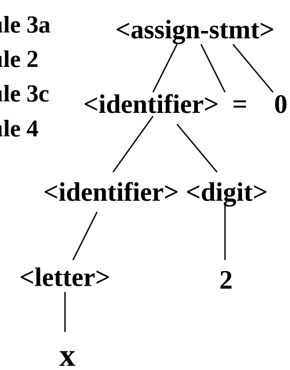
Can we recognize x2 = 0 as belonging to this PL?

x2 = 0	→	<letter> 2 = 0	rule 1
	→	<identifier> 2 = 0	rule 3a
	→	<identifier><digit> = 0	rule 2
	→	<identifier> = 0	rule 3c
	→	<assign-stmt>	rule 4

A *parse* of the sentence x2 = 0.

Parse Tree

x2 = 0	→	<letter> 2 = 0	rule 1
	→	<identifier> 2 = 0	rule 3a
	→	<identifier><digit> = 0	rule 2
	→	<identifier> = 0	rule 3c
	→	<assign-stmt>	rule 4



In parse tree, each internal node is a nonterminal; its children are the rhs of a rule for that nonterminal.

Grammars are not Unique

1 <letter>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

2 <digit>::= 0|1|2|3|4|5|6|7|8|9

3' <id> ::= <letter> | <id> <letterordigit>

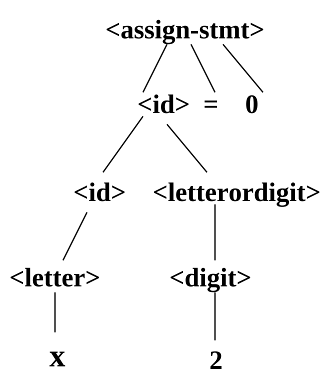
4' <assign-stmt> ::= <id> = 0

5' <letterordigit> ::= <letter> | <digit>

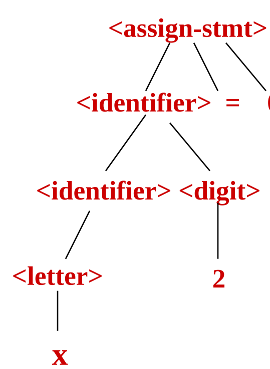
This grammar generates the same language (i.e, set of trees whose frontiers are the same) , but has different parse trees than the previous grammar.

Example

2nd grammar tree



1st grammar tree



Many grammars can correspond to 1 PL, but only 1 PL should correspond to any useful grammar!

Terms

- **Grammar**
  - a formalism that describes which sequences of terminals are meaningful in a PL
  - <finite set of terminals, nonterminals, production rules, special symbol>
- **Context-free grammar**
  - correspond to PLs whose rules have only 1 nonterminal on the lhs

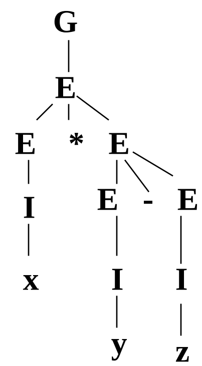
Terms

- **Sentence**
  - a finite sequence of terminals, constructed according to the rules of the grammar for that PL
- **Sentential form**
  - a finite sequence of terminals and nonterminals, constructed according to the rules of the grammar for that PL
- **Derivation**
- **Parse**

Ambiguity

1 G ::= E
E ::= E [2]- E | E [3]* E | [4]I
5 I ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

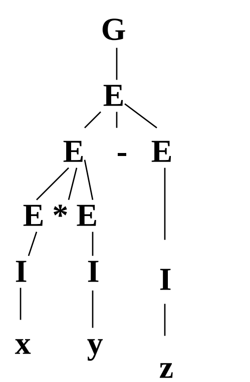
G →1 E
→3 E * E
→4 I * E
→5 x * E
→2 x * E - E
→4 x * I - E
→5 x * y - E
→4 x * y - I
→5 x * y - z



Ambiguity

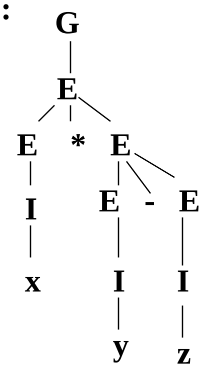
1 G ::= E
E ::= E [2]- E | E [3]* E | [4]I
5 I ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

G →1 E
→2 E - E
→3 E * E - E
→4 I * E - E
→5 x * E - E
→4 x * I - E
→5 x * y - E
→4 x * y - I
→5 x * y - z

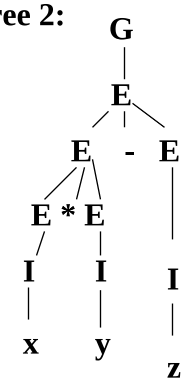


Comparison

Tree 1:



Tree 2:



Which tree is correct?
Can we rewrite the grammar to only generate one of them?

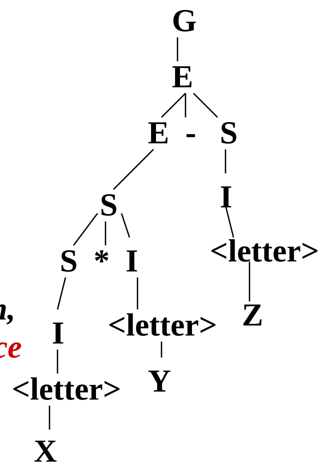
Ambiguity

- **Ambiguity**
  - **If there are 2 different canonical derivations (or alternatively, 2 parse trees) for the same sentence then the grammar is *ambiguous***
  - **There is no algorithm which can tell if an arbitrary context-free grammar is ambiguous**
  - **Solution**
    - **Change grammar to reflect operator precedences**

    X*Y-Z means ((X*Y) - Z)

A Better Grammar

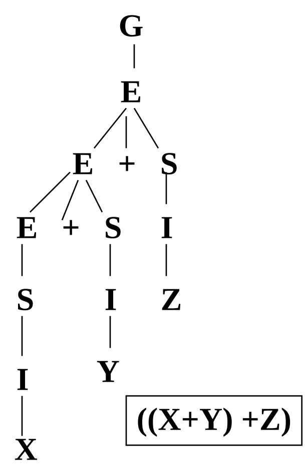
G::= E
E::= S | E - S
S::= I | S * I
I ::= <letter>
<letter>::=a|b|c|d|e|f|g|h|i|j|k|l|m|n
|o|p|q|r|s|t|u|v|w|x|y|z



Note: since S is operand of - operation, this forces * to have higher precedence than -.

Associativity in the Grammar

G::= E
E::= E + S | S
S::= I | S * I
I ::= <letter>
<letter>::=a|b|c|d|e|f|g|h|i|j|k|l|m|n
|o|p|q|r|s|t|u|v|w|x|y|z



How parse X+Y+Z?
Tree shows that + is left associative because E's rule is left recursive.

Right Associativity

G::= E
E::= S ^ E | S
S::= 0|1|2|3|4|5|6|7|8|9

What is 2^3^4? 8^4 or 2^{81}?