# Formal Languages - 3

- Ambiguity in PLs
  - Problems with if-then-else constructs
  - Harder problems
- Chomsky hierarchy for formal languages
  - Regular and context-free languages
  - Type 1: Context-sensitive languages
  - Type 0 languages and Turing machines



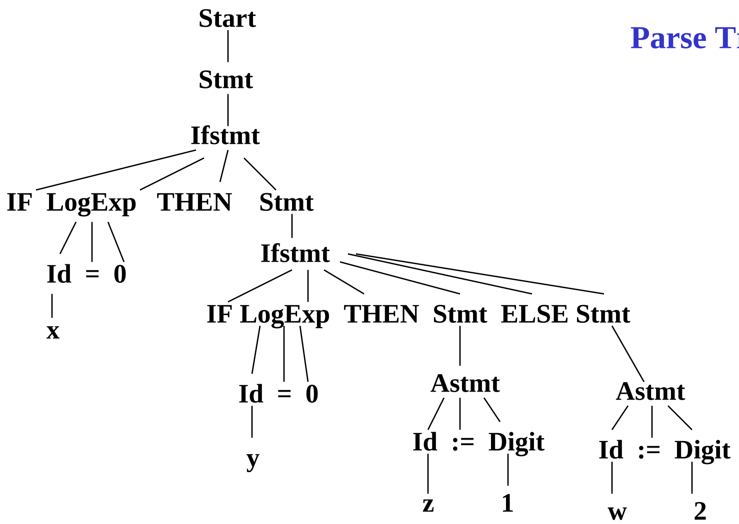
# Dangling Else Ambiguity (Pascal)

| | |
|---|---|
| Start ::= | Stmt |
| Stmt ::= | Ifstmt \| Astmt |
| Ifstmt ::= | IF LogExp THEN Stmt \| IF LogExp THEN Stmt ELSE Stmt |
| Astmt ::= | Id := Digit |
| Digit ::= | 0\|1\|2\|3\|4\|5\|6\|7\|8\|9 |
| LogExp::= | Id = 0 |
| Id ::= | 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 are compound if statements parsed using this grammar??

**IF x = 0 THEN IF y = 0 THEN z := 1 ELSE w := 2;**

**Parse Tree 1**

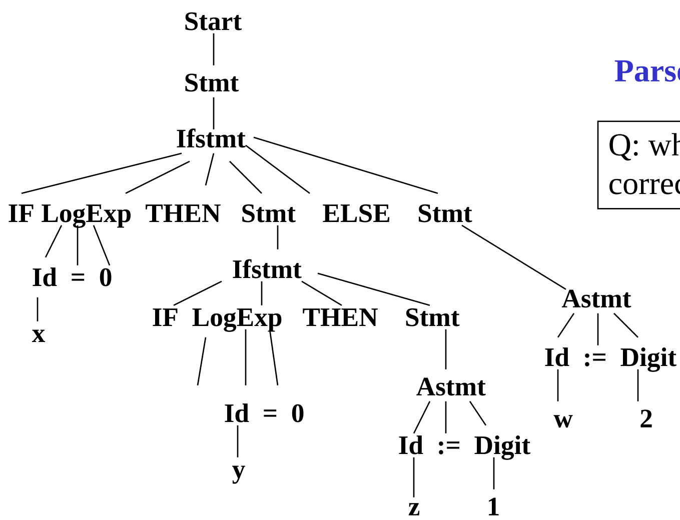





**IF x = 0 THEN IF y = 0 THEN z := 1 ELSE w := 2;**

**Parse Tree 2**

Q: which tree is correct?



# How Solve the Dangling Else?

- ***Algol60:*** **use block structure**

  **if x = 0 then begin if y = 0 then z := 1 end else w := 2**

- ***Algol68:*** **use statement begin/end markers**

  **if x = 0 then if y = 0 then z := 1 fi else w := 2 fi**

- ***Pascal:*** **change the if statement grammar to disallow parse tree 2; that is,** ***always associate an else with the closest if***

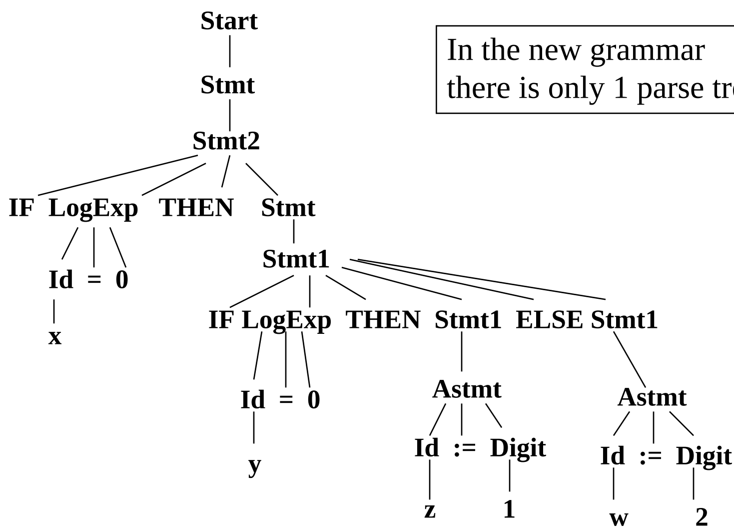


# New Pascal Grammar

| | |
|---|---|
| **Start ::=** | **Stmt** |
| **Stmt ::=** | **Stmt1 \| Stmt2** |
| **Stmt1 ::=** | **IF LogExp THEN Stmt1 ELSE Stmt1 \| Astmt** |
| **Stmt2 ::=** | **IF LogExp THEN Stmt \| IF LogExp THEN Stmt1 ELSE Stmt2** |
| **Astmt ::=** | **Id := Digit** |
| **Digit ::=** | **0\|1\|2\|3\|4\|5\|6\|7\|8\|9** |
| **LogExp::=** | **Id = 0** |
| **Id ::=** | **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: only if statements with IF..THEN..ELSE are allowed after the THEN clause of an IF-THEN-ELSE statement.***

**IF x = 0 THEN IF y = 0 THEN z := 1 ELSE w := 2;**




In the new grammar there is only 1 parse tree!

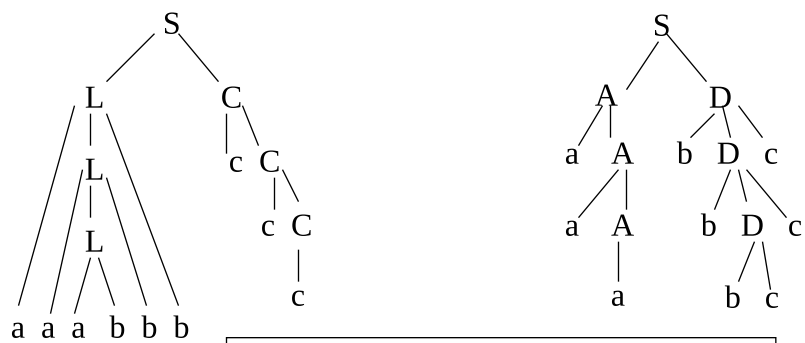


# Ambiguity

- **Sometimes we can remove an ambiguity from a grammar by restructuring the productions, but it is not always possible**
  - **An *inherently ambiguous* language does not possess an unambiguous grammar**
  - **E.g., L = { $a^i b^j c^k$ | i = j or j = k for i,j,k >=1} generated by grammar:**

| | | |
|---|---|---|
| **S ::= L C \| A D** | **L ::= a L b \| ab** | |
| **C::= c \| cC** | **D::= bDc \| bc** | **A::= a \| aA** |

# Parse Trees




parse trees for $a^3 b^3 c^3$ in L

problem is L contains a non-context-free language $\{a^n b^n c^n, n >= 1\}$



# Ambiguity

- **There is no algorithm which can examine an arbitrary context-free grammar and tell if it is ambiguous or not**
  - **This is *undecidable***
- **There is no algorithm which can examine two arbitrary context-free grammars and tell if they generate the same language**
  - **This is *undecidable***

# Chomsky Hierarchy

- **Describes categories of languages which correspond to more and more powerful recognizing automata**
- **4 level hierarchy**
  - **We've studied bottom two levels: regular and context-free languages**



# Type 3 (regular) Languages

- ***Recognizer*: finite state automaton**
- **Can do simple recursive constructs**
- **Can't count (or match parentheses)**
  - **Not regular $\{a^n b^n, n>=1\}$**
- **Can be written with all right recursive or all left recursive rules**
  - **Nonterm ::= term | Nonterm term**

# Type 2 (context-free) Languages

- *Recognizer*: push down automaton
- BNF rules with 1 nonterminal on lefthandside
- Can't check *context*
  - Not context-free $\{a^n b^n c^n, n>=1\}$
  - Programming examples
    - Check that no variable is declared twice
    - Check difference between function calls and array accesses in Fortran (both use parentheses)

DIMENSION …. F(10,10)….F(I,J)….



# Context-free Languages

- Check matchup of arguments with parameters in Pascal using nested function definitions

*procedure p (x :integer, y:real)*
  *procedure q (w: integer)*
  *… P(50,1.2)…*
  *end q*
*…Q(1)…*
*end P*

pattern seen is (parms p)(parms q) (args p) (args q)
corresponding language is $\{a^n b^m c^n d^m, m,n >= 1\}$

# Type 1 (context-sensitive) Language

- ***Recognizer*: linear bounded automaton**
- **Grammar rules can have more than 1 symbol on lefthandside as long as |rhs| >= |lhs|**
- **Can do parameter - argument matching (in number)**
- **Examples:**

  $\{a^n b^m c^n d^m, m,n >= 1\}$

  $\{a^n b^n c^n, n >= 1\}$



# Context-sensitive Example

| | | |
|---|---|---|
| **1. T ::=** | **S** (2a, 2b) | |
| **2. S ::=** | **a S B C \| a B C** | |
| **3. CB ::=** | **BC** | **--reverse B's and C's** |
| **4. aB ::=** | **ab** | |
| **5. bB ::=** | **bb** | **--expand B** |
| **6. bC ::=** | **bc** | **--expand C** |
| **7. cC ::=** | **cc** | |

**Derive aabbcc:**

$T \xrightarrow{1} S \xrightarrow{2a} a\,S\,B\,C \xrightarrow{2b} aaB\,C\,B\,C \xrightarrow{3} aa\,B\,B\,C\,C \xrightarrow{4} aabB\,C\,C \xrightarrow{5} aabbCC \xrightarrow{6} aabbcC \xrightarrow{7} aabbcc$

# Type 0 (recursively-enumerable) Languages

- *Recognizer*: Turing machine
- All languages that can be recognized by a procedure
- Subclass of Type 0: Recursive languages, languages recognized by an algorithm that always halts



# Turing Machines, Lightly

- Abstract model of computation
- <finite set of states, alphabet, blank symbol, start state, final state, transition function>
  - transition function:

  <state, tape symbol read> → <state, tape symbol wrote, {L,R,S}> where L,R,S means tape moves 1 square to the Left, Right, No move
- TM Halting problem: Given a TM in an arbitrary configuration with nonblank symbols on its tape, will the TM eventually halt? -- unsolvable!
  - There cannot exist an algorithm to solve this problem for an arbitrary choice of Turing machine on arbitrary input, although for a specific TM with specific input, there may be a solution.