

## Specification of Syntax

In English: long, complete?, unambiguous?

e.g. does the syntax of Pascal allow a program without any statements? any declarations?

In some formal notation:

Language = set of valid sentences + structure

- validity specified by **generator (which enumerates them all)** or **recognizer (which says Yes/No to any string passed to it)**

- structure used for semantics  
"I saw the man with the cat / telescope"

1-2-3 is 1-(2-3) vs. (1-2)-3

1

## Context Free Grammars

### Grammar for decimal numbers

(no 0 at front or tail):

terminals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, .

nonterminals: S, After, Before, Digit

productions:

$D \rightarrow 1 \quad D \rightarrow 2 \quad \dots \quad D \rightarrow 9$

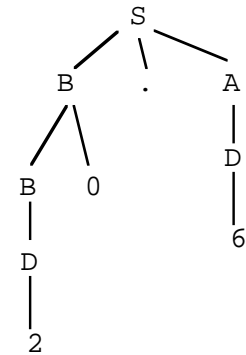
$S \rightarrow B . A \quad S \rightarrow B$

$A \rightarrow D A \quad A \rightarrow 0 A \quad A \rightarrow D$

$B \rightarrow B D \quad B \rightarrow B 0 \quad B \rightarrow D$

### "deriving/generating" 20.6

$S \Rightarrow B . A$   
 $\Rightarrow B . D$   
 $\Rightarrow B . 6$   
 $\Rightarrow B 0 . 6$   
 $\Rightarrow D 0 . 6$   
 $\Rightarrow 2 0 . 6$



3

## Grammars & Derivations

1) production rule

$S \rightarrow N V \text{ Obj}$   
 $\text{Obj} \rightarrow N$   
 $\text{Obj} \rightarrow \text{him}$   
 $N \rightarrow \text{bob} \quad N \rightarrow \text{ibm}$   
 $V \rightarrow \text{likes} \mid \text{hates}$

3) terminals / tokens

bob hates ibm

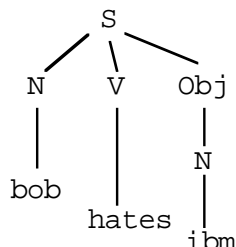
4) non-terminals

N Obj S V  
 ← start symbol

2a) derivation

$S \Rightarrow N V \text{ Obj} \Rightarrow$   
 bob V Obj  $\Rightarrow$   
 bob hates Obj  $\Rightarrow$   
 bob hates N  $\Rightarrow$   
 bob hates ibm

2b) parse tree



5) LANGUAGE: sentential form, sentence, parse tree

2

## BNF notation Sethi 1.5

BNF rules: meta-symbols < > | ::=

$\langle D \rangle ::= 1 \mid 2 \mid \dots \mid 9$

$\langle S \rangle ::= \langle B \rangle . \langle A \rangle$

Extended BNF: meta-symbols | ::= ( ) [ ] ' or \_

$D ::= '1' \mid '2' \mid \dots \mid '9'$

$S ::= B [ '.' A ]$

$A ::= ( D \mid '0' ) A \mid D$

$B ::= B ( D \mid '0' ) \mid D$

alternative rule using notation  $\{x\}$ : "0 or more x"

$A ::= \{ D \mid 0 \} D \quad B ::= D \{ D \mid 0 \}$

As desired, can't generate 02.6 nor 2.50.

But what about just 0, or 0.25?

? add ?  $B ::= 0$  ?

add  $S ::= 0 [ . A ]$

**Recursion:** symbols B and A; needed to generate infinite languages;

A is **right-recursive**; B is **left-recursive**;

4

**Grammar for valid Pascal identifiers:**

must start with letter, can be digit afterwards.

$Id ::= Letr \mid Letr \ Id \mid Dig \ Id \mid Dig$

**Problem!!!**  $Id \Rightarrow Dig \ Id \Rightarrow Dig \ Dig$

**Solution:**

$Id ::= Letr \mid Letr \ Id_0$   
 $Id_0 ::= Letr \ Id_0 \mid Dig \ Id_0 \mid \epsilon$

Null string

No more ambiguity. And to get 4-3-2 to come out right as (4-3)-2, we need left associative.

But left assoc. gets 4-3\*2 wrong as (4-3)\*2!

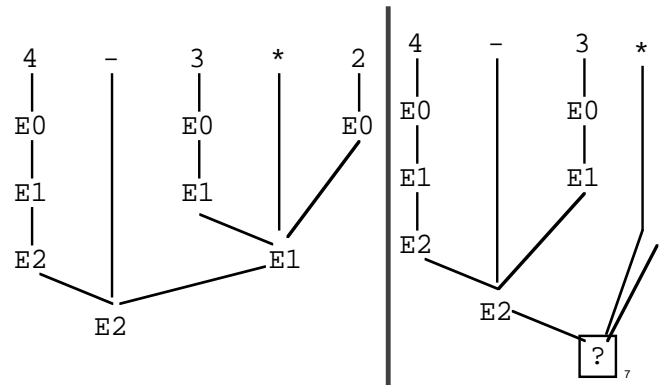
Something else going on here:

**PRECEDENCE.**

To deal with this, need multiple "levels" of non-terminals.

E0 -- basic things  
 E1 -- higher precedence  
 E2 -- lower precedence

$E_0 ::= Int$   
 $E_1 ::= E_1 * E_0 \mid E_0$   
 $E_2 ::= E_2 - E_1 \mid E_1$



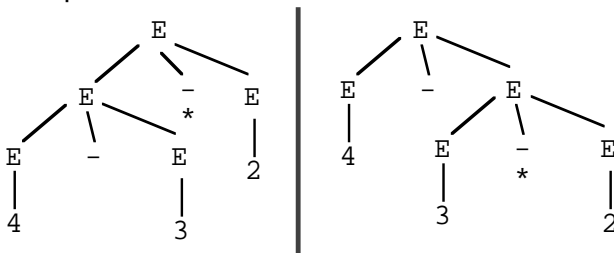
**Ambiguity, associativity**

**Grammar for arithmetic expressions:**

$E ::= E * E \mid E - E \mid 1 \mid 2 \mid \dots$

**Ambiguous:**

2 parse trees each for 4-3-2 and 4-3\*2



**LEFT associativity** ((( x - y ) - z ) - v)  
 through **LEFT recursion**

$E ::= E - Int \mid E * Int \mid Int$   
 allows only for the structure on the left

**RIGHT associative** (x - (y - (z - v)))  
 through **RIGHT recursion**

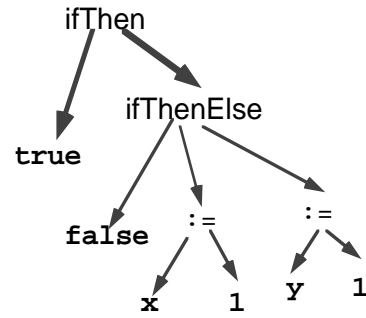
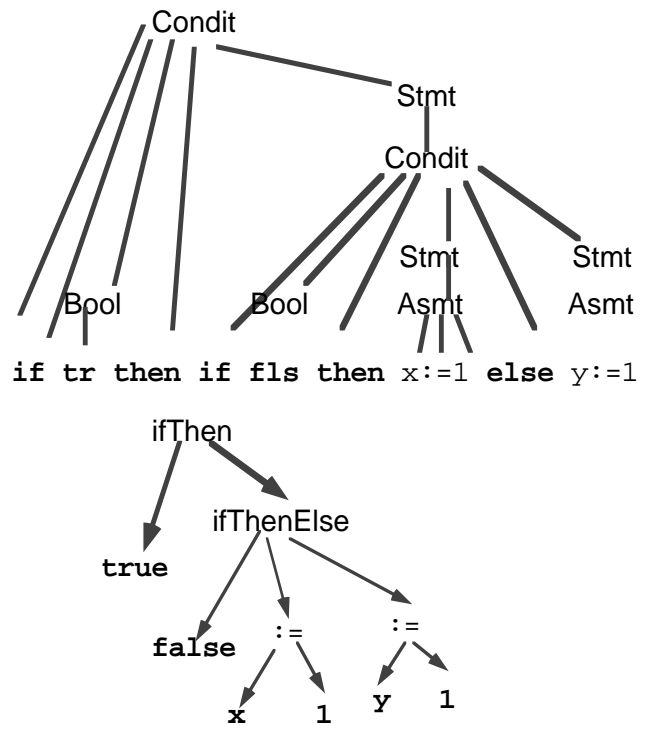
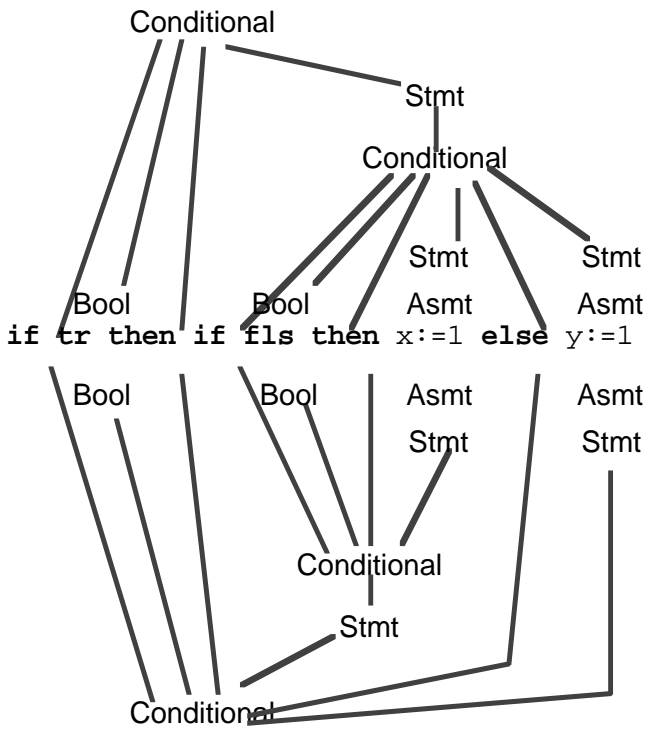
$E ::= Int - E \mid Int * E \mid Int$   
 allows only for the right

**Dangling ELSE**

Grammar for programming langs:

$Expr ::= Expr - Expr \mid Expr + Expr \mid \dots$   
 $Bool ::= \underline{tr} \mid \underline{fls} \mid Expr \leq Expr \dots$   
 $Stmt ::= Asmt \mid Loop \mid Conditional \mid \dots$   
 $Asmt ::= VarID := Expr$   
 $Loop ::= \underline{WHILE} \ Bool \ \underline{DO} \ Stmt \ \underline{END}$   
 $Conditional ::=$   
      $\underline{IF} \ Bool \ \underline{THEN} \ Stmt$   
      $\mid \underline{IF} \ Bool \ \underline{THEN} \ Stmt \ \underline{ELSE} \ Stmt$

## Concrete vs Abstract syntax



## Solutions

### Algol 60

*if tr then begin if fls then  
x:=1 end else y:=1*

### Algol 68

*if tr then if fls then x:=1 fi  
else y:=1 fi*

### Pascal: else goes with the nearest then

can be achieved with grammar that assures anything after **then** must be matched (ie, cannot end with **then** followed by some other statement)

**Stmt ::= Matched | Unmatched**

**Matched ::= if Expr then Matched else Matched  
| other**

**Unmatched ::= if Expr then Stmt  
| if Expr then Matched else Unmatched**

### Excercise: Try parsing all these combinations

if T {if T E}  
    {if T }  
  
    {if T E} E {if T E}  
    {if T } {if T }

## Regular Expressions for describing tokens

Describe sets of strings -- languages -- over some 'alphabet'  $\Sigma$  by expressions.

Name	Notation	Denotation $\delta$
letter	a	{a}
null	$\epsilon$	{ $\epsilon$ }
union	(R   S)	$\delta(R) \cup \delta(S)$
sequence	(R S)	{uw   u $\in \delta(R)$ , w $\in \delta(S)$ }
Kleene star	(R*)	$\epsilon \cup R \cup RR \cup RRR \dots$
empty set	$\emptyset$	{}

precedence: \* > . > |

e.g.:

8 9	{8,9}
9(8 0)	{98,90}
8 $\emptyset$ 9	{}
8 $\epsilon$ 9	{89}
(8 9)*	{ $\epsilon, 8, 9, 88, 89, 98, 99, 888, 889, \dots$ }
$\emptyset^*$	$\epsilon$

**Writing regular expressions for desired languages**

All binary numbers

$(0|1)^*$

Binary numbers with at least two 1's

$(0|1)^* 1 (0|1)^* 1 (0|1)^*$

Pascal identifiers

**Let** =  $a | b | c | \dots$

**Dig** =  $0 | 1 | 2 | \dots$

**Ident** = **Let** (**Dig** | **Let**)\*

Strings over {8,9} with no consecutive 8's

$9^*$  -- no 8's

|  $9^*89^*$  -- one 8

|  $9^*899^*89^*$  -- two 8's separated

|  $9^*899^*899^*89^*$  -- three 8's "

...

which is

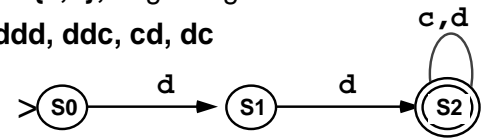
$9^*(899^*)^*89^* | 9^*$

*Hint:* ask "did you get too much?", as well as "did you get everything?"

**Examples**

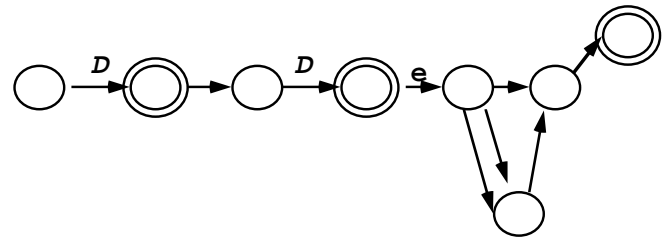
"Strings over {c,d}, beginning with dd"

Trace **ddd, ddc, cd, dc**



"Unsigned Pascal numbers:"

- integers
- decimals with 1 or more digits on each side of decimal point .
- scientific notation **eXX**



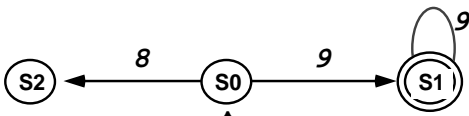
**Finite Automata recognizing regular languages**

**Diagram/chart:**

**states (nodes):** initial, finals, other

**transitions (arcs):** labels from  $\Sigma, \epsilon$

**recognition:** consecutive symbols in a string cause transitions; end in final.



**Tabular representation:** s0 is initial, s1 is final

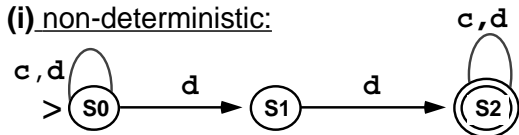
	8	9
S0	S2	S1
S1	S1	S1
S2	-	-

**Deterministic vs. non-deterministic** (Accept if there is SOME way to a final state)

	8	9
S0	S2	S1
S1	S1	S1
S2	D	D
D	D	D

"Strings over {c,d}, with **dd** as substring":

(i) **non-deterministic:**



(ii) **deterministic:**

- prefix before **dd** is null
- prefix non-null
  - start with **c** --> keep looking for a **d**
  - start with **d** --> if next is **d**, ok if next is **c**, "false start"

**From RE to FA:**  $(ee|ff)^*$

**From FA to RE:** trace paths going to final state