

The “Dangling Else” problem can be solved semantically by having a rule that states the ELSE belongs to the last preceding IF for which there is not already an associated ELSE. But the *syntax* is still ambiguous.

Modula-2 fixes the syntax in the following way.

```
<statement> ::= IF <expression>
                THEN <statementsequence> END
```

```
<statement> ::= IF <expression>
                THEN <statementsequence>
                ELSE <statementsequence> END
```

Slide Lecture 9 –254

Productions such as $S \rightarrow S + S$
are an inherent source of ambiguity.

Consider the following Grammar.

$$S \rightarrow S + S$$
$$S \rightarrow S * S$$
$$S \rightarrow (S)$$
$$S \rightarrow a$$

Slide Lecture 9 –255

We could include rules such as $S \rightarrow S - S$
but this is syntactically equivalent to $S + S$
in terms of “precedence” properties.

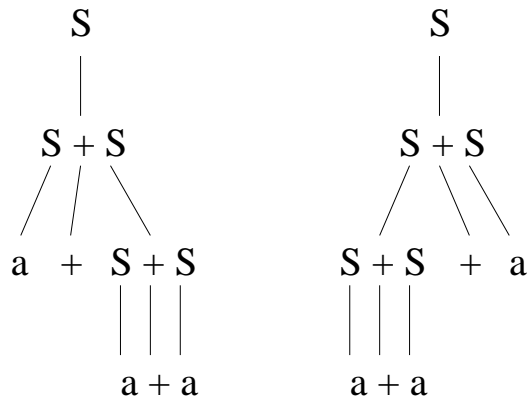
The same is true for $S * S$ and S / S .

Consider an expression as simple as

$a + a + a$

There are two parse trees.

Slide Lecture 9 -256



Slide Lecture 9 -257

This has a simple solution.

$$S1 \longrightarrow S1 + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (S1) \mid a$$

This grammar is equivalent to

$$S \longrightarrow S + S \mid S * S \mid (S) \mid a$$

but it is unambiguous.

Slide Lecture 9 –258

NORMAL FORMS and PARSE TREES

Parse Trees are graphical representations of the derivation of a string from a context free grammar. The derived string is the “yield” of the parse tree.

Given a Context Free Grammar,

$$G = (V, \Sigma, R, S)$$

1. For each symbol $A \in (V \cup \Sigma)$ the node

$$\bullet A$$

is a parse tree.

Slide Lecture 9 –259

2. For each production

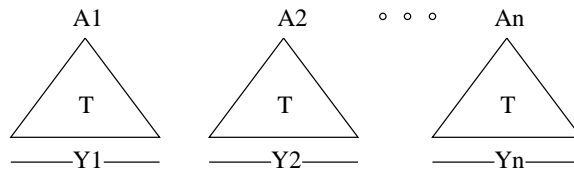
$$A \rightarrow \lambda, A \rightarrow \alpha$$

$$\begin{array}{cc} A & A \\ | & | \\ \lambda & \alpha \end{array}$$

are parse trees.

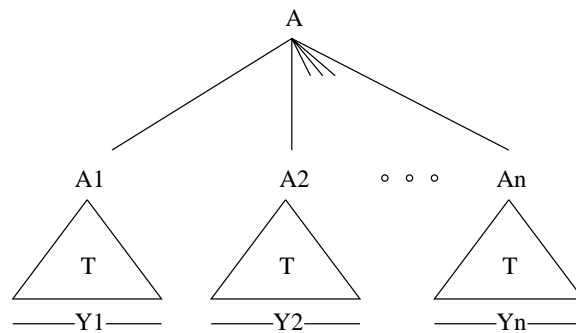
Slide Lecture 9 -260

3. If A_1, A_2, \dots, A_n are the roots of parse trees
with yield Y_1, Y_2, \dots, Y_n



Slide Lecture 9 -261

AND $A \rightarrow A_1 A_2 \dots A_n$ is a production
 then the following is parse tree:



Slide Lecture 9 -262

4. Nothing else is a parse tree.

Consider $G = (V, \Sigma, R, S)$

$V = \{ S \}$

$\Sigma = \{ (,) \}$

$R = \{ S \rightarrow SS$

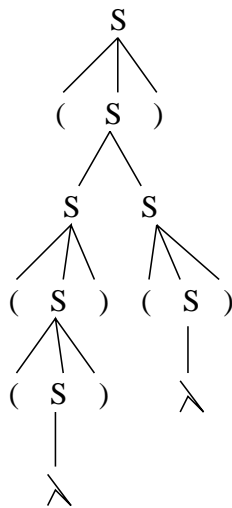
$S \rightarrow (S)$

$S \rightarrow \lambda$

Note that this is a contracting form.

Slide Lecture 9 -263

The parse tree for $((())())$ is



Slide Lecture 9 -264

Simplified Forms and Normal Forms

provide a restricted syntax so that all productions are represented in that form

We have already said that CF grammars are noncontracting.

What about rules like $A \rightarrow \lambda$?

Most Normal Forms do not include contracting productions.

How do we remove λ productions?

Slide Lecture 9 -265

If λ is in the language,
then we allow a rule

$$S \longrightarrow \lambda$$

where S is the start symbol.

Otherwise, we remove λ by substitution.

Remove 1 rule containing lambda at a time.

(The Iterative Method)

$$S \longrightarrow ABCBCD$$

$$A \longrightarrow CD$$

$$C \longrightarrow a \mid \lambda$$

$$B \longrightarrow Cb$$

$$D \longrightarrow bD \mid \lambda$$

Slide Lecture 9 -266

Consider the rule

$$S \longrightarrow ABCBCD$$

Since C and D appear 3 times and can be λ or not,
it requires at least 8 versions of just this rule to remove λ .

First remove $D \longrightarrow \lambda$

Slide Lecture 9 -267

S \rightarrow ABCBCD

S \rightarrow ABCBC

A \rightarrow CD

A \rightarrow C

C \rightarrow a | λ

B \rightarrow Cb

D \rightarrow bD

D \rightarrow b

Next remove C $\rightarrow \lambda$

Slide Lecture 9 -268

S \rightarrow ABCBCD | ABBD | ABBCD | ABCBD

S \rightarrow ABCBC | ABB | ABCB | ABBC

A \rightarrow CD

A \rightarrow D

A \rightarrow C

A \rightarrow λ

C \rightarrow a

B \rightarrow Cb

B \rightarrow b

D \rightarrow bD

D \rightarrow b

We created A $\rightarrow \lambda$

Slide Lecture 9 -269

Now remove $A \rightarrow \lambda$

We keep all the rules for S that reference A, then add the same rules with A removed.

$S \rightarrow$ ABCBCD | ABBD | ABBCD | ABCBD |
ABCBC | ABB | ABCB | ABBC

$S \rightarrow$ BCBCD | BBD | BBCD | BCBD |
BCBC | BB | BCB | BBC

$A \rightarrow$ CD | D | C

$C \rightarrow$ a

$B \rightarrow$ Cb | b

$D \rightarrow$ bD | b

Slide Lecture 9 -270

Unit Productions

These are rules such as

$A \rightarrow C$ $A \rightarrow D$

Again these can be removed
iteratively by substitution.

Slide Lecture 9 -271

$A \rightarrow C$ and $C \rightarrow a$
yields

$A \rightarrow a$

$A \rightarrow D$ and $D \rightarrow bD \mid b$
yields

$A \rightarrow bD \mid b$

Slide Lecture 9 -272