Parsing (cont)

Today:

- HW due Friday
- Today: - after class
  or 1:30-2
- Next HW - due next Friday
  over flex
  submit via git

Parsing:

- Given string of input tokens, a parser must determine if the tokens generate a valid program

The basis of these are context free grammars (CFGs):

- terminals: for, +, { → lowercase
- nonterminals (one a start symbol) S
  typically uppercase or underlined
- production rules
  ↳ tell transition

Notation: start

$$
\begin{aligned}
\underline{expr} &\rightarrow \underline{expr}\ \underline{op}\ \underline{expr} \\
&\mid (\ \underline{expr}\ ) \\
&\mid \text{id (variable)} \\
\underline{op} &\rightarrow + \mid - \mid * \mid /
\end{aligned}
$$

Ex:

capital, so non-terminal

$$
\begin{aligned}
E &\to E\ A\ E \\
&\to (E) \\
&\to -E \\
&\to id
\end{aligned}
$$

terminals

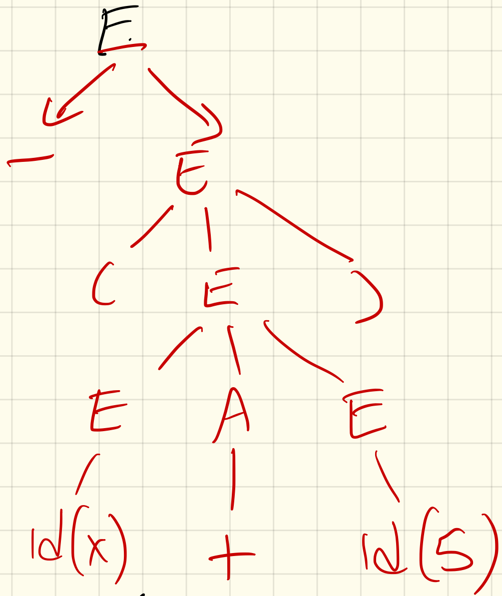
$$
\begin{aligned}
A &\to + \\
&\to - \\
&\to * \\
&\to / \\
&\to \uparrow
\end{aligned}
$$

Derivation: The process by which a grammar parses & defines a language.

Ex: Show $-(x+5)$ is accepted by the above grammar:

$$
\begin{aligned}
E &\Rightarrow -E \Rightarrow -(E) \\
&\Rightarrow -(E\ A\ E) \Rightarrow -(id(x)\ A\ E) \\
&\Rightarrow -(id(x) + E) \\
&\Rightarrow -(id(x) + id(5))
\end{aligned}
$$

<u>Parse tree</u>: A graphical representation of this derivation:



Each parent/child shows one step of the derivation

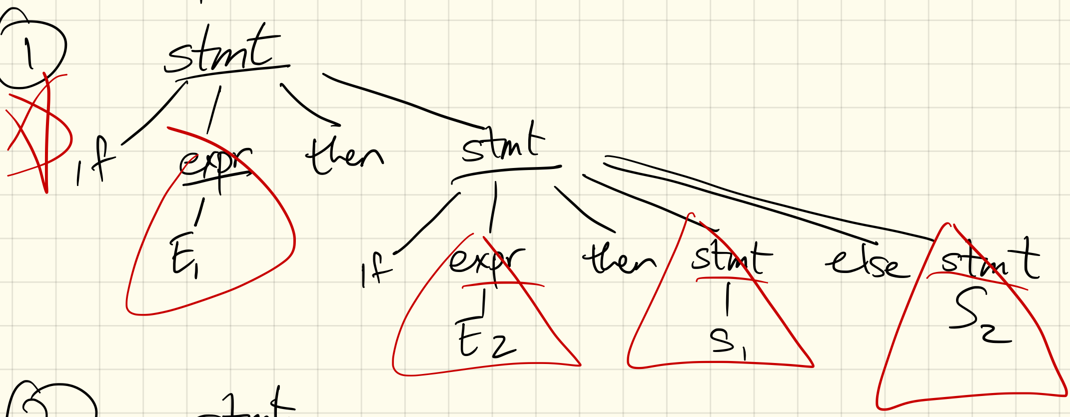
- leaves are terminals
- root is start non-terminal

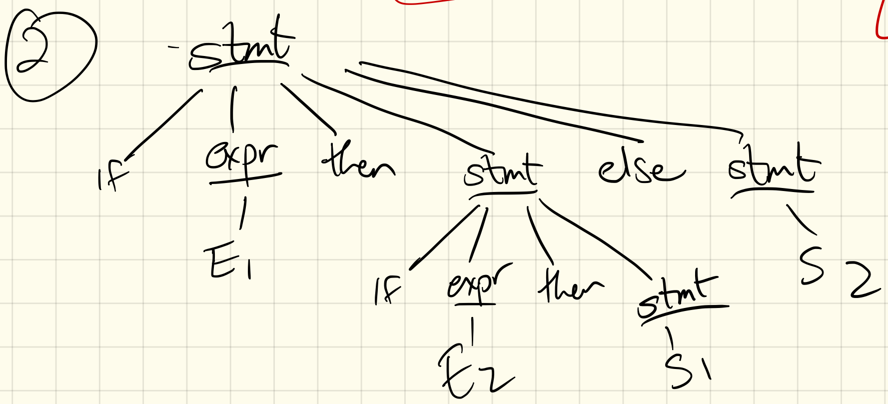
Other things

- Left most vs rightmost
- Ambiguity

Ex: if E_1 then if E_2 then S_1 else S_2

2 parse trees:





General rule:

Match each else w/ closest unmatched then

How?

- Rewrite so any statement between an "else" + a "then" must be matched (so no if-then w/ no else)

Grammar:

stmt → matched_stmt
  | unmatched_stmt

matched_stmt → if expr then matched_stmt else matched_stmt
  | other ← not an if statement

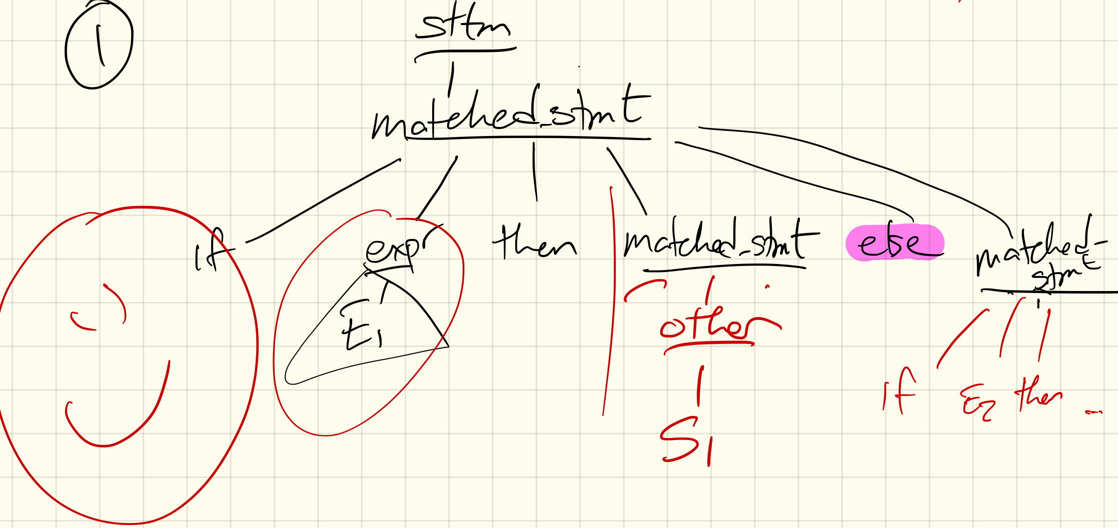
unmatched_stmt → if expr then stmt
  | if expr then matched_stmt else unmatched_stmt

next time – parsing

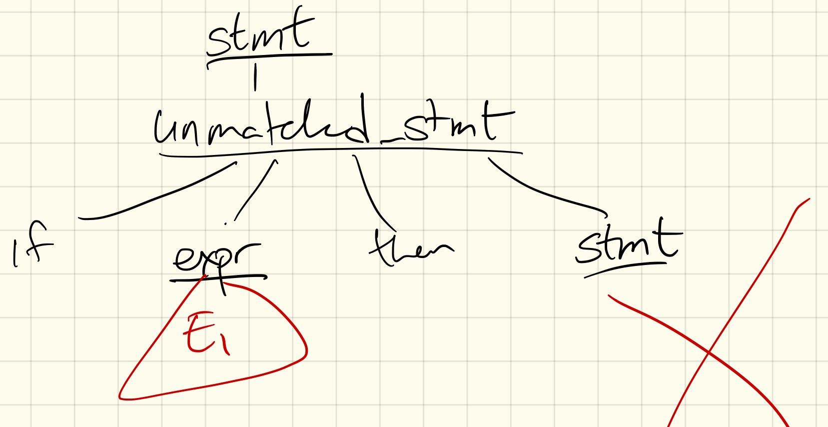
Example:

if E_1 then (S_1 else (if E_2 then S_2 else S_3))

(1)



(2)

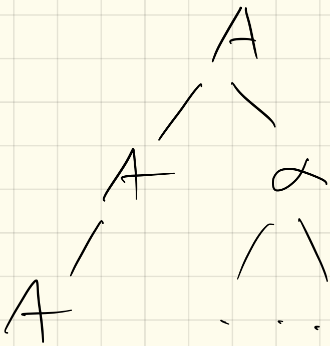


$\Rightarrow$ only 1 parsing

Dfn: A grammar is left-recursive if it has a non-terminal A with some rule

$$A \to A\alpha$$

These are bad for persers:

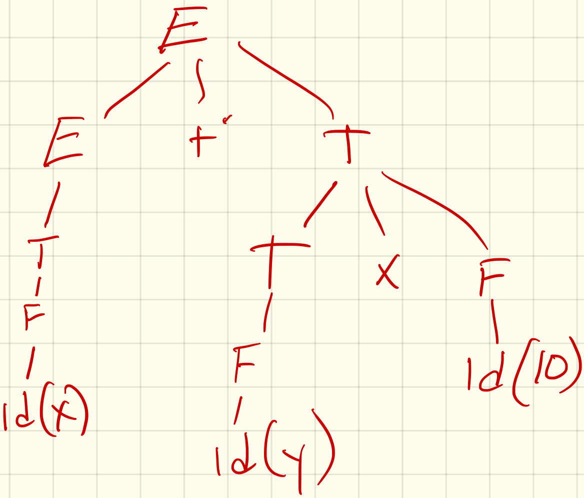


When scanning tokens & trying to build a tree, not sure when to stop!

Ex: start

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T \times F \mid F$$
$$F \rightarrow (E) \mid id$$

Parse: $x + y \times 10$



This deals nicely w/ precedence.
However, we do have <u>left recursion</u>!

To eliminate:

Rule: $A \to A\alpha \mid \beta$

any other term / non-term / list

$$A \to \beta A'$$
$$A' \to \alpha A' \mid \varepsilon$$

Ex: On

$$E \to E\overbrace{+T}^{\alpha} \mid \overset{\beta}{T}$$
$$T \to T\underbrace{\times F}_{\alpha} \mid \underset{\beta}{F}$$
$$F \to (E) \mid id$$

$$E \to TE'$$
$$E' \to +TE' \mid \varepsilon$$
$$T \to FT'$$
$$T' \to \times FT' \mid \varepsilon$$
$$F \to (E) \mid id$$

Back to the practical:

- Any CFG can be parsed
  ↳ Chomskey Normal Form
  CYK algorithm
  Run time:

This is too slow!

Most modern parsers look for certain restricted families of CFGs.

Result:

Top down parsing

Called predictive parsing.

Works well on LL(1) grammars.

Ex: $S \rightarrow cAd$

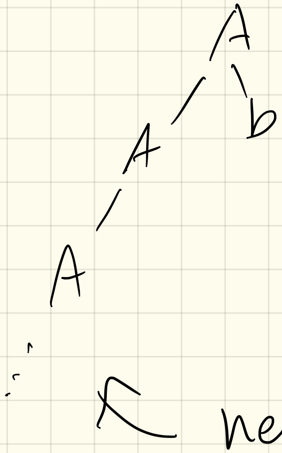
$A \rightarrow ab / a$

Parse cad:

Rule: Sterring w/ S,
apply rules until
one matches the
next input
(back track if there
is a mistake)

Note: Left recursion is *very* bad on these!

$$A \to A b$$



never matches an input or hits a conflict

So never forced to backtrack.

How predictive parsing works:

- the input string w is in an input buffer.
- Construct a predictive parsing table for G.
- if you can match a terminal, do it (+ move to next character)
- otherwise, look in table for rule to get transition that will eventually match

Hard part:

- build the table

  (need to decide a transition if at a nonterminal based on the next input terminal)

FIRST & FOLLOW Sets:

FIRST (α) ← any string of non-terminals & terminals

:= set of possible first terminals in any derivation of α by the grammar

So:

1) if x is a terminal,

$FIRST(x) =$

2) if $X \to \varepsilon$ is a production, add ε to $FIRST(X)$

3) If X is a nonterminal:

if $X \to Y_1 Y_2 \dots Y_k$ is a production:

add a if a is in $FIRST(Y_i)$ and ε is in $FIRST(Y_1), \dots, FIRST(Y_{i-1})$

add ε if ε is in $FIRST(Y_1), \dots FIRST(Y_k)$

Ex:

$$E \to TE'$$
$$E' \to +TE' \mid \varepsilon$$
$$T \to FT'$$
$$T' \to *FT' \mid \varepsilon$$
$$F \to (E) \mid id$$

FIRST (E) =

FIRST (E')

FIRST (T)

FIRST (T')

FIRST (F)