More passing

Today:

- HW due
- Next HW cover flex (on hopper)
- Git for next HW (due Friday)
- I am gone Mon + Tues (you have class)

Last time

- More parsing:
  - removing ambiguity
  - eliminating left recursion

Back to the practical:

- Any CFG can be parsed
  ↳ Chomskey Normal Form
  CYK algorithm
  Run time: $O(n^3)$

This is too slow!

Most modern parsers look for certain restricted families of CFGs.

Result:
- LL - faster, but weaker
- LR - stronger but "slower"

Have $O(n)$ time parsing

LL:

- left to right parsing
- leftmost derivation

Anything accepted by this type of parser is called an LL grammar.

Picture:

$$\underline{A}BC$$

$$\Rightarrow \underbrace{\alpha}\,\underline{B}C$$

$A \to \alpha$

Top down parsing (for LLs)

Called predictive parsing.
Works well on LL(1) grammars.

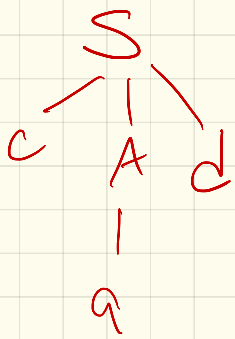
- Table based in practice

Simple Ex: $S \to cAd$

$A \to ab \mid a$

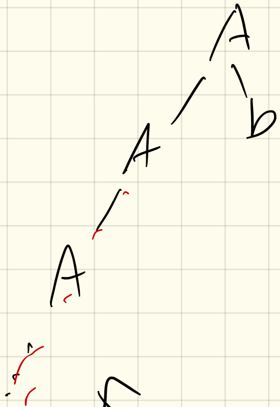
Parse cad:

Rule: Start 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(s) terminal)

LL(k):

FIRST & FOLLOW Sets (for LL(1)):

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,

$$\text{FIRST}(x) = x$$

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), ..., FIRST(Y_{i-1})

add ε if ε is in FIRST(Y_1), ... FIRST(Y_k)

Ex:

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

$FIRST(E) = \{ (, id \}$

$FIRST(E') = \{ +, \varepsilon \}$

$FIRST(T) = \{ (, id \}$

$FIRST(T') = \{ *, \varepsilon \}$

$FIRST(F) = \{ (, id \}$

FOLLOW Sets:

(We'll assume any input ends in \$, just to have an end of file character)

Rules:

1) Put \$ in FOLLOW(S) where S is start symbol.

2) Given a production:

$$A \to \alpha B \beta$$

everything in FIRST(β) goes in FOLLOW(B) (except ε, if it is there).

3) Given a production:

$$A \to \alpha B$$

or $A \to \alpha B \beta$ with $\varepsilon \in \text{FIRST}(\beta)$

then everything in FOLLOW(A) also goes in FOLLOW(B)

Ex:

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

We had:

$$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{ (, id \}$$
$$\text{FIRST}(E') = \{ +, \varepsilon \}$$
$$\text{FIRST}(T') = \{ *, \varepsilon)$$

So:

$$\text{FOLLOW}(E) = \{), \$ \}$$
$$\text{FOLLOW}(E') = \{), \$ \}$$
$$\text{FOLLOW}(T) = \{ +,), \$ \}$$
$$\text{FOLLOW}(T') = \{ +,), \$ \}$$
$$\text{FOLLOW}(F) = \{ +, *,), \$ \}$$

Then, the Table: M:

For any production $X \to \alpha$, do

1) for each terminal a in $FIRST(\alpha)$, add

$X \to \alpha$ to $M[A, a]$

2) If ε is in $FIRST(\alpha)$, add $X \to \alpha$ to $M[A, \underline{b}]$

for each terminal b in $FOLLOW(A)$.

If ε is in $FIRST(\alpha)$ and $\$$ is in $FOLLOW(A)$, add $A \to \alpha$ to $M[A, \$]$.

Any other entries are errors.

(construct on board)

End result:

Inputs

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

Then: $\underbrace{\text{Parsing!}}$

Stack	Input	Action	Matched
E \$	id + id * id \$		