More parsing

Today:

- HW due Thursday (via git)

Last time

- More parsing: CFGs.
  - removing ambiguity (or recognizing)
  - eliminating left recursion
    - ↳ set of rules to apply to get rid of LR

$$S \to x A B$$

Goal: parse (apply prod. rules) until all non-terminals (which means it is valid)

or until stuck
↳ invalid

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: 2 main families:

→ LL: more limited, faster

LR: more general, slower

Both: $O(n)$ parsers

LL:
- left to right parsing
- Leftmost derivation

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

Recall:

Left to right:
on input string, try to force a rule to recognize leftmost terminal first

Leftmost der:
if nonterm, try to resolve left one first

Top down parsing (for LLs)

Called predictive parsing.

Works well on LL(1) grammars. scan 1 token at a time

∘ Table based in practice

Simple Ex:

$S \rightarrow cAd \mid aAa \mid cAAA$

$A \rightarrow ab \mid a$

Parse cad$:

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

$S \rightarrow cAd$

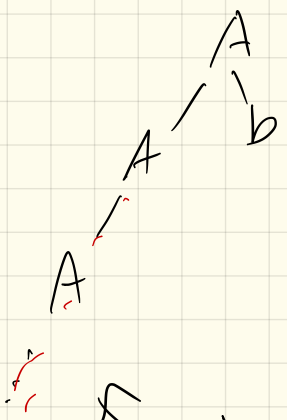
$\rightarrow \cancel{cabd}$

$\rightarrow cad\$$ ✓

Note: Left recursion is <u>very</u> bad on these!

$$A \to Ab \mid a$$

<u>a</u>b



A

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.
- Scan 1 ~~character~~ token at a time, & guess which rule should match

How?

- 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): k tokens to decide

(we'll just do LL(1))

Algorithm to construct table:

- Based upon listing "first" + "follow" sets for each non-terminal.

(Essentially, these will encode our predictions.)

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

$\text{FIRST}(\alpha)$ ← 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 $\text{FIRST}(X)$

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

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

Ex:

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

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

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: $S \to E\$$

$E \to TE'$

$E' \to +TE' \mid \varepsilon$

$T \to FT'$

$T' \to *FT' \mid \varepsilon$

$F \to (E) \mid id$

We had:

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

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

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

So:

$FOLLOW(S) = \{ \$ \}$

$FOLLOW(E) = \{ +,), *, \$ \}$

$FOLLOW(E') = \{ *, \$$

$FOLLOW(T) = \{ *, \$$

$FOLLOW(T') = \{ *, \$$

$FOLLOW(F) = \{ *, \$$

Then, the Table: M: (Next time)

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 $		