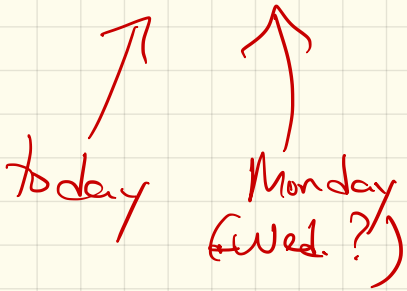
CSCI 3200

Final LL example

Today

- HW was due
  - How compilers work
  - Regular expressions!
- Next HW: over LL + LR parsing



LL Parsing:

[left-to-right
leftmost derivation]

Goal: "Fast" parsing

O(n)

One more example:

A grammar for lists/tuples:

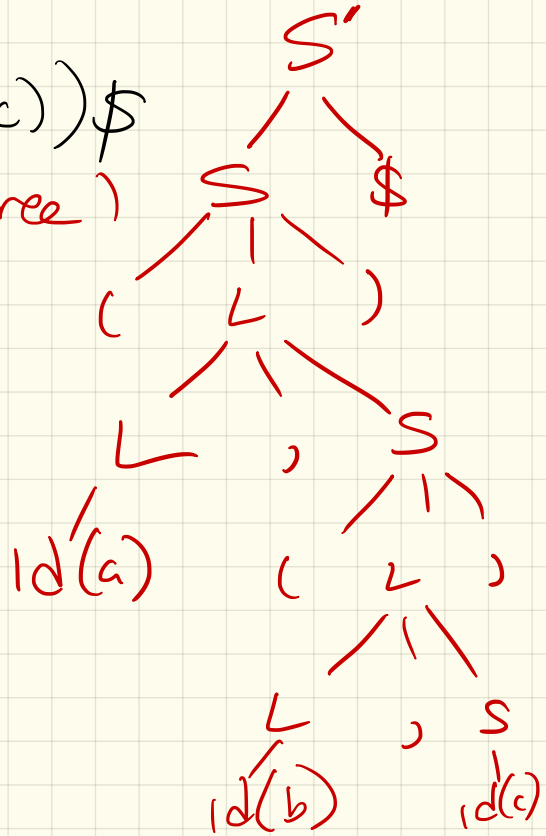
start → $S' \rightarrow S\$$

$S \rightarrow (L) \mid id$

$L \rightarrow L, S \mid id$

Ex: $(a, (b, c))\$$

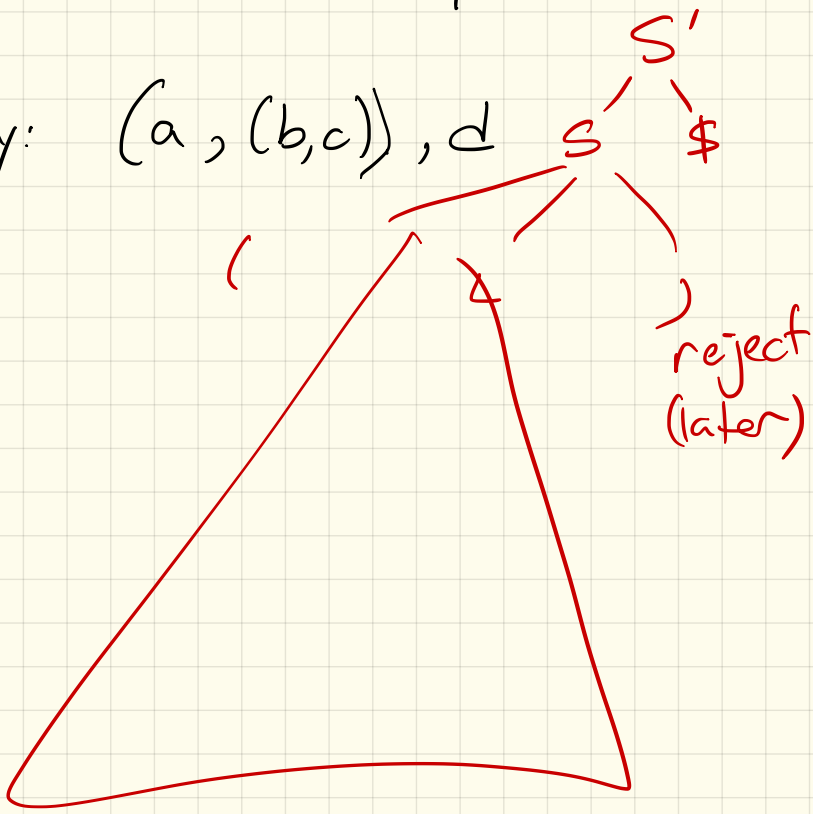
Derivation: (tree)



A (failing) example: how do rejects work?

$$S' \longrightarrow S\,\$$$
$$S \longrightarrow (L) \mid id$$
$$L \longrightarrow L, S \mid id$$

Try: (a, (b,c)), d



Problem:

$S' \rightarrow S\,\$$

$S \rightarrow (L) \mid id$

$L \rightarrow L, S \mid id$

Is this even LL?

No! left recursive

$L \rightarrow L'$

$L' \rightarrow$

LL version: same trick as before

$S' \to S\ \$$

$S \to (L) \mid id$

(Check later!) $L \to SL'$

$L' \to ,SL' \mid \varepsilon$

FIRST & FOLLOW Sets:

Table	FIRST	FOLLOW
S'	{ (, id }	{ $
S	{ (, id }	{ , ,), $
L	{ (, id }	{)
L'	{ , , ε }	{)

(Note: ε can't be in follow sets!)

Table: To generate

① For each terminal in FIRST(A), add $A \to \alpha$ to $M[A, a]$

② If $\varepsilon \in$ FIRST(A), then for each b in FOLLOW(A), add $A \to \alpha$ in $M[A, b]$

In ours, ε in FIRST(L')
Only thing in FOLLOW(L') is)

③ Any blanks become errors.

Table is key! Tells it how to parse.

Our table:

Nonterminals	(	)	id	,	$
S'	S'→S		S'→S		
S	S→(L)		S→id		
→ L	L→SL'		L→SL'		
L'		L'→ε		L'→,SL'	

State (including non-terms)

which rule we apply based on table

Matched	Stack	Input	Action
	S'$	(a,(b,c))	S'→S
	S$	(a,(b,c))	S→(L)
	(L)$	(a,(b,c))	match
(	L)$	a,(b,c))	L→SL'
"	SL')$	a,(b,c))	S→id
"	L')$	,(b,c))	match
(a	L')$	,(b,c))	L'→,SL'
"	,SL')$	,(b,c))	match
(a,	SL')$	(b,c))	

State (including non-terms)

queue

which rule we apply, based on table

Matched	Stack	Input	Action
	~~(~~ L') $	~~(~~ b, c))	match
(a	L') $	, b, c))	L' → , S
	, S L') $	, b, c))	match
(a ,	~~S~~ L') $	b, c))	S → (L)
	~~(~~ L) L') $	~~(~~ b, c))	match
(a , (	L) L') $	b, c))	L → S L'
	S L') L') $	b, c))	S → id
	id(b) L') L') $	b, c))	match
(a , (b	~~L~~) L') $	, c))	L → , S L'
	, S L') L') $	, c))	match
(a , b ,	S L') L') $	c))	S → id
	~~id~~ L') L') $	~~c~~))	match!
(a , (b , c	~~)~~ L') $	))	L' → ε
	) L') $	))	match
(a , (b , c)	L') $	)	L' → ε
	) $	)	match

Remember:

This whole approach is just to "automate" parsing.

LL is a simple yet powerful + fast class.

O(n): stack!

each stack op. is O(1)

each queue update: O(1)

downside - big table

Next time:

- LR
- Implementation
  (Bison)
  (or Yacc)