Tutorial 7

- Bottom-Up Parsing (LR(1) & SLR(1))

Bottom-Up Parsing

- Top-down parsing is ill-suited for left-recursive grammars
  - ↳ Big problem ∵ almost all programming languages are left-associative
- Bottom-Up parsing
  - ↳ given a CFG grammar $G = (N, \Sigma, P, S)$ & an input string $x \in \Sigma^*$, determine if $x \in L(G)$
  - ↳ To show $x \in L(G)$, show from x we can work backwards to find a derivation path $(\alpha_0, \alpha_1, \dots, \alpha_n)$ until we reach our start symbol S'

→ $x \Leftarrow \alpha_n \Leftarrow \alpha_{n-1} \Leftarrow \dots \Leftarrow \alpha_0 \Leftarrow S'$

Start at x find α_n ...   until we find S'

- LR Parsing
  - ↳ Start with input string $x' = \vdash x \dashv$, $x \in \Sigma^*$
  - ↳ ∀ symbols in x' we do one of the following:
    1. Shift : Consume the next input symbol & push it to the stack
    2. Reduce : If we recognize the right hand side B of a rule $(A \to B)$, then pop the right hand side of the rule (B) off the stack & push the left hand side onto the stack (A)
       - ↳ includes rules where $A \to \varepsilon$ (pushes A onto the stack, no pop)

- eg: Consider the CFG:

① $S' \to \vdash S \dashv$ ④ $X \to pX$ ⑦ $Y \to \varepsilon$

② $S \to Sab$ ⑤ $X \to \varepsilon$

③ $S \to XY$ ⑥ $Y \to q$

- defⁿ: An item is a production with a bookmark (denoted •) somewhere on the right hand side (RHS) of a rule
  - ↳ $S' \to \bullet \vdash S \dashv$ is a fresh item, none of the RHS is on the stack
  - ↳ If we push $\vdash$ on the stack, the rule updates to $S' \to \vdash \bullet S \dashv$ to tell us A is on the stack
  - ↳ Continue pushing symbols until $S' \to \vdash S \dashv \bullet$, entire RHS of the rule is on the stack ⟶ Reducible to S !
  - ↳ ☆ We can represent positions of the bookmark as states in a DFA! Transition based on symbols being pushed to the stack

- DFA production steps

① Create a start state with a single fresh item for the start rule S'

↳ ie: $\boxed{⓪ \; S' \to \bullet \vdash S \dashv}$ state label fresh item

② Select a state q_i that has ≥ 1 non-reducible item.
- ↳ For each non-reducible item, create a transition to a new state on the symbol after • in a rule
- ↳ In the new state, any $A \in N$ that follow the • should have their rules expanded into the new state as fresh items
  - ⟶ If this adds rules where • is before $B \in N$, expand for B's rules

3. Repeat 2. until no new states are discovered

4. Mark states containing reducible items as accept states

↳ ie: $\boxed{4\ S \to Sa \bullet b} \xrightarrow{b} \boxed{\boxed{10\ S \to Sab \bullet}}$ "reduce Sab to S"

- Problem: What if a state has ≥ 1 reductions or a mix of shifts & reductions?
- Shift-reduce Conflict: When a state has a shift & reduce item
  ↳ eg: n $A \to \alpha \bullet a$; $B \to \beta \bullet$
- reduce-reduce conflict: When a state has ≥ 1 reducible items
  ↳ eg: n $A \to \alpha \bullet$; $B \to \beta \bullet$
- A grammar is LR(0) iff $\nexists$ any shift-reduce and reduce-reduce conflicts in its automation
- SLR(1) uses one symbol lookahead to determine if it should shift or what reduction to apply
  ↳ With LR(0)'s DFA, uses the follow set of a reduction rule in a conflict to determine what action to take.
  → Reduce if the lookahead $\in$ Follow(A) for $A \to \beta \bullet$, else shift

eg:

0 $S' \to \bullet \vdash S \dashv$ $\xrightarrow{\vdash}$ 1 $S' \to \vdash \bullet S \dashv$

Push $\vdash$ to stack

SEN, add fresh items!

$S \to \bullet Sab$ — Already added S

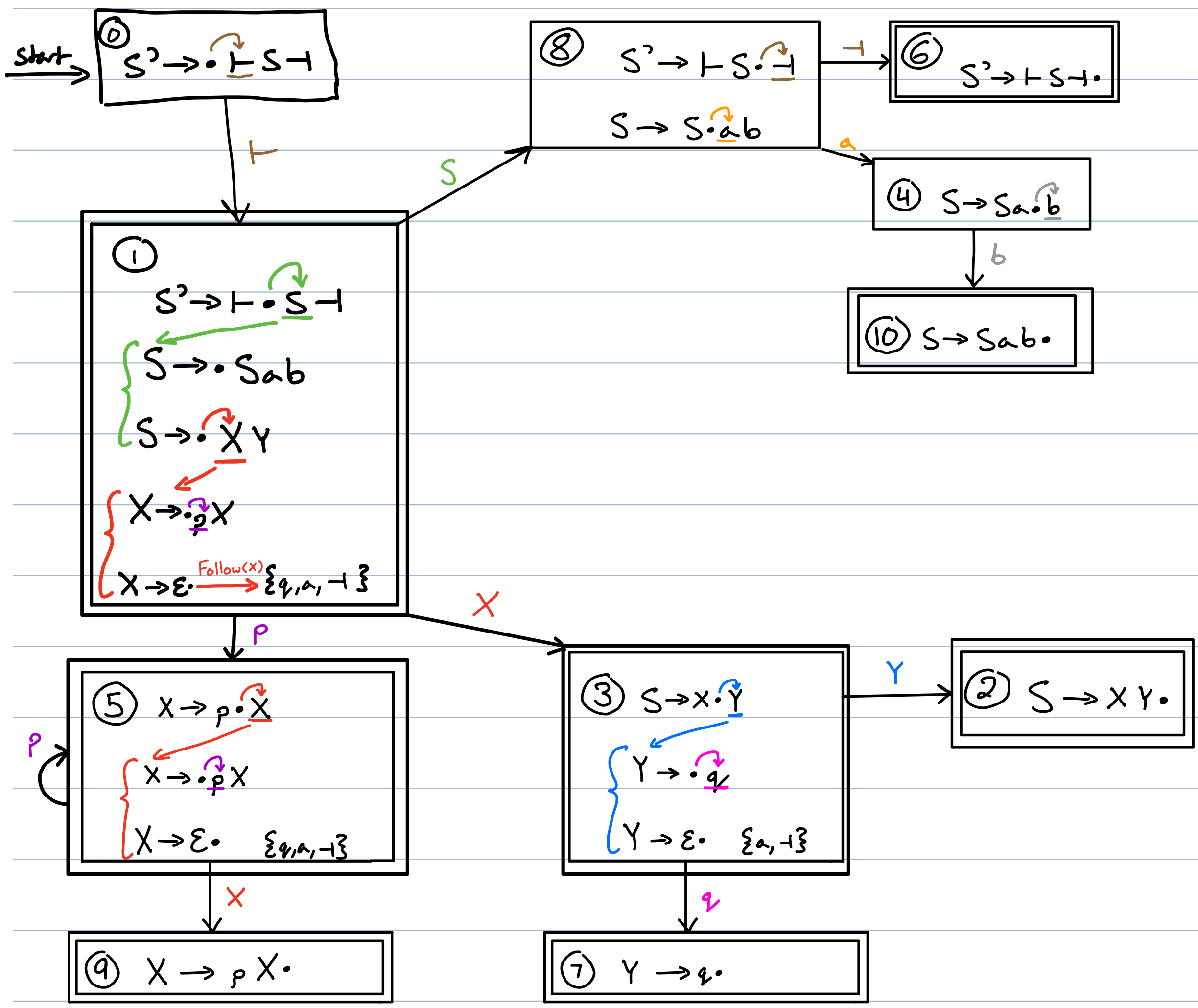
$S \to \bullet XY$ — XEN, add items

$X \to \bullet pX$ — $p \in \Sigma$, no expand

$X \to \varepsilon \bullet$ $\{q, a, \dashv\}$ — Follow(X) for SLR(1)

If the lookahead $\in$ Follow(X), we reduce, else we shift from the next symbol from input

SLR(1) Diagram



eg: Use the shift-reduce table to parse ⊢pqab⊣

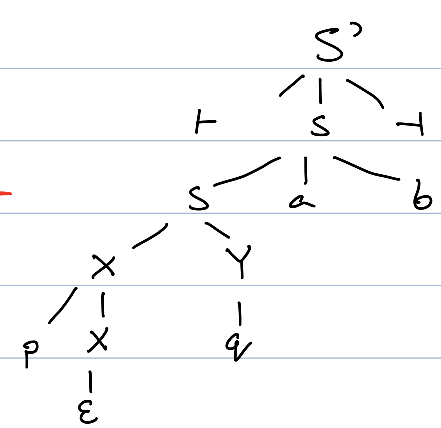
↳ LR Parsing Steps:

1. Begin with DFA state 0 on the stack
2. Use top of the state stack & 1st letter of input to determine next action
   - ↳ shift: move top of the input to the symbol stack & push the new state to the state stack
   - ↳ reduce: Remove a number of items equal to the length of the RHS rule then shift the LHS ∈ N
   - ↳ if the entry does not exist, reject ($\therefore x \notin L(G)$)
3. If we shift ⊣, accept! ($\therefore x \in L(G)$)

State Stack	Symbol Stack	Input	Action
ε	ε	$\vdash pqab \dashv$	Initialize, push 0
0	ε	$\vdash pqab \dashv$	Shift $\vdash$, push 1
$0\ 1$	$\vdash$	$pqab \dashv$	Shift p, push 5 ← $\because p \notin Follow(X)$ we don't reduce $X \to \varepsilon$
$0\ 1\ 5$	$\vdash p$	$qab \dashv$	Reduce $X \to \varepsilon$ ← $q \in Follow(X)$, pop $\lvert\varepsilon\rvert = 0$ from both stacks, shift X, push 9
$0\ 1\ \cancel{5}\ \cancel{9}$	$\vdash \cancel{p}\cancel{X}$	$qab \dashv$	Reduce $X \to pX$ ← pop $\lvert pX\rvert = 2$ from both stacks shift X, push 3
$0\ 1\ 3$	$\vdash X$	$qab \dashv$	Shift q, push 7
$0\ 1\ 3\ \cancel{7}$	$\vdash X\cancel{q}$	$ab \dashv$	Reduce $Y \to q$ — $\lvert q\rvert = 1$ pop shift Y, push 2
$0\ 1\ \cancel{3}\ \cancel{2}$	$\vdash \cancel{X}\cancel{Y}$	$ab \dashv$	Reduce $S \to XY$ — $\lvert XY\rvert = 2$ pops shift S, push 8
$0\ 1\ 8$	$\vdash S$	$ab \dashv$	Shift a, push 4
$0\ 1\ 8\ 4$	$\vdash Sa$	$b \dashv$	Shift b, push 10
$0\ 1\ \cancel{8}\ \cancel{4}\ \cancel{10}$	$\vdash \cancel{S}\cancel{a}\cancel{b}$	$\dashv$	Reduce $S \to Sab$ — $\lvert Sab\rvert = 3$ pops shift S, push 8
$0\ 1\ 8$	$\vdash S$	$\dashv$	Shift $\dashv$, push 6
$0\ 1\ 8\ 6$	$\vdash S \dashv$	ε	Accept!

- Rightmost derivation is obtained by reading the symbol stack concatenated with the remaining input bottom to top!

$$
\begin{aligned}
& S' \\
\Rightarrow\ & \vdash S \dashv \\
\Rightarrow\ & \vdash Sab \dashv \\
\Rightarrow\ & \vdash XYab \dashv \\
\Rightarrow\ & \vdash Xqab \dashv \\
\Rightarrow\ & \vdash pXqab \dashv \\
\Rightarrow\ & \vdash pqab \dashv
\end{aligned}
$$



eg: For the grammar:

① $S' \to \vdash S \dashv$

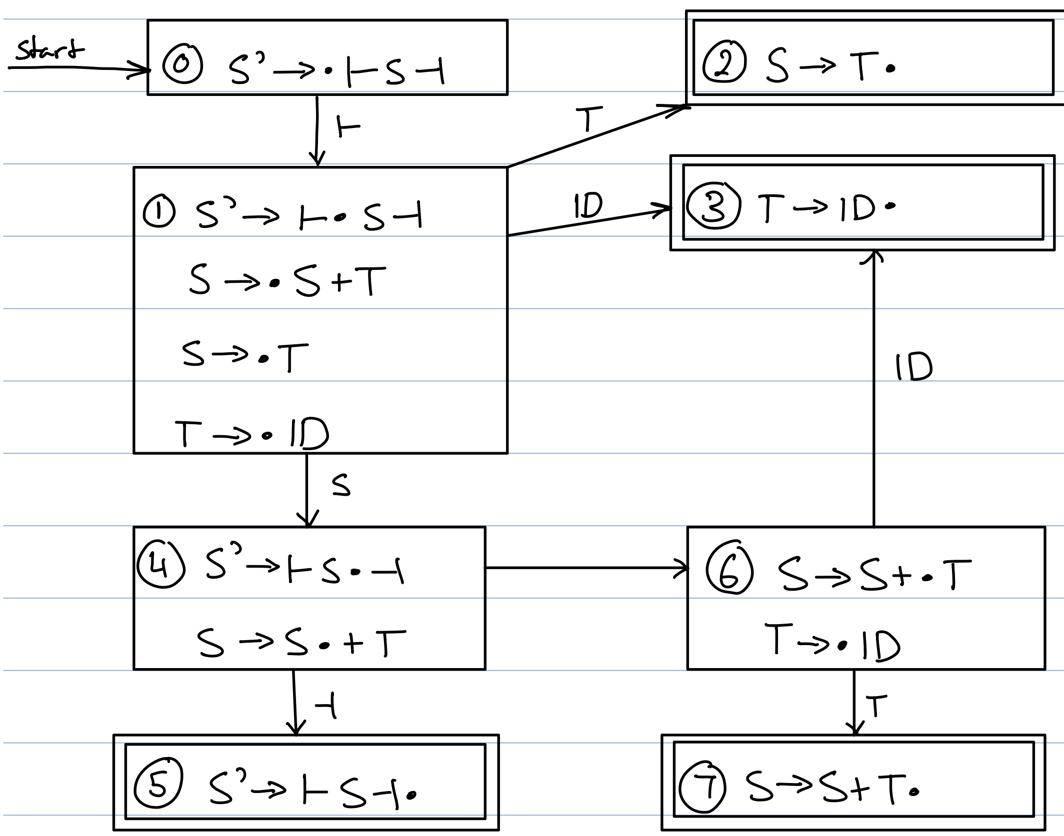
② $S \to S + T$

③ $S \to T$

④ $T \to ID$

Show that $x =$ "+" is $x \notin L(G)$ (ie: input $\vdash + \dashv$ is rejected)

SLR(1) DFA (Also LR(0) ∵ there are no conflicts)



State Stack	Symbol Stack	Input	Action
ε	ε	ε	Initialize
0	ε	$\vdash + \dashv$	Shift $\vdash$, push ①
0 1	$\vdash$	$+ \dashv$	Shift +, error!

↳ ∵ there is no transition from state ① on input "+", we error & "+" $\notin L(G)$

- SLR(1) can fix the limitations of LL(1) & LR(0), but also has a limitation itself.

with grammar:

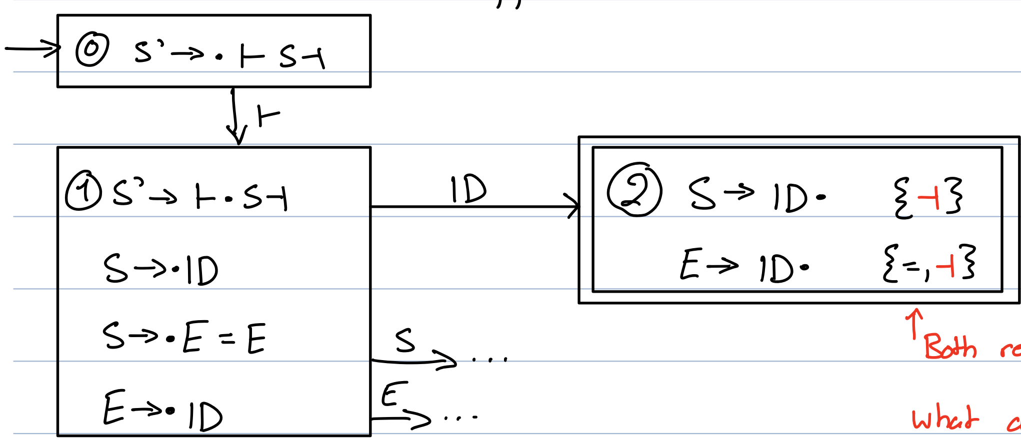
$S' \to \vdash S \dashv$

$S \to ID$

$S \to E = E$ } Follow(S) = $\{\dashv\}$

$E \to ID$ ← Follow(E) = $\{=, \dashv\}$

consider the DFA snipped:



↑ Both reduce on lookahead = ⊣, what do I choose???

- In state ②, the SLR(1)'s simple lookahead using Follow(...) does not work as $\dashv \in$ Follow(S) and $\dashv \in$ Follow(E).
  - ↳ ∵ our SLR(1) dfa has ≥1 reduce-reduce or shift-reduce conflicts, our grammar cannot be SLR(1)
- This can only be fixed by a more adaptive Follow set for reduction conflicts that gurantee no shared elements between sets
  - ↳ LR(1) does this for us!