CS164 Programming Languages and Compilers **Spring 2024**

Discussion Worksheet 4

Recall that to build the LR(1) parsing DFA:

- Every state is a closed set of LR(1) parsing items, formed with the CLOSURE operation
- If S is the start production, The start state is the closure of $[S' \to \bullet S, \$]$, where S' is some dummy nonterminal if S has multiple production rules.
- If a state "State" contains the item $[X \to a \bullet yb, b]$, we add a transition labeled y to a state that contains the items TRANSITION(State, y)

The algorithms are reproduced here for your reference, but you should learn these.

procedure CLOSURE(Items)
  repeat
    for each $[X \to a \bullet Yb, a]$ in Items **do**
      for each production $Y \to g$ **do**
        for each $b \in$FIRST(ba) **do**
          add $[Y \to \bullet g, b]$ to Items
  until Items is unchanged
  return Items

procedure TRANSITION(State, y)
  Items $\leftarrow \emptyset$
  for $[X \to a \bullet yb, b] \in$State **do**
    add $[X \to ay \bullet b, b]$ to Items
  return CLOSURE(Items)

1 LR(1) Conflicts

We are looking at strings of lowercase and uppercase letters. We want to separate the string into individual sequences of lowercase and uppercase letters. Here is a basic CFG to accomplish this task. `lower` and `upper` are tokens for lowercase and uppercase letters individually.

$$S \to \varepsilon \mid S\ L \mid S\ U$$
$$L \to \varepsilon \mid L\ \texttt{lower}$$
$$U \to \varepsilon \mid U\ \texttt{upper}$$

Exercise 1 Recall that a reduce/reduce conflict occurs when a parsing state contains at least two items:

$$[X \to \alpha\bullet, a] \quad [Y \to \beta\bullet, a]$$

1.1 There is a reduce/reduce conflict in this grammar. Identify the smallest string with reduce/reduce conflicts, the rules that can be used to parse it, and the ways the parser can reduce the string.

Solution: The smallest string exhibiting a reduce/reduce conflict is the empty string ε.
This is because the parser can derive ε via $S \to \varepsilon$, or via $S \to SL \to S\varepsilon \to \varepsilon\varepsilon$, or via $S \to SU \to S\varepsilon \to \varepsilon\varepsilon$ (or by infinitely many other derivations!), so that in particular the first reduction the parser can perform can be any of $S \to \varepsilon$, or $L \to \varepsilon$ or $U \to \varepsilon$. (Recall that the LR parser produces a right-most derivation, in reverse, so that the first reduction will be the last production used in any of the derivations given above.)

1.2 Generally, in order to remove reduce/reduce conflicts, we need to remove parsing rules or make them stricter. Change two of the parsing rules in the grammar to remove the reduce/reduce conflict. Solution:

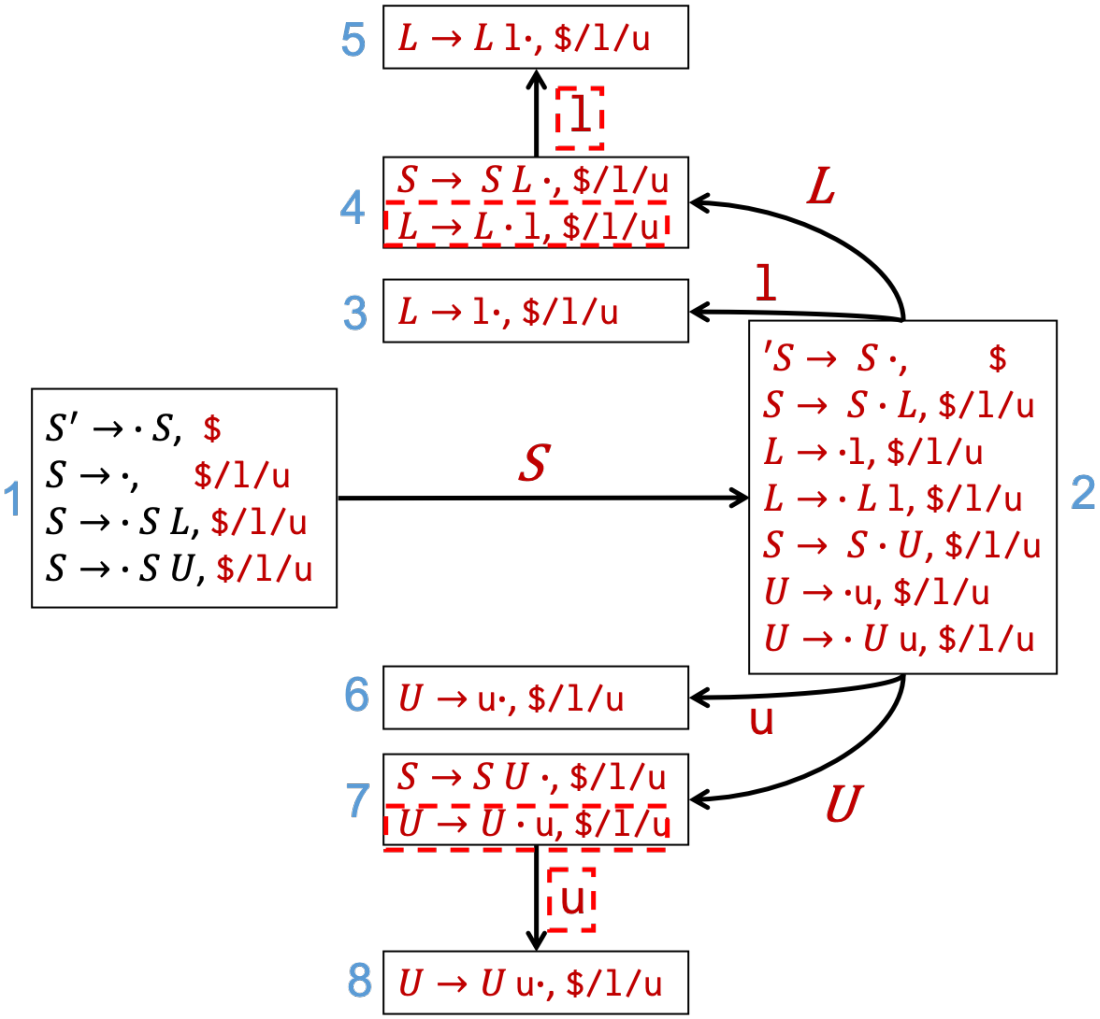
We will make the rules for L and U stricter so they cannot parse ε. Essentially, we are changing the language from $(\texttt{lower}^* \mid \texttt{upper}^*)^*$ to $(\texttt{lower}^+ \mid \texttt{upper}^+)^*$, and avoiding the possibility of deriving arbitrarily many L's and U's to derive an ε.

$$
\begin{aligned}
S &\to \varepsilon && (1)\\
&\mid S\ L && (2)\\
&\mid S\ U && (3)\\
L &\to \texttt{lower} && (4)\\
&\mid L\ \texttt{lower} && (5)\\
U &\to \texttt{upper} && (6)\\
&\mid U\ \texttt{upper} && (7)
\end{aligned}
$$

You can use these labeled production rules in the LR(1) table below.

Exercise 2 Recall that a shift/reduce conflict occurs when a state contains $[X \to \alpha \bullet a\gamma, b], [Y \to \gamma\bullet, a]$.

2.1 There is still a shift/reduce conflict in the grammar from 1.2. In fact, there are two. Draw the DFA for the grammar and identify all the conflicts. (The right box has 7 items.)



2.2 For each conflict, explain what precedence we should implement and why.

Solution:
There are two conflicts, both shift/reduce:

- In state 4, if the next token is `lower` (`l`), then we may either reduce by $S \to S\ L$ (since state 4 contains an item $S \to S\ L\ \bullet, \texttt{\$/l/u}$ with the $\bullet$ at the end of the production $S \to S\ L$), or we may shift to state 5 via the transition. We want to give precedence to the shift action, so that the parser will "maximally munch" a sequence of `lower`s in a row.
- Similarly, in state 7, if the next token is `upper` (`u`), then we may either reduce by $S \to S\ U$ or shift to state 8. For the same reason, maximal munch, we prefer to shift over reduce. This is entirely preference, though, and is not a fundamental decision.

2 LR Parsing

Recall that from the DFA we can create a parse table where:

- a state s with an item $[X \to \alpha\bullet, b]$ reduces with $X \to \alpha$ on b
- a state with transtion $s \to^b s'$ shifts on b

Exercise 3 Create a parse table from the DFA from 2.1, keeping in mind the precedences we chose in 2.2.

	`lower`	`upper`	$	S	L	U
s1	$r_{S\to\varepsilon}$	$r_{S\to\varepsilon}$	$r_{S\to\varepsilon}$	shift to s2		
s2	shift to s3	shift to s6	accept		shift to s4	shift to s7
s3	$r_{L\to\texttt{lower}}$	$r_{L\to\texttt{lower}}$	$r_{L\to\texttt{lower}}$			
s4	shift to s5	$r_{S\to S\ L}$	$r_{S\to S\ L}$			
s5	$r_{L\to L\ \texttt{lower}}$	$r_{L\to L\ \texttt{lower}}$	$r_{L\to L\ \texttt{lower}}$			
s6	$r_{U\to\texttt{upper}}$	$r_{U\to\texttt{upper}}$	$r_{U\to\texttt{upper}}$			
s7	$r_{S\to S\ U}$	shift to s8	$r_{S\to S\ U}$			
s8	$r_{U\to U\ \texttt{upper}}$	$r_{U\to U\ \texttt{upper}}$	$r_{U\to U\ \texttt{upper}}$			

Exercise 4 Is our grammar (with precedence modifications from 2.2) LR(0)? Explain.

Solution: No, in several states there are conditional shift/reduce patterns. Recall that the lookahead in an LR(k) parser is used to determine *whether* to reduce or not, and *which* production to reduce by if it will perform a reduction. An LR(0) parser has no lookahead, so at any state, it must either (1) *always* shift on the next token, or (2) *always* reduce by a fixed rule.
Since there are states in the table which have either (1) two different reduce actions, or (2) a shift and a reduce action, then the grammar is not LR(0).