Programming Languages and Compilers: Quiz #3

Due on March 21, 2024 at 3:10pm

Professor CHUNG YUNG CS Group

LEE CHIH PIN

Problem 1

Transform the following grammar into LL(1) with the name of the techniques applied.

$$
\begin{aligned}
S &\rightarrow \text{Stmt\$} \\
\text{Stmt} &\rightarrow \text{if expr then Stmt else Stmt endif} \\
&\quad \mid \text{if expr then Stmt endif} \\
&\quad \mid \text{other}
\end{aligned}
$$

Solution
To convert the given grammar into an LL(1) grammar, we must ensure that the grammar is unambiguous and that there is no left recursion or common prefixes that might confuse an LL(1) parser. This process typically involves factoring and creating new non-terminals to handle alternatives and recursion appropriately.
Given grammar:

```
S -> Stmt $
Stmt -> if expr then Stmt else Stmt endif
      | if expr then Stmt endif
      | other
```

The problem with the original grammar is that it has a common prefix `if expr then Stmt`. This means that when trying to decide which production to use, we can't do so based on the next token, because up to this point, the two productions are identical. This violates the requirement for an LL(1) grammar that each production must be distinguishable by the next token (lookahead).
Here's how you could rewrite the grammar to be LL(1) compliant:
1. **Eliminate Ambiguity/Common Prefix**: Factor out the common prefix by introducing a new non-terminal for the different suffices.
2. **Eliminate Left Recursion**: If there were left recursion, it would also need to be removed. However, in this case, there is no left recursion.
The transformed grammar would look something like this:

```
S -> Stmt $

Stmt -> if expr then Stmt StmtTail
       | other

StmtTail -> else Stmt endif
          | endif
```

In this transformation, we have introduced a new non-terminal `StmtTail` to handle the ambiguity after the `if expr then Stmt` prefix. `StmtTail` decides whether there's an `else Stmt endif` sequence following or just an `endif`.
Now, an LL(1) parser can look at the next token after `if expr then Stmt` to decide which production of `StmtTail` to use. If the next token is `else`, it uses the first production; if it is `endif`, it uses the second production. This satisfies the LL(1) requirement where each production must be distinguishable by its immediate next token (lookahead of 1).