CS3342 – Assignment 2
due Feb. 17, 2024
2-day no-penalty extension until: Feb. 19, 11:59pm

1. (30pt) Consider a language where assignments can appear in the same context as expressions; the value of `a = b = c` equals the value of `c`. The following grammar, G, generates such expressions that includes assignments in addition to additions and multiplications:

$$
\begin{array}{llcl}
0. & \textit{program} & \longrightarrow & \textit{exp}\ \texttt{\$\$} \\
1. & \textit{exp} & \longrightarrow & \texttt{id}\ \texttt{=}\ \textit{exp} \\
2. & \textit{exp} & \longrightarrow & \textit{term}\ \textit{term\_tail} \\
3. & \textit{term\_tail} & \longrightarrow & \texttt{+}\ \textit{term}\ \textit{term\_tail} \\
4. & \textit{term\_tail} & \longrightarrow & \varepsilon \\
5. & \textit{term} & \longrightarrow & \textit{factor}\ \textit{factor\_tail} \\
6. & \textit{factor\_tail} & \longrightarrow & \texttt{*}\ \textit{factor}\ \textit{factor\_tail} \\
7. & \textit{factor\_tail} & \longrightarrow & \varepsilon \\
8. & \textit{factor} & \longrightarrow & \texttt{(}\ \textit{exp}\ \texttt{)} \\
9. & \textit{factor} & \longrightarrow & \texttt{id}
\end{array}
$$

   (a) (3pt) Show a parse tree for the string: `id = id * (id = id + id * id)$$`.

   (b) (10pt) For each production $A \longrightarrow \alpha$, compute $\textsc{first}(\alpha)$ and $\textsc{follow}(A)$ using the algorithm below; $\textsc{first}(\alpha)$ is computed by `string_FIRST`(α). For each token added, indicate the pair $(\mathit{step}, \mathit{prod})$ used to add it, where $0 \le \mathit{step} \le 3$ is the step in the algorithm (marked as $\boxed{0}$, $\boxed{1}$, $\boxed{2}$, $\boxed{3}$ below) and $0 \le \mathit{prod} \le 9$ is the production involved; indicate $(0, -)$ when step $\boxed{0}$ is used for terminals.

   (c) (5pt) For each production $i, 1 \le i \le 9$, compute $\textsc{predict}(i)$.

   (d) (2pt) Using the information computed above, show that this grammar is not LL(1). (See definition on the slide 19 of the LR-parsing chapter.)

   (e) (10pt) Modify this grammar to make it LL(1). Explain clearly your changes and prove it is LL(1).

–– EPS values and FIRST sets for all symbols:
  for all terminals c, EPS(c) := false; FIRST(c) := {c} $\boxed{0}$
  for all nonterminals X, EPS(X) := if $X \longrightarrow \epsilon$ then true else false; FIRST(X) := $\varnothing$
  repeat
    ⟨outer⟩ for all productions $X \longrightarrow Y_1\ Y_2\ \ldots\ Y_k$,
      ⟨inner⟩ for i in $1\,..\,k$
        $\boxed{1}$ add FIRST(Y_i) to FIRST(X)
        if not EPS(Y_i) (yet) then continue outer loop
      EPS(X) := true
  until no further progress

–– Subroutines for strings, similar to inner loop above:

  function string_EPS($X_1\ X_2\ \ldots\ X_n$)
    for i in $1\,..\,n$
      if not EPS(X_i) then return false
    return true

  function string_FIRST($X_1\ X_2\ \ldots\ X_n$)
    return_value := $\varnothing$
    for i in $1\,..\,n$
      add FIRST(X_i) to return_value
      if not EPS(X_i) then return

–– FOLLOW sets for all symbols:
  for all symbols X, FOLLOW(X) := $\varnothing$
  repeat
    for all productions $A \longrightarrow \alpha\ B\ \beta$,
      $\boxed{2}$ add string_FIRST(β) to FOLLOW(B)
    for all productions $A \longrightarrow \alpha\ B$
        or $A \longrightarrow \alpha\ B\ \beta$, where string_EPS($\beta$) = true,
      $\boxed{3}$ add FOLLOW(A) to FOLLOW(B)
  until no further progress

–– PREDICT sets for all productions:
  for all productions $A \longrightarrow \alpha$
    PREDICT($A \longrightarrow \alpha$) := string_FIRST(α) $\cup$ (if string_EPS(α) then FOLLOW(A) else $\varnothing$)

2. (30pt) Consider Boolean expressions containing operands (`id`), operators (`and`, `or`), and parentheses, where `and` has higher precedence than `or`.

   (a) (10pt) Write an SLR(1) grammar, G, which is not LL(1), for such expressions, which obeys the precedences indicated.

   (b) (5pt) Compute the FIRST(X) and FOLLOW(X) sets for all nonterminals X and PREDICT(i) sets for all productions i.

   (c) (5pt) Prove that G is not LL(1).

   (d) (10pt) Prove that G is SLR(1) by drawing the SLR graph and show there are no conflicts. Build the graph as shown in the examples we did in class (and done by jflap), not the condensed form in the textbook. For each state with potential conflicts (two LR-items, one with the dot in the middle, one with the dot at the end), explain clearly why there is no shift/reduce conflict.

3. (40pt) Consider the C-style `switch` statement.

   (a) (25pt) Write an S-attributed LL(1) grammar that generates C-style `switch` statements and checks that all labels of the arms of the `switch` instructions are distinct. In order to do that, the starting nonterminal, S, will have an attribute `dup` that will store all the duplicate values. There are no duplicate values on the arms of the `switch` statement if and only if S`.dup` $= \emptyset$. Therefore, your grammar is required to eventually compute the attribute `dup` of S.

   For simplicity, assume that the conditional expression of the `switch` statement and the constant expressions labelling the arms are `expr` tokens and that each arm has a statement that is a `stmt` token; the `break` and `default` parts are omitted as their role is irrelevant for our problem. Each `expr` has an attribute `val` provided by the scanner that gives the value of the expression.

   Explain why your grammar works as required. For LL(1), you can use jflap to compute the parse table and show there is no conflict; include the jflap answer (whole window) in your answer.

   (b) (15pt) Using the above attributed grammar, draw a decorated parse tree for the following `switch` instruction:

```
switch ( expr ) {
    case 2 :
    case 3 : stmt
    case 2 : stmt
    case 1 :
    case 2 :
    case 1: stmt
}
```

   Show all attributes and arrows indicated what attributes are used to compute each value.

Notes: **JFLAP:** You are allowed to use JFLAP to help you solve the assignment. You still need to explain clearly your solution. Also, make sure you understand what it does; JFLAP will not be available during exams!

LLMs: You are allowed to use LLMs (Large Language Models), such as ChatGPT, but, again, they will not be available during exams.

LaTeX: For those interested, the best program for scientific writing is LaTeX. It is far superior to all the other programs, it is free, and you can start using it in minutes: https://tobi.oetiker.ch/lshort/lshort.pdf. It is also available online at https://www.overleaf.com/.