Review

Mark all correct answers in each of the following questions.

1. An *integer expression* is (for the purposes of this question) a sequence of integers with the four arithmetic operations between them. Each integer is represented by a non-empty sequence of digits, optionally preceded by a sign. The following are some examples of integer expressions:

   - $123.$
   - $+3 - +5/222 * -7/ - 4.$
   - $+03 - +050/000000222 * -7/ - 04.$
   - $00 - 000/00 * 00 - 00000.$

   The following are not integer expressions:

   - $3 + + + 3.$
   - $2 * (3 + 4).$

   (a) The following constitutes a regular definition of an integer expression:

   $$\begin{array}{lcl} \textit{digit} & \rightarrow & [0-9] \\ \textit{number} & \rightarrow & [+-]?\ \textit{digit}^{+} \\ \textit{operator} & \rightarrow & + \mid - \mid * \mid / \\ \textit{integerExpression} & \rightarrow & \textit{number}\ (\textit{operator}\ \textit{number})^{*} \end{array}$$

   (b) As you have seen above, an integer expression may have a zero after a division sign. The following regular definition describes the

set of all integer expressions with the exception of those containing some representation of zero after a division sign:

$$
\begin{array}{lcl}
\textit{digit} & \rightarrow & [0-9] \\
\textit{nonZeroDigit} & \rightarrow & [1-9] \\
\textit{number} & \rightarrow & [+-]?\ \textit{digit}^{+} \\
\textit{nonZeroNumber} & \rightarrow & [+-]?\ \textit{nonZeroDigit}\ \textit{digit}^{*} \\
\textit{plusMinusTimes} & \rightarrow & + \mid - \mid * \\
\textit{over} & \rightarrow & / \\
\textit{integerExpression} & \rightarrow & \textit{number} \\
 & & ((\textit{plusMinusTimes}\ \textit{number}) \mid \\
 & & (\textit{over}\ \textit{nonZeroNumber}))^{*}
\end{array}
$$

(c) Going back to part (a), we want to add the possibility of exponentiation in integer expressions. Exponentiation may be denoted by either the symbol $\hat{}$ or by $**$. Thus, the following become integer expressions:

- $123 \hat{} 456$.
- $+0 \hat{} 00/0 * *0 * 0$.

We accomplish the change by updating the definition of *operator* as follows:

$$\textit{operator} \rightarrow + \mid - \mid * \mid / \mid \hat{} \mid **$$

The change is correct (whether or not the rest of the regular definition, as given in part (a) is correct). However, the lexical analyzer will have a difficulty with exponents denoted by $**$. It will read the first $*$, and then expect to find another number. When it will read the second $*$, it will give a compilation error.

(d) The grammar defined by the following rules accepts the language of all integer expressions:

$$
\begin{array}{lcl}
\textit{intExp} & \rightarrow & \textit{intExp}\ \textit{expPart} \mid \textit{integer} \\
\textit{expPart} & \rightarrow & \textit{expPart}\ \textit{op}\ \textit{integer} \mid \varepsilon \\
\textit{integer} & \rightarrow & \textit{optionalSign}\ \textit{intPart}\ \textit{digit} \\
\textit{op} & \rightarrow & + \mid - \mid * \mid / \\
\textit{optionalSign} & \rightarrow & + \mid - \mid \varepsilon \\
\textit{intPart} & \rightarrow & \textit{intPart}\ \textit{digit} \mid \varepsilon \\
\textit{digit} & \rightarrow & 0 \mid 1 \mid \ldots \mid 9
\end{array}
$$

(e) The grammar in part (d) (whether or not it accepts the language of integer expressions) is left-recursive. By eliminating all direct left-recursion we obtain a grammar with neither direct nor indirect left-recursion.

(f) Define an integer as *large* if its absolute value is strictly larger than 1024. It is impossible to provide a regular definition of integer expressions consisting only of large integers. However, it is possible define a context-free grammar accepting the language of all such integer expressions.

Solution

1. (a) The definition is correct. Each symbol is defined before being used; *digit* is a digit (character between 0 and 9), *number* is a sequence of one or more *digit*s that may be preceded by either a plus or a minus sign, *operator* is one of the four basic arithmetic operations, and *integerExpression* is a sequence of one or more *number*s, separated by *operator*s.

   (b) Here, *nonZeroNumber* requires the number to start with a non-zero digit. Thus, the string 01, for example, will not be considered a *nonZeroNumber*.

   (c) When reading the input and splitting it into tokens, the lexical analyzer looks for the longest strings that agree with one of the token definitions. Hence, in our case, after reading a $*$, the lexical analyzer distinguishes between the cases where the following character is $+$, $-$, a digit, $*$, or another character. In the first three cases the preceding $*$ will be understood as a full lexeme, and the next character as starting a new *number* token. In the fourth case, the string $**$ will be understood as a full lexeme of type *operator*. In the last case there will be a compilation error.

   (d) One checks easily that *intPart* produces any sequence of digits, *integer* produces any unsigned or signed integer, *expPart* produces

any sequence of *op-integer* pairs. Finally, *intExp* produces an *integer* followed by any sequence of *expPart*s, namely an *integer* followed by *op-integer* pairs, as required.

(e) The recursive non-terminals are *intExp*, *expPart*, and *intPart*. Eliminating direct recursion according to the algorithm shown in class (without taking advantage of obvious shortcuts that are possible in our case), we obtain the grammar:

$$
\begin{array}{lcl}
\mathit{intExp} & \rightarrow & \mathit{integer}\ \mathit{intExp}' \\
\mathit{intExp}' & \rightarrow & \mathit{expPart}\ \mathit{intExp}' \mid \varepsilon \\
\mathit{expPart} & \rightarrow & \mathit{expPart}' \\
\mathit{expPart}' & \rightarrow & \mathit{op}\ \mathit{integer}\ \mathit{expPart}' \mid \varepsilon \\
\mathit{integer} & \rightarrow & \mathit{optionalSign}\ \mathit{intPart}\ \mathit{digit} \\
\mathit{op} & \rightarrow & + \mid - \mid * \mid / \\
\mathit{optionalSign} & \rightarrow & + \mid - \mid \varepsilon \\
\mathit{intPart} & \rightarrow & \mathit{intPart}' \\
\mathit{intPart}' & \rightarrow & \mathit{digit}\ \mathit{intPart}' \mid \varepsilon \\
\mathit{digit} & \rightarrow & 0 \mid 1 \mid \ldots \mid 9
\end{array}
$$

For each of the rules, the rules for the first letter on the right-hand side (if it is a non-terminal) appear only later in our list. Hence we obviously have no indirect left-recursion.

(f) The collection of all unsigned large integers is the complement in $\{0, 1, \ldots, 9\}^*$ of the set $\{0\}^*\{0, 1, \ldots, 1024\}$, and hence regular. A regular definition of large integers is the following:

$$
\begin{array}{lcl}
\mathit{longLarge} & \rightarrow & [1-9]\ \mathit{digit}\ \mathit{digit}\ \mathit{digit}\ \mathit{digit}^+ \\
\mathit{lastThreeDigits} & \rightarrow & [1\text{-}9]\ \mathit{digit}\ \mathit{digit} \mid 0[3\text{-}9]\ \mathit{digit} \mid 02[5\text{-}9] \\
\mathit{lengthFourLarge} & \rightarrow & [2\text{-}9]\ \mathit{digit}\ \mathit{digit}\ \mathit{digit} \mid 1\ \mathit{lastThreeDigits} \\
\mathit{unsignedLarge} & \rightarrow & \mathit{longLarge} \mid \mathit{lengthFourLarge} \\
\mathit{largeNumber} & \rightarrow & [+-]?\ 0^*\ \mathit{unsignedLarge}
\end{array}
$$

Here, the idea is that *longLarge* describes unsigned integers of 5 digits and above; these integers are all large. Other large unsigned integers must be of four digits. Those that start with any digit but 1 are necessarily large, as are those starting with 1 and then any digit but 0, those starting with 10 and then any digit larger than 2, and finally those starting with 102 and then any digit above 4.

All these may be preceded by a sign and any number of leading zeros.

Thus, (a), (d) and (e) are correct.