Midterm I. (22C:123, Spring 2000)

Close Books and Notes (except one sheet of notes)

1. (30) The following Prolog logic grammar has some errors for ordinary algebraic expressions.

```
expr(Z) --> term(X), "+", expr(Y), {Z is X + Y}.
expr(Z) --> term(X), "-", expr(Y), {Z is X - Y}.
expr(Z) --> term(Z).

term(Z) --> number(X), "*", term(Y), {Z is X * Y}.
term(Z) --> number(X), "/", term(Y), {Z is X / Y}.
term(Z) --> number(Z).

number(C) --> "+", number(C).
number(C) --> "-", number(X), {C is -X}.
number(X) --> [C], {"0" =< C, C=<"9", X is C-"0"}.
```

For instance, with the query `expr(X, ‘‘9-2-3’’, [])`, it produces `X = 10` instead of `X = 4`.

(a) Please give the value of X for the query `expr(X, ‘‘8/2/2’’, [])`.
Answer: 8 (or 8.0).

(b) Explain in details why such errors exist.
Answer: Because both `expr` and `term` are defined right recursively, they become left-to-right associative after the parsing, i.e., 8/2/2 is parsed as 8/(2/2) and $9 - 2 - 3$ is parsed as $9 - (2 - 3)$.

(c) Provide a new logic grammar for $+, -, *$, and $/$, which are both syntactically and semantically correct, and will be executed correctly in Prolog.
Answer: We cannot simply make it left-recursive because Prolog will loop.

```
expr(Z) --> term(X), restexpr(X,Z).
restexpr(X,Z) --> ‘‘+’’, term(Y), {W is X + Y}, restexpr(W, Z).
restexpr(X,Z) --> ‘‘-’’, term(Y), {W is X - Y}, restexpr(W, Z).
restexpr(Z,Z) --> [].

term(Z) --> number(X), restterm(X, Z).
restterm(X,Z) --> ‘‘*’’, number(Y), {W is X * Y}, restterm(W,Z).
restterm(X,Z) --> ‘‘/’’, number(Y), {W is X / Y}, restterm(W,Z).
restterm(Z,Z) --> [].

number(C) --> "+", number(C).
number(C) --> "-", number(X), {C is -X}.
number(X) --> [C], {"0" =< C, C=<"9", X is C-"0"}.
```

2. (40) We propose in WREN to accept floating numbers as specified by the following grammars:

```
 <float> ::= <sign> <digit> . <number> E <sign> <number>
<number> ::= <digit> | <digit> <number>
  <sign> ::=   | - | +
 <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

That is, the value of $1.45E3$ is $1.45 \times 10^3 = 1450$ and the value of $-1.45E-2$ is -0.0145.

(a) Please define formally an attribute grammar so that the attribute `value` of $\langle\texttt{float}\rangle$ will give the value of that floating number.

Answer:

We define one attribute *val* (the same as `value`), which returns a real number, for $\langle\texttt{float}\rangle$, $\langle\texttt{number}\rangle$, $\langle\texttt{sign}\rangle$, and $\langle\texttt{digit}\rangle$. We also define another attribute *len*, which returns an integer, for $\langle\texttt{number}\rangle$. Both attributes are synthesized.

The attributed grammar is given below:

$$\begin{array}{rcl}
\langle\texttt{float}\rangle & ::= & \langle\texttt{sign}\rangle\langle\texttt{digit}\rangle_1.\langle\texttt{number}\rangle E\langle\texttt{sign}\rangle_2\langle\texttt{number}\rangle_2 \\
 & & val(\langle\texttt{float}\rangle) \leftarrow val(\langle\texttt{sign}\rangle) * (val(\langle\texttt{digit}\rangle) + val(\langle\texttt{number}\rangle) * 10^{-len(\langle\texttt{number}\rangle)}) * \\
 & & \quad 10^{val(\langle\texttt{sign}\rangle_2)*val(\langle\texttt{number}\rangle_2)} \\
\langle\texttt{number}\rangle & ::= & \langle\texttt{digit}\rangle \\
 & & val(\langle\texttt{number}\rangle) \leftarrow val(\langle\texttt{digit}\rangle) \\
 & & len(\langle\texttt{number}\rangle) \leftarrow 1 \\
 & | & \langle\texttt{digit}\rangle\langle\texttt{number}\rangle_2 \\
 & & val(\langle\texttt{number}\rangle) \leftarrow val(\langle\texttt{digit}\rangle) * 10^{len(\langle\texttt{number}\rangle_2)} \\
 & & len(\langle\texttt{number}\rangle) \leftarrow len(\langle\texttt{number}\rangle_2) + 1 \\
\langle\texttt{sign}\rangle & ::= & \epsilon \mid + \\
 & & val(\langle\texttt{sign}\rangle) \leftarrow 1 \\
 & & - \\
 & & val(\langle\texttt{sign}\rangle) \leftarrow -1 \\
\langle\texttt{digit}\rangle & ::= & 0 \\
 & & val(\langle\texttt{digit}\rangle) \leftarrow 0 \\
 & & \ldots \\
 & | & 9 \\
 & & val(\langle\texttt{digit}\rangle) \leftarrow 9
\end{array}$$

(b) Provide a Prolog logic grammar which will implement the attribute grammar correctly.

Answer:

```
float(V) --> sign(S), digit(D), ‘‘.’’, num(N, L), ‘‘E’’, digit(D2),
               num(N2,_), { V is S*(D+N*exp(10,-L))*exp(10,D2*N2) }.
num(N,L) --> digit(D), num(M,L2), { N is D*exp(10,L2)+M, L is L2+1 }.
num(N,1) --> digit(N).
```

```
sign(1)  --> ``+''.
sign(-1) --> ``-''.
sign(1)  --> [].
digit(0) --> ``0''.
 ...
digit(9) --> ``9''.
```

3. (30) Suppose that we define a list of lambda expressions also as a lambda expression in the lambda calculus as follows, where a list of expressions is denoted, by $[E_1, E_2, ..., E_n]$:

$$E ::= \ v \mid c \mid (E\ E) \mid (\lambda v\ .\ E) \mid [E, ..., E]$$

where v denotes variables and c denotes constants.

The composition rule involving lists is defined as follows:

$$\begin{array}{ll} [E_1, E_2, ..., E_n]\ E = [E_1\ E, E_2\ E, ..., E_n\ E] & (1) \\ E\ [E_1, E_2, ..., E_n] = [E\ E_1, E\ E_2, ..., E\ E_n] & (2) \end{array}$$

Rule (1) has higher priority than rule (2). As usual, the composition rules have higher priority than the abstraction rule.

In the following exercise, we assume that the predefined constants are the numbers and *mul* (multiplication). We also have a shorthand *Twice*:

$$Twice \stackrel{\text{def}}{=} (\lambda f\ .\ \lambda x\ .\ f(f\ x))$$

Please provide every step of reducing the following lambda expression

$$Twice\ (\lambda n\ .\ [(mul\ 2\ n), (mul\ 3\ n)])\ 5,$$

using both the normal order reduction and the applicative order reduction, respectively.

Let

$$A \stackrel{\text{def}}{=} (\lambda n\ .\ [(mul\ 2\ n), (mul\ 3\ n)]).$$

Normal order reduction:

$$\begin{array}{ll}
 & Twice\ (\lambda n\ .\ [(mul\ 2\ n), (mul\ 3\ n)])\ 5 \\
= & (\lambda f\ .\ \lambda x\ .\ f(f\ x))\ \underline{A}\ 5 \\
\rightarrow_\beta & (\lambda x\ .\ A(A\ x))\ \underline{5} \\
\rightarrow_\beta & A(A\ 5) \\
= & (\lambda n\ .\ [(mul\ 2\ n), (mul\ 3\ n)])\underline{(A\ 5)} \\
\rightarrow_\beta & [(mul\ 2\ (A\ 5)), (mul\ 3\ (A\ 5))]) \\
= & [(mul\ 2\ ((\lambda n\ .\ [(mul\ 2\ n), (mul\ 3\ n)])\ \underline{5})), (mul\ 3\ (A\ 5))]) \\
\rightarrow_\beta & [(mul\ 2\ ([(mul\ 2\ 5), (mul\ 3\ 5)])), (mul\ 3\ (A\ 5))] \\
\rightarrow & [[(mul\ 2\ (mul\ 2\ 5)), (mul\ 2\ (mul\ 3\ 5))], (mul\ 3\ (A\ 5))] \\
\rightarrow_\delta^* & [[20, 30], (mul\ 3\ (A\ \underline{5}))] \\
\rightarrow_\beta & [[20, 30], (mul\ 3\ [(mul\ 2\ 5), (mul\ 3\ 5)])] \\
\rightarrow^* & [[20, 30], [30, 45]]
\end{array}$$

Applicative order reduction:

$$\begin{array}{ll}
 & Twice\ (\lambda n\ .\ [(mul\ 2\ n), (mul\ 3\ n)])\ 5 \\
= & (\lambda f\ .\ \lambda x\ .\ f(f\ x))\ \underline{A}\ 5 \\
\rightarrow_\beta & (\lambda x\ .\ A(A\ x))\ 5 \\
= & (\lambda x\ .\ A((\lambda n\ .\ [(mul\ 2\ n), (mul\ 3\ n)])\ \underline{x}))\ 5 \\
\rightarrow_\beta & (\lambda x\ .\ A\ \underline{[(mul\ 2\ x), (mul\ 3\ x)]})\ 5 \\
\rightarrow_\beta & (\lambda x\ .\ [(mul\ 2\ [(mul\ 2\ x), (mul\ 3\ x)]), (mul\ 3\ [(mul\ 2\ x), (mul\ 3\ x)])])\ 5 \\
\rightarrow^* & (\lambda x\ .\ [[(mul\ 2\ (mul\ 2\ x)), (mul\ 2\ (mul\ 3\ x))], [(mul\ 3\ (mul\ 2\ x)), (mul\ 3\ (mul\ 3\ x))]])\ \underline{5} \\
\rightarrow_\beta & [[(mul\ 2\ (mul\ 2\ 5)), (mul\ 2\ (mul\ 3\ 5))], [(mul\ 3\ (mul\ 2\ 5)), (mul\ 3\ (mul\ 3\ 5))]] \\
\rightarrow_\delta^* & [[20, 30], [30, 45]]
\end{array}$$