Solution Set 8

DATE: 1 DECEMBER, 1992

1. Prove the following functions are primitive recursive,

   (a) $x < y$. The functions $sgn(n)$ and $monus(x, y)$ are primitive recursive. Hence by composition, so is

   $$< (x, y) = sgn(monus(U_2^2(x, y), U_1^2(x, y))).$$

   (b) x^y. The function $times(x, y)$ is primitive recursive. Hence by primitive recursion, so is,

   $$power(x, y) = \begin{cases} succ(Z(y)) & x = 0 \\ h(power(x - 1, y), x - 1, y) & x > 0 \end{cases}$$

   where,

   $$h(x, y, z) = times(U_1^3(x, y, z), U_3^3(x, y, z)).$$

   (c) $|\, x - y \,|$. Addition and monus are primitive recursive. Hence by composition so is

   $$|\, x - y \,| = add(monus(x, y), monus(U_2^2(x, y), U_1^2(x, y))).$$

2. Show that the predicates $=, \neq$ and $\leq$ are primitive recursive. We can use Corollary 6: A booleans combination of primitive recursive predicates is primitive recursive. The three predicates can be written as boolean combinations of $< (x, y)$ and $< (U_2^2(x, y), U_1^2(x, y))$.

3. Discuss the ambiguity of the grammar,

   $$\begin{aligned} S &\rightarrow S_1 \,|\, S_2 \\ S_1 &\rightarrow \textbf{ if } C \textbf{ then } S_1 \textbf{ else } S_2 \,|\, a \,|\, b \\ S_2 &\rightarrow \textbf{ if } C \textbf{ then } S \,|\, \textbf{ if } C \textbf{ then } S_1 \textbf{ else } S_2 \,|\, a \,|\, b \\ C &\rightarrow p|q \end{aligned}$$

   This is Example 12, page 232 of *A Programming Approach to Computability.*

The grammar *is* ambiguous. The statement,

$$\textbf{if } p \textbf{ then } \textbf{ if } p \textbf{ then } a \textbf{ else } \textbf{ if } p \textbf{ then } a \textbf{ else } b$$

can be parsed in two ways, giving rise to two distinct parse trees. I can't get my paint program to run, so instead of a nice tree graphic, I will have to use parentheses notation. It should be clear what the parentheses are trying to say. Here is the first parse tree:

$$\begin{aligned} S &\rightarrow \textbf{if } C \textbf{ then } S_1 \textbf{ else } S_2 \\ &\rightarrow \textbf{if } C \textbf{ then } (\ \textbf{if } C \textbf{ then } S_1 \textbf{ else } S_2) \textbf{ else } S_2 \\ &\rightarrow \textbf{if } C \textbf{ then } (\ \textbf{if } C \textbf{ then } S_1 \textbf{ else } (\ \textbf{if } C \textbf{ then } S)) \textbf{ else } S_2 \\ &\rightarrow \textbf{if } p \textbf{ then } (\ \textbf{if } p \textbf{ then } a \textbf{ else } (\ \textbf{if } p \textbf{ then } a)) \textbf{ else } b, \end{aligned}$$

and here, the second:

$$\begin{aligned} S &\rightarrow \textbf{if } C \textbf{ then } S \\ &\rightarrow \textbf{if } C \textbf{ then } (\ \textbf{if } C \textbf{ then } S_1 \textbf{ else } S_2) \\ &\rightarrow \textbf{if } C \textbf{ then } (\ \textbf{if } C \textbf{ then } S_1 \textbf{ else } (\ \textbf{if } C \textbf{ then } S_1 \textbf{ else } S_2)) \\ &\rightarrow \textbf{if } p \textbf{ then } (\ \textbf{if } p \textbf{ then } a \textbf{ else } (\ \textbf{if } p \textbf{ then } a \textbf{ else } b)), \end{aligned}$$

We present an unambiguous grammar for if-then-else.

$$\begin{aligned} S^* &\rightarrow \textbf{if } C \textbf{ then } S^* \,|\, \textbf{if } C \textbf{ then } S_3 \textbf{ else } S^* \,|\, a \,|\, b \\ S_3 &\rightarrow \textbf{if } C \textbf{ then } S_3 \textbf{ else } S_3 \,|\, a \,|\, b \\ C &\rightarrow p \,|\, q \end{aligned}$$

We prove that this grammar is unambiguous and that it produces all correct if-then-else strings. In fact, the grammar associates "then" and "else" clauses according the the standard Pascal rule. To simplify things somewhat, we will replace **if** C **then** by the single symbol **ifen** . Given a ifen-else string, we can label each "ifen" and "else"

with an integer that indicates how they are to group. Let a counter begin at 0 and scan the string left to right. At each "ifen", assign the counter value to the "ifen" and increment the counter by one. At each "else", decrement the counter by one and assign the resulting value to the "else". Note that an "else" associates with the first "ifen" to its left which carries the same label. To illustrate, we prove a useful Lemma.

Lemma 1 *The S_3 production is unambiguous and produces any correct ifen-else string provided that it has no more "ifen's" than "else's".*

PROOF: As the labeling proceeds from left to right, the counter is always set to the excess number of "then's" over "else's" scanned so far. The first return to zero of the counter will occur with the "else" given by the first S_3 production. Therefore, this "else" is unambiguously tied to the first "ifen". We now argue by induction on the number of "else's" appearing in the string that any string resulting from S_3 is unambiguous.

If no "else" appears, the parse tree is simply $S_3 \to a \,|\, b$. If there is some "else" which appears, the labeling indicates which "else" divides the left subtree from the right subtree. Each subtree is a string generated from S_3 with at least one less "else". By induction, each of this substrings gives a unique parse tree. Therefore, the entire parse tree is unique.

Any correctly generated ifen-else string with as many "ifen's" as "else's" is either a, b or has an "else" labeled with a zero. Taking the "ifen" part of the string associated with the "else" of label zero, it has as many "ifen's" as "else's" (that is why the "else" has label zero). Therefore, the "else" part also has as many "ifen's" as "else's". By induction on the number of "else's" in the strings, we can conclude that the string can be generated by S_3. □

Theorem 1 *The S^* production is unambiguous and produces any correct ifen-else string.*

PROOF: We prove this by induction on the number of "else's" in the string. If "else" does not appear, the production is obviously unambiguous. Suppose unambiguous any string with less than k "else's".

Given a string with k "else's", apply the labeling and choose the leftmost "else" from among the "else's" with smallest label, call it "else´". Suppose that the label of "else´" is i. Then the string is the result of i productions of the form $S^* \to$ **ifen** S^* followed by

$$S^* \to \textbf{ifen}\ S_3\ \text{else}'\ S^*$$

The string binding to S_3 has as many "ifen's" as "else's" and so, by the Lemma, it is unambiguously generated. The string binding to S^* has fewer than k "else's" and so by induction it is unambiguously generated. Therefore, the entire string is unambiguously generated.

We argue that any valid if-then-else string results from S^*. Any valid if-then-else string can be assigned a labeling and the counter never goes negative. Therefore, if an "else" appears, we can choose the leftmost "else" among those of smallest label, this label being zero or larger. Call it "else´". Associating "else´" with the unique "ifen" to its left of equal label, the number of intervening 'ifen's" and "else's" are equal. So the Lemma provides a production for the intervening string starting from S_3. To the left of the "ifen" associate to "else´" there are no "else's". So repeated productions of the form $S^* \to$ **ifen** S^* complete the consturction of the string to the left of "else´". Arguing by induction on the number of "else's", the string to the right of "else´" can be generated starting from S^*. □