CS 5000 Homework 8

Sally Devitry (A01980316)

Problem 1

You may have heard of the TIMTOWTDI (pronounced by some as "Tim Toady") principle of software engineering - There Is More Than One Way To Do It. Many software engineers and systems architects use it in their daily work. This problem offers a mathematical justification, after a fashion, of this principle. Let f(x1,...,xn) be a partially computable function. Show that f is computed by infinitely many L-programs whose length l ≤ k, for some natural number k ≥ 0.

Solution

A partially computable function f can be computed by infinitely many L-programs because a program can be any length and have any number of instructions because there is an infinite number of ways the instructions can be arranged. No-ops can be added infinitely, with each that is added giving a unique L-program. The same thing can be done by cycling between the increment and decrement operations in L.

Problem 2

An L-program is a straightline if it contains no instructions, labeled or unlabeled, of the form IF V != 0 GOTO L, for some label L. Show that if P is a straightline L-program of length k, for some natural number k, then $\psi_p^{(1)}(x) \leq k$.

Solution

If P is a straightline L-program of length k, then $\psi_p^{(1)}(x) \leq k$ because the value of the variable in the program cannot be greater than the number of lines. Only one increment operation can be done per line and thus the value of the variable cannot be incremented more then k times, meaning its value will always be less than or equal to k.
Take the following example program of length k=3, with the variable Y.

1 $Y \leftarrow 1$
2 $Y \leftarrow 1$
3 $Y \leftarrow 1$

This program gives the largest value of Y possible without using any GOTOs in 3 lines, and the value of the variable will always be less than or equal to k, the number of lines.

Problem 3

Here's a problem from the formal theory of compilation. Let L++ be a programming language that extends the programming language L by adding one instruction V ← k, where k ∈ N. Show that a function is partially computable in L++ if and only if it is partially computable.

Solution

To prove that a function is partially computable in L++ if and only if it is partially computable, we must prove two cases.
1. A function can be computed by an L++ program if it can be computed by an L program.
2. A function can be computed by an L program if it can be computed by an L++ program.
Obviously, because all of the L language is included in L++, we can say that a function computed by an L program can also be computed by an L++ program (1).
To show that a function in L++ will be partially computable we can note that the assignment operator that is added to the L++ language, V ← k, can really be thought of as a combination of the more simple operations in L. For example, if Y = 0, to assign $Y \leftarrow 3$, we need only increment Y three times. Since the newly added instruction of variable assignment is really just several increments or decrements in a row, a function in L++ can be computed in L. Thus, a function is partially computable in L++ if and only if it is partially computable(2).

Problem 4

Let f(x) be a function of one argument. Let the n-th iteration of f be defined as
$f_n(x) = f(...f(x)...).(1)$
For example, $f_0(x) = f(x) = x, f_1(x) = f(x), f_2(x) = f(f(x)), f_3(x) = f(f(f(x)))$, etc.
Let $i_f(n, x) = f_n(x)$.
Show that if f is primitive recursive, then so is $i_f(n, x)$. Assuming that f is primitive recursive, is if computable? If yes, explain why; if not, explain why not.

Solution

The function f(x) is recursive as it calls itself. It is primitive recursive as it can be obtained from the initial functions (successor, null constant, and projection) by a finite number of applications of composition.
We also proved in class that the class of primitive recursive functions is primitively recursively closed. Thus, if f is primitive recursive, then $i_f(n, x) = f_n(x)$ is primitive recursive as well because composition is a primitive recursive operation.

Yes, it is computable because a program can be written for this function in L. L can perform recursion by using GOTO operations that jump to their own label. We also proved in class, Corallary 3.4, that every

primitive recursive function is computable by assuming that a function is primitive recursive and using previously proved Theroem 3.3. and 3.1 to show that f is in the class of computable functions.

Problem 5

Let COMP be the class of functions obtained from the initial functions by a finite sequence of compositions (no primitive recusions!).
1. Show that for every function $f(x_1, ..., x_n) \in COMP$, either $f(x_1, ..., x_n) = k$, for some constant k, or $f(x_1, ..., x_n) = x_i + k$, for $1 \leq i \leq n$ and some constant k.
2. An n-ary function f is monotone if for all n-tuples $(x_1, ..., x_n), (y_1, ..., y_n)$ such that $x_i \leq y_i, 1 \leq i \leq n, f(x_1, ..., x_n) \leq f(y_1, ..., y_n)$.
Show that every function in COMP is monotone.
3. Is COMP a proper subset of the class of primitive recursive functions?
4. Is COMP a proper subset of the class of computable functions?

Solution

The initial functions are
1. The successor function : $s(x) = x + 1$
2. The null function : $n(x) = 0$
3. The projection function : $u_i^n(x_1, ...x_n) = x_i, 1 \leq n$

1. Only the following operations can happen to a variable, say Y, if it is in COMP.
a. $Y \leftarrow 0$
b. $Y \leftarrow x_i$
c. $Y \leftarrow Y + 1$

Let k = any number of the c. operation above.
Now because of composition, the function could take the form of the following.
$f(x) = 0 + k$,
$f(x) = x_i + k$
Renaming x, to a for clarity, the following is true of f.
$f(a) = k$
$f(a) = a + k$
$f(f(a)) = k$
$f(f(a)) = a + k + k$

The $k + k$ above will still always just result in a constant eventually, so we can call $k + k$, k. As we can see, for every function $f(x_1, ...x_n) \in$COMP, either $f(x_1, ...x_n) = k$ or $f(x_1, ...x_n) = x_i + k$

2. From part 1, we know that for every function $f(x_1, ...x_n) \in$COMP, either $f(x_1, ...x_n) = k$ or $f(x_1, ...x_n) = x_i + k$
Because of the above statement, $(f(a) = k)$, we know that
$f(x_1, ...x_n) = f(y_1, ...y_n)$.

And because of the above statement, $(f(a) = a + k)$, we know that
$f(x_1, ...x_n) < f(y_1, ...y_n)$.

This means that $f(x_1, ...x_n) \leq f(y_1, ...y_n)$. All the functions in COMP are monotone.

3. Yes, COMP is a proper subset of the class of primitive recursive functions. All of the initial functions are by definition primitive recursive and the class of primitive recursive functions is closed under composition(Corallary 3.2).

4. Yes, COMP is a proper subset of the class of computable functions. If all functions in COMP are primitive recursive (3.), then all the functions in COMP will be computable. Corallary 3.4 from the lecture slides proves that all primitive recursive functions are computable.