
HW 11 – Proving Properties of Recursive Functions

CS 421 – Spring 2008

Revision 1.0

Assigned April 17, 2008

Due Thursday, April 24, 2008, submit in class

Extension due April 24, 2008, 5:00pm, 20% penalty

Guideline:

- Write your name at the top of this page. Use empty space below each problem for your solution. If you need more space, you're allowed to attach additional papers.
- If you're submitting late, hand in your homework either to Prof. Kamin in room 4237 or to Dongyun Jin in room 2111, by 5:00pm on April 24, Thursday. Note that 20% penalty applies.
- Submit hard-copy only. No electronic submission.

Problem 1. (12pts) Given definition $f(x, y) = \text{if } p(x) \text{ then } y \text{ else } h(f(k(x), y))$ and corresponding $F = \lambda f. \lambda(x, y). \text{if } p(x) \text{ then } y \text{ else } h(f(k(x), y))$, prove $\forall x, y. h(f(x, y)) = f(x, h(y))$. Use *computation induction*, with admissible predicate ϕ :

$$\phi(g) : \forall x, y. h(g(x, y)) = g(x, h(y))$$

Recall that you have to first prove $\phi(\Omega)$ (base case) and then $\forall g. \phi(g) \rightarrow \phi(F(g))$ (induction step).

(Continue Solution of Problem 1)

Problem 2. (12pts) The principle of *recursion induction* is a method of proving two recursive definitions equivalent. (This was perhaps the first method ever published for proving properties of recursive functions; John McCarthy — the creator of Lisp — published it in 1963.)

Principle of recursion induction: Two functions f_1 and f_2 (however defined — not necessarily recursively) can be proven equal if there is a recursive definition $f(x) = \dots$ and corresponding $F = \lambda f. \lambda x. \dots$ such that: (1) $f_1 = F(f_1)$, (2) $f_2 = F(f_2)$, and (3) f can be shown to be total (i.e. $f(x) \neq \omega$, for all x).

Use recursion induction to prove, for all x, y , $\text{reverse}(x @ y) = \text{reverse}(y) @ \text{reverse}(x)$, where reverse has the definition:

```
reverse x = if x=[] then [] else reverse(tl x) @ [hd x]
```

For this proof, define

```
f1 = λx, y. reverse(x @ y)
f2 = λx, y. reverse(y) @ reverse(x)
f = λx, y. if [] then reverse(y) else f(tl x, y) @ [hd x]
F = λf. λx, y. if [] then reverse(y) else f(tl x, y) @ [hd x]
```

Then, you need to prove: (1) $f_1 = F(f_1)$, (2) $f_2 = F(f_2)$, and (3) f is total. As usual, part (3) should be proven by structural induction on x .

For this proof, you may assume (without proof) that reverse is total, and you may also assume these facts about lists: for non-empty x , and for any y , $\text{tail}(x)@y = \text{tail}(x@y)$ and $\text{hd}(x) = \text{hd}(x@y)$.

(Continue Solution of Problem 2)

Problem 3. (12pts) Given definitions

$$f_1(x) = \text{if } x=0 \text{ then } 1 \text{ else } x * f_1(x-1)$$

$$f_2(x, y) = \text{if } x=y \text{ then } 1 \text{ else } f_2(x, y+1) * (y+1)$$

prove: for all x , $f_2(x, 0) = f_1(x)$.

This can be proven by arithmetic induction on x , using the strengthened induction hypothesis:

$$\phi(x) : \forall y. f_2(x + y, y) f_1(y) = f_1(x + y)$$

From this, we can obtain, as a particular instance, $f_2(x + 0, 0) f_1(0) = f_1(x + 0)$, which, since $f_1(0) = 1$, gives the result.

Problem 4. (12pts) The function `expand` was a homework exercise earlier in the semester. Its argument is of type $(int * string) list$. It replaces each pair (i, s) with i copies of s . Here is one definition:

```
let rec expand x = match x with
  [] -> []
  | (0,s)::xs -> expand xs
  | (i,s)::xs -> s :: expand( (i-1,s)::xs )
```

Prove that `expand` is a total function on lists of the correct type. The problem is to define a *well-founded* ordering \prec on lists such that the arguments passed to recursive calls are always smaller than the arguments to the current call. In particular, for any s, xs , and i , $xs \prec (0,s)::xs$ and $(i-1,s)::xs \prec (i,s)::xs$.