Problem1:

I have submitted my “lookup.sml”.

It support the lookup of the tree and exception.

Problem2:

Consider the **datatype** ‘a gametree = node **of** ‘a \* ‘a gamtree **list** discussed in class.

Assuming that the strength assessment function for a game position is of the form

**fun** assess = minimax ◦ treemap(strength) ◦ prune(5) ◦ game

where

assess: position  int

minimax: int gametree  int

strength: position gametree  int

treemap: (position gametree  int)  position gametree  int gametree

prune: int  position gametree  position gametree

game: position  position gametree

Note that ‘position’ is the type for a typical position (or configuration) of a game. Define treemap

and prune using ML notation. Refer to the paper, “Why Functional Programming Matters,” by

John Hughes, available from the web, for guidance on how to write your solution.

Answer:

fun prune(0,a@x)= a@[]

| prune(n,a@x)=a @ map(prune(n-1))x;

Fun treemap(m:int,p:position gametree)=treemap(0,t)=f 0:t

|treemap(n,t)=f n:t

Problem 3:

[30%] Consider the following definition in a lazy functional language:

1.fun primes = sieveall(numsfrom(2));

2.fun sieveall(p::t) = p :: sieveall(sieve(t,p));

3.fun sieve(h::t, p) = if h mod p = 0 then sieve(t,p) else h :: sieve(t,p);

4.fun numsfrom(n) = n :: numsfrom(n+1);

Translate the above code into a non-lazy functional language with higher-order functions. The

translation should use the approach that was illustrated in class for numsfrom.

Answer:

1.Translated as

fun numsfrom(n) = let fun thk1() = n

fun thk2() = numsfrom(n+1)

in (thk1, thk2)

end;

2.Translated as

fun sieveall(p::t) =let fun thk1() =p

fun thk2()=t

in

thk1()::sieveall(sieve(thk2(),thk1()));

3.Tranlsated as

fun sieve(h::t,p) = let fun thk1()=h

fun thk2()=t

fun thk3() = p

in

if thk1() mod thk3() =0 then sieve(thk2(),thk3()) else thk1()::sieve(thk2(),thk3());