CS442 Assignment 4

Nissan Pow
20187246
npow

March 19, 2007

Part A

```
val callcc=SMLofNJ.Cont.callcc
val throw = SMLofNJ.Cont.throw

(* ('a -> 'b -> 'b ?.cont -> 'b) -> 'b -> 'a list -> 'b *)
fun scfoldl f i [] = i
|   scfoldl f i (x::xs) = (callcc (fn k => (scfoldl f (f x i k) xs)))

fun exists p L =
let
  fun f x i k =
    if p x then throw k true else false
in
  scfoldl f false L
end

fun even num = num mod 2 = 0;

exists even [1,3,5,7,9,2,3,5,7,9];
```

Parts B-D

```
% handle promises
interpret([a, d | T], [promise(P)|R]) :- !,
  getPromise(T,P,R).

% Recursively reduce the tail of the list until it is no longer reducible
% and then do the same with the tail of the tail of the list. At this point
% it is safe to call r1 to perform the application
interpret([a | X], RESULT) :- !,
  interpret(X, [R|RS]),
  interpret(RS, RR),
  r1([a, R | RR], RESULT).
```

```
% Terminate if head is anything other than a
interpret(X, X) :- !.

% Rules for applying primitive operators
r1([a, k, X | T], [k(X) | T]) :- !.
r1([a, k(X), _ | T], [X | T]) :- !.
r1([a, s, X | T], [s(X) | T]) :- !.
r1([a, s(X), Y | T], [s(X,Y) | T]) :- !.
r1([a, s(X,Y), Z | T], R) :- ! , interpret([a, a, X, Z, a, Y, Z | T], R).
r1([a, i, X | T], [X | T]) :- !.
r1([a, v, _ | T], [v | T]) :- !.
r1([a, dot(X) | T], T) :- ! , put(X).
r1([a, r | T], T) :- ! , nl.
r1([a, promise(P) | T], R) :- !,
  append(P,T,PT),
  interpret([a | PT],R).

% returns the promise as P, and the rest of the expression in R
getPromise([a | X], [a|PPPP], RRS) :- !,
  getPromise(X, PP, RS),
  getPromise(RS, PPP, RRS),
  append(PP, PPP, PPPP).

getPromise([X|T], [X], T) :- !.

interpretFromText(S, R) :- !,
  convert(S, SS),
  interpret(SS, RR),
  remove_closure(RR,RRR),
  convert(RRRR,RRR),
  string_to_list(R, RRRR).

% convert(X,Y) converts a list X of numbers to their character equivalent, with the result sto
convert([],[]) :- !.
convert([46,X|T], [dot(X)|R]) :- ! , convert(T,R).
convert([96|T], [a|R]) :- ! , convert(T,R).
convert([97|T], [a|R]) :- ! , convert(T,R).
convert([100|T], [d|R]) :- ! , convert(T,R).
convert([105|T], [i|R]) :- ! , convert(T,R).
convert([107|T], [k|R]) :- ! , convert(T,R).
convert([114|T], [r|R]) :- ! , convert(T,R).
convert([115|T], [s|R]) :- ! , convert(T,R).
convert([118|T], [v|R]) :- ! , convert(T,R).
convert([X|T], [X|R]) :- ! , convert(T,R).

remove_closure([],[]) :- !.
```

```
remove_closure([s(A,B)|T],R) :- !,
  remove_closure([A],AA),
  remove_closure([B],BB),
  remove_closure(T,TT),
  append(AA,BB,AB),
  append(AB,TT,ABT),
  append("‘‘s",ABT,R).

remove_closure([s(A)|T],R) :- !,
  remove_closure([A],AA),
  remove_closure(T,TT),
  append(AA,TT,AATT),
  append("‘s",AATT,R).

remove_closure([k(A)|T],R) :- !,
  remove_closure([A],AA),
  remove_closure(T,TT),
  append(AA,TT,AT),
  append("‘k",AT,R).

remove_closure(A,A) :- !.

% convert lambda expression X into an unlambda expression R
unlambdafy(X,R) :-
  u2(X,R2),
  c2(R2,R3),
  convert(R3,R4),
  string_to_list(R,R4).

% this does all the unlambdafication
u2(var(X),[X]) :- !.
u2(func(d),[d]) :- !.
u2(func(i),[i]) :- !.
u2(func(r),[r]) :- !.
u2(func(v),[v]) :- !.
u2(func(dot(X)),[.,X]) :- !.
u2(app(M,N),R) :- !,
  u2(M,M2),
  c2(M2,M3),
  convert(M3,M4),
  u2(N,N2),
  c2(N2,N3),
  convert(N3,N4),
  append([a|M4], N4, R).
u2(abs(var(X),E),R) :- !,
  u2(E,R2),
```

```
  c2(R2,R3),
  convert(R3,R4),
  remove_X(X,R4,R).

remove_X(_,[],[]) :- !.

% [x](M,N) = S ([x]M) ([x]N)
remove_X(X,[a|T],ess(R1,R2)) :- !,
  getPromise(T,M,N),
  c2(M,M2),
  convert(M2,M3),
  c2(N,N2),
  convert(N2,N3),
  remove_X(X,M3,R1),
  remove_X(X,N3,R2).

% [x]y = I    if x=y
% [x]y = K y  otherwise
% [x]f = K f
% [x]I = K I
% [x]K = K K
% [x]S = K S
remove_X(X,[X],"i") :- !.
remove_X(_,[A],kay([A])) :- !.

reduce(L,R) :-
  unlambdafy(L,UL),
  convert(UL,RR),
  interpret(RR,R), !.

reduce(L,R) :-
  unlambdafy(L,UL),
  string_to_list(UL,LL),
  convert(LL,RR),
  interpret(RR,R), !.

reduce(L,R) :-
  unlambdafy(L,UL),
  interpret(UL,R), !.

% might need to apply c2
reduce(L,R) :-
  unlambdafy(L,UL),
  convert(UL,RR),
  interpret(RR,R2),
  c2(R2,R), !.
```

```
reduce(L,R) :-
  unlambdafy(L,UL),
  string_to_list(UL,LL),
  convert(LL,RR),
  interpret(RR,R2),
  c2(R2,R), !.

reduce(L,R) :-
  unlambdafy(L,UL),
  interpret(UL,R2),
  c2(R2,R), !.

% remove all this ess/kay stuff, and put in the appropriate amount of a's
c2(ess(A),R) :- !,
  c2(A,AA),
  append("as",AA,R).

c2(ess(A,B),R) :- !,
  c2(A,AA),
  c2(B,BB),
  append("aas",AA,R1),
  append(R1,BB,R).

c2(ess(A,B,C),R) :- !,
  c2(A,AA),
  c2(B,BB),
  c2(C,CC),
  append("aaas",AA,R1),
  append(R1,BB,R2),
  append(R2,CC,R).

c2(kay(A),R) :- !,
  c2(A,AA),
  append("ak",AA,R).

c2(kay(A,B),R) :- !,
  c2(A,AA),
  c2(B,BB),
  append("aak",AA,R1),
  append(R1,BB,R).

c2(A,A) :- !.
```

Transcript

```
Welcome to SWI-Prolog (Multi-threaded, Version 5.4.7)
Copyright (c) 1990-2003 University of Amsterdam.
```

```
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [a4].
% a4 compiled 0.01 sec, 10,432 bytes

Yes
?- interpret([a,a,a,s,k,k,k],Y).

Y = [k]

Yes
?- interpret([a,a,d,a,dot(i),i,a,dot(h),i],R).
hi

R = [i]

Yes
?- interpretFromText("‘‘‘skkk",R).

R = "k"

Yes
?- interpretFromText("‘.G‘.o‘.o‘.d‘.b‘.y‘.e‘. ‘.W‘.o‘.r‘.l‘.d‘ri",R).

dlroW eybdooG

R = "i"

Yes
?- interpretFromText("‘v‘.G‘.o‘.o‘.d‘.b‘.y‘.e‘. ‘.W‘.o‘.r‘.l‘.d‘ri",R).

dlroW eybdooG

R = "v"

Yes
?- interpretFromText("‘‘‘s‘‘s‘‘sii‘ki‘k.*‘‘s‘‘s‘ks‘‘s‘k‘s‘ks‘‘s
                     ‘‘s‘ks‘‘s‘k‘s‘kr‘‘s‘k‘sikk‘k‘‘s‘ksk",R).

*
*
**
***
```

```
*****
********
*************
**********************
**********************************
**************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************

% Plus much more stuff here... this goes on forever

?- unlambdafy(var(x),R).

R = "x"

Yes
?- unlambdafy(func(r),R).

R = "r"

Yes
?- unlambdafy(func(dot(x)),R).

R = ".x"

Yes
?- unlambdafy(func(v),R).

R = "v"

Yes
?- unlambdafy(func(d),R).

R = "d"

Yes
?- unlambdafy(func(x),R).

No

?- unlambdafy(app(app(func(d),app(func(r),func(i))),func(v)),R).
```

```
R = "aadariv"

Yes
?- reduce(app(app(func(d),app(func(r),func(i))),func(v)),R).


R = [v]

Yes

?- reduce(app(func(d),app(func(r),func(i))),R).

R = [promise([a, r, i])]

Yes
```