Topics in computer architecture

Functional programming and architecture

P.J. Drongowski
SandSoftwareSound.net

```
{member

  |or @ &equals @ distl }

{equals

  (and @ [ atom@1, atom@2 ] -> = @ [1, 2] ;
   (or @ [ atom@1, atom@2 ] -> %F ;
    and @ [ subset@[2,1], subset ]
   )
  ) }

{subset

  |and @ &member @ distr }
```

def member $\equiv \backslash\vee \bullet \alpha$equals $\bullet$ distl.

def equals $\equiv$

$(\wedge \bullet [\ \text{atom}\bullet 1, \text{atom}\bullet 2\ ] \Rightarrow = \bullet [\ 1, 2\ ]\ ;$

$(\vee \bullet [\ \text{atom}\bullet 1, \text{atom}\bullet 2\ ] \Rightarrow \#F\ ;$

$\wedge \bullet [\ \text{subset} \bullet [\ 2, 1\ ],\ \text{subset}\ ]$

$)$

$)$.

def subset $\equiv \backslash\wedge \bullet \alpha$member $\bullet$ distr.

```
mu(Selector,L,Z) :- integer(Selector), selector(Selector,L,Z).
  selector(0,_,bottom).
  selector(1,[X|L],X).
  selector(I,[_|L],Z) :- selector(T,L,Z), I is T+1.

mu([cons|L],X,Z) :- construct(L,X,Z).
  construct([],X,[]).
  construct([F|L],X,[FX|FL]) :- mu(F,X,FX), construct(L,X,FL).

mu(tl,[],bottom) :- printerror("tl applied to null list.").
mu(tl,X,bottom)  :- atomic(X), printerror("tl applied to atom.").
mu(tl,[_],[]).
mu(tl,[_|L],L).

mu(last,[],bottom) :- printerror("last applied to null list.").
mu(last,X,bottom)  :- atomic(X), printerror("last applied to atom.").
mu(last,L,Z) :- last(L,Z).
  last([X],X).
  last([_|L],Z) :- last(L,Z).

mu(reverse,L,Z) :- revapp(L,[],Z).
  revapp([X|L1],L2,L3) :- revapp(L1,[X|L2],L3).
  revapp([],L,L).

mu(apndl,[Y|L],Z) :- apndl(Y,L,Z).
  apndl(Y,[],[Y]).
  apndl(Y,L,[Y|L]).

mu(apndr,[L|Y],Z) :- apndr(L,Y,Z).
  apndr([],Y,[Y]).
  apndr(L,Y,Z) :- append(L,[Y],Z).

mu(rotl,[X|L],Z) :- rotl(X,L,Z).
  rotl([],_,[]).
  rotl(X,[],[X]).
  rotl(X,L,Z)      :- append(L,[X],Z).
  rotl(_,_,bottom) :- printerror("rotl value error.").
```

```
mu(null,bottom,bottom).
mu(null,X,Z) :- X == [] -> Z is 1 ; Z is 0.

mu(atom,bottom,1).
mu(atom,X,Z) :- atomic(X) -> Z is 1 ; Z is 0.

mu([const,_],bottom,bottom).
mu([const,X],_,X).

mu(id,X,X).

mu(length,X,bottom) :-
   atomic(X),
   printerror("Length applied to atom.").
mu(length,L,Z) :- length(L,Z).

/mu(add,[A,B],Z)  :- number(A), number(B), Z is A+B.
mu(sub,[A,B],Z)  :- number(A), number(B), Z is A-B.
mu(mul,[A,B],Z)  :- number(A), number(B), Z is A*B.
mu(div,[A,B],Z)  :- number(A), number(B), Z is A/B.
mu(idiv,[A,B],Z) :- number(A), number(B), Z is A//B.
mu(imod,[A,B],Z) :- number(A), number(B), Z is A mod B.

mu(neg,A,Z)  :- number(A), Z is 0-A.

mu(eq,[A,B],Z) :- number(A), number(B), A ==  B -> Z is 1 ; Z is 0.
mu(ne,[A,B],Z) :- number(A), number(B), A =:= B -> Z is 1 ; Z is 0.
mu(lt,[A,B],Z) :- number(A), number(B), A  <  B -> Z is 1 ; Z is 0.
mu(le,[A,B],Z) :- number(A), number(B), A =<  B -> Z is 1 ; Z is 0.
mu(gt,[A,B],Z) :- number(A), number(B), A  >  B -> Z is 1 ; Z is 0.
mu(ge,[A,B],Z) :- number(A), number(B), A >=  B -> Z is 1 ; Z is 0.
```

```
mu([compIL],X,Z)       :- composition(L,X,Z).
   composition([],X,X).
   composition([FIL],X,Z) :- composition(L,X,T), mu(F,T,Z).

mu([forall,F],X,Z) :- forall(F,X,Z).
   forall(F,[],[]).
   forall(F,[XIXL],[ZIZL]) :- mu(F,X,Z), forall(F,XL,ZL).

mu([insert,F],L,Z) :- insert(F,L,Z).
   insert(F,[X],X).
   insert(F,[XIL],Z) :- insert(F,L,ZL), mu(F,[X,ZL],Z).

mu([corr,F],[A,B],Z) :- corr(F,A,B,Z).
   corr(F,[],[],[]).
   corr(F,[AIAL],[BIBL],[ZIZL]) :- mu(F,[A,B],Z), corr(F,AL,BL,ZL).

mu([cond,P,F,G],X,Z) :-
   mu(P, X, C),
   (C == 1 -> mu(F, X, Z) ; mu(G, X, Z)).

mu([while,P,F],X,Z) :- mu(P, X, C), C == 1 -> mu(F, X, Z).

/**********************
 * Apply defined function  *
 **********************/

mu(F,X,Z) :- def(F,FB), mu(FB, X, Z).

/*
 * If the interpreter reaches this point, either it could not match
 * a clause head or some internal failure (e.g. domain error) was
 * detected.) The atom "bottom" is returned.
 */

mu(_,_,bottom) :- printerror("Could not find function.").
```

```
tr(Source, FFP) :-

  fp_def(Structure, Source, []), !,
  ffp(Structure, FFP).

/*******************************
 * Translate FP to intermediate form *
 *******************************/

fp_def(i_def(N,B))   --> [def], fp_id(N), ['='], fp_fn(B), ['.'].

fp_fn(F)             --> ['('], fp_fn(F), [')'].

fp_fn(i_cons([F|C])) --> ['['], fp_fn(F), fp_cons_list(C).
fp_cons_list([F|C])  --> [','], fp_fn(F), fp_cons_list(C).
fp_cons_list([])     --> [']'].

fp_fn(i_const(O))    --> ['#'], fp_object(O).
fp_fn(i_forall(F))   --> ['@'], fp_fn(F).
fp_fn(i_insert(F))   --> ['/'], fp_fn(F).
fp_fn(i_corr(F))     --> ['\\'], fp_fn(F).
fp_fn(i_while(F,G))  --> [while], fp_fn(F), fp_fn(G).
fp_fn(i_bu(F,G))     --> [bu], fp_fn(F), fp_fn(G).

fp_fn(i_cond(P,F,G)) -->
        [cond], fp_fn(P), ['->'], fp_fn(F), [';'], fp_fn(G).

fp_fn(i_fn(F))       --> fp_id(F).
fp_fn(i_comp(F,C))   --> fp_fn(F), ['*'], fp_fn(C).

fp_object(O)         --> fp_id(O).
fp_object([O|L])     --> ['<'], fp_object(O), fp_sequence(L).
fp_sequence([O|L])   --> [','], fp_object(O), fp_sequence(L).
fp_sequence([])      --> ['>'].

fp_id(A) --> [A], {atomic(A)}.
```

```
/*******************************
 * Translate intermediate form to FFP *
 *******************************/

ffp(i_def(N,B), [def, N, Bffp] ) :- ffp(B, Bffp).

ffp(i_cons(C), [cons|Cffp] )  :- ffp_cons(C, Cffp).
  ffp_cons([], []).
  ffp_cons([C|CL], [Cffp|CLffp]) :- ffp(C, Cffp), ffp_cons(CL, CLffp).

ffp(i_const(K), [const,K]  ).

ffp(i_forall(F), [forall, Fffp] ) :- ffp(F, Fffp).
ffp(i_insert(F), [insert, Fffp] ) :- ffp(F, Fffp).
ffp(i_corr(F), [corr, Fffp]   ) :- ffp(F, Fffp).
ffp(i_while(P,G), [while, Pffp, Gffp] ) :- ffp(P, Pffp), ffp(G, Gffp).
ffp(i_bu(F,G), [bu, Fffp, Gffp] ) :- ffp(F, Fffp), ffp(G, Gffp).

ffp(i_cond(P,F,G), [cond, Pffp, Fffp, Gffp] )
          :- ffp(P, Pffp), ffp(F, Fffp), ffp(G, Gffp).

ffp(i_fn(F) , F ).
ffp(i_comp(F,C), [comp, Fffp, Cffp]) :- ffp(F, Fffp), ffp(C, Cffp).
```