```
----- Operazioni base relative allo Heap e agli entry points -----


deref( a: address ) : address

  <Tag, b> := Store[a];

  while  (Tag = REF) and (b <> a)  do
    a := b;
    <Tag, b> := Store[a]
  endwhile;

  return a

end deref


bind( a1, a2: address )  // a1, a2 dereferenziati

  <Tag1, b1> := Store[a1];
  <Tag2, b2> := Store[a2];

  if  Tag1 = REF  then
    Store[a1] := Store[a2];
    trail( a1 )
  else
    Store[a2] := Store[a1];
    trail( a2 )
  endif

end bind


unify( a1, a2: address )  // a1, a2 dereferenziati

  reset( PDL );

  push( PDL, < deref(a1), deref(a2) > );

  while  not empty( PDL )  do

    <a1, a2> := pop( PDL );

    if  a1 <> a2  then

      <Tag1, b1> := Store[a1];
      <Tag2, b2> := Store[a2];

      if  (Tag1 = REF) or (Tag2 = REF)  then
        bind( a1, a2 );
      else if  Heap[b1] = Heap[b2] = f/n  then
        for  i := n  downto  1  do
          push( PDL, < deref(b1+i), deref(b2+i) > )
        endfor
      else
        fail
      endif
    endif

  endwhile

end unify
```

```
----- Istruzioni per la costruzione dei termini nello Heap -----


put_str  f/n  Xi :

  Xi := Heap[H] := < STR, H+1 >;
  Heap[H+1] := f/n;
  H := H + 2;
  P := next( P )

end put_str


set_var  Xi :

  Xi := Heap[H] := < REF, H >;
  H := H + 1;
  P := next( P )

end set_var


set_val  Xi :

  Heap[H] := Xi;
  H := H + 1;
  P := next( P )

end set_val


----- Istruzioni per l'unificazione dei termini nello Heap -----


get_str  f/n  Xi :

  a := deref( @Xi );
  <Tag, b> := Store[a];
  if  Tag = REF  then
    Mode := Write;
    Heap[H] := < STR, H+1 >;
    Heap[H+1] := f/n;
    bind( a, H );
    H := H + 2
  else if  Store[b] = f/n  then
    Mode := Read;
    S := b + 1
  else
    fail
  endif;
  P := next( P )

end get_str


uni_var  Xi :

  if  Mode = Write  then
    Xi := Heap[H] := < REF, H >;
    H := H + 1
  else
```

```
    Xi := Heap[S];
    S := S + 1
  endif;
  P := next( P )

end uni_var


uni_val  Xi :

  if  Mode = Write  then
    Heap[H] := Xi;
    H := H + 1
  else
    unify( @Xi, S );
    S := S + 1
  endif;
  P := next( P )

end uni_val


----- Istruzioni per trattare il passaggio di termini come argomenti -----


put_str  f/n  Aj :  ...


get_str  f/n  Aj :  ...


put_var  Xi  Aj :

  Aj := Xi := Heap[H] := < REF, H >;
  H := H + 1;
  P := next( P )

end put_var


put_val  Xi  Aj :

  Aj := Xi;
  P := next( P )

end put_val


get_var  Xi  Aj :

  Xi := Aj;
  P := next( P )

end get_var


get_val  Xi  Aj :

  unify( @Xi, @Aj );
  P := next( P )

end get_val
```

```
----- Interpretazione procedurale dei predicati -----


call  q/n :

  Arity = n;
  CP := next( P );
  P := @{q/n}

end call


proceed :

  P := CP

end call


----- Allocazione di (environment / choice-point) frame nello Stack -----


new_frame() : address

  if  E > B  then
    return  E + 3 + Stack[E+2]
  else
    return  B + 7 + Stack[B]
  endif

end new_frame


----- Controllo procedurale: Environment Stack -----


// Yk = Stack[E+2+k]


alloc  k :

  E' := new_frame();
  endif;
  Stack[E']   := E;
  Stack[E'+1] := CP;
  Stack[E'+2] := k;
  E := E';
  P := next( P )

end alloc


dealloc :

  P := Stack[E+1];
  E := Stack[E]

end dealloc


----- Operazioni relative al backtracking -----
```

```
trail( v: address )

  Trail[TR] := v;
  TR := TR + 1

end trail


unwind_trail( t1, t2: address )

  for  t := t1  to  t2-1  do
    Heap[ Trail[t] ] := < REF, Trail[t] >
  endfor

end unwind_trail


fail

  n := Stack[B];
  P := Stack[B+n+6]

end fail


----- Controllo delle 'scelte': Choice-Point Stack -----


try_me_else  BP :

  B' := new_frame();
  Stack[B'] := n := Arity;
  for  j := 1  to  n  do
    Stack[B'+j] := Aj
  endfor;
  Stack[B'+n+1] := CP;
  Stack[B'+n+2] := H;
  Stack[B'+n+3] := E;
  Stack[B'+n+4] := B;
  Stack[B'+n+5] := TR;
  Stack[B'+n+6] := BP;
  B := B';
  P := next( P )

end alloc


retry_me_else  BP :

  n := Stack[B];
  for  j := 1  to  n  do
    Aj := Stack[B'+j]
  endfor;
  unwind_trail( Stack[B+n+5], TR );
  CP := Stack[B+n+1];
  H  := Stack[B+n+2];
  E  := Stack[B+n+3];
  TR := Stack[B+n+5];
  Stack[B+n+1] := BP;
  P := next( P )

end alloc
```

```
trust_me :

  n := Stack[B];
  for  j := 1  to  n  do
    Aj := Stack[B'+j]
  endfor;
  unwind_trail( Stack[B+n+5], TR );
  CP := Stack[B+n+1];
  H  := Stack[B+n+2];
  E  := Stack[B+n+3];
  TR := Stack[B+n+5];
  B  := Stack[B+n+4];
  P := next( P )

end alloc
```