Lidor Portal, 208396259: Ali Jabreen, 318425337

PPL ASSIGNMENT 5 – THEORETICAL

1.1

a.

Let us define the operator: $\boldsymbol{tail^n(lzl) := \begin{cases} (head\ lzl) & , n = 0 \\ (tail\ (tail\ (\dots (tail\ lzl)\ ) \dots)\ & , n > 0 \end{cases}}$

Definition: Two lazy-lists $lz1$, $lz2$ are said to be equal if:
both lists are empty $(empty - lzl?\ lz1\ \&\&\ empty - lzl?\ \ lz2)$, or
for every $n \in \mathbb{N}$: $tail^n(lz1) = tail^n(lz2)$.

To put it mathematically: $lz1 =_{lzl} lz2 \iff (empty - lzl?\ lz1 \wedge empty - lzl?\ lz2) \vee$
$\forall\ n \in \mathbb{N}: tail^n(lz1) = tail^n(lz2)$

b.

$\boldsymbol{(define\ even - square - 1}$
$\boldsymbol{(lzl - filter\ (lambda\ (x)\ (=\ (modulo\ x\ 2)\ 0))}$
$\boldsymbol{(lzl - map\ (lambda\ (x)\ (*\ x\ x))\ (integers - from\ 0))))}$

$\boldsymbol{(define\ even - square - 2}$
$\boldsymbol{(lzl - map\ (lambda\ (x)\ (*\ x\ x))}$
$\boldsymbol{(lzl - filter\ (lambda\ (x)\ (=\ (modulo\ x\ 2)\ 0))\ (integers - from\ 0))))}$

Theorem: let $even - square - 1$ and $even - square - 2$ be the lazy lists defined above respectively, then $even - square - 1 =_{lzl} even - square - 2$.

Proof:

Let us denote $even - square - 1$ and $even - square - 2$ by $es1$ and $es2$ respectively.

Let us also denote f and p by:

$\left(define\ f\ \left(lambda\ (x)(*\ x\ x)\right)\right)$, $\left(define\ p\ \left(lambda\ (x)\ (=\ (modulo\ n\ 2)\ 0)\right)\right)$

Since both lists are not empty, to prove their equivalence, we use induction on n:

Base case: when $n = 0$:

denote:
$ints_0 = \ (integers - from\ 0\ ) = (0\ .\ (lambda\ ()\ (\ integers - from\ 1))$,
$mints = (lzl - map\ \ f\ ints_0)$
$fmints = (lzl - filter\ p\ mints)$

By the definitions of $lzl - map$ and $lzl - filter$ we get:
$mints = (\ (f\ (head\ ints_0)\ )\ \ .\ \ (lambda\ (\quad)(lzl - map\ f\ (tail\ ints_0)\ )\ )\ ) =$
$= (0\ \ .\ \ (lambda\ (\quad)(lzl - map\ f\ (tail\ ints_0)\ )\ )\ )$

$fmints = (\ 0\ .\ (lambda\ (\quad)\ (lzl - filter\ p\ (tail\ mints)\ )\ )\ )$

Therefore, we get $es1 = fmints = (0\ .\ (lambda\ (\quad)(lzl - filter\ p\ (\ tail\ mints\ )\ )\ )\ )$

Similarly, we get $es2$:

First, denote:

$ints_0 = (integers - from\ 0\ ) = (0\ .(lambda\ ()\ (\ integers - from\ 1)),$
$fints = (lzl - filter\ \ p\ ints_0)$
$mfints = (lzl - map\ f\ fints).$

By the definitions of $lzl - map$ and $lzl - filter$ we get:
$fints = (\ (head\ ints_0)\ \ .\ \ (lambda\ (\quad)(lzl - filter\ p\ (tail\ ints_0)\ )\ )\ ) =$
$(\ 0\ \ .\ \ (lambda\ (\quad)(lzl - filter\ p\ (tail\ ints_0)\ )\ )\ )$

$mfints = (\ 0^2\ .(\ lambda\ ()\ (\ lzl - map\ f\ (tail\ fints)\ )\ )\ )$
$\qquad\qquad = (\ 0\ .(\ lambda\ ()\ (\ lzl - map\ f\ (tail\ fints)\ )\ )\ )$

Therefore, we get $es2 = mfints = (\ 0\ .(\ lambda\ (\quad)\ (\ lzl - map\ f\ (tail\ fints)\ )\ )\ )$.
By the definition of $tail^n$, we get : $tail^0(es1) = (head\ es1) = 0 = (head\ es2) = tail^0(es2).$

Induction step:

Let $k \in \mathbb{N}$ be given and suppose that $tail^n(es1) = tail^n(es2)$ is true for $n = k$. Then let $tail^n(es1) = (\ 4n^2\ .x) = tail^n(es2)$, where x is $(cdr\ tail^n(es1)\ ) = (cdr\ tail^n(es2)\ )$,

Then by the definition of the tail operator we get $tail^{n+1}(es1) = tail\ (\ tail^n(es1)\ )$, and by the induction hypothesis we get $tail^{n+1}(es1) = tail(\ tail^n(es1)\ ) =_* tail(\ tail^n(es2)\ ) = tail^{n+1}(es2)$, and the proof of the induction step is complete.
Thus, we conclude, $es1 = even - sqaures - 1 =_{lzl} even - squares - 2 = es2$. ■

2.a.

Definition: Let f be a procedure that either fails or succeeds and let $f\$$ be the 'Success-Fail-Continuations' version corresponding to f . f and $f\$$ are said to be equivalent if:
(f fails ⇔ $f\$$ fails) ∨ ((f succeeds ⇔ $f\$$ succeeds) ∧
$(\forall\ x_1, \ldots, x_n, succ - cont, fail - cont$:
$$(succ - cont\ (\ f\ x_1 \ldots x_n\ )) = (f\$\ x_1 \ldots x_n\ succ - cont\ fail - cont)\ ) .$$
and we denote $f =_{sf-cps} f\$$.

d.

```
(define get-value
  (lambda (assoc-list key)
    (cond ((empty? assoc-list) 'fail)
          ( (eq? (car (car assoc-list) ) key) (cdr (car assoc-list) ) )
          (else (get-value (cdr assoc-list) key)))))
```

```
(define get-value$
  (lambda (assoc-list key success fail)
    (cond ((empty? assoc-list) (fail))
          ((eq? (car (car assoc-list) ) key) (success (cdr (car assoc-list) ) ) )
          (else (get-value$ (cdr assoc-list) key
                            (lambda (res) (success res) ) fail)))))
```

Lidor Portal, 208396259: Ali Jabreen, 318425337

Theorem: let $get - value$ and $get - value\$$ be the procedures defined above respectively, then $get - value =_{sf-cps} get - value\$$.

Proof: Let us denote $get - value$ and $get - value\$$ by gv and $gv\$$ respectively.

The procedure f results in either a success when it returns a value of type T , or a fail.
We use induction on n, where n is the length of the list:

Let $lst, key, success, fail$ be given.

Base case: $n = 0$:

Based on the definitions of gv and $gv\$$ we get that $(gv\ lst\ key) = 'fail$,
$(gv\$\ lst\ key\ success\ fail) = (fail)$, i.e. both gv and $gv\$$ fail, by definition, $gv =_{sf-cps} gv\$$.

Induction step:

Suppose that for any lst of length $k < n$, and for any $key, success, fail$: $gv =_{sf-cps} gv\$$.

Let lst be of length n, and let $key, success, fail$ be given, and since lst is not empty, we know this step is not a fail. Then,

If $(\boldsymbol{eq}?\ \ (\boldsymbol{car}\ (\boldsymbol{car}\ \boldsymbol{lst})\ )\ \ \boldsymbol{key})$, we get:

$(gv\$\ lst\ key\ success\ fail) = (success\ (cdr\ (car\ lst)\ )\ ) =_{*} (\ success\ (\ gv\ lst\ key\ )\ )$,
therefore, by the definition of $=_{fs-cps}$, we get $gv =_{fs-cps} gv\$$.

($*$ by the definition of gv).

Otherwise, $(\boldsymbol{get} - \boldsymbol{value}\$\ (\boldsymbol{cdr\ assoc} - \boldsymbol{list})\ \ \boldsymbol{key}\ \ \boldsymbol{success}\ \ \boldsymbol{fail})$ is returned, and by the induction hypothesis($**$) we get:

$$(gv\$\ lst\ key\ success\ fail) = \ (gv\$\ (cdr\ lst)\ \ key\ \ (lambda\ (res)\ (success\ res)\ )\ fail) =_{**}$$
$$=_{**} (\ (lambda\ (res)\ (success\ res)\ )\ (\ gv\ (cdr\ lst)\ \ key\ )\ ) = (success\ (\ gv\ (cdr\ lst)\ key\ )\ )$$
$$= \cdots$$

By the definition of gv , for this else clause, we get $(gv\ (cdr\ lst)\ \ key\ )\ ) = (\ gv\ \ lst\ \ key\ )$.

... $= (success\ (\ gv\ lst\ key)\ )$.

Thus, by the definition of $=_{fs-cps}$, we conclude $gv =_{fs-cps} gv\$$. ■

3.1.

a.
$\boldsymbol{Unify}\ [\ \boldsymbol{t}(\boldsymbol{s}(\boldsymbol{s}), \boldsymbol{G}, \boldsymbol{H}, \boldsymbol{p}, \boldsymbol{t}, (\boldsymbol{E}), \boldsymbol{s}),$
$\boldsymbol{t}(\boldsymbol{s}(\boldsymbol{H}), \boldsymbol{G}, \boldsymbol{p}, \boldsymbol{p}, \boldsymbol{t}, (\boldsymbol{E}), \boldsymbol{K})]$

since t is a predicate, we get the following equations:

1. $s(H) = s(s)$	8. $H = s$
2. $G = G$	
3. $H = p$	
4. $p = p$	
5. $t = t$	
6. $E = E$	
7. $K = s$	

1. we pop eq1 and apply $S = \{\}$ to it to get eq1'=eq1. Since both sides are composite, according to step 7 of the algorithm, we get the new equation: 8. $H = s, S = \{\}$.
2. we pop eq2 and apply $S = \{\}$ to it to get eq2'=eq2. Since G and G are the same variable, according to step 4.3 of our algorithm, we continue and get $S = \{\}$.
3. we pop eq3 and apply $S = \{\}$ to it to get eq3'=eq3. Since H is a variable and s is a different term, according to step 4.2 of our algorithm we apply that equation to the current substitution and add it to it to get $S = \{H = p\}$.
4. we pop eq4 and apply $S = \{H = p\}$ to it to get eq4'=eq4. Since p and p are the same atoms, according to step 6 of our algorithm, we continue and $S = \{H = p\}$.
5. we pop eq5 and apply $S = \{H = p\}$ to it to get eq5'=eq5. Since p and p are the same atoms, according to step 6 of our algorithm, we continue and $S = \{H = p\}$.
6. we pop eq6 and apply $S = \{H = p\}$ to it to get eq6'=eq6. Since E and E are the same variable, according to step 4.3 of our algorithm, we continue and get $S = \{H = p\}$.
7. we pop eq7 and apply $S = \{H = p\}$ to it to get eq7'=eq7. Since K is a variable and s is a different term, according to step 4.2 of our algorithm we apply that equation to the current substitution and add it to it to get $S = \{H = p, K = s\}$.
8. we pop eq8 and apply $S = \{H = p, K = s\}$ to it to get eq8'= $(p = s)$. Since p and s are different atoms, according to step 6 of our algorithm, we return **FAIL.**

The result of this unification is **FAIL.**

b.

$\boldsymbol{Unify}\ [\ \boldsymbol{g(c, v(U), g, G, U, E, v(M))},$
$\qquad \boldsymbol{g(c, M, g, v(M), v(G), g, v(M))}]$

since g is a predicate, we get the following equations:

1. $c = c$
2. $M = v(U)$
3. $g = g$
4. $G = v(M)$
5. $U = v(G)$
6. $E = g$
7. $v(M) = v(M)$

1. we pop eq1 and apply $S = \{\}$ to it to get eq1'=eq1. Since c and c are the same atoms, according to step 6 of our algorithm, we continue and $S = \{\}$.
2. we pop eq2 and apply $S = \{\}$ to it to get eq2'=eq2. Since M is a variable and $v(U)$ is a different term, according to step 4.2 of our algorithm, we apply that equation to the current substitution and add it to it to get $S = \{M = v(U)\}$t.
3. we pop eq3 and apply $S = \{M = v(U)\}$ to it to get eq3'=eq3. Since g and g are the same atoms, according to step 6 of our algorithm, we continue and $S = \{M = v(U)\}$.
4. we pop eq4 and apply $S = \{M = v(U)\}$ to it to get eq4'= $\left(G = v\big(v(U)\big)\right)$. Since G is a variable and $v(v(U))$ is a different term, according to step 4.2 of our algorithm, we apply that equation to the current substitution and add it to it to get $S = \left\{M = v(U), G = v\big(v(U)\big)\right\}$.
5. we pop eq5 and apply $S = \left\{M = v(U), G = v\big(v(U)\big)\right\}$ to it to get eq5'=$\left(U = v\left(v\big(v(U)\big)\right)\right)$. Since $v\left(v\big(v(U)\big)\right)$ is a composite term that includes U, according to occurs-check, the unification fails.
The result of this unification is **FAIL.**

c.

$\boldsymbol{Unify}$ [$\boldsymbol{s}([\boldsymbol{v}|[\,[\boldsymbol{v}|\boldsymbol{V}]|\boldsymbol{A}]\,])$,
$\boldsymbol{s}([\boldsymbol{v}|\,[\boldsymbol{v}|\boldsymbol{A}]])$]

since s is a predicate, we get the following equations:

1. $[v|[\,[v|V]\,|\,A\,]\,] \;=[\,v\,|\,[\,v\,|\,A\,]\,]$
2. $v = v$
3. $[\,[v|V]\,|\,A] = [\,v\,|\,A\,]$
4. $[\,v\,|\,V\,] = v$
5. $A = A$

1. we pop eq1 and apply $S = \{\}$ to it to get eq1'=eq1. Since both sides are composite, according to step 7 of the algorithm, we get the new equations: 2. $v = v$, 3. $[\,[v|V]\,|\,A] = [\,v\,|\,A\,]$, $S = \{\}$.
2. we pop eq2 and apply $S = \{\}$ to it to get eq2' =eq2. Since v and v are the same atoms, according to step 6 of our algorithm, we continue and $S = \{\}$.
3. we pop eq3 and apply $S = \{\}$ to it to get eq3' =eq3. Since both sides are composite, according to step 7 of the algorithm, we get the new equations: 4. $[\,v\,|\,V\,] = v$, 5. $A = A$, $S = \{\}$.
4. We pop eq4 and apply $S = \{\}$ to it to get eq4'=eq4. Since one side is a composite and the other side is an atom, according to step 9 of our algorithm, we return **FAIL.**

The result of this unification is **FAIL.**

3.3.

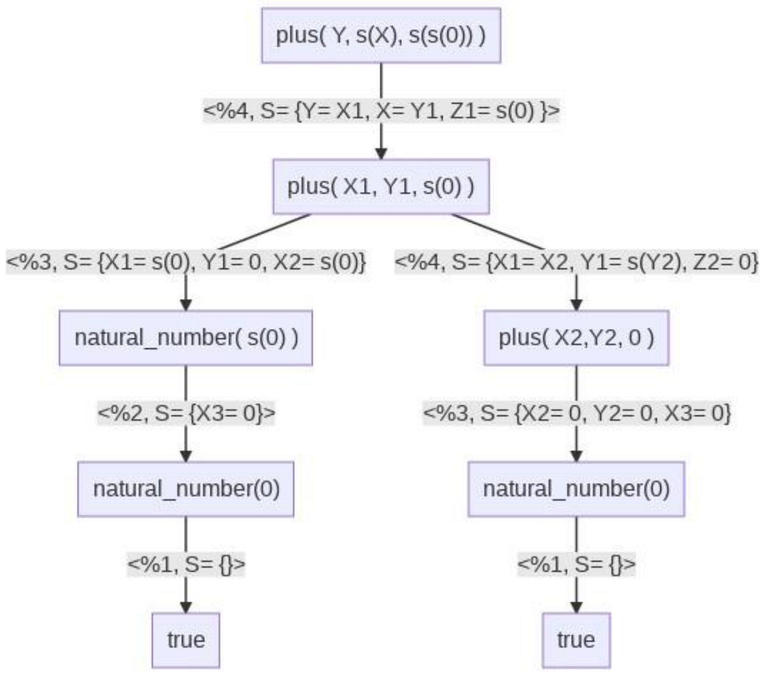
a. given the following program:

% Signature: natural_number(N)/1
% Purpose: N is a natural number.
natural_number(zero). %1
natural_number(s(X)) :- natural_number(X). %2

% Signature: plus(X, Y, Z)/3
% Purpose: Z is the sum of X and Y.
plus(X, zero, X) :- natural_number(X). %1
plus(X, s(Y), s(Z)) :- plus(X, Y, Z). %2

Let us denote: $natural\_nubmer(0).$ %1
$natural\_number\big(s(X)\big){:}-natural\_number(X).$ %2
$plus(\mathrm{X}, 0, \mathrm{X}){:}-natural\_number(\mathrm{X}).$ %3
$plus(\,X, s(Y), s(Z)\,) : -plus(X,Y,Z)$ %4

Lidor Portal, 208396259: Ali Jabreen, 318425337



Left path: $\boldsymbol{S = \{Y = X1,\ X = Y1,\ Z1 = s(0)\} \circ \{X1 = s(0), Y1 = 0, X2 = s(0)\} \circ \{X3 = 0\} = \{Y = s(0),\ X = 0,\ Z1 = s(0), X1 = s(0),\ Y1 = 0,\ X2 = s(0),\ X3 = 0\}}$

Restricting S to X, Y, we get $S = \{Y = s(0),\ X = 0\}$.

Right path: $\boldsymbol{S = \{Y = X1,\ X = Y1,\ Z1 = s(0)\} \circ \{X1 = X2,\ Y1 = s(Y2),\ Z2 = 0\} \circ \{X2 = 0, Y2 = 0,\ X3 = 0\} = \{Y = 0\ ,\ X = s(0),\ Z1 = s(0),\ X1 = 0\ ,\ Y1 = s(0),\ Z2 = 0,\ X2 = \ 0,\ Y2 = 0,\ X3 = 0\}}$

Restricting S to X, Y, we get $S = \{Y = 0,\ X = S(0)\}$.

b.

$\boldsymbol{Answers = \{S_1 = \{Y = s(0), X = 0\}, S_2 = \{Y = 0, X = s(0)\}\ \}}$

c.

This is a success proof tree, since there is at least one successful computation path (left/right path).

d.

This tree is finite, since all paths in the tree are of finite length.