PPL222

Assignment 5 Solution

Question 1 - CPS

1.b

Let us define the equivalence of high-order function g and its CPS version g$ as follows:
For any CPS-equivalent parameters f1…fn and f1$...fn$
(g$ f1$...fn$ cont) is CPS-equivalent to (cont (g f1…fn))

Following this definition, we show that pipe$ is equivalent to pipe, by induction on the size of the list.

<u>Base: N=1</u>

(cont (pipe(f1$))) = (cont f1$)
(pipe$ f1$ cont) = (cont (lambda (x cont2) (f1$ x cont2))) = (cont f1$)

<u>Induction step</u>: Assuming (pipe$ f1$ … fn$ cont) = (cont (pipe f1$ … fn$))

(pipe$ (f1$ … fn$ fn+1$ cont)) =
(pipe$ f2$ … fn+1$ (lambda (f2-n$) (cont (lambda (x cont2) (f1$ x (lambda (res) (fn2-n$ res cont2))))))) =
(
  (lambda (f2-n$) (cont (lambda (x cont2) (f1$ x (lambda (res) (fn2-n$ res cont2))))))
  (pipe f2$ … fn+1$)
)
=
(cont (lambda (x cont2) ((pipe f2$ … fn+1$) x (lambda (res) (fn2-n$ res cont2))))
= (cont (f2-n$ (pipe f1$ f2$ … fn+1$))
= (cont (pipe f1$ … fn+1$))

Question 2 - Lazy lists

2.a

We say that two given lazy lists / generators are equivalent if their i^{th} application yields the same value for any i > 0.

2.b

We show that both fibs1 and fibs2 yield in the i-th application the i-th Fibonacci number:

<u>Proposition 1</u>: In the n-th application of *fibs1*, the parameters *a* and *b* are the n-th and (n+1)-th Fibonacci numbers.

Base i=1: fibs1 is instantiated with a = 0 = fib_1, b = 1 = fib_2
Induction step: if the proposition holds for *i=n*, in the n-th application *a* = fib_n and *b*= fib_{n+1}.
According to the code of *fibs1*, the n+1 application will yield *b* [*(cons b (…))*], which is, according to the induction assumption, fib_{n+1}

<u>Proposition 2</u>: In the i-th application of *fibs2* the number yield is the i-th Fibonacci number.

Base i=1, i=2: in the first application *fibs2* yields *0* = fib_1. in the second application *fibs2* yields *1* = fib_2.
Induction step: the (n+1)-th application of *fibs2* yields the sum of the (n-1)-th and (n-2)-th applications, which are according to the induction assumption fib_{n-1} and fib_{n-2}, which is by definition fib_n

Question 3 - Logic programing

3.1 Unification

What is the result of the operations? Provide all the algorithm steps. Explain in case of failure.

a. unify[p(v(v(d(M),M,ntuf3),X)), p(v(d(B),v(B,ntuf3),KtM))]
    - Same predicates (p), same arity (1)
    -> equations = [p(v(v(d(M),M,ntuf3),X)) = p(v(d(B),v(B,ntuf3),KtM))]

    - Compound terms, Same predicate (p), same arity (1)
    -> split equation
    -> equations = [v(v(d(M),M,ntuf3),X) = v(d(B),v(B,ntuf3),KtM)]

    - Compound terms, Same predicate (v), Different arities (2,3)

-> Failure

b. unify[n(d(D),D,d,k,n(N),K),n(d(d),D,d,k,n(N),d)]

    - Same predicates (n), same arity (6)
    -> equations: [d(D) = d(d), D=D, d=d, k=k, n(N)=n(N), K=d]

    - d(D) = d(d)
    -> split equation D=d

-> equations: [d=d, k=k, n(N)=n(N), K=d, D=d]
-> {}

- d=d

-> equations: [k=k, n(N)=n(N), K=d, D=d]
-> {}

- k=k

-> equations: [n(N)=n(N), K=d, D=d]
-> {}

- n(N) = n(N)

-> split equation N=N
-> equations: [k=D, D=d, N=N]
-> {}

- K = d

-> equations: [D=d, N=N]
-> { d = K}

- D = d

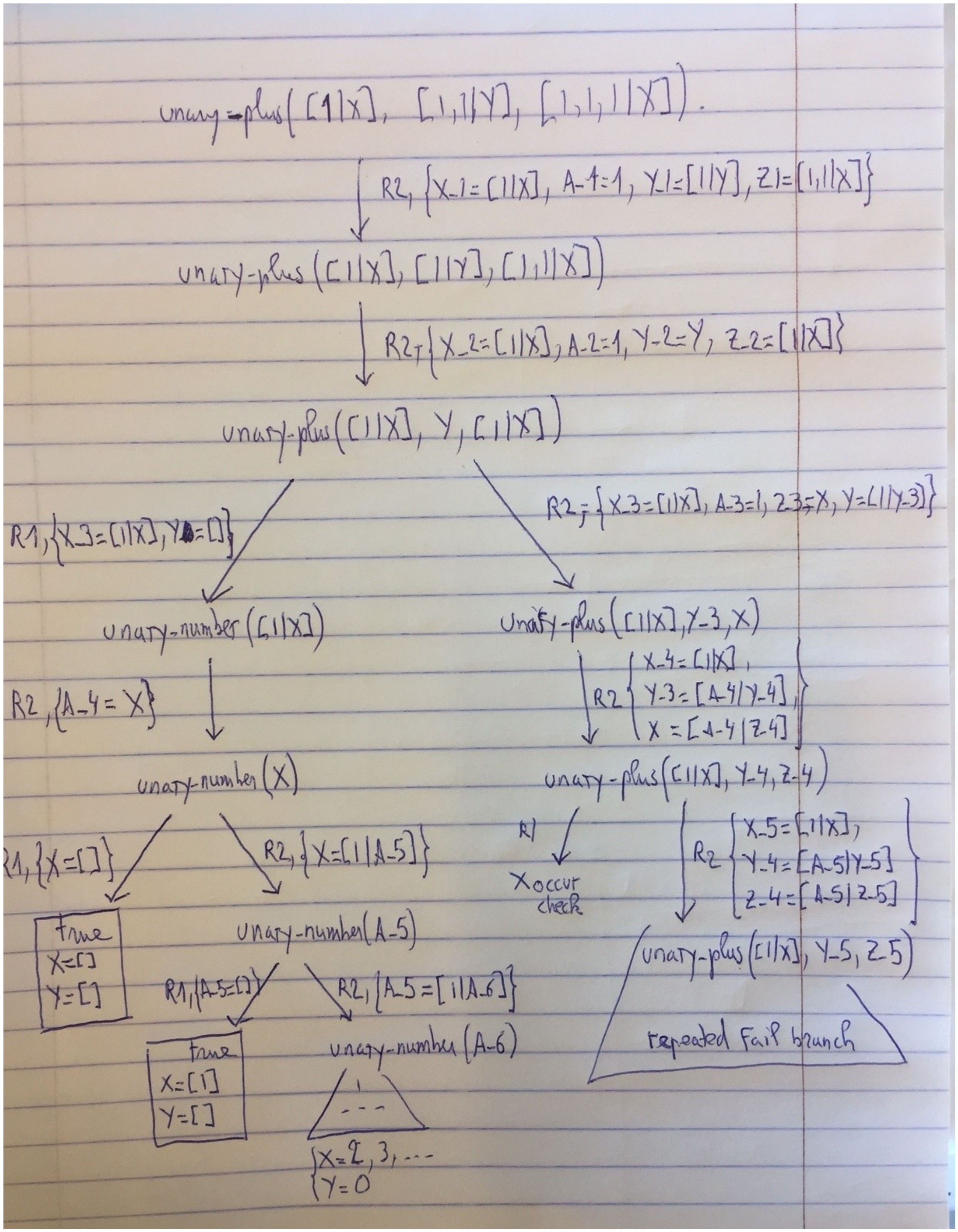
-> equations: [N=N]
-> { d = K, d=D}

- N = N

-> equations: []
-> { K = d, D = d}

3.3 Proof tree

a.


unary-plus([1|X], [1,1|Y], [1,1,1|X]).
R2, {X_1=[1|X], A_1=1, Y_1=[1|Y], Z1=[1,1|X]}
unary-plus([1|X], [1|Y], [1,1|X])
R2, {X_2=[1|X], A_2=1, Y_2=Y, Z_2=[1|X]}
unary-plus([1|X], Y, [1|X])
R1, {X_3=[1|X], Y=[]}
R2, {X_3=[1|X], A_3=1, Z_3=X, Y=[1|Y_3]}
unary-number([1|X])
unary-plus([1|X], Y_3, X)
R2, {A_4 = X}
R2 {X_4=[1|X], Y_3=[A_4|Y_4], X=[A_4|Z_4]}
unary-number(X)
unary-plus([1|X], Y_4, Z_4)
R1, {X=[]}
R2, {X=[1|A_5]}
R1
X occur check
R2 {X_5=[1|X], Y_4=[A_5|Y_5], Z_4=[A_5|Z_5]}
true
X=[]
Y=[]
unary-number(A_5)
unary-plus([1|X], Y_5, Z_5)
R1, {A_5=[]}
R2, {A_5=[1|A_6]}
unary-number(A_6)
repeated fail branch
true
X=[1]
Y=[]
X=2, 3, ...
Y=0

b. Is this a success or a failure proof tree?

See definitions from the material (p.352):
Finite success proof tree: A finite tree with a successful path.
(Finite) Failure proof tree: A finite tree with no successful path.
Infinite success proof tree: An infinite tree with a successful path.
Infinite failure proof tree: An infinite tree with no successful path. Dangerous to explore.

This tree is an infinite success proof tree.

c. Is this tree finite or infinite?

This tree has an infinite number of success nodes and an infinite number of failure nodes.

The query corresponds to the equation:
unary_plus([1|X], [1,1|Y], [1,1,1|X])

X+1 + Y+2 = X+3
which results in:
X+Y+3 = X+3
It has an infinite number of success answers:
{X=0, Y=0}
{X=1, Y=0}
{X=2, Y=0}
...

Which are encoded as unary numbers as:
X=[], Y=[]
X=[1], Y=[]
X=[1,1], Y=[]
...

d. Is this query provable from the given program?

A query is provable from a program , denoted , iff for some goal and rule selection rules Gsel and Rsel, the proof tree algorithm for answer-query(Q, P, Gsel, Rsel) computes a success tree.
For our case, if Gsel and Rsel return the goals and the rules by their order in the query/program, the computation reaches a success node (the leftmost node in the above proof tree).

e. As discussed in class, a language with a functor constructs infinite proof paths without circularity, and though undecidable.