Jason Zhang

PL Assignment 9

Collaborators: Dan Qian

1.

a. $1+1 \cong 3$

equivalence does not hold

Let $C =$ If $\bullet = 2$ Then 0 Else $(0\ 0)$

$C[1+1] =$ If $(1+1)=2$ Then 0 Else $(0\ 0) \to 0$

$C[3] =$ If $3=2$ Then 0 Else $(0\ 0) \to (0\ 0)$ which cannot evaluate

thus $1+1 \ncong 3$

b.

$x \cong z$

equivalence does not hold

Let $C = (\text{Fun } x \to \bullet + 1)\ 1$

$C[x] = (\text{Fun } x \to x+1)\ 1 \Rightarrow 2$

$C[z] = (\text{Fun } x \to z+1)\ 1 \Rightarrow z+1$ cannot evaluate with a free variable

c. $(\text{fun } x \to e)(e') \cong e[e'/x]$ (e' no free variables x)

$e[e'/x] \cong (\text{Fun } x \to e)(e')$ by the β rule

$(\text{Fun } x \to e)(e') \cong (\text{Fun } x \to e)(e')$ holds by reflexivity

thus equivalence does hold

d.

Fun x -> x $\cong$ Fun x -> x+0

Equivalence does not hold

let C = (•) True

C[Fun x -> x] = (Fun x -> x) True -> True

C[Fun x -> x+0] = (Fun x -> x+0) True -> True +0 cannot evaluate Bool + Int

e. (Fun x -> x) (Fun y -> y) $\cong$ (Fun z -> z)

using the left side: when we apply:

(Fun x -> x) (Fun y -> y) -> (Fun y -> y)

using α Equivalnce: Fun y -> y $\cong$ (Function z -> (y [z/y]) -> (Fun z -> z)

we now get:

(Fun z -> z) $\cong$ (Fun z -> z) which is true by reflexivity property

thus the equivalence holds.

f.

x+y+z $\cong$ z+y+x

we can define a new rule based on the commutative property of addition: n+n' $\cong$ n'+n

thus based on this rule x+y+z $\cong$ z+y+x

thus this equivalence holds

2. <u>They are the same.</u> proof by contradiction.

Let's assume $e \cong e'$ (by the regular definition 2.16) and when $C[e] \Rightarrow 0$ $C[e'] \not\Rightarrow 0$ (vice versa), for any context C. Since $e \cong e'$, $C[e]$ and $C[e']$ terminate to some v, v' respectively. WLOG, if in some context C_0, $C_0[e]$ terminates to 0 (succinctly $C_0[e] \Rightarrow 0$), then e must have satisfied some expression S in C_0 that led to a termination of 0. Since $e \cong e'$, then e' must have satisfied the same expression S in C_0 which leads to a termination of 0. Thus we get $v = v' = 0$. Therefore the variant of 2.16 holds.

3. Include S into our grammar of C

$C ::= \bullet \mid \ldots \text{Fb Stuff} \ldots \mid \text{Ref } e \mid e := e \mid ! e \mid$

$v ::= \ldots \text{Fb} \ldots \mid \bar{c}$

$\bar{c}$ (cell names)

Def: $e \cong e'$ if and only if for all contexts C (defined above) such that $C[e]$ and $C[e']$ are closed, $\langle C[e], S_0 \rangle \Rightarrow \langle v, S \rangle$ if and only if $\langle C[e'], S_0 \rangle \Rightarrow \langle v', S \rangle$ (S_0 initial state, S final state)

4. Fun f -> f0 + f1 $\cong$ Fun f -> f1 + f0

This will not hold in FbS due to possible differing state changes when running f1 and f0 in different orders.

We can produce a C to show that it does not work:

Let C = $\bullet$ (Let r = Ref (0) In Fun x -> If x = 0 Then
    r := r + 1
  Else
    If !r = 1 Then
      1
    Else
      (0 0)

C [Fun f -> f0 + f1] => 2

C [Fun f -> f1 + f0] => (0 0)
which does not evaluate thus

Fun f -> f0 + f1 $\not\cong$ Fun f -> f1 + f0