Modelling a Language

January 24, 2020

1 Syntax

$$x \in \textsc{Var}, n \in \textsc{Nat}, b \in \textsc{Bool}, v \in \textsc{Val}, e \in \textsc{Exp}, d \in \textsc{Dic}, o \in \textsc{Obs}$$

$$\begin{array}{rcl}
n & ::= & \texttt{z} \mid \texttt{s}\ n \\
b & ::= & \texttt{true} \mid \texttt{false} \mid \neg b \\
v & ::= & n \mid b \\
e & ::= & n \mid b \mid \texttt{s}\ e \mid \neg e \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \lambda x.e \mid e\ e \mid d \mid d(e) \\
 & & \mid \texttt{nat-fold}\ e\ e\ e \\
d & ::= & \cdot \mid d[x \mapsto v] \\
o & ::= & v
\end{array}$$

Remark 1. A natural number $n \in \textsc{Nat}$ is (1) zero, denoted by $\texttt{z}$, or (2) the successor of a natural number, denoted by $\texttt{s}\ n$.

A boolean $b \in \textsc{Bool}$ is (1) $\texttt{true}$, (2) $\texttt{false}$, or (3) the negation of a boolean, denoted by $\neg b$.

A value $v \in \textsc{Value}$ is (1) a natural number n, or (2) a boolean b.

An expression $e \in \textsc{Exp}$ is (1) a natural number n, (2) a boolean b, (3) the successor of an expression $\texttt{s}\ e$, (4) the negation of an expression $\neg e$, (5) an if-then-else expression $\texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$, (6) a function $\lambda x.e$, (7) a function application $e\ e$, (8) a dictionary d, (9) a dictionary projection $d(e)$, or (10) a nat-fold expression $\texttt{nat-fold}\ e\ e\ e$.

A dictionary $d \in \textsc{Dic}$ is (1) an empty dictionary, denoted by $\cdot$, or (2) an dictionary d with an update that the variable x maps to the value v, denoted by $d[x \mapsto v]$.

An observation $o \in \textsc{Obs}$ is a value v.

Remark 2. Introduction forms include: $\texttt{z}$, $\texttt{s}\ n$, $\texttt{true}$, $\texttt{false}$, $\lambda x.e$, d

Elimination forms include: $\neg e$, $\texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$, $e\ e$, $d(e)$, $\texttt{nat-fold}\ e\ e\ e$

Remark 3. It is straightforward that expressions are well-defined as all self-referencing premises are "smaller" than the conclusion. For example, the subexpressions e's in the expression $\texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$ are all "smaller" than the expression itself. Although n, b and d are defined elsewhere, they all are well-defined if we examine their respective syntactic structures. For example, if we examine $d \in \textsc{Dic}$, the subterm d in $d[x \mapsto v]$ is "smaller" than the term itself.

2 Semantics

2.1 Reduction relation

$$\boxed{e \longrightarrow e}$$

$$\frac{}{\neg\texttt{true} \longrightarrow \texttt{false}}\ (\text{not-t})$$

$$\frac{}{\neg\texttt{false} \longrightarrow \texttt{true}}\ (\text{not-f})$$

$$\frac{}{\texttt{if true then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_1}\ (\text{if-t})$$

$$\frac{}{\texttt{if false then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_2}\ (\text{if-f})$$

$$\frac{}{(\lambda x.e_1)\ e_2 \longrightarrow e_1[e_2/x]}\ (\text{app})$$

$$\frac{}{d[x \mapsto v](x) \longrightarrow v}\ (\text{dic-proj-eq})$$

$$\frac{x \not\equiv y}{d[x \mapsto v](y) \longrightarrow d(y)} \text{ (dic-proj-df)}$$

$$\frac{}{\texttt{nat-fold}\ \texttt{z}\ e_2\ e_3 \longrightarrow e_2} \text{ (nat-fold-0)}$$

$$\frac{}{\texttt{nat-fold}\ (\texttt{s}\ e)\ e_2\ e_3 \longrightarrow ((e_3\ e)\ (\texttt{nat-fold}\ e\ e_2\ e_3))} \text{ (nat-fold-s)}$$

2.2 Conversion relation

$$\boxed{e \longrightarrow^1 e}$$

$$\frac{e_1 \longrightarrow e_2}{e_1 \longrightarrow^1 e_2} \text{ (step)}$$

$$\frac{b_1 \longrightarrow^1 b_{11}}{\texttt{s}\ b_1 \longrightarrow^1 \texttt{s}\ b_{11}} \text{ (suc-cp-b1)}$$

$$\frac{b_1 \longrightarrow^1 b_{11}}{\neg b_1 \longrightarrow^1 \neg b_{11}} \text{ (not-cp-b1)}$$

$$\frac{e_1 \longrightarrow^1 e_{11}}{\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \longrightarrow^1 \texttt{if}\ e_{11}\ \texttt{then}\ e_2\ \texttt{else}\ e_3} \text{ (if-cp-e1)}$$

$$\frac{e_2 \longrightarrow^1 e_{21}}{\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \longrightarrow^1 \texttt{if}\ e_1\ \texttt{then}\ e_{21}\ \texttt{else}\ e_3} \text{ (if-cp-e2)}$$

$$\frac{e_3 \longrightarrow^1 e_{31}}{\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \longrightarrow^1 \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_{31}} \text{ (if-cp-e3)}$$

$$\frac{e_1 \longrightarrow^1 e_{11}}{\lambda x.e_1 \longrightarrow^1 \lambda x.e_{11}} \text{ (fun-cp-e1)}$$

$$\frac{e_1 \longrightarrow^1 e_{11}}{e_1\ e_2 \longrightarrow^1 e_{11}\ e_2} \text{ (app-cp-e1)}$$

$$\frac{e_2 \longrightarrow^1 e_{21}}{e_1\ e_2 \longrightarrow^1 e_1\ e_{21}} \text{ (app-cp-e2)}$$

$$\frac{e_1 \longrightarrow^1 e_{11}}{\texttt{nat-fold}\ e_1\ e_2\ e_3 \longrightarrow^1 \texttt{nat-fold}\ e_{11}\ e_2\ e_3} \text{ (nat-fold-cp-e1)}$$

$$\frac{e_2 \longrightarrow^1 e_{21}}{\texttt{nat-fold}\ e_1\ e_2\ e_3 \longrightarrow^1 \texttt{nat-fold}\ e_1\ e_{21}\ e_3} \text{ (nat-fold-cp-e2)}$$

$$\frac{e_3 \longrightarrow^1 e_{31}}{\texttt{nat-fold}\ e_1\ e_2\ e_3 \longrightarrow^1 \texttt{nat-fold}\ e_1\ e_2\ e_{31}} \text{ (nat-fold-cp-e3)}$$

$$\boxed{e \longrightarrow^* e}$$

$$\frac{}{e_1 \longrightarrow^* e_1} \text{ (refl)}$$

$$\frac{e_1 \longrightarrow^1 e_2 \quad e_2 \longrightarrow^* e_3}{e_1 \longrightarrow^* e_3} \text{ (trans)}$$

2.3 Evaluation function

$$\boxed{\texttt{eval}(e) = o}$$

$$\frac{e \longrightarrow^* o}{\texttt{eval}(e) = o} \text{ (eval)}$$

3 Properties and Proofs

3.1 Properties

Problem 4. Explain whether all syntactic expressions are well-defined, in the sense that they produce valid observations in the evaluation function.

Solution 5. For the above definitions, the answer is negative. Proceed by cases on $e \in \textsc{Exp}$.

Case 1. ($e = n$ or $e = b$). These two cases will reduce to an observation.

Case 2. ($e = \texttt{s}\ e_1$). Unless e_1 is a natural number $n \in \textsc{Nat}$, it does not reduce to an observation.

Case 3. ($e = \neg e_1$). Unless e_1 is a boolean $n \in \textsc{Bool}$, it does not reduce to an observation.

Case 4. ($e = \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3$). If e_1 reduce to $\texttt{true}$ (or $\texttt{false}$), and e_2 (or e_3) reduces to an observation, then e reduces to an observation. Otherwise, it does not reduce to an observation.

Case 5. ($e = \lambda x.e_1$). It is not an observation and does not reduce to an observation.

Case 6. ($e = e_1\ e_2$). We expect e_1 to reduce to a function of the form $\lambda x.e_{11}$. We expect $e_{11}[e_2/x]$ to reduce to an observation. If either does not hold, it does not reduce to an observation.

Case 7. ($e = d$). Analogous to the ($e = \lambda x.e_1$) case with respect to the definition of $d \in \textsc{Dic}$.

Case 8. ($e = d(e_1)$). Analogous to the ($e = e_1\ e_2$) case with respect to the definition of $d \in \textsc{Dic}$.

Case 9. ($\texttt{nat-fold}\ e_1\ e_2\ e_3$). We expect e_1 to reduce to a natural number $n \in \textsc{Nat}$. We expect e_3 to reduce to a function $\lambda x.\lambda y.e_{31}$. We expect its reduction according to the rules (nat-fold-0) and (nat-fold-s) will lead to an observation. If either does not hold, it does not reduce to an observation.

There may be a way to make all syntactic expressions are "well-defined": (1) add canonical forms such as $\lambda x.e$ and d to $o \in \textsc{Obs}$. (2) add an error term to $o \in \textsc{Obs}$. (3) If an expression is stuck, make it reduce to the error term.

3.2 About dictionaries

Proposition 6. $\texttt{eval}((\cdot[a \mapsto 5][b \mapsto 120][c \mapsto \texttt{false}])(b)) = 120$.

Proof. We immediately get:

$$\dfrac{\dfrac{\dfrac{}{(\cdot[a \mapsto 5][b \mapsto 120][c \mapsto \texttt{false}])(b) \longrightarrow (\cdot[a \mapsto 5][b \mapsto 120])(b)}\,\text{(dic-proj-df)}}{(\cdot[a \mapsto 5][b \mapsto 120][c \mapsto \texttt{false}])(b) \longrightarrow^1 (\cdot[a \mapsto 5][b \mapsto 120])(b)}\,\text{(step)} \quad \dfrac{\dfrac{\dfrac{}{(\cdot[a \mapsto 5][b \mapsto 120])(b) \longrightarrow 120}\,\text{(dic-proj-eq)}}{(\cdot[a \mapsto 5][b \mapsto 120])(b) \longrightarrow^1 120}\,\text{(step)} \quad \dfrac{}{120 \longrightarrow^* 120}\,\text{(refl)}}{(\cdot[a \mapsto 5][b \mapsto 120])(b) \longrightarrow^* 120}\,\text{(trans)}}{(\cdot[a \mapsto 5][b \mapsto 120][c \mapsto \texttt{false}])(b) \longrightarrow^* 120}\,\text{(trans)}$$

Thus, we have:

$$\dfrac{\dfrac{\vdots}{(\cdot[a \mapsto 5][b \mapsto 120][c \mapsto \texttt{false}])(b) \longrightarrow^* 120}}{\texttt{eval}((\cdot[a \mapsto 5][b \mapsto 120][c \mapsto \texttt{false}])(b)) = 120}\,\text{(eval)}$$

$\square$

3.3 About numbers: `is-zero?`

Remark 7. Note in this subsection and the next one, we may omit mentioning uninteresting conversion relations when showing a sequence of reductions.

Definition 8. $\texttt{is-zero?}\ e = \texttt{nat-fold}\ e\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false})$.

Proposition 9. $\texttt{eval}(\texttt{is-zero?}\ \texttt{z}) = \texttt{true}$.

Proof. To prove $\texttt{eval}(\texttt{is-zero?}\ \texttt{z}) = \texttt{true}$, by (eval) and by the definition of is-zero?, it is sufficient to show

$$\texttt{nat-fold}\ \texttt{z}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) \longrightarrow^* \texttt{true}$$

We have:

$$\dfrac{\dfrac{\dfrac{}{\texttt{nat-fold}\ \texttt{z}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) \longrightarrow \texttt{true}}\ (\text{nat-fold-0})}{\texttt{nat-fold}\ \texttt{z}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) \longrightarrow^1 \texttt{true}}\ (\text{step}) \qquad \dfrac{}{\texttt{true} \longrightarrow^* \texttt{true}}\ (\text{refl})}{\texttt{nat-fold}\ \texttt{z}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) \longrightarrow^* \texttt{true}}\ (\text{trans})$$

□

Proposition 10. $\texttt{eval}(\texttt{is-zero?}\ \texttt{s(z)}) = \texttt{false}$.

Proof. By (eval) and by the definition of is-zero?, it is sufficient to show

$$\texttt{nat-fold}\ \texttt{s(z)}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) \longrightarrow^* \texttt{false}$$

We have:

$$\begin{array}{llll}
 & \texttt{nat-fold}\ \texttt{s(z)}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) & & (\#0) \\
\longrightarrow^* & ((\lambda x.\lambda y.\texttt{false})\ \texttt{z})\ (\texttt{nat-fold}\ \texttt{z}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false})) & \text{by (nat-fold-s)} & (\#1) \\
\longrightarrow^* & (\lambda y.\texttt{false})\ (\texttt{nat-fold}\ \texttt{z}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false})) & \text{by (app)} & (\#2) \\
\longrightarrow^* & \texttt{false} & \text{by (app)} & (\#3)
\end{array}$$

By (step), (refl) and (trans),

$$\texttt{nat-fold}\ \texttt{s(z)}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) \longrightarrow^* \texttt{false}$$

□

Remark 11. Note that after (#1) and after (#2), we may apply (app-cp-e2) and try to reduce the sub-expression ($\texttt{nat-fold}\ \texttt{z}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false})$). We will get the same result in the end.

Proposition 12. $\texttt{eval}(\texttt{is-zero?}\ \texttt{s(s(z))}) = \texttt{false}$.

Proof. By (eval) and by the definition of is-zero?, it is sufficient to show:

$$\texttt{nat-fold}\ \texttt{s(s(z))}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) \longrightarrow^* \texttt{false}$$

We have:

$$\begin{array}{llll}
 & \texttt{nat-fold}\ \texttt{s(s(z))}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) & & (\#0) \\
\longrightarrow^* & ((\lambda x.\lambda y.\texttt{false})\ \texttt{s(z)})\ (\texttt{nat-fold}\ \texttt{s(z)}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false})) & \text{by (nat-fold-s)} & (\#1) \\
\longrightarrow^* & (\lambda y.\texttt{false})\ (\texttt{nat-fold}\ \texttt{s(z)}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false})) & \text{by (app)} & (\#2) \\
\longrightarrow^* & \texttt{false} & \text{by (app)} & (\#3)
\end{array}$$

By (step), (refl) and (trans),

$$\texttt{nat-fold}\ \texttt{s(s(z))}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false}) \longrightarrow^* \texttt{false}$$

□

Remark 13. Note that after (#1) and after (#2), we may apply (app-cp-e2) and try to reduce the sub-expression ($\texttt{nat-fold}\ \texttt{s(z)}\ \texttt{true}\ (\lambda x.\lambda y.\texttt{false})$). We will get the same result in the end.

3.4 About numbers: +

Definition 14. $e_1 + e_2 = \texttt{nat-fold}\ e_1\ e_2\ (\lambda x.\lambda y.\texttt{s}(y))$.

Proposition 15. $\texttt{eval}(\texttt{z} + \texttt{z}) = \texttt{z}$.

Proof. To prove $\texttt{eval}(\texttt{z} + \texttt{z}) = \texttt{z}$, by (eval) and the definition of $+$, it is sufficient to show

$$\texttt{nat-fold z z } (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{z}$$

We have:

$$\dfrac{\dfrac{\dfrac{}{\texttt{nat-fold z z } (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow \texttt{z}}\,\text{(nat-fold-0)}}{\texttt{nat-fold z z } (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^1 \texttt{z}}\,\text{(step)} \qquad \dfrac{}{\texttt{z} \longrightarrow^* \texttt{z}}\,\text{(refl)}}{\texttt{nat-fold z z } (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{z}}\,\text{(trans)}$$

□

Proposition 16. $\texttt{eval}(\texttt{z} + \texttt{s}(\texttt{z})) = \texttt{s}(\texttt{z})$.

Proof. By (eval) and by the definition of $+$, it is sufficient to show

$$\texttt{nat-fold z s(z) } (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{z})$$

We have:

$$\begin{array}{llll} & \texttt{nat-fold z s(z) } (\lambda x.\lambda y.\texttt{s}(y)) & & (\#0) \\ \longrightarrow^* & \texttt{s}(\texttt{z}) & \text{by (nat-fold-0)} & (\#1) \end{array}$$

By (step), (refl) and (trans),

$$\texttt{nat-fold z s(z) } (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{z})$$

□

Proposition 17. $\texttt{eval}(\texttt{s}(\texttt{z}) + \texttt{s}(\texttt{z})) = \texttt{s}(\texttt{s}(\texttt{z}))$.

Proof. By (eval) and by the definition of $+$, it is sufficient to show

$$\texttt{nat-fold s(z) s(z) } (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{s}(\texttt{z}))$$

We have:

$$\begin{array}{llll} & \texttt{nat-fold s(z) s(z) } (\lambda x.\lambda y.\texttt{s}(y)) & & (\#0) \\ \longrightarrow^* & (\lambda x.\lambda y.\texttt{s}(y))\ \texttt{z}\ (\texttt{nat-fold z s(z) } (\lambda x.\lambda y.\texttt{s}(y))) & \text{by (nat-fold-s)} & (\#1) \\ \longrightarrow^* & (\lambda y.\texttt{s}(y))\ (\texttt{nat-fold z s(z) } (\lambda x.\lambda y.\texttt{s}(y))) & \text{by (app)} & (\#2) \\ \longrightarrow^* & (\lambda y.\texttt{s}(y))\ \texttt{s}(\texttt{z}) & \text{by (nat-fold-0)} & (\#3) \\ \longrightarrow^* & \texttt{s}(\texttt{s}(\texttt{z})) & \text{by (app)} & (\#4) \end{array}$$

By (step), (refl) and (trans),

$$\texttt{nat-fold s(z) s(z) } (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{s}(\texttt{z}))$$

□

Lemma 18. $\texttt{nat-fold } \underbrace{\texttt{s}(\ldots\texttt{s}}_{n\ \texttt{s}\text{'s}}(\texttt{z})\ldots)\ e_2\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \underbrace{\texttt{s}(\ldots\texttt{s}}_{n\ \texttt{s}\text{'s}}(\texttt{nat-fold z } e_2\ (\lambda x.\lambda y.\texttt{s}(y)))\ldots)$ *where* $n \in \mathbb{N}$.

Proof. Proceed by induction on $n \in \mathbb{N}$.

Case 1. $(n = 0)$. By (refl), we have:

$$\texttt{nat-fold z } e_2\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{nat-fold z } e_2\ (\lambda x.\lambda y.\texttt{s}(y))$$

Case 2. $(n = i + 1$ where $i \geq 0)$. We have:

$$\begin{array}{lll} & \texttt{nat-fold } \underbrace{\texttt{s}(\ldots\texttt{s}}_{(i+1)\ \texttt{s}\text{'s}}(\texttt{z})\ldots)\ e_2\ (\lambda x.\lambda y.\texttt{s}(y)) & \\ \longrightarrow^* & (\lambda x.\lambda y.\texttt{s}(y))\ \underbrace{\texttt{s}(\ldots\texttt{s}}_{i\ \texttt{s}\text{'s}}(\texttt{z})\ldots)\ (\texttt{nat-fold } \underbrace{\texttt{s}(\ldots\texttt{s}}_{i\ \texttt{s}\text{'s}}(\texttt{z})\ldots)\ e_2\ (\lambda x.\lambda y.\texttt{s}(y))) & \text{by (nat-fold-s)} \\ \longrightarrow^* & (\lambda y.\texttt{s}(y))\ (\texttt{nat-fold } \underbrace{\texttt{s}(\ldots\texttt{s}}_{(i)\ \texttt{s}\text{'s}}(\texttt{z})\ldots)\ e_2\ (\lambda x.\lambda y.\texttt{s}(y))) & \text{by (app)} \\ \longrightarrow^* & (\lambda y.\texttt{s}(y))\ \underbrace{\texttt{s}(\ldots\texttt{s}}_{i\ \texttt{s}\text{'s}}(\texttt{nat-fold z } e_2\ (\lambda x.\lambda y.\texttt{s}(y)))\ldots) & \text{by inductive hypothesis} \\ \longrightarrow^* & \underbrace{\texttt{s}(\ldots\texttt{s}}_{(i+1)\ \texttt{s}\text{'s}}(\texttt{nat-fold z } e_2\ (\lambda x.\lambda y.\texttt{s}(y)))\ldots) & \text{by (app)} \end{array}$$

$\square$

Proposition 19. $\texttt{eval}(\texttt{s}(\texttt{z}) + \texttt{s}(\texttt{s}(\texttt{z}))) = \texttt{s}(\texttt{s}(\texttt{s}(\texttt{z})))$.

Proof. By (eval) and by the definition of +, it is sufficient to show

$$\texttt{nat-fold}\ \texttt{s}(\texttt{z})\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{s}(\texttt{s}(\texttt{z})))$$

By Lemma 18,

$$\texttt{nat-fold}\ \texttt{s}(\texttt{z})\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{nat-fold}\ \texttt{z}\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)))$$

By (nat-fold-0),

$$\texttt{nat-fold}\ \texttt{z}\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow \texttt{s}(\texttt{s}(\texttt{z}))$$

By (suc-cp-b1),

$$\texttt{s}(\texttt{nat-fold}\ \texttt{z}\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y))) \longrightarrow^1 \texttt{s}(\texttt{s}(\texttt{s}(\texttt{z})))$$

Thus,

$$\texttt{nat-fold}\ \texttt{s}(\texttt{z})\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{s}(\texttt{s}(\texttt{z})))$$

$\square$

Proposition 20. $\texttt{eval}(\texttt{s}(\texttt{s}(\texttt{z})) + \texttt{s}(\texttt{s}(\texttt{z}))) = \texttt{s}(\texttt{s}(\texttt{s}(\texttt{s}(\texttt{z}))))$.

Proof. By (eval) and by the definition of +, it is sufficient to show

$$\texttt{nat-fold}\ \texttt{s}(\texttt{s}(\texttt{z}))\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{s}(\texttt{s}(\texttt{s}(\texttt{z}))))$$

By Lemma 18,

$$\texttt{nat-fold}\ \texttt{s}(\texttt{s}(\texttt{z}))\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{s}(\texttt{nat-fold}\ \texttt{z}\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y))))$$

By (nat-fold-0),

$$\texttt{nat-fold}\ \texttt{z}\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow \texttt{s}(\texttt{s}(\texttt{z}))$$

By (suc-cp-b1),

$$\texttt{s}(\texttt{s}(\texttt{nat-fold}\ \texttt{z}\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)))) \longrightarrow^1 \texttt{s}(\texttt{s}(\texttt{s}(\texttt{s}(\texttt{z}))))$$

Thus,

$$\texttt{nat-fold}\ \texttt{s}(\texttt{s}(\texttt{z}))\ \texttt{s}(\texttt{s}(\texttt{z}))\ (\lambda x.\lambda y.\texttt{s}(y)) \longrightarrow^* \texttt{s}(\texttt{s}(\texttt{s}(\texttt{s}(\texttt{z})))$$

$\square$