COMP527 Final Project - On Paper Part

1 Formalization

$$
\begin{array}{rlll}
\textbf{Types} & \tau & ::= & \mathsf{nat} \mid \tau_1 \to \tau_2 \\
\textbf{Expressions:} & e & ::= & x & \text{variable} \\
& & \mid & z & \text{zero} \\
& & \mid & \mathsf{succ}(e) & \text{successor} \\
& & \mid & \mathsf{rec}(e; e_0; x.y.e_1) & \text{recursion} \\
& & \mid & \lambda(x:\tau).\, e & \text{abstraction} \\
& & \mid & \mathsf{app}(e_1, e_2) \text{ (written as } e_1(e_0)) & \text{application}
\end{array}
$$

Typing Rules :

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}$$

$$\frac{}{\Gamma \vdash z : \mathsf{nat}}$$

$$\frac{\Gamma \vdash e : \mathsf{nat}}{\Gamma \vdash \mathsf{succ}(e) : \mathsf{nat}}$$

$$\frac{\Gamma \vdash e : \mathsf{nat} \qquad \Gamma \vdash e_0 : \tau \qquad \Gamma, x : \mathsf{nat}, y : \tau \vdash e_1 : \tau}{\Gamma \vdash \mathsf{rec}(e; e_0; x.y.e_1) : \tau}$$

$$\frac{\Gamma,\ x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda(x : \tau_1).\ e : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash \mathsf{app}(e_1, e_2) : \tau_2}$$

Reduction Rules :

$$\lambda(x : \tau.\, t).\, u \overset{\beta}{\to}\ t[x := u]$$

$$\mathsf{rec}(z;\ t_0;\ x.y.\, t_s) \to\ t_0$$

$$\mathsf{rec}(\mathsf{succ}(n);\ t_0;\ x.y.\, t_s) \to\ t_s[x := n,\ y := \mathsf{rec}(n;\ t_0;\ x.y.\, t_s)]$$

$$\frac{e\ \to\ e'}{C[e]\ \to\ C[e']} \quad \text{where:}$$

$$C\ ::= [\,] \mid z \mid \mathsf{succ}(C) \mid C\, e \mid e\, C \mid \mathsf{rec}\big(C;\ e_0;\ x.y.e_s\big)$$

Examples

$$\begin{aligned}
\mathsf{succ} &= \lambda(n : \mathsf{nat}).\ \mathsf{succ}(n) \\
\mathsf{pred} &= \lambda(n : \mathsf{nat}).\ \mathsf{rec}(n;\ z;\ x.y.\ x) \\
\mathsf{add} &= \lambda(m : \mathsf{nat}).\ \lambda(n : \mathsf{nat}).\ \mathsf{rec}(m;\ n;\ x.y.\ \mathsf{succ}(y)) \\
\mathsf{subtract} &= \lambda(m : \mathsf{nat}).\ \lambda(n : \mathsf{nat})\ \mathsf{rec}(n;\ m;\ x.y.\ \mathsf{pred}(y)) \\
\mathsf{multiply} &= \lambda(m : \mathsf{nat}).\ \lambda(n : \mathsf{nat}).\ \mathsf{rec}(m;\ z;\ x.y.\ \mathsf{add}(n, y)) \\
\mathsf{exp} &= \lambda(m : \mathsf{nat}).\ \lambda(n : \mathsf{nat}).\ \mathsf{rec}(n;\ \mathsf{succ}(z);\ x.y.\ \mathsf{multiply}(m, y))
\end{aligned}$$

Proofs of Examples

1. Successor:

$$\begin{aligned}
\mathsf{succ}\,(\mathsf{succ}(\mathsf{succ}(z))) &= (\lambda(x : \mathsf{nat}).\, \mathsf{succ}(x))\,(\mathsf{succ}(\mathsf{succ}(z))) \\
&\xrightarrow{\beta} \mathsf{succ}\big(\mathsf{succ}(\mathsf{succ}(z))\big)
\end{aligned}$$

2. Predecessor:

$$\begin{aligned}
\mathsf{pred}\,(\mathsf{succ}(\mathsf{succ}(z))) &= (\lambda(x : \mathsf{nat}).\, \mathsf{rec}(x;\ z;\ x.y.\ x))\,(\mathsf{succ}(\mathsf{succ}(z))) \\
&\xrightarrow{\beta} \mathsf{rec}\big(\mathsf{succ}(\mathsf{succ}(z));\ z;\ x.y.\ x\big) \\
&\xrightarrow{\text{red-}s} \mathsf{rec}\big(\mathsf{succ}(z);\ z;\ x.y.\ x\big) \\
&\xrightarrow{\text{red-}s} \mathsf{rec}\big(z; z;\ x.y.\ x\big) \\
&\xrightarrow{\text{red-}0} z
\end{aligned}$$

3. Addition:

$$\begin{aligned}
\mathsf{add}\big(\mathsf{succ}(\mathsf{succ}(z)),\ \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(z)))\big) &= (\lambda m\, n.\, \mathsf{rec}(m;\ n;\ x.y.\ \mathsf{succ}(y)))\,(\mathsf{succ}(\mathsf{succ}(z))) \\
&\quad (\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(z)))) \\
&\xrightarrow{\beta} \mathsf{rec}\big(2;\ 3;\ x.y.\ \mathsf{succ}(y)\big) \\
&\xrightarrow{\text{red-}s} s\big(\mathsf{rec}(1;\ 3;\ x.y.\ \mathsf{succ}(y))\big) \\
&\xrightarrow{\text{red-}s} s\big(\mathsf{succ}(\mathsf{rec}(0;\ 3;\ x.y.\ \mathsf{succ}(y)))\big) \\
&\xrightarrow{\text{red-}0} \mathsf{succ}\big(\mathsf{succ}(3)\big) \ \rightarrow\ \mathsf{succ}(4) \ \rightarrow\ 5
\end{aligned}$$

4. Subtraction:

$$\begin{aligned}
\mathsf{subtract}\big(\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(z))),\ \mathsf{succ}(\mathsf{succ}(z))\big) &= (\lambda m\, n.\, \mathsf{rec}(n;\ m;\ x.y.\ \mathsf{pred}(y)))\,(\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(z)))) \\
&\quad (\mathsf{succ}(\mathsf{succ}(z))) \\
&\xrightarrow{\beta} \mathsf{rec}\big(2;\ 3;\ x.y.\ \mathsf{pred}(y)\big) \\
&\xrightarrow{\text{red-}s} \mathsf{pred}\big(\mathsf{rec}(1;\ 3;\ x.y.\ \mathsf{pred}(y))\big) \\
&\xrightarrow{\text{red-}s} \mathsf{pred}\big(\mathsf{pred}(\mathsf{rec}(0;\ 3;\ x.y.\ \mathsf{pred}(y)))\big) \\
&\xrightarrow{\text{red-}0} \mathsf{pred}(\mathsf{pred}(3)) \ \rightarrow\ \mathsf{pred}(3) = 2 \ \rightarrow\ \mathsf{pred}(2) = 1
\end{aligned}$$

5. Multiplication:

$$\begin{aligned}
\mathsf{multiply}\big(\mathsf{succ}(\mathsf{succ}(z)),\ \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(z)))\big) &= (\lambda m\, n.\, \mathsf{rec}(m;\ z;\ x.y.\, \mathsf{add}(n, y)))\ (\mathsf{succ}(\mathsf{succ}(z)))\\
&\quad (\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(z))))\\
&\xrightarrow{\beta} \mathsf{rec}\big(2;\ 0;\ x.y.\, \mathsf{add}(3, y)\big)\\
&\xrightarrow{\mathsf{red}\text{-}s} \mathsf{add}\big(3,\ \mathsf{rec}(1;\ 0;\ x.y.\, \mathsf{add}(3, y))\big)\\
&\xrightarrow{\mathsf{red}\text{-}s} \mathsf{add}\big(3,\ \mathsf{add}(3,\ \mathsf{rec}(0;\ 0; \dots))\big)\\
&\xrightarrow{\mathsf{red}\text{-}0} \mathsf{add}(3,\ \mathsf{add}(3, 0))\ =\ \mathsf{add}(3, 3) = 6
\end{aligned}$$

6. Exponentiation:

$$\begin{aligned}
\mathsf{exp}\big(\mathsf{succ}(\mathsf{succ}(z)),\ \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(z)))\big) &= (\lambda m\, n.\, \mathsf{rec}(n;\ \mathsf{succ}(z);\ x.y.\, \mathsf{multiply}(m, y)))\ (\mathsf{succ}(\mathsf{succ}(z)))\\
&\quad (\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(z))))\\
&\xrightarrow{\beta} \mathsf{rec}\big(3;\ 1;\ x.y.\, \mathsf{multiply}(2, y)\big)\\
&\xrightarrow{\mathsf{red}\text{-}s} \mathsf{multiply}\big(2,\ \mathsf{rec}(2;\ 1;\ x.y.\, \mathsf{multiply}(2, y))\big)\\
&\xrightarrow{\mathsf{red}\text{-}s} \mathsf{multiply}\big(2,\ \mathsf{multiply}(2,\ \mathsf{rec}(1;\ 1; \dots))\big)\\
&\xrightarrow{\mathsf{red}\text{-}s} \mathsf{multiply}\big(2,\ \mathsf{multiply}(2,\ \mathsf{multiply}(2,\ \mathsf{rec}(0;\ 1; \dots)))\big)\\
&\xrightarrow{\mathsf{red}\text{-}0} \mathsf{multiply}(2,\ \mathsf{multiply}(2,\ \mathsf{multiply}(2, 1))) = 8
\end{aligned}$$

2 Definitions and Theorems

Definition 1 (Normal Form)**.** *A term e is in* normal form *if there is no term e' such that*

$$e \ \to\ e'.$$

In other words, no reduction rule applies to e.

Definition 2 (Weak Normalization)**.** *A (closed) term e is* weakly normalizing*, written* $\mathsf{WN}(e)$*, if there exists a finite reduction sequence*

$$e \ \to\ e_1 \ \to\ \cdots \ \to\ e_n$$

such that e_n is in normal form.

Definition 3 (Reducibility Relation)**.** *For each type τ we define the predicate $R_\tau(e)$ on closed terms e by induction on τ:*

$$R_{\mathsf{nat}}(e) \ :=\ \big(\vdash e : \mathsf{nat}\big)\ \wedge\ \mathsf{WN}(e), \tag{1}$$

$$R_{\sigma\to\tau}(e) \ :=\ \big(\vdash e : \sigma \to \tau\big)\ \wedge\ \big(\forall\, v.\ R_\sigma(v)\ \to\ R_\tau(e\,(v))\big). \tag{2}$$

We often write

$$\mathcal{R}(\tau)\ =\ \{\, e \mid R_\tau(e)\,\}.$$

Theorem 1 (Confluence)**.**

$$\frac{e\ \to^*\ e_1 \qquad e\ \to^*\ e_2}{\exists e_3.\ e_1\ \to^*\ e_3\ \wedge\ e_2\ \to^*\ e_3}$$

3 Proof of Weak Normalization

Lemma 1.

$$e \to e' \,\wedge\, \mathsf{WN}(e) \,\Rightarrow\, \mathsf{WN}(e').$$

Proof. Suppose $e \to e'$ and $\mathsf{WN}(e)$, so there is n with $e \to^* n$ and n in normal form. By confluence, there exists m such that

$$e' \;\to^*\; m \quad \text{and} \quad n \;\to^*\; m.$$

But n is normal, so $m = n$. Therefore $e' \to^* n$, i.e. $\mathsf{WN}(e')$. □

Lemma 2.

$$e \to e' \,\wedge\, \mathsf{WN}(e') \,\Rightarrow\, \mathsf{WN}(e).$$

Proof. By definition of WN, from $e \to e'$ and $e' \to^* n$ (with n normal) we get

$$e \;\to\; e' \;\to^*\; n,$$

i.e. $e \to^* n$, so $\mathsf{WN}(e)$. □

Lemma 3.

If $R_\tau(e)$ and $e \to e'$, then $R_\tau(e')$.

Proof. By induction on the definition of $R_\tau(e)$.

- $\tau = \mathsf{nat}$. Then

  $$R_{\mathsf{nat}}(e) = (\Gamma \vdash e : \mathsf{nat}) \,\wedge\, \mathsf{WN}(e).$$

  By lemma 1, if $e \to e'$ and $\mathsf{WN}(e)$ then also $\mathsf{WN}(e')$. Thus $R_{\mathsf{nat}}(e')$.

- $\tau = \sigma \to \rho$. We have

  $$R_{\sigma\to\rho}(e) \;=\; \Gamma \vdash e : \sigma \to \rho \,\wedge\, \mathsf{WN}(e) \,\wedge\, \forall v\,(R_\sigma(v) \implies R_\rho(e\,v)).$$

  Since if $e \to e'$, then for any $v \in R_\sigma$, $e\,v \to e'\,v$ (by reduction rule), thus for type ρ we get $R_\rho(e'\,v)$. Hence $R_{\sigma\to\rho}(e')$.

□

Lemma 4.

If $e \to e'$ and $R_\tau(e')$, then $R_\tau(e)$.

Proof. Similarly, by induction on the definition of $R_\tau(e)$.

- $\tau = \mathsf{nat}$. Then

  $$R_{\mathsf{nat}}(e') = (\Gamma \vdash e' : \mathsf{nat}) \,\wedge\, \mathsf{WN}(e').$$

  By lemma 2, if $e \to e'$ and $\mathsf{WN}(e')$ then also $\mathsf{WN}(e)$. Thus $R_{\mathsf{nat}}(e)$.

- $\tau = \sigma \to \rho$. We have

$$R_{\sigma\to\rho}(e') \;=\; \Gamma \vdash e : \sigma \to \rho \;\wedge\; \mathsf{WN}(e') \;\wedge\; \forall v\,(R_\sigma(v) \implies R_\rho(e'\,v)).$$

  Since if $e \to e'$, then for any $v \in R_\sigma$, $e\,v \to e'\,v$ (by reduction rule), thus for type ρ we get $R_\rho(e\,v)$. Hence $R_{\sigma\to\rho}(e)$.

$\square$

Lemma 5 (Fundamental Lemma). *If $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau$ and each v_i is a closed term with $R_{\tau_i}(v_i)$, then*

$$R_\tau\big(e[x_1 := v_1, \ldots, x_n := v_n]\big).$$

Proof. Prove by induction on $\Gamma \vdash e : \tau$.

- $e = z$. Then $(e[x_1 := v_1, \ldots, x_n := v_n])$=z, and z is normal by definition. Thus $R_\tau\big(e[x_1 := v_1, \ldots, x_n := v_n]\big)$.
- $e = x_i$. Then $(e[x_1 := v_1, \ldots, x_n := v_n]) = v_i$, and $R_{\tau_i}(v_i)$ holds by hypothesis. Thus $R_\tau\big(e[x_1 := v_1, \ldots, x_n := v_n]\big)$ holds.
- $e = \mathsf{succ}(e')$. Then $R_\tau(e')$ by definition, and $e[x_1 := v_1, \ldots, x_n := v_n] \to \mathsf{succ}(e'[x_1 := v_1, \ldots, x_n := v_n])$. Thus $R_\tau\big(\mathsf{succ}(e'[x_1 := v_1, \ldots, x_n := v_n])\big)$ by the system rules, which means $R_\tau\big(e[x_1 := v_1, \ldots, x_n := v_n]\big)$ holds.
- $e = \mathsf{rec}(n; e_0; x.y.e_1)$. As our assumptions, we have $\Gamma \vdash n : \mathsf{nat}$, $\Gamma \vdash e_0 : \tau$, $\Gamma, x : \mathsf{nat}, y : \tau \vdash e_1 : \tau$, and then we will prove by induction:
  - Base case:$(e = \mathsf{rec}(z; e_0; x.y.e_1)) \to e_0$. Hold by our assumption.
  - Inductive case: We have induction hypothesis $R_\tau\big(\mathsf{rec}(n; e_0; x.y.e_1)\big)$, and we want to show that $R_\tau\big(\mathsf{rec}(\mathsf{succ}(n); e_0; x.y.e_1)\big)$.
    By our reduction rules $\mathsf{rec}(\mathsf{succ}(n);\ e_0;\ x.y.e_1) \to\ \ e_1[x := n,\ y := \mathsf{rec}(n;\ e_0;\ x.y.e_1)]$, and thus from our assumptions we have $R_\tau\big(e_1[x := n,\ y := \mathsf{rec}(n;\ e_0;\ x.y.e_1)]\big)$.
- $e = \lambda(x : \tau).\, e'$. As $\Gamma,\ x : \tau \vdash e' : \tau'$ we can get $\Gamma \vdash \lambda x.e' : \tau \to \tau'$. From the reduction rules $\lambda(x : \tau.\, e').[x_1 := v_1, \ldots, x_n := v_n] \to e'[x_1 := v_1, \ldots, x_n := v_n]$, and we have the IH of $R_{\tau'}\big(e'[x_1 := v_1, \ldots, x_n := v_n]\big)$, and thus it holds.
- $e = \mathsf{app}(e_1, e_2) = e_1(e_2)$. As $\Gamma \vdash e_2 : \tau'$, $\Gamma \vdash e_1 : \tau' \to \tau$, and our IH that $R_{\tau'}(e_2[x_1 := v_1, \ldots, x_n := v_n])$ and $R_{\tau'\to\tau}(e_1[x_1 := v_1, \ldots, x_n := v_n])$, and thus by definition we have $R_\tau(e_1[x_1 := v_1, \ldots, x_n := v_n]\ e_2[x_1 := v_1, \ldots, x_n := v_n])$, which means $R_\tau(e_1 e_2)$ holds.

$\square$

Theorem 2 (Weak Normalization). *If $\vdash e : \tau$ and e is closed, then* $\mathsf{WN}(e)$.

Proof. By the Fundamental Lemma in the empty context, $R_\tau(e)$ holds. In particular, if $\tau = \mathsf{nat}$ then $\mathsf{WN}(e)$ by definition of R_{nat}. If τ is a function type, we have checked that e itself cannot be an infinite reducible head, so e must reach a normal form in finitely many steps. Therefore, weak normalization holds. $\square$