Queens' College Cambridge

Computation Theory

Alistair O'Brien

Department of Computer Science

April 17, 2021

Contents

1 Register Machines

Definition 1.0.1. (**Register Machine**) A *register machine* M is the pair $(\mathcal{R}, P)$ where $\mathcal{R} \subseteq \mathscr{R}$ and $\mathcal{R}$ is finite, and P is a program, a total function $P : \mathscr{L}_{\leq n} \to \mathscr{I}(\mathcal{R})$, where $I \in \mathscr{I}(\mathcal{R})$ is the set of $\mathcal{R}$-register instructions, defined by the grammar:

$$\begin{aligned} I \;\; \texttt{::=} \;\; & R^+ \to L \\ \texttt{|} \;\; & R^- \to L_1, L_2 \\ \texttt{|} \;\; & \texttt{HALT} \end{aligned}$$

where $R \in \mathcal{R}, L \in \mathscr{L}$, the set of labels.

- Programs P are often defined graphically.

Definition 1.0.2. (**Configuration**) A register machine *configuration* for the machine $M = (\mathcal{R}, P)$ is the pair (L, s) where $s : \mathcal{R} \to \mathbb{N}$ is a $\mathcal{R}$-store. The set of $\mathcal{R}$-configurations is denoted $\mathscr{C}(\mathcal{R})$.

- **Notation**: We write $R_i = x$ (in the configuration c) to denote $c = (L, s)$ with $s(R_i) = x$.
- The initial configuration is defined by $c_0 = (L_0, s)$ where s is the *initial* store.

Definition 1.0.3. (**Transition Relation**) The *transition relation* on the register machine $M = (\mathcal{R}, P)$, denoted $\longrightarrow_M : \mathscr{C}(\mathcal{R}) \leftrightarrow \mathscr{C}(\mathcal{R})$, is inductively defined by

$$\text{(Add)}\ \frac{P(L) = R^+ \to L'}{(L, s) \longrightarrow_M (L', s' \cup \{(R, s(R) + 1)\})}$$

$$\text{(Sub1)}\ \frac{P(L) = R^- \to L', L'' \qquad s(R) \neq 0}{(L, s) \longrightarrow_M (L', s' \cup \{(R, s(R) - 1)\})}\ \text{(Sub2)}\ \frac{P(L) = R^- \to L', L'' \qquad s(R) = 0}{(L, s) \longrightarrow_M (L'', s)}$$

where $s' = s \setminus \{R, s(R)\}$.

- **Notation**: $\longrightarrow_M^*$ denotes a sequence of transitions, the reflexive transitive closure of $\longrightarrow_M$.

Definition 1.0.4. (**Computation**) A *computation* of a register machine M is a sequence of transitions (infinite or finite)

$$c_0 \longrightarrow_M c_1 \longrightarrow_M \cdots ,$$

where $c_0 \in \mathcal{C}(\mathcal{R})$ is the *initial* configuration.

Definition 1.0.5. (**Halting**) A configuration $c = (L, s) \in \mathscr{C}(\mathcal{R})$ is said to be halting if $P(L) = \texttt{HALT}$, a *proper halt*, or $L \notin \mathscr{L}_{\leq n}$, an *erroneous halt*.

- For a finite computation $c_0 \longrightarrow_M^* c_m \not\longrightarrow_M$, c_m is a *halting configuration* by definition of $\longrightarrow_M$.
- A register machine M can be modified (without effecting the computation) to remove erroneous halts by adding additional `HALT` instructions.

Definition 1.0.6. (**Halting Computation**) A *halting computation* of a register machine M, denoted $(x_0, \ldots, x_n) \Downarrow_M (y_0, \ldots, y_n)$, where $\Downarrow_M : \mathbb{N}^n \rightharpoonup \mathbb{N}^n$ is defined as

$$(x_0, \ldots, x_n) \Downarrow_M (y_0, \ldots, y_n) \iff (L_0, s_0) \longrightarrow^* (L, s) \not\longrightarrow,$$

where $s_0(R_i) = x_i$ and $s(R_i) = y_i$ are $\mathcal{R}$-stores and $|\mathcal{R}| = n$.

- Arbitrary I/O convention: all other registers are initially set to 0

1.1 Computable Functions

Definition 1.1.1. (**Register Machine Computable**) $f \in \mathcal{P}\left[\mathbb{N}^n \rightharpoonup \mathbb{N}\right]$ is said to be *register machine computable* if there exists a register machine $M = (\mathcal{R}, P)$ such that $\{R_0, R_1, \ldots, R_n\} \subseteq \mathcal{R}$ and,

$$\forall (x_1, \ldots, x_n) \in \mathbb{N}^n, y \in \mathbb{N}. (0, x_1, \ldots, x_n, 0, \ldots) \Downarrow_M (y, 0, \ldots) \iff f(x_1, \ldots, x_n) = y.$$

- **Examples**:
  - TODO
- Derived instruction: $R \leftarrow R'$, copies R' into R:

$$\begin{tikzcd}
\texttt{START} \arrow[r] & R^- \arrow[loop below] \arrow[r, two heads] & R_t^- \arrow[loop below] \arrow[r, two heads] & R'^- \arrow[d] \arrow[r, two heads] & R_t^- \arrow[d] \arrow[r, two heads] & \texttt{HALT} \\
 & & & R^+ \arrow[d] & R'^+ \arrow[u, bend left] & \\
 & & & R_t^+ \arrow[uu, bend left] & &
\end{tikzcd}$$

- Computable functions $f \in \mathcal{P}\left[\mathbb{N}^n \rightharpoonup \mathbb{N}\right]$ with register machine $F = (\mathcal{R}_f, P_f)$ results in derived instruction, denoted $Y \leftarrow f(X_1, \ldots, X_n)$, given by

$$\texttt{START} \longrightarrow R_0, R_1, \ldots, R_n \leftarrow 0, X_1, \ldots, X_n \longrightarrow F \longrightarrow Y, R_0 \leftarrow R_0, 0 \longrightarrow \texttt{HALT}$$

  Calling Convention: All contents of registers $Y, \mathcal{R}_f$ (if used) are copied by the caller before the derived instruction is executed. Registers $\mathcal{R}_f \setminus \{R_0, \ldots, R_n\}$ are zeroed by the *caller*.

Definition 1.1.2. (**Composition**) The composition of $f \in \mathcal{P}\left[\mathbb{N}^n \rightharpoonup \mathbb{N}\right]$ with $g_1, \ldots, g_n \in \mathcal{P}\left[\mathbb{N}^m \rightharpoonup \mathbb{N}\right]$, denoted $f \circ \{g_1, \ldots, g_n\} : \mathbb{N}^m \rightharpoonup \mathbb{N}$, defined by

$$f \circ \{g_1, \ldots, g_n\}(\mathbf{x}) = f(g_1(\mathbf{x}), \ldots, g_n(\mathbf{x})),$$

where $\mathbf{x} \in \mathbb{N}^m$.

Theorem 1.1.1. If $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}]$ and $g_1, \ldots, g_n \in \mathcal{P}[\mathbb{N}^m \rightharpoonup \mathbb{N}]$ are computable, then $f \circ \{g_1, \ldots, g_n\} \in \mathcal{P}[\mathbb{N}^m \rightharpoonup \mathbb{N}]$ is computable.

Proof. Let $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}]$ and $g_1, \ldots, g_n \in \mathcal{P}[\mathbb{N}^m \rightharpoonup \mathbb{N}]$ be arbitrary partial functions on $\mathbb{N}$.

Let us assume that f and $g_1, \ldots, g_n$ are computable, that is to say there exists register machines $F = (\mathcal{R}_f, P_f)$ and $G_i = (\mathcal{R}_g^i, P_g^i)$, s.t

$$\forall (x_1, \ldots, x_n) \in \mathbb{N}^n, y \in \mathbb{N}.(0, x_1, \ldots, x_n, 0, \ldots) \Downarrow_F (y, 0, \ldots) \iff f(x_1, \ldots, x_n) = y$$
$$\forall (x_1, \ldots, x_n) \in \mathbb{N}^m, y \in \mathbb{N}.(0, x_1, \ldots, x_m, 0, \ldots) \Downarrow_{G_i} (y, 0, \ldots) \iff g_i(x_1, \ldots, x_m) = y$$

Let $\mathbf{R} = \left\{\mathcal{R}_f, \mathcal{R}_g^1, \ldots, \mathcal{R}_g^n\right\}$. Without loss of generality, we assume that

$$\begin{aligned} &\forall \mathcal{R}^i, \mathcal{R}^j \in \{\mathcal{R} \setminus \{R_0, \ldots, R_N\} \in \mathcal{P}(\mathscr{R}) : \mathcal{R} \in \mathbf{R}\}_{i \in \mathcal{I}} . \\ &\qquad i \neq j \implies \mathcal{R}^i \cap \mathcal{R}^j = \emptyset \end{aligned}$$

where $N = \max\{m, n\} \in \mathbb{N}$.

We wish to show that $f \circ \{g_1, \ldots, g_n\}$ is computable. We introduce the register machine $M = (\mathcal{R}, P)$, where

$$\mathcal{R} = \bigcup_{\mathcal{R} \in \mathbf{R}} \mathcal{R} \cup \{R_t, X_1, \ldots, X_m, Y_1, \ldots, Y_n\},$$

where $\{R_t, X_1, \ldots, X_m, Y_1, \ldots, Y_n\} \cap \mathcal{R} = \emptyset$ for all $\mathcal{R} \in \mathbf{R}$, with program P (in graphical form):

$$\begin{tikzcd}
\texttt{START} \arrow[d] & \\
X_1, \ldots, X_m \leftarrow R_1, \ldots, R_m \arrow[r] & R_1, \ldots, R_m \leftarrow 0 \arrow[dl] \\
Y_1 \leftarrow g_1(X_1, \ldots, X_m) \arrow[r] & Y_2 \leftarrow g_2(X_1, \ldots, X_m) \arrow[dl] \\
\cdots \arrow[r] & \cdots \arrow[dl] \\
Y_n \leftarrow g_n(X_1, \ldots, X_m) \arrow[r] & R_0 \leftarrow f(Y_1, \ldots, Y_n) \arrow[dl] \\
X_1, \ldots, X_n, Y_1, \ldots, Y_m \leftarrow 0, \ldots, 0 \arrow[r] & \texttt{HALT}
\end{tikzcd}$$

□

1.2 Partial Recursive Functions

1.2.1 Primitive Recursion

Definition 1.2.1. (**Primitive Recursion**) Let $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}], g \in \mathcal{P}[\mathbb{N}^{n+2} \rightharpoonup \mathbb{N}]$. The primitive recursive function from f and g is a function $h \in \mathcal{P}[\mathbb{N}^{n+1} \rightharpoonup \mathbb{N}]$ satisfying

$$\begin{aligned} h(\mathbf{x}, 0) &= f(\mathbf{x}) \\ h(\mathbf{x}, y) &= g(\mathbf{x}, y, h(\mathbf{x}, y)) \end{aligned}$$

where $\mathbf{x} \in \mathbb{N}^n, y \in \mathbb{N}$.

- **Notation**: $\rho^n(f, g)$ denotes the primitive recursive function from f and g.

Definition 1.2.2. (**Primitive Recursive Functions**) The class of *primitive recursive functions* is the set $\mathscr{P}_0 \in \mathcal{P}\left[\bigcup_k \mathbb{N}^k \to \mathbb{N}\right]$ inductively defined by

$\pi_i^n : \mathbb{N}^n \to \mathbb{N}$	$\pi_i^n(x_1, \ldots, x_n) = x_i$	(Proj) $\dfrac{}{\pi_i^n}$
$\text{zero}^n : \mathbb{N}^n \to \mathbb{N}$	$\text{zero}^n(\mathbf{x}) = 0$	(Zero) $\dfrac{}{\text{zero}^n}$
$\text{succ} : \mathbb{N} \to \mathbb{N}$	$\text{succ}(n) = n + 1$	(Succ) $\dfrac{}{\text{succ}}$
$f \circ [g_1, \ldots, g_m] : \mathbb{N}^n \to \mathbb{N}$ $g_i : \mathbb{N}^n \to \mathbb{N}$ $h : \mathbb{N}^m \to \mathbb{N}$	$f(\mathbf{x}) = h(g_1(\mathbf{x}), \ldots, g_m(\mathbf{x}))$	(Comp) $\dfrac{\forall 1 \le i \le m. g_i \qquad h}{f \circ [g_1, \ldots, g_m]}$
$f : \mathbb{N}^n \to \mathbb{N}, g : \mathbb{N}^{n-1} \to \mathbb{N}$ $h : \mathbb{N}^{n+1} \to \mathbb{N}$	$f(\mathbf{x}, 0) = g(\mathbf{x})$ $f(\mathbf{x}, y+1) = h(\mathbf{x}, y, f(\mathbf{x}, y))$	(Rec) $\dfrac{g \qquad h}{f}$

Theorem 1.2.1. All primitive recursive functions $f \in \mathscr{P}_0$ are RM computable.

Proof. We proceed by induction on the definition of $\mathscr{P}_0$, with the statement

$$P(f) = f \text{ is RM computable.}$$

Base Case: For the axiom: $\dfrac{}{\pi_i^n}$, we have the following register machine $M = (\{R_0, R_1, \ldots, R_n\}, P)$ with program P:

$$\begin{tikzcd}
\texttt{START} \arrow[r] & R_i^- \arrow[r, two heads] \arrow[d, shift left] & R_1^- \arrow[r, two heads] \arrow[loop below] & \cdots \arrow[r, two heads] & R_n^- \arrow[r, two heads] \arrow[loop below] & \texttt{HALT} \\
& R_0^+ \arrow[u, bend left] & & & &
\end{tikzcd}$$

such that M computes π_i^n. So we have $P(\pi_i^n)$.
Similar arguments are given for zero^n*, succ*

Inductive Step: For the rule $\dfrac{\forall 1 \le i \le m. g_i \qquad f}{f \circ [g_1, \ldots, g_m]}$, we wish to show that $(\forall 1 \le i \le m. P(g_i)) \land P(f) \implies P(f \circ \{g_1, \ldots, g_m\})$. Let us assume that $P(g_1), \ldots, P(g_m), P(f)$ hold. Then by theorem ??, we have $P(f \circ \{g_1, \ldots, g_m\})$.

For the rule $\dfrac{g \quad h}{\rho^n(g,h)}$, we wish to show $P(g) \wedge P(h) \implies P(\rho^n(g,h))$. Let us assume that $P(g)$ and $P(h)$ holds, that is to say there exists register machines $G = (\mathcal{R}_g, P_g)$ and $H = (\mathcal{R}_h, P_h)$ s.t

$$\forall (x_1, \ldots, x_n) \in \mathbb{N}^n, y \in \mathbb{N}.(0, x_1, \ldots, x_n, 0, \ldots) \Downarrow_F (y, 0, \ldots) \iff g(\underbrace{x_1, \ldots, x_n}_{\mathbf{x}}) = y$$

$$\forall (x_1, \ldots, x_n) \in \mathbb{N}^n, c, y_h, y \in \mathbb{N}.(0, x_1, \ldots, x_n, c, y_h, 0, \ldots) \Downarrow_H (y, 0, \ldots) \iff h(\underbrace{x_1, \ldots, x_n}_{\mathbf{x}}, c, y_h) = y$$

Let $\mathbf{R} = \{\mathcal{R}_f, \mathcal{R}_g\}$.Without loss of generality, we assume that

$$\begin{aligned} &\forall \mathcal{R}^i, \mathcal{R}^j \in \{\mathcal{R} \setminus \{R_0, \ldots, R_{n+2}\} \in \mathcal{P}(\mathscr{R}) : \mathcal{R} \in \mathbf{R}\}_{i \in \mathcal{I}} . \\ &\qquad i \neq j \implies \mathcal{R}^i \cap \mathcal{R}^j = \emptyset \end{aligned}$$

We wish to show that $\rho^n(g,h)$ is computable. We introduce the register machine $M = (\mathcal{R}, P)$, where

$$\mathcal{R} = \bigcup_{\mathcal{R} \in \mathbf{R}} \mathcal{R} \cup \{R_t, X_1, \ldots, X_{n+1}, C, Y_h\},$$

where $\{R_t, X_1, \ldots, X_{n+1}, C, Y_h\} \cap \mathcal{R}$ for all $\mathcal{R} \in \mathbf{R}$. and P (in graphical form) is given by:

$$\begin{tikzcd}
\texttt{START} \arrow[r] & X_1, \ldots, X_{n+1}, C, Y \leftarrow R_1, \ldots, R_{n+1}, 0, 0 \arrow[d] & & & \\
& R_0, \ldots, R_{n+2} \leftarrow 0 \arrow[r] & Y \leftarrow f(X_1, \ldots, X_n) \arrow[d] & & \\
& & \text{if } C = X_{n+1} \arrow[r, two heads] \arrow[d] & R_0 \leftarrow Y \arrow[r, two heads] & \texttt{HALT} \\
& C^+ \arrow[ur] & Y \leftarrow g(X_1, \ldots, X_n, C, Y) \arrow[l] & &
\end{tikzcd}$$

such that M computes $\rho^n(g,h)$. So we have $P(\rho^n(g,h))$.

By the Principle of Rule Induction, we conclude that the statement $P(f)$ holds for all $f \in \mathscr{P}_0$. $\square$

1.2.2 Minimization

- **Problem**: Primitive recursion provides a *bounded recursion* $\implies \mathscr{P}_0$ is not equivalent to the set of RM computable functions.
- **Solution**: *Minimization*

Definition 1.2.3. (**Minimization**) Let $f : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ be a partial function. The *minimization* (or *unbounded search*) $\mu^n f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ s.t $\mu^n f(\mathbf{x}) = y$ where $\forall x < y. f(\mathbf{x}, y) \downarrow \wedge f(\mathbf{x}, y) > 0$, hence y is the *least* y.

Definition 1.2.4. (**Partial Recursive Functions**) The class of *partial recursive functions* is the set $\mathscr{P}_1 \in \mathcal{P}\left[\bigcup_k \mathbb{N}^k \to \mathbb{N}\right]$ inductively defined by

$\pi_i^n : \mathbb{N}^n \to \mathbb{N}$	$\pi_i^n(x_1, \dots, x_n) = x_i$	(Proj) $\dfrac{}{\pi_i^n}$
$\text{zero}^n : \mathbb{N}^n \to \mathbb{N}$	$\text{zero}^n(\mathbf{x}) = 0$	(Zero) $\dfrac{}{\text{zero}^n}$
$\text{succ} : \mathbb{N} \to \mathbb{N}$	$\text{succ}(n) = n + 1$	(Succ) $\dfrac{}{\text{succ}}$
$f \circ [g_1, \dots, g_m] : \mathbb{N}^n \to \mathbb{N}$ $g_i : \mathbb{N}^n \to \mathbb{N}$ $h : \mathbb{N}^m \to \mathbb{N}$	$f(\mathbf{x}) = h(g_1(\mathbf{x}), \dots, g_m(\mathbf{x}))$	(Comp) $\dfrac{\forall 1 \le i \le m. g_i \qquad h}{f \circ [g_1, \dots, g_m]}$
$f : \mathbb{N}^n \to \mathbb{N}, g : \mathbb{N}^{n-1} \to \mathbb{N}$ $h : \mathbb{N}^{n+1} \to \mathbb{N}$	$f(\mathbf{x}, 0) = g(\mathbf{x})$ $f(\mathbf{x}, y + 1) = h(\mathbf{x}, y, f(\mathbf{x}, y))$	(Rec) $\dfrac{g \qquad h}{f}$
$f : \mathbb{N}^n \to \mathbb{N}, g : \mathbb{N}^{n+1} \to \mathbb{N}$	$f = \mu^n g$	(μ) $\dfrac{g}{f}$

- $\mathscr{P}_0 \subset \mathscr{P}_1$.

Theorem 1.2.2. All partial recursive functions $f \in \mathscr{P}_1$ are RM computable.

Proof. We proceed by induction on the definition of $\mathscr{P}_1$, with the statement

$$P(f) = f \text{ is RM computable.}$$

Base Case: See theorem ??.

Inductive Step: For the rules $\dfrac{\forall 1 \leq i \leq m . g_i \qquad h}{f \circ [g_1, \dots, g_m]}$ and $\dfrac{g \qquad h}{\rho^n(g,h)}$ see theorem ??.

For the rule $\dfrac{g}{\mu^n g}$, we wish to show $P(g) \implies P(\mu^n g)$. Let us assume that $P(g)$ holds, that is to say there exists a register machine $G = (\mathcal{R}_g, P_g)$ s.t

$$\forall (x_1, \dots, x_n, x_{n+1}) \in \mathbb{N}^{n+1}, y \in \mathbb{N}. (0, x_1, \dots, x_{n+1}, 0, \dots) \Downarrow_G (y, \dots) \iff g(x_1, \dots, x_{n+1}) = y.$$

We introduce the register machine $M = (\mathcal{R}, P)$, where $\mathcal{R} = \mathcal{R}_g \cup \{R_t, X_1, \dots, X_n, C\}$ s.t $\{R_t, X_1, \dots, X_n, C, Y\} \cap \mathcal{R}_g = \emptyset$ and P (in graphical form) is given by:

$$\begin{tikzcd}
\texttt{START} \arrow[d] & & & & \\
X_1, \dots, X_n, C, Y \leftarrow R_1, \dots, R_n, 0, 0 \arrow[rr] & & Y \leftarrow f(X_1, \dots, X_n, C) \arrow[d] & & \\
& C^+ \arrow[ur] & Y^- \arrow[l] \arrow[r, two heads] & R_0 \leftarrow C \arrow[r] & \texttt{HALT}
\end{tikzcd}$$

□

Theorem 1.2.3. All register machine computable functions $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}]$ are partial recursive, that is $f \in \mathscr{P}_1$.

Proof. Let $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}]$ be an arbitrary partial function.

Let us assume that f is RM computable, that is to say there exists a register machine $F = (\mathcal{R}_f, P_f)$ s.t

$$\forall (x_1, \dots, x_n) \in \mathbb{N}^n, y \in \mathbb{N}. (0, x_1, \dots, x_n, \dots) \Downarrow_F (y, 0, \dots) \iff f(x_1, \dots, x_n) = y.$$

Without loss of generality, assume $\mathcal{R} = \{R_0, \dots, R_N\}$, where $N \geq n$.

We define the following encoding for $\mathcal{R}$-stores:

$$\mathcal{E}[\![s]\!]_s^{\mathbb{N}} = \mathcal{E}[\![[s(R_0), \dots, s(R_n)]]\!]_\ell^{\mathbb{N}},$$

with it's decoding function

$$\mathcal{D}[\![e]\!]_s^{\mathbb{N}} = s,$$

where $\mathcal{D}[\![e]\!]_\ell^{\mathbb{N}} = [x_0, \dots, x_n]$ and $s(R_i) = x_i$.

We define the following partial-recursive functions

$$\text{value}_i(\mathcal{E}\,[\![s]\!]_s^{\mathbb{N}}) = s(R_i)$$

TODO $\square$

1.3 Universal Register Machines

- **Idea**: Register machine U that computes register machines.

1.3.1 Program Encodings

- **Problem**: Register machine programs P encoded by natural numbers $\mathbb{N}$

Definition 1.3.1. Let $\langle\!\langle \cdot, \cdot \rangle\!\rangle : \mathbb{N}^2 \to \mathbb{N}_{>0}$ and $\langle \cdot, \cdot \rangle : \mathbb{N}^2 \to \mathbb{N}$, defined by

$$\begin{aligned} \langle\!\langle x, y \rangle\!\rangle &= 2^x(2y+1) \\ \langle x, y \rangle &= 2^x(2y+1) - 1 \end{aligned}$$

Lemma 1.3.1. $\langle\!\langle \cdot, \cdot \rangle\!\rangle$ and $\langle \cdot, \cdot \rangle$ are bijections.

- Binary representations are given by

$$\text{bin}\left(\langle\!\langle x, y \rangle\!\rangle\right) = \text{bin}(y)1\underbrace{0\cdots 0}_{x-\text{times}} \qquad \text{bin}\left(\langle x, y \rangle\right) = \text{bin}(y)0\underbrace{1\cdots 1}_{x-\text{times}}$$

Lemma 1.3.2. $\langle\!\langle \cdot, \cdot \rangle\!\rangle$ and $\langle\!\langle \cdot \rangle\!\rangle^{-1}$ are register machine computable.

Proof. For $\langle\!\langle \cdot, \cdot \rangle\!\rangle$, we have the register machine $M = (\mathcal{R}, P)$ with $\mathcal{R} = \{R_0, R_x, R_y\}$ with program $\mathcal{P}$ (in graphical form):

$$\begin{tikzcd}
\texttt{START} \arrow[r] & R_0^+ \arrow[r] & R_y^- \arrow[r, two heads] \arrow[d] & R_0^- \arrow[r, two heads] \arrow[d, bend left] & R_x^- \arrow[r, two heads] \arrow[ll, bend right] & R_y^- \arrow[r, two heads] \arrow[d, bend left] & \texttt{HALT} \\
& & R_0^+ \arrow[ul] & R_y^+ \arrow[u, bend left] & & R_0^+ \arrow[u, bend left] &
\end{tikzcd}$$

For $\langle\!\langle\cdot\rangle\!\rangle^{-1}$, we have the register machine $M = (\mathcal{R}, P)$ with $\mathcal{R} = \{R_x, R_y, R_1\}$ with program P (in graphical form):

$$\begin{tikzcd}
& \texttt{EXIT} & & & \\
\texttt{START} \arrow[r] & R_1^- \arrow[u, two heads] \arrow[r] & R_1^+ \arrow[d] & & \\
R_x^+ \arrow[r] & R_y^- \arrow[r, two heads] \arrow[d, bend left] & R_1^- \arrow[r] \arrow[ll, bend right] & R_1^- \arrow[r, two heads] \arrow[dl] & \texttt{HALT} \\
& R_1^+ \arrow[u, bend left] & R_y^+ \arrow[u] & &
\end{tikzcd}$$

$\square$

Corollary 1.3.0.1. $\langle\cdot,\cdot\rangle$ and $\langle\cdot\rangle^{-1}$ are register machine computable.

- **Idea**: Encode instructions as naturals $\mathbb{N}$

Definition 1.3.2. For $\mathcal{R}$-register instructions $\mathscr{I}(\mathcal{R})$, the encoding function $\mathcal{E}\,[\![\cdot]\!]_I^{\mathbb{N}} : \mathscr{I}(\mathcal{R}) \to \mathbb{N}$, is defined on the structure of $\mathscr{I}$ as

$$\begin{aligned}
\mathcal{E}\left[\!\left[R_i^+ \to L_j\right]\!\right]_I^{\mathbb{N}} &= \langle\!\langle 2i, j\rangle\!\rangle \\
\mathcal{E}\left[\!\left[R_i^- \to L_j, L_k\right]\!\right]_I^{\mathbb{N}} &= \langle\!\langle 2i+1, \langle j, k\rangle\rangle\!\rangle \\
\mathcal{E}\,[\![\mathrm{HALT}]\!]_I^{\mathbb{N}} &= 0
\end{aligned}$$

where $R_i \in \mathcal{R}$.

Corollary 1.3.0.2. $\mathcal{E}\,[\![\cdot]\!]_I^{\mathbb{N}} : \mathscr{I}(\mathscr{R}) \to \mathbb{N}$ is bijective. It's inverse, the decoding function is denoted $\mathcal{D}\,[\![\cdot]\!]_\ell^{\mathbb{N}} : \mathbb{N} \to \mathscr{I}(\mathscr{R})$.

Definition 1.3.3. (**Lists**) The set of lists, denoted **Lists**$(\mathbb{A})$, on $\mathbb{A}$ is defined

$$\ell \ ::= \ [] \ \mid \ a \ :: \ \ell$$

where $a \in \mathbb{A}$.

- **Notation**: $[a_1, \ldots, a_n] = a_1 :: (a_2 :: (\cdots a_n :: []) \cdots)$

- **Idea**: Encode programs as lists on $\mathbb{N}$.

Definition 1.3.4. For a list $\ell \in \mathbf{Lists}(\mathbb{N})$, the encoding function $\mathcal{E}\,[\![\cdot]\!]_\ell^{\mathbb{N}} : \mathbf{Lists}(\mathbb{N}) \to \mathbb{N}$, is inductively defined:

$$\begin{aligned} \mathcal{E}\,[\![[]]\!]_\ell^{\mathbb{N}} &= 0 \\ \mathcal{E}\,[\![n :: \ell]\!]_\ell^{\mathbb{N}} &= \langle\!\langle n, \mathcal{E}\,[\![\ell]\!]_\ell^{\mathbb{N}} \rangle\!\rangle \end{aligned}$$

Lemma 1.3.3. $\mathcal{E}\,[\![\cdot]\!]_\ell^{\mathbb{N}} : \mathbf{Lists}(\mathbb{N}) \to \mathbb{N}$ is bijective. It's inverse, the decoding function is denoted $\mathcal{D}\,[\![\cdot]\!]_\ell^{\mathbb{N}} : \mathbb{N} \to \mathbf{Lists}(\mathbb{N})$.

- Binary representation is given by

$$\mathrm{bin}\left(\mathcal{E}\,[\![[x_1, \ldots, x_n]]\!]_\ell^{\mathbb{N}}\right) = 1\underbrace{0\ldots0}_{x_n}1\underbrace{0\ldots0}_{x_{n-1}}\cdots 1\underbrace{0\ldots0}_{x_1}.$$

Definition 1.3.5. For a program $P : \mathscr{L}_{\leq n} \to \mathscr{I}(\mathcal{R})$, where $P(L_i) = I_i$. We define the encoding function $\mathcal{E}\,[\![\cdot]\!]_P^{\mathbb{N}} : \mathscr{P}_n(\mathcal{R}) \to \mathbb{N}$ is defined as

$$\mathcal{E}\,[\![P]\!]_P^{\mathbb{N}} = \mathcal{E}\left[\!\!\left[[\mathcal{E}\,[\![I_0]\!]_I^{\mathbb{N}}, \mathcal{E}\,[\![I_1]\!]_I^{\mathbb{N}}, \ldots, \mathcal{E}\,[\![I_n]\!]_I^{\mathbb{N}}]\right]\!\!\right]_\ell^{\mathbb{N}}.$$

It's decoding function $\mathcal{D}\,[\![\cdot]\!]_P^{\mathbb{N}} : \mathbb{N} \to \mathscr{P}_n(\mathcal{R})$ is defined as

$$\mathcal{D}\,[\![e]\!]_P^{\mathbb{N}} = P,$$

where $\mathcal{D}\,[\![e]\!]_\ell^{\mathbb{N}} = [x_0, \ldots, x_n]$, and $P(L_i) = \mathcal{D}\,[\![x_i]\!]_I^{\mathbb{N}}$.

1.3.2 Universal Register Machine U

- Universal register machine is the partial function $U : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ where $\varphi_e = f$, s.t $\varphi_e(x) = U(e, x)$ and $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is the partial computable function w/ program P_f s.t $e = \mathcal{E}\,[\![P_f]\!]_P$.
- Universal register machine pseudocode:
  1. $I \leftarrow \mathcal{D}\,[\![e]\!]_\ell[\mathtt{PC}]$. Stores the current instruction (encoded) in the register I.
  2. Check whether the current instruction is a `HALT`. If so, store R_0 (in the context of s) in R_0.

```
if (I = 0) {
    R_0 ← D[[s]]_ℓ[0];
    HALT;
}
```

3. Decode instruction I into type T and component U: $T, U \leftarrow \langle\!\langle I \rangle\!\rangle^{-1}$. If $T = 2i$ (even) then current instruction is $R_i^+ \to L_u$, or $T = 2i+1$ (odd) then current instruction is $R_i^- \to L_j, L_k$ where $U = \langle j, k \rangle$.
4. Compute $i \leftarrow \left\lfloor \frac{T}{2} \right\rfloor$. Fetch current value of R_i (in the context of s), store in R: $R \leftarrow \mathcal{D}[\![s]\!]_\ell[i]$
5. Execute I (using T, U) on R:

```
execute(T, U, R) {
    j, k = ⟨U⟩^{-1};

    return T is even
        ? R + 1, U
        : (R = 0
        ? R, k
        : R − 1, j
        );
}
```

Update the store w/ the new value of R_i: `update(s, i, R)`. Then `GOTO 1`.

Theorem 1.3.1. (**Computability of U**)
IMAGE

- The map $e \mapsto \varphi_e$ allows us to *index* or *enumerate* the set of computable functions $f : \mathbb{N} \rightharpoonup \mathbb{N}$. Thus there are $\aleph_0$ computable functions.

1.4 Decidability

1.4.1 Register Machine Decidability

Definition 1.4.1. (**Register Machine Decidable**) A set $S \subseteq \mathbb{N}$ is *register machine decidable* if the characteristic function $\chi_S : \mathbb{N} \to \{0, 1\}$ is *register machine computable.*

- There are $2^{\aleph_0}$ subsets of $\mathbb{N}$ and $\aleph_0$ computable functions $\implies$ most sets are *undecidable*.

Definition 1.4.2. (**Reduction**) A reduction $f : S_1 \to S_2$ of S_1 to S_2, where $S_1, S_2 \subseteq \mathbb{N}$ is a computable function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ s.t

$$\forall x \in \mathbb{N}. x \in S_1 \iff f(x) \in S_2.$$

- A reduction from S_1 to S_2 reduces S_1 to S_2, hence if S_2 is decidable, then S_1 must be.

Lemma 1.4.1. For all reductions $f : S_1 \to S_2$ from S_1 to S_2,

$$S_2 \text{ is decidable} \implies S_1 \text{ is decidable.}$$

Proof. Let $S_1, S_2 \subseteq \mathbb{N}$ be arbitrary. Let $f : S_1 \to S_2$ be an arbitrary S_1 to S_2 reduction.

Let us assume that S_2 is decidable. Hence $\chi_{S_2} : \mathbb{N} \to \{0, 1\}$ is computable. By definition ??,

$$\forall x \in \mathbb{N}. \chi_{S_1}(x) = 1 \iff \chi_{S_2}(f(x)) = 1.$$

Hence $\chi_{S_1} = \chi_{S_2} \circ f$. By theorem ??, χ_{S_1} is computable. Hence S_1 is decidable. □

Corollary 1.4.0.1. For all reductions $f : S_1 \to S_2$ from S_1 to S_2,

$$S_1 \text{ is undecidable} \implies S_2 \text{ is undecidable.}$$

Proof. Contrapositive of lemma ?? □

- Corollary ?? provides a method for proving whether a S is undecidable:
  - Determine a reduction $f : H \to S$ where H is the *halting problem* (see section ??)

1.4.2 The Halting Problem

Definition 1.4.3. (**Halting Problem**) The halting problem H is the set

$$H = \left\{(e, x) \in \mathbb{N}^2 : \varphi_e(x)\downarrow\right\}.$$

- Define $K = \{e \in \mathbb{N} : \varphi_e(e)\downarrow\}$

Lemma 1.4.2. The partial function $f : \mathbb{N} \to \mathbb{N}^2$, $f(e) = (e, e)$ is a reduction from K to H

Theorem 1.4.1. H is undecidable.

Proof. By lemma ?? and corollary ??, we wish to show that K is undecidable.

We proceed by contradiction. Let us assume that K is decidable, hence there exists a RM $M = (\mathcal{R}_K, P_K)$ that computes $\chi_K : \mathbb{N} \to \{0, 1\}$.

Let $M' = (\mathcal{R}_K, P_{K'})$ be the RM by replacing `HALT` (and erroneous halts) in M with: IMAGE

This yields the computable function:

$$\varphi_e(x) = \begin{cases} 0 & x \notin K \\ \uparrow & x \in K \end{cases},$$

where $e = \mathcal{E}\left[\!\left[P_{K'}\right]\!\right]_P$. Note that

$$e \in K \iff \varphi_e(e)\downarrow \iff e \notin K$$

A contradiction!

□

2 Turing Machines

2.1 Turing Machines

Definition 2.1.1. (**Turing Machines**) A Turing machine is the 4-tuple (Q, Σ, q_0, δ):

(i) Q is a finite set of *states*, disjoint from $\{\texttt{acc}, \texttt{rej}\}$.

(ii) Σ is a finite alphabet, disjoint from Q and $\{\triangleright, \_\}$

(iii) $q_0 \in Q$ is the initial state.

(iv) $\delta : (Q \times \Sigma) \to (Q \cup \{\text{acc}, \text{rej}\}) \times \Sigma \times \{\text{L}, \text{N}, \text{R}\}$ is the transition function, satisfying $\forall q \in S. \exists q' \in Q \cup \{\text{acc}, \text{rej}\} . \delta(q, \triangleright) = (q', \triangleright, R)$.

- (iv) condition: never overwrites or moves left of the start of tape

Definition 2.1.2. (**Configuration**) A turing machine *configuration* for $M = (Q, \Sigma, q_0, \delta)$ is the tuple (q, w, u) where:

- $q \in Q \cup \{\text{acc}, \text{rej}\}$
- $w = va \in \Sigma^+$ a non-empty string of symbols, where v is left of the head and a is the current symbol.
- $u \in \Sigma^*$ is the string of symbols right of the tape head (up to $\_$ symbols).

- The initial configuration $c_0 = (q_0, \triangleright, u)$.

Definition 2.1.3. (**Transition Relation**) The transition relation for $M = (Q, \Sigma, q_0, \delta)$, denoted $\longrightarrow_M : \mathscr{C} \rightarrow\!\!\!\!\!+ \mathscr{C}$ is inductively defined by

$$\frac{\delta(q, a) = (q', a', L)}{(q, va, u) \longrightarrow_M (q', v, a'u)}$$

$$\frac{\delta(q, a) = (q', a', N)}{(q, va, u) \longrightarrow_M (q', va', u)}$$

$$\frac{\delta(q, a) = (q', a', R)}{(q, va, bu) \longrightarrow_M (q', va'b, u)} [u \in \Sigma^+]$$

$$\frac{\delta(q, a) = (q', a', R)}{(q, va, \varepsilon) \longrightarrow_M (q', va'\llcorner, \varepsilon)}$$

- See definition ?? for a *computation*: $c_0 \longrightarrow_M c_1 \longrightarrow_M \cdots$.
- A configuration $c = (q, w, u)$ is halting if $q \in \{\text{acc}, \text{rej}\}$.

Definition 2.1.4. (**Halting Computation**) A *halting computation* of a Turing machine $M = (Q, \Sigma, q_0, \delta)$, denoted $uw \Downarrow_M u'w'$, where $\Downarrow_M : \Sigma^* \rightharpoonup \Sigma^*$, defined by

$$uw \Downarrow_M u'w' \iff (q_0, u, w) \longrightarrow_M^* (q, u', w') \not\longrightarrow .$$

2.2 Computable Functions

Definition 2.2.1. (**Unary Encoding**) A string $u \in \Sigma^* = \{\triangleright, \llcorner, 0, 1\}$ encodes a lists of naturals $\ell = [n_1, \dots, n_k]$ iff u is of the form:

$$u = \triangleright\llcorner \dots \llcorner 0 \underbrace{1 \dots 1}_{n_1} \llcorner \underbrace{1 \dots 1}_{n_2} \llcorner \dots \llcorner \underbrace{1 \dots 1}_{n_k} 0 \llcorner \dots$$

Definition 2.2.2. (**Computable**) A function $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}]$ is Turing computable iff there exists a Turing machine M iff

$$(\triangleright\llcorner \dots \llcorner 0)(\llcorner \underbrace{1 \dots 1}_{x_1} \dots \underbrace{1 \dots 1}_{x_n} 0 \llcorner \dots) \Downarrow_M \triangleright\llcorner 0 \underbrace{1 \dots 1}_{y} \llcorner \dots 0 \llcorner \dots$$
$$\iff f(x_1, \dots, x_n) = y$$

Theorem 2.2.1. A partial function f is Turing computable $\iff$ f is register machine computable.

2.2.1 Church-Turing Thesis

Theorem 2.2.2. (**Church Turing Thesis**) Every (intuitive) model of computation is equivalent to a Turing machine.

3 The Lambda-Calculus

3.1 Syntax

Definition 3.1.1. (**Lambda Calculus**) $V = \{x_1, \ldots\}$ is the countably infinite set of variables. The *alphabet* of the lambda calculus is given by $\Sigma = \Sigma_P \cup \{\lambda, ., \} \cup \{(,)\}$.

The formal language, or syntax, of the lambda calculus, denoted Λ, is:

$$
\begin{aligned}
M, N \;\; &::= \; x \in V \\
&\mid \; (M_1 \;\; M_2) \\
&\mid \; (\lambda x.M)
\end{aligned}
$$

- Precedence of operators: $\lambda <$ application.
- Syntactic equivalence between terms $M, N \in \Lambda$ is defined by $\equiv: \Lambda \longrightarrow \Lambda$.
- **Notation** : We often write $\lambda x_1 \ldots x_n.M \stackrel{\Delta}{\equiv} \lambda x_1.\lambda x_2. \ldots \lambda x_n.M$
- $\lambda x.M$ *binds* x in M. A variable x is *free* in M if it not bound.

Definition 3.1.2. (**Free and bound variables**) For any term $M \in \Lambda$, $fv(M)$ and $var(M)$ are the sets of *free* variables and *variables* in M, respectively. Inductively defined by

$$
\begin{aligned}
fv(x) &= \{x\} & var(x) &= \{x\} \\
fv(M_1 \; M_2) &= fv(M_1) \cup fv(M_2) & var(M_1 \; M_2) &= var(M_1) \cup var(M_2) \\
fv(\lambda x.M) &= fv(M) \setminus \{x\} & var(\lambda x.M) &= var(M) \cup \{x\}
\end{aligned}
$$

- The set of bound variables of t, denoted $bv(t)$, is $bv(t) = var(t) \setminus fv(t)$.
- A term $M \in \Lambda$ is *closed* or a *combinator* if $fv(M) = \emptyset$.

3.1.1 α-Equivalence

Definition 3.1.3. (**Substitution**) A **substitution** θ is a partial function $\theta : V \rightharpoonup \Lambda$.

- **Notation**: $\{t_1/x_1, \ldots, t_n/x_n\}$ denotes a substitution θ, where $\theta(x_i) = t_i$ and $t/x \in \theta \iff \theta(x) = t$.

Definition 3.1.4. (α-**equivalence**) The $\equiv_\alpha : \Lambda \longleftrightarrow \Lambda$ is inductively defined by

$$\frac{}{x \equiv_\alpha x} \qquad \frac{z \notin var(t) \cup var(s) \qquad \{z/x\}\, M \equiv_\alpha \{z/y\}\, N}{\lambda x.M \equiv_\alpha \lambda y.N} \qquad \frac{M_1 \equiv_\alpha M_2 \qquad N_1 \equiv_\alpha N_2}{M_1\ N_1 \equiv_\alpha M_2\ N_2}.$$

Theorem 3.1.1. $\equiv_\alpha : \Lambda \longleftrightarrow \Lambda$ is an equivalence relation.

- $\equiv_\alpha$ introduces a *unique* (canonical) form of the term. e.g. de Brunjin indexes, etc.

Definition 3.1.5. (**Application**) The application of a substitution θ to $M \in \Lambda$, denoted $\theta\, M$, is inductively defined by

$$\theta x = \begin{cases} \theta(x) & \text{if } x \in \operatorname{dom}\theta \\ x & \text{otherwise} \end{cases}$$

$$\theta\ \lambda x.M = \begin{cases} \lambda x.\,[(\theta \setminus \{t/x\})M] & t/x \in \theta \\ \lambda x.\theta M & x \notin \operatorname{dom}\theta \wedge x \notin fv(\operatorname{rng}\theta) \end{cases}$$

$$\theta\ M_1\ M_2 = (\theta\ M_1)\ (\theta\ M_2)$$

- The condition $x \notin \operatorname{dom}\theta \wedge x \notin fv(\operatorname{rng}\theta)$ avoids *name capture.* This definition of application is said to be *capture avoiding.*

- $\equiv_\alpha$ is used to "rename" variables e.g. $\{y/x\}\,(\lambda y.x) \equiv_\alpha \{y/x\}\,(\lambda z.x) = \lambda z.y$.

3.2 Semantics

- **Idea**: Semantics are defined using substitutions $\implies$ β-reduction.

3.2.1 β-Reduction and Equivalence

- λ-abstractions can be applied to λ-terms: e.g. $(\lambda x.M)\ N$ reduces to $\{N/x\}\,M$.
- $(\lambda x.M)\ N$ is a β-redex (reduceable expression) and $\{N/x\}\,M$ is the corresponding β-reduct.

Definition 3.2.1. (**β-Reduction**) The β-reduction relation $\longrightarrow_\beta : \Lambda \;\longrightarrow\!\!\!\!\!\!\!+\;\; \Lambda$ (or *transition relation*) is inductively defined by:

$$\frac{}{(\lambda x.M)\ N \longrightarrow_\beta \{N/x\}\,M} \qquad \frac{M \longrightarrow_\beta M'}{\lambda x.M \rightarrow \lambda x.M'}$$

$$\frac{M \longrightarrow_\beta M'}{M\ N \longrightarrow_\beta M'\ N} \qquad \frac{N \longrightarrow_\beta N'}{M\ N \longrightarrow_\beta M\ N'}$$

$$\frac{N \equiv_\alpha M \qquad M \longrightarrow_\beta M' \qquad M' \equiv_\alpha N'}{N \longrightarrow_\beta N'}$$

- $\longrightarrow_\beta^*$ is the reflexive transitive closure of $\longrightarrow_\beta$ w/ $\equiv_\alpha$ used as the equivalence relation.

Theorem 3.2.1. (**Church-Rosser Theorem**) The Church-Rosser theorem states that for all $M, M_1, M_2 \in \Lambda$:

$$M \longrightarrow_\beta^* M_1 \wedge M \longrightarrow_\beta^* M_2 \implies \exists M' \in \Lambda . M_1 \longrightarrow_\beta^* M' \wedge M_2 \longrightarrow_\beta^* M'.$$

Corollary 3.2.1.1. For all $M_1, M_2 \in \Lambda$,

$$M_1 =_\beta M_2 \iff \exists M \in \Lambda . M_1 \longrightarrow_\beta^* M \longleftarrow_\beta^* M_2.$$

Proof. Let $M_1, M_2 \in \Lambda$ be arbitrary.
$(\implies)$. We proceed by rule induction on $M_1 =_\beta M_2$ with the statement

$$P(M_1, M_2) = \exists M \in \Lambda . M_1 \longrightarrow_\beta^* M \longleftarrow_\beta^* M_2.$$

Base Case: For the axiom: $\dfrac{M_1 \longrightarrow_\beta^* M_2}{M_1 =_\beta M_2}$ we have $M_1 \longrightarrow_\beta^* M_2$. We introduce $M = M_2$, since we have $M_1 \longrightarrow_\beta^* M_2$ and $M_2 \longrightarrow_\beta^* M_2$. So we have $P(M_1, M_2)$.

Inductive Step: For the rule: $\dfrac{M_2 =_\beta M_1}{M_1 =_\beta M_2}$, we wish to show that $P(M_2, M_1) \implies P(M_1, M_2)$. This follows by the commutativity of $\wedge$. So we have $P(M_1, M_2)$.

By the Principle of Rule Induction, we conclude that $P(M_1, M_2)$ holds for all $M_1 =_\beta M_2$.

$(\impliedby)$. Let us assume there exists $M \in \Lambda$ s.t $M_1 \longrightarrow^*_\beta M$ and $M_2 \longrightarrow^*_\beta M$. Then we have $M_1 =_\beta M$ and $M_2 =_\beta M$. By transitivity of $=_\beta$, we have $M_1 =_\beta M_2$. □

- **Idea**: $\longrightarrow^*_\beta$ and it's inverse defines an equivalence: *β-equivalence*

Definition 3.2.2. (β**-Equivalence**) The β-equivalence relation $=_\beta : \Lambda \nrightarrow \Lambda$ is inductively defined by:

$$\frac{M \longrightarrow^*_\beta M'}{M =_\beta M'} \qquad \frac{M =_\beta M'}{M' =_\beta M}$$

3.2.2 β-Normal Forms

- **Idea**: Church-Rosser $\implies$ a unique normal form for all $M \in \Lambda$.

Definition 3.2.3. (β**-Normal Form**) A term $M \in \Lambda$ is in β-normal form (β-nf) if it contains no β-redexes, that is

$$\nexists x \in V, N, N' \in \Lambda.(\lambda x.N)\ N' \in st(M).$$

- A term $M \in \Lambda$ has a β-nf $N \in \Lambda$ iff $M =_\beta N$ and N is in β-nf.

Theorem 3.2.2. For all terms $M \in \Lambda$, If M has a β-nf $N \in \Lambda$, then N is unique.

Proof. Let $M \in \Lambda$ be an arbitrary λ-term. We wish to show that

$$\forall N, N' \in \Lambda. M =_\beta N \text{ is in } \beta\text{-nf} \wedge M =_\beta N' \text{ is in } \beta\text{-nf} \implies N \equiv_\alpha N'.$$

Let $N, N' \in \Lambda$ be arbitrary. Let us assume that $M =_\beta N$, $M =_\beta N'$ and N, N' are in β-nf. By theorem ??, there exists $M' \in \Lambda$ s.t $N \longrightarrow^*_\beta M' \longleftarrow^*_\beta N'$. Since N, N' are β-nf, then $N \equiv_\alpha N'$. □

- Non-terminating terms, e.g. $\Omega \triangleq (\lambda x.xx)(\lambda x.xx)$ has no β-nf.
- A λ-term may have a β-nf *and* be non-terminating (since $\longrightarrow_\beta$) is **non-deterministic**. e.g. $(\lambda x.y)\Omega$.
- **Problem**: non-determinism of $\longrightarrow_\beta$
- **Solution**: normal-order reduction

Definition 3.2.4. (**Normal-Order Reduction**) The β normal-order -reduction relation $\longrightarrow_{\eta\beta}$: $\Lambda \;\longrightarrow\!\!\!\!\!\!+\; \Lambda$ (or *transition relation*) is inductively defined by:

$$\frac{}{(\lambda x.M)\ N \longrightarrow_{\eta\beta} \{N/x\}\ M} \qquad \frac{M \longrightarrow_{\eta\beta} M'}{M\ N \longrightarrow_{\eta\beta} M'\ N}$$

$$\frac{N \equiv_\alpha M \qquad M \longrightarrow_{\eta\beta} M' \qquad M' \equiv_\alpha N'}{N \longrightarrow_{\eta\beta} N'}$$

Theorem 3.2.3.

$$\forall M \in \Lambda. \exists N \in \Lambda. M \longrightarrow^*_{\eta\beta} N \not\longrightarrow_{\eta\beta} \implies N \text{ is } \beta\text{-nf of } M$$

3.3 Computable Functions

3.3.1 λ-Computable Functions

3.3.1.1 Church Numerals, Booleans and Pairs

Definition 3.3.1. (**Church Numerals**) The Church numeral of $n \in \mathbb{N}$, denoted $\underline{n}$ is defined as

$$\underline{n} \triangleq \lambda f x. f^n\ x$$

where

$$M^0\ N \triangleq N$$
$$M^{n+1}\ N \triangleq M\ (M^n\ N)$$

- A Church numeral represents a *fold* of f (applied n times): $\underline{n}\ M\ N =_\beta M^n\ N$. S

Theorem 3.3.1. For all $n \in \mathbb{N}$,

$$\mathsf{Succ}\ \underline{n} =_\beta \underline{n+1}$$

where $\mathsf{Succ} \triangleq \lambda n f x. f\ (n\ f\ x)$.

Proof. Let $n \in \mathbb{N}$ be arbitrary. We have

$$\begin{aligned} \mathsf{Succ}\ \underline{n} &=_\beta \lambda f x. f\ (\underline{n}\ f\ x) \\ &=_\beta \lambda f x. f\ (f^n\ x) \\ &\triangleq \lambda f x. f^{n+1} x \\ &\triangleq \underline{n+1} \end{aligned}$$

□

- Predecessor function $pred(n)$: fold the function $f(x, y) = (x + 1, x)$ n times with initial pair $(0, 0)$ and project the second element $(n, n - 1)$.

Theorem 3.3.2. For all $n \in \mathbb{N}$,

$$\begin{aligned} \mathsf{Pred}\ \underline{0} &=_\beta \underline{0} \\ \mathsf{Pred}\ \underline{n+1} &=_\beta \underline{n} \end{aligned}$$

where

$$\begin{aligned} \mathsf{Pred} &\triangleq \lambda n f x. \mathsf{Snd}\ (n\ (G\ f)\ (\mathsf{Pair}\ x\ x)) \\ G &\triangleq \lambda f p. \mathsf{Pair}\ (f\ (\mathsf{Fst}\ p))\ (\mathsf{Fst}\ p) \end{aligned}$$

Proof. We have

$$\begin{aligned} \mathsf{Pred}\ \underline{0} &=_\beta \lambda f x. \mathsf{Snd}\ (\underline{0}\ (G\ f)\ (\mathsf{Pair}\ x\ x)) \\ &=_\beta \lambda f x. \mathsf{Snd}\ ((\lambda f x. x)\ (G\ f)\ (\mathsf{Pair}\ x\ x)) \\ &=_\beta \lambda f x. \mathsf{Snd}\ (\mathsf{Pair}\ x\ x) \\ &=_\beta \lambda f x. x \\ &\triangleq \underline{0} \end{aligned}$$

Remainder is inductive proof on the fold of f □

Definition 3.3.2. (**Church Boolean**) The boolean values `true` and `false` are defined as

$$\begin{aligned} \mathsf{True} &\triangleq \lambda xy.x \\ \mathsf{False} &\triangleq \lambda xy.y \end{aligned}$$

- $\mathsf{True}\ M\ N =_\beta M$ and $\mathsf{False}\ M\ N =_\beta N$. So we define

$$\mathsf{If} \triangleq \lambda bxy.b\ x\ y.$$

- Note $\underline{0} \equiv_\alpha \mathsf{False}$

Theorem 3.3.3. We have

(i) $\mathsf{Eq}_0\ \underline{0} =_\beta \mathsf{True}$

(ii) For all $n \in \mathbb{N}$, $\mathsf{Eq}_0\ \underline{n+1} =_\beta \mathsf{False}$

where

$$\mathsf{Eq}_0 \triangleq \lambda x.x\ (\lambda y.\mathsf{False})\ \mathsf{True}.$$

Proof. For (i), we have

$$\begin{aligned} \mathsf{Eq}_0\ \underline{0} &=_\beta \underline{0}\ (\lambda y.\mathsf{False})\ \mathsf{True} \\ &=_\beta \mathsf{True} \end{aligned}$$

For (ii), let $n \in \mathbb{N}$ be arbitrary.

$$\begin{aligned} \mathsf{Eq}_0\ \underline{n+1} &=_\beta \underline{n+1}\ (\lambda y.\mathsf{False})\ \mathsf{True} \\ &=_\beta (\lambda y.\mathsf{False})^{n+1}\ \mathsf{True} \\ &=_\beta \mathsf{False} \end{aligned}$$

□

Definition 3.3.3. (**Church Pairs**) The Church pair of (M, N), denoted $\mathsf{Pair}\ M\ N$, is defined as

$$\mathsf{Pair} \triangleq \lambda xyf.f\ x\ y.$$

Theorem 3.3.4. We have

$$\mathsf{Fst}\ (\mathsf{Pair}\ M\ N) =_\beta M$$
$$\mathsf{Snd}\ (\mathsf{Pair}\ M\ N) =_\beta N$$

where

$$\mathsf{Fst} \triangleq \lambda p.p\ (\lambda xy.x)$$
$$\mathsf{Snd} \triangleq \lambda p.p\ (\lambda xy.y)$$

Proof. We have

$$\begin{aligned}
\mathsf{Fst}\ (\mathsf{Pair}\ M\ N) &=_\beta (\mathsf{Pair}\ M\ N)\ (\lambda xy.x) \\
&=_\beta (\lambda f.f\ M\ N)\ (\lambda xy.x) \\
&=_\beta (\lambda xy.x)\ M\ N \\
&=_\beta M \\
\mathsf{Snd}\ (\mathsf{Pair}\ M\ N) &=_\beta (\mathsf{Pair}\ M\ N)\ (\lambda xy.y) \\
&=_\beta (\lambda f.f\ M\ N)\ (\lambda xy.y) \\
&=_\beta (\lambda xy.y)\ M\ N \\
&=_\beta N
\end{aligned}$$

□

3.3.1.2 λ-Computable

Definition 3.3.4. (**λ-Computable**) $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}]$ is λ-*computable* if there exists a closed λ-term $F \in \Lambda$ s.t for all $(x_1, \ldots, x_n) \in \mathbb{N}^n, y \in \mathbb{N}$:

(i) $f(x_1, \ldots, x_n) = y \implies F\ \underline{x_1}\ \cdots\ \underline{x_n} =_\beta \underline{y}$, or

(ii) $f(x_1, \ldots, x_n)\uparrow \implies F\ \underline{x_1}\ \cdots\ \underline{x_n}$ has no β-nf.

- **Examples**:
  - TODO

Theorem 3.3.5. If $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}]$, $g_1, \ldots, g_n \in \mathcal{P}[\mathbb{N}^m \rightharpoonup \mathbb{N}]$ are λ-computable, then $f \circ \{g_1, \ldots, g_n\} \in \mathcal{P}[\mathbb{N}^m \rightharpoonup \mathbb{N}]$ is λ-computable, with the combinator

$$\begin{aligned}
\mathsf{F} \circ \{\mathsf{G}_1, \ldots, \mathsf{G}_n\} \triangleq \lambda x_1 \ldots x_m. & \\
& (\mathsf{G}_1\ x_1\ \ldots\ x_m\ \mathsf{I}) \ldots (\mathsf{G}_n\ x_1\ \ldots\ x_m\ \mathsf{I}) \\
& \mathsf{F}\ (\mathsf{G}_1\ x_1\ \ldots\ x_m) \ldots (\mathsf{G}_n\ x_1\ \ldots\ x_m)
\end{aligned}$$

Proof. See supervision work (for $n = 1$). □

3.3.2 Partial Recursion

Theorem 3.3.6. For all $f \in \bigcup \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}]$, $f \in \mathscr{P}_1$ is partial recursive $\iff$ it is λ-computable.

- The basic functions π_i^n, zero^n and succ have the combinators:
  - **Projection**: $\pi_i^n : \mathbb{N}^n \to \mathbb{N}$ is defined as
  $$\mathsf{Proj}_i^n \triangleq \lambda x_1 \dots x_n . x_i.$$
  - **Zero**: $\text{zero}^n : \mathbb{N}^n \to \mathbb{N}$ is defined as
  $$\mathsf{Zero}^n \triangleq \lambda x_1 \dots x_n . \underline{0}.$$
  - **Successor**: $\text{succ} : \mathbb{N} \to \mathbb{N}$ is defined as
  $$\mathsf{Succ} \triangleq \lambda n f x . f\ (n\ f\ x).$$
- Composition $\mathsf{F} \circ \{\mathsf{G}_1, \dots, \mathsf{G}_n\}$ is given by theorem ??

3.3.2.1 Fixed Point Combinator Y

- **Problem**: Representing a recursive function `let` $f = \underbrace{\dots f \dots f \dots}_{M}.$
- **Solution**: Y fixed point combinator, with fixed point property: $\mathsf{Y}\ M =_\beta M\ (\mathsf{Y}\ M)$.
- **Derivation**:
  - The multiple occurrences of f may be factored: `let` $f = (\underbrace{\lambda r.(\dots r \dots r \dots)}_{M})$ `in` $f\ f$.

    r must be replaced by $(r\ r)$ since f has an additional argument (itself): `let` $f = (\underbrace{\lambda r.(\dots (r\ \ r) \dots (r\ \ r) \dots)}_{M'})$ `in` $f\ \ f$

```
let fact =
    λ fact n.
        if n = 0 then 1
        else n × fact fact (n − 1)
in fact fact 3
```

– The multiple occurrences of $(r\ r)$ may be factored:
$\texttt{let}\ f = (\lambda x.\underbrace{\lambda r.(\dots r \dots r \dots)}_{M}\ (x\ x))\ \texttt{in}\ f\ f.$

– Define $\texttt{let}\ x\ \texttt{=}\ M\ \texttt{in}\ N \triangleq \texttt{(}\lambda x\texttt{.}\ N\texttt{)}\ M$. So we have

$$f \triangleq (\lambda x.\underbrace{\lambda r.(\dots r \dots r \dots)}_{M}\ (x\ x))\ (\lambda x.\underbrace{\lambda r.(\dots r \dots r \dots)}_{M}\ (x\ x))$$

– The multiple occurrences of M may be factored:

$$\mathsf{Y} \triangleq \lambda m.(\lambda x.m\ (x\ x))\ (\lambda x.m\ (x\ x)),$$

with $f \triangleq \mathsf{Y}\ M$.

Theorem 3.3.7. The Y combinator satisfies the fix point property: for all $M \in \Lambda$, $\mathsf{Y}\ M =_\beta M\ (\mathsf{Y}\ M)$.

Proof. Let $M \in \Lambda$ be arbitrary. We have

$$\begin{aligned} \mathsf{Y}\ M &=_\beta (\lambda x.M\ (x\ x))\ (\lambda x.M\ (x\ x)) \\ &=_\beta M\big((\lambda x.M\ (x\ x))\ (\lambda x.M\ (x\ x))\big) \\ &\triangleq M\ (\mathsf{Y}\ M) \end{aligned}$$

□

3.3.2.2 Primitive Recursion and Minimization

- Primitive recursion may be expressed as the fixed point: $\rho^n(f,g)(\mathbf{x},x) = h(\mathbf{x},x)$ with $h = \Phi^n_{f,g}(h)$ where

$$\Phi^n_{f,g}(h)(\mathbf{x},x) = \begin{cases} f(\mathbf{x}) & \text{if } x = 0 \\ g(\mathbf{x}, x-1, h(\mathbf{x}, x-1)) & \text{otherwise} \end{cases}$$

Theorem 3.3.8. If $f \in \mathcal{P}[\mathbb{N}^n \rightharpoonup \mathbb{N}], g \in \mathcal{P}[\mathbb{N}^{n+2} \rightharpoonup \mathbb{N}]$ are λ-computable, then $\rho^n(f,g)$ is λ-computable, with the combinator

$$\begin{aligned} \mathsf{R}^n(\mathsf{F},\mathsf{G}) \triangleq \mathsf{Y}\big(\lambda h.\lambda \mathbf{x} x.\ & \\ & \mathsf{If}\ (\mathsf{Eq}_0\ x)\ (\mathsf{F}\ \mathbf{x}) \\ & \qquad (\mathsf{G}\ \mathbf{x}\ (\mathsf{Pred}\ x)\ (h\ \mathbf{x}\ (\mathsf{Pred}\ x)))\big) \end{aligned}$$

where F, G are the combinators of f, g.

- Minimization may also be represented by a fixed point equation: $\mu^n f = g(\mathbf{x}, 0)$ with $g = \Psi_f(g)$ where

$$\Psi_f(g)(\mathbf{x}, x) = \begin{cases} x & \text{if } f(\mathbf{x}, x) = 0 \\ g(\mathbf{x}, x+1) & \text{otherwise} \end{cases}$$

Theorem 3.3.9. If $f \in \mathcal{P}[\mathbb{N}^{n+1} \rightharpoonup \mathbb{N}]$ is λ-computable then $\mu^n f$ is λ-computable, with the combinator

$$\begin{aligned} \mathsf{M}^n(\mathsf{F}) \triangleq \lambda\mathbf{x}.\mathsf{Y}\big(\lambda g.\lambda\mathbf{x}x. \\ &\mathsf{If}\ (\mathsf{Eq}_0\ (\mathsf{F}\ \mathbf{x}\ x))\ x \\ &\qquad (h\ \mathbf{x}\ (\mathsf{Succ}\ x))\big)\ \mathbf{x}\ \underline{0} \end{aligned}$$