1 Register Machines

A reg machine instr is of format:

$$L_1 : R_x^+ \to L_y$$
$$L_2 : R_x^- \to L_y, L_z$$
$$L_3 : \text{HALT}$$

On subtract, jump to first iff $R_x > 0$. A **configuration** has form $(l, r_0, \cdots r_n)$ where l is the current label and r_i is the contents of R_i. A computation is a finite seq of configs starting with c_0, ending in a **halting config** or erroneous config. Regmachs computation is *deterministic*, so the relation between initial and final configs is a *partial func* $(\forall \langle x,y\rangle \in f.[\langle x,y'\rangle \in f \implies y = y'])$.

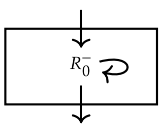
$f(x) = y$	$\langle x,y\rangle \in f$
$f(x)\downarrow$	$\exists y \in Y.[f(x) = y]$
$f(x)\downarrow$	$\nexists y \in Y.[f(x) = y]$
$X \rightharpoonup Y$	All partial funcs (X to Y)
$X \to Y$	All total funcs (X to Y)

A total func: $\forall x \in X.[f(x)\downarrow]$. A partial func f is **regmach computable** if $\exists$ regmach M with $n+1$ regs s.t. $\forall (x_1, \cdots, x_n) \in \mathbb{N}^n. \forall y \in \mathbb{N}$ the computation of M starting with $R_0 = 0, R_1 = x_1, \cdots, R_n = x_n$ halts with $R_0 = y$ iff $f(x_1, \cdots, x_n) = y$.

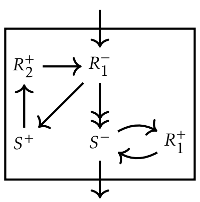
Pairs can be encoded as $\langle\langle x,y\rangle\rangle \triangleq 2^x(2y+1)$ and $\langle x,y\rangle \triangleq 2^x(2y+1) - 1$. where $\langle\langle\cdot,\cdot\rangle\rangle$ is a *bijection* $\mathbb{N}^2 \to \mathbb{N}^+$ and $\langle\cdot,\cdot\rangle$ is a *bijection* $\mathbb{N}^2 \to \mathbb{N}$. Observe $0b\langle\langle x,y\rangle\rangle = 0by1 + x$ 0s and $0b\langle x,y\rangle = 0by0 + x$ 1s.

Lists may be encoded with pairs with $\ulcorner[]\urcorner \triangleq 0$ and $\ulcorner x :: l\urcorner \triangleq \langle\langle x, \ulcorner l\urcorner\rangle\rangle$, giving a bijection $l \mapsto \ulcorner l\urcorner$ from $\mathbb{N}$ to $\mathbb{N}$. *For example, [1,2,3] is encoded as 0b100010010.*

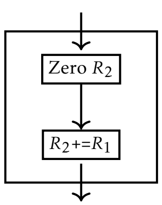
Programs: $\ulcorner P\urcorner \triangleq \ulcorner[\ulcorner I_0\urcorner, \cdots, \ulcorner I_n\urcorner]\urcorner$ where $\ulcorner R_i^+ \to L_j\urcorner \triangleq \langle\langle 2i, j\rangle\rangle$, $\ulcorner R_i^- \to L_j, L_k\urcorner \triangleq \langle\langle 2i+1, \langle j,k\rangle\rangle\rangle$ and $\ulcorner\text{HALT}\urcorner \triangleq 0$. Its easy to show any x decodes to a unique instruction, so any **program index** $x \in \mathbb{N}$ decodes to a unique program.



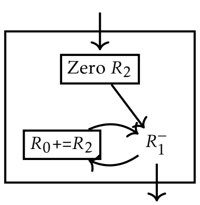
(a) Zero R_1



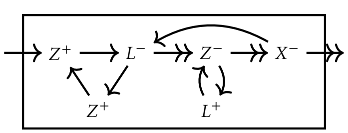
(b) Add R_1 to R_2



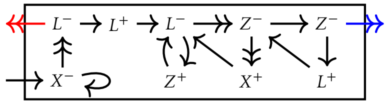
(c) Copy R_1 to R_2



(d) Mult R_1 by R_2 to R_0



(e) Push X to stack L



(f) Pop X from stack L

2 Halting Problem

A RM H decides the **Halting problem** if $\forall e, a_1, \cdots a_n \in \mathbb{N}$, starting H with $R_0 = 0, R_1 = e, R_2 = \ulcorner[a_1, \cdots, a_n]\urcorner$ always halts with $R_0 = 1$ iff the RM prog e halts when started with $R_0 = 0, R_1 = a_1, \cdots R_n = a_n$. We **z** no such H exists:

1. Assume H decides the Halting problem. Let H' replace START in H with START;Z ::= R_1;push Z to R_1.
2. Let C replace HALT in H' with zero R_1;HALT, let c be the index of C.
3. C started with $R_1 = c$ **halts** *iff* H' started with $R_1 = c$ **halts** with $R_0 = 0$, *iff* H started with $R_1 = c, R_2 = \ulcorner[c]\urcorner$ **halts** with $R_0 = 0$, *iff* $prog(c) = C$ started with $R_1 = c$ **doesn't halt**. ↯

$\forall e \in \mathbb{N}.\phi_e \in \mathbb{N} \rightharpoonup \mathbb{N}$ is part func computed by regmach prog e: $\forall x,y \in \mathbb{N}.\phi_e(x) = y$ holds iff the computation of $prog(e)$ with $R_0 = 0, R_1 = x$ halts with $R_0 = y$. Thus, $e \mapsto \phi_e$ defines a **surjection** from $\mathbb{N}$ to all **computable** part funcs from $\mathbb{N} \rightharpoonup \mathbb{N}$ (*countable*), but $\mathbb{N} \rightharpoonup \mathbb{N}$ is *uncountable*, and contains **uncomputable** funcs.

The **characteristic func** of $S \subseteq \mathbb{N}$ is $\chi_S \in \mathbb{N} \to \mathbb{N}$ given by $\chi_S(x) \triangleq \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases}$. S is **decidable** if χ_S is *computable*. To *prove* S is **undecidable**, show its decidability implies the decidability of the halting problem.

3 Turing Machines

Turing Machine $M = \langle Q, \Sigma, s, \delta\rangle$, of **states** Q, **possible tape symbols** Σ, **initial state** $s \in Q$ and **transition func** $\delta \in (Q \times \Sigma) \rightharpoonup (Q \times \Sigma \times \{L,R\})$. A **configuration** $\langle q, w, u\rangle$ has **current state** $q \in Q$, **left** / **right tape content** $w, u \in \Sigma^*$. Initial config $\langle s, \epsilon, u\rangle$.

We can get the *first* and *last* symbols with:

$$\texttt{first}(w) = \begin{cases} (a,v) & \text{if } w = a \cdot v \\ (\sqcup, \epsilon) & \text{if } w = \epsilon \end{cases}$$

$$\texttt{last}(w) = \begin{cases} (a,v) & \text{if } w = v \cdot a \\ (\sqcup, \epsilon) & \text{if } w = \epsilon \end{cases}$$

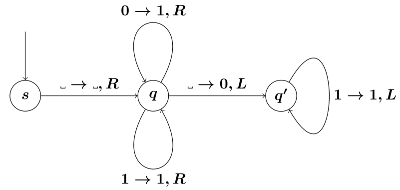
Given $M = \langle Q, \Sigma, s, \delta\rangle$, define $\langle q,w,u\rangle \to_M \langle q',w',u'\rangle$ where $\texttt{first}(u) = (a, u')$:

$$\frac{\delta(q,a) = (q',a',L) \land \texttt{last}(w) = (b,w')}{\langle q,w,u\rangle \to_M \langle q',w',ba'u'\rangle}$$

$$\frac{\delta(q,a) = (q',a',R)}{\langle q,w,u\rangle \to_M \langle q',wa',u'\rangle}$$

- *If state q, a is read from tape, and δ says move left, mach moves to q′, writes a′ to tape, and moves left.*
- *If state q, a is read from tape, and δ says move right, mach moves to q′, writes a′ to tape, and moves right.*

$\langle q,w,u\rangle$ is in **normal form** if $\delta(q,a)\uparrow$ for $\texttt{first}(u) = (a,u')$. A **computation** of M is an infinite config seq where $c_0 = \langle s, \epsilon, u'\rangle$ and $\forall i \in \mathbb{N}.[c_i \to_M c_{i+1}]$. It *halts iff* the seq is finite i.e. the *last config is in normal form*. e.g. graphical representation of M:

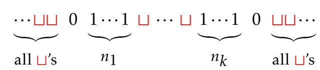


We can draw δ in a table.

We can prove any turing mach M can be mapped to a regmach:

1. Fix numerical encoding of M's states, symbols, tape and configs.
2. Implement δ as a regmach prog.
3. Implement a regmach program to repeatedly apply $\to_M$.

A tape over $\Sigma = \{\sqcup, 0, 1\}$ codes a **list of numbers** $[n_1, \cdots, n_k]$ as:



$f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ is **turing computable** iff $\exists M$ s.t. starting M on a tape coding $[x_1, \cdots x_n]$ halts iff $f(x_1, \cdots x_n)\downarrow$ and in that case the final tape holds a list whose first element is y where $f(x_1, \cdots x_n) = y$. *A part fun is turing computable iff it is regmach computable.*

4 Lambda Calculus

λ-calc is *var* | *abstraction* | *application*.

$$M \triangleq x \mid \lambda x.M \mid MM$$

Free vars are not *bound* in an abstraction. Terms with no free vars are **closed** or **grounded**:

$$\mathsf{FV}(x) = \{x\}$$
$$\mathsf{FV}(\lambda x.M) = \mathsf{FV}(M) - \{x\}$$
$$\mathsf{FV}(MN) = \mathsf{FV}(M) \cup \mathsf{FV}(N)$$

For **α-equivalence**, we can rename *bound* vars without changing term meaning (*e.g.* $\lambda x.x =_\alpha \lambda y.y$). To help, we can *substitute* $M[N/x]$, replacing x in M with N. We *cannot substitute bound var*, and *must rename conflicts*.

β-reduction means applying a func to an arg (*e.g.* $(\lambda x.x)y \to_\beta y$). More formally $(\lambda x.M)N \to_\beta M[N/x]$. Rules for $\to_\beta$:

$$\frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'}$$

$$\frac{M \to_\beta M'}{MN \to_\beta M'N} \qquad \frac{N \to_\beta N'}{MN \to_\beta MN'}$$

$$\frac{M =_\alpha M' \land M' \to_\beta N' \land N' =_\alpha N}{M \to_\beta N}$$

A **normal form** has no β-red. Any term that has one *will reach it*. A **multi step β red** $\to_\beta^*$ enforces the reflexive transitive closure of β-red under α-conv:

$$\frac{M =_\alpha M'}{M \to_\beta^* M'} \qquad \frac{M \to_\beta M'' \land M'' \to_\beta^* M'}{M \to_\beta^* M'}$$

Confluence states $\forall M, M_1, M_2.[M \to_\beta^* M_1 \land M \to_\beta^* M_2 \implies \exists M'.[M_1 \to_\beta^* M' \land M_2 \to_\beta^* M']]$. This can prove that normal forms are unique.

β-equiv is the smallest equiv relation containing $\to_\beta$ with symmetry: $M_1 =_\beta M_2 \Leftrightarrow \exists M'.[M_1 \to_\beta^* M' \land M_2 \to_\beta^* M']$.

A **redex** is a reductible expr. Not all terms have a NF (*e.g.* $(\lambda x.xx)(\lambda x.xx)$), and some terms only have a NF under certain reduction strats. For redex $E = (\lambda x.M)N$:

- Redexes in M or N are **inside** E.
- E is **outside** any redexes in M or N.
- E is **outermost** if no redexes contain it.
- E is **innermost** if it contains no redexes.

There are many *reduction strategies*:

1. **Normal Order** reduces leftmost, outermost redex first. Always terminates at a NF, but can ppform computations on unevaluated func bodies.
2. **Call by Name** reduces leftmost, outermost redex first, ignoring inside λ abstractions. Does not always reduce to NF, evals args when needed.
3. **Call by Value** reduces leftmost, innermost redex first, ignoring inside λ abstractions. Does not always reduce to NF, evals args before passing to func body.

A part func $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is **λ-definable** iff $\exists$ closed M s.t. $f(x_1, \cdots x_n) = y \Leftrightarrow Mx_1, \cdots x_n =_\beta y$ and $f(x_1, \cdots x_n)\uparrow \Leftrightarrow Mx_1, \cdots x_n$ has no NF. The **Church-Turing thesis** states f is **λ-computable** iff it is λ-definable, and hence turing mach computable and regmach computable.

Church numerals are defined as $n \triangleq \lambda f.\lambda x.(f(\overset{n}{\cdots}(fx)\overset{n}{\cdots}))$. To encode $m+n$, we place the body of n into that of m: $mf(nfx)$. Hence:

- $\texttt{plus} \triangleq_\beta \lambda m.\lambda n.\lambda f.\lambda x.mf(nfx)$ *Adds Church numerals m and n*
- $\texttt{mult} \triangleq_\beta \lambda m.\lambda n.\lambda f.m(nf)$ *Multiplies Church numerals m and n*
- $\texttt{exp} \triangleq_\beta \lambda m.\lambda n.nm$ *Exponentiates Church numerals as* m^n
- $\texttt{succ} \triangleq_\beta \lambda n.\lambda f.\lambda x.f(nfx)$ *Returns the successor of n (i.e., n+1)*
- $\texttt{pred} \triangleq_\beta \lambda n.\lambda f.\lambda x.n(\lambda g.\lambda h.h(gf))(\lambda u.x)(\lambda u.u)$ *Computes the predecessor of n, clamping at 0*
- $\texttt{sub} \triangleq_\beta \lambda m.\lambda n.n\,\texttt{pred}\,m$ *max(0, n − m)*
- $\texttt{ifz} \triangleq_\beta \lambda m.\lambda x_1.\lambda x_2.m(\lambda z.x_2)x_1$ *x_1 if $m = 0$, otherwise x_2*
- $\texttt{double} \triangleq_\beta \lambda n.\lambda f.\lambda x.n(f \circ f)x$
- $\texttt{div} \triangleq_\beta \mathsf{Y}(\lambda r.\lambda m.\lambda n.\ \texttt{ifz}(\texttt{sub}\,m\,n)\,\texttt{zero}(\lambda\_.\texttt{succ}(r(\texttt{sub}\,m\,n)n)))$ *Returns $\lfloor m/n \rfloor$*
- $\texttt{mod} \triangleq_\beta \mathsf{Y}(\lambda r.\lambda m.\lambda n.\ \texttt{ifz}(\texttt{sub}\,m\,n)m(\lambda\_.r(\texttt{sub}\,m\,n)n))$ *Remainder of m divided by n*
- $\texttt{pair} \triangleq_\beta \lambda v_1 v_2.(\lambda p.pv_1v_2)$
- $\texttt{fst} \triangleq_\beta \lambda p.p(\lambda w_1w_2.w_1)$
- $\texttt{snd} \triangleq_\beta \lambda p.p(\lambda w_1w_2.w_2)$

Common **combinators** (no free vars) are:

I	$\mathsf{I} \triangleq \beta\lambda x.x$
K	$\mathsf{K} \triangleq \beta\lambda xy.x$
S	$\mathsf{S} \triangleq \beta\lambda xyz.xz(yz)$
T	$\mathsf{T} \triangleq \beta\lambda xy.yx$
C	$\mathsf{C} \triangleq \beta\lambda xyz.xzy$
B	$\mathsf{B} \triangleq \beta\lambda xyz.x(yz)$
B′	$\mathsf{B'} \triangleq \beta\lambda xyz.y(xz)$
W	$\mathsf{W} \triangleq \beta\lambda xy.xyy$
Y	$\mathsf{Y} \triangleq \beta\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

For example, we define recursive $\texttt{fact} \triangleq_\beta \lambda n.\texttt{ifz}\ n\ 1\ (\texttt{mult}\ n\ (\texttt{fact}(\texttt{pred}\ n)))$. Instead, we could use the Y-combinator, removing recursion. Then: $\texttt{fact} \triangleq_\beta \mathsf{Y}(\lambda f.\lambda n.\texttt{ifz}\ n\ 1\ (\texttt{mult}\ n\ (f(\texttt{pred}\ n))))$.

5 Operational Semantics

While lang is defined as:

$$B \in \texttt{Bool} \triangleq \texttt{true} \,|\, \texttt{false} \,|\, E_1 =_s E_2 \,|\, E_1 <_s E_2$$
$$|\, B_1 \&_s B_2 \,|\, \neg_s B$$
$$E \in \texttt{Exp} \triangleq x \,|\, n \,|\, E_1 +_s E_2 \quad \text{s.t. } x \in \texttt{Var}, N \in \mathbb{N}$$
$$C \in \texttt{Com} \triangleq x ::= E \,|\, \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2$$
$$|\, C_1 ;\, C_2 \,|\, \texttt{skip} \,|\, \texttt{while } B \texttt{ do } C$$

We also define a smaller *SimpExp* as $E \in \texttt{SimpExp} \triangleq n \,|\, E_1 +_s E_2$. A **big step semantic** $\Downarrow \subseteq \texttt{SimpExp} \times \mathbb{N}$ where $E \Downarrow n$ means E evals to n (*final answer*):

$$\text{B-NUM}\ \frac{}{n \Downarrow n}$$
$$\text{B-ADD}\ \frac{E_1 \Downarrow n_1 \quad E_2 \Downarrow n_2}{E_1 +_s E_2 \Downarrow n_3}\ n_3 = n_1 + n_2$$

It is **determinant** $\forall E \in \texttt{SimpExp}.\forall n_1, n_2 \in \mathbb{N}.[E \Downarrow n_1 \wedge E \Downarrow n_2 \implies n_1 = n_2]$ and **total** $\forall E \in \texttt{SimpExp}.\exists n \in \mathbb{N}.E \Downarrow n$.

An example **derivation tree** is:

$$F\ \frac{B\ \dfrac{\langle x+1\rangle, [x \mapsto 0] \Downarrow 1 \quad s[x \mapsto 1] = s'}{\langle x := x+1, [x \mapsto 0] \Downarrow [x \mapsto 1]} \quad F\ \dfrac{\cdots}{\cdots}}{\langle \texttt{loop}(x := x+1), [x \mapsto 0]\rangle \Downarrow s''}$$

A **small step semantic** $\rightarrow \subseteq \texttt{SimpExp} \times \texttt{SimpExp}$ where $E \rightarrow E'$ means E *reduces* to E'. To eval, use these rules in-order:

$$\text{S-LEFT}\ \frac{E_1 \rightarrow E_1'}{E_1 +_s E_2 \rightarrow E_1' +_s E_2}$$
$$\text{S-RIGHT}\ \frac{E_2 \rightarrow E_2'}{n +_s E_2 \rightarrow n +_s E_2'}$$
$$\text{S-ADD}\ \frac{}{n_1 +_s n_2 \rightarrow n_3}\ n_3 = n_1 + n_2$$

A *reflexive transitive closure* $E \rightarrow^* E'$ holds iff $E = E'$ or $\exists$ a finite seq. $E \rightarrow \cdots \rightarrow E'$. E is in **normal form** iff $\nexists E'.E \rightarrow E'$. $\rightarrow$ is: **Deterministic** $\forall E, E_1, E_2.[E \rightarrow E_1 \wedge E \rightarrow E_2 \implies E_1 = E_2]$. **Confluent** $\forall E, E_1, E_2.[E \rightarrow^* E_1 \wedge E \rightarrow^* E_2 \implies \exists E'.[E_1 \rightarrow^* E' \wedge E_2 \rightarrow^* E']]$. **Weakly Normalized** $\forall E.\exists E'.[E \rightarrow^* E' \wedge E'$ is normal$]$. **Strongly Normalized** $\forall E.[\nexists$ inf seq. $E_1, \cdots, E_n$ s.t.$\forall i \in \mathbb{N}.E_i \rightarrow E_{i+1}]$. **Has Unique NF** $\forall E, E_1, E_2.[E \rightarrow^* E_1 \wedge E \rightarrow^* E_2 \wedge E_1, E_2$ in NF $\implies E_1 = E_2]$. We can relate $\forall E \in \mathit{SimpExp}.\forall n \in \mathbb{N}[E \Downarrow n \Leftrightarrow E \rightarrow^* n]$.

State $\triangleq \texttt{Var} \rightharpoonup \mathbb{N}$ (e.g. $s_1 = (x \mapsto 1)$). A **configuration** is $\langle E, s\rangle$. A **state update**:

$$s[v \mapsto n](u) = \begin{cases} n & \text{if } u = v \\ s(u) & \text{otherwise} \end{cases}$$

$\rightarrow_e, \rightarrow_b, \rightarrow_c$ are *deterministic* & *confluent*:

$$\text{EXP.L}\ \frac{\langle E_1, s\rangle \rightarrow_e \langle E_1', s'\rangle}{\langle E_1 +_s E_2, s\rangle \rightarrow_e \langle E_1' +_s E_2, s'\rangle}$$
$$\text{EXP.R}\ \frac{\langle E, s\rangle \rightarrow_e \langle E', s'\rangle}{\langle n +_s E, s\rangle \rightarrow_e \langle n +_s E', s'\rangle}$$
$$\text{EXP.VAR}\ \frac{}{\langle x, s\rangle \rightarrow_e \langle n, s\rangle}\ n = s(x)$$
$$\text{EXP.ADD}\ \frac{}{\langle n_1 +_s n_2, s\rangle \rightarrow_e \langle n_3, s\rangle}\ n_3 = n_1 + n_2$$
$$\text{ASS.EXP}\ \frac{\langle E, s\rangle \rightarrow_e \langle E', s'\rangle}{\langle x ::= E, s\rangle \rightarrow_c \langle x ::= E', s'\rangle}$$
$$\text{ASS.NUM}\ \frac{}{\langle x ::= n, s\rangle \rightarrow_c \langle \varnothing, s[x \mapsto n]\rangle}$$
$$\text{SEQ.L}\ \frac{\langle C_1, s\rangle \rightarrow_c \langle C_1', s'\rangle}{\langle C_1 ; C_2, s\rangle \rightarrow_c \langle C_1' ; C_2, s'\rangle}$$
$$\text{SEQ.SKP}\ \frac{}{\langle \varnothing ; C, s\rangle \rightarrow_c \langle C, s\rangle}$$
$$\text{CND.T}\ \frac{}{\langle \texttt{tt}\ ?\ C_1 : C_2, s\rangle \rightarrow_c \langle C_1, s\rangle}$$
$$\text{CND.F}\ \frac{}{\langle \texttt{ff}\ ?\ C_1 : C_2, s\rangle \rightarrow_c \langle C_2, s\rangle}$$
$$\text{CND.B}\ \frac{\langle B, s\rangle \rightarrow_b \langle B', s'\rangle}{\langle B\ ?\ C_1 : C_2, s\rangle \rightarrow_c \langle B'\ ?\ C_1 : C_2, s'\rangle}$$
$$\text{WHILE}\ \frac{}{\langle \Updownarrow B : C, s\rangle \rightarrow_c \langle B\ ?\ (C; \Updownarrow B : C) : \varnothing, s\rangle}$$

Answer configs are of form $\langle n, s\rangle$, $\langle \texttt{bv}, s\rangle$, or $\langle \varnothing, s\rangle$. **Stuck configs** are non-answer NF configs. $\rightarrow_c$ is **not normalizing**, e.g. $\Updownarrow \texttt{tt} : \varnothing$.

An op is **strict** if it needs to eval an arg. E.g. $+_s$ is *strict*, and $\&_s$ is *left-strict*. $\&_s$:

$$\frac{B_1 \rightarrow B_1'}{B_1 \&_s B_2 \rightarrow B_1' \&_s B_2}$$
$$\frac{}{\texttt{ff} \&_s B_2 \rightarrow \texttt{ff}} \qquad \frac{}{\texttt{tt} \&_s B_2 \rightarrow B_2}$$

6 Inductive Proofs

An *inductive principle* on $\texttt{SimpExpr}$ is:

$$\forall n \in \mathbb{N}.P(n)$$
$$\wedge \forall E_1, E_2.[P(E_1) \wedge P(E_2) \implies P(E_1 +_s E_2)]$$
$$\wedge \forall E_1, E_2.[P(E_1) \wedge P(E_2) \implies P(E_1 \times_s E_2)]$$
$$\implies \forall E.P(E)$$

1. **Base Case** state ***to show*** and prove for each case. Ass arb LHS, and prove RHS.
2. **Inductive Case** state ***to show*** for $i + 1$th. Then state ***inductive hypothesis*** assuming ith, or $j \leq i + 1$th. Prove each case. The inductive hypothesis are constructed from the top of a rule.

Base Case for $E = n$ and $\mathcal{S}(E) = n$:
To Show $\forall n \in \mathbb{Z}.\forall s \in \texttt{State}.\forall m \in \mathbb{Z}.[\cdots]$. ***ONE LESS RESRICTION THAN WHAT WE ARE PROVING***
(1) $\cdots$
$\vdots$

Inductive Case for $E = E_1 + E_2$ and $\mathcal{S}(E_1 + E_2) = \mathcal{S}(E_1) + \mathcal{S}(E_2)$:
Take E_1, E_2 arbitrary.
Inductive Hypothesis: $\forall s \in \texttt{State}.\forall m \in \mathbb{Z}.[\cdots]$. ***ONE LESS RESRICTION THAN WHAT WE ARE PROVING***
To Show: *Same as in base case.*
(9) $\cdots$
$\vdots$

7 Denotational Semantics

An **expr ctx** $C^e \triangleq \cdot \,|\, E +_s C^e \,|\, C^e +_s E \,|\, \cdots$ where $\cdot$ is the **ctx hole** (e.g. $C^e[\cdot] = \cdot +_s 2$). **Ctx application** is done by filling the hole $C^e[E]$, defined recursively as:

$$(\cdot)[E] \triangleq E$$
$$(C^e +_s E')[E] \triangleq C^e[E] +_s E'$$
$$(E' +_s C^e)[E] \triangleq E' +_s C^e[E]$$

This allows to combine $\texttt{EXP.L}$ and $\texttt{EXP.R}$:

$$\text{EXP.E}\ \frac{\langle E, s\rangle \rightarrow_e \langle E', s'\rangle}{\langle C^e[E], s\rangle \rightarrow_e \langle C^e[E'], s'\rangle}$$

Weak contextual equivalence means $E_1 \sim E_2 \triangleq \forall n \in \mathbb{N}.[E_1 \rightarrow^* n \Leftrightarrow E_2 \rightarrow^* n]$, **contextual equivalence** means $E_1 \cong E_2 \triangleq \forall C^e[\cdot].[C^e[E_1] \sim C^e[E_2]]$.
Denotational semantics describe the *meaning* of a prog. **Interpretation** $[\![\_]\!] \subseteq \texttt{SimpExp} \times \mathbb{N}$ over a **semantic domain** $\mathbb{N}$:

$$[\![n]\!] \triangleq n$$
$$[\![E_1 +_s E_2]\!] \triangleq [\![E_1]\!] + [\![E_2]\!]$$

We can now prove properties on $+_s$. Also, we can define a *func between meanings*: $\forall C^e.\exists f \subseteq \mathbb{N}^2.[[\![C^e[E]]\!] = f([\![E]\!])]$. Consequently, $\forall C^e.\forall E_1, E_2.[[\![E_1]\!] = [\![E_2]\!] \implies [\![C^e[E_1]]\!] = [\![C^e[E_2]]\!]]$, and hence $[\![E]\!] = n \Leftrightarrow E \Downarrow n$. Also, $E_1 \cong E_2 \Leftrightarrow [\![E_1]\!] = [\![E_2]\!]$.
Let Σ be the set of all states, and $\Sigma_\perp \Sigma \cup \{\perp\}$. The **semantic domain of commands** is given by the set of **state transformers** $\mathcal{S}_c \triangleq [\Sigma \rightarrow \Sigma_\perp]$, a set of total functions that take an initial state, and return either a final state of $\perp$. Similarly, $\mathcal{S}_e \triangleq [\Sigma \rightarrow \mathbb{N}_\perp]$ and $\mathcal{S}_b \triangleq [\Sigma \rightarrow \mathbb{B}_\perp]$.

$[\![\_]\!]^e : \texttt{Exp} \rightarrow \mathcal{S}_e$ is defined as:

$$[\![n]\!]^e(s) \triangleq n$$
$$[\![x]\!]^e(s) \triangleq se_{lup}(x)(s)$$
$$[\![E_1 +_s E_2]\!]^e(s) \triangleq se_{plus}([\![E_1]\!]^e, [\![E_2]\!]^e)(s)$$

Lookup $se_{lup}(x)(s) = \begin{cases} s(x) & s(x) \downarrow \\ \perp & \texttt{o.w.} \end{cases}$

Addition $se_{plus}(e_1, e_2)(s) = \begin{cases} e_1(s) + e_2(s) & e_1(s) \downarrow \wedge e_2(s) \downarrow \\ \perp & \text{o.w.} \end{cases}$

$[\![\_]\!]^c : \texttt{Com} \rightarrow \mathcal{S}_c$ is defined as:

$$[\![x ::= E]\!]^c(s) \triangleq sc_{asg}([\![E]\!]^e, x)(s)$$
$$[\![\varnothing]\!]^c(s) \triangleq s$$
$$[\![C_1 ; C_2]\!]^c(s) \triangleq st_{seq}([\![C_1]\!]^c, [\![C_2]\!]^c)(s)$$
$$[\![B ? C_1 : C_2]\!]^c(s)] \triangleq st_{cnd}([\![B]\!]^b, [\![C_1]\!]^c, [\![C_2]\!]^c)(s)$$
$$[\![\Updownarrow B : C]\!]^c(s) \triangleq ([\![B]\!]^b, [\![C]\!]^c)(s)$$

Assignment $sc_{asg}(e, x)(s) = \begin{cases} s[x \mapsto e(s)] & e(s) \downarrow \\ \perp & \texttt{o.w.} \end{cases}$

Sequencing $sc_{seq}(c_1, c_2)(s) = \begin{cases} c_2(c_1(s)) & c_1(s) \downarrow \\ \perp & \texttt{o.w.} \end{cases}$

Conditional $sc_{cnd}(b, c_1, c_2)(s = \begin{cases} c_1(s) & b(s) \downarrow \wedge b(s) = \texttt{tt} \\ c_1(s) & b(s) \downarrow \wedge b(s) = \texttt{ff} \\ \perp & \texttt{o.w.} \end{cases}$

$st_{whl}(b, c) = st_{cnd}(b, st_{seq}(c, st_{whl}(b, c)), st_{skp})$, but this is not a definition. Instead:

Assume $F : (A \rightarrow A_\perp) \rightarrow (A \rightarrow A_\perp)$:

1. Define seq $\forall i \in \mathbb{N}.f_i : A \rightarrow A_\perp$ where $f_0 \triangleq \perp$ and $f_{i+1} \triangleq F(f_i)$.
2. Define func $f_\infty : A \rightarrow A_\perp$ st: $f_\infty(a) \triangleq \begin{cases} a' & \exists i.[f_i(a) = a' \neq \perp] \\ \perp & \text{o.w.} \end{cases}$
3. Ass $\forall i, a.[f_i(a) \downarrow \Rightarrow f_{i+1}(a) = f_i(a)]$.

By **fixpoint theorem**, f_∞ is a fixpoint of F. For any other fixpoint g of F, $\forall a.[f_\infty(a) \downarrow \Rightarrow f_\infty(a) = g(a)]$.

$W(c_w) \triangleq st_{cnd}(b, st_{seq}(c, c_w), st_{skp})$, so $sc_{whl}(b, c)$ is a *fixpoint* of W.

1. Define $\forall i.w_i : \mathcal{S}_c$ where $w_0 \triangleq \perp$ and $w_{i+1} \triangleq W(w_i)$.
2. Define $w_\infty : \mathcal{S}_c$ where:
$$w_\infty(s) \triangleq \begin{cases} s' & \exists i.[w_i(s) = s' \neq \perp] \\ \perp & \text{o.w.} \end{cases}$$
3. $\forall s \in \Sigma$ ass $w_i(s) \downarrow \Rightarrow w_{i+1}(s) = w_i(s)$.
4. By **FT**, $sc_{whl}(b, c) = w_\infty$.

8 Hoare Logic

A formalism relating the initial and terminating state of a program. Written with **command** C, **precond** P and **postcond** Q. It is a deductive proof system of **triples** $\{P\}\ C\ \{Q\}$. **Assertion lang** is a lang for defining state predicates and their semantics. It is an instance of first order logic with equality, with values as integers and assertions as properties of $\texttt{while}$ statements:

$$v \in \texttt{Vars} \triangleq \texttt{id} \,|\, \texttt{id}_{\texttt{log}}$$
$$t \in \texttt{Terms} \triangleq v$$
$$P, Q \in \texttt{Ass} \triangleq \texttt{tt} \,|\, \texttt{ff} \,|\, P \wedge Q \,|\, P \vee Q \,|\, P \rightarrow Q$$
$$|\, \forall \texttt{id}.P \,|\, \exists \texttt{id}.P \,|\, t_1 = t_2$$

Note that $\neg P \triangleq P \rightarrow \texttt{ff}$. A term t in state s is written as $[\![t]\!](s)$ where $[\![\cdot]\!] \subseteq \texttt{Terms} \times \texttt{State} \rightarrow \mathbb{Z}$, and $[\![P]\!]$ is the set of states that satisfy P in:

$$[\![v]\!](s) = s(v)$$
$$[\![\texttt{ff}]\!] \triangleq \emptyset$$
$$[\![\texttt{tt}]\!] \triangleq \texttt{State}$$
$$[\![P \vee Q]\!] \triangleq [\![P]\!] \cup [\![Q]\!]$$
$$[\![P \wedge Q]\!] \triangleq [\![P]\!] \cap [\![Q]\!]$$
$$[\![P \rightarrow Q]\!] \triangleq [\![\neg P \vee Q]\!]$$
$$[\![t_1 = t_2]\!] \triangleq \{s \,|\, [\![t_1]\!](s) = [\![t_2]\!](s)\}$$
$$[\![\forall \texttt{v}.P]\!] \triangleq \{s \,|\, \forall n \in \ltimes.s[v \mapsto b] \in [\![P]\!]\}$$
$$[\![\exists \texttt{v}.P]\!] \triangleq \{s \,|\, \exists n \in \ltimes.s[v \mapsto b] \in [\![P]\!]\}$$

Partial correctness states that for any asserts P, Q and command C, $\vDash \{P\}\ C\ \{Q\} \triangleq \forall s, s'.[s \in [\![P]\!] \wedge \langle C, s\rangle \Downarrow s' \Rightarrow s \in [\![Q]\!]]$.
We can define rules of Hoare logic:

$$\text{H-ASGN}\ \frac{}{\vdash \{P[E/X]\} \quad x ::= E \quad \{P\}}$$
$$\text{H-SEQ}\ \frac{\vdash \{P\}\ C_1\ \{Q\} \quad \vdash \{Q\}\ C_2\ \{R\}}{\vdash \{P\} \quad C_1 ; C_2 \quad \{R\}}$$
$$\text{H-IF}\ \frac{\vdash \{P \wedge B\}\ C_1\ \{Q\} \quad \{P \wedge \neg B\}\ C_2\ \{Q\}}{\vdash \{P\} \quad B\ ?\ C_1 : C_2 \quad \{Q\}}$$
$$\text{H-WHL}\ \frac{\vdash \{P \wedge B\} \quad C \quad \{P\}}{\vdash \{P\} \quad \Updownarrow B : C \quad \{P \wedge \neg B\}}$$
$$\text{H-CNSQ}\ \frac{\vdash_L P + s \Rightarrow P_W \quad \vdash_L Q_S \Rightarrow Q_W \quad \vdash \{P_W\} \quad C \quad \{Q_S\}}{\vdash \{P_S\} \quad C \quad \{Q_W\}}$$

Soundness means any triple that can be derived holds semantically:
$\vdash \{P\}\ C\ \{Q\} \implies \vDash \{P\}\ C\ \{Q\}$.
Completeness means any triple that holds semantically can be derived:
$\vDash \{P\}\ C\ \{Q\} \implies \vdash \{P\}\ C\ \{Q\}$.