Models of Computation II

Herbert Wiklicky

herbert@doc.ic.ac.uk or **herbert@imperial.ac.uk**

Lectures on Tuesdays and Fridays in room 311 and 308

Tutorials on Fridays (typically)

Notes, Videos, etc. on Materials, Panopto, etc. and

https://www.doc.ic.ac.uk/∼herbert/teaching.html

Thaks to Philippa Gardner and many others.

Algorithms, informally

People tried to find an algorithm to solve Hilbert's Entscheidungsproblem, without success.

A natural question was then to ask whether it was possible to **prove** that such an algorithm did not exist. To ask this question properly, it was necessary to provide a **formal** definition of algorithm.

Common features of the (historical) examples of algorithms:

- **finite** description of the procedure in terms of elementary operations;
- **deterministic**, next step is uniquely determined if there is one;
- procedure may not terminate on some input data, but we can recognise when it does terminate and **what** the **result** will be.

Algorithms as Special Functions

Turing and Church's equivalent definitions of algorithm capture the notion of **computable function**: an algorithm expects some input, does some calculation and, if it terminates, returns a unique result.

We first study **register machines**, which provide a simple definition of algorithm. We describe the **universal register machine** and introduce the **halting problem**, which is probably the most famous example of a problem that is not computable.

We then move to **Turing machines** and **Church's λ-calculus**.

Register Machines, informally

Register machines operate on natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ stored in (idealized) registers using the following "elementary operations":

- add 1 to the contents of a register
- test whether the contents of a register is 0
- subtract 1 from the contents of a register if it is non-zero
- jumps ("goto")
- conditionals ("if_then_else_")

Register Machines

Definition

A **register machine** (sometimes abbreviated to RM) is specified by:

- finitely many **registers** $R_0, R_1, \ldots, R_n$, each capable of storing a natural number;
- a **program** consisting of a finite list of instructions of the form $label : body$ where, for $i = 0, 1, 2, \ldots$, the $(i+1)^{\mathrm{th}}$ instruction has label L_i. The instruction **body** takes the form:

$R^+ \to L'$	add 1 to contents of register R and jump to instruction labelled L'
$R^- \to L', L''$	if contents of R is > 0, then subtract 1 and jump to L', else jump to L''
$HALT$	stop executing instructions

Example

Registers

$R_0 \ R_1 \ R_2$

Program

$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : HALT$

Example Computation

L_i	R_0	R_1	R_2
0	0	1	2
1	0	0	2
0	1	0	2
2	1	0	2
3	1	0	1
2	2	0	1
3	2	0	0
2	3	0	0
4	3	0	0

Register Machine Configuration

A register machine **configuration** has the form:

$$c = (\ell, r_0, \ldots, r_n)$$

where ℓ = current label and r_i = current contents of R_i.

Notation "$R_i = x$ [in configuration c]" means $c = (\ell, r_0, \ldots, r_n)$ with $r_i = x$.

Initial configurations

$$c_0 = (0, r_0, \ldots, r_n)$$

where r_i = initial contents of register R_i.

Register Machine Computation

A **computation** of a RM is a (finite or infinite) sequence of configurations

$$c_0, c_1, c_2, \ldots$$

where

- $c_0 = (0, r_0, \ldots, r_n)$ is an initial configuration;
- each $c = (\ell, r_0, \ldots, r_n)$ in the sequence determines the next configuration in the sequence (if any) by carrying out the program instruction labelled L_ℓ with registers containing r_0,...,r_n.

Halting Computations

For a finite computation $c_0, c_1, \ldots, c_m$, the last configuration $c_m = (\ell, r, \ldots)$ is a **halting** configuration: that is, the instruction labelled L_ℓ is

either $HALT$ (a ' proper halt')

or $R^+ \to L$, or $R^- \to L, L'$ with $R > 0$, or $R^- \to L', L$ with $R = 0$ and there is no instruction labelled L in the program (an 'erroneous halt')

For example, the program $\boxed{\begin{array}{l} L_0 : R_1^+ \to L_2 \\ L_1 : HALT \end{array}}$ halts erroneously.

Non-halting Computations

There are computations which never halt. For example, the program

$$\boxed{\begin{array}{l} L_0 : R_1^+ \to L_0 \\ L_1 : HALT \end{array}}$$

only has infinite computation sequences

$$(0, r), (0, r+1), (0, r+2), \ldots$$

Graphical representation

- One node in the graph for each instruction $label : body$, with the node labelled by the register of the instruction body; notation $[L]$ denotes the register of the body of label L
- Arcs represent jumps between instructions
- Initial instruction $START$.

Instruction	**Representation**
$R^+ \to L$	$R^+ \longrightarrow [L]$
$R^- \to L, L'$	$R^- \begin{array}{l} \nearrow [L] \\ \searrow [L'] \end{array}$
$HALT$	$HALT$
L_0	$START \longrightarrow [L_0]$

Example

Registers

$R_0 \ R_1 \ R_2$

Program

$L_0 : R_1^- \to L_1, L_2$

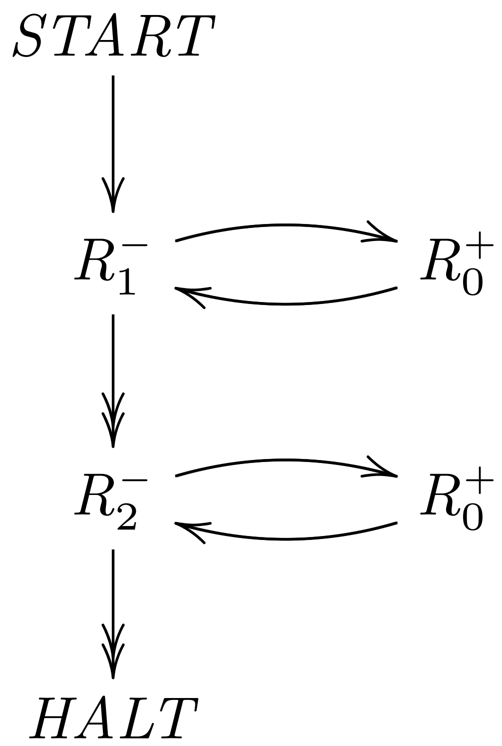
$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : HALT$

Graphical Representation



Claim: starting from initial configuration $(0, 0, x, y)$, this machine's computation halts with configuration $(4, x + y, 0, 0)$.

Partial functions

Register machine computation is **deterministic**: in any non-halting configuration, the next configuration is uniquely determined by the program.

So the relation between initial and final register contents defined by a register machine program is a **partial function**...

Definition A **partial function** from a set X to a set Y is specified by any subset $f \subseteq X \times Y$ satisfying

$$(x, y) \in f \text{ and } (x, y') \in f \text{ implies } y = y'.$$

Partial Functions

Notation

- "$f(x) = y$" means $(x, y) \in f$
- "$f(x)\downarrow$" means $\exists y \in Y\ (f(x) = y)$
- "$f(x)\uparrow$" means $\neg\exists y \in Y\ (f(x) = y)$
- $X \rightharpoonup Y$ = set of all partial functions from X to Y

  $X \to Y$ = set of all (**total**) functions from X to Y

Definition. A **partial function** from a set X to a set Y is **total** if it satisfies

$$f(x)\downarrow$$

for all $x \in X$.

Computable functions

Definition. The partial function $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ is **(register machine) computable** if there is a register machine M with at least $n + 1$ registers R_0, R_1, ..., R_n (and maybe more) such that for all $(x_1, \ldots, x_n) \in \mathbb{N}^n$ and all $y \in \mathbb{N}$,

> the computation of M starting with $R_0 = 0$, $R_1 = x_1$, ..., $R_n = x_n$ and all other registers set to 0, halts with $R_0 = y$

if and only if $f(x_1, \ldots, x_n) = y$.

Example

Registers

$R_0 \;\; R_1 \;\; R_2$

Program

$L_0 : R_1^- \to L_1, L_2$

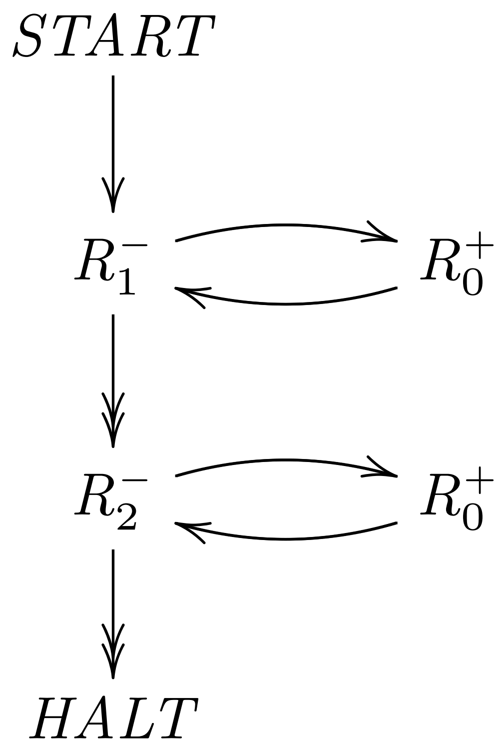
$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

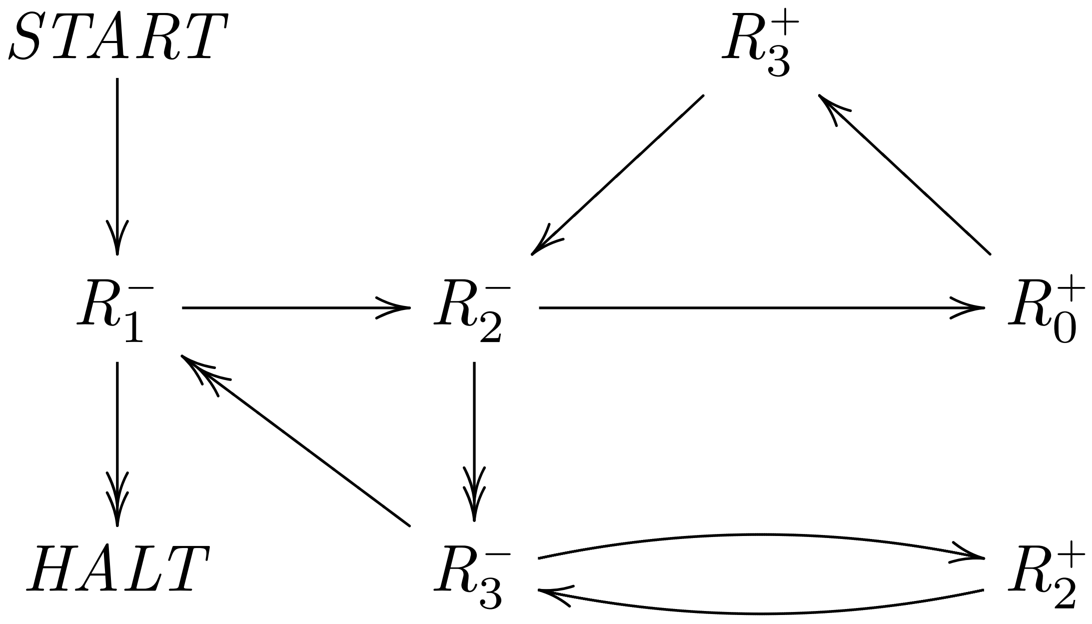
$L_4 : HALT$

Graphical Representation



If the machine starts with registers $(R_0, R_1, R_2) = (0, x, y)$, then it halts with registers $(R_0, R_1, R_2) = (x + y, 0, 0)$.

Multiplication $f(x,y) \triangleq xy$ is computable



If the machine starts with registers $(R_0, R_1, R_2, R_3) = (0, x, y, 0)$, then it halts with registers $(R_0, R_1, R_2, R_3) = (xy, 0, y, 0)$.

The Halting Problem

The Halting Problem is the decision problem with

- the set S of all pairs (A, D), where A is an algorithm and D is some input datum on which the algorithm is designed to operate;
- the property $A(D)\downarrow$ holds for $(A, D) \in S$ if algorithm A when applied to D eventually produces a result: that is, eventually halts.

Turing and Church's work shows that the Halting Problem is unsolvable (undecidable): that is, there is no algorithm H such that, for all $(A, D) \in S$,

$$\begin{array}{rcll} H(A, D) & = & 1 & A(D)\downarrow \\ & = & 0 & \text{otherwise} \end{array}$$

Numerical Coding of Pairs

Definition

For $x, y \in \mathbb{N}$, define $\begin{cases} \langle\!\langle x, y \rangle\!\rangle \triangleq 2^x(2y+1) \\ \langle x, y \rangle \triangleq 2^x(2y+1) - 1 \end{cases}$

Example $27 = 0\mathtt{b}11011 = \langle\!\langle 0, 13 \rangle\!\rangle = \langle 2, 3 \rangle$

Result

$\langle\!\langle -, - \rangle\!\rangle$ gives a bijection between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}^+ = \{n \in \mathbb{N} \mid n \neq 0\}$.

$\langle -, - \rangle$ gives a bijection between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$.

Recall the definition of bijection from discrete maths.

Numerical Coding of Pairs

Definition

For $x, y \in \mathbb{N}$, define $\begin{cases} \langle\langle x, y \rangle\rangle & \triangleq & 2^x(2y+1) \\ \langle x, y \rangle & \triangleq & 2^x(2y+1) - 1 \end{cases}$

Sketch Proof of Result

It is enough to observe that

$$\boxed{0\mathrm{b}\langle\langle x, y \rangle\rangle} = \boxed{0\mathrm{b}y \mid 1 \mid 0 \cdots 0} \quad x \text{ number of } 0\text{s}$$

$$\boxed{0\mathrm{b}\langle x, y \rangle} = \boxed{0\mathrm{b}y \mid 0 \mid 1 \cdots 1} \quad x \text{ number of } 1\text{s}$$

where $0\mathrm{b}x \triangleq x$ in binary. $\triangleq$ means 'is defined to be'.

Numerical Coding of Lists

Let $List\,\mathbb{N}$ be the set of all finite lists of natural numbers, defined by:

- **empty list**: $[\,]$
- **list cons**: $x :: \ell \in List\,\mathbb{N}$ if $x \in \mathbb{N}$ and $\ell \in List\,\mathbb{N}$

Notation: $[x_1, x_2, \ldots, x_n] \;\triangleq\; x_1 :: (x_2 :: (\cdots x_n :: [] \cdots))$

Numerical Coding of Lists

Let $List\,\mathbb{N}$ be the set of all finite lists of natural numbers.

For $\ell \in List\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [] \urcorner & \triangleq \; 0 \\ \ulcorner x :: \ell \urcorner & \triangleq \; \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = 2^x (2 \cdot \ulcorner \ell \urcorner + 1) \end{cases}$$

Thus, $\ulcorner [x_1, x_2, \ldots, x_n] \urcorner = \langle\!\langle x_1, \langle\!\langle x_2, \cdots \langle\!\langle x_n, 0 \rangle\!\rangle \cdots \rangle\!\rangle \rangle\!\rangle$

Numerical Coding of Lists

Let $\mathit{List}\,\mathbb{N}$ be the set of all finite lists of natural numbers.

For $\ell \in \mathit{List}\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [] \urcorner & \triangleq \ 0 \\ \ulcorner x :: \ell \urcorner & \triangleq \ \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = 2^x(2 \cdot \ulcorner \ell \urcorner + 1) \end{cases}$$

Examples

$$\ulcorner [3] \urcorner = \ulcorner 3 :: [] \urcorner = \langle\!\langle 3, 0 \rangle\!\rangle = 2^3(2 \cdot 0 + 1) = 8$$

$$\ulcorner [1,3] \urcorner = \langle\!\langle 1, \ulcorner [3] \urcorner \rangle\!\rangle = \langle\!\langle 1, 8 \rangle\!\rangle = 34$$

$$\ulcorner [2,1,3] \urcorner = \langle\!\langle 2, \ulcorner [1,3] \urcorner \rangle\!\rangle = \langle\!\langle 2, 34 \rangle\!\rangle = 276$$

Numerical Coding of Lists

Let $List\,\mathbb{N}$ be the set of all finite lists of natural numbers.

For $\ell \in List\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [\,] \urcorner & \triangleq \; 0 \\ \ulcorner x :: \ell \urcorner & \triangleq \; \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = 2^x (2 \cdot \ulcorner \ell \urcorner + 1) \end{cases}$$

Result The function $\ell \mapsto \ulcorner \ell \urcorner$ gives a bijection from $List\,\mathbb{N}$ to $\mathbb{N}$.

Numerical Coding of Lists

Let $\mathit{List}\,\mathbb{N}$ be the set of all finite lists of natural numbers.

For $\ell \in \mathit{List}\,\mathbb{N}$, define $\ulcorner \ell \urcorner \in \mathbb{N}$ by induction on the length of the list ℓ:

$$\begin{cases} \ulcorner [] \urcorner & \triangleq \; 0 \\ \ulcorner x :: \ell \urcorner & \triangleq \; \langle\!\langle x, \ulcorner \ell \urcorner \rangle\!\rangle = 2^x (2 \cdot \ulcorner \ell \urcorner + 1) \end{cases}$$

Result The function $\ell \mapsto \ulcorner \ell \urcorner$ gives a bijection from $\mathit{List}\,\mathbb{N}$ to $\mathbb{N}$.

Sketch Proof

The proof follows by observing that

$$\boxed{0\mathrm{b}\ulcorner [x_1, x_2, \ldots, x_n] \urcorner} = \boxed{1 \;\big|\; \underbrace{0 \cdots 0}_{x_n\,0s}}\;\boxed{1 \;\big|\; \underbrace{0 \cdots 0}_{x_{n-1}\,0s}} \cdots \boxed{1 \;\big|\; \underbrace{0 \cdots 0}_{x_1\,0s}}$$

Recall Register Machines

Definition

A **register machine** (sometimes abbreviated to RM) is specified by:

- finitely many **registers** $R_0, R_1, \ldots, R_n$, each capable of storing a natural number;
- a **program** consisting of a finite list of instructions of the form $label : body$ where, for $i = 0, 1, 2, \ldots$, the $(i+1)^{\mathrm{th}}$ instruction has label L_i. The instruction **body** takes the form:

$R^+ \to L'$	add 1 to contents of register R and jump to instruction labelled L'
$R^- \to L', L''$	if contents of R is > 0, then subtract 1 and jump to L', else jump to L''
$HALT$	stop executing instructions

Numerical Coding of Programs

If P is the RM program $\boxed{\begin{array}{c} L_0 : body_0 \\ L_1 : body_1 \\ \vdots \\ L_n : body_n \end{array}}$ then its numerical code is

$$\ulcorner P \urcorner \triangleq \ulcorner [\ulcorner body_0 \urcorner, \ldots, \ulcorner body_n \urcorner] \urcorner$$

where the numerical code $\ulcorner body \urcorner$ of an instruction body is defined

by: $$\begin{cases} \ulcorner R_i^+ \to L_j \urcorner & \triangleq \langle\!\langle 2i, j \rangle\!\rangle \\ \ulcorner R_i^- \to L_j, L_k \urcorner & \triangleq \langle\!\langle 2i+1, \langle j, k \rangle \rangle\!\rangle \\ \ulcorner HALT \urcorner & \triangleq 0 \end{cases}$$

Recall Addition $f(x, y) \triangleq x + y$ is Computable

Registers

$R_0 \;\; R_1 \;\; R_2$

Program

$L_0 : R_1^- \to L_1, L_2$

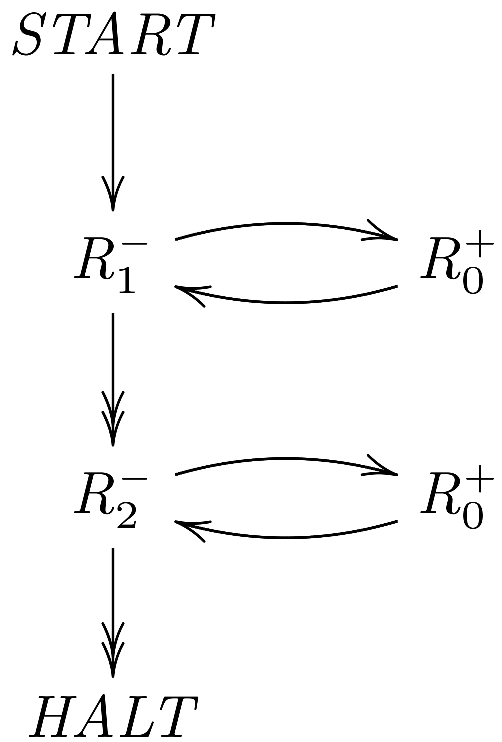
$L_1 : R_0^+ \to L_0$

$L_2 : R_2^- \to L_3, L_4$

$L_3 : R_0^+ \to L_2$

$L_4 : HALT$

Graphical Representation



If the machine starts with registers $(R_0, R_1, R_2) = (0, x, y)$, it halts with registers $(R_0, R_1, R_2) = (x + y, 0, 0)$.

Coding of the RM for Addition

$\ulcorner P \urcorner \triangleq \ulcorner [\ulcorner B_0 \urcorner, \ldots, \ulcorner B_4 \urcorner] \urcorner$ where

$$
\begin{aligned}
\ulcorner B_0 \urcorner &= \ulcorner R_1^- \to L_1, L_2 \urcorner = \langle\langle (2 \times 1) + 1, \langle 1, 2 \rangle \rangle\rangle \\
&= \langle\langle 3, 9 \rangle\rangle = 8 \times (18 + 1) = 152 \\
\ulcorner B_1 \urcorner &= \ulcorner R_0^+ \to L_0 \urcorner = \langle\langle 2 \times 0, 0 \rangle\rangle = 1 \\
\ulcorner B_2 \urcorner &= \ulcorner R_2^- \to L_3, L_4 \urcorner = \langle\langle (2 \times 2) + 1, \langle 3, 4 \rangle \rangle\rangle \\
&= \langle\langle 5, (8 \times 9) - 1 \rangle\rangle = \langle\langle 5, 71 \rangle\rangle \\
&= 2^5 \times ((2 \times 71) + 1) = 32 \times 143 = 4576 \\
\ulcorner B_3 \urcorner &= \ulcorner R_0^+ \to L_2 \urcorner = \langle\langle 2 \times 0, 2 \rangle\rangle = 5 \\
\ulcorner B_4 \urcorner &= \ulcorner HALT \urcorner = 0
\end{aligned}
$$

Decoding Numbers as Bodies and Programs

Any $x \in \mathbb{N}$ decodes to a unique instruction $body(x)$:

if $x = 0$ then $body(x)$ is $HALT$,
else ($x > 0$ and) let $x = \langle\!\langle y, z \rangle\!\rangle$ in
if $y = 2i$ is even, then $body(x)$ is $R_i^+ \to L_z$,
else $y = 2i + 1$ is odd, let $z = \langle j, k \rangle$ in
$body(x)$ is $R_i^- \to L_j, L_k$

So any $e \in \mathbb{N}$ decodes to a unique program $prog(e)$, called the register machine **program with index** e:

$$prog(e) \triangleq \boxed{\begin{array}{c} L_0 : body(x_0) \\ \vdots \\ L_n : body(x_n) \end{array}} \text{ where } e = \ulcorner [x_0, \dots, x_n] \urcorner$$

Example of $prog(e)$

- $786432 = 2^{19} + 2^{18} = 0\mathtt{b}11\underbrace{0\ldots0}_{18\text{ "0"}s} = \ulcorner[18, 0]\urcorner$
- $18 = 0\mathtt{b}10010 = \langle\!\langle 1, 4\rangle\!\rangle = \langle\!\langle 1, \langle 0, 2\rangle\rangle\!\rangle = \ulcorner R_0^- \to L_0, L_2\urcorner$
- $0 = \ulcorner HALT\urcorner$

So $prog(786432) = \boxed{\begin{array}{l} L_0 : R_0^- \to L_0, L_2 \\ L_1 : HALT \end{array}}$