Definition. A register machine 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 \( \text{label : body} \), where for \( i = 0, 1, 2, \ldots \), the \((i + 1)^{\text{th}}\) instruction has label \( L_i \).

Instruction \text{body} takes one of three forms:

| \( R^+ \to L' \) | 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 from it and jump to \( L' \), else jump to \( L'' \) |
| \text{HALT} | stop executing instructions |
Recall: **Computable functions**

**Definition.** \( f \in \mathbb{N}^n \rightarrow \mathbb{N} \) is (register machine) **computable** if there is a register machine \( M \) with at least \( n + 1 \) registers \( R_0, R_1, \ldots, R_n \) (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 \( R_0 = 0 \), \( R_1 = x_1, \ldots, R_n = x_n \) and all other registers set to 0, halts with \( R_0 = y \)

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

**N.B.** there may be many different \( M \) that compute the same partial function \( f \).
Coding programs as numbers
Turing/Church solution of the Entscheidungsproblem uses the idea that *(formal descriptions of) algorithms can be the data on which algorithms act.*

To realize this idea with Register Machines we have to be able to code RM programs as numbers. *(In general, such codings are often called Gödel numberings.)*
"Effective" numerical codes

RM program

initial contents of R1, ..., Rn

Prog, [x₁, ..., xₙ] → y

final contents of RO (if halts)

run the RM
"Effective" numerical codes

\[ \text{Prog}, [x_1, \ldots, x_n] \rightarrow y \]

\[ \text{code} \downarrow \uparrow \text{decode} \]

\[ \langle \text{Prog}, [x_1, \ldots, x_n] \rangle \]

Want numerical codings

\[ \langle -,- \rangle, \, [-], \, [\cdots] \]

So that

- decode \rightarrow run

is RM computable
Numerical coding of pairs

\[ \{0,1,2,3,\ldots\} \]

For \( x, y \in \mathbb{N} \), define

\[
\begin{align*}
\langle x, y \rangle & \triangleq 2^x (2y + 1) \\
\langle x, y \rangle & \triangleq 2^x (2y + 1) - 1
\end{align*}
\]

Left-hand side is equal to the right-hand side by definition
Numerical coding of pairs

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

\[
\begin{array}{c|cccccc}
\langle x, y \rangle & 0 & 1 & 2 & \ldots \\
0 & 1 & 3 & 5 & \ldots \\
1 & 2 & 6 & 10 & \ldots \\
2 & 4 & 12 & 20 & \ldots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
\quad
\begin{array}{c|cccccc}
\langle x, y \rangle & 0 & 1 & 2 & \ldots \\
0 & 0 & 2 & 4 & \ldots \\
1 & 1 & 5 & 9 & \ldots \\
2 & 3 & 11 & 19 & \ldots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
\]
Numerical coding of pairs

For \( x, y \in \mathbb{N} \), define

\[
\begin{align*}
\langle x, y \rangle & \triangleq 2^x(2y + 1) \\
\langle x, y \rangle & \triangleq 2^x(2y + 1) - 1
\end{align*}
\]

So

\[
\begin{align*}
0b\langle x, y \rangle & = 0by10\ldots0 \\
0b\langle x, y \rangle & = 0by01\ldots1
\end{align*}
\]

(Notation: \( 0bx \triangleq x \text{ in binary.} \))

E.g. \( 27 = 0b11011 = \langle 0, 13 \rangle = \langle 2, 3 \rangle \)
Numerical coding of pairs

For \( x, y \in \mathbb{N} \), define

\[
\langle x, y \rangle \triangleq 2^x(2y + 1)
\]

\[
\langle x, y \rangle \triangleq 2^x(2y + 1) - 1
\]

So

\[
\begin{align*}
0b\langle x, y \rangle & = 0by10\cdots0 \\
0b\langle x, y \rangle & = 0by01\cdots1
\end{align*}
\]

\( \langle -, - \rangle \) gives a bijection (one-one correspondence) between \( \mathbb{N} \times \mathbb{N} \) and \( \mathbb{N} \).

\( \langle -, - \rangle \) gives a bijection between \( \mathbb{N} \times \mathbb{N} \) and \( \{n \in \mathbb{N} \mid n \neq 0\} \).
Numerical coding of lists

\[ \text{list } \mathbb{N} \triangleq \text{set of all finite lists of natural numbers, using ML notation for lists:} \]

- empty list: \([\,\,]\)
- list-cons: \(x :: \ell \in \text{list } \mathbb{N}\) (given \(x \in \mathbb{N}\) and \(\ell \in \text{list } \mathbb{N}\))
- \([x_1, x_2, \ldots, x_n] \triangleq x_1 :: (x_2 :: (\cdots x_n :: [\,\,] \cdots ))\)
**Numerical coding of lists**

\( \text{list } \mathbb{N} \triangleq \) set of all finite lists of natural numbers, using ML notation for lists.

For \( \ell \in \text{list } \mathbb{N} \), define \( \llbracket \ell \rrbracket \in \mathbb{N} \) by induction on the length of the list \( \ell \):

\[
\begin{align*}
\llbracket \emptyset \rrbracket & \triangleq 0 \\
\llbracket x :: \ell \rrbracket & \triangleq \langle x, \llbracket \ell \rrbracket \rangle = 2^x(2 \cdot \llbracket \ell \rrbracket + 1)
\end{align*}
\]

Thus \( \llbracket [x_1, x_2, \ldots, x_n] \rrbracket = \langle x_1, \langle x_2, \ldots \langle x_n, 0 \rangle \ldots \rangle \rangle \)
Numerical coding of lists

\[ \text{list} \mathbb{N} \triangleq \text{set of all finite lists of natural numbers, using ML notation for lists.} \]

For \( \ell \in \text{list} \mathbb{N} \), define \( \mathbf{\ell} \in \mathbb{N} \) by induction on the length of the list \( \ell \):

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

For example:

\[ \mathbf{[3]} = \langle 3, \mathbf{[]} \rangle = \langle 3, 0 \rangle = 2^3 (2 \cdot 0 + 1) = 8 = 0b1000 \]

\[ \mathbf{[1,3]} = \langle 1, \mathbf{[3]} \rangle = \langle 1, 8 \rangle = 34 = 0b100010 \]

\[ \mathbf{[2,1,3]} = \langle 2, \mathbf{[1,3]} \rangle = \langle 2, 34 \rangle = 276 = 0b100010100 \]
Numerical coding of lists

\[ \text{list } \mathbb{N} \triangleq \text{set of all finite lists of natural numbers, using ML notation for lists.} \]

For \( \ell \in \text{list } \mathbb{N} \), define \( \bigcirc \ell \in \mathbb{N} \) by induction on the length of the list \( \ell \):

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

For example:

\[
\begin{align*}
\bigcirc [3] & = \bigcirc 3 :: [] = \langle 3, 0 \rangle = 2^3 (2 \cdot 0 + 1) = 8 = 0b1000 \\
\bigcirc [1, 3] & = \langle 1, \bigcirc [3] \rangle = \langle 1, 8 \rangle = 34 = 0b100010 \\
\bigcirc [2, 1, 3] & = \langle 2, \bigcirc [1, 3] \rangle = \langle 2, 34 \rangle = 276 = 0b100010100
\end{align*}
\]
Numerical coding of lists

\( \text{list } \mathbb{N} \triangleq \) set of all finite lists of natural numbers, using ML notation for lists.

For \( \ell \in \text{list } \mathbb{N} \), define \( \langle \ell \rangle \in \mathbb{N} \) by induction on the length of the list \( \ell \):

\[
\begin{align*}
\langle [] \rangle & \trianglerighteq 0 \\
\langle x :: \ell \rangle & \trianglerighteq \langle x, \langle \ell \rangle \rangle = 2^x (2 \cdot \langle \ell \rangle + 1)
\end{align*}
\]

\( 0 \nabla \langle [x_1, x_2, \ldots, x_n] \rangle = \begin{array}{ccccccc}
1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0
\end{array} \)

\( x_n \text{ 0s} \)

\( x_{n-1} \text{ 0s} \)

\( x_1 \text{ 0s} \)
**Numerical coding of lists**

\[\text{list } \mathbb{N} \triangleq \text{ set of all finite lists of natural numbers, using ML notation for lists.}\]

For \( \ell \in \text{list } \mathbb{N} \), define \( \llbracket \ell \rrbracket \in \mathbb{N} \) by induction on the length of the list \( \ell \):

\[
\begin{cases}
\llbracket [] \rrbracket \triangleq 0 \\
\llbracket x :: \ell \rrbracket \triangleq \llbracket x, \ell \rrbracket = 2^x (2 \cdot \llbracket \ell \rrbracket + 1)
\end{cases}
\]

\[
[0^b] \llbracket [x_1, x_2, \ldots, x_n] \rrbracket = \begin{bmatrix} 1 & 0 \cdots 0 & 1 & 0 \cdots 0 & \cdots & 1 & 0 \cdots 0 \end{bmatrix}
\]

Hence \( \ell \leftrightarrow \llbracket \ell \rrbracket \) gives a bijection from \( \text{list } \mathbb{N} \) to \( \mathbb{N} \).
Numerical coding of programs

If $P$ is the RM program

\[
\begin{align*}
\mathbb{L}_0 : & body_0 \\
\mathbb{L}_1 : & body_1 \\
& \vdots \\
\mathbb{L}_n : & body_n
\end{align*}
\]

then its numerical code is

\[
\llbracket P \rrbracket \triangleq \llbracket body_0 \rrbracket, \ldots, \llbracket body_n \rrbracket
\]

where the numerical code \(\llbracket body \rrbracket\) of an instruction body is defined by:

\[
\begin{align*}
\llbracket R_i \rightarrow L_j \rrbracket & \triangleq \langle 2i, j \rangle \\
\llbracket R_i \leftarrow L_j, L_k \rrbracket & \triangleq \langle 2i + 1, \langle j, k \rangle \rangle \\
\llbracket \text{HALT} \rrbracket & \triangleq 0
\end{align*}
\]
Any $x \in \mathbb{N}$ decodes to a unique instruction $\text{body}(x)$:

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

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

\[
\text{prog}(e) \triangleq \begin{cases} 
L_0 : \text{body}(x_0) \\
\vdots \\
L_n : \text{body}(x_n) 
\end{cases}
\text{ where } e = \left[ x_0, \ldots, x_n \right] \]
Example of $prog(e)$

$786432 = 2^{19} + 2^{18} = 0b110\ldots0 = [18, 0]$

18 "0"s

$18 = 0b10010 = \langle 1, 4 \rangle = \langle 1, \langle 0, 2 \rangle \rangle = R_0^- \rightarrow L_0, L_2$

$0 = \text{HALT}$

So $prog(786432) = \begin{cases} 
L_0 : R_0^- \rightarrow L_0, L_2 \\
L_1 : \text{HALT}
\end{cases}$
Example of \( \text{prog}(e) \)

- \( 786432 = 2^{19} + 2^{18} = 0b110\ldots0 = \llbracket 18, 0 \rrbracket \)
  - 18 "0"s
- \( 18 = 0b10010 = \langle 1, 4 \rangle = \langle 1, \langle 0, 2 \rangle \rangle = \llbracket R_0^- \rightarrow L_0, L_2 \rrbracket \)
- \( 0 = \llbracket \text{HALT} \rrbracket \)

So \( \text{prog}(786432) = \begin{cases} L_0 : R_0^- \rightarrow L_0, L_2 \\ L_1 : \text{HALT} \end{cases} \)

N.B. jump to label with no body (erroneous halt)

What function is computed by a RM with \( \text{prog}(786432) \) as its program?
$666 = \begin{array}{c}
\text{0b1010011010} \\
= \begin{bmatrix} 1, 1, 0, 2, 1 \end{bmatrix}
\end{array}$

$\text{prog}(666) = \begin{bmatrix}
L_0 : R_0^+ \rightarrow L_0 \\
L_4 : R_0^+ \rightarrow L_0 \\
L_2 : \text{HALT} \\
L_3 : R_0^- \rightarrow L_0, L_0 \\
L_4 : R_0^+ \rightarrow L_0
\end{bmatrix}$

(never halts!)

What partial function does this compute?
Example of $\text{prog}(e)$

- $786432 = 2^{19} + 2^{18} = 0b110\ldots0 = \langle[18, 0]\rangle$
  \[18 \text{ "0"s}\]
- $18 = 0b10010 = \langle1, 4\rangle = \langle1, \langle0, 2\rangle\rangle = \langle R_0^- \rightarrow L_0, L_2 \rangle$
- $0 = \langle \text{HALT} \rangle$

So $\text{prog}(786432) = \begin{cases} L_0 : R_0^- \rightarrow L_0, L_2 \\ L_1 : \text{HALT} \end{cases}$

N.B. In case $e = 0$ we have $0 = \langle[]\rangle$, so $\text{prog}(0)$ is the program with an empty list of instructions, which by convention we regard as a RM that does nothing (i.e. that halts immediately).
"Effective" numerical codes

\[ \text{Prog}, [x_1, \ldots, x_n] \rightarrow y \]

code \quad \downarrow \quad \uparrow \text{decode}

\langle \text{Prog}, [x_1, \ldots, x_n] \rangle

a number

Want numerical codings

\langle \cdots \rangle, \leftarrow \rightarrow, \left[ \cdots \right] \rightarrow \left[ \left[ \cdots \right] \right]

So that

\text{decode} \quad \text{run}

is RM computable
Universal register machine, $U$
High-level specification

Universal RM $U$ carries out the following computation, starting with $R_0 = 0$, $R_1 = e$ (code of a program), $R_2 = a$ (code of a list of arguments) and all other registers zeroed:

- decode $e$ as a RM program $P$
- decode $a$ as a list of register values $a_1, \ldots, a_n$
- carry out the computation of the RM program $P$ starting with $R_0 = 0, R_1 = a_1, \ldots, R_n = a_n$ (and any other registers occurring in $P$ set to 0).
Mnemonics for the registers of $U$ and the role they play in its program:

\begin{align*}
R_1 & \equiv P \text{ code of the RM to be simulated} \\
R_2 & \equiv A \text{ code of current register contents of simulated RM} \\
R_3 & \equiv PC \text{ program counter—number of the current instruction} \\
& \quad \text{(counting from 0)} \\
R_4 & \equiv N \text{ code of the current instruction body} \\
R_5 & \equiv C \text{ type of the current instruction body} \\
R_6 & \equiv R \text{ current value of the register to be incremented or decremented by current instruction (if not } HALT) \\
R_7 & \equiv S, \ R_8 \equiv T \text{ and } R_9 \equiv Z \text{ are auxiliary registers.}
\end{align*}
Overall structure of U’s program

1. copy PCth item of list in P to N (halting if PC > length of list); goto 2

2. if N = 0 then copy 0th item of list in A to R₀ and halt, else (decode N as ⟨y, z⟩; C ::= y; N ::= z; goto 3)

{at this point either C = 2i is even and current instruction is Rᵢ⁺ → Lᵢ, or C = 2i + 1 is odd and current instruction is Rᵢ⁻ → Lᵢ, Lⱼ, Lₖ where z = ⟨j, k⟩}

3. copy ith item of list in A to R; goto 4

4. execute current instruction on R; update PC to next label; restore register values to A; goto 1
Overall structure of **U**’s program

1. copy PC\textsuperscript{th} item of list in **P** to **N** (halting if PC > length of list); goto 2

2. if N = 0 then copy 0\textsuperscript{th} item of list in **A** to **R\textsubscript{0}** and halt, else (decode N as \(\langle y, z \rangle\); C := y; N := z; goto 3)

\{at this point either C = 2i is even and current instruction is \(R_i^+ \rightarrow L_z\), or C = 2i + 1 is odd and current instruction is \(R_i^- \rightarrow L_j, L_k\) where z = \(\langle j, k \rangle\}\}

3. copy i\textsuperscript{th} item of list in **A** to **R**; goto 4

4. execute current instruction on **R**; update PC to next label; restore register values to **A**; goto 1

To implement this, we need RMs for manipulating (codes of) lists of numbers...