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

Thanks to Philippa Gardner and many others.
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.
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 $\lambda$-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_")
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 $\text{label : body}$ where, for $i = 0, 1, 2, \ldots$, the $(i + 1)\text{th}$ instruction has label $L_i$. The instruction body takes the form:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^+ \rightarrow L'$</td>
<td>add 1 to contents of register $R$ and jump to instruction labelled $L'$</td>
</tr>
<tr>
<td>$R^- \rightarrow L', L''$</td>
<td>if contents of $R$ is $&gt; 0$, then subtract 1 and jump to $L'$, else jump to $L''$</td>
</tr>
<tr>
<td>$\text{HALT}$</td>
<td>stop executing instructions</td>
</tr>
</tbody>
</table>
### Example

#### Registers

- $R_0$, $R_1$, $R_2$

#### Program

- $L_0: R_1^- \rightarrow L_1, L_2$
- $L_1: R_0^+ \rightarrow L_0$
- $L_2: R_2^- \rightarrow L_3, L_4$
- $L_3: R_0^+ \rightarrow L_2$
- $L_4: HALT$

#### Example Computation

<table>
<thead>
<tr>
<th>$L_i$</th>
<th>$R_0$</th>
<th>$R_1$</th>
<th>$R_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
A register machine configuration has the form:

\[ c = (\ell, r_0, \ldots, r_n) \]

where \( \ell \) = 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 \( R_i \).
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 \( L_\ell \) with registers containing \( r_0, \ldots, r_n \).
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 $L_\ell$ is

**either** $HALT$ (a ‘proper halt’)

**or** $R^+ \rightarrow L$, or $R^- \rightarrow L, L'$ with $R > 0$, or $R^- \rightarrow L', L$ with $R = 0$

and there is no instruction labelled $L$ in the program (an ‘erroneous halt’)

For example, the program

$L_0 : R_1^+ \rightarrow L_2$

$L_1 : HALT$

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

\[ L_0 : R_1^+ \rightarrow L_0, \]

\[ L_1 : HALT \]

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*.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R^+ \rightarrow L)</td>
<td>(R^+ \rightarrow [L])</td>
</tr>
<tr>
<td>(R^- \rightarrow L, L')</td>
<td>(R^- \rightarrow [L]) (R^- \rightarrow [L'])</td>
</tr>
<tr>
<td><em>HALT</em></td>
<td><em>HALT</em></td>
</tr>
<tr>
<td>(L_0)</td>
<td><em>START</em> (\rightarrow [L_0])</td>
</tr>
</tbody>
</table>
**Example**

**Registers**

$R_0 \quad R_1 \quad R_2$

**Program**

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

$L_1 : R_0^+ \rightarrow L_0$

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

$L_3 : R_0^+ \rightarrow L_2$

$L_4 : HALT$

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

---

**Graphical Representation**

```
START

R_1^- \leftrightarrow R_0^+

R_2^- \leftrightarrow R_0^+

HALT
```
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

$$(x, y) \in f \text{ and } (x, y') \in f \implies y = y'.$$
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 \(\neg \exists y \in Y \ (f(x) = y)\)
- \(X \rightarrow Y\) = set of all partial functions from \(X\) to \(Y\)
- \(X \rightarrow^\text{total} Y\) = 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 $f \in \mathbb{N}^n \rightarrow \mathbb{N}$ 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 $f(x_1, \ldots, x_n) = y$. 
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 $(A, D) \in S$,

$$H(A, D) = 1 \quad A(D) \downarrow$$
$$= 0 \quad \text{otherwise}$$
Numerical Coding of Pairs

Definition

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

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

Example

\( 27 = 0b11011 = \langle 0, 13 \rangle = \langle 2, 3 \rangle \)

Result

\( \langle -, - \rangle \) 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

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

**Sketch Proof of Result**

It is enough to observe that

\[
\begin{align*}
0b \langle x, y \rangle &= 0by 1 0 \cdots 0 & \text{x number of 0s} \\
0b \langle x, y \rangle &= 0by 0 1 \cdots 1 & \text{x number of 1s}
\end{align*}
\]

where $0bx \triangleq x$ 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**: $[x_1, x_2, \ldots, x_n] \triangleq x_1 :: (x_2 :: (\cdots x_n :: [] \cdots))$
Let $\text{List } \mathbb{N}$ be the set of all finite lists of natural numbers.

For $\ell \in \text{List } \mathbb{N}$, define $\lceil \ell \rceil \in \mathbb{N}$ by induction on the length of the list

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

Thus, $\lceil [x_1, x_2, \ldots, x_n] \rceil = \langle x_1, \langle x_2, \cdots \langle x_n, 0 \rangle \cdots \rangle \rangle$
Let \( \text{List N} \) be the set of all finite lists of natural numbers.

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

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

Examples

\[
\begin{align*}
\llbracket [3] \rrbracket & = \llbracket 3 :: \emptyset \rrbracket = \langle 3, 0 \rangle = 2^3 (2 \cdot 0 + 1) = 8 \\
\llbracket [1, 3] \rrbracket & = \langle 1, \llbracket [3] \rrbracket \rangle = \langle 1, 8 \rangle = 34 \\
\llbracket [2, 1, 3] \rrbracket & = \langle 2, \llbracket [1, 3] \rrbracket \rangle = \langle 2, 34 \rangle = 276
\end{align*}
\]
Let $List \mathbb{N}$ be the set of all finite lists of natural numbers.

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

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

**Result** The function $\ell \mapsto \ceiling{\ell}$ gives a bijection from $List \mathbb{N}$ to $\mathbb{N}$. 
**Numerical Coding of Lists**

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

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

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

**Result** The function $\ell \mapsto \llbracket \ell \rrbracket$ gives a bijection from $\text{List } \mathbb{N}$ to $\mathbb{N}$.

**Sketch Proof**

The proof follows by observing that

\[
0b^{\llbracket [x_1, x_2, \ldots, x_n] \rrbracket} = 1 \begin{array}{l}
\overline{0 \cdots 0} \\
\overline{x_n 0s}
\end{array} 1 \begin{array}{l}
\overline{0 \cdots 0} \\
\overline{x_{n-1} 0s}
\end{array} \cdots 1 \begin{array}{l}
\overline{0 \cdots 0} \\
\overline{x_1 0s}
\end{array}
\]
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 $\text{label : body}$ where, for $i = 0, 1, 2, \ldots$, the $(i + 1)^{\text{th}}$ instruction has label $L_i$. The instruction body takes the form:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^+ \rightarrow L'$</td>
<td>add 1 to contents of register $R$ and jump to instruction labelled $L'$</td>
</tr>
<tr>
<td>$R^- \rightarrow L', L''$</td>
<td>if contents of $R$ is $&gt; 0$, then subtract 1 and jump to $L'$, else jump to $L''$</td>
</tr>
<tr>
<td>$\text{HALT}$</td>
<td>stop executing instructions</td>
</tr>
</tbody>
</table>
If $P$ is the RM program

\[
\begin{align*}
L_0 &: \text{body}_0 \\
L_1 &: \text{body}_1 \\
&\quad \vdots \\
L_n &: \text{body}_n
\end{align*}
\]

then its numerical code is

\[
[\llbracket \text{body}_0 \rrbracket, \ldots, \llbracket \text{body}_n \rrbracket]\]

where the numerical code $\llbracket \text{body} \rrbracket$ of an instruction body is defined by:

\[
\begin{align*}
\llbracket R^+_i \rightarrow L_j \rrbracket &\triangleq \llbracket \langle 2i, j \rangle \rrbracket \\
\llbracket R^-_i \rightarrow L_j, L_k \rrbracket &\triangleq \llbracket \langle 2i + 1, \langle j, k \rangle \rangle \rrbracket \\
\llbracket \text{HALT} \rrbracket &\triangleq 0
\end{align*}
\]
Recall Addition $f(x, y) \triangleq x + y$ is Computable

<table>
<thead>
<tr>
<th>Registers</th>
<th>Graphical Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_0$ $R_1$ $R_2$</td>
<td>START $\downarrow$</td>
</tr>
<tr>
<td>$L_0 : R_1^- \rightarrow L_1, L_2$</td>
<td>$\downarrow R_1^-$ $\leftrightarrow R_0^+$ $\downarrow$</td>
</tr>
<tr>
<td>$L_1 : R_0^+ \rightarrow L_0$</td>
<td>$\downarrow$ $\downarrow$</td>
</tr>
<tr>
<td>$L_2 : R_2^- \rightarrow L_3, L_4$</td>
<td>$\downarrow R_2^- \leftrightarrow R_0^+$ $\downarrow$</td>
</tr>
<tr>
<td>$L_3 : R_0^+ \rightarrow L_2$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$L_4 : HALT$</td>
<td>$\downarrow$</td>
</tr>
</tbody>
</table>

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

\[ P \downarrow \triangleq \langle \left\lceil B_0 \right\rceil, \ldots, \left\lceil B_4 \right\rceil \rangle \] where

\[ \left\lceil B_0 \right\rceil = \left\lceil R_1^- \rightarrow L_1, L_2^\downarrow = \langle (2 \times 1) + 1, \langle 1, 2 \rangle \rangle \right\rceil = \langle 3, 9 \rangle = 8 \times (18 + 1) = 152 \]

\[ \left\lceil B_1 \right\rceil = \left\lceil R_0^+ \rightarrow L_0^\downarrow = \langle 2 \times 0, 0 \rangle \right\rceil = 1 \]

\[ \left\lceil B_2 \right\rceil = \left\lceil R_2^- \rightarrow L_3, L_4^\downarrow = \langle (2 \times 2) + 1, \langle 3, 4 \rangle \rangle \right\rceil = \langle 5, (8 \times 9) - 1 \rangle = \langle 5, 71 \rangle \]

\[ = 2^5 \times ((2 \times 71) + 1) = 32 \times 143 = 4576 \]

\[ \left\lceil B_3 \right\rceil = \left\lceil R_0^+ \rightarrow L_2^\downarrow = \langle 2 \times 0, 2 \rangle \right\rceil = 5 \]

\[ \left\lceil B_4 \right\rceil = \left\lceil \text{HALT} \right\rceil = 0 \]
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 $x = \langle y, z \rangle$ in

if $y = 2i$ is even, then $body(x)$ is $R_i^+ \rightarrow L_z$,
else $y = 2i + 1$ is odd, let $z = \langle j, k \rangle$ in

$body(x)$ is $R_i^- \rightarrow L_j, L_k$

So any $e \in \mathbb{N}$ decodes to a unique program $prog(e)$, called the

register machine program with index $e$:

$$
prog(e) \triangleq \begin{array}{l}
L_0 : body(x_0) \\
\vdots \\
L_n : body(x_n)
\end{array}
$$

where $e = \lceil [x_0, \ldots, x_n] \rceil$
Example of $\text{prog}(e)$

- $786432 = 2^{19} + 2^{18} = 0b110\ldots0 = \langle [18, 0] \rangle$
- $18 = 0b10010 = \langle 1, 4 \rangle = \langle 1, \langle 0, 2 \rangle \rangle = \langle R_0^-, L_0, L_2^- \rangle$
- $0 = \langle \text{HALT} \rangle$

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