Models of Computation II

Herbert Wiklicky

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

Hybrid Lectures on Fridays in room 308 and 311 and on Teams between 4pm and 6pm;

Tutorials on Mondays 11am to 12 noon virtually on Teams

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

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

Thanks 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 $\text{label : body}$ 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>$R^+ \to L'$</td>
<td>add 1 to contents of register $R$ and jump to instruction labelled $L'$</td>
</tr>
<tr>
<td>$R^- \to L', L''$</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

- \( L_0 : R_1^- \rightarrow L_1, L_2 \)
- \( L_1 : R_0^+ \rightarrow L_0 \)
- \( L_2 : R_2^- \rightarrow L_3, L_4 \)
- \( L_3 : R_0^+ \rightarrow L_2 \)
- \( L_4 : \text{HALT} \)

#### 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 \( r_0, \ldots, r_n \).
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

\[
\begin{align*}
L_0 & : R_1^+ \rightarrow L_0 \\
L_1 & : \text{HALT}
\end{align*}
\]

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^+ \quad \rightarrow \quad [L])</td>
</tr>
<tr>
<td>(R^- \rightarrow L, L')</td>
<td>(R^- \quad \rightarrow \quad [L])</td>
</tr>
<tr>
<td>(HALT)</td>
<td>(HALT)</td>
</tr>
<tr>
<td>(L_0)</td>
<td>(START \quad \rightarrow \quad [L_0])</td>
</tr>
</tbody>
</table>
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 $\neg \exists y \in Y (f(x) = y)$

- $X \rightarrow Y = \text{set of all partial functions from } X \text{ to } Y$
- $X \rightarrow^{\text{total}} Y = \text{set of all (total) functions from } X \text{ to } Y$

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

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

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

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

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

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

Sketch Proof of Result

It is enough to observe that

\[
0b\langle x, y \rangle = \begin{array}{c}
0b \langle x, y \rangle \\
0b \langle x, y \rangle
\end{array}
\begin{array}{c}
y \text{ number of } 0s
\end{array}
\begin{array}{c}
10 \cdots 0
\end{array}
\]

\[
0b\langle x, y \rangle = \begin{array}{c}
0b \langle x, y \rangle \\
0b \langle x, y \rangle
\end{array}
\begin{array}{c}
1 \text{ number of } 1s
\end{array}
\begin{array}{c}
0 1 \cdots 1
\end{array}
\]

where \( 0b x \triangleq x \) in binary. \( \triangleq \) means ‘is defined to be’.
Numerical Coding of Lists

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

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

**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 $\langle \ell \rangle \in \mathbb{N}$ by induction on the length of the list

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

Thus, $\langle [x_1, x_2, \ldots, x_n] \rangle = \langle x_1, \langle x_2, \ldots \rangle \langle x_n, 0 \rangle \rangle$
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 $\lceil \ell \rceil \in \mathbb{N}$ by induction on the length of the list $\ell$:

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

Examples

$\lceil [3] \rceil = \lceil 3 :: [] \rceil = \langle \langle 3, 0 \rangle \rangle = 2^{3}(2 \cdot 0 + 1) = 8$

$\lceil [1, 3] \rceil = \langle 1, \lceil [3] \rceil \rangle = \langle 1, 8 \rangle = 34$

$\lceil [2, 1, 3] \rceil = \langle 2, \lceil [1, 3] \rceil \rangle = \langle 2, 34 \rangle = 276$
Let $\mathit{List}\ N$ be the set of all finite lists of natural numbers.

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

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

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

- $\llbracket \[] \rrbracket \triangleq 0$
- $\llbracket x :: \ell \rrbracket \triangleq \langle \langle x, \llbracket \ell \rrbracket \rangle = 2^x(2 \cdot \llbracket \ell \rrbracket + 1)$

**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

$$0b\llbracket x_1, x_2, \ldots, x_n \rrbracket = \begin{array}{c}
1 \overbrace{0 \cdots 0}^{x_n \text{ s}} \\
1 \overbrace{0 \cdots 0}^{x_{n-1} \text{ s}} \\
1 \overbrace{0 \cdots 0}^{x_1 \text{ s}}
\end{array}$$
**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 
  \[ \text{label} : \text{body} \] 
  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><strong>HALT</strong></td>
<td>stop executing instructions</td>
</tr>
</tbody>
</table>
If $P$ is the RM program

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

then its numerical code is

\[
\lbrack \lbrack \lbrack \lbrack \text{body}_0 \rbrack, \ldots, \lbrack \lbrack \text{body}_n \rbrack \rbrack \rbrack
\]

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

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

**Registers**

$R_0 \ R_1 \ R_2$

**Program**

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

**Graphical Representation**

```
START
R_1^- \rightarrow R_1^+ \rightarrow R_0^+
R_2^- \rightarrow R_2^+ \rightarrow R_0^+
HALT
```

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

\[ P^- \triangleq \langle \langle B_0^-, \ldots, B_4^- \rangle \rangle \]

where

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

\[ B_1^- = \langle R_0^+ \rightarrow L_0^- = 2 \times 0, 0 \rangle = 1 \]

\[ B_2^- = \langle R_2^- \rightarrow L_3, L_4^- = (2 \times 2) + 1, \langle 3, 4 \rangle \rangle \]
\[ = \langle 5, (8 \times 9) - 1 \rangle = \langle 5, 71 \rangle \]
\[ = 2^5 \times ((2 \times 71) + 1) = 32 \times 143 = 4576 \]

\[ B_3^- = \langle R_0^+ \rightarrow L_2^- = 2 \times 0, 2 \rangle = 5 \]

\[ B_4^- = \langle HALT^- \rangle = 0 \]
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** $e$:

$$
prog(e) \triangleq \begin{array}{c}
L_0 : body(x_0) \\
\vdots \\
L_n : body(x_n)
\end{array}
\text{where } e = \lceil [x_0, \ldots, x_n] \rceil$$
Example of \( \text{prog}(e) \)

- \( 786432 = 2^{19} + 2^{18} = 0b110\ldots0 = \lceil [18, 0] \rceil \)
- \( 18 = 0b10010 = \langle 1, 4 \rangle = \langle 1, \langle 0, 2 \rangle \rangle = \lceil R_0^- \rightarrow L_0, L_2^- \rceil \)
- \( 0 = \lceil HALT \rceil \)

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