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 $(i + 1)^{th}$ 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>HALT</td>
<td>stop executing instructions</td>
</tr>
</tbody>
</table>

Recall the Codings of Pairs, Lists and programs

Pairs For $x, y \in \mathbb{N}$, define

$$\begin{cases} 
\langle x, y \rangle \triangleq 2^x(2y + 1) \\
\langle x \rangle \triangleq 2^x(2y + 1) - 1 
\end{cases}$$

Lists For $\ell \in \text{List} \mathbb{N}$, define $\gamma \ell \in \mathbb{N}$ by

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

Programs $\gamma P \triangleq \gamma [\gamma body_0 \gamma, \ldots, \gamma body_n \gamma]$ where $\gamma body \gamma$ is:

$$\begin{cases} 
\gamma R^+_i \to L_j \triangleq \langle 2i, j \rangle \\
\gamma R^-_i \to L_j, L_k \triangleq \langle 2i + 1, \langle j, k \rangle \rangle \\
\gamma \text{HALT} \gamma \triangleq 0 
\end{cases}$$
Recall Addition \( f(x, y) \triangleq x + y \) is Computable

**Registers**
\[ R_0 \ R_1 \ R_2 \]

**Program**
\[
\begin{align*}
L_0 &: R_1^- \rightarrow L_1, L_2 \\
L_1 &: R_0^+ \rightarrow L_0 \\
L_2 &: R_2^- \rightarrow L_3, L_4 \\
L_3 &: R_0^+ \rightarrow L_2 \\
L_4 &: HALT
\end{align*}
\]

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

\[ \gamma P^- \triangleq \gamma[\gamma B_0^-, \ldots, \gamma B_4^-]^- \] where
\[ \gamma B_0^- = \gamma R_1^- \rightarrow L_1, L_2^- = \langle (2 \times 1) + 1, (1, 2) \rangle = \langle 3, 9 \rangle = 8 \times (18 + 1) = 152 \]
\[ \gamma B_1^- = \gamma R_0^+ \rightarrow L_0^- = \langle 2 \times 0, 0 \rangle = 1 \]
\[ \gamma B_2^- = \gamma R_2^- \rightarrow L_3, L_4^- = \langle (2 \times 2) + 1, (3, 4) \rangle = \langle 5, (8 \times 9) - 1 \rangle = \langle 5, 71 \rangle = 2^5 \times ((2 \times 71) + 1) = 32 \times 143 = 4576 \]
\[ \gamma B_3^- = \gamma R_0^+ \rightarrow L_2^- = \langle 2 \times 0, 2 \rangle = 5 \]
\[ \gamma B_4^- = \gamma HALT^- = 0. \]
Decoding Numbers as Bodies and Programs

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

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

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

\[
\text{prog}(e) \triangleq \begin{cases}
L_0 : \text{body}(x_0) \\
\vdots \\
L_n : \text{body}(x_n)
\end{cases}
\]

where \( e = \lceil x_0, \ldots, x_n \rceil \)

Example of \( \text{prog}(e) \)

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

So \( \text{prog}(786432) = \begin{cases}
L_0 : R_0^- \rightarrow L_0, L_2 \\
L_1 : \text{HALT}
\end{cases} \)
Notice that, when $e = 0$, we have $0 = \left\lfloor [] \right\rfloor$ 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). Also, notice in slide 6 the jump to a label with no body (an erroneous halt). Again, choose some numbers and see what the register-machine programs they correspond to.

**Gadgets**

To construct register machines which perform complex operations, we introduce *gadgets* which are small components used to perform small specific operations. A gadget is defined informally as a partial register-machine graph that has a designated initial label and one or more designated exit labels (which contain no instructions). The gadget operates on registers specified in the gadget's name, and are used for input and output — we call these the input/output registers. The gadget may use other registers for temporary storage — we call these *scratch registers*. The gadget assumes the scratch registers are initially set to 0, and *must* ensure that they are set back to 0 when the gadget exits. Ensuring that the scratch registers are reset to 0 is important so that the gadget may be safely used within loops.

A gadget is a partial register-machine graph.

It has one entry wire, and one or more exit wires.

The gadget operates on input and output registers specified in the gadget's name.

The gadget may use other registers, called scratch registers, for temporary storage.

The gadget assumes the scratch registers are initially set to 0, and *must* ensure that they are set back to 0 when the gadget exits.
The gadget “zero $R_0$” sets register $R_0$ to be zero, whatever its initial value:
Gadget: “add $R_1$ to $R_2$”

The gadget “add $R_1$ to $R_2$” adds the initial value of $R_1$ to register $R_2$, storing the result in $R_2$ but restoring $R_1$ to its initial value.

We can compose gadgets, constructing bigger gadgets and eventually complete register machines. To construct such bigger gadgets, we rename the registers used by each gadget: all of its scratch registers are renamed to things that do not occur in the rest of the machine, and its input/output registers are renamed to whichever registers the program requires. We then ‘wire up’ the gadgets to make bigger gadgets, by joining (possibly many) exit wires to the unique entry wire. For example, consider the gadget “copy $R_1$ to $R_2$” defined on slide 10. We will use this gadget to construct the universal register machine. The gadget “copy $R_1$ to $R_2$” copies the value of register $R_1$ into register $R_2$, leaving $R_1$ with its initial value. It does this by joining together the “zero” and “add” gadgets: it first sets register $R_2$ to zero, then adds the value of $R_1$ to $R_2$ and leaves the value of $R_1$ the same, using some unnamed scratch register inside the “add” gadget.
The gadget “copy $R_1$ to $R_2$” copies the value of register $R_1$ into register $R_2$, leaving $R_1$ with its initial value:

Now to construct the instance “copy $R_1$ to $R_3$” of the gadget, we rename the scratch register $R_3$ to be something completely fresh, say $R_7$, and rename $R_1$ and $R_2$ to $R_1$ and $R_3$ respectively. Joining these gadgets together, we obtain the larger gadget “copy $R_1$ to $R_2$ and $R_3$.”
Slide 11

Gadget: “copy $R_1$ to $R_2$ and $R_3$”

entry

- copy $R_1$ to $R_2$
- copy $R_1$ to $R_3$

exit

Slide 12

Gadget: “copy $R_1$ to $R_2$ and $R_3$”

entry

- zero $R_2$
- add $R_1$ to $R_2$
- zero $R_3$
- add $R_1$ to $R_3$

exit
Recall the register machine for multiplication given earlier in the course, which multiplies $R_1$ by $R_2$ and stores the result in $R_0$, possibly overwriting the initial values of $R_1$ and $R_2$. We can construct this register machine using the “zero” and “add” gadgets.
Gadgets: “multiply $R_1$ by $R_2$ to $R_0$”

We can implement “multiply $R_1$ by $R_2$ to $R_0$” by repeated addition:

As well as the “copy” gadget, we require two more gadgets to define the universal register machine: “push $X$ to $L$” and “pop $L$ to $X$”. Given input values $X = x$ and $L = l$, the gadget “push $X$ to $L$” returns the value $X = 0$ and $L = 2^x(2^l + 1)$. Given input value $L = \langle x, l \rangle$ and $X = y$, the gadget “pop $L$ to $X$” returns $X = x$ and $L = l$. Given input $L = 0$, the gadget returns $X = 0$ and $L = 0$. 
Gadget: “push $X$ to $L$”

The gadget “push $X$ to $L$”:

Given input values $X = x$, $L = \ell$, and $Z = 0$, it returns the output values $X = 0$, $L = \langle x, \ell \rangle = 2^x(2\ell + 1)$ and $Z = 0$:

$L = 2^{x-X}(2\ell + 1)$,

$Z + 2L = 2^{x-X}(2\ell + 1)$

$Z = 0$

$X = x$

$L = \ell$

$Z = 0$

$Z + L = 2^{x-X}(2\ell + 1)$

$L = 2^x(2\ell + 1)$

$Z = 0$
The gadget “pop $L$ to $X$":

If $L = 0$ then return $X = 0$ and go to “empty”. If $L = \langle \langle x, \ell \rangle \rangle$ then return $X = x$ and $L = \ell$, and go to “done”.

Slide 18

$n = 2^X (L + Z)$

$L = n, \quad X = y, \quad Z = 0$

$n = 2^{X+1} L, Z = 0$

$n = 2^X (2L + Z)$

$n = 2^X (2L + 1), Z = 0$
The Universal Register Machine

The universal register machine 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:

- $R_0$ result of the simulated RM computation (if any).
- $R_1 \equiv P$ code of the RM to be simulated
- $R_2 \equiv A$ code of current register contents of simulated RM
- $R_3 \equiv PC$ program counter—number of the current instruction (counting from 0)
- $R_4 \equiv N$ code of the current instruction body
- $R_5 \equiv C$ type of the current instruction body
- $R_6 \equiv R$ current value of the register to be incremented or decremented by current instruction (if not HALT)
- $R_7 \equiv S$ and $R_8 \equiv T$ are auxiliary registers.
- $R_9$... other scratch registers.

**Overall structure of the URM**

1. copy $PC$th item of list in $P$ to $N$ (halting if $PC >$ length of list); goto 2
2. if $N = 0$ then halt, else decode $N$ as $\langle y, z \rangle$; $C ::= y; N ::= z$; goto 3
3. copy $i$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
The gadget “copy $R_1$ to $R_2$” copies the value of register $R_1$ into register $R_2$, leaving $R_1$ with its initial value:

Given input values $X = x$, $L = \ell$ and $Z = 0$, it returns the output values $X = 0$, $L = \langle x, \ell \rangle = 2^x(2\ell + 1)$ and $Z = 0$.
Gadget: “pop $L$ to $X$”

The gadget “pop $L$ to $X$”:

Slide 24

If $L = 0$ then return $X = 0$ and go to “empty”. If $L = \langle x, \ell \rangle$ then return $X = x$ and $L = \ell$, and go to “done”.

The Universal Register Machine

Slide 25
The Universal Register Machine

Slide 26

Slide 27
The Universal Register Machine

START

- push \( R_0 \) to \( A \)
- copy \( P \) to \( T \)
- pop \( T \) to \( N \)
- pop \( A \) to \( R_0 \)
- pop \( N \) to \( C \)
- pop \( A \) to \( R \)
- push \( R \) to \( S \)

- empty
- done

- \( R^- \)
- \( C^- \)
- \( N^+ \)
- \( C^- \)
- \( R^- \)

- \( P^- \)
- \( C^- \)
- \( N^+ \)
- \( C^- \)
- \( R^- \)

Slide 28

The Universal Register Machine

START

- push \( R_0 \) to \( A \)
- copy \( P \) to \( T \)
- pop \( T \) to \( N \)
- pop \( A \) to \( R_0 \)
- pop \( N \) to \( C \)
- pop \( A \) to \( R \)
- push \( R \) to \( S \)

- empty
- done

- \( R^- \)
- \( C^- \)
- \( N^+ \)
- \( C^- \)
- \( R^- \)

- \( P^- \)
- \( C^- \)
- \( N^+ \)
- \( C^- \)
- \( R^- \)

Slide 29