Math 260A — Mathematical Logic — Scribe Notes
UCSD — Spring Quarter 2012
Instructor: Sam Buss

Notes by: Andy Parrish
Friday, April 20, 2012

1 Sequence Coding

Primitive recursive functions can only take a fixed number of inputs. In order to simulate Turing Machines, we need to be able to be able to manage sequences of arbitrary length. To do this, we should encode a sequence as a single number.

Definition Given a sequence $a_0, \ldots, a_{\ell-1}$ of ℓ natural numbers, define

$$\langle a_0, \ldots, a_{\ell-1} \rangle = p_0^{a_0} p_1^{a_1} \cdots p_{\ell-1}^{a_{\ell-1}} p_\ell,$$

where p_i denotes the i^{th} prime, with $p_0 = 2$.

Examples

$$\begin{aligned} \langle 3, 1 \rangle &= 2^3 \cdot 3^1 \cdot 5 = 120 \\ \langle 3, 1, 0 \rangle &= 2^3 \cdot 3^1 \cdot 5^0 \cdot 7 = 168. \end{aligned}$$

We will also need helper functions when working with sequences:

$$\begin{array}{rcl} \mathrm{Len}(w) & = & \max\{i \le w+2 \ : \ p_i | w\} \\ \mathrm{Seq}(w) & \iff & w \ge 2 \text{ and } p_{\mathrm{Len}(w)}^2 \nmid w \iff w \text{ denotes a valid sequence.} \end{array}$$

Using results from the previous lecture, both of these are primitive recursive functions.

Additionally, we will want to extract individual entries from the sequence. To do this, we take the Gödel β function:

$$\beta(i, w) = \begin{cases} 0 & \text{if } \neg\mathrm{Seq}(w) \text{ or } i \ge \mathrm{Len}(w) \\ \max\{j < w \ : \ p_i^j | w\} & \text{otherwise,} \end{cases}$$

which is again primitive recursive.

We can decode sequences using these functions. How do we encode them?

Well, the function $a \mapsto \langle a \rangle$ is primitive recursive; the function value is $2^a \cdot 3$.

This gets a sequence started. To add to it, we define the function

$$w^\frown a = (w/p_{\mathrm{Len}(a)}) \cdot p^a_{\mathrm{Len}(w)} \cdot p_{\mathrm{Len}(w)+1}.$$

This gives the result of pushing entry a onto the end of the sequence encoded by w. This defines our function $\mathrm{PushRight}(w,a) = w^\frown a$.

It is possible, if trickier, to define the related function $\mathrm{PushLeft}(w,a)$, which pushes new value a onto the beginning of the sequence encoded by w.

$$\mathrm{PushLeft}(w,a) = \min \left\{ v \le w^2 \cdot 2^a \;\middle|\; \begin{array}{l} \mathrm{Seq}(v) \text{ and } \mathrm{Len}(v) = \mathrm{Len}(w)+1, \\ \beta(0,v) = a, \text{ and} \\ \forall i \le \mathrm{Len}(w), \beta(i+1,v) = \beta(i,w). \end{array} \right\}.$$

Additional care is needed if $\neg\mathrm{Seq}(w)$, which we ignore.

We similarly define functions $\mathrm{PopLeft}(w)$ and $\mathrm{PopRight}(w)$ which respectively remove the leftmost and rightmost elements of w and return the resulting sequence. The exact details of the definition are not interesting, but we show the format.

$$\mathrm{PopLeft}(w) = \left\{ \begin{array}{ll} \langle\,\rangle & \text{if } \mathrm{Len}(w) \le 1 \\ \min\{v \le w \mid \ldots\} & \text{otherwise.} \end{array} \right\}.$$

PopRight is similar.

2 Sequence coding and Turing Machines

Consider a Turing machine M whose tape reads:

$$\ldots a_0 a_1 \ldots a_{\ell-1} a_\ell \ldots$$

and is currently in state $q \in Q$, with tape head on a_k.

We identify the states with integers, and specifically designate some states, e.g. $q_H \leftrightarrow 0, q_Y \leftrightarrow 1, q_N \leftrightarrow 2$.

Identify the alphabet $\{0, 1, \}$ with $\{0, 1, 2\}$.

We encode the configuration of M by the code

$$w = \langle q, \langle a_k, a_{k+1}, \ldots, a_\ell \rangle, \langle a_{k-1}, a_{k-2}, \ldots, a_0 \rangle\rangle.$$

Define a predicate Halt_M by

$$\mathrm{Halt}_M \iff (\beta(0,w) = q_H) \iff w \text{ codes a halting configuration.}$$

Also define a function Init_M by

$$\mathrm{Init}_M(x) = \text{Gödel number of the initial configuration of } M \text{ on input } x.$$

There are two conventions here, and each is reasonable — either $\mathrm{Init}_M(x)$ is given by $\langle q_0, \langle 0, \ldots, 0\rangle, \langle\rangle\rangle$, (with x many 0's), or by $\langle q_0, \langle 1, 0, 1, 1, 0\rangle, \langle\rangle\rangle$, (using the binary representation of x.

Our next goal is to define a function Next_M so that $\mathrm{Next}_M(w)$ is the configuration reached by M in one step from the configuration given by w.

To construct this, it is helpful to have several functions:

- $\mathrm{State}(w) = \beta(0, w)$ gives the current state.
- $\mathrm{FirstSym}(w) = \begin{cases} \beta(0, w) & \text{if } \mathrm{Len}(w) \geq 1 \\ & \text{if } \mathrm{Len}(w) = 0 \end{cases}$
  gives the first entry of a list
- $\mathrm{CurSym}(w) = \mathrm{FirstSym}(\beta(1, w))$ gives the symbol at the location of the tape head.

The definition of Next_M has a large but finite number of cases, depending on $\mathrm{State}(w)$ and $\mathrm{CurSym}(w)$.

For example, suppose

$$\delta(q_0, \sigma_0) = (\sigma_1, N, q_2)$$

$$\delta(q_2, \sigma_1) = (\sigma_2, L, q_3)$$

is a partial definition of the transition function δ.

Then $\mathrm{Next}_M(w)$ should be

$$\langle q_2, \mathrm{PushLeft}(\sigma_1, \mathrm{PopLeft}(\beta(1, w))), \beta(2, w)\rangle$$

when $\mathrm{State}_M = q_0$ and $\mathrm{CurSym}(w) = \sigma_0$, and

$$\langle q_3, \mathrm{PushLeft}(\mathrm{FirstSym}(\beta(2, w)), \mathrm{PushLeft}(\sigma_2, \mathrm{PopLeft}(\beta(1, w)))), \mathrm{PopLeft}(\beta(2, w))\rangle$$

when $\mathrm{State}_M = q_2$ and $\mathrm{CurSym}(w) = \sigma_1$.

The full definition of Next_M will take into account for every possible state-symbol pair, and also handle cases with invalid input.

With these functions defined, we may finally define the predicate Comp_M, which can recognize valid computations.

Formally, this is given by

$$
\begin{aligned}
\mathrm{Comp}_M(x,v) \quad &\iff \quad \begin{cases} v = \langle w_0, \ldots, w_{\ell-1}\rangle \\ w_0 = \mathrm{Init}_M(x) \\ w_{\ell-1} = \text{ halting configuration} \\ w_{i+1} = \mathrm{Next}(w_i) \forall i < \ell - 1 \end{cases} \\
&\iff \quad \begin{array}{l} \mathrm{Seq}(v) \wedge \mathrm{Len}(v) \geq 1 \wedge \beta(0,v) = \mathrm{Init}_M(x) \wedge \\ \quad (\forall i < \mathrm{Len}(w) - 1)(\beta(i+1,v) = \mathrm{Next}_M(\beta(i,v)) \wedge \\ \quad \mathrm{Halt}_M(\beta(\mathrm{Len}(v)-1,v))) \end{array} \\
&\iff \quad v \text{ codes a complete valid computation with input } x.
\end{aligned}
$$