Arithmetization of Turing Machines

Some Basic Encodings.

$\text{pair} : \mathbb{N}^2 \xrightarrow[\text{onto}]{1\text{-}1} \mathbb{N}$ $\quad \text{pair}(x,y) = 2^x(2y+1) - 1$

Notice that pair is a bijection. Firstly notice that $\text{pair}(0,0) = 0$. Hence for any $z \in \mathbb{N}$,

if z is even then it represents a pair of the form $(0,y)$

Otherwise it is a pair with a non-zero first component.

We may extract the components of a pair using the following functions

$\text{fst} : \mathbb{N} \to \mathbb{N}$ $\quad \text{fst}(z) = \mu x < z+1 \left[2^x \mid (z+1) \wedge 2^{x+1} \nmid (z+1) \right]$

$\text{snd} : \mathbb{N} \to \mathbb{N}$ $\quad \text{snd}(z) = \left((z+1) / 2^{\text{fst}(z)} - 1 \right) / 2$

Clearly then we have the following defining equations relating pair, fst, snd

$$\text{pair}(\text{fst}(z), \text{snd}(z)) = z$$

$$\text{fst}(\text{pair}(x,y)) = x$$

$$\text{snd}(\text{pair}(x,y)) = y \,.$$

$\text{tuple}^k : \mathbb{N}^k \longrightarrow \mathbb{N}$. for any $k > 1$.

$\text{tuple}(x_1, \ldots x_k) = \text{pair}(x_1, \text{tuple}^{k-1}(x_2, \ldots, x_k))$

where $\text{tuple}^2 = \text{pair}$

For any $\text{tuple}^k(x_1, \ldots, x_k) = z$ we define for $k \geq 2$, $\text{proj}^k_i(z) = \text{fst}(\text{snd}^{i-1}(z))$ for $1 \leq i \leq k$ where snd^j stands for the j-fold composition of snd and snd^0 is the identity function. Clearly then for any $z \in \mathbb{N}$ we have

$$z = \text{tuple}(\text{proj}^k_1(z), \ldots, \text{proj}^k_k(z))$$

and for any $x_1, \ldots, x_k \in \mathbb{N}$, $1 \leq i \leq k$

$$x_i = \text{proj}^k_i(\text{tuple}^k(x_1, \ldots, x_k))$$

The Coding of lists of numbers

Intuitively for any (nonempty) list of naturals $[a_0, \ldots, a_{k-1}]$ we encode it as the binary number

$\underbrace{1 0 \ldots 0}_{a_{k-1}} \underbrace{1 0 \ldots 0}_{a_{k-2}} \ldots \underbrace{1 0 \ldots 0}_{a_0}$ which consists of the catenation

of the bit-string representations of the numbers $2^{a_{k-1}}, \ldots, 2^{a_0}$ respectively in that order. The number 0 represents the empty list of numbers.

$$\text{List} : \mathbb{N}^* \xrightarrow[\text{onto}]{1\text{-}1} \mathbb{N}$$

such that

$$\text{List}\,[\,] = 0$$

$$\text{List}\,[a_0, \ldots, a_{k-1}] = 2^{a_0}\left(1 + 2^{a_1+1}\left(1 + 2^{a_2+1}\left(\ldots\left(1 + 2^{a_{k-1}+1}\right)\ldots\right)\right)\right)$$

$$= \left(\sum_{j=0}^{k-1} 2^{\left(\sum_{i\le j} a_i + j\right)}\right)$$

$$= 2^{a_0} + 2^{a_0+a_1+1} + \ldots + 2^{a_0+a_1+\cdots+a_{k-1}+k-1}$$

We then proceed to construct the list operations hd, tl and cons as follows:

1) hd and tl are undefined for 0 which represents the empty list.

2) For $z > 0$, $z \in \mathbb{N}$, the head of the list is obtained as the exponent of the greatest power of 2 which divides z.

$$\boxed{\text{hd}(z) \triangleq \nu y < z\,[2^y \mid z]}$$

3) From the definition

$$2^{a_0}(1+2^{a_1+1}(1+\ldots(1+2^{a_{k-1}+1})\ldots))$$

it follows that since $a_0 = hd(z)$

$$\boxed{tl(z) = \left(\frac{z}{2^{hd(z)}} - 1\right)/2} = 2^{a_1}(1+\ldots(1+2^{a_{k-1}+1})\ldots)$$

4) Again from the definition

$$\text{List}(a_0,\ldots,a_{k-1}) = 2^{a_0}(1+2^{a_1+1}(1+\ldots(1+2^{a_{k-1}+1})\ldots))$$

it follows that $\text{List}(a :: [a_0,\ldots,a_{k-1}])$

$$= \text{List}(a, a_0,\ldots,a_{k-1})$$

$$= 2^{a}(1+2^{a_0+1}(1+2^{a_1+1}(1+\ldots(1+2^{a_{k-1}+1})\ldots)))$$

$$= 2^{a}(1+2z)$$

Hence for any $z \geq 0$

$$\boxed{\text{cons}(a,z) = 2^{a}(1+2z)}$$

It then follows quite trivially that for any $z \in \mathbb{N}$,

$$\boxed{z = 0 \quad \vee \quad z = \text{cons}(hd(z), tl(z))}$$

Claims:

1. For any $p \in \mathbb{N}$ such that $p = \text{pair}(x, y)$

$$p = 0 \quad \text{iff} \quad \text{fst}(p) = x = 0 = y = \text{snd}(p)$$

2. For any $m \in \mathbb{N}$ such that $m = \text{tuple}^k(x_1, \ldots, x_k)$

$$m = 0 \quad \text{iff} \quad \text{for all } i: 1 \leq i \leq k: \text{proj}_i^k(m) = x_i = 0$$

3. For any $\ell \in \mathbb{N}$ such that $\ell = \text{list}(L)$ where $L \in \mathbb{N}^*$

$$\ell = 0 \quad \text{iff} \quad L = []$$

For any $m, \ell \in \mathbb{N}$ where ℓ encodes a list of numbers a useful predicate to have is the membership predicate which we may define as follows.

$$\text{mem}(m, \ell) \triangleq \ell = 0 \,?\, 0 : (m = \text{hd}(\ell) \,?\, 1 : \text{mem}(m, \text{tL}(\ell)))$$

Although we have used the if-then-else form to define it, it is really a primitive recursive predicate. We leave it as an exercise for the intelligent reader.

Exercise: Define mem as a primitive recursive predicate.

Arithmetizing Turing Machines

Let $T = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, \{q_F\} \rangle$ be a **deterministic Turing machine** satisfying the following conditions:

1) $\Sigma \subseteq \Gamma - \{\square\}$, $\square \in \Gamma$
2) There is no transition out of the final state q_F
3) $\delta : (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$ is a finite partial function such that
   - 3.1) there is no transition out of q_F on any $a \in \Gamma$
   - 3.2) for every $q \in Q - \{q_F\}$ and each $a \in \Gamma$ there is a triple (q', a', D) such that
     $$\delta(q, a) = (q', a', D)$$
     where $q' \in Q$, $a' \in \Gamma$ and $D \in \{L, R\}$
4) Each instantaneous description ID is a 3-tuple of the form $(\alpha, (q, c), \beta)$ where $\alpha, \beta \in (\Gamma - \{\square\})^*$, $c \in \Gamma$ and $q \in Q$.
5) $q_0 \neq q_F$ and the machine does not have an empty δ.

A consequence of 3.1) and 3.2) is that T cannot be stuck in any state except q_F. It is always possible to ensure that the Turing machine either goes on forever or terminates its execution in the final state.

Consider the following coding mechanism for each of the important elements of a Turing machine.

Let $\Gamma = \{\square, a_1, \ldots, a_k\}$ be the tape symbols assigned respectively the numbers $0 \ldots k$. and let $Q = \{q_0, \ldots, q_m = q_F\}$ be assigned numbers $k+1, \ldots, m+k+1$ respectively. Implicitly we have introduced total orderings

$$\square < a_1 < \cdots < a_k \qquad \text{——— (1)}$$

and

$$q_0 < \cdots < q_F \qquad \text{——— (2)}$$

Given any non-negative integer n we may define simple primitive-recursive predicates

$$\text{istapesym}(n) \triangleq 0 \leq n \leq k$$

and

$$\text{isstate}(n) \triangleq k < n \leq m+k+1$$

which determine whether a given integer denotes a tape symbol or a state.

Since the Turing machine is deterministic

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$$

and δ is a function on a finite domain we consider the "graph" of δ as a sequence of elements $((q, a), (q', a', D))$

ordered lexicographically on the domain of function δ.

i.e. $((q_1, a_1), (q_1', a_1', D_1)) < ((q_2, a_2), (q_2', a_2', D_2))$

iff $q_1 < q_2$ or $(q_1 = q_2$ and $a_1 < a_2)$

For any component of T including T itself we use $\ulcorner\ \urcorner$ to denote the code assigned to the component by our arithmetization.

Further if $\ulcorner q \urcorner$ and $\ulcorner a \urcorner$ denote respectively the codes assigned to the states (by assignment (2)) and tape symbols (by assignment (1)) respectively then

$$\ulcorner((q,a),(q',a',D))\urcorner \triangleq \text{pair}(\text{pair}(\ulcorner q \urcorner, \ulcorner a \urcorner), \text{tuple}^{(3)}(\ulcorner q' \urcorner, \ulcorner a' \urcorner, \ulcorner D \urcorner))$$

where $\ulcorner L \urcorner \triangleq 0$ and $\ulcorner R \urcorner \triangleq 1$

It is obvious from the definitions of the transitions that for each $((q,a),(q',a',D)) \in \delta$, if $\ulcorner((q,a),(q',a',D'))\urcorner = z$

$$\text{arg}(z) \triangleq \text{fst}(z) = x \qquad \text{val}(z) = \text{snd}(z) = y$$

$$\text{src}(z) \triangleq \text{fst}(x) = \ulcorner q \urcorner \qquad \text{tgt}(z) \triangleq \text{proj}_1^{(3)}(y) = \ulcorner q' \urcorner$$

$$\text{rd}(z) \triangleq \text{snd}(x) = \ulcorner a \urcorner \qquad \text{wr}(z) \triangleq \text{proj}_2^{(3)}(y) = \ulcorner a' \urcorner$$

$$\text{dir}(z) \triangleq \text{proj}_3^{(3)}(z) = \ulcorner D \urcorner$$

We may order the transitions as follows. We notice that $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, D\}$ is a function and hence δ may be expressed as a list of arg-val pairs, lexicographically ordered by the total ordering on $Q \times \Gamma$. For any $(q,a), (q',a') \in Q \times \Gamma$ we have

$$(q,a) < (q',a') \text{ iff } q < q' \text{ or } (q = q' \text{ and } a < a')$$

δ	$\square$	a_1	...	a_k
q_0				
⋮				
q_{m-1}				

We express the transition table as a list in "row-major" since the lexicographic ordering on $Q \times \Gamma$ defines precisely a row-major ordering of the table.

The transition function δ may then be encoded as

$$\ulcorner \delta \urcorner = \text{List}(\ulcorner qu_1 \urcorner, \ldots, \ulcorner qu_\ell \urcorner)$$

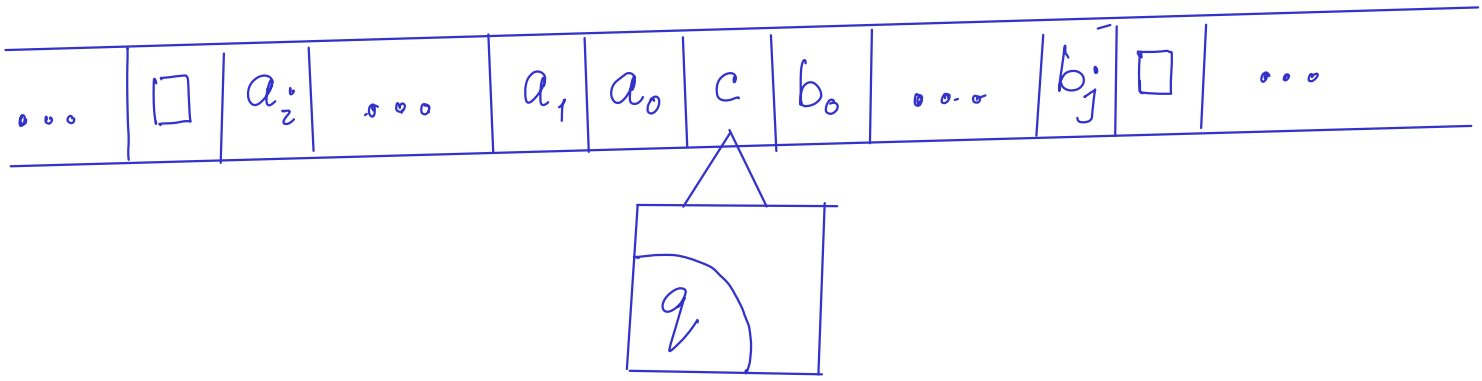
where $\ell = (k+1)m$ and each qu_i is of the form $((q,a),(q',a',D))$.

Since k and m are crucial parameters in the encoding which need to be known in advance for purposes of decoding and determining validity we complete the encoding as

$$\ulcorner T \urcorner \triangleq \text{tuple}^{(3)}(\ulcorner \delta \urcorner, k, m)$$
$$\text{delta}(t) \triangleq \text{proj}_1^{(3)}(t)$$
$$\text{kpar}(t) \triangleq \text{proj}_2^{(3)}(t)$$
$$\text{mpar}(t) \triangleq \text{proj}_3^{(3)}(t)$$

Encoding Turing machine configurations

We now proceed to the task of encoding Turing machine configurations



Since the parameters k, m are crucial to coding, decoding and determining the validity of IDs for a given Turing machine, we define

$$\text{config}(\ulcorner ID \urcorner, k, m) = \text{tuple}^{(3)}(\ulcorner ID \urcorner, k, m)$$

For both Turing machine codes as well as ID codes we have the functions kpar and mpar.

$$\text{kpar}(n) \triangleq \text{proj}_2^{(3)}(n)$$
$$\text{mpar}(n) \triangleq \text{proj}_3^{(3)}(n)$$

$$ID(n) \triangleq \text{proj}_1^{(3)}(n)$$

where the machine is in state q and the read-write head is on a symbol c, $\alpha = a_i \dots a_0$ and $\beta = b_0 b_1 \dots b_j$ are the tape contents. That is, the instantaneous description ID of the machine is given by the tuple

$$\begin{aligned} ID &= (\alpha, (q, c), \beta) \\ &= (a_i \dots a_0, (q, c), b_0 \dots b_j) \end{aligned}$$

We then code ID as follows.

$$\begin{aligned} \ulcorner ID \urcorner = \text{tuple}^{(3)}(& \text{list}(\ulcorner a_0 \urcorner, \dots, \ulcorner a_i \urcorner), \\ & \text{pair}(\ulcorner q \urcorner, \ulcorner c \urcorner), \\ & \text{list}(\ulcorner b_0 \urcorner, \dots, \ulcorner b_j \urcorner) \\ ) & \end{aligned}$$

Again notice that for any configuration $c \in \mathbb{N}$ the individual components and sub-components may be extracted as follows

$$\begin{aligned} \text{left}(c) &\triangleq \text{proj}_1^{(3)}(z) = \text{list}(\ulcorner a_0 \urcorner, \dots, \ulcorner a_i \urcorner) \\ \text{right}(c) &\triangleq \text{proj}_3^{(3)}(z) = \text{list}(\ulcorner b_0 \urcorner, \dots, \ulcorner b_j \urcorner) \\ \text{state}(c) &\triangleq \text{fst}(\text{proj}_2^{(3)}(z)) \\ \text{sym}(c) &\triangleq \text{snd}(\text{proj}_2^{(3)}(z)) \\ \text{isfinal}(c) &\triangleq \text{state}(z) = k+m+1 \end{aligned}$$

where

$z = ID(c)$

$k = kpar(c)$

$m = mpar(c)$

We may also define the following predicates primitive recursively for each non-negative integer $z \in \mathbb{N}$.

$$\begin{aligned} \text{isID}(c) \triangleq\ & \text{isstring}(\text{left}(z)) \wedge \\ & \text{isstring}(\text{right}(z)) \wedge \\ & \text{isstate}(\text{state}(z)) \wedge \\ & \text{ists}(\text{sym}(z)) \end{aligned}$$

where for any $z \in \mathbb{N}$

$$\text{isstring}(z) \triangleq \text{istapesym}(\text{hd}(z)) \wedge \text{isstring}(\text{tl}(z))$$

Similarly we may define a primitive recursive predicate

istm(z) which determines whether the transition function δ of a "purported" Turing machine does satisfy all the conditions of a deterministic Turing machine as we have outlined earlier. This predicate though complex may be defined by a huge conjunction of smaller conditions. We leave it as an exercise (in capturing details) to the intelligent reader and proceed to define the working of the Turing machine as an arithmetic function.

Remark on undefinedness. Because our tape symbols and states are finite, there are many natural numbers which may not be meaningful as codes of either machines or instantaneous descriptions. The number 0 is one such. It can not be the code of any Turing machine unless the machine has an empty transition table. But that would violate condition (5). Similarly it can not be the code of any instantaneous description because all states have non-zero codes.

Claims

1. 0 is not the code of any deterministic Turing machine.
2. 0 is not the code of any instantaneous description.

Simulating Turing machine computations

Let T be the Turing machine encoded as above. Assume T computes some function from $\Sigma^* \longrightarrow \Sigma^*$. Given an input $x \in \Sigma^*$ the initial configuration of the machine is given by the ordered pair (δ, ID_0) where

$$ID_0 = \varepsilon q_0 x$$

it yields another ordered pair (d, id') which represents the next instantaneous description (if any).

next $(z) \triangleq$ **if** isfinal (y) **then** pair $(x, 0)$
else trans (x, y)

where

$z = \ulcorner (x, y) \urcorner$

$x = \ulcorner T \urcorner$ for some Turing machine T

$y = \ulcorner C \urcorner$ for a configuration C of machine T

$ID = \ulcorner (\alpha, (q, c), \beta) \urcorner$ is the instantaneous description of configuration C

$\delta(q, c) = (q', c', D)$, δ is the transition function of T

trans $(x, y) = \ulcorner (\alpha', (q', c'), \beta') \urcorner$

For any such machine T and initial configuration C_0 we define for $z_0 = \text{pair}(\ulcorner T \urcorner, \ulcorner C_0 \urcorner)$

$$\text{numsteps}(z_0) \triangleq \mu n [\text{snd}(\text{next}^n(z_0)) = 0]$$

i.e. numsteps yields the least number of steps of computation required to bring the machine to its final state.

Note that unlike all the previous functions that we have defined, numsteps (z_0) may be undefined if the computation does not terminate.

The output of the Turing machine T at the end of the computation (if it does terminate is given) by

$$\text{output}(z_0) \triangleq \ulcorner \beta_F' \urcorner \quad \text{for} \quad z_0 = \ulcorner (x, y_0) \urcorner$$

where

$$z_F = \text{next}^{n-1}(z_0) = \ulcorner (x, y_F) \urcorner$$
$$n = \text{numsteps}(z_0)$$
$$y_F = \ulcorner (ID_F, k, m) \urcorner$$
$$ID_F = (\alpha_F, (q_F, c_F), \beta_F)$$
$$\beta_F' = \text{cons}(c_F, \beta_F)$$

Notice that n in the above definition can never be 0 since our Turing machines by our conditions must perform at least one step.

Further notice that numsteps yields a value that is one more than that required to bring the Turing machine to a final configuration so that the next step yields a 0 as the final configuration.