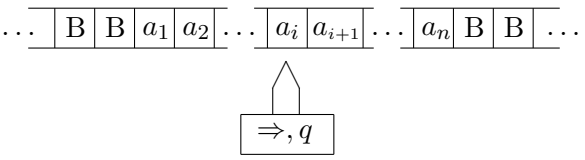
Computability and lambda-calculus

Jacques Garrigue, 2013/11/05

1 Turing Machine

Turing machines are the most elementary complete model of computation. They were invented in 1936 by the British mathematician Alan Turing in the process of giving a formal definition of computation.

A Turing machine is composed of a *tape* where *symbols* are written, a *head* reading it, and a controller *state*. The tape has infinite length, but both the number of symbols and states are finite; moreover the number of symbols written on the tape at any time is finite too.



Intuitively the operation of a Turing machine is composed of 3 steps.

1. One reads the symbol a_i written on the tape under the head at position i.
2. Based on this symbol and on the current state q, one writes a new symbol a'_i at the same position, and changes the state to q'.
3. Similarly, one decides a movement direction, moving either to the left (position $i-1$) or the right (position $i+1$).

By repeating the 3 elementary steps, one can compute all the programs computable by any computer.

Let's define more precisely the symbols, the states, and the transition between those.

Definition 1 A Turing machine is a 5-uple $M = (K, \Sigma, \delta, q_0, H)$.

K: a non-empty finite set. The elements of K are called *states*.

Σ: a non-empty finite alphabet. The elements of Σ are called *symbols*. Σ contains the special *blank symbol* B.

q_0: an element of K, called the *initial state*.

H: a subset of K, containing the *halting states*.

δ: $(K \setminus H) \times \Sigma \to \Sigma \times \{L, R\} \times K$ is called the *transition function*.

After defining a machine, we still need to define its *global state*.

Definition 2 The global state of a Turing machine M is the triple of a tape state, a position, and a controller state:

$$t = (T, i, q)$$

T: $\mathbf{Z} \to K$, such that $T(n) \neq \mathrm{B}$ for only a finite number of n's.

i: an integer of $\mathbf{Z}$.

q: a state of K.

Definition 3 *The transition of a Turing machine M from global state t to t' is written $t \vdash_M t'$. It is defined as follows.*

$$\delta(q, T(i)) = (a, d, q') \Rightarrow (T, i, q) \vdash_M (T', i', q')$$

- *if* $T'(i) = a$ *and* $k \neq i$, *then* $T'(k) = T(k)$
- *if* $d = R$ *then* $i' = i + 1$, *if* $d = L$ *then* $i' = i - 1$

We write $\vdash_M^*$ for the transitive and reflexive closure of $\vdash_M$. Moreove, if q' is an halting state, and $(T, 0, q_0) \vdash_M^* (T', n, q')$, then (T', n, q') is called the *result* of applying M to T, and we write $T \triangleright_M (T', n, q')$.

Example 1 A Turing machine checking correspondence of parentheses.

Σ = $\{\mathrm{B}, \mathrm{E}, \mathrm{L}, \mathrm{R}\}$
B is blank, E is erased, L is a left parenthesis, R a right one
K = $\{R, L, C, T, F\}$
R means searching right, L searching left, C final check, T, F denote True and False
q_0 = R
H = $\{T, F\}$

δ =

$q \backslash a$	B	E	L	R
R	$(a, \leftarrow, C)$	$(a, \rightarrow, R)$	$(a, \rightarrow, R)$	$(\mathrm{E}, \leftarrow, L)$
L	$(a, \rightarrow, F)$	$(a, \leftarrow, L)$	$(\mathrm{E}, \rightarrow, R)$	$(a, \rightarrow, F)$
C	$(a, \rightarrow, T)$	$(a, \leftarrow, C)$	$(a, \rightarrow, F)$	$(a, \rightarrow, F)$

Example 2 Addition of two natural numbers $a + b = c$

Initial state of the tape (everywhere else is blank). We use binary format: $a = \sum_0^k a_i \cdot 2^i, b = \sum_0^l b_i \cdot 2^j$.

a_0	a_1	...	a_k	M	I_0	b_0	...	b_l

Final state of the tape, from which we can read $c = \sum_0^m c_i \cdot 2^i$.

M	c_0	c_1	...	c_m

Σ = $\{\mathrm{B}, 0, 1, \mathrm{M}, \mathrm{I}_0, \mathrm{I}_1\}$
K = $\{L, R, A_0, A_1, A_0', A_1', A_0'', A_1'', W_0, W_1, F, T\}$
q_0 = R

δ =

$q \backslash a$	B	0	1	M	I_0	I_1
L	$(a, \rightarrow, R)$	$(a, \leftarrow, L)$	$(a, \leftarrow, L)$	$(a, \leftarrow, L)$		
R		$(\mathrm{B}, \rightarrow, A_0)$	$(\mathrm{B}, \rightarrow, A_1)$	$(a, \rightarrow, F)$		
A_i	$(i, \leftarrow, L)$	$(a, \rightarrow, A_i)$	$(a, \rightarrow, A_i)$	$(a, \rightarrow, A_i)$	$(a, \rightarrow, A_i')$	$(a, \rightarrow, A_i'')$
A_i'	$(a, \leftarrow, W_i)$	$(\mathrm{I}_0, \leftarrow, W_i)$	$(\mathrm{I}_i, \leftarrow, W_{1-i})$			
A_i''	$(\mathrm{I}_i, \leftarrow, W_{1-i})$	$(\mathrm{I}_i, \leftarrow, W_{1-i})$	$(\mathrm{I}_1, \leftarrow, W_i)$			
W_i					$(i, \leftarrow, L)$	$(i, \leftarrow, L)$
F	$(a, \leftarrow, T)$	$(a, \rightarrow, F)$	$(a, \rightarrow, F)$		$(a, \rightarrow, A_0')$	$(a, \rightarrow, A_0'')$

Exercise 1

1. Define a Turing machine converting a natural number in unary notation (*i.e,* a sequence of n 1's) to binary notation. Hint: this amounts to adding 1 n times, starting from 0.

2. Define a Turing machine which copies the contents of the tape up to a mark M to the right end of the tape (*i.e.* replacing blank symbols).

Definition 4 *A function* $f(x)$ *is Turing computable if there is a Turing machine* M_f*, a coding function* c *from the domain of* f *to sequences of symbols of* M_f*, and a decoding function* d *from sequences of symbols of* M_f *to the codomain of* f*, such that for any valid input* x *of* f*, we have* $c(x) \triangleright_M (T', i, q)$*, with* $d(T') = f(x)$.

One can think of many encodings. In the case of natural numbers, binary notation seems sufficient. Tuples can be encoded using parentheses and commas. Note however that the encoding is going to depend of the alphabet used by the machine. For this reason, we will write $\overline{x}^M$ for x encoded using the alphabet of M.

2 Universal Turing machine

Definition 5 *A Turing machine U is* universal *if, for any Turing maching M and any input tape X, executing U on a tape where the respective encodings $\overline{M}^U$ and $\overline{X}^U$ are written will give the result $\overline{Y}^U$ whenever applying M to X would have given Y.*

Here, we need to encode two kinds of things. First, X and Y are already encoded in the alphabet of M. But the alphabet of U may be smaller than that of M. For this reason we choose to encode each symbol of M as binary number of fixed length. For instance, if M has 30 symbols, then U will need a tape 5 times longer to encode them.

To simplify the translation, we choose to first convert M into a machine using only 2 symbols.

Theorem 1 *For any Turing machineM, one can construct an equivalent Turing machine M', whose alphabet is $\Sigma_2 = \{0, 1, \mathrm{B}\}$. Moreover, the tape of M' can be restricted to extend only in one direction.*

Thanks to this, it is enough to consider Turing machines with alphabet Σ_2 and a tape extending in only one direction.

The encoding of the machine itself is more complex. However, since we were able to write it on paper, we should be able to write it on a tape. It is sufficient to right the initial and final states, and the 5-uples of the transition function.

Here is a snapshot of the tape while U is running.

			$q(t)$	$s(t)$	M's 5-uples	
...	P	...			$\overline{M}^U$	
	M's tape		M's state	symbol at P	M's description	

U should do the following operations

1. Search the 5-uple corresponding to $(q(t), s(t))$ in the table. Finish if not found.
2. Replace P by $q(t+1)$ from the 5-uple.
3. Move either left or right, read the symbol there, and replace it by P.
4. Write the symbol as $s(t+1)$, and go back to 1.

Theorem 2 *One can construct a universal Turing machine.*

3 Undecidable problems

There are problems no Turing machine can solve. Moreover, if one accepets the Church-Turing Thesis, those problems cannot be solved by any effective procedure.

Definition 6 *An* effective procedure *is a set of rules together with a mechanism to apply them.*

Church-Turing Thesis Any effective procedure can be implemented by a Turing machine.

As the definition of "effective procedure" is overly general, one cannot prove this thesis. However, all mechanisms discovered up to know fall into the Turing thesis.

Below, following almost all computer scientists, we will assume the Church-Turing Thesis.

3.1 Halting problem

One cannot easily predict the number of steps needed before a Turing machine stops. In some cases it may never stop. It would be very nice to be able to tell in advance whether a Turing machine will eventually stop or not.

The halting problem is, given a Turing machine M and a tape T, to decide whether when we apply M to T the execution will eventually stop or not.

Since this task is difficult for a human, we want the computer to do that, using an effective procedure. If we admit the Church-Turing thesis, we are looking for a machine H such that, given $\overline{M}$ and T as input, it will eventually stops indicating either *Yes* or *No*. For $\overline{M}$, we can use the same encoding as with the universal machine.

Unfortunately, there is not such machine.

Theorem 3 *There is no effective procedure allowing to decide whether an arbitrary Turing machine applied to some input will enventually halt or not.*

Proof We assume the existence of such a Turing machine H, and transform it in the following way.

First we define a machine H' which, receiving as single input $\overline{M}$, will decide whether M applied to $\overline{M}$ halts. That is, $H'(\overline{M}) = H(\overline{M}, \overline{M})$

Next, we define a machine H^* such that it halts if H' returns *No*, and goes into an infinite loop if it returns *Yes*.

Finally, we want to decide whether $H^*(\overline{H^*})$ halts, by executing $H(\overline{H^*}, \overline{H^*}) = H'(\overline{H^*})$. If H returns *Yes*, then it means that $H^*(\overline{H^*})$ halts, which is in contradiction with the definition of H^*. If it returns *No*, then $H^*(\overline{H^*})$ halts, and we have again a contradiction. □

3.2 Printing problem

Theorem 4 *There is no effective procedure deciding whether a Turing machine will eventually write some symbol S_0 to its tape or not.*

Proof Starting from a Turing machine M whose alphabet does not contain S_0, we modify it by adding an extra action after each of its halting states, such that it writes S_0 upon termination.

As a result, deciding the printing problem for this machine would be equivalent to deciding the halting problem for M, which is impossible in general. □

3.3 Empty tape halting problem and Uniform halting problem

Theorem 5 *There is no effective procedure deciding whether an arbitrary Turing machine halts given an empty tape as input.*

Theorem 6 *There is no effective procedure deciding whether an arbitrary Turing machine always halts for any input.*

Proof We assume that A and C are two symbols not included in M's alphabet, and modify it as follows. Before starting execution, we write $\boxed{A}\boxed{B}\boxed{C}$ to the tape, moving the head back so that it looks at the middle B. Then, if we meet an A going to the left, we replace it by $\boxed{A}\boxed{B}$, and if we meet a C going to the right, we replace it by $\boxed{B}\boxed{C}$.

As a result, the uniform halting problem for this new machine is equivalent to the empty tape halting problem for M. □

Exercise 2 *Prove that the empty tape halting problem is undecidable, by reducing it to the general halting problem.*