Theory of Computation

Thomas Zeugmann

Hokkaido University
Laboratory for Algorithmics

http://www-alg.ist.hokudai.ac.jp/~thomas/ToC/

Lecture 12: Turing Machines

Turing Machines I

After having dealt with partial recursive functions, we turn our attention to Turing machines introduced by Alan Turing.

His idea was to formalize the notion of "intuitively computable" functions by using the four properties of an algorithm which we have stated at the beginning of Lecture 11. Starting from these properties, he observed that the primitive operations could be reduced to a level such that a machine can execute the whole algorithm. For the sake of simplicity, here we consider one-tape Turing machines.

Turing Machines II

A one-tape Turing machine consists of an infinite tape which is divided into cells. Each cell can contain exactly one of the tape-symbols. Initially, we assume that all cells of the tape contain the symbol $*$ except those in which the actual input has been written. Moreover, we enumerate the tape cells as shown in Figure 1.

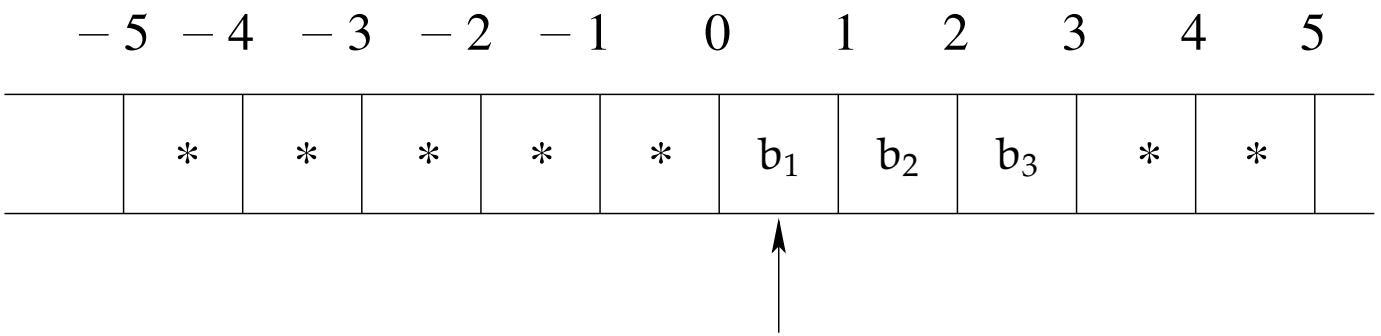

Figure 1: The tape of a Turing machine with input $b_1 b_2 b_3$.

Turing Machines III

Furthermore, the Turing machine possesses a read-write head. This head can observe one cell at a time. Additionally, the machine has a finite number of states it can be in and a set of instructions it can execute. Initially, it is always in the start state z_s and the head is observing the leftmost symbol of the input, i.e., the cell 0. We indicate the position of the head by an arrow pointing to it.

Turing Machines III

Furthermore, the Turing machine possesses a read-write head. This head can observe one cell at a time. Additionally, the machine has a finite number of states it can be in and a set of instructions it can execute. Initially, it is always in the start state z_s and the head is observing the leftmost symbol of the input, i.e., the cell 0. We indicate the position of the head by an arrow pointing to it.

Then, the machine works as follows. When in state z and reading tape symbol b, it writes tape symbol b' into the observed cell, changes its state to z' and moves the head either to the left (denoted by L) or to the right (denoted by R) or does not move the head (denoted by N) provided (z, b, b', H, z') is in the instruction set of the Turing machine, where $H \in \{L, N, R\}$. The execution of one instruction is called *step*.

Turing Machines IV

When the machine reaches the *distinguished* state z_f (the *final* state), it stops. Thus, formally, we can define a Turing machine as follows.

Definition 1

$M = [B, Z, A]$ is called ***deterministic one-tape Turing machine*** if B, Z, A are non-empty finite sets such that $B \cap Z = \emptyset$ and

(1) $\mathrm{card}(B) \geqslant 2 \quad (B = \{*, |, \ldots\}) \quad$ (tape-symbols),

(2) $\mathrm{card}(Z) \geqslant 2 \quad (Z = \{z_s, z_f, \ldots\}) \quad$ (set of states),

(3) $A \subseteq Z \setminus \{z_f\} \times B \times B \times \{L, N, R\} \times Z \quad$ (instruction set), where for every $z \in Z \setminus \{z_f\}$ and every $b \in B$ there is precisely one 5-tuple $(z, b, \cdot, \cdot, \cdot)$.

Turing Machines V

Often, we represent the instruction set A in a table, e.g.,

	$*$	$\vert$	b_2	$\ldots$	b_m
z_s	$b'Nz_3$				
z_1	.				
.	.				
.	.				
.	.				
z_n	.				

A Turing table

If the instruction set is small, it often convenient to write $zb \rightarrow b'Hz'$, where $H \in \{L, N, R\}$ instead of (z, b, b', H, z'). Also, we usually refer to the instruction set of a Turing machine M as to the ***program*** of M.

Turing Computations I

Next, we have to explain how a Turing machine is computing a function. Our primary concern are functions from $\mathbb{N}^n$ to $\mathbb{N}$, i.e., $f\colon \mathbb{N}^n \to \mathbb{N}$. Therefore, the inputs are tuples $(x_1, \ldots, x_n) \in \mathbb{N}^n$.

We shall reserve the special tape symbol # to separate x_i from x_{i+1}. Moreover, for the sake of simplicity, in the following we shall assume that numbers are unary encoded, e.g., number 0 is represented by $*$, number 1 by $|$, number 2 by $||$, number 3 by $|||$, a.s.o. Note that this convention is no restriction as long as we do not consider the *complexity* of a Turing computation.

Turing Computations II

Furthermore, it is convenient to introduce the following notations. Let $f\colon \mathbb{N}^n \to \mathbb{N}$ be any function. If the value $f(x_1,\ldots,x_n)$ is not defined for a tuple $(x_1,\ldots,x_n) \in \mathbb{N}^n$ then we write $f(x_1,\ldots,x_n)\uparrow$. If $f(x_1,\ldots,x_n)$ is defined then we write $f(x_1,\ldots,x_n)\downarrow$.

Turing Computations III

Definition 2

Let M be any Turing machine, let $n \in \mathbb{N}^+$ and let $f\colon \mathbb{N}^n \to \mathbb{N}$ be any function. We say that M ***computes*** the function f if for all $(x_1, \ldots, x_n) \in \mathbb{N}^n$ the following conditions are satisfied:

(1) If $f(x_1, \ldots, x_n) \downarrow$ and if $x_1\#\ldots\#x_n$ is written on the empty tape of M and M is started on the leftmost symbol of $x_1\#\ldots\#x_n$ in state z_s, then M stops after having executed finitely many steps in state z_f. Moreover, if $f(x_1, \ldots, x_n) = 0$, then the symbol observed by M in state z_f is $*$. If $f(x_1, \ldots, x_n) \neq 0$, then the string beginning in the cell observed by M in state z_f (read from left to right) of consecutive $|$ denotes the results (cf. Figure 2).

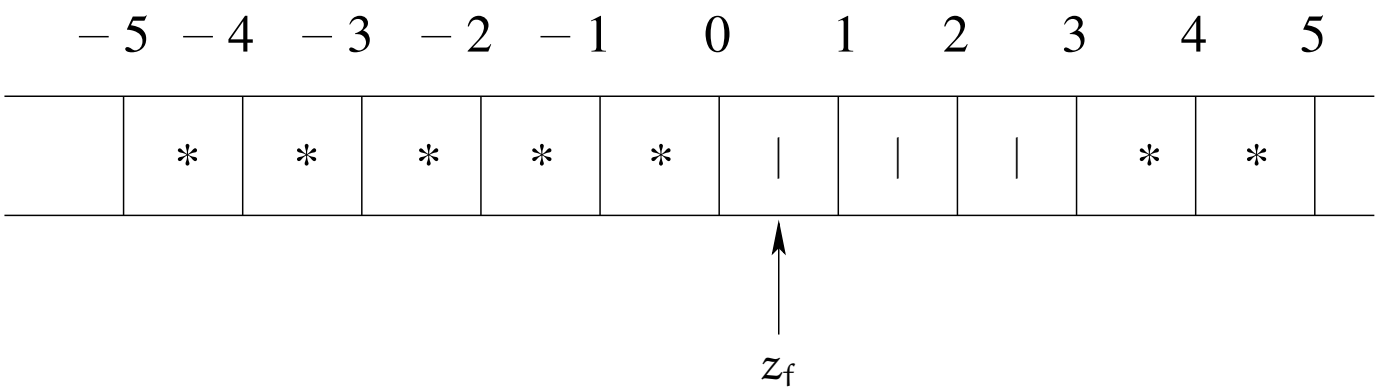

Figure 2: The tape of a Turing machine with result 3 (written as |||).

Definition 3 (continued)

(2) If $f(x_1, \ldots, x_n) \uparrow$ and if $x_1\#\ldots\#x_n$ is written on the empty tape of M (beginning in cell 0) and M is started on the leftmost symbol of $x_1\#\ldots\#x_n$ in state z_s then M does not stop.

Turing Computations IV

By f_M^n we denote the function from $\mathbb{N}^n$ to $\mathbb{N}$ computed by Turing machine M.

Definition 4

Let $n \in \mathbb{N}^+$ and let $f\colon \mathbb{N}^n \to \mathbb{N}$ be any function. The function f is said to be ***Turing computable*** if there exists a Turing machine M such that $f_M^n = f$. Furthermore, we set

$$\begin{aligned} \mathcal{T}^n &= \text{the set of all } n\text{-ary Turing computable functions.} \\ \mathcal{T} &= \bigcup_{n \geqslant 1} \mathcal{T}^n = \text{the set of all Turing computable functions.} \end{aligned}$$

Turing Computations V

Now, it is only natural to ask which functions are Turing computable. The answer is provided by the following theorem.

Theorem 1

The class of Turing computable functions is equal to the class of partial recursive functions, i.e., $\mathcal{T} = \mathcal{P}$.

Proof I

For showing $\mathcal{P} \subseteq \mathcal{T}$ it suffices to prove the following:

(1) The functions Z, S and V are Turing computable.

(2) The class of Turing computable functions is closed under the Operations (2.1) through (2.6) defined in Lecture 11.

Proof II

A Turing machine computing the constant zero function can be easily defined as follows. Let $M = [\{*, |\}, \{z_s, z_f\}, A]$, where A is the following set of instructions:

$$\begin{aligned} z_s| &\rightarrow |Lz_f \\ z_s* &\rightarrow *Nz_f\ . \end{aligned}$$

That is, if the input is not zero, then M move its head one position to the left and stops. By our definition of a Turing machine, then M observes in cell -1 a $*$, and thus its output is 0. If the input is zero, then M observes in cell 0 a $*$, leaves it unchanged, does not move its head and stops. Clearly, $f^1_M(x) = 0$ for all $x \in \mathbb{N}$.

Proof III

The successor function S is computed by the following Turing machine $M = [\{*, |\}, \{z_s, z_f\}, A]$, where A is the following set of instructions:

$$\begin{aligned} z_s| &\rightarrow |\,Lz_s \\ z_s* &\rightarrow |\,Nz_f\ . \end{aligned}$$

That is, the machine just adds a $|$ to its input and stops. Thus, we have $f^1_M(x) = S(x)$ for all $x \in \mathbb{N}$.

Proof IV

Furthermore, the predecessor function is computed by $M = [\{*, |\}, \{z_s, z_f\}, A]$, where A is the following set of instructions:

$$\begin{aligned} z_s| &\rightarrow *Rz_f \\ z_s* &\rightarrow *Nz_f\ . \end{aligned}$$

Now, the Turing machine either observes a $|$ in cell 0 which it removes and then the head goes one cell to the right or it observes a $*$, and stops without moving its head. Consequently, $f^1_M(x) = V(x)$ for all $x \in \mathbb{N}$. This proves Part (1).

Proof V

Next, we sketch the proof of Part (2). This is done in a series of claims.

Claim 1. (Introduction of fictitious variables)
Let $n \in \mathbb{N}^+$*; then we have: if* $\tau \in \mathcal{T}^n$ *and*
$\psi(x_1, \ldots, x_n, x_{n+1}) = \tau(x_1, \ldots, x_n)$*, then* $\psi \in \mathcal{T}^{n+1}$.

Intuitively, it is clear that Claim 1 holds. In order to compute ψ, all we have to do is to remove x_{n+1} from the input tape, then moving the head back into the initial position and then we start the Turing program for τ. Consequently, $\psi \in \mathcal{T}^{n+1}$. We omit the details.

Proof VI

Claim 2. (Identifying variables)
Let $n \in \mathbb{N}^+$*; then we have: if* $\tau \in \mathcal{T}^{n+1}$ *and* $\psi(x_1, \ldots, x_n) = \tau(x_1, \ldots, x_n, x_n)$*, then* $\psi \in \mathcal{T}^n$.

For proving Claim 2, we only need a Turing program that copies the last variable (that is, x_n). Thus, the initial tape inscription

$$* \; * \; x_1 \# \ldots \# x_n \; * \; *$$

is transformed into

$$* \; * \; x_1 \# \ldots \# x_n \# x_n \; * \; *$$

and the head is moved back into its initial position and M is put into the initial state of the program computing τ. Now, we start the program for τ. Consequently, $\psi \in \mathcal{T}^n$. Again, we omit the details.

Proof VII

Claim 3. (Permuting variables)
Let $n \in \mathbb{N}^+$, $n \geqslant 2$ *and let* $i \in \{1, \ldots, n\}$*; then we have:*
if $\tau \in \mathcal{T}^n$ *and*
$\psi(x_1, \ldots, x_i, x_{i+1}, \ldots, x_n) = \tau(x_1, \ldots, x_{i+1}, x_i, \ldots, x_n)$*, then*
$\psi \in \mathcal{T}^n$.

Claim 3 can be shown *mutatis mutandis* as Claim 2, and we therefore omit its proof here.

Proof VIII

Claim 4. (Composition)
Let $n \in \mathbb{N}$ *and* $m \in \mathbb{N}^+$. *Let* $\tau \in \mathcal{T}^{n+1}$, *let* $\psi \in \mathcal{T}^m$ *and let* $\phi(x_1, \ldots, x_n, y_1, \ldots, y_m) = \tau(x_1, \ldots, x_n, \psi(y_1, \ldots, y_m))$.
Then $\phi \in \mathcal{T}^{n+m}$.

The proof of Claim 4 is a bit more complicated. Clearly, the idea is to move the head to the right until it observes the first symbol of y_1. Then we could start the Turing program for ψ. If $\psi(y_1, \ldots, y_m) \uparrow$, then the machine for ϕ also diverges on input $x_1, \ldots, x_n, y_1, \ldots, y_m$. But if $\psi(y_1, \ldots, y_m) \downarrow$, our goal would be to obtain the new tape inscription

$$* \; * \; x_1 \# \ldots \# x_n \# \psi(y_1, \ldots, y_m) * \; *$$

and then to move the head to the left such that it observes the first symbol of x_1. This would allow us to start then the Turing program for τ.

Proof IX

So, the difficulty we have to overcome is to ensure that the computation of $\psi(y_1, \ldots, y_m)$ does not overwrite the x_i.
Thus, we need the following lemma.

Lemma 2 (M^+)

For every Turing machine M there exists a Turing machine M^+ such that

(1) $f^n_M = f^n_{M^+}$ *for all* n.

(2) M^+ *is never moving left to the initial cell observed when starting its computation.*

(3) *For all* $x_1, \ldots, x_n$: *If* $f^n_{M^+}(x_1, \ldots, x_n) \downarrow$, *then the computation stops with the head observing the same cell it has observed when the computation started and right to the result computed there are only* $*$ *on the tape.*

Proof of Lemma M^+

The idea to show Lemma M^+ is as follows. M^+ does the following.

- It moves the whole input one cell to the right,
- it marks the initial cell with a special symbol, say L,
- it marks the first cell right to the moved input with a special symbol, say E,
- it works then as M does except the following three exceptions.

Proof of Lemma M^+

- If M^+ reaches the cell marked with E, say in state z, then it moves the marker E one cell to the right, moves the head then one position to the left and writes a $*$ into this cell (that is into the cell that originally contained E) and continues to simulate M in state z with the head at this position (that is, the cell originally containing E and now containing $*$).
- If M in state z enters the cell containing L, then the whole tape inscription between L and E (including E but excluding L) is moved one position to the right. In the cell rightmost to L a $*$ is written and M^+ continues to simulate M observing this cell.

Proof of Lemma M^+

- If M stops, then M^+ moves the whole result left such that the first symbol of the result is now located in the cell originally containing L. Furthermore, into all cells starting from the first position right to the moved result and ending in E a $*$ is written and then the head is moved back to the leftmost cell containing the result. Then M^+ also stops.

This proves Lemma M^+.

Having Lemma M^+, now Claim 4 follows as described above.

The remaining claims for primitive recursion and μ-recursion are left as an exercise. This shows $\mathcal{P} \subseteq \mathcal{T}$.

Proof X

Finally, we have to show $\mathcal{T} \subseteq \mathcal{P}$. Let $n \in \mathbb{N}+$, let $f \in \mathcal{T}^n$ and let M be any Turing machine computing f. We define the functions t (time) and r (result) as follows.

$$t(x_1, \ldots, x_n, y) = \begin{cases} 1\text{ , if } M, \text{ when started on } x_1, \ldots, x_n, \\ \quad \text{stops after having executed at most} \\ \quad y \text{ steps;} \\ 0\text{ , otherwise.} \end{cases}$$

$$r(x_1, \ldots, x_n, y) = \begin{cases} f(x_1, \ldots, x_n)\text{ , } & \text{if } t(x_1, \ldots, x_n, y) = 1; \\ 0\text{ , } & \text{otherwise.} \end{cases}$$

Now, one can show that $t,\ r \in \mathit{Prim}$. Furthermore, Kleene showed the following normal form theorem using t and r as defined above.

Proof XI

Theorem 3 (Kleene)

For every $f \in \mathcal{T}^n$, $n \in \mathbb{N}^+$ *there are functions* t, $r \in$ *Prim such that*

$$f(x_1, \ldots, x_n) = r(x_1, \ldots, x_n, \mu y[t(x_1, \ldots, x_n, y) = 1]) \tag{1}$$

for all $x_1, \ldots, x_n \in \mathbb{N}^n$.

We do not prove this theorem here, since a proof is beyond the scope of this course.
Now, $\mathcal{T} \subseteq \mathcal{P}$ follows from the primitive-recursiveness of t and r and Equation (1), since the latter one shows that one has to apply the μ-operator exactly ones (Operation (2.6)) and the resulting function is composed with r by using Operation (2.4).
Consequently, $f \in \mathcal{P}$. ∎

Church's Thesis

The latter theorem is of fundamental epistemological importance. Though we started from completely different perspectives, we finally arrived the same set of computable functions. Subsequently, different approaches have been proposed to formalize the notion of "intuitively computable" functions. These approaches comprise, among others, Post algorithms, Markov algorithms, Random-access machines (abbr. RAM), and Church' λ-calculus. As it turned out, all these formalizations define the same set of computable functions, i.e., the resulting class of functions is equal to the Turing computable functions. This led Church to his famous thesis.

Church's Thesis. *The class of the "intuitively computable" functions is equal to the class of Turing computable functions.*

The Universal Turing Machine I

Now, we aim to show that there is *one* Turing machine which can compute all partial recursive functions.

The Universal Turing Machine I

Now, we aim to show that there is *one* Turing machine which can compute all partial recursive functions.

First, using our results concerning pairing functions, it is easy to see that we can encode any n-tuple of natural numbers into a natural number. Moreover, as we have seen, this encoding is even primitive recursive. Thus, in the following, we use $\mathcal{P}$ to denote the set of all partial recursive functions from $\mathbb{N}$ to $\mathbb{N}$.

The Universal Turing Machine I

Now, we aim to show that there is *one* Turing machine which can compute all partial recursive functions.

First, using our results concerning pairing functions, it is easy to see that we can encode any n-tuple of natural numbers into a natural number. Moreover, as we have seen, this encoding is even primitive recursive. Thus, in the following, we use $\mathcal{P}$ to denote the set of all partial recursive functions from $\mathbb{N}$ to $\mathbb{N}$.

Next, let us consider any partial recursive function $\psi(i, x)$, i.e., $\psi\colon \mathbb{N}^2 \to \mathbb{N}$. Thus, if we fix the first argument i, then we obtain a partial recursive function of one argument denoted ψ_i.

The Universal Turing Machine II

Thus, we can visualize all functions of one argument computed by ψ as follows (cf. Figure 3).

ψ_0	$\psi(0,0)$	$\psi(0,1)$	$\psi(0,2)$	$\psi(0,3)$	$\psi(0,4)$	$\ldots$
ψ_1	$\psi(1,0)$	$\psi(1,1)$	$\psi(1,2)$	$\psi(1,3)$	$\psi(1,4)$	$\ldots$
ψ_2	$\psi(2,0)$	$\psi(2,1)$	$\psi(2,2)$	$\psi(2,3)$	$\psi(2,4)$	$\ldots$
ψ_3	$\psi(3,0)$	$\psi(3,1)$	$\psi(3,2)$	$\psi(3,3)$	$\psi(3,4)$	$\ldots$
ψ_4	$\psi(4,0)$	$\psi(4,1)$	$\psi(4,2)$	$\psi(4,3)$	$\psi(4,4)$	$\ldots$
ψ_5	$\psi(5,0)$	$\ldots$				
.						
.						
.						
ψ_i	$\ldots$	$\ldots$				
$\ldots$						

Figure 3: A two dimensional array representing all ψ_i.

The Universal Turing Machine III

For having an example, consider $\psi(i, x) = ix$; then e.g., $\psi_7(x) = 7x$.
Therefore, it is justified to call every function $\psi \in \mathcal{P}^2$ a *numbering*.

Definition 5

A function $\psi \in \mathcal{P}^2$ is said to be ***universal*** for $\mathcal{P}$ if

$$\{\psi_i \mid i \in \mathbb{N}\} = \mathcal{P} .$$

The Universal Turing Machine III

For having an example, consider $\psi(i, x) = ix$; then e.g., $\psi_7(x) = 7x$.
Therefore, it is justified to call every function $\psi \in \mathcal{P}^2$ a *numbering*.

Definition 5

A function $\psi \in \mathcal{P}^2$ is said to be ***universal*** for $\mathcal{P}$ if

$$\{\psi_i \mid i \in \mathbb{N}\} = \mathcal{P} .$$

Clearly, now the interesting question is whether or not a universal $\psi \in \mathcal{P}^2$ for $\mathcal{P}$ does exist. If there is a universal ψ for $\mathcal{P}$, then, since $\mathcal{P} = \mathcal{T}$, we know that ψ is Turing computable, too. Therefore, we could interpret any Turing machine computing ψ as a ***universal Turing machine***. The following theorem establishes the existence of a universal ψ.

The Universal Turing Machine IV

Theorem 4

There exists a universal function $\psi \in \mathcal{P}^2$ for $\mathcal{P}$.

The Universal Turing Machine V

The idea is easily explained. By our theorem $\mathcal{P} = \mathcal{T}$ we know that for every $\tau \in \mathcal{P}$ there is Turing machine M such that $f^1_M = \tau$. Therefore, we aim to encode every Turing machine into a natural number. Thus, we need an injective partial recursive function *cod* such that $cod(M) \in \mathbb{N}$. Furthermore, in order to make this idea work we also need a general recursive function *decod* such that

$$decod(cod(M)) = \text{Program of } M\,.$$

If the input i to *decod* is not a correct encoding of some Turing machine, then we set $decod(i) = 0$.

The Universal Turing Machine VI

The universal function ψ is then described by a Turing machine U taking two arguments as input, i.e., i and x. When started as usual, it first computes $decod(i)$. If $decod(i) = 0$, then it computes the function Z (constant zero). Otherwise, it should simulate the program of the machine M returned by $decod(i)$.

The Universal Turing Machine VI

The universal function ψ is then described by a Turing machine $\mathcal{U}$ taking two arguments as input, i.e., i and x. When started as usual, it first computes $decod(i)$. If $decod(i) = 0$, then it computes the function Z (constant zero). Otherwise, it should simulate the program of the machine M returned by $decod(i)$.

For realizing this behavior, the following additional conditions must be met:

(1) $\mathcal{U}$ is not allowed to overwrite the program obtained from the computation of $decod(i)$,

(2) $\mathcal{U}$ must be realized by using only finitely many tape symbols and a finite set of states.

The Universal Turing Machine VII

Next, we shortly explain how all our conditions can be realized. For the sake of better readability, in the following we always denote the tape symbols by b_i and the state sets always starts with $z_s, z_f, \ldots$. Let

$$M = [\{b_1, \ldots, b_m\}, \{z_s, z_f, z_1, \ldots, z_k\}, A]$$

be given.

The Universal Turing Machine VIII

Then we use the following coding (here we write 0^n to denote the string consisting of exactly n zeros):

$$
\begin{aligned}
cod(L) &= 101 \\
cod(R) &= 1001 \\
cod(N) &= 10001 \\
cod(z_s) &= 10^4 1 \\
cod(z_f) &= 10^6 1 \\
cod(z_\ell) &= 10^{2(\ell+3)} 1 && \text{for all } \ell \in \{1, \ldots, k\}, \\
cod(b_\ell) &= 10^{2(\ell+1)+1} 1 && \text{for all } \ell \in \{1, \ldots, m\}.
\end{aligned}
$$

The Universal Turing Machine IX

The instruction set is then encoded by concatenating the codings of its parts, that is

$$cod(zb \rightarrow b'Hz') = cod(z)cod(b)cod(b')cod(H)cod(z') .$$

For example, $z_s b_1 \rightarrow b_2 N z_1$ is then encoded as

$$10^4 110^5 110^7 11000110^8 1 .$$

Now, we have m tape symbols and $k+2$ states. Thus, there must be $m(k+1)$ many instructions $I_1, \ldots, I_{m(k+1)}$ which we assume to be written down in canonical order. Consequently, we finally encode the program of M by concatenating the encodings of all these instructions, i.e.,

$$cod(M) = cod(I_1) \cdots cod(I_{m(k+1)}) .$$

This string is interpreted as a natural number written in binary.

The Universal Turing Machine X

Now, it is easy to see that *cod* is injective, i.e., if $M \neq M'$ then $cod(M) \neq cod(M')$.

Furthermore, if we use the function *cod* as described above, then *decode* reduces to check that an admissible string is given. If it is, the program of M can be directly read from the string.

Finally, we have to describe how the simulation is done. First, we have to ensure that U is not destroying the program of M. This is essentially done as outlined in Lemma M^+.

The Universal Turing Machine XI

Thus, it remains to explain how the Condition (2) is realized. Clearly, U cannot memorize the actual state of M during simulation in its state set, since this would potentially require an unlimited number of states. But U can mark the actual state in which M is on its tape (e.g., by using bold letters).

In order to ensure that U is only using finitely many tape symbols, U is not using directly b_ℓ from M's tape alphabet but just $cod(\mathsf{b}_\ell) = 10^{2(\ell+1)+1}1$. This requires just two tape symbols for the simulation. We omit the details.
The Turing machine U can thus be expressed as a partial recursive function $\psi \in \mathcal{P}^2$ via our theorem $\mathcal{P} = \mathcal{T}$. ∎

Accepting Languages I

Next, we define what does it mean that a Turing machine is accepting a language L.

Definition 6

A language $L \subseteq \Sigma^*$ is ***accepted*** by Turing machine M if for every string $w \in \Sigma^*$ the following conditions are satisfied.
If w is written on the empty tape of M (beginning in cell 0) and the Turing machine M is started on the leftmost symbol of w in state z_s then M stops after having executed finitely many steps in state z_f. Moreover,

(1) if $w \in L$ then the cell observed by M in state z_f contains a $|$. In this case we also write $M(w) = |$.

(2) If $w \notin L$ then the cell observed by M in state z_f contains a $*$. In this case we also write $M(w) = *$.

Accepting Languages II

Of course, in order to accept a language $L \subseteq \Sigma^*$ by a Turing machine $M = [B, Z, A]$ we always have to assume that $\Sigma \subseteq B$. Moreover, for every Turing machine M we define

$$L(M) =_{df} \{w \mid w \in \Sigma^* \ \wedge \ M(w) = |\}\,,$$

and we refer to $L(M)$ as to the language accepted by M.

The book contains several examples.

Thank you!