Lesson 6: Turing machines and the notion of algorithms

Theme: Turing machines and the notion of algorithms.

1 Multi-tape Turing machines

A multi-tape Turing machine is a Turing machine that has a few tapes. On each tape, the Turing machine has one head. Formally, it is defined as follows. Let \( k \geq 1 \). A \( k \)-tape Turing machine is a system \( M = \langle \Sigma, \Gamma, Q, q_0, q_{acc}, q_{rej}, \delta \rangle \), where \( \delta \) is the transition function:

\[
\delta : (Q - \{q_{acc}, q_{rej}\}) \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\text{Left}, \text{Right}\}^k
\]

A transition in \( \delta \) is written in the form:

\[
(q, a_1, \ldots, a_k) \rightarrow (p, b_1, \ldots, b_k, \alpha_1, \ldots, \alpha_k).
\]

Intuitively, it means that if the TM is in state \( q \), and on each \( i = 1, \ldots, k \), the head on tape \( i \) is reading \( a_i \), then it enters state \( p \), and for \( i = 1, \ldots, k \), the head on tape \( i \) writes the symbol \( b_i \) and moves according to \( \alpha_i \).

A configuration of \( M \) is a string of the form \( (q, \triangleleft u_1, \ldots, \triangleleft u_k) \), where \( q \in Q \), each \( u_i \) is a string over \( \Gamma \cup \{\bullet\} \) and the symbol \( \bullet \) appears exactly once in each \( u_i \). The symbol \( \bullet \) denotes the position of the head. As before, the symbol \( \triangleleft \) is the left-end marker of each tape.

The initial configuration of \( M \) on input \( w \) is \( (q_0, \triangleleft \bullet w, \triangleleft \bullet, \ldots, \triangleleft \bullet) \), i.e., the first tape initially contains the input word and all the other tapes are initially blank. The notion of “one step computation” \( C \vdash C' \) is defined similarly as in the standard Turing machine. Likewise, the conditions of acceptance and rejection are defined as when the Turing machine enters the accepting and rejecting states, respectively.

**Theorem 6.1** For every \( k \)-tape TM \( M \), where \( k \geq 2 \), there is a single tape TM \( M' \) such that for every input word \( w \), the following holds.

- If \( M \) accepts \( w \), then \( M' \) accepts \( w \).
- If \( M \) rejects \( w \), then \( M' \) rejects \( w \).
- If \( M \) does not halt on \( w \), then \( M' \) does not halt on \( w \).

**Proof.** Let \( M = \langle \Sigma, \Gamma, Q, q_0, F, \delta \rangle \) be a \( k \)-tape TM with \( k \geq 2 \). We will design a TM \( M' \) that simulates the run of \( M \) on \( w \) using only one tape. The idea is that a configuration \( (q, \triangleleft u_1, \ldots, \triangleleft u_k) \) (of \( M \)) can be viewed as a string over the alphabet \( Q \cup \Gamma \cup \{\bullet, \tilde{\triangleleft}\} \):

\[
q\tilde{\triangleleft}u_1 \cdots \tilde{\triangleleft}u_k
\]

This string can be stored in just one tape. Here \( \tilde{\triangleleft} \) is a new symbol to represent the symbol \( \triangleleft \) of \( M \).

As an algorithm, \( M' \) works as follows. On input word \( w \), do the following.

- Let \( C \) be the string \( q_0 \tilde{\triangleleft} w \tilde{\triangleleft} \cdots \tilde{\triangleleft} \bullet \), i.e., the initial configuration of \( M \) on \( w \).
- While \( C \) is not a halting configuration of \( M \), do the following.
  - Scan the string \( C \) from left to right to find out the symbol read by each “head.”
– Move the head back to the beginning of the tape.
– Change the state and the position of each head in $C$ according to the transition function $\delta$.
This can be done by scanning the string $C$ from left to right and when it encounters the symbol $\bullet$, it change its “position” according to $\delta$.

• If $C$ is an accepting configuration, ACCEPT. If $C$ is a rejecting configuration, REJECT.

Note that $M'$ uses only one string variable $C$, so in principle it suffices to use only one tape to store this string $C$. Note that when the head in tape-1 moves right to the new cell (i.e., the length of $u_1$ increases), $M'$ has to “shift” right all the strings $\prec u_2, \ldots, \prec u_k$.

Alternatively we can also “compress” the content of each cell on every tape into one symbol. For example, for $k = 3$, a configuration $(q, \prec 01 \bullet 1 \sqcup, \prec \bullet 1 \sqcup, \prec \bullet 1 \sqcup)$ can be encoded as:

$q, \prec (0, 1, 0) (1, \bullet \sqcup, \bullet 1) (\bullet 1, \sqcup, \sqcup) \sqcup$

where each $(0, 1, 0)(1, \bullet \sqcup, \bullet 1), (\bullet 1, \sqcup, \sqcup)$ is represented by a symbol in the tape alphabet of $M'$. This encoding can be illustrated as follows.

| tape-1 | $\prec$ | 0 | 1 | $\bullet$ | $\sqcup$ |
| tape-2 | $\prec$ | 1 | $\bullet$ | $\sqcup$ | $\sqcup$ |
| tape-3 | $\prec$ | 0 | $\bullet$ | $\sqcup$ | $\sqcup$ |
| the encoding | $\prec$ | $(0, 1, 0)$ | $(1, \bullet \sqcup, \bullet 1)$ | $(\bullet 1, \sqcup, \sqcup)$ | $\sqcup$ |

More precisely, the string $C$ is of the form $(q, \prec a_1 a_2 \cdot \cdot \cdot a_n \sqcup)$, where each $a_i \in (\Gamma \cup \bullet \Gamma)^k$ and $\bullet \Gamma$ denotes a new alphabet whose symbols are used to represent $\bullet a$, for each $a \in \Gamma$.

**Remark 6.2** In the proof of Theorem [6.1], the TM $M'$ simulates the run of $M$ by remembering the “last” configuration $C$ which contains the state $q$. Since the TM $M$ has only some fixed number of states, the TM $M'$ can actually remember the state $q$ in its state. So, in principle the string $C$ does not need to contain the state $q$. It suffices that $C$ contains only the content of each tape, i.e., of the form: $\sim u_1 \cdot \cdot \cdot \sim u_k$.

**Remark 6.3** Intuitively one can view a tape in TM as a variable in a computer program. A $k$-tape TM can be viewed as a computer program that uses $k$ variables. Likewise, a computer program that uses $k$ variables can be viewed as a $k$-tape TM. See the Appendix.

In view of Remark [6.3] to avoid being overly technical in our presentation, the term “Turing machine” and “algorithm” will often be used interchangeably. When we describe a TM (especially in the proofs in this and the subsequent lessons), we will often just describe an algorithm in some acceptable format. We will write capital ACCEPT to denote that the TM enters the accept state and REJECT to denote that the TM enters the reject state.

2 Some theorems on decidable and recognizable languages

**Theorem 6.4**

• If a language $L$ (over the alphabet $\Sigma$) is decidable, so is its complement $\Sigma^* - L$.
• If both a language $L$ and its complement $\Sigma^* - L$ are recognizable, then $L$ is decidable.
Theorem 6.4. We may assume that both are 1-tape TM. The proof is actually similar to the second item of Theorem 1.3 (in Lesson 1), where to prove that regular languages are closed under intersection, we can “run” two DFA simultaneously by taking the Cartesian product $Q_1 \times Q_2$ as the set of states.

\[ Q_1 \times Q_2 \]

\[ \text{Theorem 6.5} \quad \text{Decidable languages are closed under union, intersection, concatenation and Kleene star.} \]

\[ \text{Proof.} \quad \text{In the following let } M_1 \text{ and } M_2 \text{ be the TM that decide languages } L_1 \text{ and } L_2, \text{ respectively.} \]

\[ L_1 \cup L_2 \]

\[ L_1 \cdot L_2 \]

\[ L_1^* \]

\[ \text{Note that there are only } |w| + 1 \text{ possible pairs } (v_1, v_2) \text{ where } v_1 v_2 = w. \] (It is possible that $v_1$ or $v_2$ is $\varepsilon$.)

\[ \text{Theorem 6.6} \quad \text{Recognizable languages are closed under union and intersection.} \]
Proof. In the following let $M_1$ and $M_2$ be the TM that recognize languages $L_1$ and $L_2$, respectively. We may assume that both are 1-tape TM. The proof is actually similar to the second item of Theorem 6.4.

(Closure under union) The TM $M_\cup$ that recognizes $L_1 \cup L_2$ works as follows. It has two tapes. First, it copies the input word into the second tape. Then, it runs $M_1$ and $M_2$ on the first and second tape simultaneously. It accepts if and only if at least one of $M_1$ or $M_2$ accepts.

By definition of $M_1$ and $M_2$, every word $w \in L_1 \cup L_2$ is accepted by at least one of $M_1$ or $M_2$. Thus, $M_\cup$ recognizes the language $L_1 \cup L_2$ correctly.

(Closure under intersection) Similar to above.

Remark 6.7 We should note that recognizable languages are also closed under concatenation and Kleene star. In fact, we can already prove it in this lesson, but the proof will be quite technical. So we will postpone the proof until Lesson 9. There we will use the notion of “nondeterministic” TM to obtain a neater and clearer proof.

We should stress that recognizable languages are not closed under complement. We will see this in Lesson 7.

Appendix

A An informal definition of algorithm

We define an algorithm (informally) as consisting of one “main” Boolean function of the form:

```plaintext
Boolean main (string w) {
    statement;
    ...;
    statement;
}
```

and some (finite number of) functions of the form:

```plaintext
⟨value-type⟩ function ⟨function-name⟩ ([⟨var-name⟩,...,⟨var-name⟩]) {
    statement;
    ...;
    statement;
}
```

Statements in the algorithm are of the following form:

- $⟨var-name⟩ := ⟨expression⟩$;
- $⟨var-name⟩ := ⟨function-name⟩(⟨var-name⟩,...,⟨var-name⟩)$;
- `return ⟨variable-name⟩/⟨some-value⟩;`
• if ⟨condition⟩
  {
    statement;
    :
    statement;
  }
else
  {
    statement;
    :
    statement;
  }

Note that we define our algorithm to mimic closely the C++ language so that we can be "convinced" every C++ program can be rewritten as our algorithm.

In our algorithm variables can only store Boolean or string values. Note that there is no while-loop, since it can be implemented as a recursive function.

We loosely define an "expression" as any reasonable "basic" computation which includes:

• Concatenating two strings.
• Shift left/right of a string.
• Change the symbol in a position in a string.

Since 0-1 strings can be used to represent numbers, "basic" computation also includes:

• Adding/subtracting/multiplying/dividing two numbers.
• Enumerating all the numbers between 1 and some number $n$.
• Measuring the length of a 0-1 string.
• Enumerating all the 0-1 strings with length between 1 and some number $n$.

"Condition" for if statement includes:

• Checking whether two numbers are equal, or whether one number is bigger than the other.
• Checking whether two strings are equal, or whether one string is lexicographically "bigger" than the other.

One can argue that every C++ computer program can be written as an algorithm defined above. Note that when we write an algorithm (or any computer program, in fact), it uses only a fixed number of variables (including variables for data structures such as linked list, arrays, etc). Multi-tape Turing machines and our definition of algorithms above are equivalent in the sense that a $k$-tape Turing machine can be viewed as an algorithm that uses $k$ variables, and conversely, an algorithm that uses $k$ variables can be viewed as a $k$-tape Turing machine.