COMP4141 Theory of Computation

Lecture 8 Turing Machines

Ron van der Meyden

CSE, UNSW

Revision: 2013/04/04

(Credits: D Dill, K Englehardt, M Sipser, W Thomas, T Wilke)

Turing Machines

Models of computation so far have been highly restricted.
The next model is really the most general model of computation.

This material is full of surprises. The proof that some problems are undecidable is one of the greatest intellectual accomplishments of humanity.

First surprise: There *is* a most general model of computation.

It seems that all general models of computation have the same power. This is called the "Church-Turing thesis." It's not really provable, but it seems to be true.

Turing Machines

Finite control with a *tape* which is used for input and for unbounded storage. There is a "head" that can read/write the tape.

- Tape is infinite in both directions.
- There is a special "blank" symbol ($\sqcup$).
- All but a finite number of positions are blank at any given time.

Definition

A *move* of a TM is based on the current state and the symbol currently being scanned, and it changes state, writes a new symbol, and moves left or right.

Def. of Turing Machines

A TM is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ with components:

- Q : states
- Σ : input symbols, not containing the *blank symbol* $\sqcup$
- $\Gamma \supseteq \Sigma$: tape symbols, containing $\sqcup$
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$
- q_0 : start state
- q_{accept} : accept state
- q_{reject} : reject state ($\neq q_{\text{accept}}$)

TM configurations

A *configuration* for a TM is a triple $(u, q, v) \in \Gamma^* \times Q \times \Gamma^*$ typically written uqv.

It represents the situation in which

- q is the current state.
- The tape content is uv
- the head is at the first symbol of v
- The infinite string of blanks to the left and right of the non-blanks uv are suppressed.

Special TM configurations

For $w \in \Sigma^*$, configurations of the form $q_0 w$ are called *start configurations*.

Configurations of the form $uq_{\text{accept}}v$ are called *accepting configurations*.

Configurations of the form $uq_{\text{reject}}v$ are called *rejecting configurations*.

NB

Various other first definitions of TMs exist in the literature but it is usually easy to move between the different formats.

Definition

For $a, b, c \in \Gamma$ $u, v \in \Gamma^*$, and $q, q' \in Q$.
Configuration $uaqbv$ *yields* $uq'acv$ if $\delta(q, b) = (q', c, \mathrm{L})$.
Configuration $uaqbv$ *yields* $uacq'v$ if $\delta(q, b) = (q', c, \mathrm{R})$.

Definition

A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ *accepts* input $w \in \Sigma^*$ if a sequence $C_1, \ldots, C_n$ of configurations exists, where

1. $C_1 = q_0 w$,
2. each C_i yields C_{i+1}, and
3. C_n is an accepting configuration.

The *language* $L(M)$ *of* M is the set of strings accepted by M.

Definition

$L \in \Sigma^*$ is *Turing-recognisable* (or *recursively enumerable*) if some TM M recognises it, i.e. $L = L(M)$.

A TM can reject some input either by eventually entering q_{reject} or by *diverging*, e.g., entering a loop or running off to the right. Divergence causes trouble. TMs that never diverge are called *deciders*.

Definition

$L \in \Sigma^*$ is *(Turing-)decidable* (or *recursive*) if some TM decides it.

Stay put TMs

So far, our TMs had to move left or right in each step. Suppose we add a 3rd option—to stay put, so the type of a TM's transition function becomes:

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

Theorem

Every stay put TM has an equivalent ordinary TM.

Proof.

Replace each S transition with two transitions, a R followed by a L. $\square$

As we shall see, the TM model is fairly *robust* when it comes to variations and extensions.

Sipser's TM

Sipser's TMs work with a *left-bounded tape*. The head starts on the left-most square of the tape with the head on the first letter of the input.

Should the transition function suggest to move left when the head is already at the left-most position, the head stays put.

Theorem

Every doubly infinite tape TM has an equivalent left-bounded TM.

Proof idea.

To simulate a doubly infinite tape TM by a left-bounded one, we split the left-bounded tape into an upper and a lower half. The upper half represents the right half of the tape (from the initial head position onwards) and the lower half represents the left half. An extra state component keeps track of the half in which the head currently is. □

Multitape TMs

A *multitape TM* has multiple tapes, each with its own head for reading and writing. Initially the input appears on tape 1, and the others start out blank. The type of the transition function of a k-tape TM becomes:

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k$$

Theorem

Every multitape TM has an equivalent single-tape TM.

[Sipser] has a proof but here's a different one inspired by [Hopcroft, Motwani & Ullman].

Proof idea.

Consider a k-tape TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$. Simulate M with a single-tape TM $N = (Q', \Sigma', \Gamma', \delta', q_0', q_{\text{accept}}', q_{\text{reject}}')$ where the crux is that the single tape of N is split into $2k$ tracks, half of which hold the tapes of M and the other half marks the positions of the k heads:

$$\Gamma' = (\Gamma \times \{\sqcup, \bullet\})^k$$

To simulate a move of M, N's head must visit the k head markers. To avoid getting lost, it must remember (in its state) how many of the markers are to its left. After visiting all head markers and storing the scanned k symbols (again in its state) N knows what M would do based on the k symbols and M's current state (stored again). So it does that by revisiting the k heads and updating the tracks and state. N's accepting (rejecting) state is entered via a S transition whenever the stored state component of the simulated M is q_{accept} (q_{reject}). $\square$

Non-Deterministic TMs

[We never discussed this but DPDAs are weaker than PDAs.]
The type of the transition function of a non-deterministic TM becomes:

$$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{\mathrm{L,R,S}\}}$$

Theorem

Every non-deterministic TM has an equivalent deterministic TM.

[A variant of this theorem holds for deciders.]

Proof idea.

We simulate any non-deterministic TM N with a deterministic TM D that breadth-first searches for q_{accept} on all possible branches of N's non-deterministic computation. If D ever finds q_{accept} on one of these branches, it accepts. Otherwise it will diverge. $\square$

Enumerators

An enumerator is a TM-like construct that starts with a blank input tape and, if it doesn't halt, may print an infinite list of strings onto the tape. It does so by every now and then entering a *print state* which means that the terminal string to the right of the head (incl. what's under the head) is printed/enumerated.

The language of an enumerator is the set of all strings it eventually prints out. (It may print strings repeatedly and in any order.)

The origin of the term *recursively enumerable* for Turing-recognisable is

Theorem

A language is Turing-recognisable iff some enumerator enumerates it.

Proof idea.

"$\Leftarrow$": Let E be an enumerator. The TM M_E works as follows. On input w it runs E and, each time E prints a word v, compares w with v and accepts if they're equal.

"$\Rightarrow$": Let M be a TM. Let $(w_i)_{i \in \mathbb{N}}$ be an enumeration of Σ^*. The enumerator E_M works as follows. Repeat for $i = 1, 2, \ldots$ to run M for i steps on each of the first i words $w_0, \ldots, w_{i-1}$ in the enumeration. If i-step computation on w_j accepts, print w_j. $\square$