Lecture 14: Recursive Languages

Instructor: Ketan Mulmuley
Scriber: Yuan Li

February 24, 2015

1 Recursive Languages

Definition 1.1. *A language $L \subseteq \Sigma^*$ is called* recursively enumerable *(r. e.) or computably enumerable if there exists a Turing machine M such that $L = L(M)$.*

Recall that $L(M) = \{w \in \Sigma^* : q_0 w \mapsto_M^* \alpha_1 p \alpha_2$, where $p \in F$, $\alpha_1, \alpha_2 \in \Gamma^*\}$, and M can loop on some input, in which that input is not in $L(M)$.

Definition 1.2. *A language L is called* recursive *if there exists a Turing machine M such that $L = L(M)$ and M halts on all inputs.*

Turing machine can also be used to compute functions. Suppose on input

$$\underbrace{00\cdots0}_{k_1}1\underbrace{00\cdots0}_{k_2}1\cdots1\underbrace{00\cdots0}_{k_l}B,$$

Turing machine M finally halts and output

$$\underbrace{00\cdots0}_{f(k_1,k_2,\ldots,k_l)}\ B,$$

where $f : \mathbb{N}^l \to \mathbb{N}$. Then we say Turing machine M computes f.

Formally, given a TM M, define $f(k_1, k_2, \ldots, k_l) = \perp$ (undefined) if M does not halt or does not halt with output in correct form; $f(k_1, k_2, \ldots, k_l) = m$ if M stops with 0^m on the tape.

Definition 1.3. *We say that $f : \mathbb{N}^l \to \mathbb{N}$ is* totally recursive *if $f = f_M$ for some TM M that always halts with output in the correct form, and thus $f(k_1, k_2, \ldots, k_l)$ is defined everywhere.*

We say that $f : \mathbb{N}^l \to \mathbb{N} \cup \{\perp\}$ is partially recursive *if $f = f_M$ for some TM M that may not always halt (with output in correct format), and thus $f(k_1, k_2, \ldots, k_l)$ is not defined everywhere.*

Proposition 1.4. *If $f : \mathbb{N}^l \to \mathbb{N}$ is totally recursive, then language $L = \{((k_1, k_2, \ldots, k_l), m) : f(k_1, \ldots, k_l) = m\}$ is recursive.*

Proof. In order to prove L is recursive, we need to prove there exists a Turing machine M such that $L(M) = L$ and M halts on all inputs.

Since $f : \mathbb{N}^l \to \mathbb{N}$ is totally recursive, by definition, there exists a TM N which, on input $(k_1, k_2, \ldots, k_l)$, (always halts and) outputs $f(k_1, \ldots, k_l)$. Our Turing machine M works as follows, on input $(k_1, k_2, \ldots, k_l, m)$, simulate N on $(k_1, k_2, \ldots, k_l)$, until N halts with output m'. Accept if and only if $m = m'$. It clear that M accepts language L, and M halts on all input, since N halts on all $(k_1, k_2, \ldots, k_l)$. □

Proposition 1.5. *If $f : \mathbb{N}^l \to \mathbb{N}$ is partially recursive, then $L = \{((k_1, k_2, \ldots, k_l), m) : f(k_1, \ldots, k_l) = m\}$ is recursively enumerable.*

The proof is similar to the Proposition 1.4, which is omitted.

Proposition 1.6. *Conversely, if $L = \{((k_1, k_2, \ldots, k_l), m)\}$ is recursively enumerable, and it is a* graph function, *i.e., for all $k_1, k_2, \ldots, k_l$, there exists at most one m such that $((k_1, k_2, \ldots, k_l), m) \in L$. Then the function $f : \mathbb{N}^l \to \mathbb{N} \cup \{\perp\}$, defined by $f(k_1, k_2, \ldots, k_l) = m$ if there exists m such that $((k_1, k_2, \ldots, k_l), m) \in L$, is partially recursive.*

Proof. In order to prove f is partially recursive, we need to construct Turing machine M, which outputs $f(k_1, k_2, \ldots, k_l)$ on input $(k_1, k_2, \ldots, k_l)$, if $f(k_1, k_2, \ldots, k_l)$ is defined. Since L is r. e., there exists a Turing machine N such that $L(N) = L$. Our machine M does the following: enumerate $(i, j) \in \mathbb{N} \times \mathbb{N}$, and when (i, j) is enumerated, simulate N on $((k_1, \ldots, k_l), i)$ for j steps, if M halts, output i. □

The following is similar to prove:

Proposition 1.7. *If $L = \{((k_1, k_2, \ldots, k_l), m)\}$ is recursive, and for all $k_1, k_2, \ldots, k_l \in \mathbb{N}$, there exists exactly one $m \in \mathbb{N}$ such that $((k_1, k_2, \ldots, k_l), m) \in L$. Then the function $f : \mathbb{N}^l \to \mathbb{N}$, defined by $f(k_1, k_2, \ldots, k_l) = m$ for the unique m such that $((k_1, k_2, \ldots, k_l), m) \in L$, is totally recursive.*

2 Enhancing Turing Machine

Turing machine is the simplest programming language which is as powerful as C, C++, Java, etc. However, it is not easy to program on Turing machine. Imagine writing a TM for weather forecasting will be very difficult.

Now we will enhance the convenience (not the power) of Turing machine.

Let us add a cache (finite storage in finite content) in the head, which can be implemented using usual one-tape Turing machine with new state $Q' = Q \times \Gamma^n$, where n is the length of the cache.

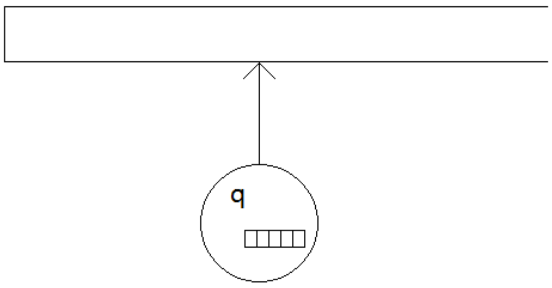

Figure 1: TM with a cache

Suppose we want to have multiple tracks (and only one head) for a Turing machine. It is equivalent to a one-tape Turing machine with new tape alphabet Γ^k, where k is the number of tracks.

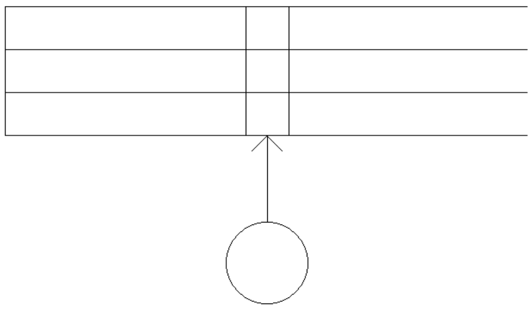

Figure 2: TM with multiple tracks

In the usual Turing machine, there is one tape infinite in *one* direction.

Suppose we want to have two-way infinite tape. It can be implemented using a two-track Turing machine, where the head read one tape at a time.

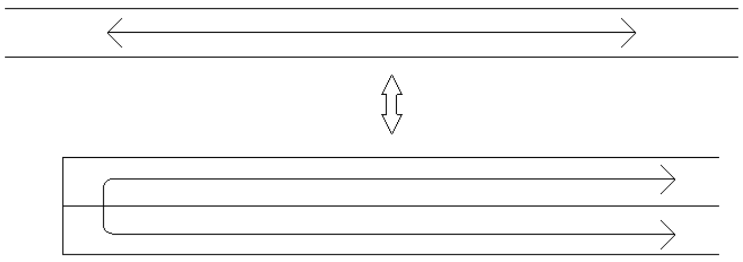
Figure 3: Two-way infinite tape TM

Suppose we want to implement a multitape TM as the following figure suggests. At each step, depending on states q_1, q_2, q_3, and input symbols

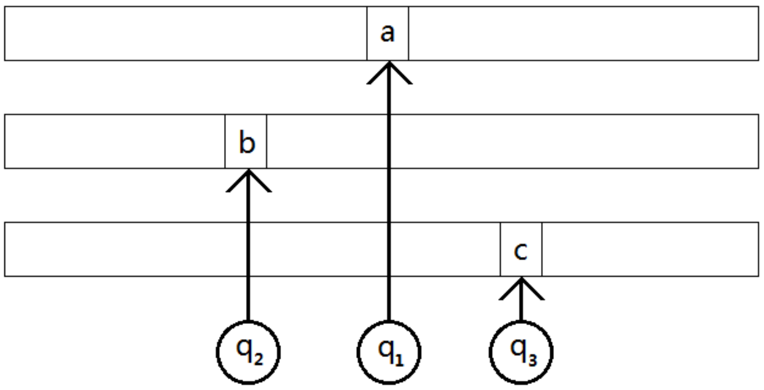

Figure 4: Multitape TM

a, b, c on each tape, the machine changes a to a', b to b', c to c', moves q_1 left or right, q_2 left or right, q_3 left or right, and changes state q_1 to q_1', q_2 to q_2', q_3 to q_3'.

This multitape Turing machine can be implemented using multitrack TM (2 tracks per tape), where the first track for each tape keeps track of the location of the head (marked by X, all others are blank). The new head has state Q^k, where k is the number of tapes, and a cache of size k, which stores the symbols each head points to. For each step of the old machine, the new multitrack TM first reads all symbols in each tape and put them in the cache, depending on current state (q_1, q_2, q_3), the new machine changes to a new state (q_1', q_2', q_3'), and then goes to each position under symbol X, changes the symbol, and moves X either left or right.

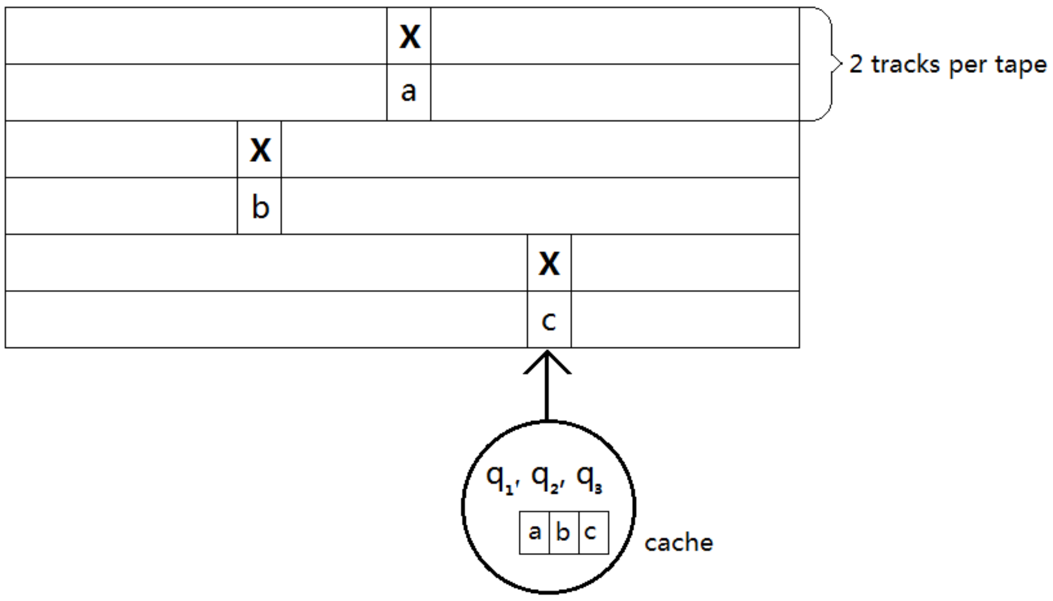

Figure 5: Simulate multitape TM by multitrack TM with cache

It is easy to see the cost of simulating each move is proportional to the distance between the leftmost and rightmost heads, and thus the blowup in time is quadratic.