Lesson 5: Alternating Turing machines

Theme: The notion of alternating Turing machine, the relation with deterministic Turing machine and the polynomial hierarchy.

1 Definition

A 1-tape *alternating Turing machine* (ATM) is a system $\mathcal{M} = \langle \Sigma, \Gamma, Q, U, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, \delta \rangle$, where each component is as follows.

- $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$ are the input and tape alphabets, respectively.
- Q is a finite set of states.
- $U \subseteq Q$ is a finite subset of Q.
- $q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}$ are the initial state, accepting state and rejecting state, respectively.
- $\delta \subseteq (Q - \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \times Q \times \Gamma \times \{\mathtt{Left}, \mathtt{Right}\}$.

Note that ATM is very much like NTM, except that it has one extra component U. The states in U are called *universal* states, and the states in $Q - U$ are called *existential* states.

The notions of *initial/halting/accepting/rejecting* configuration are defined similarly as in NTM/DTM. A configuration C is called *existential/universal* configuration, if the the state in C is an existential/universal state. The notion of "one step computation" $C \vdash C'$ for ATM is also similar to the one for DTM/NTM. When $C \vdash C'$, we say that C' is one of the next configuration of C (w.r.t. $\mathcal{M}$).

On input word w, *the run of $\mathcal{M}$ on w* is a *tree T* where each node in the tree is labelled with a configuration of $\mathcal{M}$ according to the following rules.

- The root node of T is labelled with the initial configuration of $\mathcal{M}$ on w.
- Every other node x in T is labelled as follows.

  If x is labelled with a configuration C and $C_1, \ldots, C_n$ are all the next configurations of C, then x has n children $y_1, \ldots, y_n$ labelled with $C_1, \ldots, C_n$, respectively.

Note that if x is labelled with C that does not have next configuration, then it is a leaf node, i.e., it does not have any children.

Let T be the run of $\mathcal{M}$ on w and let x be a node in T. We say that x *leads to acceptance*, if the following holds.

- x is a leaf node labelled with an accepting configuration.
- If x is labelled with an existential configuration, then one of its children leads to acceptance.
- If x is labelled with a universal configuration, then all of its children lead to acceptance.

We say that T is *accepting run*, if its root node leads to acceptance. The ATM $\mathcal{M}$ accepts w, if the run of $\mathcal{M}$ on w is accepting run. As before, $L(\mathcal{M}) \stackrel{\mathsf{def}}{=} \{w : \mathcal{M} \text{ accepts } w\}$.

Note that NTM is simply ATM where all the states are existential, and DTM is simply NTM where every configuration (except the accepting/rejecting configuration) has exactly one next configuration. The generalization of ATM to multiple tapes is straightforward.

2 Time and space complexity for ATM

Let $\mathcal{M}$ be a ATM, $w \in \Sigma^*$, $t \in \mathbb{N}$ and let $f : \mathbb{N} \to \mathbb{N}$ be a function.

- *$\mathcal{M}$ decides w in time t (or, in t steps)*, if the run of $\mathcal{M}$ on w has depth at most t.
- *$\mathcal{M}$ decides w in space t (or, uses t cells/space)*, if in the run of $\mathcal{M}$ on w, every node is labelled with configuration of length t.
- *$\mathcal{M}$ runs in time/space $O(f(n))$*, if there is $c > 0$ such that for sufficiently long word w, $\mathcal{M}$ decides w in time/space $c \cdot f(|w|)$.
- *$\mathcal{M}$ decides a language L in time/space $O(f(n))$*, if $\mathcal{M}$ runs in time/space $O(f(n))$ and $L(\mathcal{M}) = L$.
- $\textsc{Atime}[f(n)] \overset{\text{def}}{=} \{L : \text{there is ATM } \mathcal{M} \text{ that decides } L \text{ in time } O(f(n))\}$.
- $\textsc{Aspace}[f(n)] \overset{\text{def}}{=} \{L : \text{there is ATM } \mathcal{M} \text{ that decides } L \text{ in space } O(f(n))\}$.

Analoguous to the DTM/NTM, we can define the classes of languages accepted by ATM run in algorithmic/polynomial/exponential time/space.

$$\begin{aligned}
\mathbf{AL} &\overset{\text{def}}{=} \{L : \text{there is ATM } \mathcal{M} \text{ that decides } L \text{ in space } O(\log n)\} \\
\mathbf{AP} &\overset{\text{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Atime}[f(n)] \\
\mathbf{APSPACE} &\overset{\text{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Aspace}[f(n)] \\
\mathbf{AEXP} &\overset{\text{def}}{=} \bigcup_{f(n)=\mathsf{poly}(n)} \textsc{Atime}[2^{f(n)}]
\end{aligned}$$

The following lemma links time/space complexity classes for ATM with those for DTM.

Lemma 5.1 *Let $T : \mathbb{N} \to \mathbb{N}$ and $S : \mathbb{N} \to \mathbb{N}$ such that $T(n) \geqslant n$ and $S(n) \geqslant \log n$, for every n.*

(a) $\textsc{Atime}[T(n)] \subseteq \textsc{Dspace}[T(n)]$.

(b) $\textsc{Dspace}[S(n)] \subseteq \textsc{Atime}[S(n)^2]$.

(c) $\textsc{Aspace}[S(n)] \subseteq \textsc{Dtime}[2^{O(S(n))}]$.

(d) $\textsc{Dtime}[T(n)] \subseteq \textsc{Aspace}[\log T(n)]$.

Proof. (a) and (c) is by straightforward simulation of ATM with DTM. (b) is similar to the proof of Savitch's theorem. (d) is similar to the proof of Theorem 5.5 below, i.e., by viewing the computation of DTM as a boolean circuit. ∎

Theorem 5.2 (Chandra, Kozen, Stockmeyer 1981)

- $\mathbf{AL} = \mathbf{P}$.
- $\mathbf{AP} = \mathbf{PSPACE}$.
- $\mathbf{APSPACE} = \mathbf{EXP}$.
- $\mathbf{AEXP} = \mathbf{EXPSPACE}$.
- $\cdots$.

3 The polynomial hierarchy

For every integer $i \geqslant 1$, the class $\mathbf{\Sigma}_i^p$ is defined as follows. A language $L \subseteq \{0,1\}^*$ is in $\mathbf{\Sigma}_i^p$, if there is a polynomial $q(n)$ and a polynomial time DTM $\mathcal{M}$ such that for every $w \in \{0,1\}^*$, $w \in L$ if and only if the following holds.

$$\exists y_1 \in \{0,1\}^{q(|w|)} \ \forall y_2 \in \{0,1\}^{q(|w|)} \ \cdots \ Q y_i \in \{0,1\}^{q(|w|)} \ \mathcal{M} \text{ accepts } (w, y_1, \ldots, y_i) \qquad (1)$$

where $Q = \exists$, if i is odd and $Q = \forall$, if i is even.

The class $\mathbf{\Pi}_i^p$ is defined as above, but the sequence of quantifiers in (1) starts with $\forall$. Alternatively, it can also be defined as $\mathbf{\Pi}_i^p \stackrel{\text{def}}{=} \{\overline{L} : L \in \mathbf{\Sigma}_i^p\}$. Note that $\mathbf{NP} = \mathbf{\Sigma}_1^p$ and $\mathbf{coNP} = \mathbf{\Pi}_1^p$.

Remark 5.3 The class $\mathbf{\Sigma}_i^p$ can also be defined as follows. A language L is in $\mathbf{\Sigma}_i^p$, if there is a polynomial time ATM $\mathcal{M}$ that decides L such that for every input word $w \in \{0,1\}^*$, the run of $\mathcal{M}$ on w can be divided into i layers. Each layer consists of nodes of the same depth in the run. (Recall that the run of an ATM is a tree.) In the first layer all nodes are labeled with existential configurations, in the second layer with universal configurations, and so on. It is not difficult to show that this definition is equivalent to the one above.

The *polynomial time hierarchy* (or, in short, *polynomial hierarchy*) is defined as the following class.

$$\mathbf{PH} \stackrel{\text{def}}{=} \bigcup_{i=1}^{\infty} \mathbf{\Sigma}_i^p$$

Note that $\mathbf{PH} \subseteq \mathbf{PSPACE}$.

It is conjectured that $\mathbf{\Sigma}_1^p \subsetneq \mathbf{\Sigma}_2^p \subsetneq \mathbf{\Sigma}_3^p \subsetneq \cdots$. In this case, we say that *the polynomial hierarchy does not collapse.* We say that *the polynomial hierarchy collapses*, if there is i such that $\mathbf{PH} = \mathbf{\Sigma}_i^p$, in which case we also say that *the polynomial hierarchy collapses to level* i.

We define the notion of hardness and completeness for each $\mathbf{\Sigma}_i^p$ as follows. For $i \geqslant 1$, a language K is $\mathbf{\Sigma}_i^p$*-hard*, if for every $L \in \mathbf{\Sigma}_i^p$, $L \leqslant_p K$. It is $\mathbf{\Sigma}_i^p$*-complete*, if it is in $\mathbf{\Sigma}_i^p$ and it is $\mathbf{\Sigma}_i^p$-hard. The same notion can be defined analogously for $\mathbf{PH}$ and each $\mathbf{\Pi}_i^p$.

Define the language Σ_i-SAT as consisting of true QBF of the form:

$$\exists \bar{x}_1 \ \forall \bar{x}_2 \ \cdots \ Q \bar{x}_i \ \varphi(\bar{x}_1, \ldots, \bar{x}_i)$$

where $\varphi(\bar{x}_1, \ldots, \bar{x}_i)$ is quantifier-free Boolean formula and $Q = \exists$, if i is odd, and $Q = \forall$, if i is even. Here $\bar{x}_1, \ldots, \bar{x}_i$ are all vectors of boolean variables. In other words, Σ_i-SAT is a subset of TQBF where the number of quantifier alternation is limited to $(i-1)$. The language Π_i-SAT is defined analogously with the starting quantifiers being $\forall$.

Theorem 5.4

- *For every* $i \geqslant 1$, Σ_i-SAT *is* $\mathbf{\Sigma}_i^p$*-complete and* Π_i-SAT *is* $\mathbf{\Pi}_i^p$*-complete.*
- *If* $\mathbf{\Sigma}_i^p = \mathbf{\Pi}_i^p$ *for some* $i \geqslant 1$*, then the polynomial hierarchy collapses.*
- *If there is language that is* $\mathbf{PH}$*-complete, then the polynomial hierarchy collapses.*

Appendix

A P-complete languages

Boolean circuits. Let $n \in \mathbb{N}$, where $n \geqslant 1$. An n-input *Boolean circuit* C is a directed acyclic graph with n *source* vertices (i.e., vertices with no incoming edges) and 1 *sink* vertex (i.e., vertex with no outgoing edge).

The source vertices are labelled with $x_1, \ldots, x_n$. The non-source vertices, called *gates*, are labelled with one of $\wedge, \vee, \neg$. The vertices labelled with $\wedge$ and $\vee$ have two incoming edges, whereas the vertices labelled with $\neg$ have one incoming edge. The *size* of C, denoted by $|C|$, is the number of vertices in C.

On input $w = x_1 \cdots x_n$, where each $x_i \in \{0,1\}$, we write $C(w)$ to denote the output of C on w, where $\wedge, \vee, \neg$ are interpreted as "and," "or" and "negation," respectively and 0 and 1 as false and true, respectively.

(Boolean) straight line programs. It is sometimes more convenient to view a boolean circuit a straight line program. The following is an example of straight line program, where the input is $w = x_1 \cdots x_n$.

$$\begin{aligned}
&1\text{:}\quad p_1 := x_1 \wedge x_3.\\
&2\text{:}\quad p_2 := \neg x_4.\\
&3\text{:}\quad p_3 := p_1 \vee p_2.\\
&\vdots\\
&\ell\text{:}\quad p_\ell := p_i \wedge p_j.
\end{aligned}$$

Intuitively, straight line programs are programs without **if** branch and **while** loop, hence, the name "straight line" programs. It is assumed that such program always outputs the value in the variable in the last line. In our example above, it outputs the value of variable p_ℓ.

Define the following problem.

CIRCUIT-EVAL	
Input:	An n input boolean circuit C and $w \in \{0,1\}^n$.
Task:	Output $C(w)$.

It can also be defined as the language CIRCUIT-EVAL $\overset{\text{def}}{=} \{(C,w) : C(w) = 1\}$.

For our proof of Theorem 5.5 below, it is also convenient to assume that vertices labelled with $\wedge$ and $\vee$ can have more than 2 incoming edges.

Theorem 5.5 CIRCUIT-EVAL *is* **P***-complete via log-space reductions.*

Proof. Follows the reduction for the **NP**-completeness of SAT. ∎