Chapter 1

Computational Models

1.1. Why Turing Machines?

There are two reasons for the use of Turing machines as a model of computation. The first is that they provide a compromise between the mathematical simplicity of models such as the lambda calculus or recursive functions, and the computational realism one would find in random access machines or Pascal programs. (Note, for instance, that the entire semantics of Turing machines will be given in the short Section 1.2.) The second reason is that the choice actually makes little difference, since all reasonable models are equivalent in the domains of computability and coarse analysis.

1.2. Definition of Alternating Turing Machines

Alternating Turing machines were introduced by Chandra, Kozen, and Stockmeyer [3]. For simplicity, the particular variant defined here will have one tape and be used for set recognition. It should be noted that the definitions can be extended easily to multiple tapes or function computation.

Definition 1.1: An *alternating Turing machine* is an 8-tuple $M = (Q, E, A, F, \Gamma, \Sigma, q_0, \delta)$ satisfying the following properties:

- Q is a finite set (the "states"), partitioned into 3 subsets:
  1. E (the "existential states"),
  2. A (the "universal states"), and
  3. F (the "final states").
- Γ is a finite set (the "worktape alphabet").
- $\Sigma \subseteq \Gamma$. (Σ is the "input alphabet".)
- $\not\flat \in \Gamma - \Sigma$. ($\not\flat$ is the "blank symbol".)
- $Q \cap \Gamma = \emptyset$.

- $q_0 \in Q$. (q_0 is the "start state".)
- $\delta : (Q - F) \times \Gamma \to 2^{Q \times (\Gamma - \{\not b\}) \times \{L,R\}}$. (δ is the "transition function".)

The next two definitions specify what the computation steps of a Turing machine look like.

Definition 1.2: A *configuration* of an alternating Turing machine M is a string uqv, where $q \in Q$, and $u, v \in \Gamma^*$.

(The intuition is that M is in state q, $uv \in \Gamma^*$ is the nonblank content of the tape, and M's head is reading the first symbol of v, or $\not b$ if $v = \epsilon$, the empty string. In general, the configurations uqv, $uqv\not b$, and $\not buqv$ are considered equivalent, so we can always assume that $u \neq \epsilon$ and $v \neq \epsilon$.)

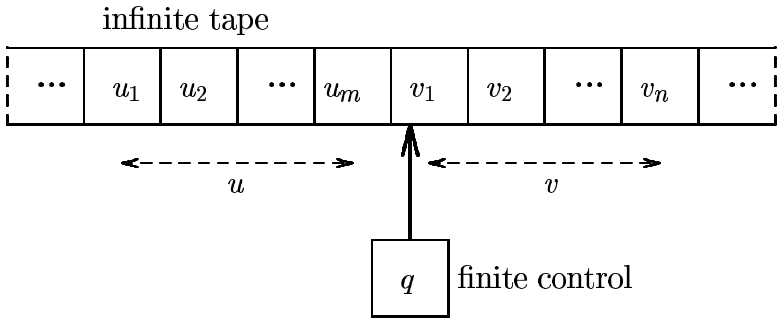

Figure 1.1: Visualizing an Alternating Turing Machine in Configuration uqv

Definition 1.3: The *one-step transition relation* $\vdash_M$ for an alternating Turing machine M is defined as follows. For any $u, v \in \Gamma^*$, $q, q' \in Q$, and $a, b, c \in \Gamma$,

$$uqav \vdash_M ucq'v \quad \text{if and only if} \quad (q', c, R) \in \delta(q, a), \text{ and}$$

$$ubqav \vdash_M uq'bcv \quad \text{if and only if} \quad (q', c, L) \in \delta(q, a).$$

Definition 1.4: If P and P' are configurations of M and $P \vdash_M P'$, P' is called an *immediate successor* of P.

The remaining definitions specify how a Turing machine starts and finishes.

Definition 1.5: A configuration uqv is *halting* if and only if there is no configuration P such that $uqv \vdash_M P$.

(This can happen if $q \in F$, of if $\delta(q, a) = \emptyset$, where a is the first symbol of v.)

Definition 1.6:

- If $q \in F$, or $q \in A$ and uqv is halting, then uqv is a *final* configuration.
- If $q \in E$, then uqv is an *existential* configuration.
- If $q \in A$, then uqv is a *universal* configuration.

Definition 1.7: A configuration P is said to be *accepting* if and only if one of the following three conditions holds:

1. P is a final configuration, or
2. P is an existential configuration and $(\exists Q)((P \vdash_M Q)$ & (Q is accepting)), or
3. P is a universal configuration and $(\forall Q)((P \vdash_M Q) \Rightarrow (Q$ is accepting)).

Definition 1.8: M *accepts* the input string $x \in \Sigma^*$ if and only if $q_0 x$ is an accepting configuration.

Definition 1.9: The *language* accepted by M is $L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$.

1.3. Nondeterministic and Deterministic Turing Machines

Given the definition of alternating Turing machines, it is straightforward to define the more familiar deterministic and nondeterministic Turing machines:

Definition 1.10: A *nondeterministic Turing machine* is an alternating Turing machine for which $A = \emptyset$.

Definition 1.11: A *deterministic Turing machine* is a nondeterministic Turing machine for which $|\delta(q, a)| \leq 1$ for all $q \in Q - F$ and $a \in \Gamma$.

Our alternating Turing machines are defined to have one tape, but the definition can be extended to k tapes. In this case, configurations become k-tuples $(u_1 q v_1, u_2 q v_2, \ldots, u_k q v_k)$, and the transition function, as an example of the generalization, becomes $\delta : (Q - F) \times \Gamma^k \rightarrow 2^{Q \times (\Gamma - \{\not b\})^k \times \{L,N,R\}^k}$, where N denotes no move of the tape head on the corresponding tape.

1.4. Examples of Turing Machines

Example 1.12: Given an undirected graph $G = (V, E)$, a *Hamiltonian cycle* is a cycle that passes through each vertex exactly once. Assume $|V| = v$. The nondeterministic Turing machine M described below, given an encoding of such a graph, determines whether it has a Hamiltonian

cycle. M nondeterministically chooses and records on its tape an ordered list $(u_0, u_1, \ldots, u_{v-1})$ of v of the vertices. If any vertex appears twice on this list, M rejects (i.e., halts in a nonfinal configuration). If there is an i such that $\{u_i, u_{(i+1) \bmod v}\} \notin E$, M rejects. Otherwise, M accepts (i.e., halts in a final configuration).

Example 1.13: As an example of an alternating Turing machine, consider the problem of determining if white has a winning strategy in chess. An alternating Turing machine M maintains an encoding of the current board position on its tape and, beginning from the initial position, alternates between

- existentially choosing white's next move from among the legal alternatives and
- universally choosing black's next move from among the legal alternatives,

until the game reaches a terminal configuration. M then accepts if and only if white has won in this configuration.

In words, what M is doing is checking that "there is a first move by white such that, for all second moves by black, there is a third move by white such that ... white wins."

1.5. Computation Trees and Accepting Subtrees

Definition 1.14: Let M be an alternating Turing machine, and P be a configuration of M. The *P-computation tree of M* is a tree (possibly infinite) with nodes labeled by configurations of M such that

1. the root is labeled P, and
2. each node labeled Q has a child labeled R for each R such that $Q \vdash_M R$.

Definition 1.15: Let M be an alternating Turing machine, and x an input. The *computation tree of M on x* is the (q_0x)-computation tree of M.

Example 1.16: The computation tree of a deterministic Turing machine on a given input is a (possibly infinite) simple path. Since, in each configuration, there is at most one possible choice of the next configuration, the computation tree never branches. Note that the computation tree is finite in this case if and only if the Turing machine halts. Notice also that the same configuration may label two different nodes of the computation tree, but in the case of a deterministic Turing machine that would mean the machine never halts.

Definition 1.17: Let M be an alternating Turing machine, and P a configuration of M. An *accepting P-subtree A of M* is a subtree of the P-computation tree C of M, with the following properties:

1. A includes the root of C,

2. for every node v of A labeled by an existential configuration, v has one of its children from C,

3. for every node v of A labeled by a universal configuration, v has all of its children from C,

4. A has no infinite paths, and

5. all of A's leaves are labeled by final configurations.

Definition 1.18: Let M be an alternating Turing machine, and x an input. An *accepting subtree* of M on x is an accepting (q_0x)-subtree of M.

Example 1.19: An accepting subtree of a nondeterministic Turing machine is a simple (finite) path whose internal nodes are labeled by existential configurations and whose single leaf is labeled by a final configuration. In Example 1.12, for instance, this path would correspond to the computation that correctly guesses (and verifies) a Hamiltonian cycle in the input graph.

The following theorem relates the definition of acceptance to the notion of an accepting subtree.

Theorem 1.20: M accepts x if and only if there is an accepting subtree of M on x.

Proof: Left as an exercise. (Hint: Using induction, prove the generalization that P is an accepting configuration if and only if there is an accepting P-subtree of M.) □

Definition 1.21: A set is *recursively enumerable* if and only if it is accepted by some deterministic Turing machine.

Theorem 1.22: Nondeterministic Turing machines and alternating Turing machines accept exactly the recursively enumerable sets.

Proof: That they accept at least the recursively enumerable sets follows from the fact that deterministic Turing machines are a special case of nondeterministic Turing machines and alternating Turing machines. The other direction is a corollary of results that will be proved in Chapter 4. These results (Theorems 4.7 and 4.11) show not only that alternating Turing machines can be simulated by deterministic Turing machines, but how efficient that simulation is. □

1.6. Exercises

1. (a) Give definitions analogous to those in Section 1.2 for a k-tape alternating Turing machine.
   (b) Prove that any k-tape alternating Turing machine can be simulated by a one-tape alternating Turing machine.

2. (a) Describe a nondeterministic Turing machine that, given the encoding of a directed graph G and two distinguished vertices s and t, accepts if and only if there is a path in G from s to t.
   (b) Describe a deterministic Turing machine that accepts the same language.

3. Prove Theorem 1.20.