The Turing Model of Computation

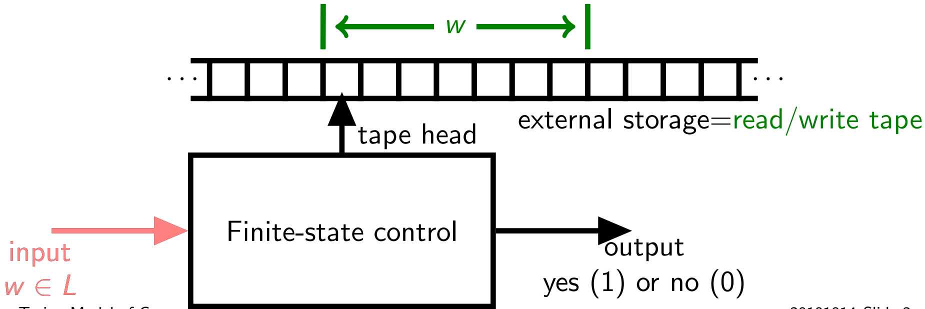
5DV037 — Fundamentals of Computer Science

Umeå University
Department of Computing Science

Stephen J. Hegner
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

The Idea of a Turing Machine

- The Turing machine is an abstract model of a general computer.
- It is named for the British mathematician Alan Turing (1912-1954).
- In this model, the auxiliary storage is both readable and writable in a general way.
- The tape is taken to be infinite at both ends
  - ... although many authors use only a semi-infinite tape.
- The input is typically encoded as an initial tape configuration, rather than a separate input stream.



Formal Definition of a Deterministic Turing Machine

- In this context, deterministic machines will be considered first.
- Nondeterministic machines will be considered case later.
- A *deterministic Turing machine* or *DTM* is a seven tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$$

in which
  - Q is finite set of *states*;
  - Σ is an alphabet, called the *input alphabet*;
  - Γ is an alphabet, called the *tape alphabet*;
  - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ is a partial function, the *state-transition function*;
  - $q_0 \in Q$ is the *initial state*;
  - $\sqcup \in \Gamma \setminus \Sigma$ is the *blank symbol*;
  - $F \subseteq Q$ is the set of *final* or *accepting states*.

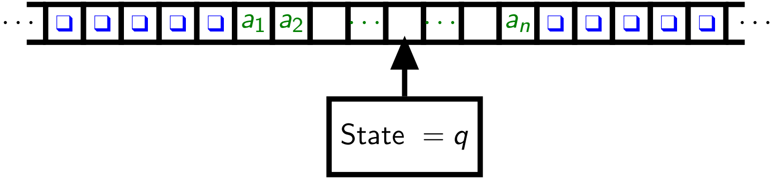
The Operation of a DTM

$$\underset{\text{Current state}}{Q} \times \underset{\text{Tape symbol to be processed}}{\Gamma} \xrightarrow{\delta} \underset{\text{New state}}{Q} \times \underset{\text{Replacement tape symbol}}{\Gamma} \times \underset{\text{Move direction of tape head}}{\{L, R, S\}}$$

- The new symbol replaces the current symbol on the tape.
- The directions are encoded as follows:
  - L = move one square to the left;
  - R = move one square to the right;
  - S = remain on the same tape square.
- The textbook does not include S, but it is a very convenient extension.
- The function δ may be partial (not defined for all inputs).
- But it is deterministic (at most one move from each configuration).

The Contents of the Tape of a DTM

- In a DTM, it is always the case that all but finitely many of the tape squares contain the blank symbol ❑.
- Thus, the tape contents may be represented as a string which is either empty or else of the form $a_1 a_2 \ldots a_n$, in which:
  - every symbol to the left of a_1 is ❑;
  - every symbol to the right of a_n is ❑;
  - $a_1 \neq$ ❑ and $a_n \neq$ ❑, (but may be at the same tape position).
- The intermediate elements $a_2 \ldots a_{n-1}$ may be ❑.



The Form of an ID for a DTM

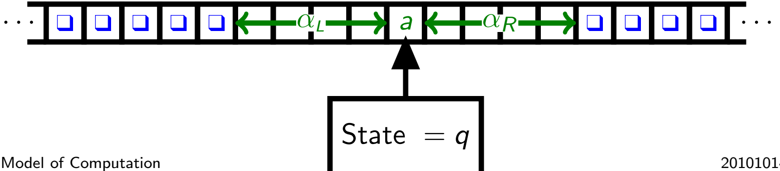
- To represent the ID of a DTM, in addition to the state and the tape contents, it is necessary to represent the position of the tape head.
- The idea is to represent the tape contents as a triple

$$\langle \alpha_L, a, \alpha_R \rangle$$

  in which:
  - $\alpha_L \in \mathcal{L}(\lambda + (\Gamma \setminus \{❑\}) \cdot \Gamma^*)$, $\alpha_R \in \mathcal{L}(\lambda + \Gamma^* \cdot (\Gamma \setminus \{❑\}))$ as illustrated.
  - $a \in \Gamma =$ contents of the current tape square.
- An *ID* for $M = (Q, \Sigma, \Gamma, \delta, q_0, ❑, F)$ is then a quadruple

$$\langle q, \alpha_L, a, \alpha_R \rangle$$

  in which $q \in Q$ and $\langle \alpha_L, a, \alpha_R \rangle$ is as above.
- $\mathrm{ID}\langle M \rangle$ denotes the set of all IDs of M.



The Representation of IDs in the Textbook

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$.

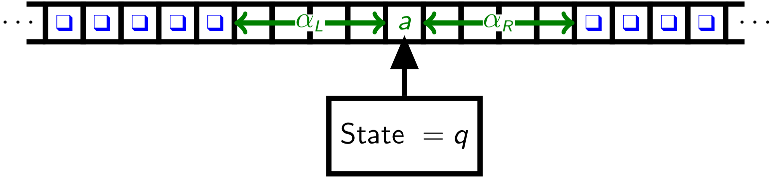
- In the textbook, the ID

$$\langle q, \alpha_L, a, \alpha_R \rangle$$

  is written as

$$\alpha_L \; q \; a\alpha_R$$

- Formally, the representations are completely equivalent.
- However, the textbook representation requires that the names of states be disjoint from those of tape symbols, and be clearly identified.
- This can become confusing, and so will not be used in these slides.



The Move Relation for at DTM

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$.

- The move relation $\vdash_M$ is defined in a natural way.
- Let $\langle q, a_1 \dots a_m, a, b_1, \dots b_n \rangle$ be an ID for M.
  - $\langle q, a_1 \dots a_m, a, b_1 \dots b_n \rangle \vdash_M \langle q', a_1 \dots a_{m-1}, a_m, a' b_1 \dots b_n \rangle$ iff $\delta(q, a) = (q', a', L)$.
  - $\langle q, a_1 \dots a_m, a, b_1 \dots b_n \rangle \vdash_M \langle q', a_1 \dots a_m a', b_1, b_2 \dots b_n \rangle$ iff $\delta(q, a) = (q', a', R)$.
  - $\langle q, a_1 \dots a_m, a, b_1 \dots b_n \rangle \vdash_M \langle q', a_1 \dots a_m, a', b_1 \dots b_n \rangle$ iff $\delta(q, a) = (q', a', S)$.
- If $a_1 \dots a_m$ or $b_1 \dots b_n$ is empty, fill in with the blank symbol:
  - $\langle q, \lambda, a, b_1 \dots b_n \rangle \vdash_M \langle q', \lambda, \sqcup, a' b_1 \dots b_n \rangle$ iff $\delta(q, a) = (q', a', L)$.
  - $\langle q, a_1 \dots a_m, a, \lambda \rangle \vdash_M \langle q', a_1 \dots a_m a', \sqcup, \lambda \rangle$ iff $\delta(q, a) = (q', a', R)$.
- $\vdash_M^*$ is defined to be the reflexive and transitive closure of $\vdash_M$.
- For a DTM, both $\vdash_M$ and $\vdash_M^*$ are partial functions.

Computations and Halt States

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$.

- $\langle q, \alpha_L, a, \alpha_R \rangle \in \mathrm{ID}\langle M \rangle$ is a *halt configuration* if $\delta(q, a)$ is undefined.
- In other words, a halt configuration is one which does not admit any further moves.
- $\langle q, \alpha_L, a, \alpha_R \rangle \in \mathrm{ID}\langle M \rangle$ is an *accepting configuration* if $q \in F$.

Computing with a DTM:

- The key idea is to run M until it reaches a halt configuration.
- Once M halts, the result of the computation is encoded on the tape and/or the final state.

- Define the *global transition function* of M to be the partial function $\hat{\delta}^*_M : \mathrm{ID}\langle M \rangle \to \mathrm{ID}\langle M \rangle$ with $\hat{\delta}^*_M(D) = D'$ iff
  - $D \vdash^*_M D'$, and
  - D' is a halt configuration for M.
- Note that, $\hat{\delta}^*_M(D)$ is undefined iff the computation starting with configuration D runs forever.

Initial Configurations for and Acceptance by DTMs

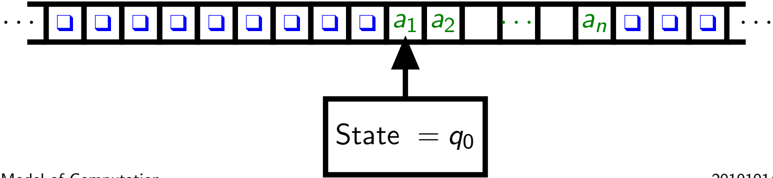
Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$.

- Recall that in a DTM, the input is encoded on the tape.
- Let $\alpha \in \Sigma^*$. The *input configuration for $\alpha = a_1 a_2 \ldots a_n$*, denoted $\mathcal{I}\langle M, \alpha\rangle$, is

$$\langle q_0, \lambda, \mathrm{First}\langle\alpha\rangle, \mathrm{Rest}\langle\alpha\rangle\rangle = \langle q_0, \lambda, a_1, a_2 \ldots a_n\rangle$$

- The *language accepted by M*, denoted $\mathcal{L}(M)$, is the set of all $\alpha \in \Sigma^*$ such that
  - $\hat{\delta}^*_M(\mathcal{I}\langle M, \alpha\rangle)$ is defined, and
  - it is an accepting configuration; *i.e.*,

$$\hat{\delta}^*_M(\mathcal{I}\langle M, \alpha\rangle) = \langle q, \beta_1, b, \beta_2\rangle \text{ for some } q \in F.$$



Languages Accepted by DTMs

- The language $L \subseteq \Sigma^*$ is called *Turing acceptable* or *Turing recognizable* or *semidecidable* if there is a DTM M with $\mathcal{L}(M) = L$.

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$.

- Note that $\alpha \in \mathcal{L}(M)$ iff
  - $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is defined, and
  - The state of $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is accepting.
- Thus, $\alpha \notin \mathcal{L}(M)$ iff
  - $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is not defined, or
  - $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is defined, but its state is not accepting.
- In other words, a DTM can reject a string by failing to halt.
- This notion forms a major part of what will be studied in this part of the course.
- Specifically, it will be shown that it is not possible, in general, to determine whether the machine will halt or not.
- This is the so-called *halting problem*.

Deciders and Recursive Languages

- A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ is called a *decider* if $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is defined for every $\alpha \in \Sigma^*$.

- In other words, M is a decider if its computation on every input string halts.

- It is guaranteed never to run forever on any input string.

- A language $L \subseteq \Sigma^*$ is *Turing decidable* or *decidable* or *recursive* if there is a decider M with $\mathcal{L}(M) = L$.

Amazing Fact (Turing): There exist languages which are Turing acceptable but not Turing decidable. $\Box$

- Establishing this fact, and understanding its consequences, will form the focus of study for the next few weeks.

The Relationship between Deciders and Accepters

Observation: If the language $L \subseteq \Sigma^*$ is Turing decidable, then so too is its complement $\overline{L} = \Sigma^* \setminus L$.

Proof: If $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ be a decider for L, then $M' = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, Q \setminus F)$ is a decider for $\overline{L}$.

- It halts when one of these two emulations does.
- Thus, if L is Turing decidable, then both L and $\overline{L}$ are Turing acceptable.
- The converse is also the case.

Theorem: The language $L \subseteq \Sigma^*$ is Turing decidable iff both L and $\overline{L}$ are Turing acceptable.

Proof: The idea is to build a DTM M'' which emulates the behavior of both an accepter M_L for L and an accepter $M_{\overline{L}}$ for $\overline{L}$.

- The machine "timeshares" the two emulations; one must eventually halt..
- To build such a machine is tedious but straightforward.
- It will be shown later in these slides that in lieu of a formal proof, appeal to a universal principle (the Church-Turing thesis) may be made. $\square$

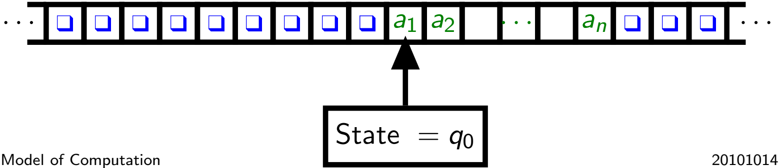
Computation of Functions by DTMs

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$.

- Let $\alpha \in \Sigma^*$. An *output configuration for $\alpha = a_1 a_2 \ldots a_n$* is of the form

$$\langle q, \alpha', \mathrm{First}\langle\alpha\rangle, \mathrm{Rest}\langle\alpha\rangle\rangle = \begin{cases} \langle q, \alpha', a_1, a_2 \ldots a_n\rangle & \text{if } \alpha \neq \lambda \\ \langle q, \alpha', \square, \lambda\rangle & \text{if } \alpha = \lambda \end{cases}$$

  for some $q \in F$ and $\alpha' \in ((\Gamma \setminus \square)^* \cdot \{\square\}) \cup \{\lambda\}$.
- Thus, to the right of the tape head, an output configuration look just like an input configuration, save that the state must be in F.
- The string to the left of the tape head must be λ or end with a blank, but otherwise there is no restriction.
- Roughly, the machine computes $\beta = b_1 b_2 \ldots b_m$ from input α if $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is an output configuration for β.
- The textbook requires $\alpha' = \lambda$, but this generalization will prove useful.



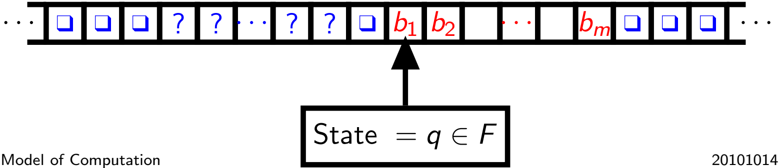
Computation of Functions by DTMs

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$.

- Let $\alpha \in \Sigma^*$. An *output configuration for $\alpha = a_1 a_2 \ldots a_n$* is of the form

$$\langle q, \alpha', \mathrm{First}\langle\alpha\rangle, \mathrm{Rest}\langle\alpha\rangle\rangle = \begin{cases} \langle q, \alpha', a_1, a_2 \ldots a_n\rangle & \text{if } \alpha \neq \lambda \\ \langle q, \alpha', \square, \lambda\rangle & \text{if } \alpha = \lambda \end{cases}$$

  for some $q \in F$ and $\alpha' \in ((\Gamma \setminus \square)^* \cdot \{\square\}) \cup \{\lambda\}$.
- Thus, to the right of the tape head, an output configuration look just like an input configuration, save that the state must be in F.
- The string to the left of the tape head must be λ or end with a blank, but otherwise there is no restriction.
- Roughly, the machine computes $\beta = b_1 b_2 \ldots b_m$ from input α if $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is an output configuration for β.
- The textbook requires $\alpha' = \lambda$, but this generalization will prove useful.

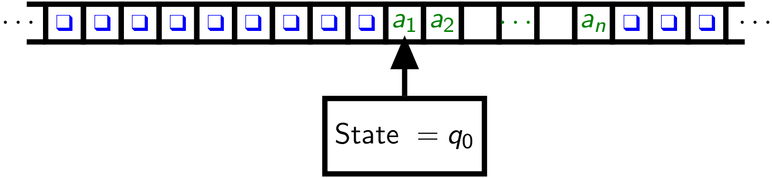


Computations of Functions by DTMs — 2

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$.

- The function f_M computed by M is defined iff for every input configuration $\mathcal{I}\langle M, \alpha \rangle$, either
  - $\hat{\delta}^*(\mathcal{I}\langle M, \alpha \rangle)$ is some output configuration for a string β; or else
  - $\hat{\delta}^*(\mathcal{I}\langle M, \alpha \rangle)$ is undefined.
- In this case, $f_M : \Sigma^* \to \Sigma^*$ defined by

$$\alpha \mapsto \begin{cases} \beta & \text{if } \hat{\delta}^*(\mathcal{I}\langle M, \alpha \rangle) \text{ is an output configuration for } \beta \\ \text{undefined} & \text{otherwise} \end{cases}$$

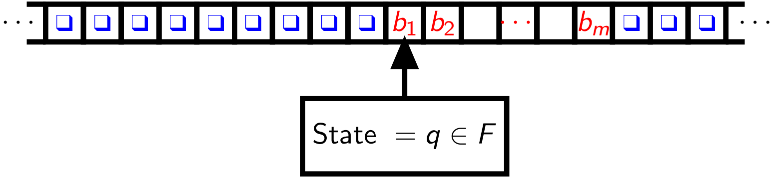


Computations of Functions by DTMs — 2

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$.

- The function f_M computed by M is defined iff for every input configuration $\mathcal{I}\langle M, \alpha\rangle$, either
  - $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is some output configuration for a string β; or else
  - $\hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle)$ is undefined.
- In this case, $f_M : \Sigma^* \to \Sigma^*$ defined by

$$\alpha \mapsto \begin{cases} \beta & \text{if } \hat{\delta}^*(\mathcal{I}\langle M, \alpha\rangle) \text{ is an output configuration for } \beta \\ \text{undefined} & \text{otherwise} \end{cases}$$



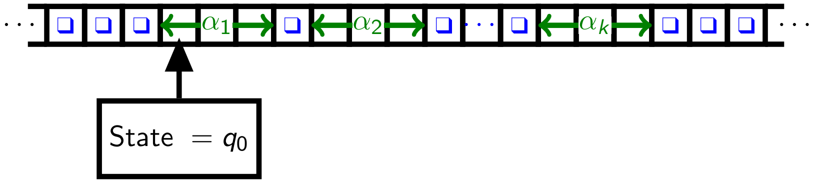
Computation of Multi-Argument Functions by DTMs

Context: A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$.

- Multi-argument input configurations are specified in a natural way.
- Just put the arguments on the input tape, separated by blanks.
- If the input is

$$(\alpha_1, \alpha_2, \ldots, \alpha_k) \in \Sigma^* \times \Sigma^* \times \ldots \times \Sigma^*$$

  then the input configuration is as illustrated below.
- $f_M^{(k)} : (\Sigma^*)^k \to \Sigma^*$ is defined in the obvious way.
- Multiple outputs are formulated similarly.



The DTM as a Model of Computation

- The concept of a Turing machine was developed during the 1930's, by a mathematician, before digital computers were a reality.
- It is conceptually simple, although very tedious, to program a DTM.
- Even simple tasks which are trivial to describe in a modern programming language become very tedious chores with a DTM.

Question: What is the utility of the DTM, then?

Answer: It is the tool for establishing certain theoretical results.

- Turing machines are very useful in the study of complexity in particular because:
  - They admit a very simple definition of what a single step in a computation is.
  - They admit a natural model of nondeterminism, which is a central idea in modern complexity theory.
- In any case, in this course, the programming of DTMs will not be a focus.
- This choice of model is not as crucial as it might appear because of the *Church-Turing thesis*.

The Church-Turing Thesis

Question: Is there an upper limit on what a computer can do, without regard for how efficiently it can do it?

Answer: The *Church-Turing Thesis* or just *Turing Thesis* says that there is, and that this limit is defined by the DTM.

- It is not something which can be proven, because there are infinitely many different models of computation.
- However, this thesis is supported by reductions of many hundreds (if not thousands) of distinct models of computation.
- This includes:
  - All sorts of programming languages.
  - All sorts of nondeterministic models.
  - Many specialized models.
- It has been shown that none of these models is more powerful than the DTM.

An Application of the Church-Turing Thesis

- Recall the earlier claim:

Theorem: The language $L \subseteq \Sigma^*$ is Turing decidable iff both L and $\overline{L}$ are Turing acceptable.

Proof: The idea is to build a DTM M''which emulates the behavior of both an accepter M_L for L and an accepter $M_{\overline{L}}$ for $\overline{L}$.

- The machine "timeshares" the two emulations.
- To build such a machine is tedious but straightforward.
- Rather than spelling out in detail how to build the emulating machine, it is possible to invoke the Church-Turing thesis.
- It is certainly possible to build such an emulator in a modern programming language.
- Thus, it must be possible to build a DTM which does the same thing. $\Box$

Universal Models of Computation

- Call a computational model *universal* if it is equivalent in power to the DTM.
  - Also called *Turing equivalent*.
- Virtually all modern programming languages are universal models..
  - modulo idealization to no bound on values for variables.
- NPDAs and FAs are not universal.

Why? The language $\{a^k b^k c^k \mid k \in \mathbb{N}\}$ is not a CFL (and hence not acceptable by any NPDA), but it is clearly possible to write a program in C to accept it.

- In this course, two other models of universal computation will be considered:
  - Nondeterministic Turing machines
    - because of their importance in the study of complexity theory.
  - A simple language of while programs
    - because it provides an simple alternative notion of universal computation which is much more familiar to computer scientists.

Nondeterministic Turing Machines

- A *nondeterministic Turing machine* (*NDTM*) is defined exactly as a DTM, save that the transition function has the structure:

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

  providing a finite set of alternatives at each point.

- This function is usually taken to be total, since the lack of a transition may be modelled via the empty set.

- The DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ may be modelled as an NDTM $M' = (Q, \Sigma, \Gamma, \delta', q_0, \sqcup, F)$ by defining

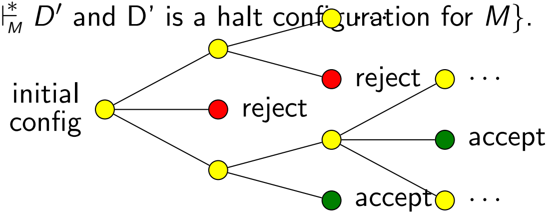
$$\delta'(q,a) = \begin{cases} \{\delta(q,a)\} & \text{if } \delta(q,a) \text{ is defined} \\ \emptyset & \text{if } \delta(q,a) \text{ is not defined} \end{cases}$$

- The move relation $\vdash_M$ and its transitive closure $\vdash_M$ are defined as in the deterministic case, but they are no longer functions.

Global Transition and Acceptance for NDTMs

Context: An NDTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$.

- Define the *global transition function* of M to be the function $\hat{\delta}^*_M : \mathrm{ID}\langle M\rangle \to 2^{\mathrm{ID}\langle M\rangle}$ with $\hat{\delta}^*_M(D) = \{D' \in \mathrm{ID}\langle M\rangle \mid D \vdash^*_M D'$ and D' is a halt configuration for $M\}$.



- A string $\alpha \in \Sigma^*$ is accepted by M if some computation from $\mathcal{I}\langle M, \alpha\rangle$ leads to a halt in an accepting state.

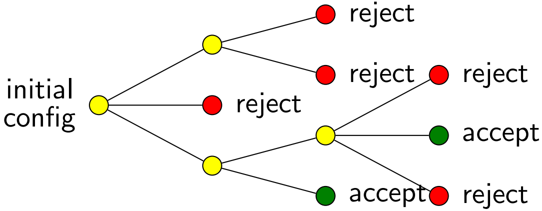
- $\mathcal{L}(M) = \{\alpha \in \Sigma^* \mid \hat{\delta}^*_M(\mathcal{I}\langle M, \alpha\rangle)$ contains an accepting configuration $\}$.
- Note the asymmetry between acceptance and rejection, as in the case of NFAs and NPDAs.

NDTMs as Deciders

Context: An NDTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$.

- The NDTM M is a *decider* if no infinite computations from any input configuration is possible.

- More precisely, M is a decider if for every $\alpha \in \Sigma^*$, there is an $N \in \mathbb{N}$ such that every computation $\mathcal{I}\langle M, \alpha \rangle \vdash_M D_1 \vdash_M D_2 \ldots \vdash_M D_k$ has $k \leq N$.



Theorem (equivalence of NDTMs and DTMs): Let $L \in \Sigma^*$.

(a) If $L = \mathcal{L}(M)$ for some NDTM M, then $L = \mathcal{L}(M')$ for some DTM M'.

(b) In (a), if M is a decider, then M' may be chosen to be a decider as well. $\square$

Computation of Functions by NDTMs

- The computation of a function by a DTM was defined in terms of input and output configurations.
- For a given input configuration $\mathcal{I}\langle M, \alpha \rangle$, if the machine halts in an output configuration, the string associated with that configuration is the output value.
- This idea does not extend easily to NDTMs, because there may be many distinct final configurations.
- Thus, the notion of a NDTM is used primarily with decision problems, which may be answered with "yes" or "no".
- The central problem of this form is the acceptance of a language.

An Abstract Formulation of the Notion of Algorithm

- A DTM M which halts for all inputs "of interest" defines an algorithm.
- If $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ is a decider, then the inputs of interest are initial configurations of strings in Σ^*.
- A *decision problem* on Σ^* is defined by a total function $f : \Sigma^* \to \{0, 1\}$.
  - If the answer to input $\alpha \in \Sigma^*$ is true, then $f(\alpha) = 1$.
  - If the answer to input $\alpha \in \Sigma^*$ is false, then $f(\alpha) = 0$.
- This is nothing more than the acceptance/decision problem for the language $\mathcal{L}(f) = \{\alpha \in \Sigma^* \mid f(\alpha) = 1\}$.
- Formally, a *deterministic algorithm* for f is a deterministic decider (*i.e.*, a DTM) for $\mathcal{L}(f)$.
- Similarly, a *nondeterministic algorithm* for f is a nondeterministic decider (*i.e.*, an NDTM) for $\mathcal{L}(f)$.
- In terms of existence, these two notions are equivalent.
- However, the time complexity (number of steps required to reach a decision) may be very different.
- This idea will prove important in the study of complexity theory.

Abstract Algorithms for General Problems

- For a more general problem which computes a total function

  $$f : \Sigma^* \to \Sigma^*$$

  or even a multi-input total function

  $$f : (\Sigma^*)^k \to \Sigma^*$$

  for some $k > 1$ an abstract algorithm is defined by a DTM which computes f.
- Since f is total, such a DTM must halt on all input configurations.
- This notion is applicable only to DTMs.

Variations on the Turing Model

- There are many minor variations on the Turing machine model.
  - No *stay option* "S" of the tape head.
  - *Semi-infinite tape* rather rather than infinite in both directions.
  - *Off-line machines* (separate input file).
  - *Multitape* Turing machines.
  - Turing machines with *multidimensional tapes*.
  - Nondeterministic versions of all of these.

Fact: In each case, the computational power is equivalent to that of the basic DTM.

Proof: In each case, the details have been worked out be earlier researchers. □

- Another, more interesting equivalent model of computation is based upon conventional imperative programming languages, rather than low-level machines.
- This model will be developed briefly later, time permitting.

DTMs with a Single Accepting State

- There is a minor variation which will be useful in that which follows.

Algorithm: Given a DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$, construct a DTM $M' = (Q', \Sigma, \Gamma, \delta', q_0, \sqcup, F')$ which accepts the same language and computes the same function as M, but which has exactly one final state.

Construction: Put $Q' = Q \cup \{q_f\}$ with $q_f \notin Q$, and $F' = \{q_f\}$.

- Define δ' to have all of the transitions of δ plus those of the form $\delta'(q, a) = (q_f, a, S)$ whenever both of following conditions hold:
  - $q \in F$, and
  - $\delta(q, a)$ is undefined. $\square$