Models of Computation

7: Turing machines

Models of Algoritms

- From 1930s, the increased demand to create a mathematical model of algorithms.
- Several experiments:
    - Kurt Gödel: recursive functions
    - Alonso Church: λ -calculus
    - Alan Turing: Turing Machine
- From the second half of the 1930s, a several theorems were created, which stated the same computing power of these models.
- Later, for many other computation models has been proven that that their computing power is equivalent to Turing machines. For example:
    - Type-0 grammar
    - Pushdown automaton with 2 or more stacks
    - RAM
    - C, Java, Python, etc…

Church-Turing thesis

- No algorithmic system is known that is more powerful than the Turing machine.

  For many algorithmic systems it is proven to be weaker or equivalent to Turing machines.

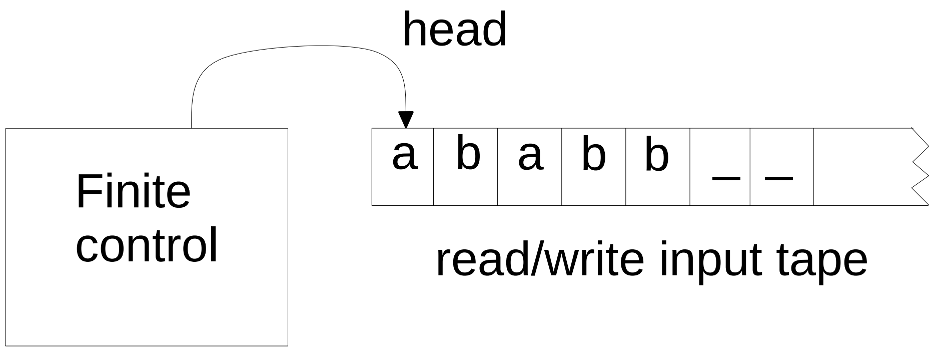
  In the 1930s, the following was formulated:

- **Church-Turing thesis**: All formalizable problems that can be solved with an algorithm, also can be solved with Turing machine.

- NOT THEOREM!

Turing Machines (TMs)

- Introduced by Alan Turing in 1936
- Head can read and write
- Head is two way (can move left or right)
- Tape is infinite (to the right)
- Infinitely many blanks "_" follow input
- Can accept or reject any time (not only at end of input)



Turing Machines (TMs)

TM example: TM recognizing
$L=\{ a^n b^n c^n \mid n \geq 0\}$

1 Scan right until reaching _ while checking if input is in $\{a,b,c\}^*$, reject if not.

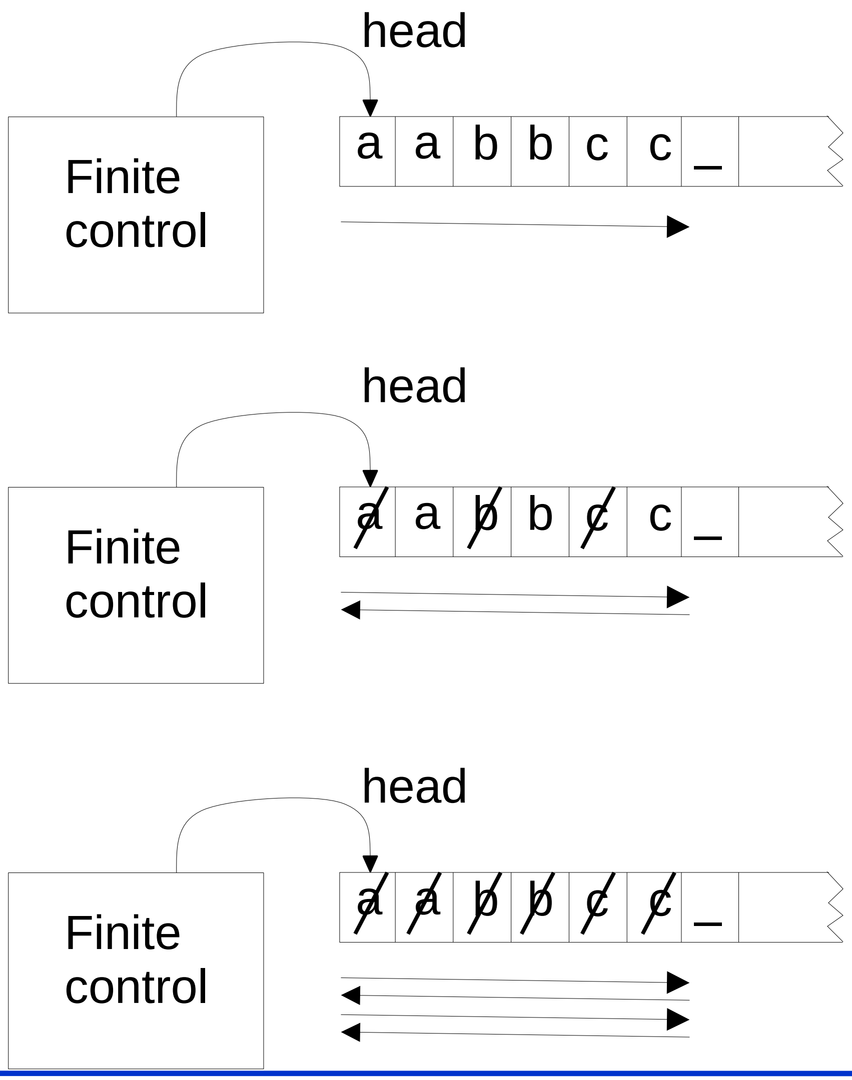
2 Return head to left end.

3 Scan right, crossing off single *a*, *b*, and *c*.

4 If the last one of each symbol, accept.

5 If the last one of some symbol but not others, reject.

6 If uncrossed symbols remain, return to left end and repeat fom 3



TM – Formal Definition

- A **Turing Machine (TM)** is a 7-tuple ($Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}$), where Q, Σ, Γ are finite sets
  - Q: set of states
  - Σ: input alphabet, (blank symbol $\_ \notin \Sigma$)
  - Γ: tape alphabet ($\Sigma \subseteq \Gamma$, $\_ \in \Gamma$)
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ transition function (L = Left, R = Right)
    - Note: deterministic
  - $q_0 \in Q$: start state
  - $q_{accept} \in Q$: accept state
  - $q_{reject} \in Q$: reject state, $q_{reject} \neq q_{accept}$.

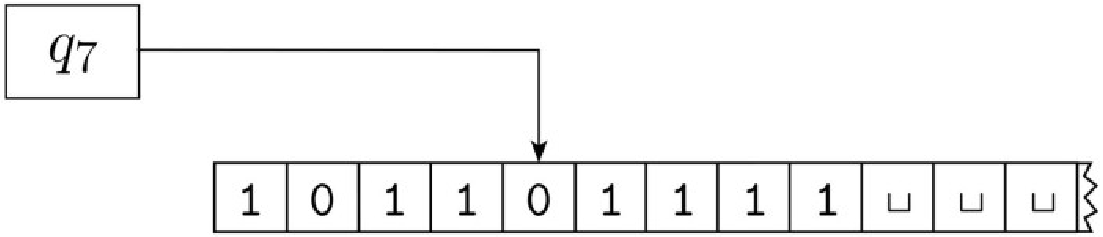
- On input w a TM M may halt (enter q_{accept} or q_{reject}) or may run forever (“loop”).
- TM M has 3 possible outcomes for each input w:
  - Accept (enter q_{accept})
  - Reject by halting (enter q_{reject})
  - Reject by looping (running forever)

TM – Computing

- TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ works as follows.
- The input word of M: $w = w_1w_2\ldots w_n \in \Sigma^*$.
    - is located on the the first n cells of the input tape
    - the other cells are empty (contain the blank symbol _).
- The first (leftmost) _ symbol indicates the end of the input.
- The read/write head is on the leftmost cell of the input tape.
- The calculation is done according to the transition function.
    - If the head has reached the left end of the input tape, then it remains in place regardless of whether the transition function indicated movement to the left (L).
- The calculation is completed when the M enters q_{accept} or q_{reject}. Then M halts.
- If neither case occurs, then M never halts (loop).

TM – Configuration

- The **configuration** of the TM is a word of the form *uqv*, where $u,v \in \Gamma^*$ and $q \in Q$.
    - *uv* : the current content of the tape and that
    - *q*: current position of the head.
    - The head is at the first symbol of *v*.
    - The last symbol of *v* is followed by blanks _ on the tape.
- Example: $1011q_7 01111$



TM – Configuration transition

- Configuration C_1 **yields** configuration C_2 if the TM can legally go from C_1 to C_2 in a single step: Let $a, b, c \in \Gamma$ and $u, v \in \Gamma^*$. Let $q_i, q_j \in Q$.
  - uaq_ibv yields uq_jacv if $\delta(q_i,b) = (q_j,c,L)$,
  - uaq_ibv yields $uacq_jv$ if $\delta(q_i,b) = (q_j,c,R)$.

- Special cases:
  - For the left-hand end of tape,
    - q_ibv yields q_jcv if $\delta(q_i,b) = (q_j,c,L)$,
      (prevents the TM from going off the left end of the tape)
    - q_ibv yields cq_jv if $\delta(q_i,b) = (q_j,c,R)$.
  - For the right-hand end of the input,
    - uaq_i is equivalent to $uaq_i\_$.

TM – Accepting, rejecting, halting

- **Start configuration** of TM M on input w is q_0w.
- **Accepting configuration:** the state of the configuration is q_{accept}.
- **Rejecting configuration:** the state of the configuration is q_{reject}.
- Accepting and rejecting configurations are **halting configuration**s
    - do not yield further configurations.
- δ can be also defined as: δ: $Q' \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$, where $Q' = Q \setminus \{q_{accept}, q_{reject}\}$.

TM – recognized language

- TM M **accepts** input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where

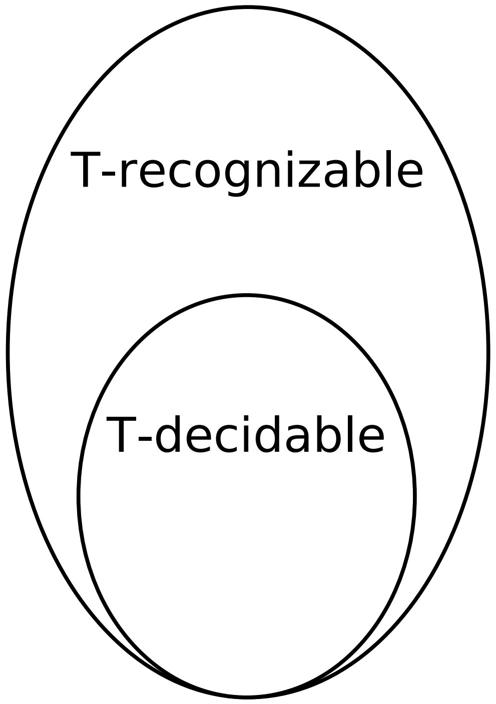
  1. C_1 is the start configuration of M on input w,
  2. each C_i yields C_{i+1}, and
  3. C_k is an accepting configuration.

- The set of words that M accepts is the **language recognized (or accepted) by *M***, denoted by $L(M)$, i.e.
  - $L(M) = \{ w \mid M \text{ accepts } w\}$.

TM – Recognizer, Decider

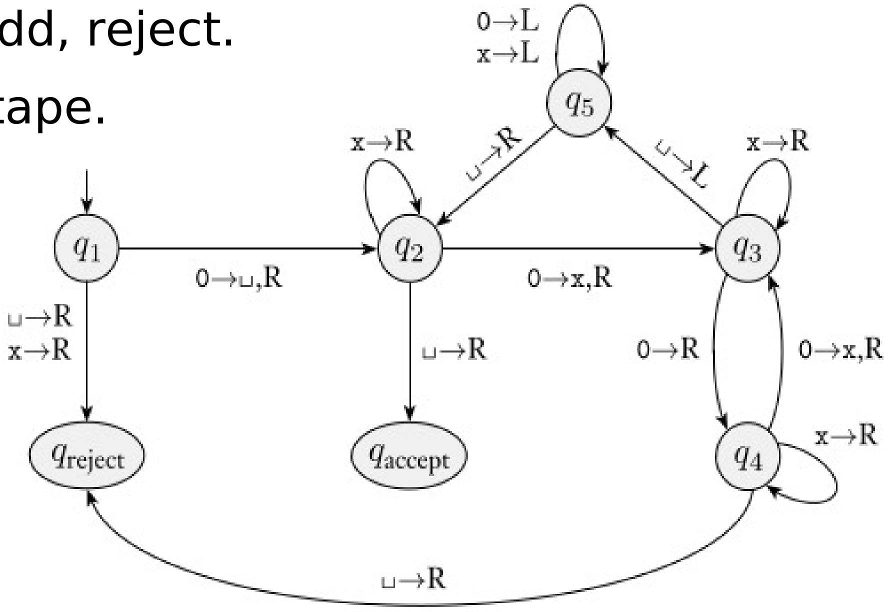
- A TM M **recognizes** L if $L = L(M)$.
- A language L is **Turing-recognizable** if $L = L(M)$ for some TM M.
- Turing-recognizable languages are also called **recursively enumerable** languages.

- A TM M is a **decider** if M halts on all inputs.
- M **decides** L if $L=L(M)$ and M is a decider.
- A language L is **Turing-decidable** if $L = L(M)$ for some decider TM M.
- Turing-deciable languages are also called **recursive** languages



Decider TM example

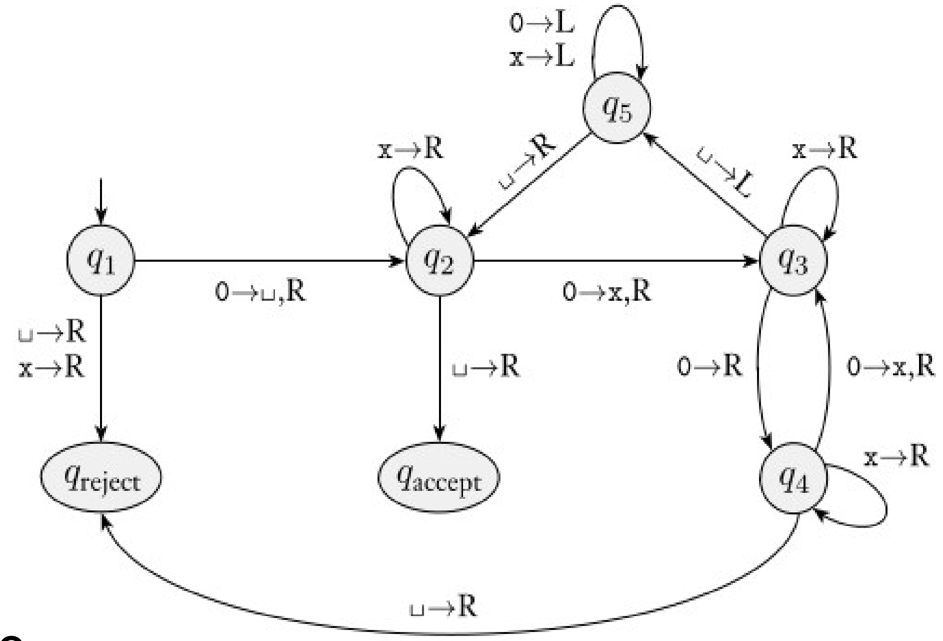
- Example: TM *M* that decides $L = \{0^{2^n} | n \geq 0\}$.
- *M* on input *w*:
  1. Scan the tape left to right crossing off every second 0.
  2. If in stage 1 the tape contained a single 0, accept.
  3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject.
  4. Return the head to the left end of the tape.
  5. Go to stage 1.
- Formal description of $M = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$:
  - $Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$,
  - $\Sigma = \{0\}$, $\Gamma = \{0, x, \_\}$.
  - δ is given with a state diagram:



Decider TM example

- Example: TM *M* that decides

$$L = \{0^{2^n} | n \geq 0\}.$$

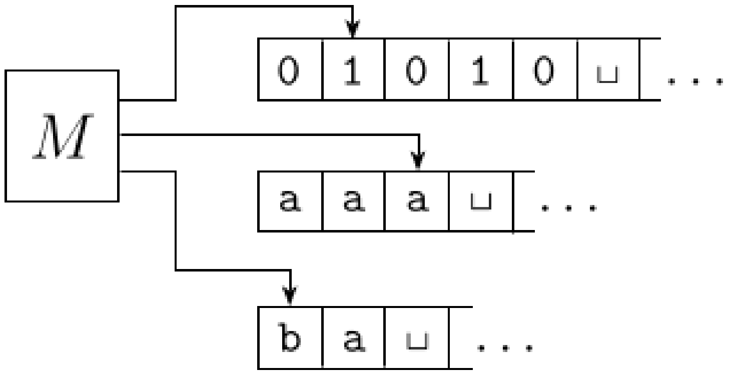


- A sample run of *M* on input 0000:

$q_1 0000$
$\sqcup q_2 000$
$\sqcup \mathtt{x} q_3 00$
$\sqcup \mathtt{x} 0 q_4 0$
$\sqcup \mathtt{x} 0 \mathtt{x} q_3 \sqcup$
$\sqcup \mathtt{x} 0 q_5 \mathtt{x} \sqcup$
$\sqcup \mathtt{x} q_5 0 \mathtt{x} \sqcup$
$\sqcup q_5 \mathtt{x} 0 \mathtt{x} \sqcup$
$q_5 \sqcup \mathtt{x} 0 \mathtt{x} \sqcup$
$\sqcup q_2 \mathtt{x} 0 \mathtt{x} \sqcup$
$\sqcup \mathtt{x} q_2 0 \mathtt{x} \sqcup$
$\sqcup \mathtt{x} \mathtt{x} q_3 \mathtt{x} \sqcup$
$\sqcup \mathtt{x} \mathtt{x} \mathtt{x} q_3 \sqcup$
$\sqcup \mathtt{x} \mathtt{x} q_5 \mathtt{x} \sqcup$
$\sqcup \mathtt{x} q_5 \mathtt{x} \mathtt{x} \sqcup$
$\sqcup q_5 \mathtt{x} \mathtt{x} \mathtt{x} \sqcup$
$q_5 \sqcup \mathtt{x} \mathtt{x} \mathtt{x} \sqcup$
$\sqcup q_2 \mathtt{x} \mathtt{x} \mathtt{x} \sqcup$
$\sqcup \mathtt{x} q_2 \mathtt{x} \mathtt{x} \sqcup$
$\sqcup \mathtt{x} \mathtt{x} q_2 \mathtt{x} \sqcup$
$\sqcup \mathtt{x} \mathtt{x} \mathtt{x} q_2 \sqcup$
$\sqcup \mathtt{x} \mathtt{x} \mathtt{x} \sqcup q_{accept}$

TM Variants – Multitape TM

- Multitape TM:
  - Each tape has its own head for reading and writing.
  - Initially the input is on tape and the others are blank.



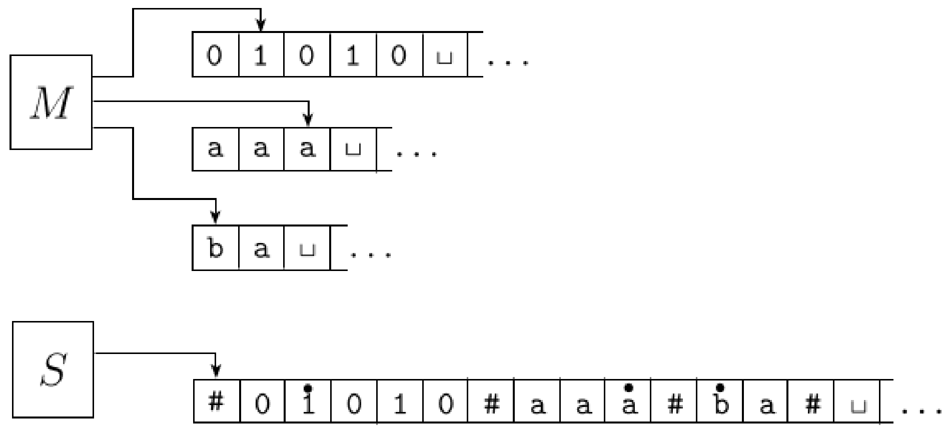
  - The transition function is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously:
    - $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$,
      where k is the number of tapes,
      S: "stay put" it can be substituted by a moving the head to the right then to the left.
    - e.g. $\delta(q_i, a_1, \ldots , a_k) = (q_j, b_1, \ldots. , b_k, L, R, \ldots , L)$

Multitape TM, Single-tape TM

Theorem: Every multitape TM has an equivalent single-tape TM.

Proof: Convert Multitape TM *M* to a single-tape TM *S.*

- *S* simulates *M* by storing the contents of multiple tapes on a single tape in “blocks” separated by a new symbol #.
- The content of the *k* tapes of *M* is represented on the single tape of *S* as *#w#_#_#_…_#*, where *w* is the content of tape 1 of *M*.
- Record head positions with dotted symbols.



Some details of *S*:

1. For each step of *M*

   a: Scan entire tape to find dotted symbols.

   b: Scan again to update according to *M*’s δ.

   c: Shift to add room as needed.

2. Accept/reject if *M* does.

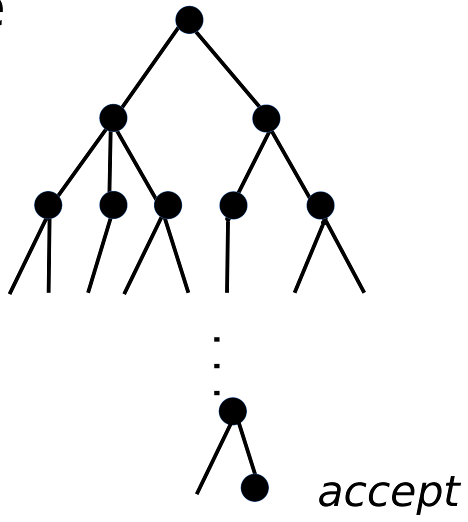
Nondeterministic TM

- A **nondeterministic TM** (NTM) is similar to a deterministic TM except for its transition function $\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$.

Theorem: Every NTM N has an equivalent deterministic TM M.

Proof: Idea:

- The computation of a NTM N with input w can be represented by a tree (computation tree)
  - Nodes correspond to configurations, root corresponds to the start configuration.
- M tries all possible branches of the nondeterministic computation of N. (BFS)
- If M ever finds the accept state on one of the branches, M accepts. Otherwise, M will not terminate.

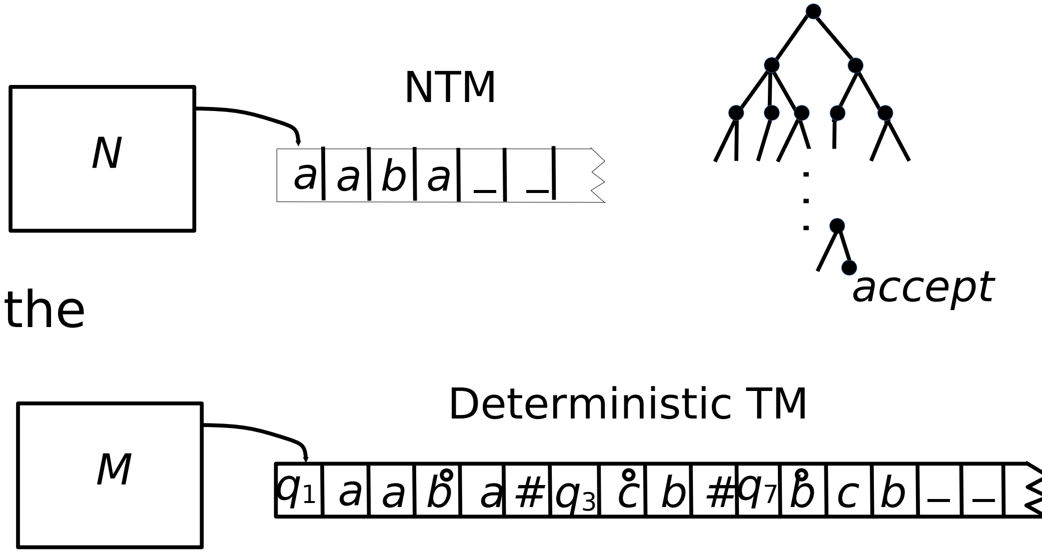


Nondeterministic TM

Theorem: Every NTM *N* has an equivalent deterministic TM *M*.

Proof:

- *M* simulates *N* by storing each thread's tape in a separate "block" on its tape separated by a new symbol #.
- Also need to store the head location,
- and the state for each thread, in the block.
- If a thread forks, then *M* copies the block.
- If a thread accepts then *M* accepts.∎

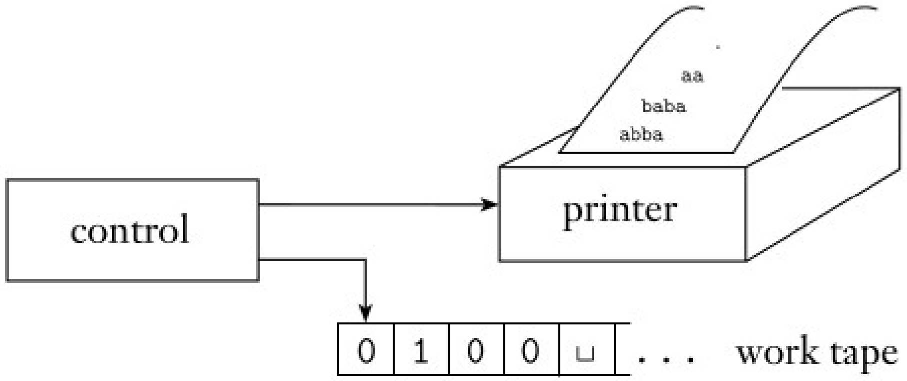


Corollaries

- **Corollary**: A language is Turing-recognizable iff some multitape TM recognizes it.
- **Corollary**: A language is Turing-recognizable iff some nondeterministic TM recognizes it.
- **Remark**: We can modify the proof of previous Theorem, s.t. if *N* terminates on all branches during the nondeterministic calculation, then *M* also terminates. A NTM *N* is called **decider** if *N* terminates in every branch, for each input word.
- **Corollary**: A language can be decided iff some nondeterministic TM decies it.

Turing Enumerators

- A **Turing Enumerator** is a deterministic TM with a printer.
- It starts on a blank tape and it can print strings $w_1, w_2, w_3, \ldots$ possibly going forever.
- Its **language** is the set of all **strings it prints**. It is a generator, not a recognizer.
- For enumerator E: $L(E) = \{w \mid E \text{ prints } w\}$.



Turing Enumerators

Theorem: A language L is Turing-recognizable iff $L=L(E)$ for some Turing enumerator E.

Proof:

- (←) Convert E to an equivalent TM M.
  - M for input w:
    - Simulate E (on blank input).
    - Whenever E prints x, test if $x=w$.
    - Accept if = and continue otherwise.
- (→) Convert TM M to equivalent enumerator E.
  - E = Simulate M on each w_i in $\Sigma^*=\{\varepsilon,0,1,00,01,10,11,\ldots\}$
    - Whenever M accepts w_i, print w_i.
    - Continue with next w_i.
  - Problem: What if M loops with w?
  - Fix: Simulate M on $w_1,w_2,\ldots,w_i$ for i steps, for $i=1,2,\ldots$
    Print those w_i which are accepted. ∎

TMs and Type-0 Grammars

Theorem: For each type-0 grammar G, an NTG can be given, which recognizes $L(G)$.

Proof:

- Let M be a TM having 3 tapes,
    - the first tape contains the input of M,
    - the second contains the rules of the grammar G,
    - the third tape always has an α string (initially, the start symbol G).
- 1. M chooses a rule $p \rightarrow q$ nondeterministically and a position in α.
- 2. If the given position starts with p, i.e. $\alpha = xpy$, then replace p with q, the new string α becomes xqy.
- 3. If the contents of tapes 1 and 3 are equal, M stops with q_{accept}. Otherwise, repeat from 1.
- Clearly, $L(M) = L(G)$ ∎

Corollary: By the previous theorem, a deterministic TM also can be given.

TMs and Type-0 Grammars

Theorem: For each deterministic TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ a grammar G generating $L(M)$ can be given.

Proof:

- The sentences of G will encode the configurations of M. G proceeds in the opposite way. Nondeterministically generates an accepting configuration and then tries to derive a start configuration from that.
- Let $G = ((\Gamma \setminus \Sigma) \cup Q \cup \{S, A, \#, \$\}, \Sigma, P, S)$, where P:
  1) $S \to \#Aq_{accept}A\$$
  2) $A \to aA \mid \varepsilon \quad (\forall a \in \Gamma)$
  3) $bq' \to qa$, if $\delta(q, a) = (q', b, R)$
  4) $q'b \to qa$, if $\delta(q, a) = (q', b, S)$
  5) $q'cb \to cqa$, if $\delta(q, a) = (q', b, L)$ $(\forall c \in \Gamma)$
  6) $\_\$ \to \$$, $\$ \to \varepsilon$, $\#\_ \to \#$, $\#q_0 \to \varepsilon$

TMs and Type-0 Grammars

Proof (cont.):

1) $S \to \#Aq_{accept}A\$$

2) $A \to aA \mid \varepsilon$ $(\forall\, a \in \Gamma\,)$

3) $bq' \to qa$, if $\delta(q, a) = (q', b, R)$

4) $q'b \to qa$, if $\delta(q, a) = (q', b, S)$

5) $q'cb \to cqa$, if $\delta(q, a) = (q', b, L)$ $(\forall\, c \in \Gamma\,)$

6) $\_\$ \to \$$, $\$ \to \varepsilon$, $\#\_ \to \#$, $\#q_0 \to \varepsilon$

1)-2) generate an accepting configuration.

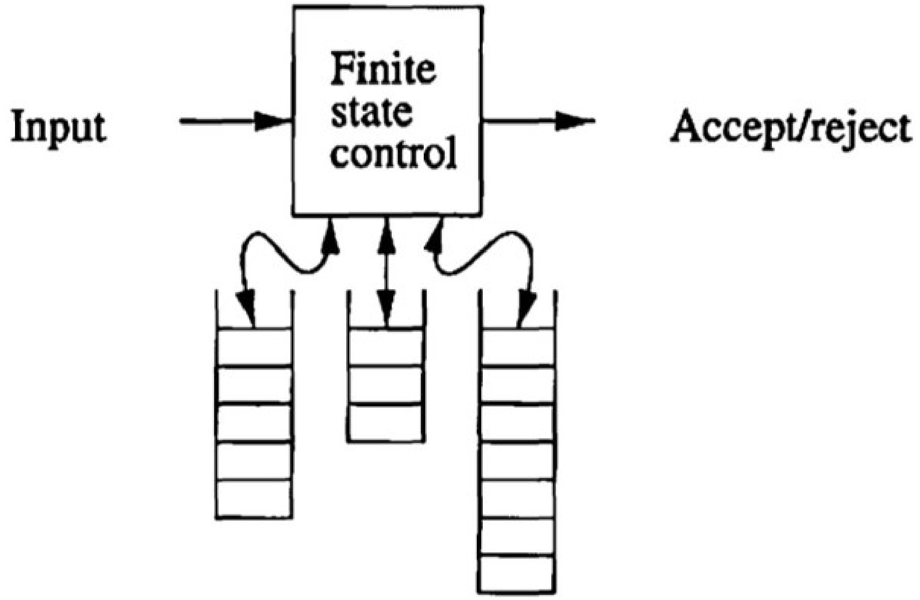
3)-5) configuration transitions are simulated in the reverse direction.
E.g. if $\alpha cqa\beta$ yields $\alpha q'cb\beta$ according to a rule $\delta(q, a) = (q', b, L)$, then by 5), $q'cb$ becomes rewritten to cqa.

6) If the sentence is a start configuration (possibly with a few extra _), then these unnecessary symbols will be cleaned.

By induction on the length of the derivation, it can be shown that
q_0w yields $\alpha q_{accept}\beta$ iff $S \Rightarrow^* \#\alpha q_{accept}\beta\$ \Rightarrow^* \#\_^i q_{accept} w\_^j\$ \Rightarrow^* w$. ∎

Multi-stack pushdown automata

- A ***k*-stack pushdown automato** ***k*-stack PDA**) is a deterministic pushdown automaton with *k* stacks.
- Finite set of states.
- Finite input alphabet.
- Finite stack alphabet, common to all stacks.
- The transition of the multi-stack machine depends on:
    - From the state of the control unit,
    - From the input symbol read,
    - From the stack symbols at the top of each stack.



Multi-stack PDA

- The multi-stack machine during a **state transition**:
  - changes state, and
  - the topmost stack symbol at the top of each stack is replaced by some number (0, 1, or more) of stack symbols.
- A configuration transition (state transition):
  - $\delta(q, a, X_1,\dots,X_k) = (p, y_1,\dots,y_k)$.
    - in state *q*, symbol X_i on the top of the *i*-th stack, $i = 1,\dots,k$, reading the input symbol *a*,
    - goes to state *p* and replaces symbol X_i on the top of the *i*-th stack with y_i, for all $i = 1,\dots,k$.
- The multi-stack machine **accepts** a word when it reaches the accepting end state.
- We assume that there is a special symbol $ at the end of the input (not part of it), called **endmarker**. It indicates that the entire input has been processed.

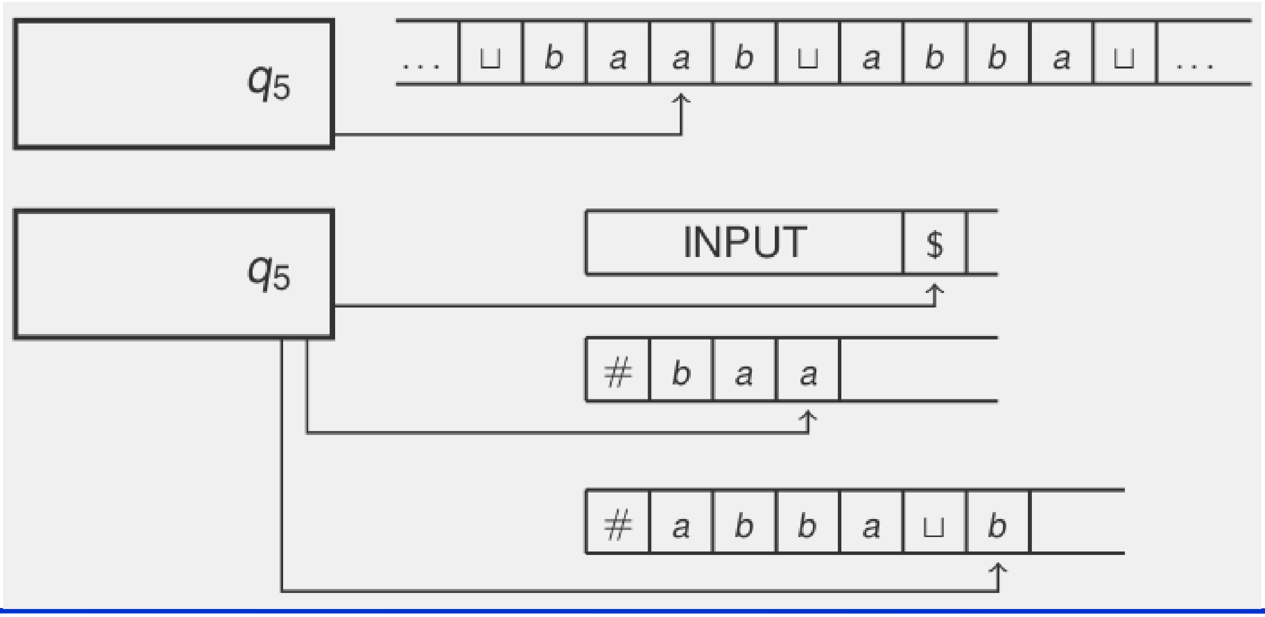
Multi-stack PDA vs. TMs

Theorem: If a language L is accepted by a k-stack PDA, then there is a TM accepting L.

Proof: It is easy to simulate a k-stack machine with a (k+1)-tape TM: we store the input on the first tape, and the other k tapes store the content of the k stacks.
If we write j letters to one of the stacks in a step, we can simulate it with the TM in j steps, letter by letter. ∎

Multi-stack PDA vs. TMs

- **Theorem**: If a language *L* is accepted by a Turing machine *M*, then there is a 2-stack PDA *S* accepting *L*
- **Proof**: Idea: One stack stores the symbols left from the head of the TM, the other stack stores the symbols right ftom the head.



Multi-stack PDA vs. TMs

Proof (cont.):

- Initially, the stacks of *S* start with the # stack symbol #
    - # is not an element of the tape alphabet of *M*.
    - # is only used to indicate the bottom of the stack.
- Assume that the word *w* is present at the input of *S*.
- S copies w to the first stack. *S* finishes copying if it reads $ ($ indicates the end of the input).
- *S* takes the symbols one by one from the first stack and puts them all in the second stack (with ε -transitions).
  Then the first stack will be empty, the second stack will contain *w*, the first letter of *w* will be the top of stack.
- *S* enters the state corresponding to the initial state of *M*.

Multi-stack PDA vs. TMs

Proof (cont.):

- *S* simulates a transition of *M* as follows:
- It is assumed that *S* knows the state of *M*, denoted by *q*.
- For each state *q* of *M*, *S* has its own corresponding state.
- *S* knows the symbol *X* currently being read by *M*: *X* is the top of the second stack of *S*. An exception is, when the second stack contains only # (stack end symbol), in this case *M* just hit a blank symbol.
- So *S* knows the next transition of *M*. *S* goes to its next state according to the next state of *M*.

Multi-stack PDA vs. TMs

Proof (cont.):

- We only need to deal with moving to the left and to the right.
- If *M* replaces a symbol *X* with *Y* and moves to the right, then *S* puts *Y* to stack-1 and removes *X* from the top of stack-2, i.e. *Y* is to the left of the head of *M*. There are two special cases:
    - If stack-2 only contains the stack end symbol # and thus *X* = _, then the contents of stack-2 does not change, (*M* moves towards another _ to the right.)
    - If *Y* = _ and # is on top of stack-1, then the content of stack-1 remains unchanged. (To the left of *M*'s head there are only _ symbols.)

Multi-stack PDA vs. TMs

Proof (cont.):

- If *M* replaces symbol *X* with *Y* and moves to the left, then *S* removes the topmost symbol – call it *Z* – of the stack-1, and replaces *X* with *ZY* in the top of stack-2.
    - I.e. the symbol (*Z*) that was to the left of the head of *M*, will be under the head. An exception is the case where *Z* is the stack end symbol #. Then _*Y* is placed on the top of stack-2 and nothingg is removed from stack-1.
- *S* accepts if the new state of *M* is accepting, otherwise *S* simulates the next step of *M*.∎

References

- Michael Sipser: Introduction to the Theory of Computation. 3rd edition, 2012.