Formale Methoden der Informatik

Block 1: Foundations of Complexity Theory

6. Turing Machines

Reinhard Pichler

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

21 October, 2013

Outline

Basic Definitions

Motivation

- Turing machines are used as a formal model of algorithms.
- Recall that SIMPLE programs were not defined completely.
- The notion of Turing machines is better suited for a formal definition of complexity classes and for formal proofs of properties (e.g. that any problem in NP can be reduced to **SAT**).

Definition

A Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$ with a finite set of states K, a finite set of symbols Σ (alphabet of M) so that $\sqcup, \triangleright \in \Sigma$, a transition function δ:

$$K \times \Sigma \to (K \cup \{\text{h}, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\},$$

a halting state h, an accepting state “yes”, a rejecting state “no”, and cursor directions: $\rightarrow$ (right), $\leftarrow$ (left), and $-$ (stay).

Example 1: Recognizing palindromes

Example

Consider a Turing machine M for recognizing palindromes:
$M = (K, \Sigma, \delta, s)$ with $K = \{s, q, q_0, q_1, q_0', q_1'\}$, $\Sigma = \{0, 1, \sqcup, \triangleright\}$ and a transition function δ defined as follows:

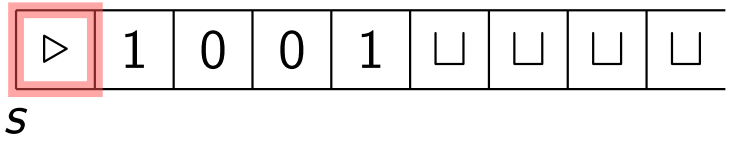
$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	0	$(q_0, \triangleright, \rightarrow)$
s	1	$(q_1, \triangleright, \rightarrow)$
s	$\sqcup$	$(\text{“yes”}, \sqcup, -)$
q_0	0	$(q_0, 0, \rightarrow)$
q_0	1	$(q_0, 1, \rightarrow)$
q_0	$\sqcup$	$(q_0', \sqcup, \leftarrow)$
q_1	0	$(q_1, 0, \rightarrow)$
q_1	1	$(q_1, 1, \rightarrow)$
q_1	$\sqcup$	$(q_1', \sqcup, \leftarrow)$

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
q_0'	0	$(q, \sqcup, \leftarrow)$
q_0'	1	$(\text{“no”}, 1, -)$
q_0'	$\triangleright$	$(\text{“yes”}, \triangleright, -)$
q_1'	1	$(q, \sqcup, \leftarrow)$
q_1'	0	$(\text{“no”}, 0, -)$
q_1'	$\triangleright$	$(\text{“yes”}, \triangleright, -)$
q	0	$(q, 0, \leftarrow)$
q	1	$(q, 1, \leftarrow)$
q	$\triangleright$	$(s, \triangleright, \rightarrow)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

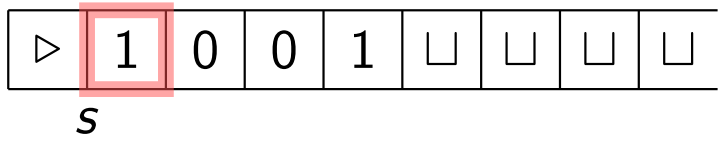


$(s, \triangleright, 1001)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

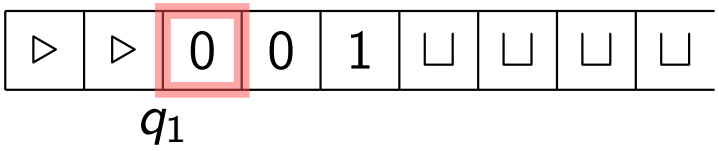


$(s, \triangleright, 1001) \quad (s, \triangleright 1, 001)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

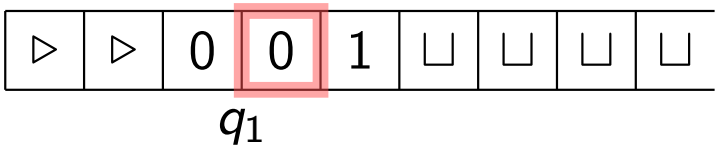


$(s, \triangleright, 1001) \quad (s, \triangleright 1, 001) \quad (q_1, \triangleright \triangleright 0, 01)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

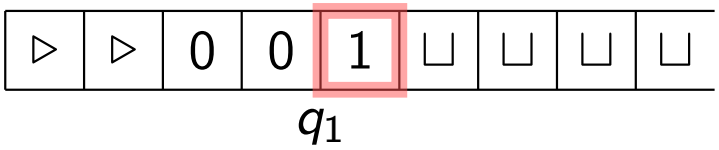


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright \triangleright 0, 01)$ $(q_1, \triangleright \triangleright 00, 1)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

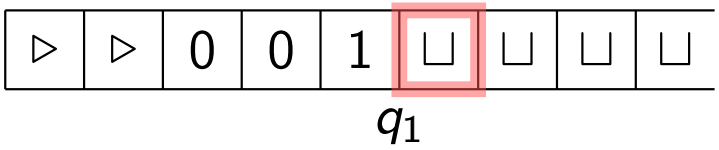


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright \triangleright 0, 01)$ $(q_1, \triangleright \triangleright 00, 1)$ $(q_1, \triangleright \triangleright 001, \epsilon)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

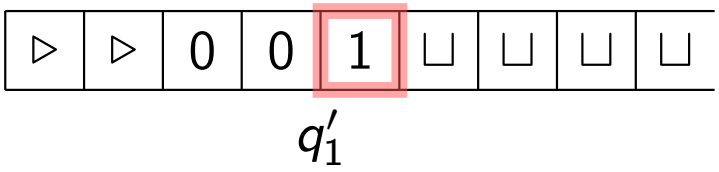


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright \triangleright 0, 01)$ $(q_1, \triangleright \triangleright 00, 1)$ $(q_1, \triangleright \triangleright 001, \epsilon)$
$(q_1, \triangleright \triangleright 001\sqcup, \epsilon)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

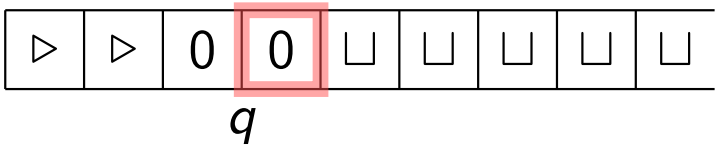


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright \triangleright 0, 01)$ $(q_1, \triangleright \triangleright 00, 1)$ $(q_1, \triangleright \triangleright 001, \epsilon)$
$(q_1, \triangleright \triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright \triangleright 001, \sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

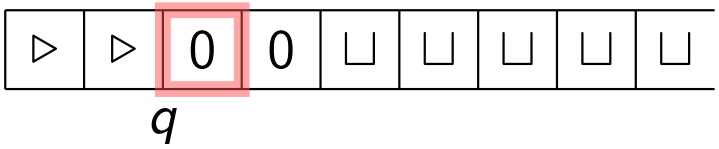


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright \triangleright 0, 01)$ $(q_1, \triangleright \triangleright 00, 1)$ $(q_1, \triangleright \triangleright 001, \epsilon)$
$(q_1, \triangleright \triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright \triangleright 001, \sqcup)$ $(q, \triangleright \triangleright 00, \sqcup\sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

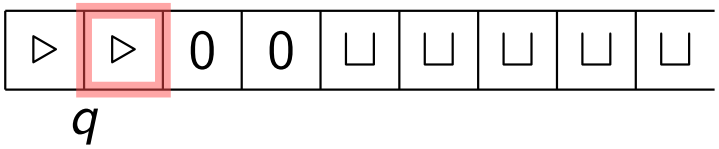


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright \triangleright 0, 01)$ $(q_1, \triangleright \triangleright 00, 1)$ $(q_1, \triangleright \triangleright 001, \epsilon)$
$(q_1, \triangleright \triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright \triangleright 001, \sqcup)$ $(q, \triangleright \triangleright 00, \sqcup\sqcup)$ $(q, \triangleright \triangleright 0, 0 \sqcup \sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

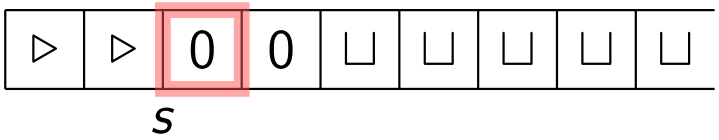


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright \triangleright 0, 01)$ $(q_1, \triangleright \triangleright 00, 1)$ $(q_1, \triangleright \triangleright 001, \epsilon)$
$(q_1, \triangleright \triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright \triangleright 001, \sqcup)$ $(q, \triangleright \triangleright 00, \sqcup\sqcup)$ $(q, \triangleright \triangleright 0, 0 \sqcup \sqcup)$
$(q, \triangleright\triangleright, 00 \sqcup \sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

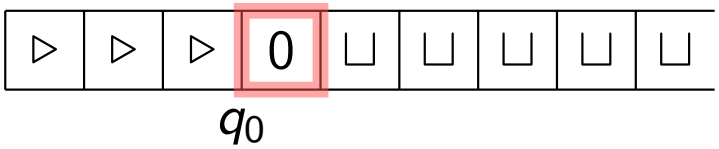


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright\triangleright 0, 01)$ $(q_1, \triangleright\triangleright 00, 1)$ $(q_1, \triangleright\triangleright 001, \epsilon)$
$(q_1, \triangleright\triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright\triangleright 001, \sqcup)$ $(q, \triangleright\triangleright 00, \sqcup\sqcup)$ $(q, \triangleright\triangleright 0, 0\sqcup\sqcup)$
$(q, \triangleright\triangleright, 00\sqcup\sqcup)$ $(s, \triangleright\triangleright 0, 0\sqcup\sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

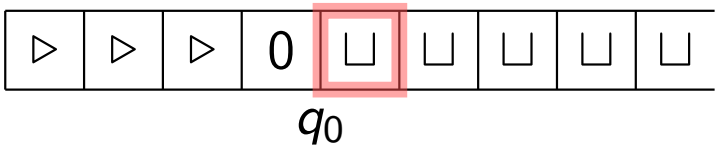


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright\triangleright 0, 01)$ $(q_1, \triangleright\triangleright 00, 1)$ $(q_1, \triangleright\triangleright 001, \epsilon)$
$(q_1, \triangleright\triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright\triangleright 001, \sqcup)$ $(q, \triangleright\triangleright 00, \sqcup\sqcup)$ $(q, \triangleright\triangleright 0, 0\sqcup\sqcup)$
$(q, \triangleright\triangleright, 00\sqcup\sqcup)$ $(s, \triangleright\triangleright 0, 0\sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

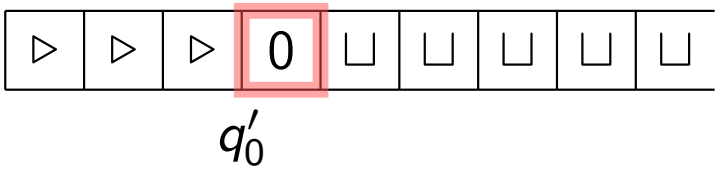


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright\triangleright 0, 01)$ $(q_1, \triangleright\triangleright 00, 1)$ $(q_1, \triangleright\triangleright 001, \epsilon)$
$(q_1, \triangleright\triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright\triangleright 001, \sqcup)$ $(q, \triangleright\triangleright 00, \sqcup\sqcup)$ $(q, \triangleright\triangleright 0, 0\sqcup\sqcup)$
$(q, \triangleright\triangleright, 00\sqcup\sqcup)$ $(s, \triangleright\triangleright 0, 0\sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0\sqcup, \sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

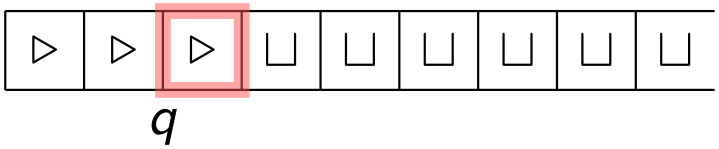


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright\triangleright 0, 01)$ $(q_1, \triangleright\triangleright 00, 1)$ $(q_1, \triangleright\triangleright 001, \epsilon)$
$(q_1, \triangleright\triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright\triangleright 001, \sqcup)$ $(q, \triangleright\triangleright 00, \sqcup\sqcup)$ $(q, \triangleright\triangleright 0, 0\sqcup\sqcup)$
$(q, \triangleright\triangleright, 00\sqcup\sqcup)$ $(s, \triangleright\triangleright 0, 0\sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0\sqcup, \sqcup)$
$(q_0', \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

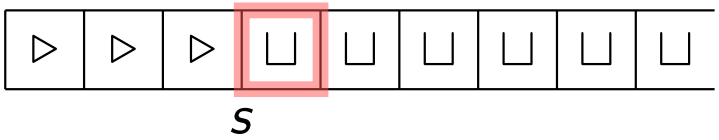


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright\triangleright 0, 01)$ $(q_1, \triangleright\triangleright 00, 1)$ $(q_1, \triangleright\triangleright 001, \epsilon)$
$(q_1, \triangleright\triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright\triangleright 001, \sqcup)$ $(q, \triangleright\triangleright 00, \sqcup\sqcup)$ $(q, \triangleright\triangleright 0, 0\sqcup\sqcup)$
$(q, \triangleright\triangleright, 00\sqcup\sqcup)$ $(s, \triangleright\triangleright 0, 0\sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0\sqcup, \sqcup)$
$(q_0', \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$ $(q, \triangleright\triangleright\triangleright, \sqcup\sqcup\sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:

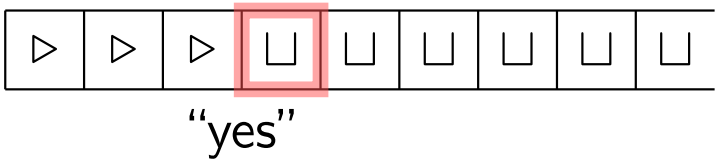


$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright\triangleright 0, 01)$ $(q_1, \triangleright\triangleright 00, 1)$ $(q_1, \triangleright\triangleright 001, \epsilon)$
$(q_1, \triangleright\triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright\triangleright 001, \sqcup)$ $(q, \triangleright\triangleright 00, \sqcup\sqcup)$ $(q, \triangleright\triangleright 0, 0\sqcup\sqcup)$
$(q, \triangleright\triangleright, 00\sqcup\sqcup)$ $(s, \triangleright\triangleright 0, 0\sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0\sqcup, \sqcup)$
$(q_0', \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$ $(q, \triangleright\triangleright\triangleright, \sqcup\sqcup\sqcup)$ $(s, \triangleright\triangleright\triangleright\sqcup, \sqcup\sqcup)$

Example 1 (continued)

Example

Consider the computation of M on the input string 1001:



$(s, \triangleright, 1001)$ $(s, \triangleright 1, 001)$ $(q_1, \triangleright\triangleright 0, 01)$ $(q_1, \triangleright\triangleright 00, 1)$ $(q_1, \triangleright\triangleright 001, \epsilon)$
$(q_1, \triangleright\triangleright 001\sqcup, \epsilon)$ $(q_1', \triangleright\triangleright 001, \sqcup)$ $(q, \triangleright\triangleright 00, \sqcup\sqcup)$ $(q, \triangleright\triangleright 0, 0\sqcup\sqcup)$
$(q, \triangleright\triangleright, 00\sqcup\sqcup)$ $(s, \triangleright\triangleright 0, 0\sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$ $(q_0, \triangleright\triangleright\triangleright 0\sqcup, \sqcup)$
$(q_0', \triangleright\triangleright\triangleright 0, \sqcup\sqcup)$ $(q, \triangleright\triangleright\triangleright, \sqcup\sqcup\sqcup)$ $(s, \triangleright\triangleright\triangleright\sqcup, \sqcup\sqcup)$ $(\text{“yes”}, \triangleright\triangleright\triangleright\sqcup, \sqcup\sqcup)$

High-level description of M

1. in state s: Move the cursor to the first symbol to the right of $\triangleright$ and store this symbol in the next state, i.e., enter state q_i (with $i \in \{0,1\}$) when i was read. If no more such symbol exists (i.e., we read $\sqcup$ in state s), then halt with "yes" (i.e., ϵ is a palindrome).
2. in state q_i (with $i \in \{0,1\}$): move to the right until the first $\sqcup$ is reached (the tape contents is left unchanged). When $\sqcup$ is finally read, enter state q_i' and move back to the last non-blank symbol.
3. in state q_i': check if the current symbol is i: If this is the case, then we "erase" i and enter state q. If in state q_0' we read 1 (or in state q_1' we read 0), then halt with "no". If we read $\triangleright$ (i.e., no more symbol $i \in \{0,1\}$ is left), then halt with "yes".
4. in state q: The first and last symbol of the string have now been erased (i.e. overwritten by $\triangleright$ and $\sqcup$, resp.). State q is used to move the cursor back to the $\triangleright$ preceding the remaining string.
5. When $\triangleright$ is read in state q, then enter s and continue as in step 1.

Transition function

- Function δ is the "program" of the machine.
- For the current state $q \in K$ and the current symbol $\sigma \in \Sigma$,
  – $\delta(q, \sigma) = (p, \rho, D)$ where p is the new state,
  – ρ is the symbol to be overwritten on σ, and
  – $D \in \{\rightarrow, \leftarrow, -\}$ is the direction in which the cursor will move.
- For any states p and q, $\delta(q, \triangleright) = (p, \rho, D)$ with $\rho = \triangleright$ and $D = \rightarrow$.
  In other words: The delimiter $\triangleright$ is never overwritten by another symbol, and the cursor never moves off the left end of the tape.
- If the machine moves off the right end of the string, it reads $\sqcup$.
  In other words: The string becomes longer but it cannot become shorter; thus it keeps track of the space used by the machine.

- The machine starts as follows:
  (i) the initial state of $M = (K, \Sigma, \delta, s)$ is s,
  (ii) the tape is initialized to $\triangleright x$ where x is a finitely long string in $(\Sigma - \{\sqcup, \triangleright\})^*$ (x is the *input* of the machine) and
  (iii) the cursor points to $\triangleright$.
- The machine halts iff one of the 3 halting states (h, "yes", "no") has been reached.
- If "yes" has been reached, the machine accepts the input.
  If "no" has been reached, the machine rejects the input.
- Output $M(x)$ of a machine M on input x:
  (i) If M accepts/rejects, then $M(x) =$ "yes"/"no".
  (ii) If h has been reached, then $M(x) = y$, where $\triangleright y \sqcup \ldots$ is the string of M at the time of halting ($y =$ string between $\triangleright$ and first $\sqcup$).
  (iii) If M never halts on input x, then $M(x) = \nearrow$
  (In this case, we say that M "diverges".)

Operational Semantics

- A configuration (q, w, u):
  $q \in K$ is the current state and $w, u \in \Sigma^*$ where
  (i) w is the string to the left of the cursor including the symbol scanned by the cursor and
  (ii) u is the string to the right of the cursor.
- The relation $\overset{M}{\to}$ (yields in one step): $(q, w, u) \overset{M}{\to} (q', w', u')$
  Let σ be the last symbol of w and $\delta(q, \sigma) = (p, \rho, D)$.
  Then $q' = p$, and w', u' are obtained according to (p, ρ, D).
- Example. If $D = \to$, then
  (i) w' is w with its last symbol replaced by ρ and the first symbol of u appended to it ($\sqcup$ if u is empty) and
  (ii) u' is u with the first symbol removed (or empty, if u is empty).

Configurations reached in several steps

- Yields in k steps: $(q, w, u) \overset{M}{\rightarrow}^{k} (q', w', u')$
  iff there are configurations $(q_i, w_i, u_i), i = 1, \ldots, k+1$ such that
  - $(q, w, u) = (q_1, w_1, u_1)$,
  - $(q_i, w_i, u_i) \overset{M}{\rightarrow} (q_{i+1}, w_{i+1}, u_{i+1}), i = 1, \ldots, k$, and
  - $(q', w', u') = (q_{k+1}, w_{k+1}, u_{k+1})$
- Yields: $(q, w, u) \overset{M}{\rightarrow}^{*} (q', w', u')$

  iff there is some $k \geq 0$ such that $(q, w, u) \overset{M}{\rightarrow}^{k} (q', w', u')$.
- Therefore, $\overset{M}{\rightarrow}^{*}$ is the transitive and reflexive closure of $\overset{M}{\rightarrow}$.

Turing Machines as Algorithms

Definition

- Let $L \subset (\Sigma - \{\triangleright, \sqcup\})^*$ be a language.
  A Turing machine M decides L iff for every string $x \in (\Sigma - \{\triangleright, \sqcup\})^*$, the following conditions hold:
  - if $x \in L$, $M(x) =$ "yes" and
  - if $x \notin L$, $M(x) =$ "no" .
- If L is decided by a Turing machine, then L is a recursive language.

Definition

- A Turing machine M accepts L iff for every string $x \in (\Sigma - \{\triangleright, \sqcup\})^*$,
  - if $x \in L$, then $M(x) =$ "yes"
  - if $x \notin L$, then either $M(x) =$ "no" or $M(x) = \nearrow$.
- If L is accepted by some Turing machine, then L is a recursively enumerable language (r.e., for short).

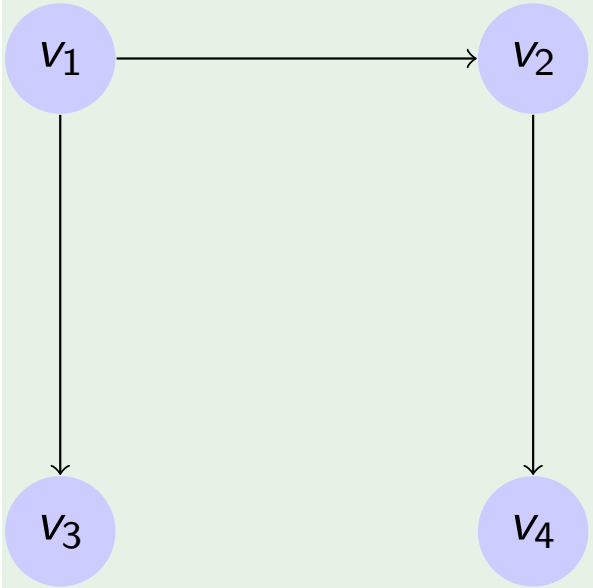
Solving problems using Turing machines

- The instances of a problem need to be represented by strings.
- Observation. Any "finite" mathematical object can be represented by a finite string over an appropriate alphabet.
- Solving a decision problem amounts to deciding the language consisting of the encodings of the "yes" instances of the problem.
- Therefore, we may identify decision problems with languages: Decidable (resp. semi-decidable) problems correspond to recursive (resp. recursively enumerable) languages.

How does representation affect solvability?

Representation of a graph

Graph:



Various string representations:

1. List of edges:
   "$\{(1,10),(1,11),(10,100)\}$"
2. adjacency matrix (as list of rows):
   "$(0110,0001,0000,0000)$"

How does representation affect solvability?

- All "reasonable" encodings are related polynomially:
  If A and B are both "reasonable" representations of the same set of instances, and representation A of an instance is a string with n symbols, then the representation B of the same instance has length at most $p(n)$ for some polynomial p.
- Exception. Unary representation of numbers requires exponentially more symbols than the binary representation.
- A reasonably succinct input representation is assumed.
  In particular, numbers are always represented in binary.

Time Bounds

Definition

- The time required by M on input x is t iff

$$(s, \triangleright, x) \overset{M}{\rightarrow}^{t} (H, w_1, u_1)$$

  with $H \in \{\text{h}, \text{“yes”}, \text{“no”}\}$.
- If $M(x) = \nearrow$, then the time required is thought to be ∞.

Definition

Machine M operates in time $f(n)$ if, for any input string x, the time required by M on x is at most $f(|x|)$.

Formally Defining P and EXPTIME

Defining the class P (i.e. problems solvable in polynomial time)

$\mathsf{P} = \{L \mid L \text{ is decided by some TM operating in time } \mathrm{O}(n^k) \text{ where } k \text{ is a constant}\}$

Defining the class EXPTIME (i.e. problems solvable in exponential time)

$\mathsf{EXPTIME} = \{L \mid L \text{ is decided by some TM operating in time } \mathrm{O}(2^{n^k}) \text{ where } k \text{ is a constant}\}$

Space Bounds

Definition

The space required by M on input x is ℓ, if

$$(s, \triangleright, x) \overset{M}{\rightarrow}^{*} (q', w', u')$$

implies $|w'u'| \leq \ell$. In other words, M uses at most ℓ tape cells.

Definition

Machine M operates in space $f(n)$ if, for any input string x, the space required by M on x is at most $f(|x|)$.

Special Treatment of Input and Output

Motivation

- With the above definition of space complexity, we treat input/output in the same way as the actual workspace.
- This does not allow us to detect sublinear space or to recognize that some exponentially big output is produced by a program with polynomial workspace (e.g., a winning strategy in Tic-Tac-Toe).

Dedicated input/output tape

- To detect sublinear space, we assume that the input to M is written onto a separate read-only input tape.
- For a function problem, we may also assume that the output is written to a write-only output tape.
- We assume that input/output tape do not contribute to space usage.

Defining Space Complexity Classes

Defining the class L (i.e. problems solvable in logarithmic space)

$\mathsf{L} = \{L \mid L \text{ is decided by some TM operating in space } \mathrm{O}(\log n)\}$

Defining the class PSPACE (i.e. problems solvable in polynomial space)

$\mathsf{PSPACE} = \{L \mid L \text{ is decided by some TM operating in space } \mathrm{O}(n^k) \text{ where } k \text{ is a constant}\}$

Defining the class EXPSPACE (i.e. problems solvable in polynomial space)

$\mathsf{EXPSPACE} = \{L \mid L \text{ is decided by some TM operating in space } \mathrm{O}(2^{n^k}) \text{ where } k \text{ is a constant}\}$

Church-Turing Thesis revisited

Church-Turing Thesis

Any "reasonable" attempt to model mathematically computer algorithms ends up with a model of computation that is equivalent to Turing machines.

Evidence for this thesis

All of the following models can be shown to have precisely the same expressive power as Turing machines:

- our programming language SIMPLE
- random access machines
- μ-recursive functions
- any conventional programming language (Java, C, ...)

Strengthening of the Church-Turing Thesis

Any "reasonable" attempt to model mathematically computer algorithms and their time/space performance ends up with a model of computation and associated time/space cost that is equivalent to Turing machines within a polynomial."

Evidence for this thesis

- Variations of the definition of Turing machines (e.g., 2-side-infinite tape, k-string TMs, etc.) lead to the same complexity classes.
- It can be shown that Turing machines can simulate random access machines with only a polynomial loss of efficiency and vice versa.

  (For a formal proof see Papadimitriou, Theorems 2.4 + 2.5)

Nondeterministic Machines

Motivation

- Nondeterministic machines are an unrealistic model of computation.
- Nevertheless, they have proved to be very useful in complexity analyses.
- Intuition: Identify complexity due to size of the search space
- Nondeterministic TMs can be simulated by deterministic TMs with an exponential loss of efficiency.
- An open question: is a polynomial simulation possible? (i.e. P = NP?)

Transition relation

Definition

A nondeterministic Turing machine (NTM) is a quadruple $N = (K, \Sigma, \Delta, s)$ like the ordinary Turing machine except that Δ is a transition relation (rather than a transition function):

$$\Delta \subset (K \times \Sigma) \times [(K \cup \{\text{h}, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\}]$$

Definition

Configurations are defined as before, but “yields” is a relation (rather than a function) for an NTM N: $(q, w, u) \overset{N}{\rightarrow} (q', w', u')$ iff there is a tuple in Δ that makes this a legal transition.

NTMs deciding languages

Definition

A nondeterministic Turing machine N decides a language L iff for any $x \in \Sigma^*$, the following holds:

$$x \in L \text{ iff } (s, \triangleright, x) \overset{N}{\rightarrow}^* (\text{“yes”}, w, u) \text{ for some strings } w \text{ and } u.$$

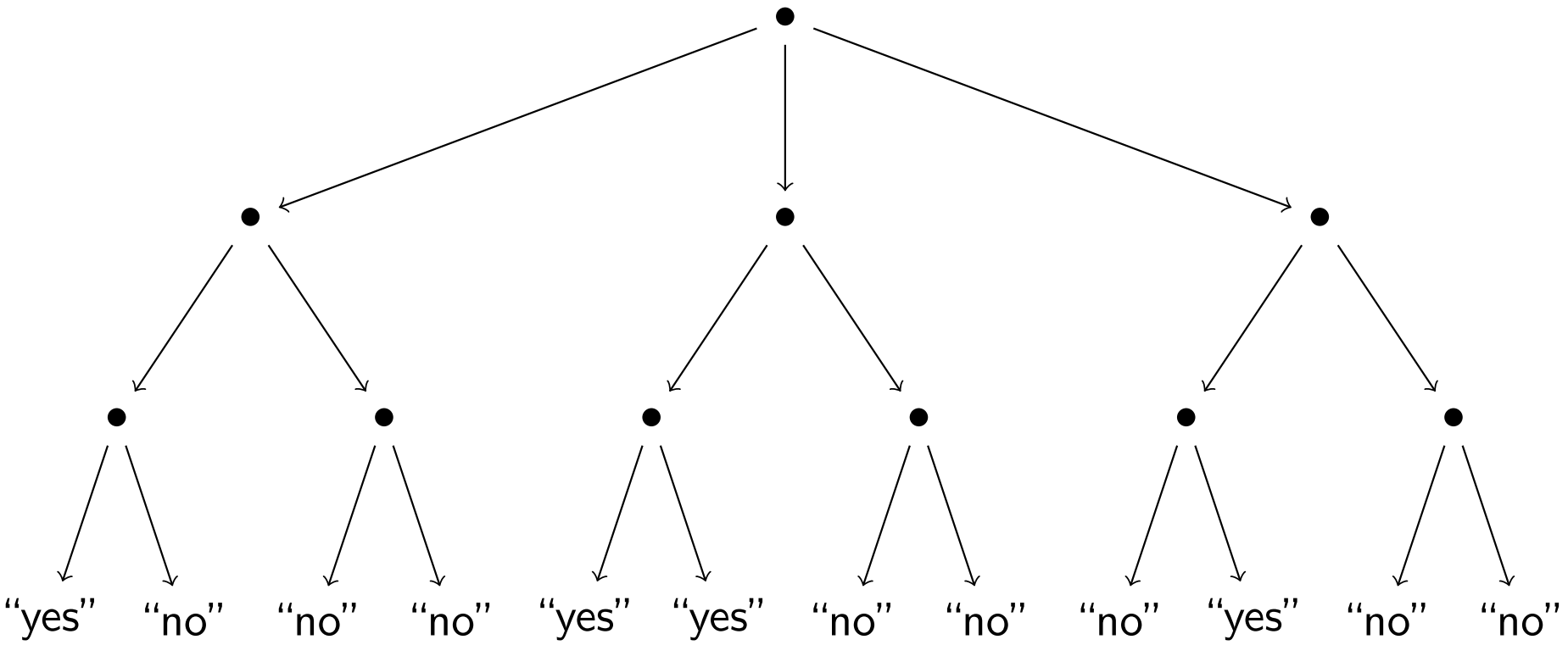
Remarks

(i) An input is accepted if there is some sequence of nondeterministic choices that results in the accepting state “yes”.

(ii) The input is rejected only if no sequence of nondeterministic choices can lead to acceptance.

(iii) Note the asymmetry between accepting and rejecting!

Nondeterministic Computation (Example)



Time Complexity

Definition

A nondeterministic Turing machine N decides a language L in time $f(n)$ iff N decides L and
for any $x \in \Sigma^*$, if $(s, \triangleright, x) \overset{N}{\rightarrow}{}^{k} (q, w, u)$, then $k \leq f(|x|)$.

$\Longrightarrow$ All possible computation paths must have length at most $f(|x|)$.

NTMs vs. DTMs

Theorem (A)

Suppose that a language L is decided by an NTM N in time $f(n)$. Then it is also decided by a deterministic TM M in time $\mathrm{O}(c^{f(n)})$, where $c > 1$ is a constant depending on N.

Theorem (B)

Suppose that a language L is decided by an NTM N in time $f(n)$. Then it is also decided by a deterministic TM M in space $\mathrm{O}((f(n))^d)$, where $d > 1$ is a constant depending on N.

Proof idea. The proof is by simulating (interpreting) N using a deterministic TM M. For this, on input x, the machine M traverses all the possible computation paths of N on x, looking for an accepting path. This causes an exponential blow-up in time (hence $\mathrm{O}(c^{f(n)})$), but can be done in space that is polynomial (hence $\mathrm{O}((f(n))^d)$).

We have applied this argument in more detail to prove NP $\subseteq$ PSPACE!

Nondeterministic Guesses

Observation

The proof idea of the previous theorem reveals another intuition of nondeterminism, namely guess and check:

- We can think of a nondeterministic computation as guessing an appropriate sequence $(c_1, c_2, \ldots, c_t)$ of transitions and checking that, with these choices, the computation is indeed successful.
- Guesses are not confined to sequences of transitions. An NTM can guess any (intermediate) result which might be computed by such a sequence of transitions, and check that the guess was correct.
- Example. NP-algorithm for TSP(D): Guess a permutation of the cities and check that the cost of the tour fits within the budget.

Defining Nondeterministic Time Complexity Classes

Defining NP via NTMs

$\mathrm{NP} = \{L \mid L \text{ is decided by some NTM in time } \mathrm{O}(n^k) \text{ where } k \text{ is a constant}\}$

The class NEXPTIME

$\mathrm{NEXPTIME} = \{L \mid L \text{ is decided by some NTM in time } \mathrm{O}(2^{n^k}) \text{ where } k \text{ is a constant}\}$

Adequacy of the Second Definition of NP

The above definition and the definition of NP in terms of polynomially balanced and polynomially decidable relation are equivalent!

From Certificates to NTMs

For a problem $\mathcal{P}$ that has a polynomially balanced and polynomially decidable relation R, we can devise an NTM N that decides $\mathcal{P}$ (i.e. $\mathcal{P}$ viewed as a language) in polynomial time:

On instance x, N uses nondeterminism to write a candidate certificate C on the tape and then checks in polynomial time whether $(x, C) \in R$.

From NTMs to Certificates

Any language L that can be decided by an NTM N in polynomial time can also be characterized in terms of a polynomially balanced and polynomially decidable certificate relation R for L:

Define R by assigning to all positive instances of L (i.e., the words $x \in L$), the polynomially long accepting computations of M on x.

Space Complexity

Definition

Given an NTM N we say that N decides language L within space $f(n)$ iff N decides L and for any $x \in (\Sigma - \{\sqcup\})^*$,

if $(s, \triangleright, x, \triangleright) \xrightarrow{N}{}^{*} (q, w', u')$,

then $|w'u'| \leq f(|x|)$.

Intuition. All possible computations require at most $f(|x|)$ space.

Sublinear space

For sublinear function (e.g. $f(n) = \log n$), we make the same assumption of a separate *read-only* tape, which does not contribute to space usage.

Defining Nondeterministic Space Complexity Classes

The class NL (nondeterministic logarithmic space)

$\mathrm{NL} = \{L \mid L \text{ is decided by some NTM within space } \mathrm{O}(\log n)\}$

The class NPSPACE

$\mathrm{NPSPACE} = \{L \mid L \text{ is decided by some NTM within space } \mathrm{O}(n^k) \text{ where } k \text{ is a constant}\}$

The class NEXPSPACE

$\mathrm{NEXPSPACE} = \{L \mid L \text{ is decided by some NTM within space } \mathrm{O}(2^{n^k}) \text{ where } k \text{ is a constant}\}$

NPSPACE = PSPACE, NEXPSPACE = EXPSPACE (surprising!)

Whether NL = L is unknown.

Relations Between Considered Classes

Theorem

$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE$

Proof

- Any TM is also an NTM.
  $\Rightarrow L \subseteq NL$, $P \subseteq NP$, and $EXPTIME \subseteq NEXPTIME$
- Recall Theorems (A) and (B).
  $\Rightarrow NP \subseteq PSPACE$ and $NEXPTIME \subseteq EXPSPACE$.

Remark. It is unknown if any of the inclusions is proper. However, it can be shown that in case of an exponential gap, the inclusion is indeed proper, i.e., $L \subset PSPACE$, $P \subset EXPTIME$, etc.

co-Problems

Definition

Any complexity class $\mathcal{C}$ has its complementary class denoted by co-$\mathcal{C}$ and defined as follows.

- For every language L in Σ^*, let $\overline{L}$ denote its complement i.e. $\overline{L} = \Sigma^* \setminus L$.
- Then co-$\mathcal{C}$ is defined as co-$\mathcal{C} = \{\overline{L} \mid L \in \mathcal{C}\}$.

Remark

- Examples. co-NL, co-NP, co-NEXPTIME, etc.
- Deterministic complexity classes $\mathcal{C}$ coincide with their co-class co-$\mathcal{C}$, e.g.: P = co-P.
- In contrast, nondeterministic complexity classes $\mathcal{C}$ may be different from their co-class co-$\mathcal{C}$, e.g.: It is unknown whether NP = co-NP or NP $\neq$ co-NP (the latter is considered more likely).

Learning Objectives

- Definition of Turing machines.
- Turing machines as a reasonable model of computation.
- Formal definition of "deterministic" complexity classes P, EXPTIME, L, PSPACE, EXPSPACE.
- Solving problems with Turing machines.
  (Decision problems can be considered as languages!)
- (Strengthening of) the Church-Turing Thesis
- Nondeterministic Turing machines. Differences between deterministic and nondeterministic TMs
- Nondeterminism as "guess and check" algorithms
- Definitions of NL, NP, NEXPTIME via nondeterministic TMs.
- The definition of complementary problems.