Complexity of Algorithms

Stefan Hetzl
Laboratoire Preuves, Programmes et Systèmes (PPS)
Université Paris Diderot – Paris 7

7th International Tbilisi Summer School in Logic and Language

Tbilisi, Georgia

September 2011

Outline

- Algorithmic problems
- Sorting algorithms
- Assymptotic run-time of algorithms
- Decision problems, Turing-machines
- The class **P**
- Propositional Logic, non-deterministic computation, **NP**
- Reductions
- Some graph theory
- **NP**-completeness
- Circuits
- Cook-Levin theorem: `SAT` is **NP**-complete

Algorithmic Problems

1. An algorithmic problem: Sorting:

   Input: A list $x_1, \ldots, x_n$ of natural numbers

   Output: A list $x_{\pi(1)}, \ldots, x_{\pi(n)}$ where π is a permutation such that $x_{\pi(i)} \leq x_{\pi(i+1)}$

   mathematically: a function

2. Solution: an algorithm

   mathematically: a program, formalised later

Selection Sort

for $i = 1 \rightarrow n - 1$ **do**
  $m \leftarrow i$;
  for $j = i + 1 \rightarrow n$ **do**
    if $A[j] < A[m]$ **then**
      $m \leftarrow j$;
    end if
  end for
  $t \leftarrow A[m]$;
  $A[m] \leftarrow A[i]$;
  $A[i] \leftarrow t$;
end for

Merge Sort

mergesort(l,r)

if $r - l > 1$ **then**

$m \leftarrow (r + l)$ div 2;

mergesort(l, m); mergesort$(m + 1, r)$;

for $i = m \rightarrow l$ **do** $B[i] \leftarrow A[i]$; **endfor**

for $j = m + 1 \rightarrow r$ **do** $B[r + m + 1 - j] \leftarrow A[j]$; **endfor**

for $k = l \rightarrow r$ **do**

if $B[i] < B[j]$ **then**

$A[k] \leftarrow B[i]; i \leftarrow i + 1;$

else

$A[k] \leftarrow B[j]; j \leftarrow j - 1;$

end if

end for

end if

Complexity of Algorithms

- Empirical run-time:

average run-time	Seq1	Seq2	Test1
Selection Sort	1.2ms	3.5ms	12.3ms
Merge Sort	1.3ms	2.8ms	4.5ms

  on an Athlon Dual-Core X2 250 2x3GHz

- Assymptotic complexity analysis:

  Use of resources depending on input size

- Resources: **time**, space, energy,...
- **Worst-case** / Average-case
- **Definition.** Let $f, g : \mathbb{N} \to \mathbb{N}$. We write $g \in O(f)$ if there are $k, n_0 \in \mathbb{N}$ s.t. for all $n \geq n_0$: $g(n) < k \cdot f(n)$.

  "g is $O(f)$"

Outline

- √ Algorithmic problems
- √ Sorting algorithms
- √ Assymptotic run-time of algorithms
- Decision problems, Turing-machines
- The class **P**
- Propositional Logic, non-deterministic computation, **NP**
- Reductions
- Some graph theory
- **NP**-completeness
- Circuits
- Cook-Levin theorem: `SAT` is **NP**-complete

Computation, more formally

- Formal algorithms $\Rightarrow$ a machine model
- Fix alphabet to $0, 1$.
- Encoding
- Fix possible results to “yes” and “no”.
- **Definition.** A set $L \in \{0,1\}^*$ is called *language*. Its decision problem is:
  Input: $x \in \{0,1\}^*$
  Output: “yes” if $x \in L$ and “no” otherwise
- $\texttt{PALINDROME}$:
  Input: $(x_1, \ldots, x_n) = x \in \{0,1\}^*$
  Output: “yes” if $(x_1, \ldots, x_n) = (x_n, \ldots, x_1)$, “no” otherwise

Turing Machines (1/2)

- A Turing machine works on a tape using a cursor
- Symbols 0, 1, blank ␣ and first $\triangleright$
- Initial configuration for input $x = (x_1, \ldots, x_n) \in \{0,1\}^*$

$$\begin{array}{cccccccc} \triangleright & x_1 & x_2 & \cdots & x_n & \sqcup & \sqcup & \ldots \\ \uparrow & & & & & & & \end{array}$$

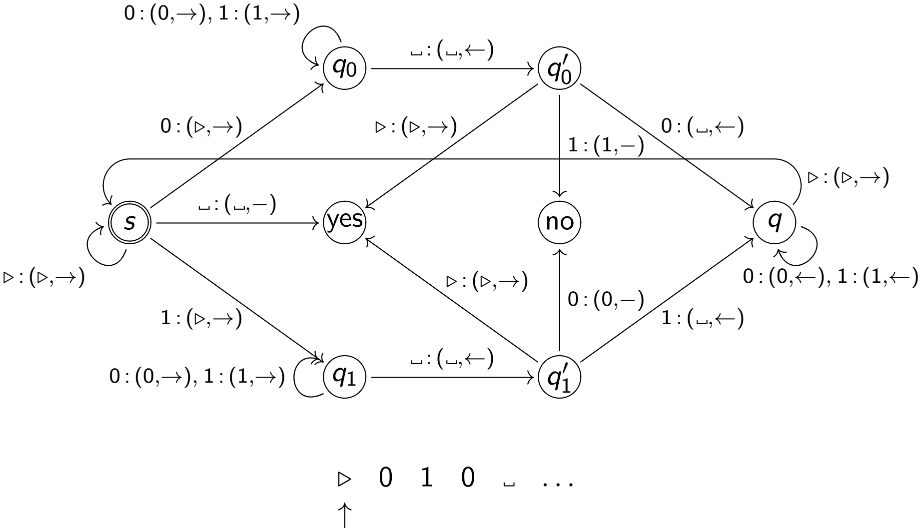
- **Definition.** A Turing machine is a tuple $M = (K, s, \delta)$ where
  - K is a finite *set of states*
  - $s \in K$ is the *initial state*
  - δ is the *transition function*
    $\delta : \; K \times \{0, 1, \sqcup, \triangleright\} \longrightarrow (K \cup \{\text{yes}, \text{no}\}) \times \{0, 1, \sqcup, \triangleright\} \times \{\leftarrow, \rightarrow, -\}$

Turing Machines (2/2)

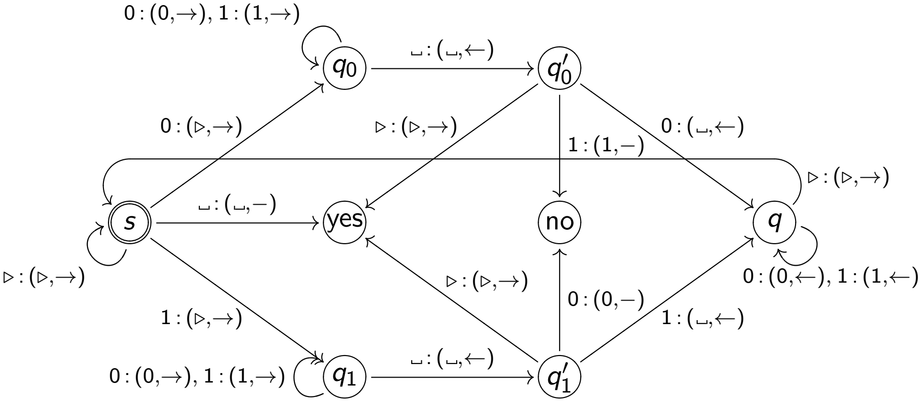
- **Definition.** A *configuration* is a tuple (q, u, v) where
  - q is a state
  - u is the tape left of and including the cursor
  - v is the tape right of the cursor

  The *initial configuration* for $x \in \{0,1\}^*$ is $(s, \triangleright, x)$.
- **Definition.** $(q, u, v) \rightarrow^{M^1} (q', u', v')$ if the configuration (q, u, v) yields the configuration (q', u', v') in one step. $\rightarrow^{M}$ is the transitive closure of $\rightarrow^{M^1}$
- **Definition.** L language, M machine. M *decides* L if for all $x \in \{0,1\}^*$:
  - $x \in L$ implies that $(s, \triangleright, x) \rightarrow^{M} (\text{yes}, u, v)$ for some u, v, and
  - $x \notin L$ implies that $(s, \triangleright, x) \rightarrow^{M} (\text{no}, u, v)$ for some u, v.
- **Definition.** M decides L in time $f(n)$ if M halts after at most $f(n)$ steps for all $x \in \{0,1\}^n$.

A Turing Machine for Palindromes

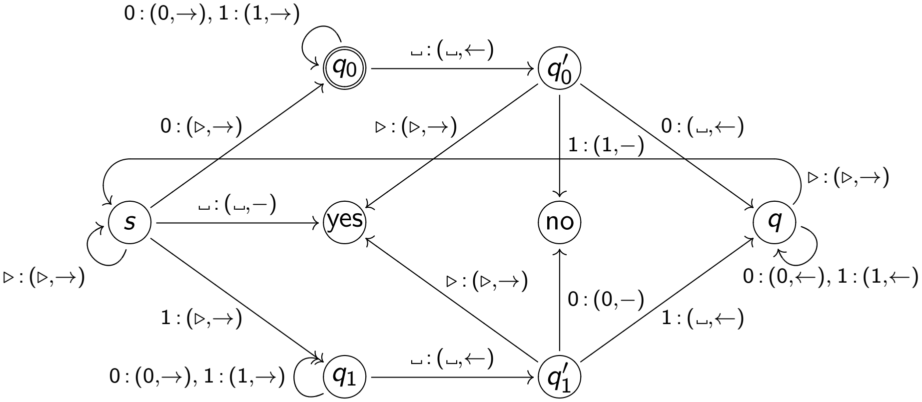


A Turing Machine for Palindromes



$$\triangleright \quad 0 \quad 1 \quad 0 \quad ⊔ \quad \ldots$$
$$\uparrow$$

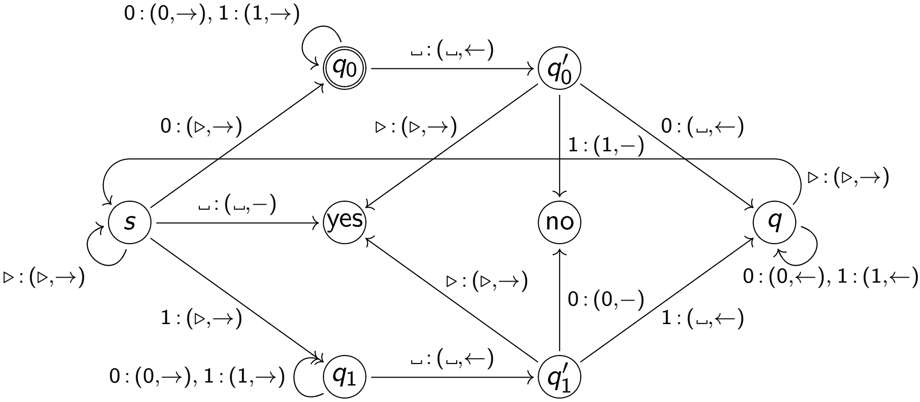
A Turing Machine for Palindromes



▷ ▷ 1 0 ␣ . . .

↑

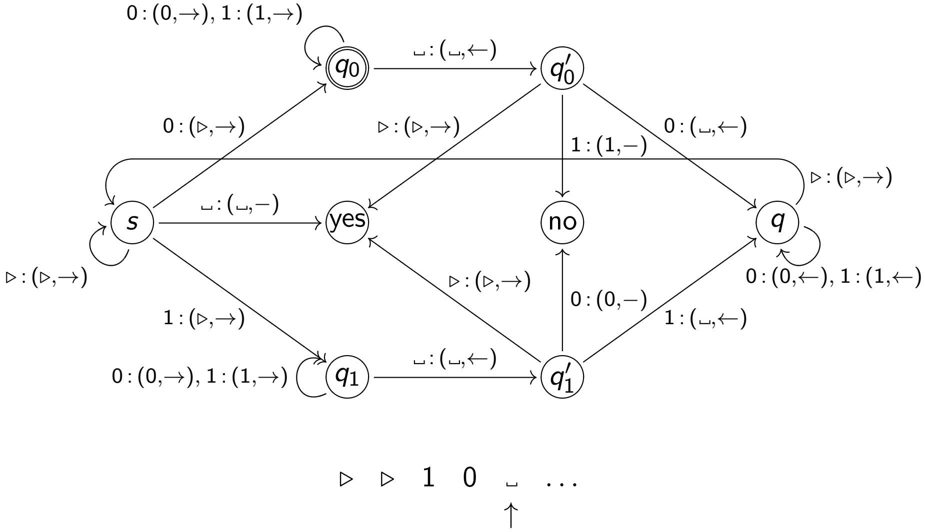
A Turing Machine for Palindromes



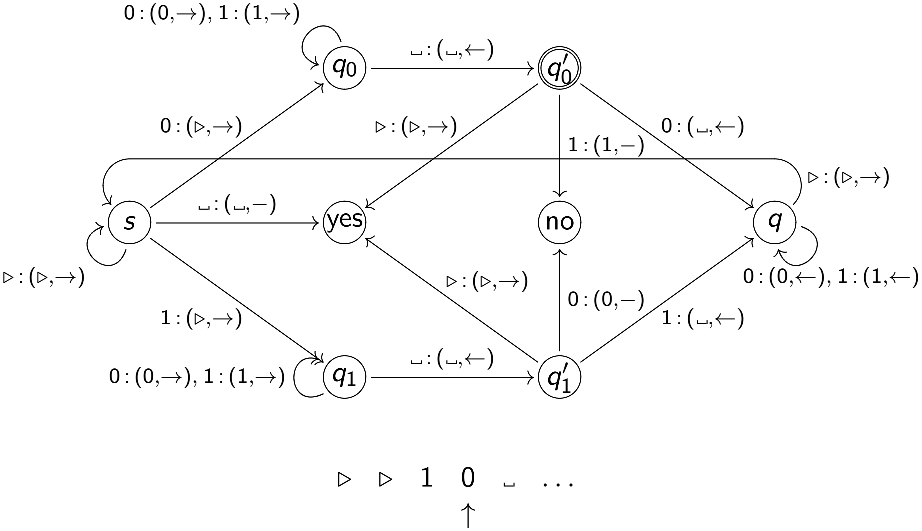
▷ ▷ 1 0 ␣ ...

↑

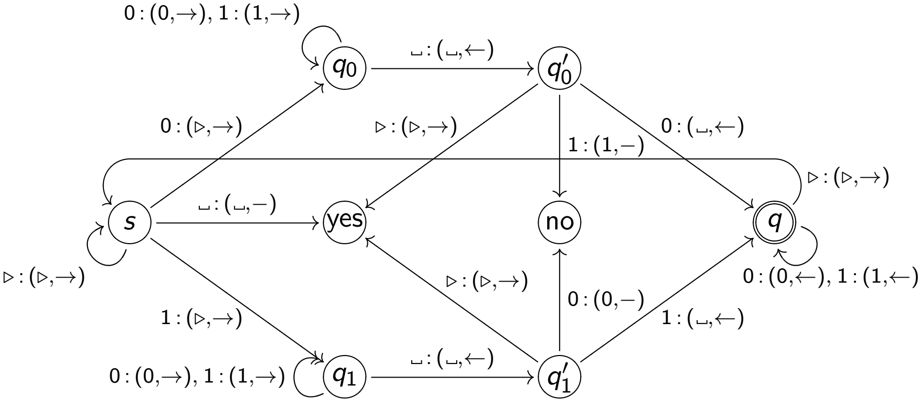
A Turing Machine for Palindromes



A Turing Machine for Palindromes



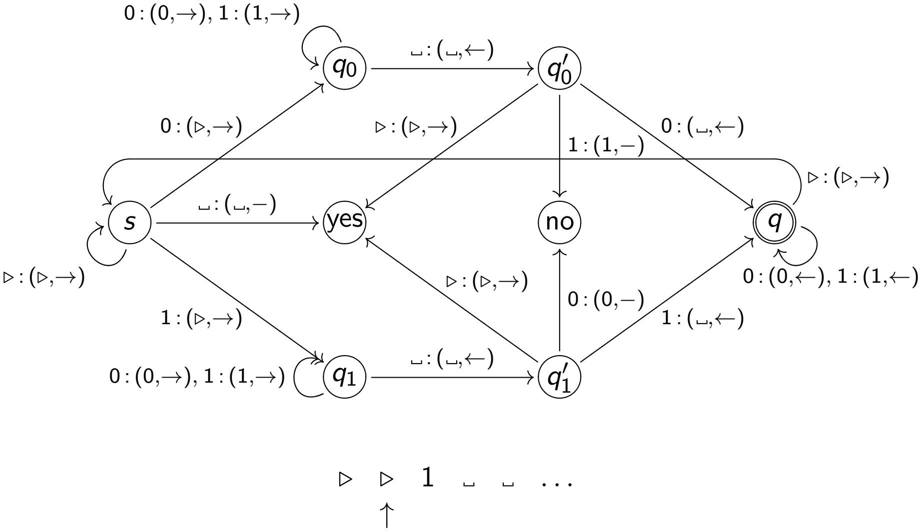
A Turing Machine for Palindromes



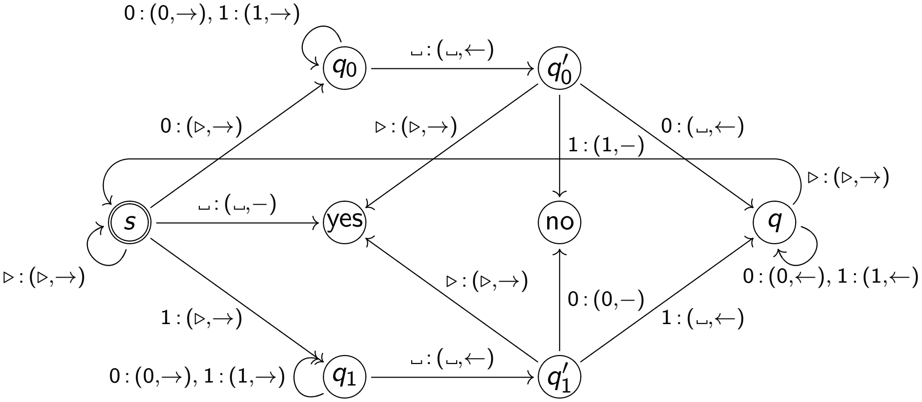
▷ ▷ 1 ␣ ␣ . . .

↑

A Turing Machine for Palindromes



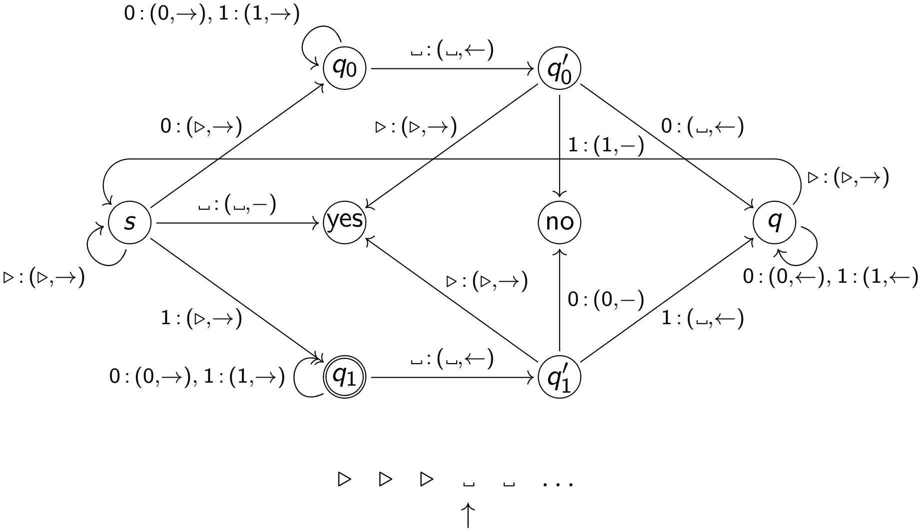
A Turing Machine for Palindromes



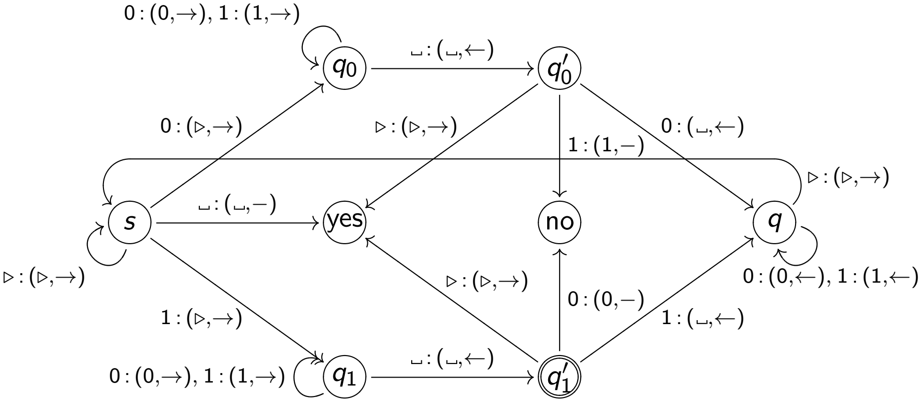
$\triangleright \quad \triangleright \quad 1 \quad \sqcup \quad \sqcup \quad \ldots$

$\uparrow$

A Turing Machine for Palindromes



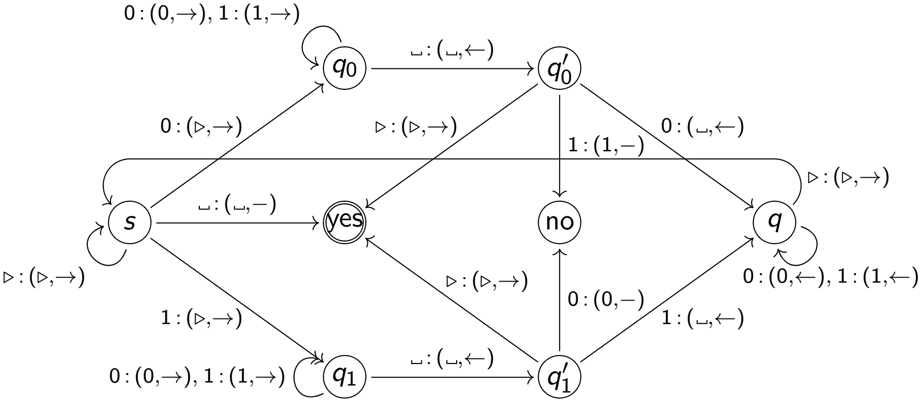
A Turing Machine for Palindromes

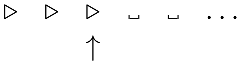


▷ ▷ ▷ ␣ ␣ . . .

↑

A Turing Machine for Palindromes





From Pseudo-Code to Turing Machines

- k-tape Turing machines
- Quadratic simulation of k-tape machine by 1-tape machine
- Translation of Pseudo-Code to Turing machine
- All known computation models equivalent up to polynomial
- **Definition. P** is the set of all languages L s.t. there is a Turing machine that decides L in polynomial time.
- Example: $\texttt{PALINDROME} \in \mathbf{P}$
- "**P** is the set of languages decidable in polynomial time"

Outline

- √ Algorithmic problems
- √ Sorting algorithms
- √ Assymptotic run-time of algorithms
- √ Decision problems, Turing-machines
- √ The class **P**
- Propositional Logic, non-deterministic computation, **NP**
- Reductions
- Some graph theory
- **NP**-completeness
- Circuits
- Cook-Levin theorem: `SAT` is **NP**-complete

Propositional Logic (1/2)

- Propositional letters: $p_1, p_2, \ldots$
- Connectives $\neg, \wedge, \vee$
- Formula: Built from p_i and connectives
- Interpretation $I : \{p_1, \ldots, p_n\} \rightarrow \{0, 1\}$
  $I(\neg\varphi) = 1 - I(\varphi)$
  $I(\varphi_1 \wedge \varphi_2) = \min\{I(\varphi_1), I(\varphi_2)\}$
  $I(\varphi_1 \vee \varphi_2) = \max\{I(\varphi_1), I(\varphi_2)\}$
- A formula φ over $p_1, \ldots, p_n$ is
  satisfiable if there is interpretation I s.t. $I(\varphi) = 1$
  tautological if for all interpretations I: $I(\varphi) = 1$

Propositional Logic (2/2)

- Conjunctive normal form (CNF):

$$\bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} A_{i,j}$$

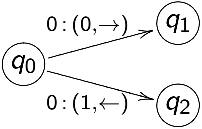
where $A_{i,j}$ is either p_k or $\neg p_k$

- SAT:
  Input: formula φ in CNF
  Output: "yes" if φ is satisfiable and "no" otherwise
- brute-force SAT-algorithm: try all interpretations (truth-table)

Does there exist a faster algoritm?

Non-Deterministic Computation (1/2)

- Deterministic Turing-machine:
  $\delta : K \times \{0, 1, \sqcup, \triangleright\} \longrightarrow (K \cup \{\text{yes}, \text{no}\}) \times \{0, 1, \sqcup, \triangleright\} \times \{\leftarrow, \rightarrow, -\}$
  total transition **function**
- Non-determinism



  transition **relation**
- **Definition.** A *non-deterministic Turing machine* is a tuple $N = (K, s, \Delta)$ where
  - K is a finite *set of states*
  - $s \in K$ is the *initial state*
  - $\Delta \subseteq K \times \{0, 1, \sqcup, \triangleright\} \times (K \cup \{\text{yes}, \text{no}\}) \times \{0, 1, \sqcup, \triangleright\} \times \{\leftarrow, \rightarrow, -\}$ is the *transition relation*

Non-Deterministic Computation (2/2)

- Configuration (q, u, v)
- **Definition.** $(q, u, v) \rightarrow^{N^1} (q', u', v')$ if the configuration (q, u, v) may yield the configuration (q', u', v') in one step. $\rightarrow^{N}$ is the transitive closure of $\rightarrow^{N^1}$
- **Definition.** L language, N non-deterministic machine. N *decides* L if for all $x \in \{0, 1\}^*$:
  - $x \in L$ iff $(s, \triangleright, x) \rightarrow^{N}$ (yes, u, v) for some u, v.
- **Definition.** N decides L in time $f(n)$ if every computation path of N halts in at most $f(n)$ steps.
- **Lemma.** Balanced Non-determinsim.

The class **NP**

- **Definition. NP** is the class of languages decidable in polynomial-time by a **non-deterministic** Turing machine.
- **Theorem.** $L \in \mathbf{NP}$ iff there is polynomial-time deterministic Turing machine M and a $c \in \mathbb{N}$ s.t. for all $x \in \{0,1\}^n$:

$$x \in L \quad \Leftrightarrow \quad \exists u \in \{0,1\}^{n^c} \text{ s.t. } M \text{ accepts } (x, u)$$

- $\mathtt{SAT} \in \mathbf{NP}$
- **P**: solution can be found in polynomial time
  NP: solution can be checked in polynomial time
  "guess-and-check"
- $\mathbf{P} \subseteq \mathbf{NP}$

Outline

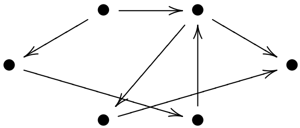
- √ Algorithmic problems
- √ Sorting algorithms
- √ Assymptotic run-time of algorithms
- √ Decision problems, Turing-machines
- √ The class **P**
- √ Propositional Logic, non-deterministic computation, **NP**
- Reductions
- Some graph theory
- **NP**-completeness
- Circuits
- Cook-Levin theorem: `SAT` is **NP**-complete

Reductions

- **Definition.** A language L_1 is *reducible* to a language L_2 if there is a polynomial-time function $R : \{0,1\}^* \to \{0,1\}^*$ s.t. $x \in L_1$ iff $R(x) \in L_2$. This is written as $L_1 \leq_p L_2$.
- $\leq_p$ is transitive

Some Graph Theory

- A graph is a set of vertices V together with a set of edges $E \subseteq V \times V$, e.g.



- A path is a list of vertices $v_1, \ldots, v_n$ s.t. $(v_i, v_{i+1}) \in E$

- Let $G = (V, E)$ be a graph. A *Hamiltonian path* is a path that contains each vertex exactly once.
- HAMILTONIAN PATH
  Input: a graph G
  Output: "yes" if G has a Hamiltonian path, "no" otherwise
- Example:

HAMILTONIAN PATH vs. SAT

- HAMILTONIAN PATH $\leq_p$ SAT
- SAT $\leq_p$ HAMILTONIAN PATH

HAMILTONIAN PATH vs. SAT

- HAMILTONIAN PATH $\leq_p$ SAT ✓
- SAT $\leq_p$ HAMILTONIAN PATH

HAMILTONIAN PATH vs. SAT

- HAMILTONIAN PATH $\leq_p$ SAT ✓
- SAT $\leq_p$ HAMILTONIAN PATH ✓

NP-Completeness

- **Definition.** A language L is called **NP**-*hard* if $L' \leq_p L$ for all $L' \in$ **NP**.
- **Definition.** A language L is called **NP**-*complete* if $L \in$ **NP** and L is **NP**-hard.

! **Cook-Levin Theorem.** `SAT` is **NP**-complete.
Proof. will use `CIRCUIT SAT`

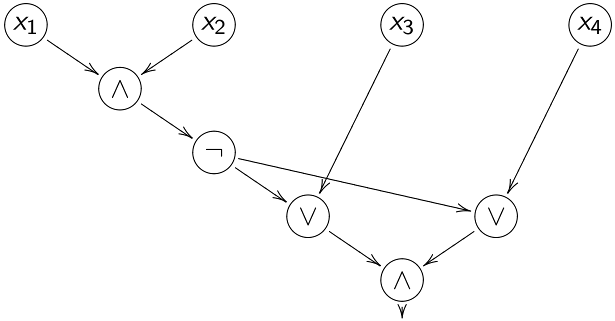
- **Corollary.** **P** = **NP** iff there is a polynomial-time algorithm for `SAT`.
- **Corollary.** `HAMILTONIAN PATH` is **NP**-complete.
- **Corollary.** **P** = **NP** iff there is a polynomial-time algorithm for `HAMILTONIAN PATH`.
- ...

Outline

- √ Algorithmic problems
- √ Sorting algorithms
- √ Assymptotic run-time of algorithms
- √ Decision problems, Turing-machines
- √ The class **P**
- √ Propositional Logic, non-deterministic computation, **NP**
- √ Reductions
- √ Some graph theory
- √ **NP**-completeness
- ▸ Circuits
- ▸ Cook-Levin theorem: `SAT` is **NP**-complete

Boolean Circuits

- **Definition.** A *boolean circuit* is a directed acyclic graph with
  - Input nodes: labelled by variables $x_1, x_2, \ldots$
  - Other nodes: labelled by operations $\wedge$, $\vee$ and $\neg$
  - One output node
- Example:



- Represents a formula, a boolean function

CIRCUIT SAT

- CIRCUIT SAT

  Input: A circuit $C(x_1, \dots, x_n)$
  Output: "yes" if there are $a_1, \dots, a_n \in \{0, 1\}$ s.t. $C(a_1, \dots, a_n) = 1$
  "no" otherwise

- **Theorem.** SAT $=_p$ CIRCUITSAT.
  Proof.
  - SAT $\leq_p$ CIRCUIT SAT: every CNF is a circuit, use

    $$\text{id} : \text{SAT} \to \text{CIRCUIT SAT}$$

    as reduction.
  - CIRCUIT SAT $\leq_p$ SAT: obtain CNF from circuit vertex by vertex.

Computation Tables

s :	$\underline{\triangleright}$	0	1	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
s :	$\triangleright$	$\underline{0}$	1	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q_0 :	$\triangleright$	$\triangleright$	$\underline{1}$	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q_0 :	$\triangleright$	$\triangleright$	1	$\underline{0}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q_0 :	$\triangleright$	$\triangleright$	1	0	$\underline{\sqcup}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q_0' :	$\triangleright$	$\triangleright$	1	$\underline{0}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q :	$\triangleright$	$\triangleright$	$\underline{1}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q :	$\triangleright$	$\underline{\triangleright}$	1	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
s :	$\triangleright$	$\triangleright$	$\underline{1}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q_1 :	$\triangleright$	$\triangleright$	$\triangleright$	$\underline{\sqcup}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q_1' :	$\triangleright$	$\triangleright$	$\underline{\triangleright}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
yes :	$\triangleright$	$\triangleright$	$\triangleright$	$\underline{\sqcup}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$

Computation Tables and Circuits

Lemma. Let M be a deterministic Turing machine with run-time bounded by $f(n)$. Then there is a sequence of circuits $C_n(x_1, \ldots, x_n)$ with $|C_n| = O(f(n)^2)$ and $C_n(\bar{s}) = M(\bar{s})$ for all $\bar{s} \in \{0,1\}^n$.

Proof.

- Codes for states, symbols and cursor position
- Input line
- Circuit C for local changes
- Circuit S for state changes
- $f(n) \times f(n)$ computation table
- Output line

The Non-Deterministic Case

Lemma. Let N be a **non**-deterministic Turing machine with run-time bounded by $f(n)$. Then there is a sequence of circuits $C_n(x_1, \ldots, x_n, y_1, \ldots, y_{f(n)})$ with $|C_n| = O(f(n)^2)$ and for all $\bar{s} \in \{0,1\}^n$: N accepts $\bar{s}$ iff there is $\bar{y} \in \{0,1\}^{f(n)}$ s.t. $C_n(\bar{s}, \bar{y}) = 1$.
Proof.

- Construction as before
- Plus: y_i for simulating non-deterministic choice at step i

The Cook-Levin Theorem

Theorem. $\texttt{SAT}$ is **NP**-complete.

Proof.

- Membership: $\texttt{SAT} \in \textbf{NP}$ ✓
- Hardness:
  - Let $L \in \textbf{NP}$, then there is non-deterministic machine N deciding L in polynomial time $f(n)$.
  - By Lemma there is sequence of circuits C_n s.t. for all $\bar{a} \in \{0,1\}^n$: $\bar{a} \in L$ iff there is $\bar{b} \in \{0,1\}^{f(n)}$ s.t. $C_n(\bar{a}, \bar{b}) = 1$.
  - The mapping

$$L \to \texttt{CIRCUIT SAT}, \quad \bar{a} \in \{0,1\}^n \mapsto C_n(\bar{a}, \bar{y})$$

    is a reduction.
  - So $L \leq_p \texttt{CIRCUIT SAT} \leq_p \texttt{SAT}$.

Summary & Outlook

- diffcult problems: show that a problem is **not** in a class
- **NP**-completeness proofs as substitute

- space complexity
- **coNP**
- $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PH} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$
- time and space hierachy theorems

- most famous open problem:
  Is there a polynomial algorithm for SAT?
  $\mathbf{P} =^? \mathbf{NP}$ one of the Clay Millenium problems, 1 mio. \$

References

- C. Papadimitriou: *Computational Complexity*
- S. Arora, B. Barak: *Computational Complexity: A Modern Approach*
- M. Garey, D. Johnson: *Computers and Intractability. A Guide to the theory of NP-completeness*
- J. Hopcroft, J. Ullman: *Introduction to Automata Theory, Languages, and Computation*
- R. Sedgewick: *Algorithms*