Algorithms & Complexity
Lectures 6&7: (Basic Flavors) of Complexity Theory

November 5 and 26, 2019

CentraleSupélec / ESSEC Business School

Dimo Brockhoff

Inria Saclay – Ile-de-France

Discussion Home Exercise

Exercise 1: Mutation Operators for the Knapsack problem

Disadvantage of a 1-bit flip: if knapsack full, cannot improve anymore without degrading the function value (no exchanges of items are possible)

Suggestion: use combination of 1- and 2-bit flip

Other option: crossover, but then a (greedy) repair mechanism is needed to make the offspring fulfill the constraint to fit into the knapsack

Exercise 2: Roulette Wheel Selection

$f(x) = x^2$, solutions a ($x = 1$), b ($x = 2$) and c ($x = 3$)

a) probability of selecting each individual with roulette wheel selection:

a: 1/(1+4+9) = 1/14= 0.0714…
b: 4/(1+4+9) = 4/14 = 4x prob(a)
c: 9/(1+4+9) = 9/14 = 9x prob(a)

b) same for g(x) = f(x) + 100:

a: 101/(101+104+109)=101/314=0.321656…
b: 104/314 = 0.33121… = about 3% more than prob(a)
c: 109/314 = 0.34713… = about 8% more than prob(a)

Discussion Home Exercise

Exercise 2: Roulette Wheel Selection

$f(x) = x^2$, solutions a ($x = 1$), b ($x = 2$) and c ($x = 3$)

c) probabilities of selecting each solution with binary tournaments, (considering maximization):

each solution picked with probability 1/3, possible outcomes:

- a selected: aa
- b selected: ab, ba, bb
- c selected: ac, bc, ca, cb, cc

Hence: prob(a) = 1/9, prob(b) = 3/9 = 1/3, prob(c) = 5/9

Independently of whether we use $f(x)$ or $g(x)$!

d) Which of the selection operators do you favor? Why?

- due to invariance: the tournament selection

Important Scientific Concept: Invariance

Many examples:

- Polaris (the North Star) has a fixed position (= invariant to movements of earth)
- speed of light independent of coordinate system
- assertions in code are implicit invariances

- and an algorithm is invariant under a given transformation of a problem if it behaves the same on both of them (technically, the definition is more involved)
  - example: rank-based algorithms are invariant under monotone transformations of the objective function

The grand aim of all science is to cover the greatest number of empirical facts by logical deduction from the smallest number of hypotheses or axioms.

Albert Einstein

Discussion Home Exercise

Exercise 3: Pure Random Search (PRS)

samples always uniformly at random in (finite) search space independent of objective function $f(x)$!

For $\boldsymbol{x} \in \{0,1\}^n$, *the function* OM *is defined as*

$$f_{\mathrm{OM}}(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{x}_i$$

For $\boldsymbol{x} \in \{0,1\}^n$, *the function* LO *is defined as*

$$f_{\mathrm{LO}}(\boldsymbol{x}) = \sum_{i=1}^{n} \prod_{j=1}^{i} \boldsymbol{x}_j$$

a) What do those functions formalize?

OM = ONEMAX, number of 1s in the bitstring

LO = LEADINGONES, length of leading block of consecutive 1s

Exercise 3: Pure Random Search (PRS)

For $\boldsymbol{x} \in \{0,1\}^n$*, the function* OM *is defined as*

$$f_{\mathrm{OM}}(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{x}_i$$

For $\boldsymbol{x} \in \{0,1\}^n$*, the function* LO *is defined as*

$$f_{\mathrm{LO}}(\boldsymbol{x}) = \sum_{i=1}^{n} \prod_{j=1}^{i} \boldsymbol{x}_j$$

b) Only optimum: $x = (1, \dots, 1) \in \mathbb{R}^n$
corresponding optimal function value: n

Exercise 3: Pure Random Search (PRS)

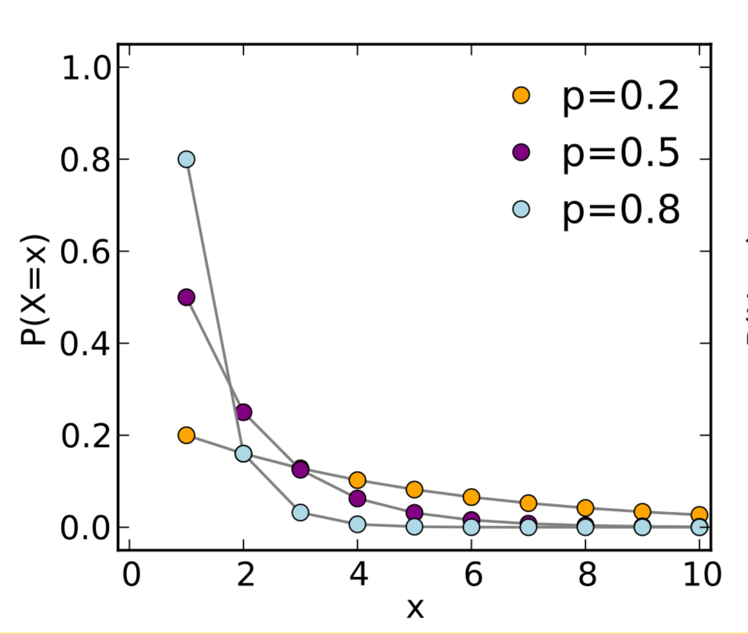
c) Expected optimization time of PRS?

→ independent of f!

→ probability to reach optimum in the current step = 1/(#search points) = $1/2^n$

→ Geometric distribution:
* Bernoulli trials of probability p
* expected number of trials to get "success" = 1/p

→ Here: expected number of samples to f until optimum is found = 2^n



Discussion Home Exercise

What if we go for a slightly more advanced algorithm: RLS?

Randomized local search (RLS):

→ Start from a uniformly sampled point $x \in \{0,1\}^n$

→ While happy:

→ $x' \leftarrow x$

→ flip a randomly chosen bit in x'

→ if $f(x') \geq f(x)$:

→ $x \leftarrow x'$

Expected time to reach the optimum?

- number of 1 bits never decreases
- and is between 0 and n (n being the optimum)
- increase number of 1 bits by flipping one of the remaining i zeros (probability: i/n), expected waiting time for this: n/i
- in total: maximally $\sum_{i=0}^{n} \frac{n}{i} = n \sum_{i=0}^{n} \frac{1}{i} = \Theta(n \log n)$ steps to find opt.

Discussio

What if we g 

m: RLS?

Randomiz

→ Start fr

→ While

→ x' ←

→ flip

→ if f

→

Expected time

- number of
- and is bet
- increase n … aining i zeros (probabilit
- in total: $\sum_{i}^{n}$

Course Overview

Thu		Topic
Thu, 12.09.2019	PM	Introduction, Combinatorics, O-notation, data structures
Tue, 24.09.2019	PM	Sorting algorithms I
Tue, 1.10.2019	PM	Sorting algorithms II, recursive algorithms
Tue, 8.10.2019	PM	Recursive and Greedy Algorithms
Tue, 15.10.2019	PM	Dynamic programming
Thu, 31.10.2019	AM	Randomized Algorithms and Blackbox Optimization
➨ Tue, 5.11.2019	PM	Complexity theory I
Tue, 26.11.2019	PM	Complexity theory II
Tue, 17.12.2019	AM	Exam (written)

back to

Randomized Algorithms and Blackbox Optimization

A Canonical Genetic Algorithm

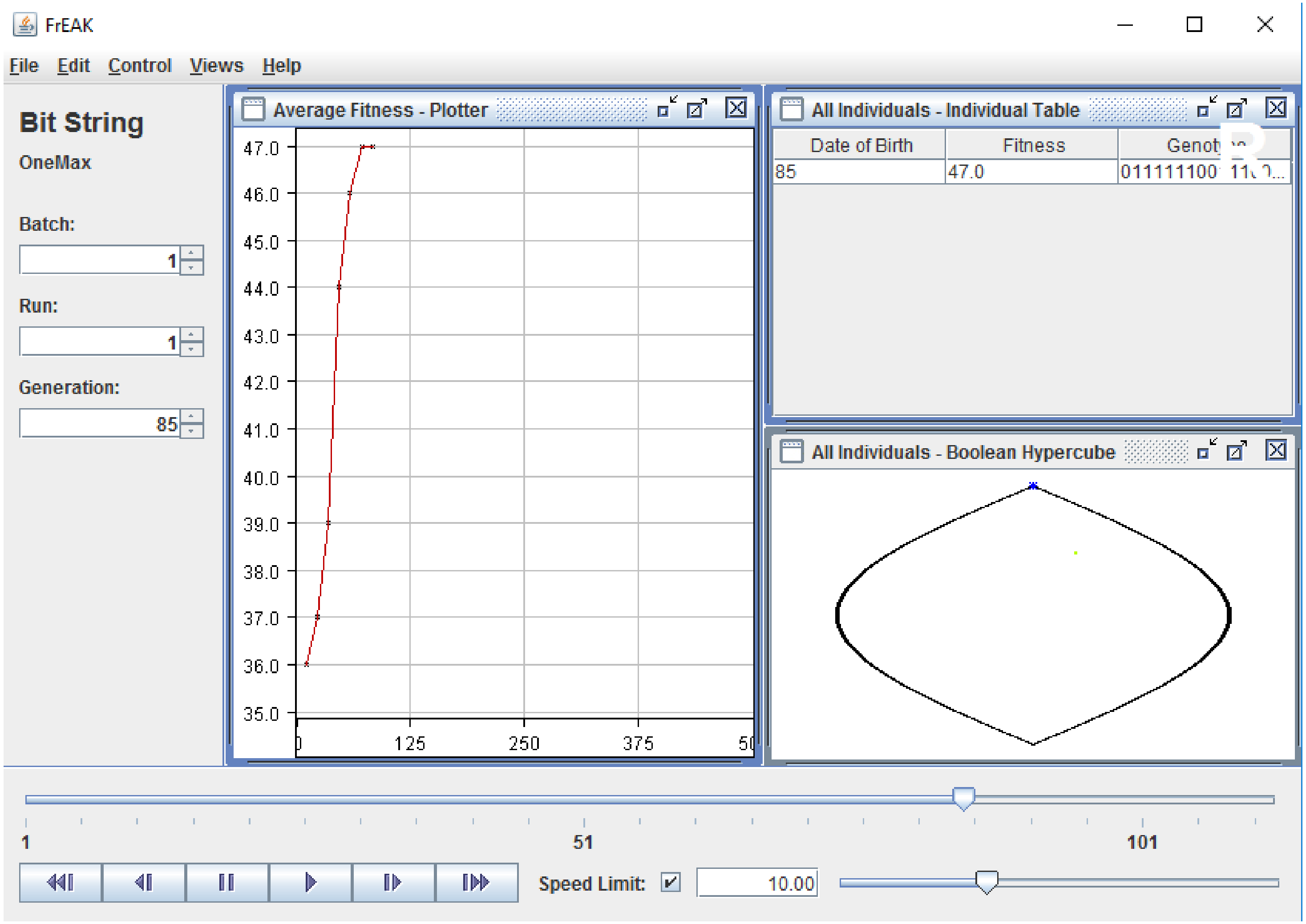
- binary search space, maximization
- uniform initialization
- generational cycle:
  - evaluation of solutions
  - mating selection (e.g. roulette wheel)
  - crossover (e.g. 1-point)
  - environmental selection (e.g. plus-selection)

You may ask: how does this fit
into the stochastic search template?
it does: population contained in state θ,
but update function difficult to write down

FrEAK

If you want to play around a bit with these algorithms:

- https://sourceforge.net/projects/freak427/



Estimation of Distribution Algorithms

- Estimation of Distribution Algorithms (EDAs) fit more obviously into the search template
- here, example of the compact Genetic Algorithm (cGA)
  - search space: $\Omega = \{0,1\}^n$
  - probability distribution: Bernoulli
    - store for each bit a probability p_i to sample a 1
    - sample bit i with probability p_i to 1 and with $(1 - p_i)$ to 0

The Compact GA

Parameters: number of variables n, learning rate K (typically $= n$)

Init:

$p = \left(\frac{1}{2}, \frac{1}{2}, \dots, \frac{1}{2}\right) \in [0,1]^n$ # probabilities to sample new solutions

While happy:

create $S = (s_1, \dots, s_n)$ by sampling each s_i with probability p_i

create $S' = (s'_1, \dots, s'_n)$ by sampling each s'_i with probability p_i

evaluate S and S' on f

if $f(S) > f(S')$: # make sure that S is the better solution

$S, S' \leftarrow S', S$

update p parameter:

for $i \in \{1, \dots, n\}$:

$p_i \leftarrow \min\{\max\{p_i + (s_i - s'_i)/K, 1/n\}, 1 - 1/n\}$

return S

Potential Master's/PhD thesis projects

Potential Research Topics for Master's/PhD Theses

http://randopt.gforge.inria.fr/thesisprojects/

Trace: • start

Thesis Projects

[[start]]

Home

Welcome!

On this page, you will find various current technical and scientific projects in the field of stochastic blackbox optimization proposed by Anne Auger, Dimo Brockhoff, and Nikolaus Hansen at Inria. Depending on the subject, the projects can be Bachelor, Master's, or PhD theses, or related to internships and might be carried out in close relationship with external collaborators, including companies.

If you are interested in (stochastic) blackbox optimization but your favorite topic is not mentioned here, feel free to contact us personally. We might always have other topics in mind, which range from theoretical studies to algorithm design but which have not yet been formalized here.

Current Openings

- Stopping Criteria for Multiobjective Optimizers (Master's project)
- Various technical projects around the COCO platform (Internships/Bachelor)
- Large-scale Stochastic Black-box Optimization (Master's project)
- The Orbit Algorithm for Expensive Numerical Blackbox Problems (Bachelor/Master's project)

Previous Announcements

- Adaptive Stochastic Search Algorithms for Constrained Optimization (Master's thesis project)
- Data Mining Performance Results of Numerical Optimizers (Master's thesis project)
- General Constraint Handling in the Stochastic Numerical Optimization Algorithm CMA-ES (CIFRE PhD)
- Designing Variants of the Covariance Matrix Adaptation Evolution Strategy to Handle Multiobjective Blackbox Problems (CIFRE PhD)

Conclusions

- EAs are generic algorithms (randomized search heuristics, meta-heuristics, ...) for black box optimization

  no or almost no assumptions on the objective function

- They are typically less efficient than problem-specific (exact) algorithms in discrete domain (in terms of #funevals)

  but competitive in the continuous case

- Allow for an easy and rapid implementation and therefore to find good solutions fast

  easy (recommended!) to incorporate problem-specific knowledge to improve the algorithm

Conclusions

I hope it became clear...

...that heuristics is what we typically can afford in practice (no guarantees and no proofs)

...what are the main ideas behind evolutionary algorithms

...and that evolutionary algorithms and genetic algorithms are no synonyms

Complexity Theory

Motivation: Analyzing Algorithm Runtimes

- we want to analyze algorithms for discrete problems
- to be more precise: want to know runtime to find the optimum

Not realistic:

- do this for any input sequence
- do this for any machine, programming language, compiler, ...

Instead:

- abstract from a real implementation to the algorithm run on an abstract machine model

  [use a model which makes useful predictions in the real world]
- analyze the algorithm runtime for all instances of a given input size (worst case, average case, ...)

Motivation: Analyzing the Optimal Algorithms

- want to know how quick an optimal algorithm would run
  - how much slower is my own one?
- want to know the general difficulty of problems
  - why can't I find an efficient algorithm for my problem?

Complexity Theory

A part of theoretical computer science that is concerned about:

- comparison of (optimization) problems regarding their difficulty
- classes of difficulties
- computability in general

Complexity Theory: Lecture Overview

- deterministic machine models
- computability
  - an example of a problem which cannot be solved by a computer
- non-determinism and the class NP
- difficult problems:
  - the classes NP-complete, NP-hard, etc.
  - polynomial reductions
- the complexity zoo

Note: complexity theory is often a full lecture by itself!

Algorithm Runtimes in Reality

Algorithm runtimes depend on

- hardware (cpu, RAM, ...)
- the used programming language
- the used compiler/interpreter
- other load on the machine
- implementation "tricks" (running on GPU, compiler options, ...)

But still, we often make general statements like

- "Mergesort is a good sorting algorithm."
- "My algorithm is quicker than yours."
- "Algorithm A is the best possible algorithm for problem P."

how comes? what does it mean?

Abstractions for Algorithm Runtime Considerations

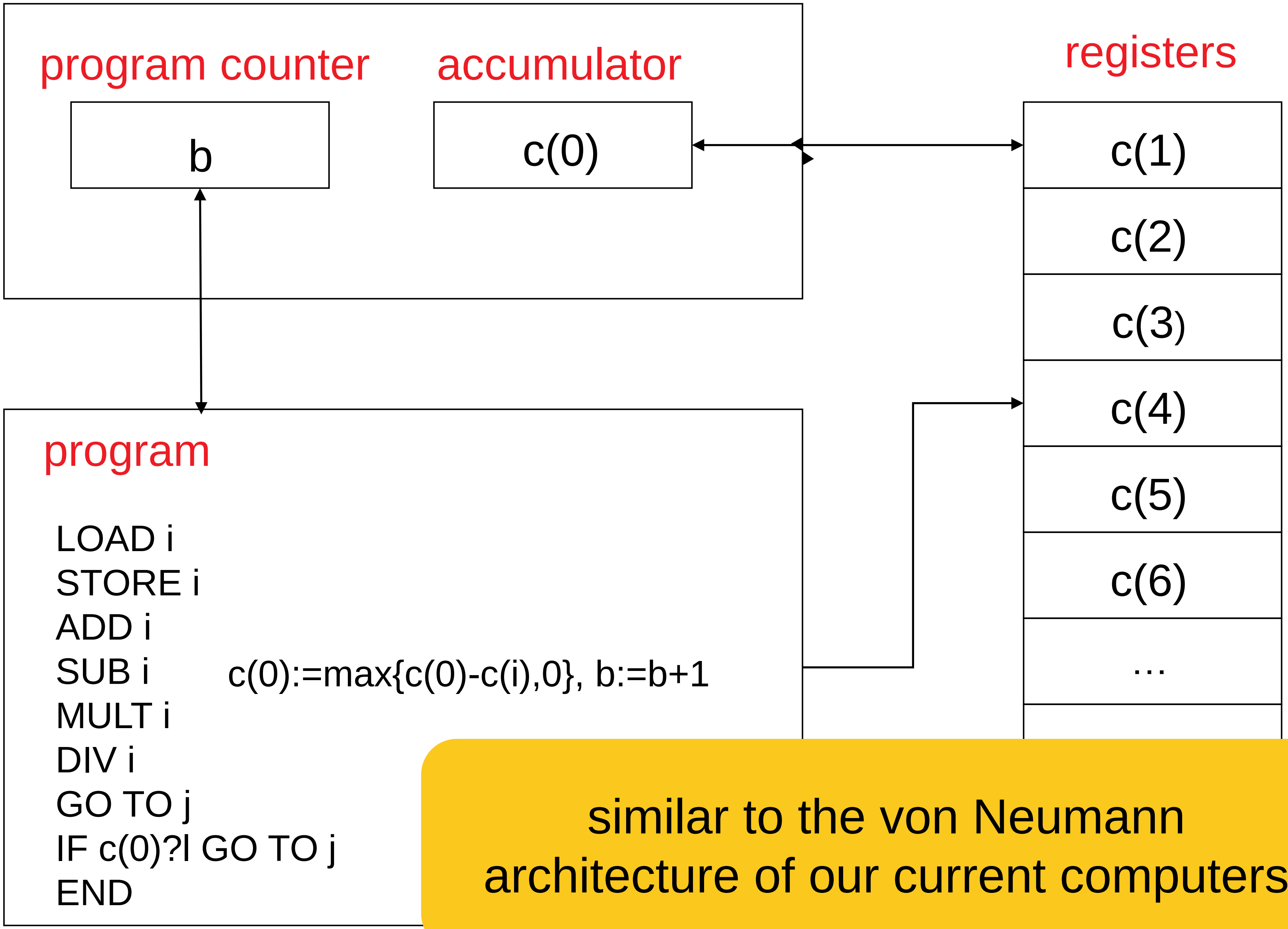
...because we abstract!

- for SORTING for example: number of comparisons as basic operation (actual runtime will again depend on hard- and software)
- often basic calculations as basic model (addition, multiplication, division, ...)
  - but what model is good?
  - are addition and multiplication e.g. equally difficult?

Important Aspects:

- relation to our real-world computers
- optimally, the choice of the model does not matter!

The Random Access Machine (RAM)



similar to the von Neumann architecture of our current computers

The Random Access Machine

is similar to the von Neumann architecture of our current computers

But:

- simpler (no pipelining, caches, ...)
- registers can contain non-negative natural numbers!

Last point not too much of a restriction:

- general natural numbers simulated by 2 registers
- rational numbers simulated by 4 registers

But probably too optimistic for measuring performance:

operations on arbitrarily large numbers might cost much more on an actual computer!

Cost Measures

Uniform Cost Measure:

- each operation costs 1

Logarithmic Cost Measure:

- each operation costs relative to the length of the arguments
- log(ARG) is cost measure if we assume binary representations of the numbers

Problem Complexity

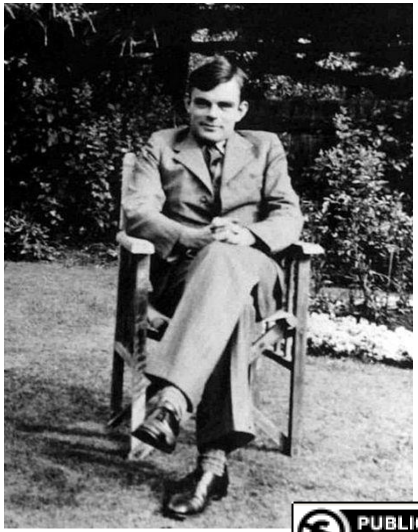
- for example for Random Access Machine and a given cost measure

Complexity of problem Π

= number of operations needed for an optimal algorithm to solve each instance of Π

- important question: how much does this complexity depend on the machine model and the cost measure?
- moreover, independent of the existance of actual computers?

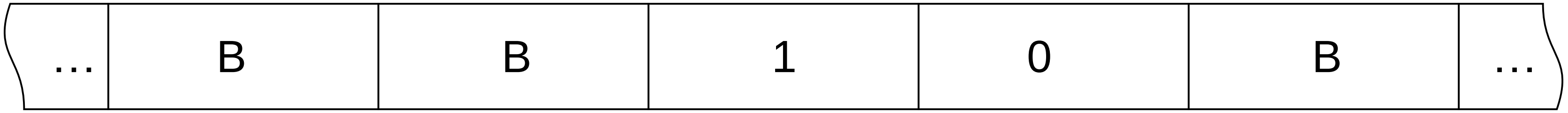
- Alan Turing (1912—1954)
- simplest ~~computer~~ model
  computation



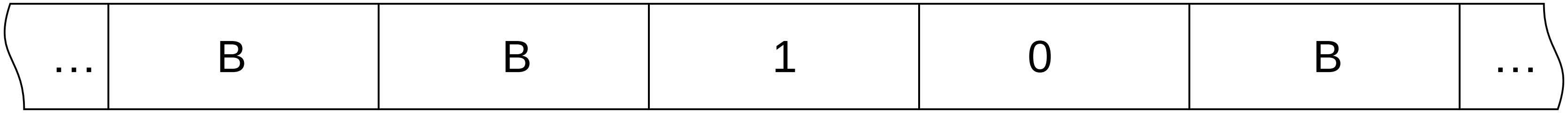
Formal definition:

$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

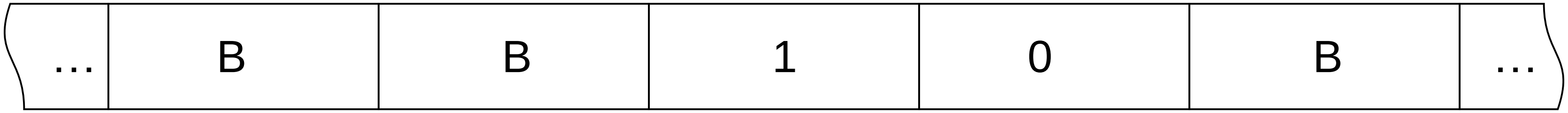
$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$


…
B
B
1
0
B
…

$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$



$$(\qquad \Sigma, \qquad\qquad B \qquad\qquad\qquad\qquad\qquad\qquad)$$

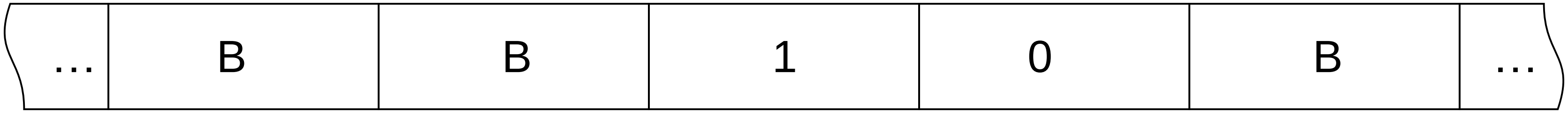


$$(\quad \Sigma, \quad B \quad)$$

Σ: input symbols

$\Sigma = \{0, 1\}$

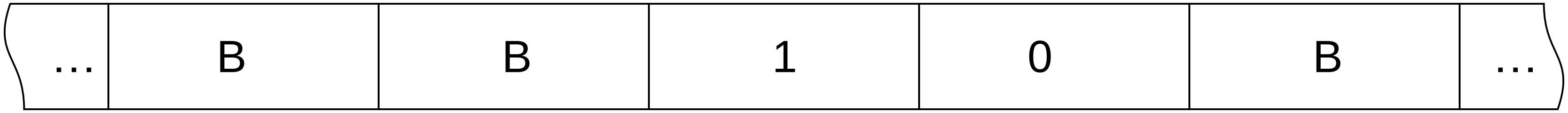
B: blank



$$(\quad \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, \quad)$$

band alphabet

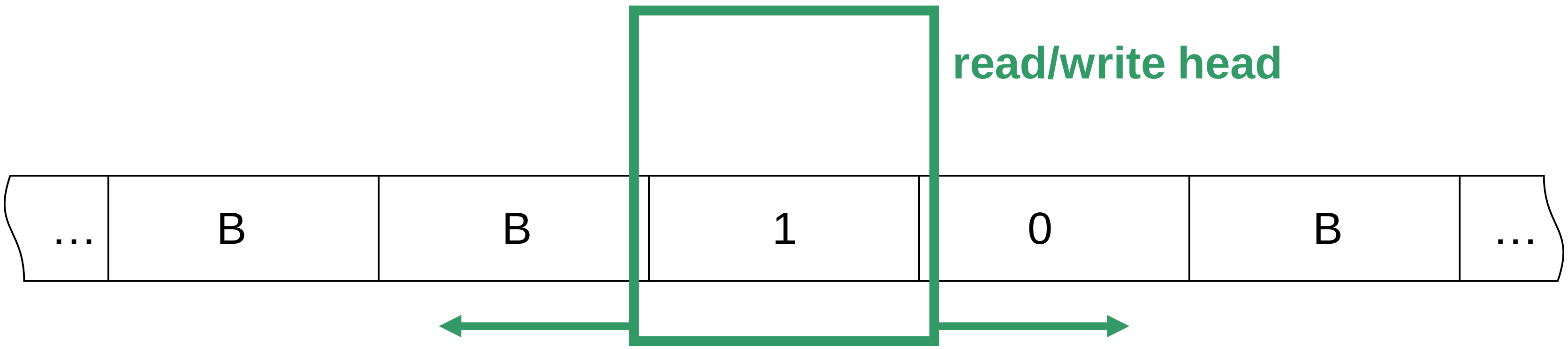
$$\Gamma = \{0, 1, B\}$$



$$\left(\quad \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, \qquad\qquad\qquad \right)$$

band alphabet

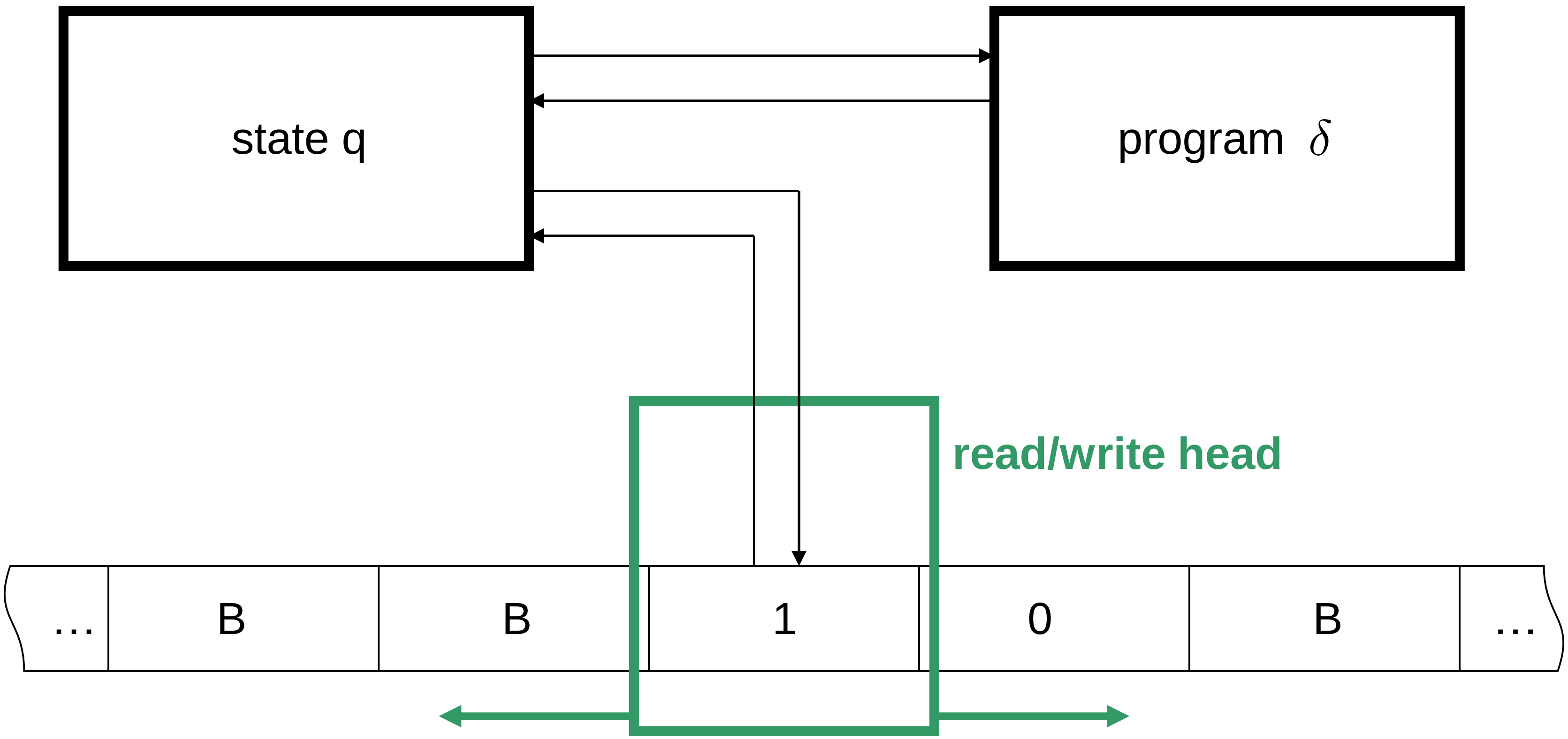
$$\Gamma = \{0, 1, B\}$$



$$(\quad \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, \qquad)$$

band alphabet

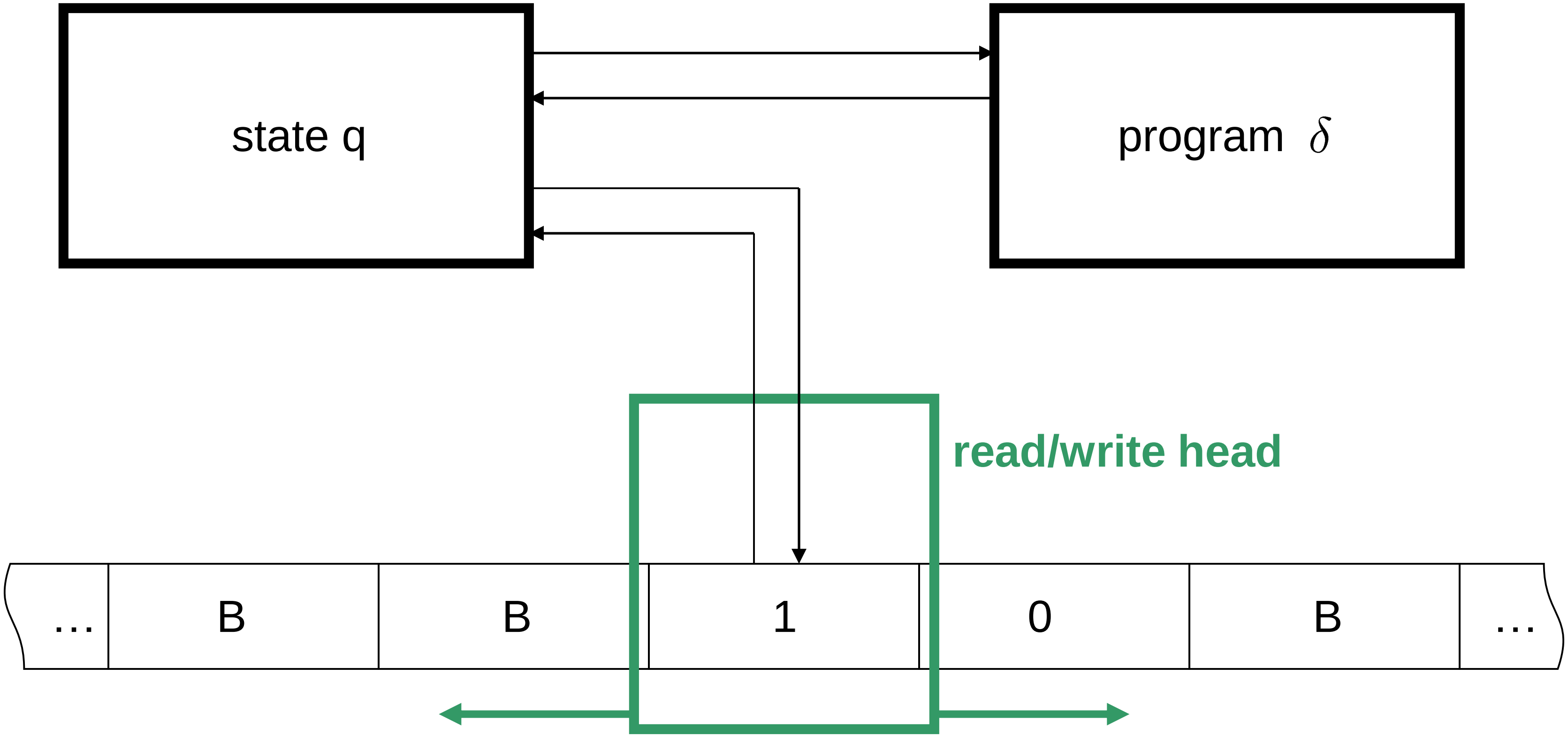
$\Gamma = \{0, 1, B\}$



$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, \qquad\qquad)$$

band alphabet

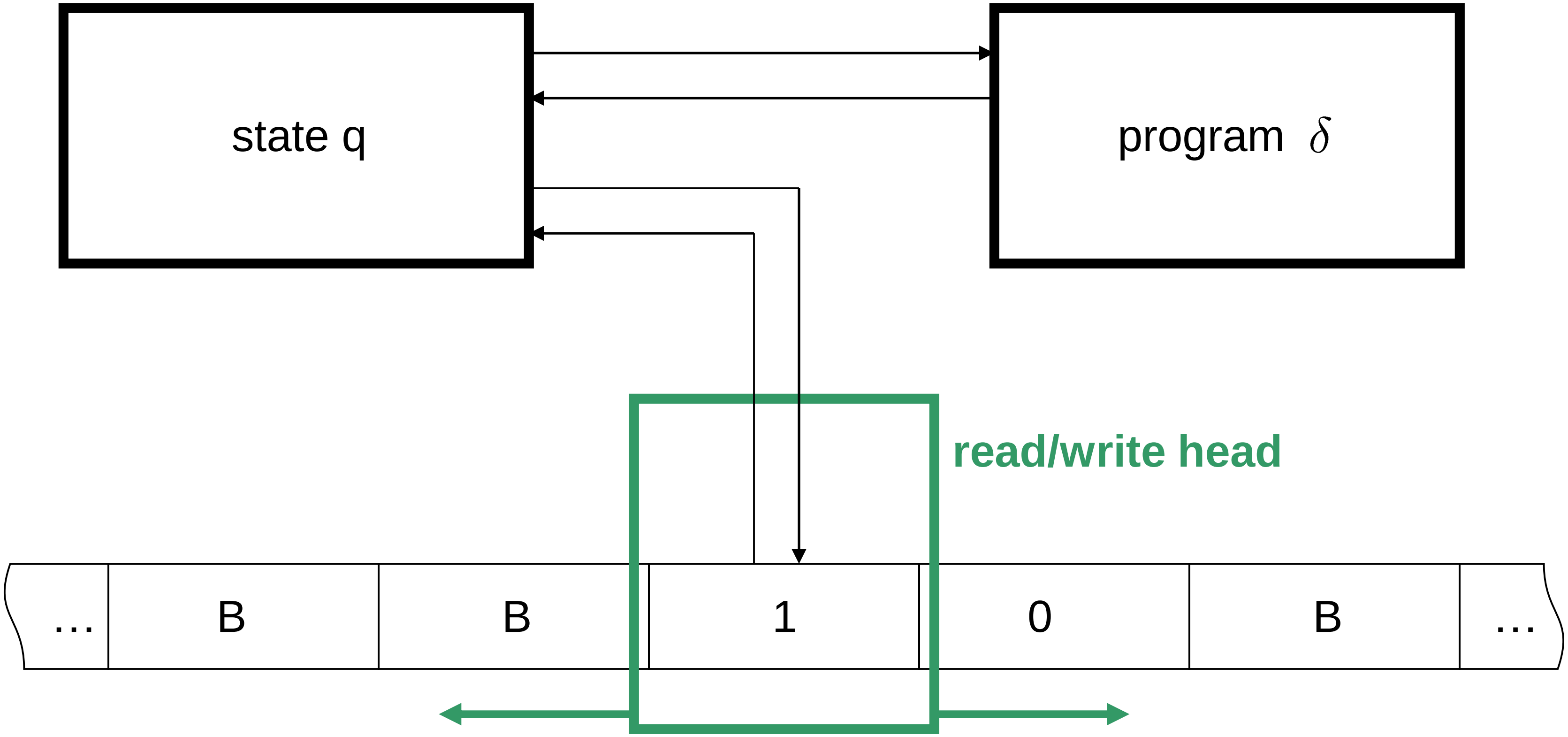
$\Gamma = \{0, 1, B\}$



$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \qquad)$$

band alphabet

$\Gamma = \{0, 1, B\}$

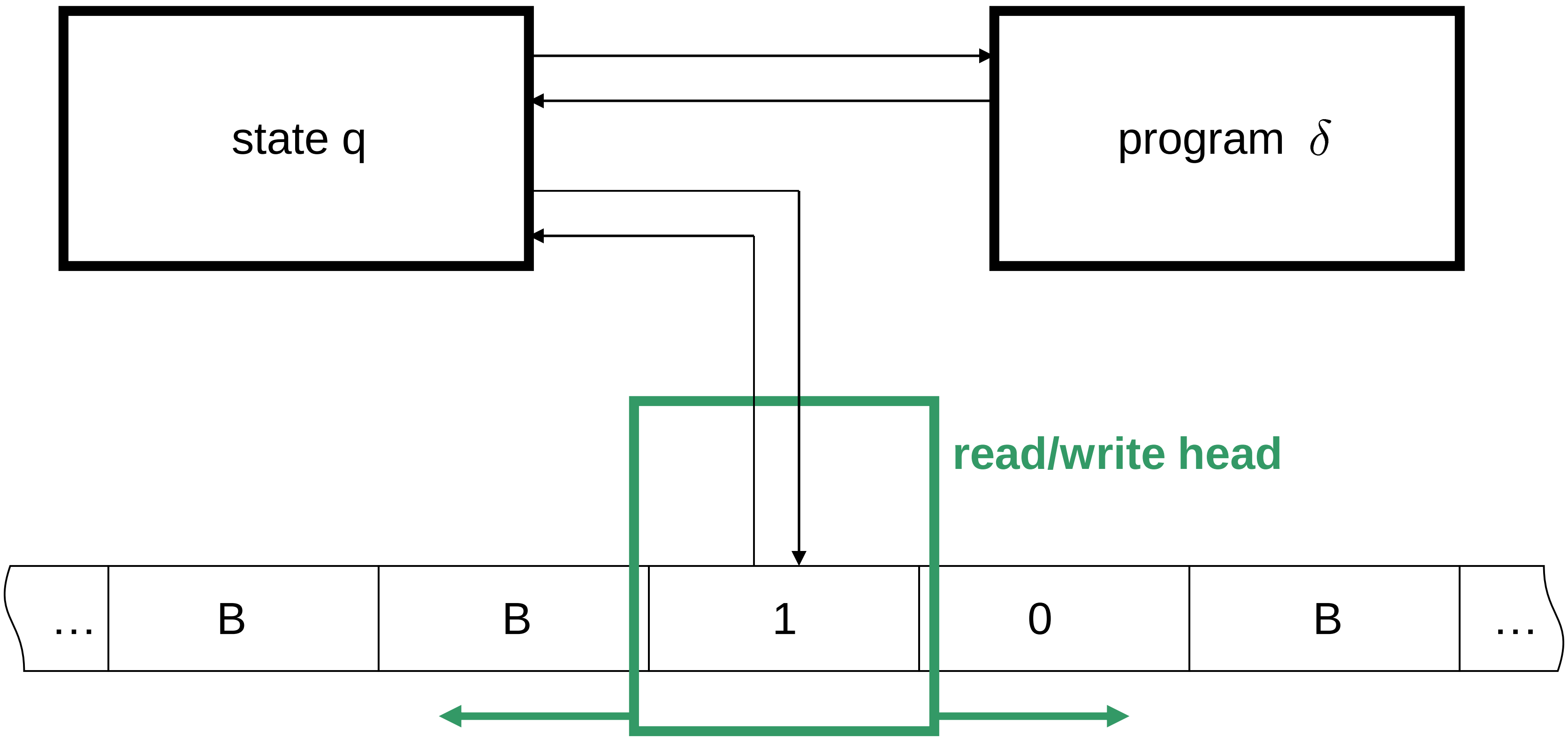


$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \quad F \subset Q)$$

accepting states

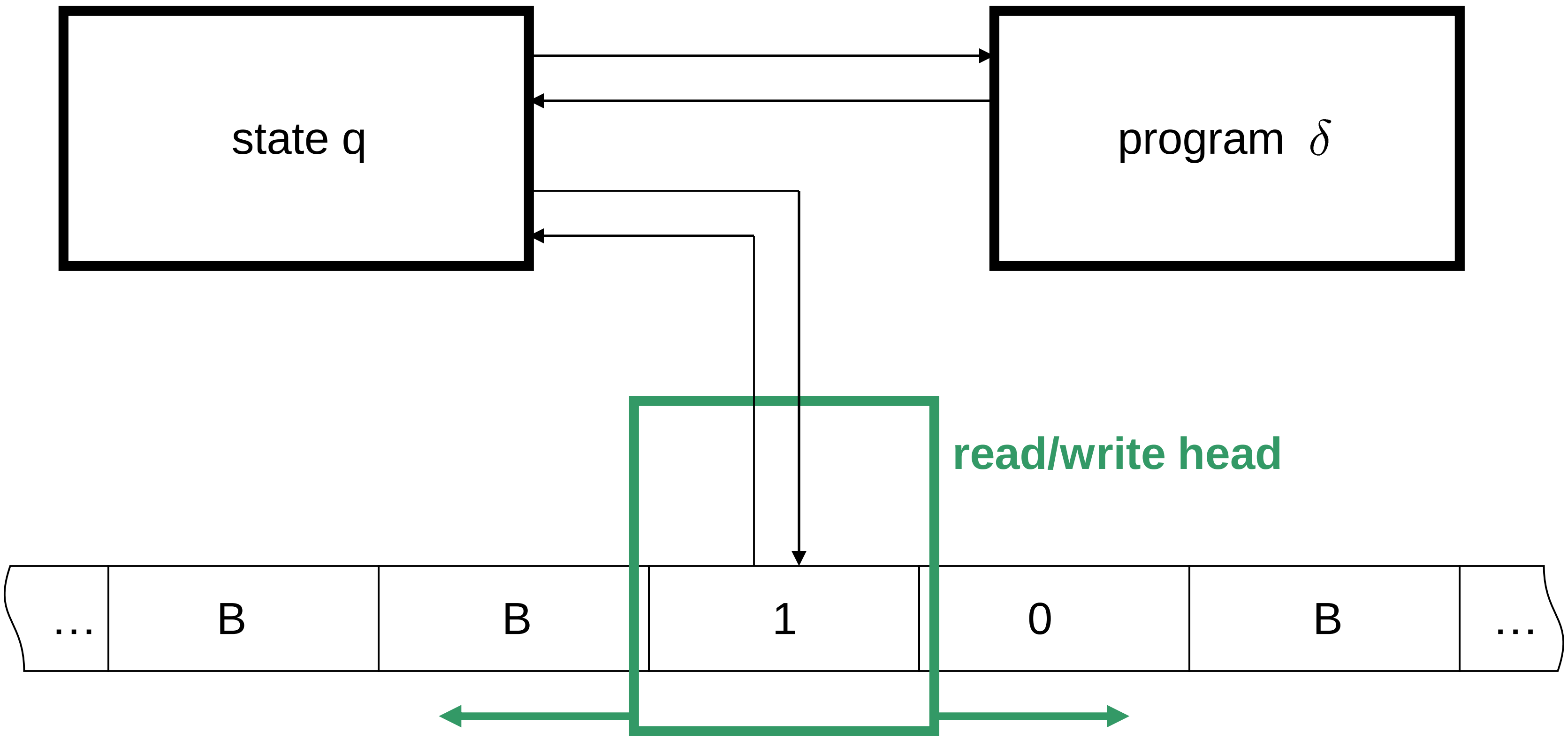
band alphabet

$\Gamma = \{0, 1, B\}$



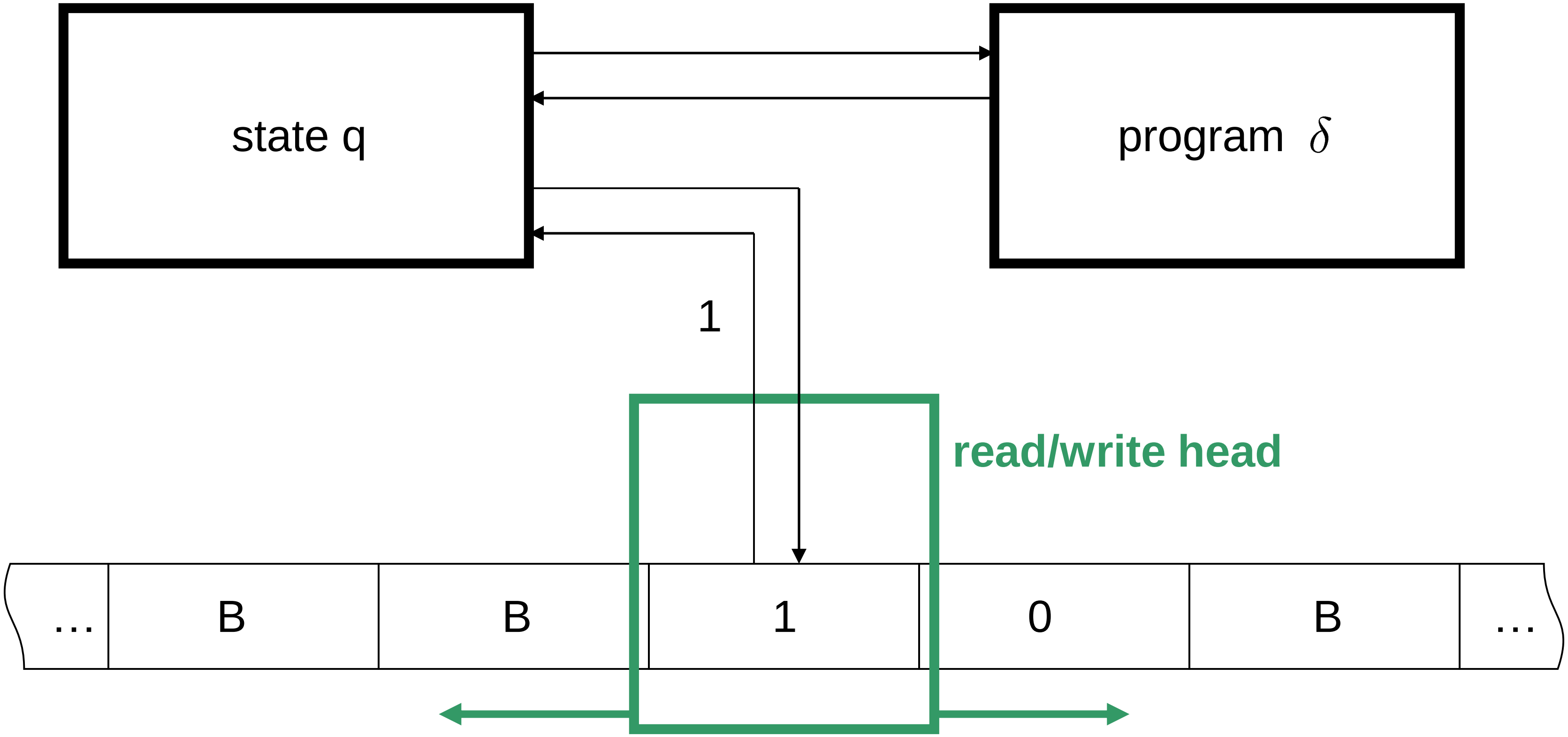
$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$



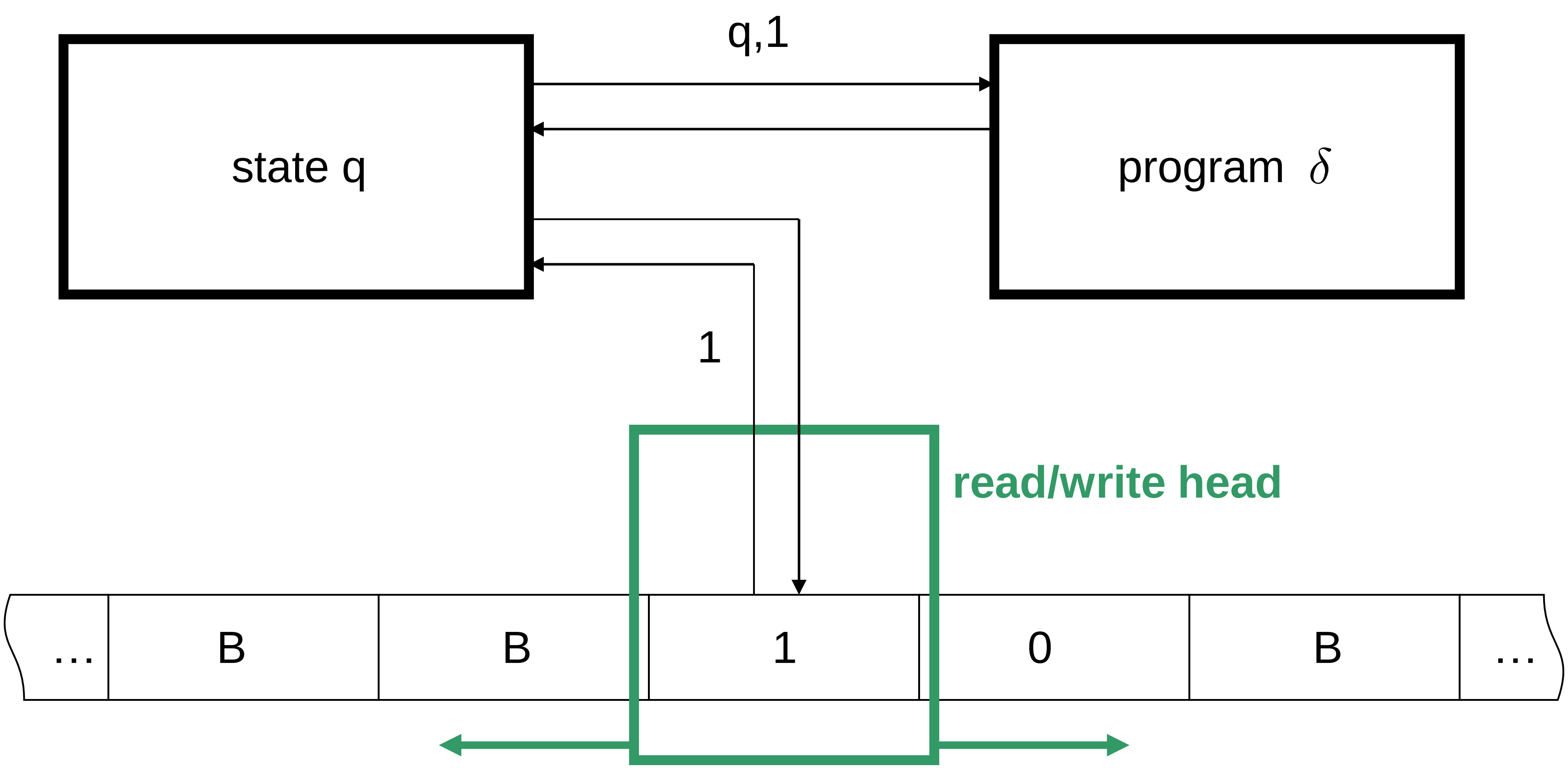
$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$



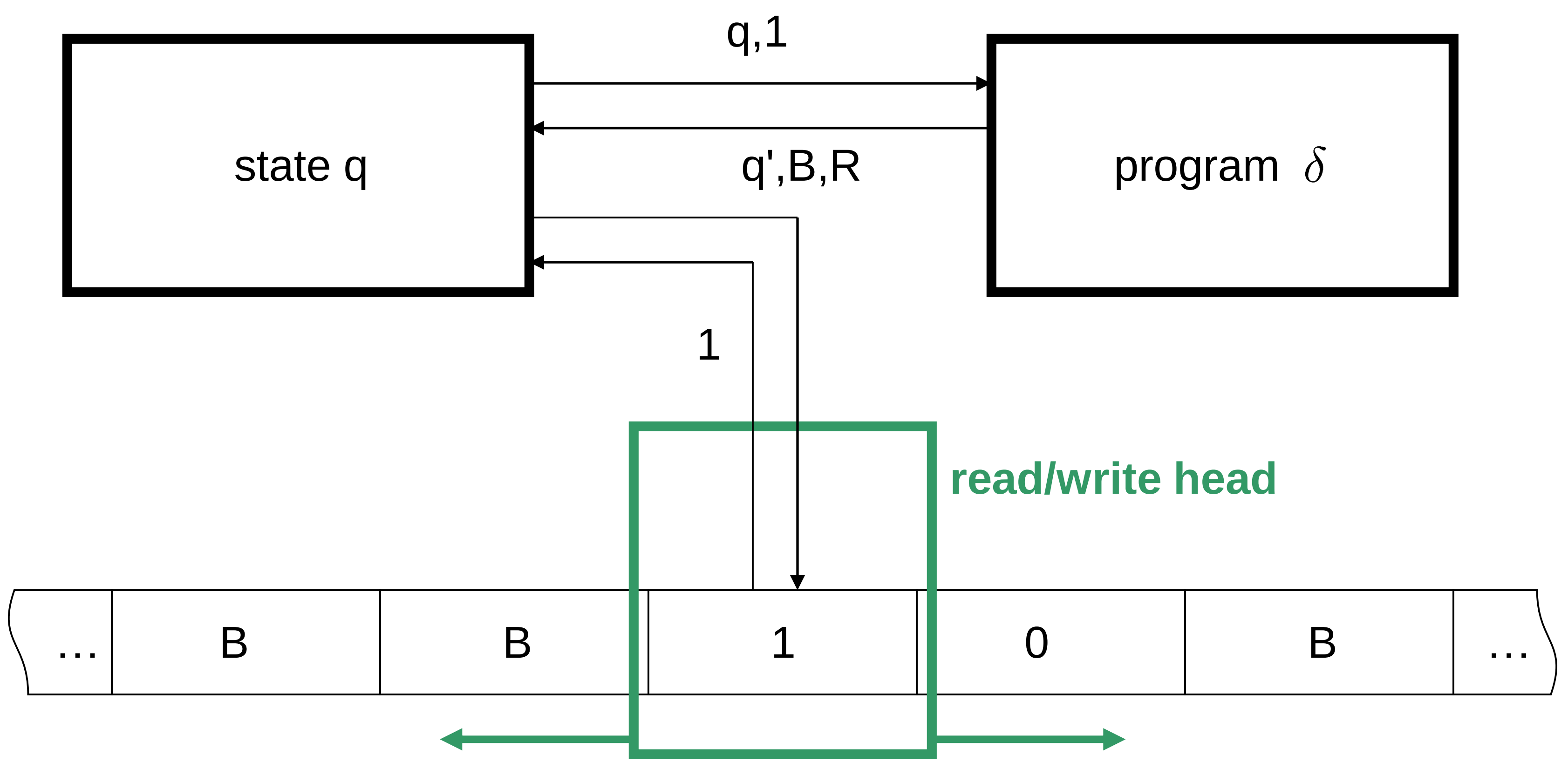
$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$



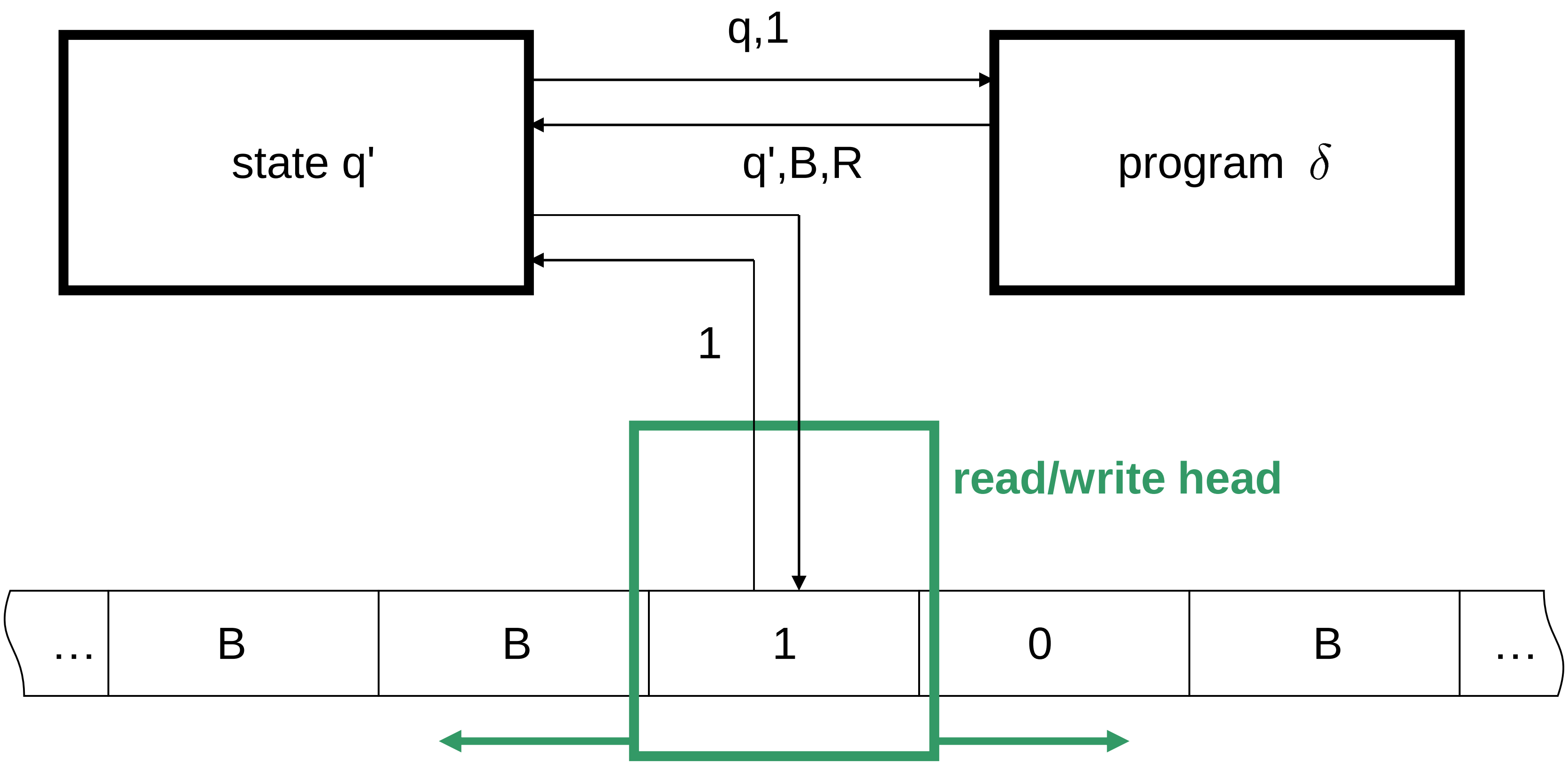
$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$



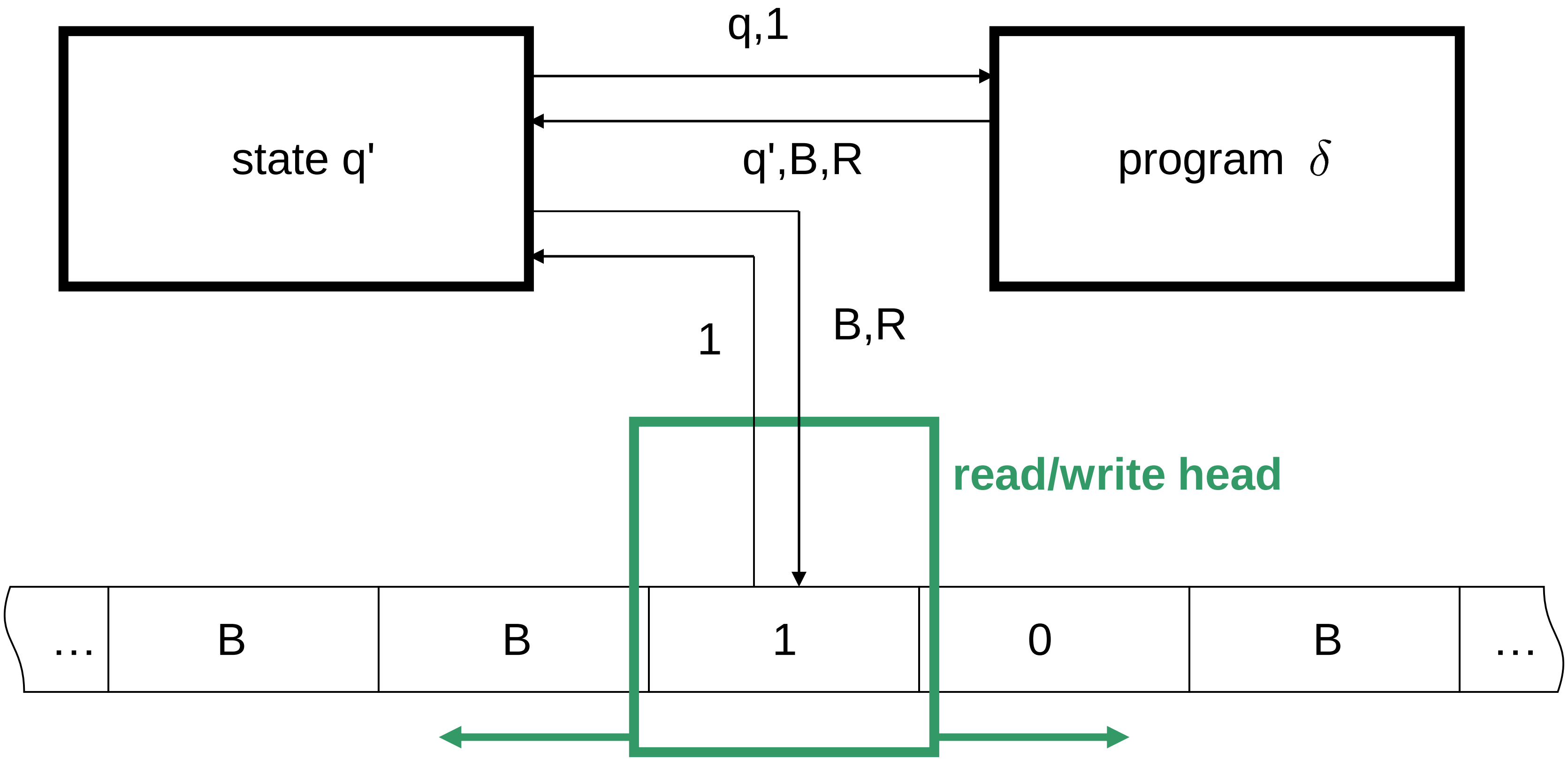
$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$



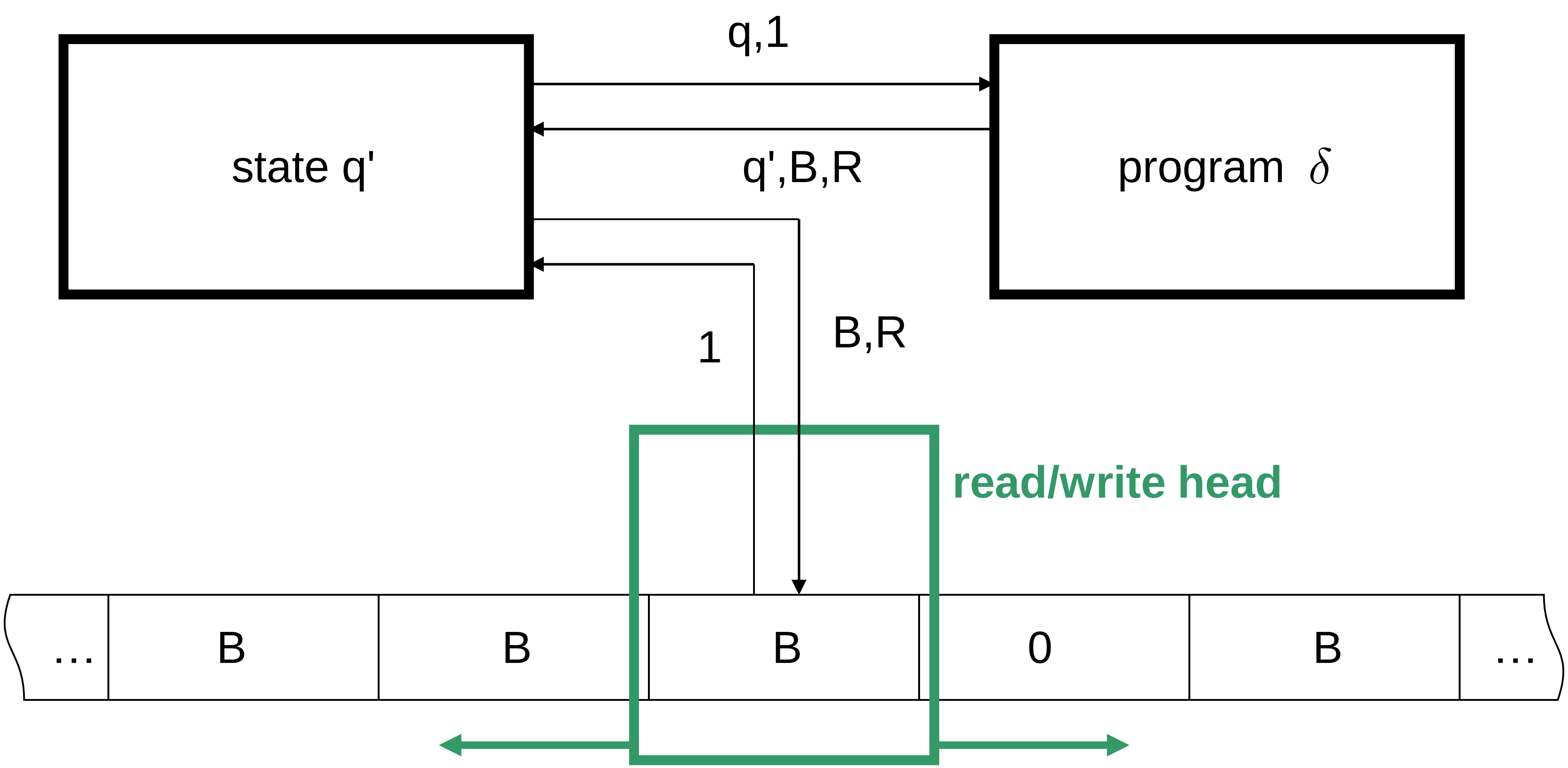
$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$



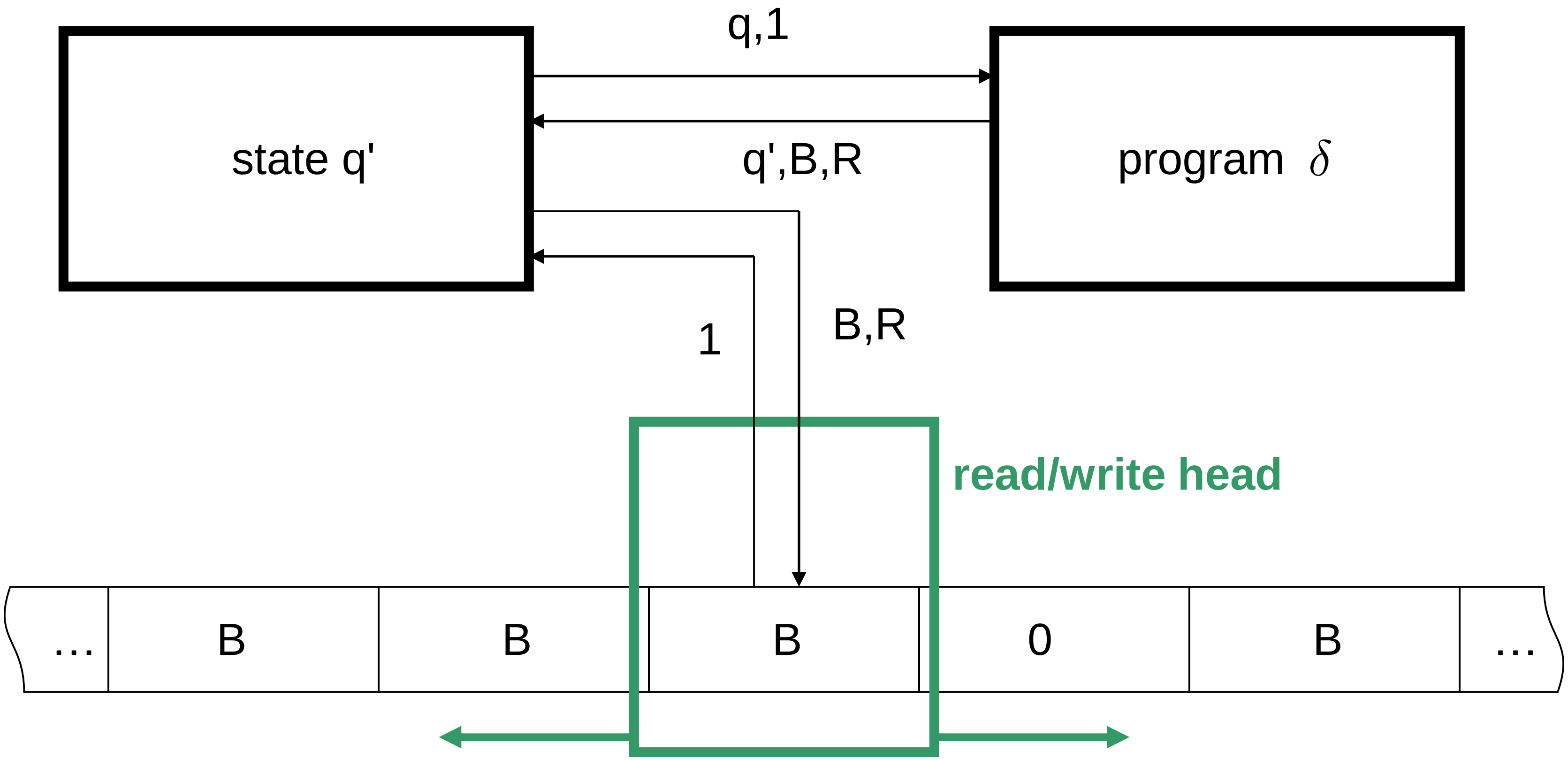
$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$



$$(Q, \Sigma, \Gamma \supset \Sigma, B \in \Gamma \setminus \Sigma, q_0 \in Q, \delta, F \subset Q)$$

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$



Interesting Facts

- instead of a RAM's random access, the TM computation is local
- deterministic TM (DTM) as powerful as RAM
  - except polynomial overhead (no proof here)

Universal Turing machines:

- get program and data as input
- simulate δ' of the program with general transition function

Church-Turing Thesis

- Every function which would naturally be regarded as computable can be computed by a Turing machine.
- not provable
- most surprising: there are functions that are not computable (undecidable)
  - halting problem: given a program P, does the universal TM halts on P?
- related to
  - incompleteness theorem
  - Entscheidungsproblem

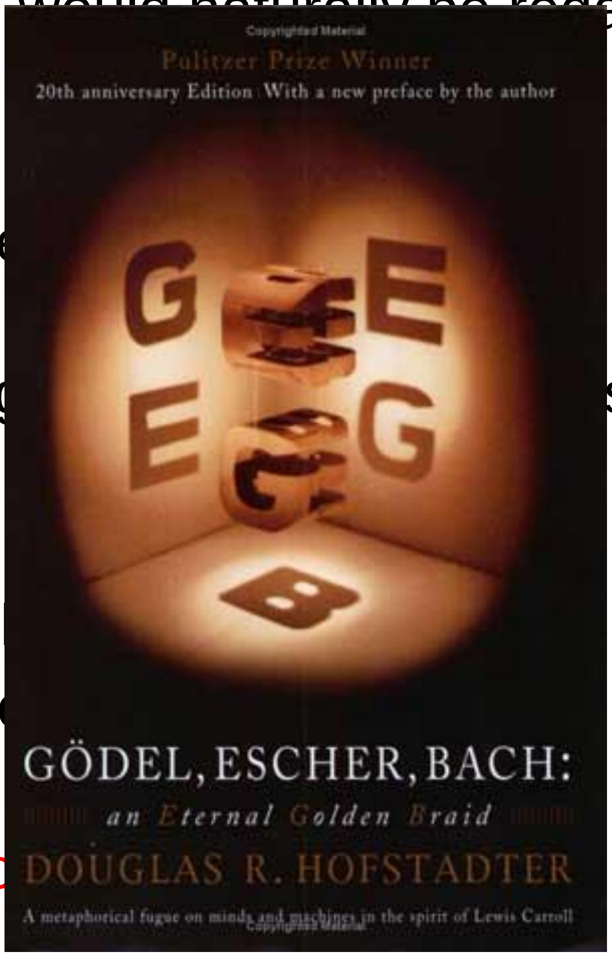
now from undecidable to decidable problems

Kurt Gödel (1906-78)

Church-Turing Thesis

- Every function which would naturally be regarded as computable can be computed by
- not provable
- most surprising: there … not computable (undecidable)
  - halting problem: g … s the universal TM halts on P?
- related to
  - incompleteness t
  - Entscheidungspr

now from undecidable to



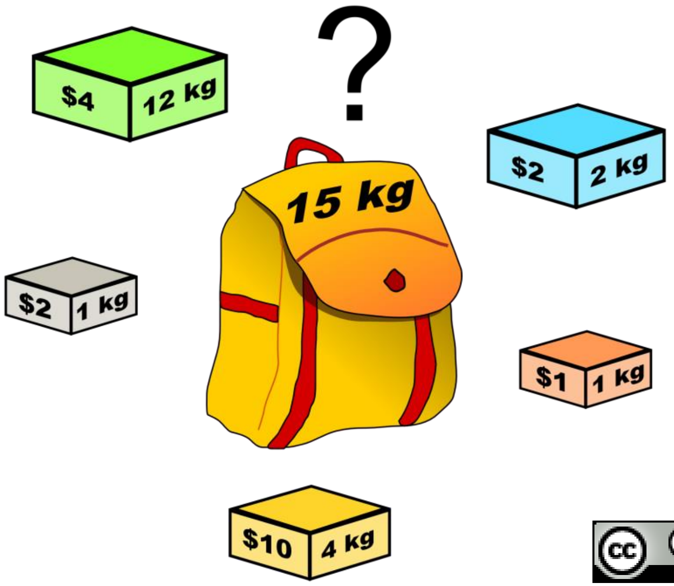
Kurt Gödel (1906-78)

Remains for today and next time...

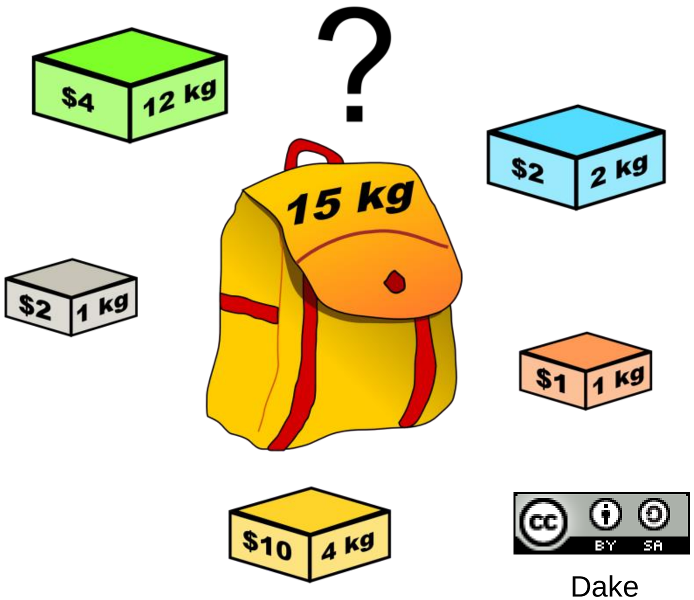
- complexity classes (in particular the famous P and NP)
- polynomial and Turing reductions
- hardness and completeness

What is P and NP?

- Complexity classes
- Set of problems with similar complexity
- Complexity = asymptotic running time of the best algorithm wrt. a given computation model (for the worst-case instance)
- Decision problems vs search problems vs optimization problems
  - Example: Knapsack Problem (short: KP)



Different Problem Types



Optimization problem:

find the best solution among all feasible ones!

- KP: “find packing with maximal value”

Search problem:

output a solution with a given structure!

- KP: “give a packing with value V”

Decision problem:

is there a solution with a certain property?

- KP: “is there a packing with value ≥V”

A decision problem is solved by a TM when it halts in an “accepting state” iff the given instance has the desired property

The Classes DTIME(t(n)) and P

$$\begin{aligned}\text{DTIME}(t(n)) \quad := \quad \{P \,|\, P &\text{ is a (decision) problem} \\ &\text{s.t. there exist an algorithm } A \\ &\text{that solves } P \text{ in time } O(t(n))\}\end{aligned}$$

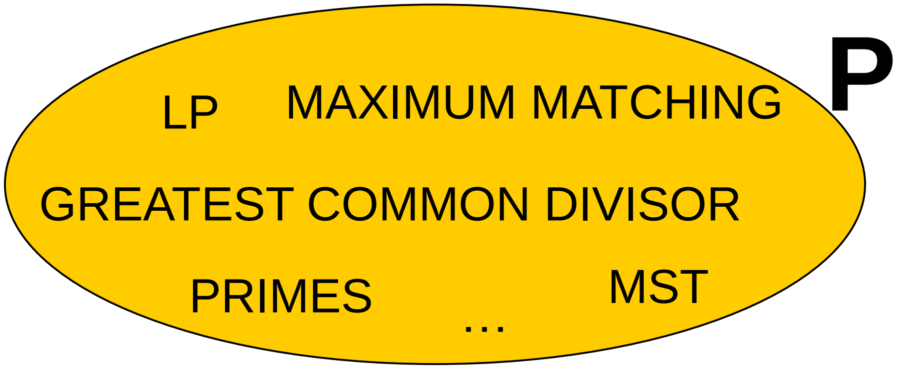
$$P = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

- Why is P defined like that? And why is P important?
  - Independent of computation model

    $P_{TM} = P_{RAM} = P_{\mu\text{-recursive functions}} = \dots$
  - Also independent of whether the TM has
    - one or more tracks
    - one or more tapes

Intuition about P

- P is the set of all problems which have polynomial time (deterministic) algorithms
- i.e., for a given problem p2P, there exists a DTM which
  - always halts in polynomial time and
  - ends in an accepting state iff the instance belongs to p, i.e., the answer to the problem p is "yes"
- P is the set of all "efficiently solvable" or "tractable" problems
  - This set is robust against changes of the computing model
  - But also not all problems in P are *practically* solvable, e.g., if the running time is $n^{1,000,000}$



Nondeterministic Turing Machines

Deterministic TM (DTM) have a deterministic transition *function:*

$$\delta_{\text{det}} : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$

Nondeterministic TM (NTM) have only a transition *relation:*

$$\delta_{\text{non-det.}} \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{R, L, N\})$$

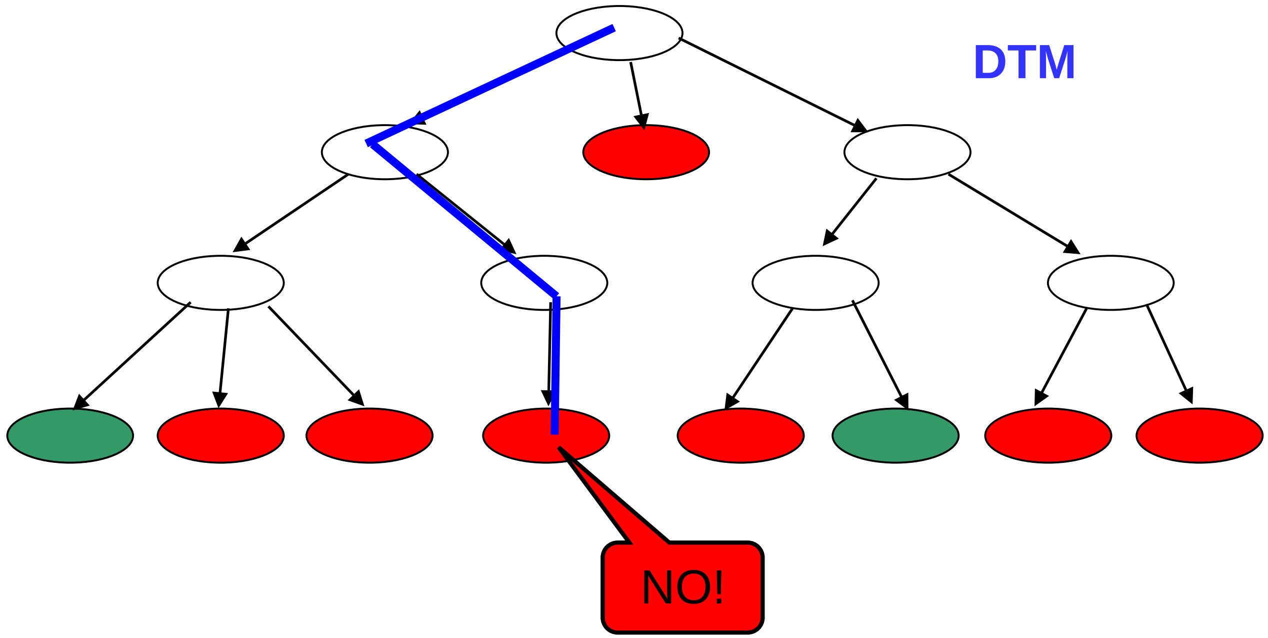
Which transitions will be actually performed?

- "lucky guesser": nondet. TM guesses the right transition
- "parallel computation": nondet. TM branches into many copies and accepts if one of the branches reaches an accepting state

NP is the set of all problems which have polynomial time nondeterministic (!) algorithms $\mathcal{NP} = \bigcup_{k \geq 1} \mathrm{NDTIME}(n^k)$

Intuition:

- If I know a solution I can prove in deterministic polynomial time whether it belongs to the answer "yes" or "no"
- "Guess" the right solution and prove it in polynomial time

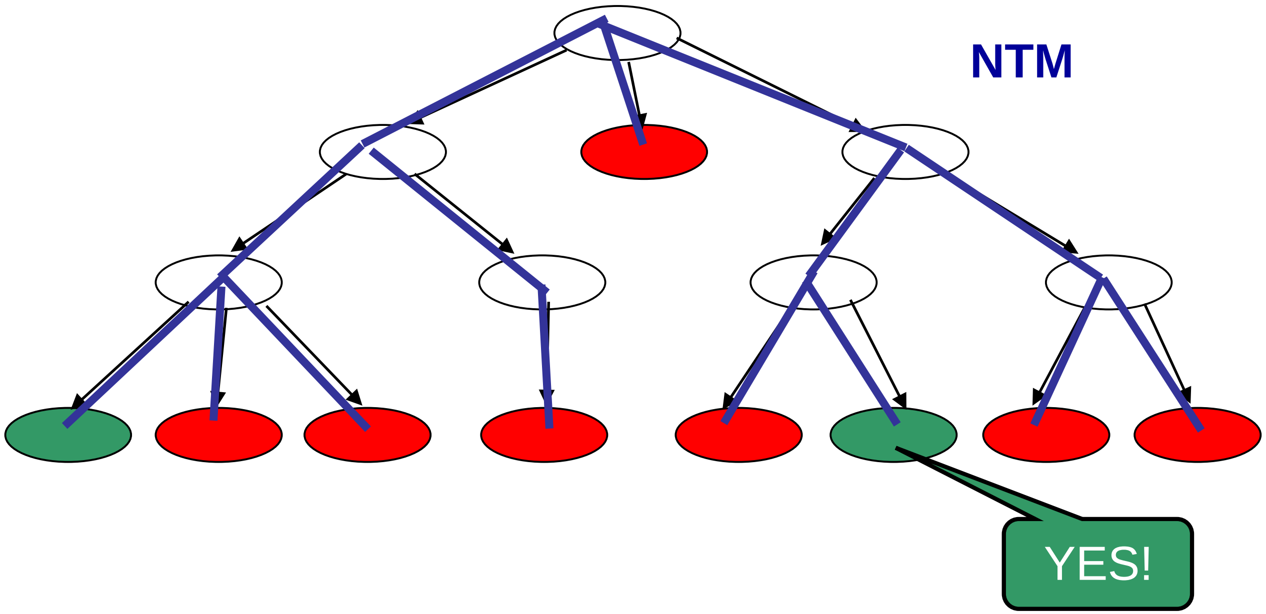


Nondeterminism and the Class NP

NP is the set of all problems which have polynomial time nondeterministic (!) algorithms $\mathcal{NP} = \bigcup_{k\geq 1} \text{NDTIME}(n^k)$

Intuition:

- If I know a solution I can proof in deterministic polynomial time whether it belongs to the answer "yes" or "no"
- "Guess" the right solution and proof it in polynomial time



Problems in NP

Knapsack Problem (KP)

- Guess which items to choose, check that the knapsack constraint is fulfilled, and sum up all profits

Travelling Salesperson Problem (TSP)

- Guess a tour and sum up all edge weights

Bin Packing (BP)

- Guess the assignment of items to bins, check that the size restrictions are fulfilled, and count the number of bins used

Facts about P=NP Hypothesis

- Clear: $\mathcal{P} \subseteq \mathcal{NP}$
- Not clear: $\mathcal{P} = \mathcal{NP}$
- What is the difference between, e.g., KP and PRIMES?
- For PRIMES, we know a polynomial time algorithm*, for KP, we don't
- Is KP "harder to solve" than PRIMES?
- Idea: classify the hardest problems in $\mathcal{NP}$
  - $\mathcal{NP}$-complete problems ($\mathcal{NPC} \subseteq \mathcal{NP}$)
  - Cook (1971), Levin (1973): SAT $\in \mathcal{NPC}$
  - Reductions

*Agrawal, Kayal, Saxena (2004): "Primes is in P", Annals of Mathematics, 160 (2004), 781–793
S. Cook (1971): "The Complexity of Theorem Proving Procedures", Proc. ACM symp. on Theory of computing, 151–158.
L. Levin (1973): "Universal'nye perebornye zadachi". Problemy Peredachi Informatsii 9 (3): 265–266.

Reductions

Idea:

if problem A can be solved by using an algorithm for problem B, then A is not harder than B (except for a polynomial overhead)

Polynomial Reduction $A \leq_p B$ (Cook, 1971)

- Transform instance of A into one for B within polynomial time by a function f
- Use oracle for B once which computes the solution for transformed instance as solution for A
- $a \in A \iff f(a) \in B$

Turing Reduction $A \leq_T B$ (Karp, 1972)

- Use oracle for problem B polynomially often to compute the solution of A
- $a \in A \iff f(a) \in B$

Important: both reductions are transitive!

see http://groups.csail.mit.edu/tds/papers/Lynch/stoc74.pdf

Example: DHC $\leq_p$ HC

Hamiltonian Cycle

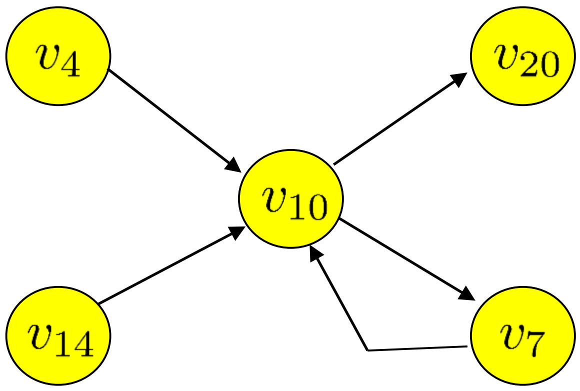
= A cycle in a graph which visits each vertex exactly once.

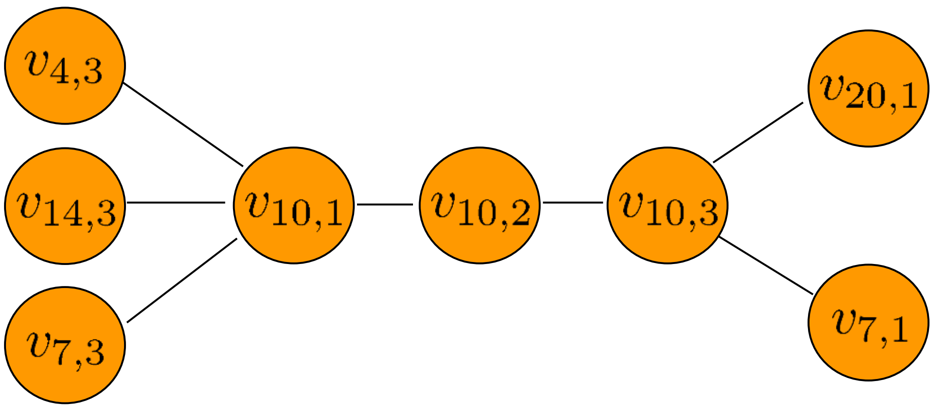
Hamiltonian Cycle Problem (HC), decision version

- given an undirected graph, is there a Hamiltonian cycle?

Directed Hamiltonian Cycle Problem (DHC)

- same for directed graphs





Example from I. Wegener (2003): "Komplexitätstheorie", Springer

- Transformation in polynomial time O(nm) possible
- Directed hamiltonian cycle in instance of DHC
  $\Longrightarrow$ Hamiltonian cycle in HC
- Hamiltonian cycle in instance of HC
  $\Longrightarrow$ order of HC is always ..., $v_{i,1}$, $v_{i,2}$, $v_{i,3}$, $v_{j,1}$, $v_{j,2}$, $v_{j,3}$, ... or
  ..., $v_{i,3}$, $v_{i,2}$, $v_{i,1}$, $v_{j,3}$, $v_{j,2}$, $v_{j,1}$, ...
  $\Longrightarrow$ take either HC or the inverted HC as solution for DHC $\square$

Different Types of Polynomial Reductions

- The last example was a reduction from a special case to a general case
- Now: one slightly more complicated example

Example: HC $\leq_p$ TSP

Observation: Hamilton Cycle Problem is a subproblem of TSP

Transformation:

Simulate same graph for TSP as the one given for HC

- Full graph actually, but weight 1 for each edge in HC graph and weight 2 for each „non-edge“ in HC
- Asking the TSP oracle whether a weight |V| tour exists

Correctness:

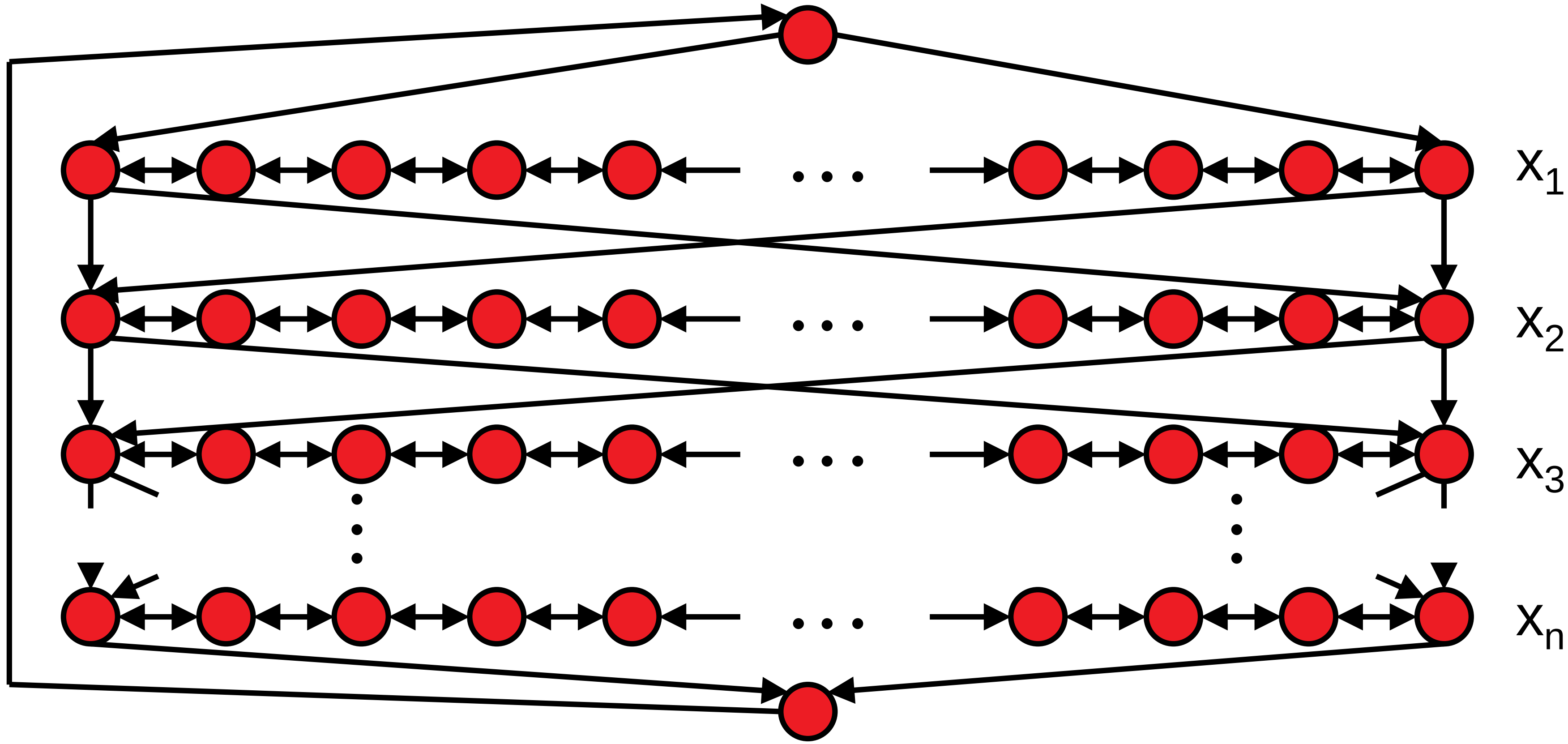
- If H is a Hamilton cycle in original graph, it is also a cycle through all cities but with weight ≤|V|
- Let T be a tour in the (transformed) TSP instance with weight ≤|V|. It cannot contain an edge with weight 2. Hence, the cycle T is also a cycle in the original HC problem.

Example: 3-SAT $\leq_p$ DHC

Given a 3-SAT instance with n variables x_i and k clauses.

Construction of DHC instance:

- basic graph with 2n many Hamilton circuits (n rows, 3k+3 columns)
- intuition: set x_i to TRUE iff its row is traversed from left to right



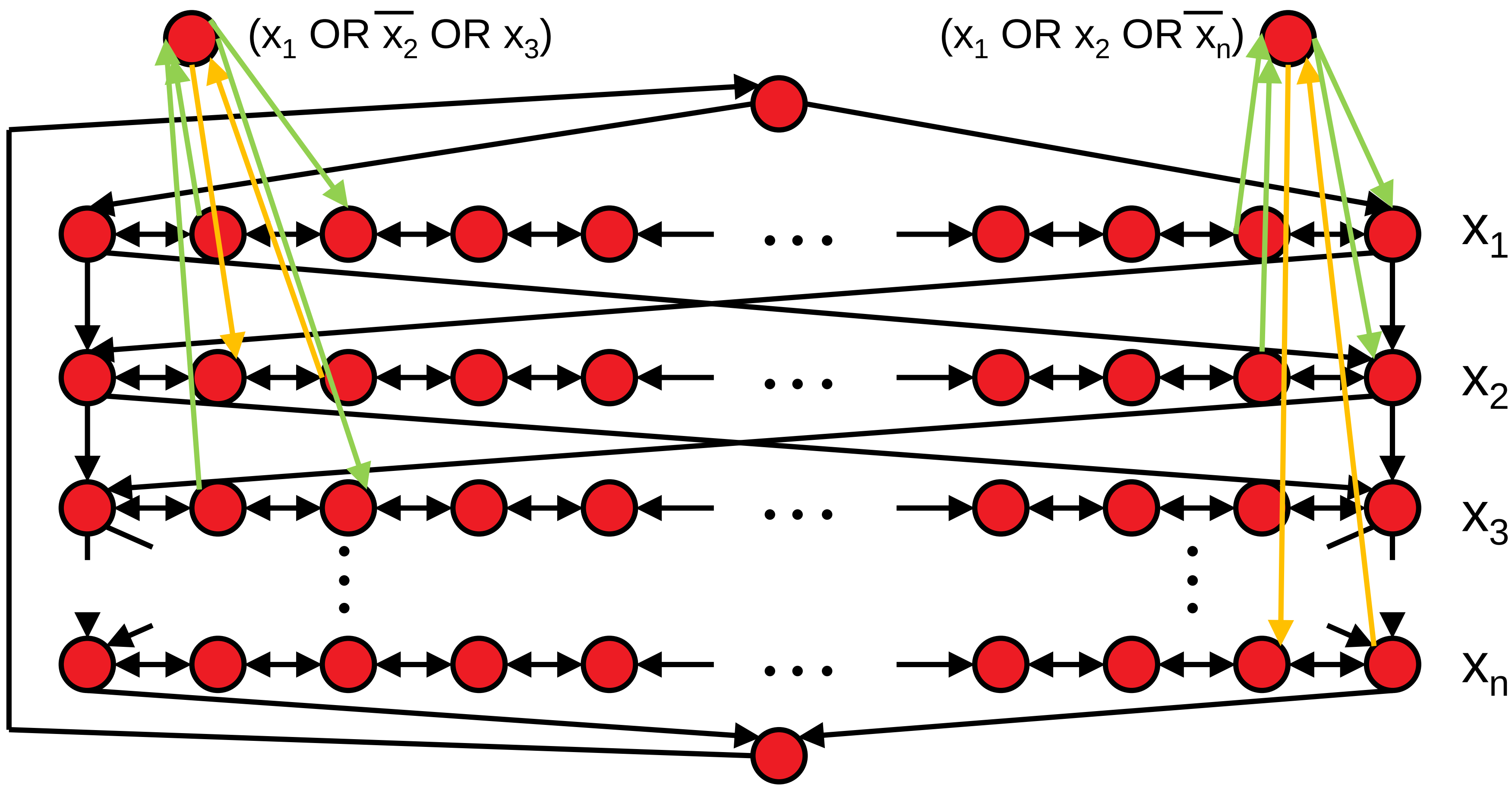
following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

Example: 3-SAT $\leq_p$ DHC

Given a 3-SAT instance with n variables x_i and k clauses.

Construction of DHC instance:

- for each clause add 1 vertex and 6 edges



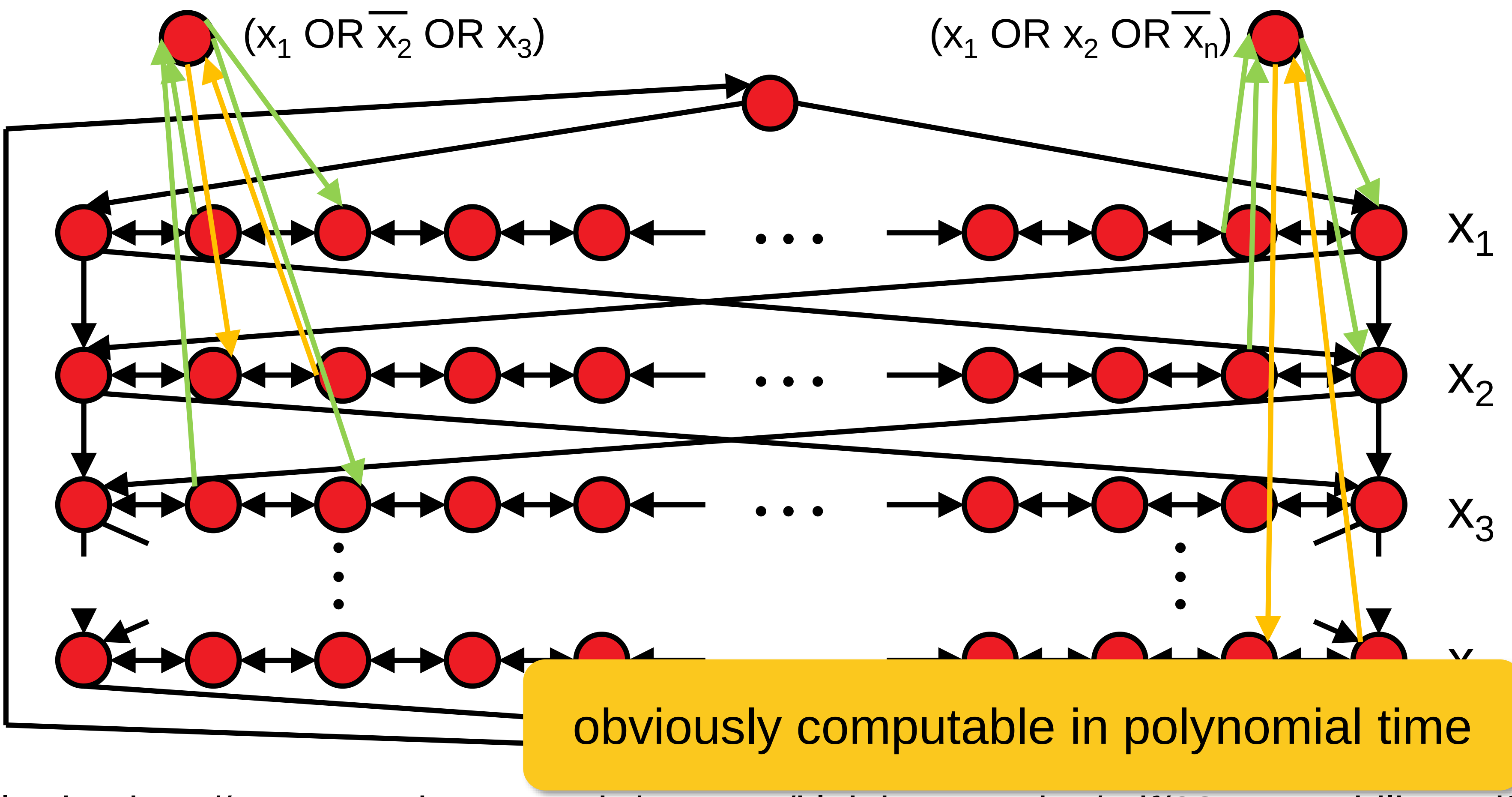
following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

Example: 3-SAT $\leq_p$ DHC

Given a 3-SAT instance with n variables x_i and k clauses.

Construction of DHC instance:

- for each clause add 1 vertex and 6 edges



following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

Proof of Correctness

3-SAT instance is satisfiable iff corresponding graph G has Hamilton cycle!

- let's show "⇨" first
- assume that the 3-SAT instance has satisfying assignment x*
- construct Hamiltonian cycle in G as follows:
  - if $x^*_i = 1$, traverse row i from left to right
  - if $x^*_i = 0$, traverse row i from right to left
  - for each clause C_j, there is at least one row i in which we are going in "correct" direction to insert the corresponding C_j vertex into the tour (we do this only once per clause vertex)

following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

Proof of Correctness

3-SAT instance is satisfiable iff corresponding graph G has Hamilton cycle!

- now, let us see "⇦"
- assume a Hamiltonian cycle H in G
- by construction, it has to visit node C_j from and to the same row
- replacing the part of H through C_j by the edge in between its neighbors defines a Hamilton cycle on $G \setminus C_j$
- doing this for all C_j allows to assign $x^*_i = 1$ if row i is traversed fully from left to right and $x^*_i = 0$ otherwise
- now since H traverses the clause vertex C_j originally, at least one of the paths through it is traversed in "correct" order and each clause is satisfied

□

following http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/08IntractabilityI.pdf

The Class NPC

- $\mathcal{NPC}$: set of all $\mathcal{NP}$-complete problems
- The "hardest problems in $\mathcal{NP}$"
- A is $\mathcal{NP}$-complete if
  - A ∈ $\mathcal{NP}$
  - All problems A_{NP}2NP can be polynomially reduced to A:

$$\forall A_{\mathcal{NP}} : A_{\mathcal{NP}} \leq_p A$$

- $\mathcal{NP}$-complete problems are the hardest of the ones in $\mathcal{NP}$ in the sense that if I can solve them in polynomial time, I can solve all problems in $\mathcal{NP}$ in polynomial time

Proving NP-completeness

How to prove that a problem A is $\mathcal{NP}$-complete?

Two possibilities:

- Either prove A ∈ $\mathcal{NP}$ and for all problems in $\mathcal{NP}$ that they can be reduced to A (complex, see Cook (1971)) or
- Prove A ∈ $\mathcal{NP}$ (simple) and a reduction from a problem B that is already known as $\mathcal{NP}$-complete to A (!)

caveat: be careful of the order in the reduction!

Theorem: 3-SAT $\in$ NPC

- proven by Cook in 1971 and independently (with a slightly different proof) by Levin in 1973
- not enough time here for the detailed proof

But idea easy to understand:

- 3-SAT $\in$ NP trivial
- Given any problem p $\in$ NPC and an instance i to that problem, construct a Boolean formula which is satisfiable iff the non-deterministic TM for p accepts instance i
- Variables for states of the TM, e.g. $T_{i,j,k}$ = true if tape cell *i* contains symbol *j* at step *k* of the computation
- Polynomially many variables and Boolean statements enough because the TM runs in polynomial time

Difference between NP-complete and NP-hard

A is NP-complete if

- $A \in \mathcal{NP}$
- $\forall B \in \mathcal{NP} : B \leq_p A$

A is NP-hard if

- $\forall B \in \mathcal{NP} : B \leq_T A$

Implications:

- An NP-hard problem is not necessarily a decision problem
- The search and optimization versions of an NP-complete problem are NP-hard

Practical Implications of Reductions

The proof of NP-completeness is typically seen as a proof of difficulty:

"I did not find an efficient algorithm for my problem, maybe I am dumb?"

vs.

"I cannot find an efficient algorithm for my problem because there is none"

vs.

"I did not find an efficient algorithm for my problem but neither did all of those famous people"

But...

Having a proof of NP-completeness or NP-hardness, does not mean that a problem is not manageable in practice:

- the average-case complexity might be reasonable
- randomized algorithms might work well
- maybe, the difficult instances are not observed

Example of success: SAT solvers

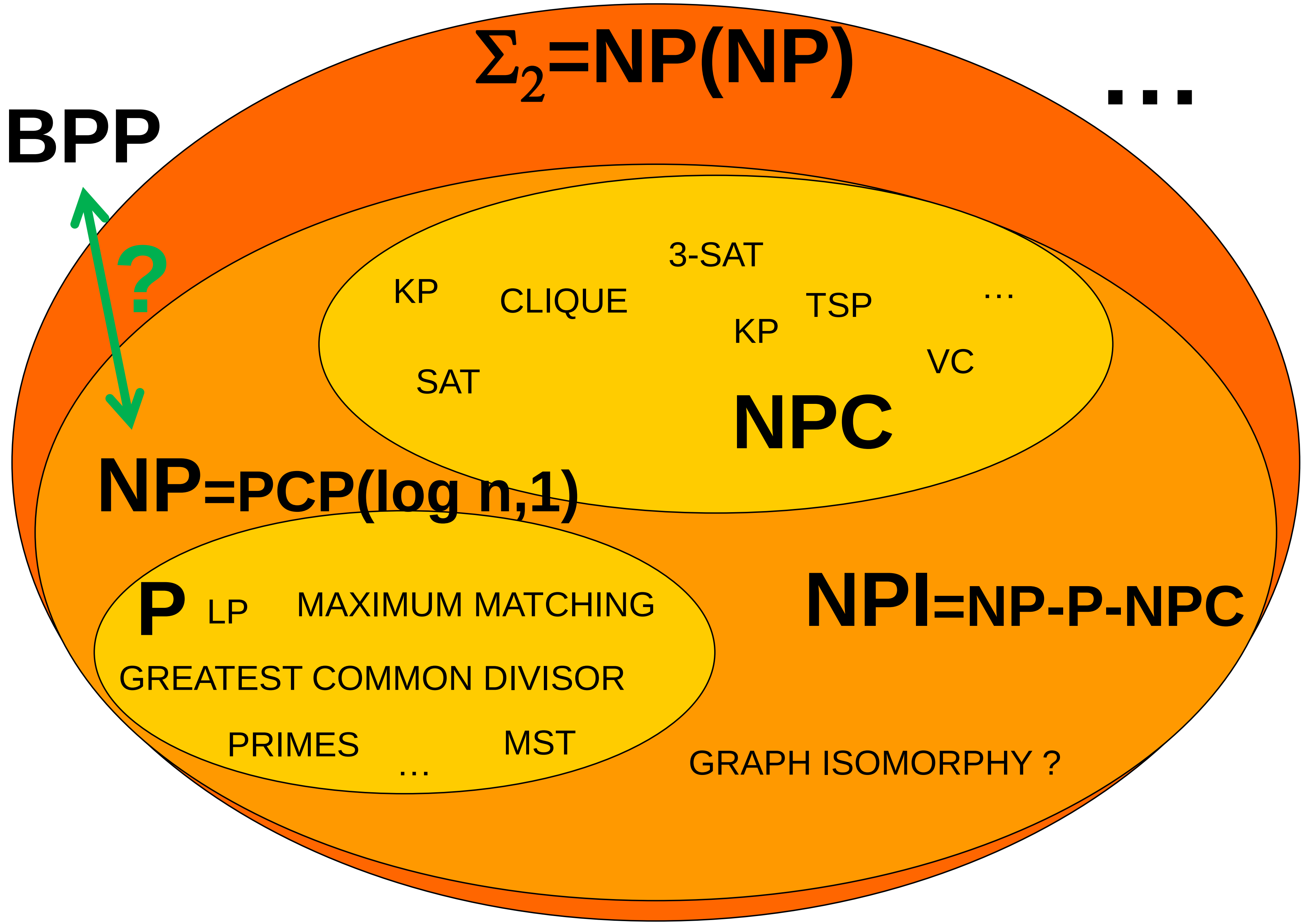
The Famous P versus NP Problem

Is P=NP?

- One of the 7 Millennium Prize problems selected by the Clay Mathematics Institute (worth 10^6 $)
- first mentioned in 1956 in letter from K. Gödel to J. von Neumann
- formalized by J. Cook in his 1971 seminal paper
- solving this problem might have significant practical implications (or not)

what do you think?

The „Complexity Zoo"



Conclusions

I hope it became clear...

...what complexity theory is about

...what is a Random Access Machine and a Turing Machine

... how a decision and an optimization problem differ

...what are the classes P, NP, and NPC

...and that complexity theory is more involved than what we could see here ☺