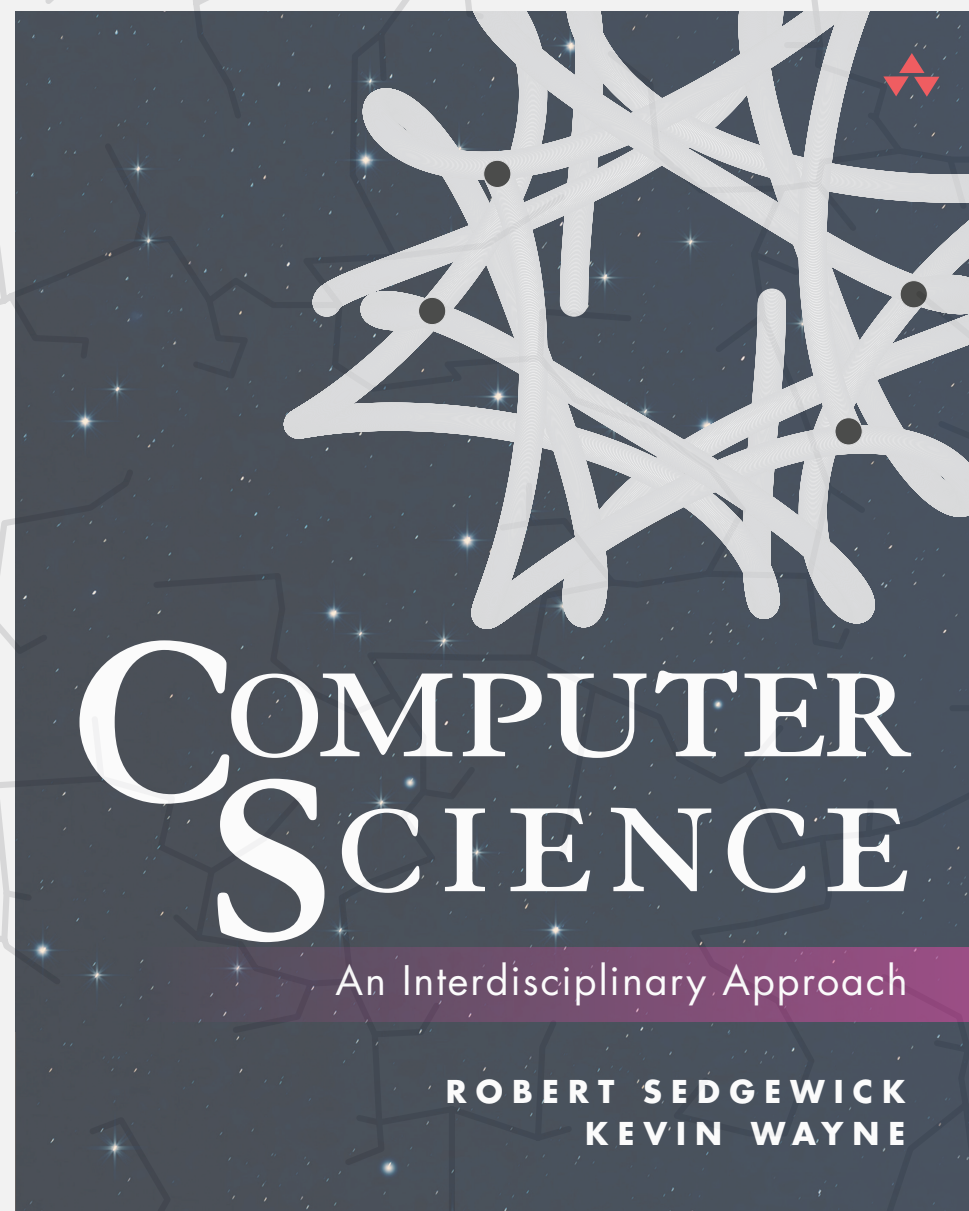


<http://introc.cs.princeton.edu>

ALL QUESTIONS THEORY

- ▶ *REs, DFAs, and NFAs*
- ▶ *Universality and computability*
- ▶ *P, NP, NP-complete*



<http://introc.cs.princeton.edu>

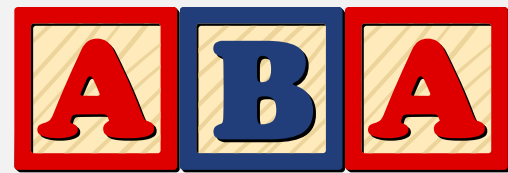
ALL QUESTIONS THEORY

- ▶ *REs, DFAs, and NFAs*
- ▶ *Universality and computability*
- ▶ *P, NP, NP-complete*

Definitions

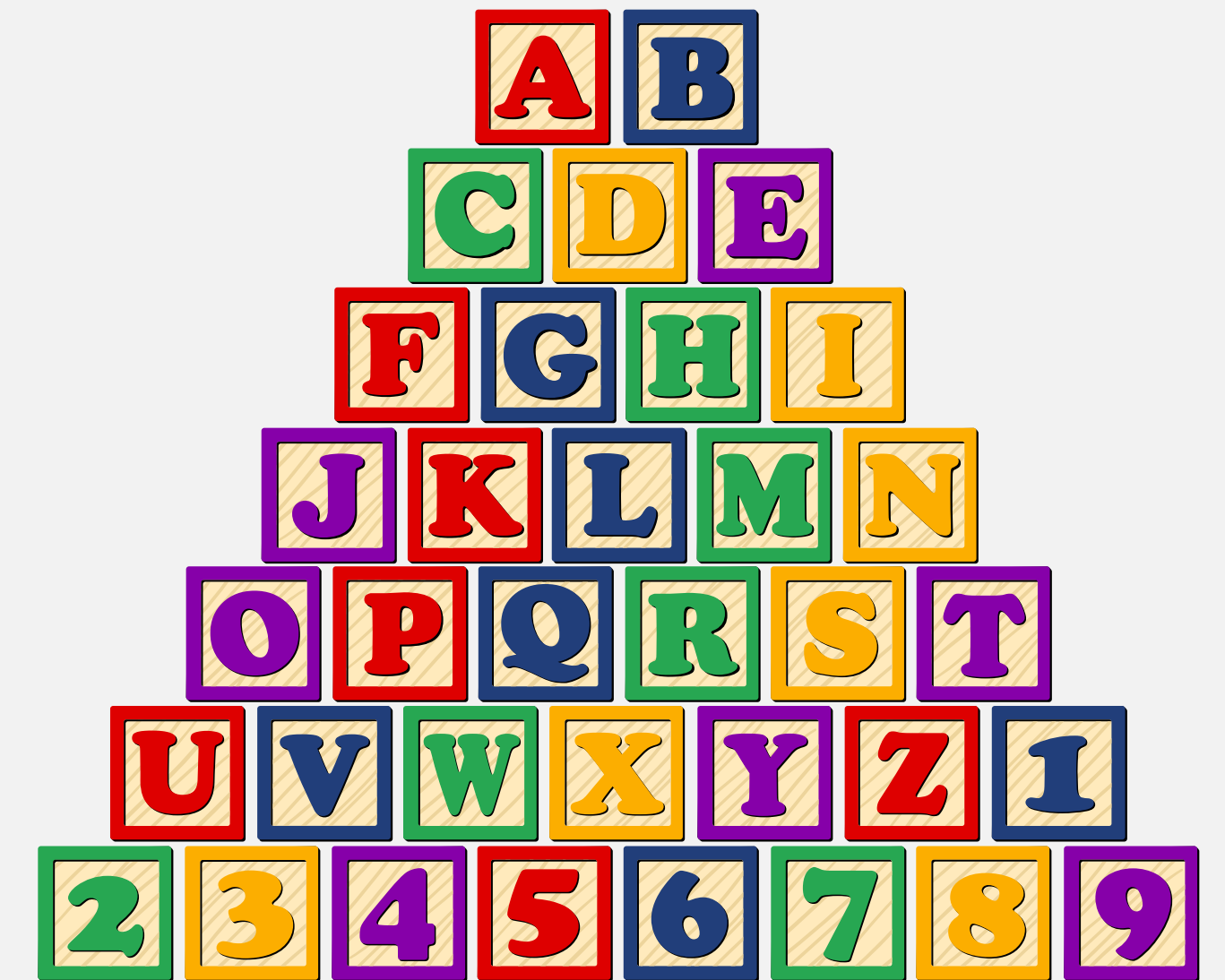
Alphabet. Finite set of symbols.

String. Sequence of alphabet symbols.



Formal language. Set of strings.

{  ,  ,  , ... }



Big ideas

Regular expression. Concise notation for **specifying** a formal language.

Relevance. Widely used in practice to

- Validate web forms.
- Express legal queries.
- Specify biological sequences.
- Define elements of a programming language.
- Enforce programming style guidelines.
-

```
<module name="LocalVariableName">
  <property name = "format"
            value = "[a-z][a-zA-Z0-9]*$"/>
</module>
```



Big ideas

Regular expression. Concise notation for **specifying** a formal language.

operation	order	example RE	matches	does not match
concatenation	3	AABAAB	AABAAB	<i>every other string</i>
union	4	AA BAAB	AA BAAB	<i>every other string</i>
closure	2	AB* A	AA ABBBBBBBBA	AB ABABA
parentheses	1	A(A B)AAB	AAAAB ABAAB	<i>every other string</i>
		(AB)* A	A ABABABABABA	AA ABBA

Spring 2015, Question 3

For each English-language description at right, design an RE that describes the formal language.

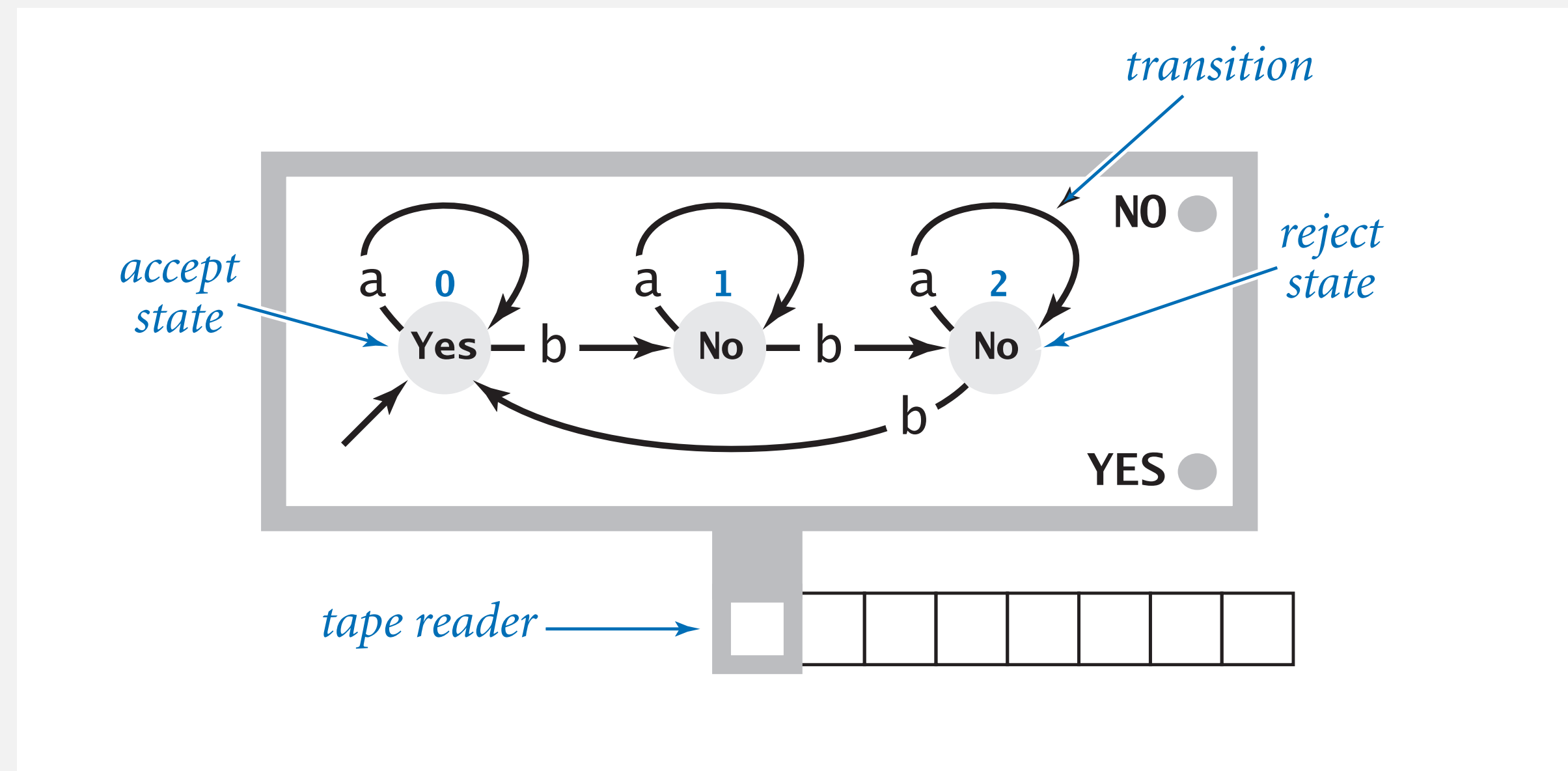
Binary strings beginning and ending with the same bit

Binary strings containing exactly two 0s that are not adjacent and no other 0s

Binary numbers divisible by 8

Big ideas

DFA. A simple machine that **recognizes** a language. For each input symbol and each state, there is exactly one possible state transition.

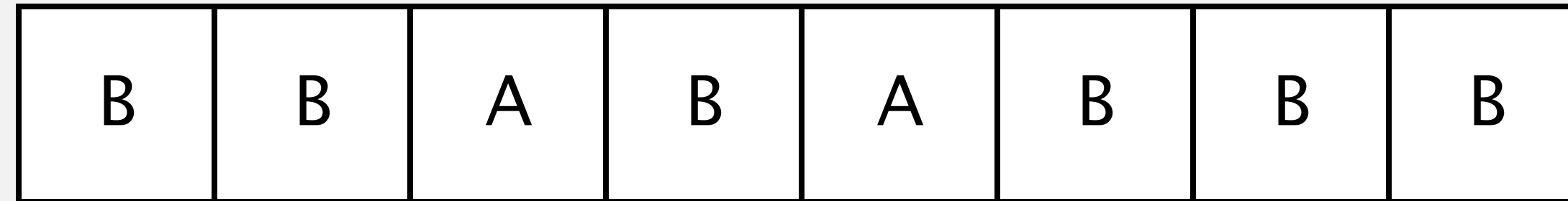


Kleene's theorem. REs, DFAs, and NFAs are equivalent models—they all characterize the regular languages.

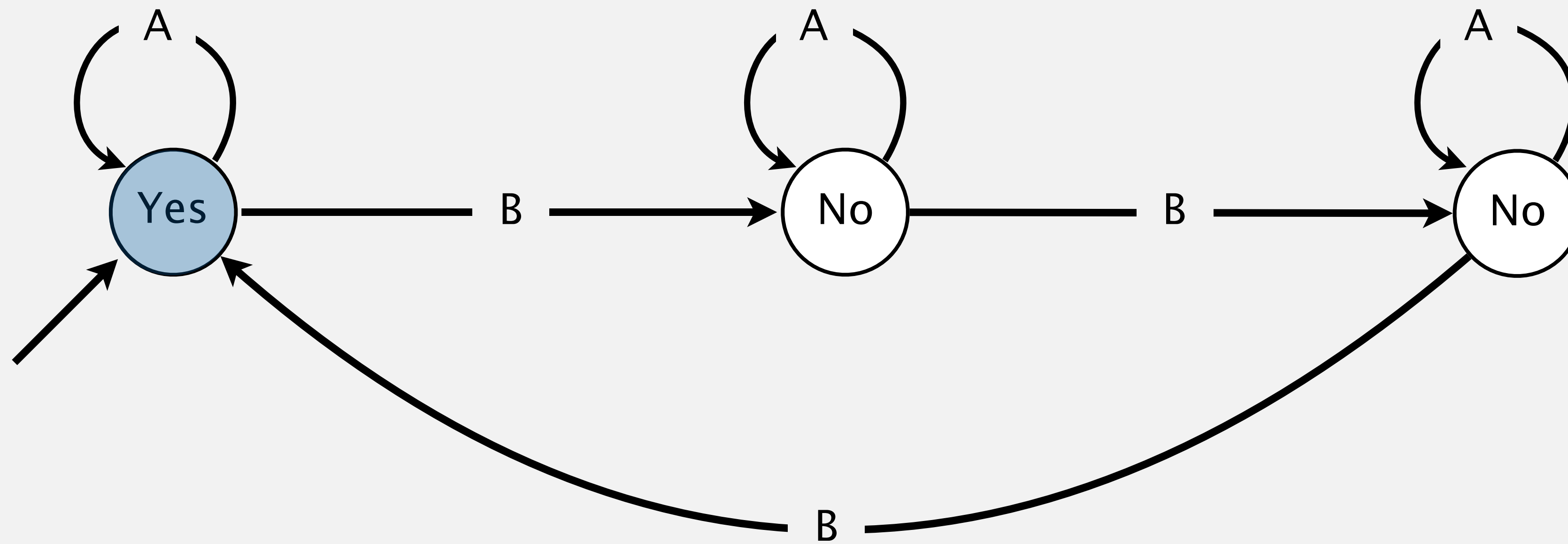
Relevance. DFAs and NFAs solve the **recognition problem** for REs.

Tracing a DFA

input string



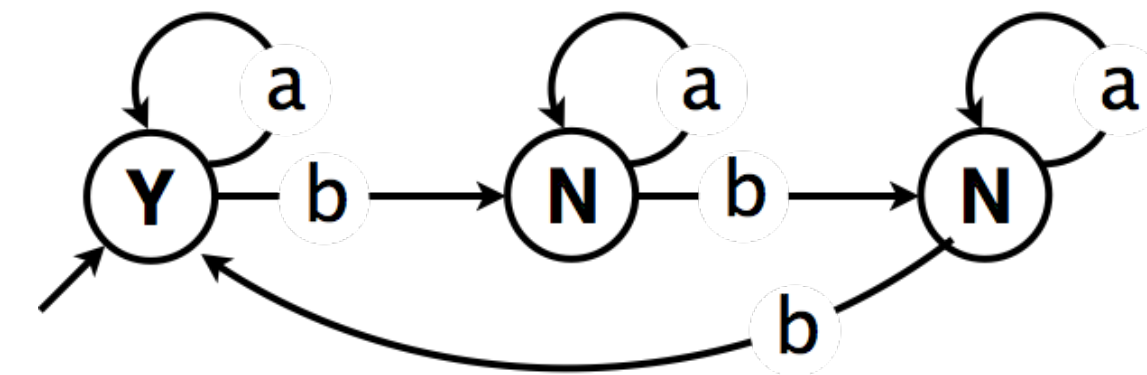
accept



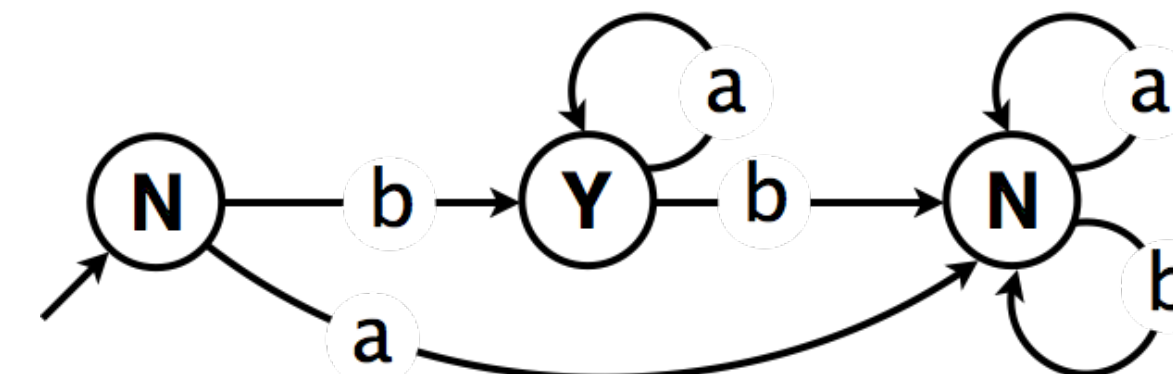
Spring 2009, Question 9

For each DFA at right, select the best-matching RE at left.

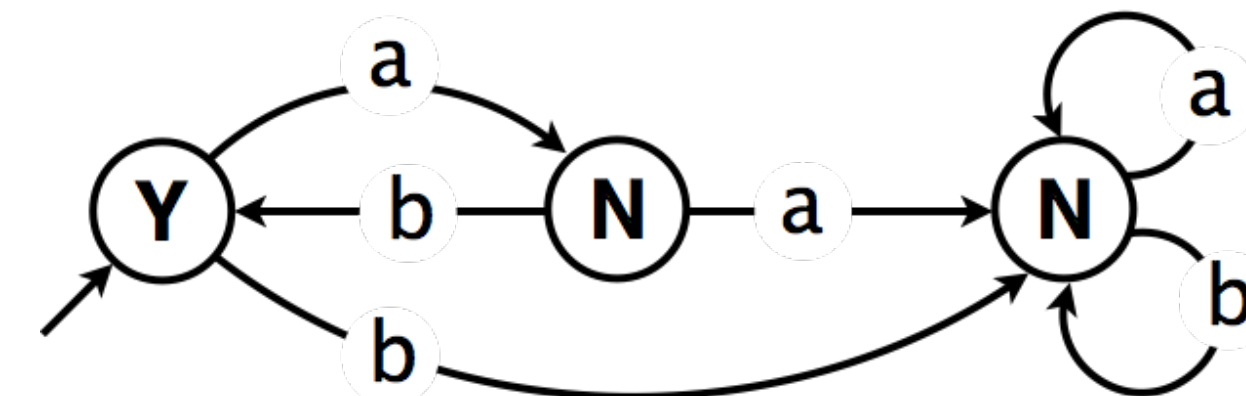
A. $a^* | (a^*ba^*ba^*ba^*)^*$



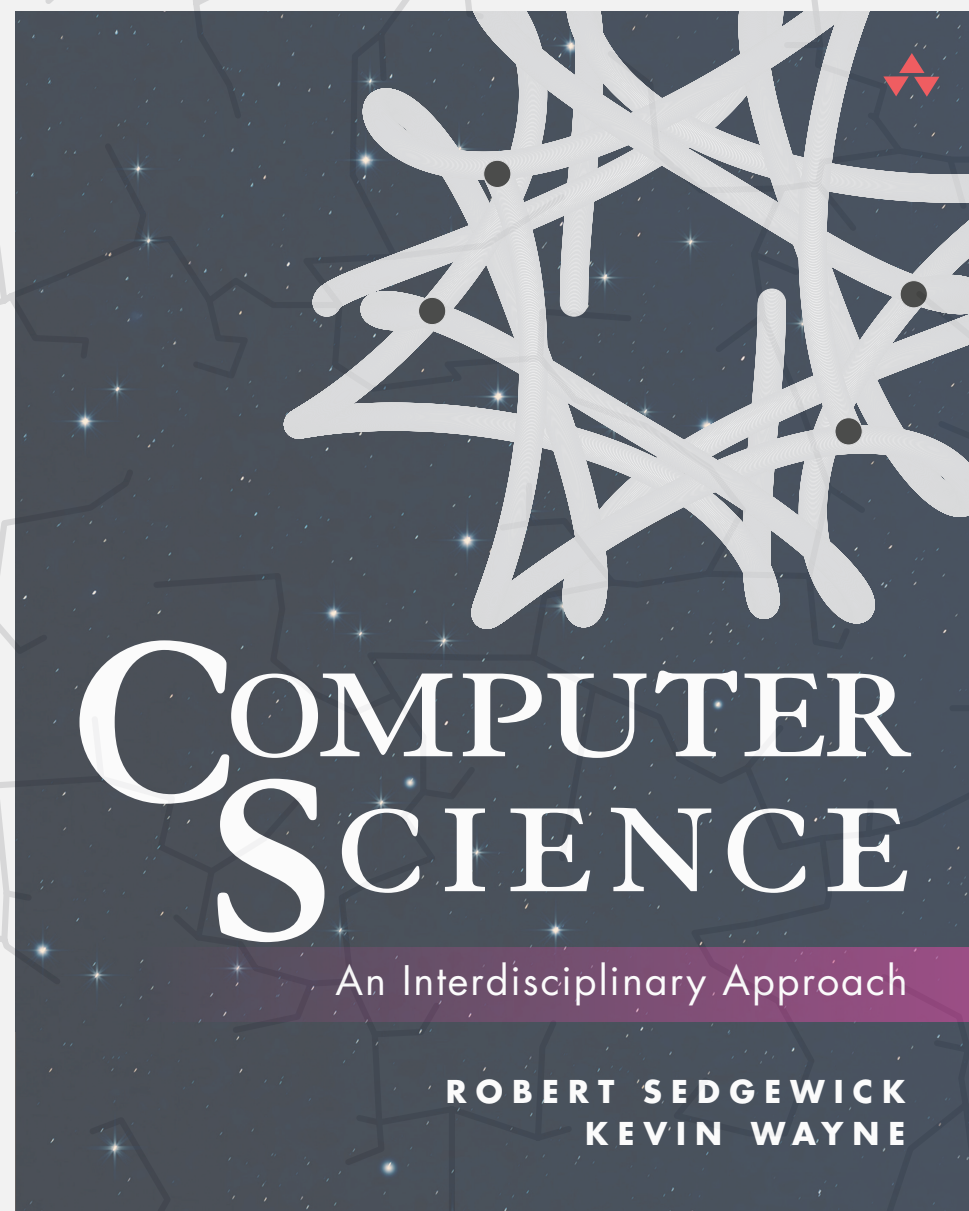
B. $(ab)^*$



C. $(a^*b)^+$



D. ba^*



<http://introcscs.princeton.edu>

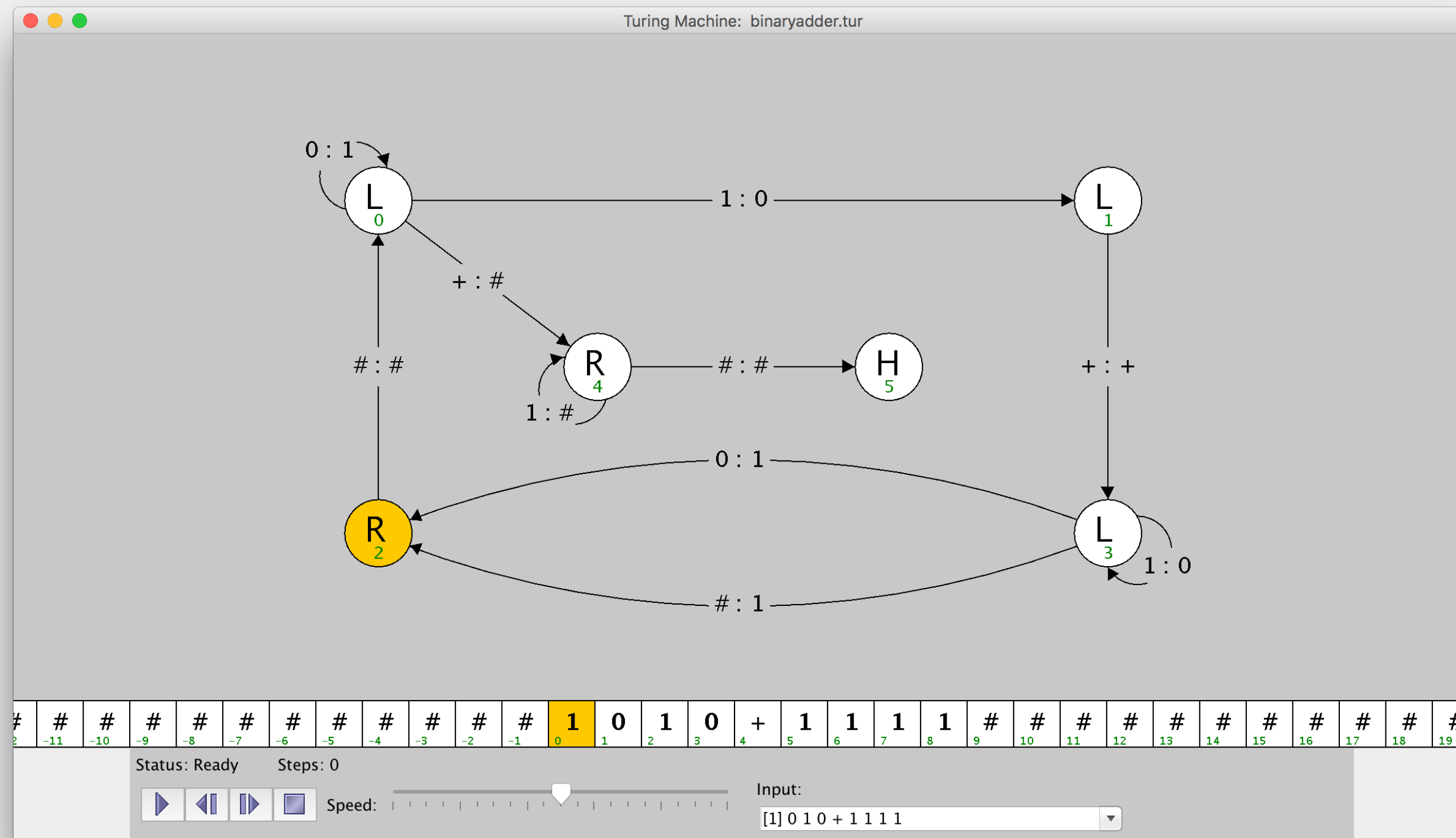
ALL QUESTIONS THEORY

- ▶ *REs, DFAs, and NFAs*
- ▶ *Universality and computability*
- ▶ *P, NP, NP-complete*

Big ideas

Turing machine. A simple, universal model of computation.

(similar to a DFA, but can read/write to tape, which is arbitrarily long)



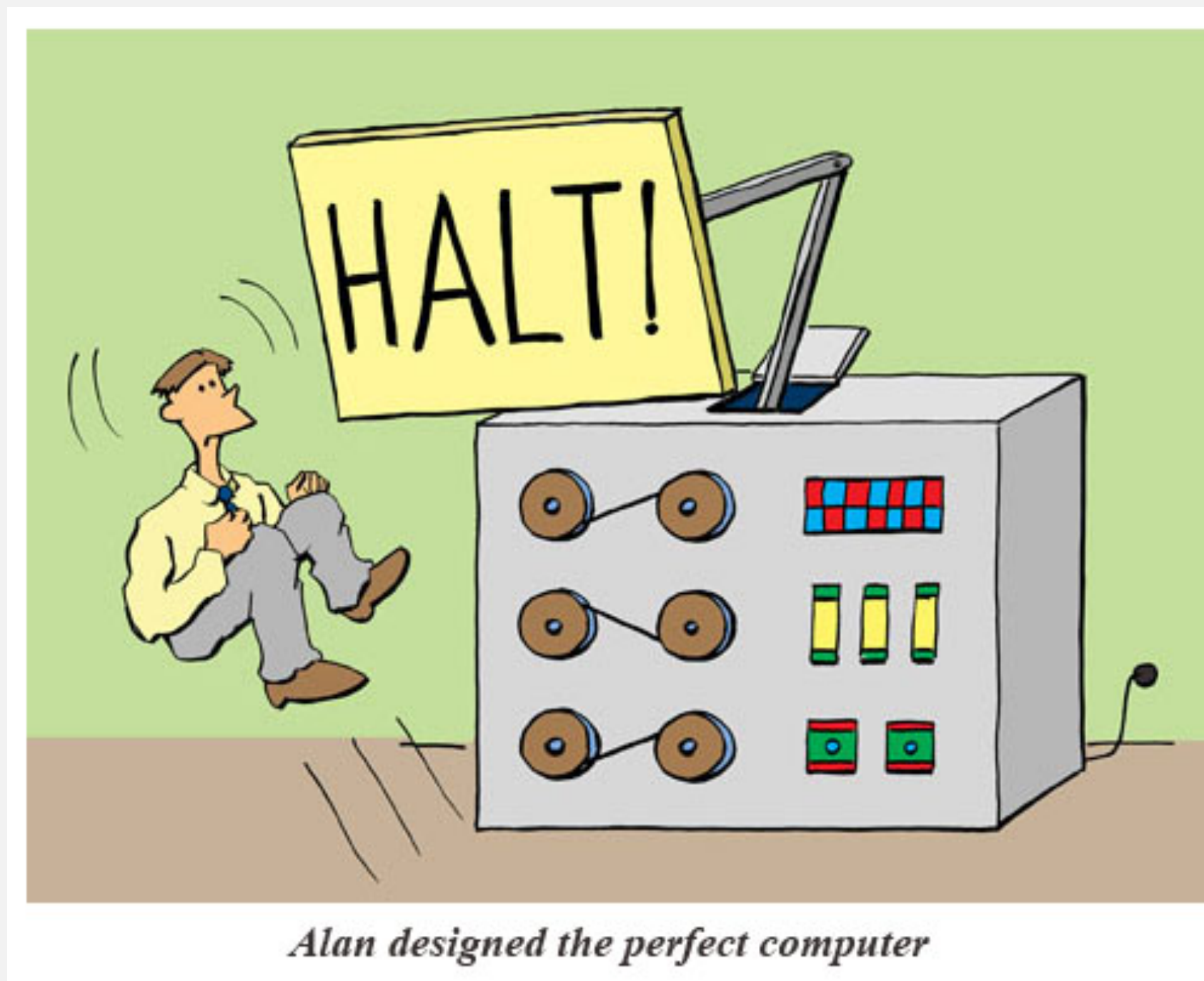
2. Effective calculability. Abbreviation of treatment. A function is said to be effectively calculable if its values can be found by some purely mechanical process. Although it is fairly easy to get an intuitive grasp of this idea it is nevertheless desirable to have some definite, mathematically expressible definition. Such a definition was first given by Gödel at Princeton in 1934 (Gödel [2], 26) following in part an unpublished suggestion of Herbrand, and has since been developed by Kleene (Kleene [2]). We shall not be concerned much here with this particular definition. Another definition of effective calculability has been given by Church (Church [3], 356-358) who identifies it with λ -definability. The author has recently suggested a definition corresponding more closely to the intuitive idea (Turing [1], see also Post [1]). It was said above "a function is effectively calculable if its values can be found by some purely mechanical process." We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine. It is possible to give a mathematical description, in a certain normal form, of the structures of these machines. The development of the idea leads to the author's definition of a computable function, and an identification of computability with effective calculability. (We shall use the expression 'computable function' to mean a function calculable by a machine, and let 'effectively calculable' refer to the intuitive idea without particular identification with any one of these definitions. We do not restrict the values taken by a computable function to be natural numbers; we may for instance have computable propositional functions.) It is not difficult though somewhat laborious, to prove these three definitions equivalent (Kleene [3], Turing [2]). In the present paper we shall make considerable use of Church's identification of effective calculability with λ -definability, or, what comes to the same, of the identification with computability and one of the equivalence theorems. In most cases where we have to deal with an effectively calculable function we shall introduce the corresponding W. F. F. with some such phrase as "the function f is effectively calculable, let F be a formula λ -defining it" or "let F be a formula such that $F(n)$ is convertible to \dots whenever n represents a positive integer". In such cases there is no difficulty in seeing how a machine could in principle be designed to calculate the values of the function concerned, and assuming this done the equivalence theorem can be applied. A statement as to what the formula F actually is may be omitted. We may introduce immediately on this basis a W. F. F. with the property that $\phi(n, m)$ conv. F if n is the greatest positive integer for which m' divides n , if any, and is 1 if there is none. We also introduce D with the properties: $D(n, n)$ conv. 2; $D(n, m)$ conv. 2; $D(n, m)$ conv. 1. There is another point to be made clear in connection with the point of view we are adopting. It is intended that all proofs that are given should be regarded no more critically than proofs in classical analysis. The subject matter, roughly speaking, is constructive systems of logic, but as the purpose is directed towards choosing a particular constructive system of logic for practical use, an attempt at this stage to put our theorems into constructive form would be putting the cart before the horse. Those computable functions which take only the values 0 and 1 are of particular importance since they determine and are determined by computable properties, as may be seen by replacing '0' and '1' by 'true' and 'false'. But besides this type of property we may have to consider a different type, which is roughly speaking, less constructive than the computable properties, but more so than the general predicates of classical mathematics. Suppose we have a computable function of the natural numbers taking natural numbers as values, then corresponding to this function there is the property of being a value of the function. Such a property we shall describe as 'axiomatic'; the reason for using this term is that it is possible to define such a property by giving a set of axioms, the property to hold for a given argument if and only if it is possible to deduce that it holds from the axioms. Axiomatic properties may also be characterized in this way. A property ψ of positive integers is axiomatic if and only if there is a computable property ϕ of two positive integers such that $\psi(x)$ is true if and only if there is a positive integer y such that $\phi(x, y)$ is true. Or again ψ is axiomatic if and only if there is a W. F. F. F such that $\psi(n)$ is true if and only if $F(n)$ conv. 2. Number theoretic theorems. By a number theoretic theorem I believe we there is no generally accepted meaning for this term, but it should be noticed that we are using it in a rather restricted sense. The most generally accepted meaning is probably this: suppose we take an arbitrary formula of the function calculus of first order and replace the function variables by primitive recursive relations. The resulting formula represents a typical number theoretic theorem in this [more general] sense; we shall mean a theorem of the form ' $\theta(x)$ vanishes for infinitely many natural numbers x ', where $\theta(x)$ is a primitive recursive function. (Primitive recursive functions of natural numbers are defined inductively as follows: "The class of primitive recursive functions is more restricted than the computable functions, but has the advantage that there is a process whereby one can tell of a set of equations whether it defines a primitive recursive function in the manner described above. If $\phi(x_1, \dots, x_n)$ is primitive recursive then $\psi(x_1, \dots, x_n) = 0$ is described as a primitive recursive between x_1, \dots, x_n ." We shall say that a problem is number theoretic if it has been shown that any solution of the problem may be put in the form of a proof of one or more number theoretic theorems. More accurately we may say that a class of problems is number theoretic if the solution of any one of them can be transformed by a uniform process into the form of proofs of number theoretic theorems. I shall now draw a few consequences from the definitions of 'number theoretic theorems', and in section 5 will try to justify confining our considerations to this type of problem. An alternative form for number theoretic theorems is 'for each natural number x there exists a natural number y such that $\phi(x, y)$ vanishes', where $\phi(x, y)$ is primitive recursive and conversely. In other words, there is a rule whereby given the function $\theta(x)$ we can find a function $\phi(x, y)$, or given $\psi(x, y)$ we can find a function $\theta(x)$, so that ' $\theta(x)$ vanishes for each x ' is a necessary and sufficient condition for 'for each x there is a y so that $\phi(x, y) = 0$ '. In fact given $\theta(x)$ we define $\phi(x, y) = \theta(y) + x(x, y)$ where $x(x, y)$ is the (primitive recursive) function with the properties $x(x, y) = 1$ ($y < x$); $x(x, y) = 0$ ($y > x$). If on the other hand we are given $\psi(x, y)$ we define $\theta(x)$ by the equations $\theta(0) = 3$; $\theta(x + 1) = 3 \cdot 2^3 \cdot \theta(x)^2$; $\theta(x) = 1$, $\theta(x) = 1$, $\theta(x) = 1$; $\theta(x) = 0$, $\theta(x) = 1$, $\theta(x) = 1$, $\theta(x) = 1$ where $\theta(x)$ is to be defined so as to mean the largest x for which $\psi(x, 3)$ is defined primitive recursively so as to have its usual meaning if x is a multiple of 3. The function $\delta(x)$ is to be defined by the equations $\delta(0) = 0$, $\delta(x + 1) = 1$. It is easily verified that the functions so defined have the desired properties. We shall now show that questions as to the truth of statements of form 'does $f(x)$ vanish identically', where $f(x)$ is a computable function, can be reduced to questions as to the truth of number theoretic theorems. It is understood that in each case the rule for the calculation of $f(x)$ is given and that one is satisfied that this rule is valid, i.e. that the machine which should calculate $f(x)$ is circle free (Turing [1], 232). The function $f(x)$ being computable is general recursive in the Herbrand-Gödel sense, and therefore by a general theorem due to Kleene (Kleene [2], 727) "Suppose $\phi(x_1, \dots, x_n)$ is primitive recursive then $\psi(x_1, \dots, x_n)$ is primitive recursive if it is defined by one of the sets of equations (a) - (e). (a) $\psi(x_1, \dots, x_n) = h(x_1, \dots, x_n, \phi(x_1, \dots, x_n))$, ($1 \leq m < n$); (b) $\psi(x_1, \dots, x_n) = f(x_1, \dots, x_n)$; (c) $\psi(x_1, \dots, x_n) = a$, where $n = 1$ and a is some particular natural number; (d) $\psi(x_1, \dots, x_n) = x_1 + 1$ ($n = 1$); (e) $\psi(x_1, \dots, x_n) = f(\phi(x_1, \dots, x_n))$; $\psi(x_1, \dots, x_n) = h(\phi(x_1, \dots, x_n))$. LMC

Relevance. Provides rigorous definition of a computer program; enables study of computation.

Big ideas

Halting problem. Given a Java program (or TM) and its input, does that program halt when run on that input?

Computability. The halting problem is **undecidable**—it's impossible to write a Java program (or TM) to solve.

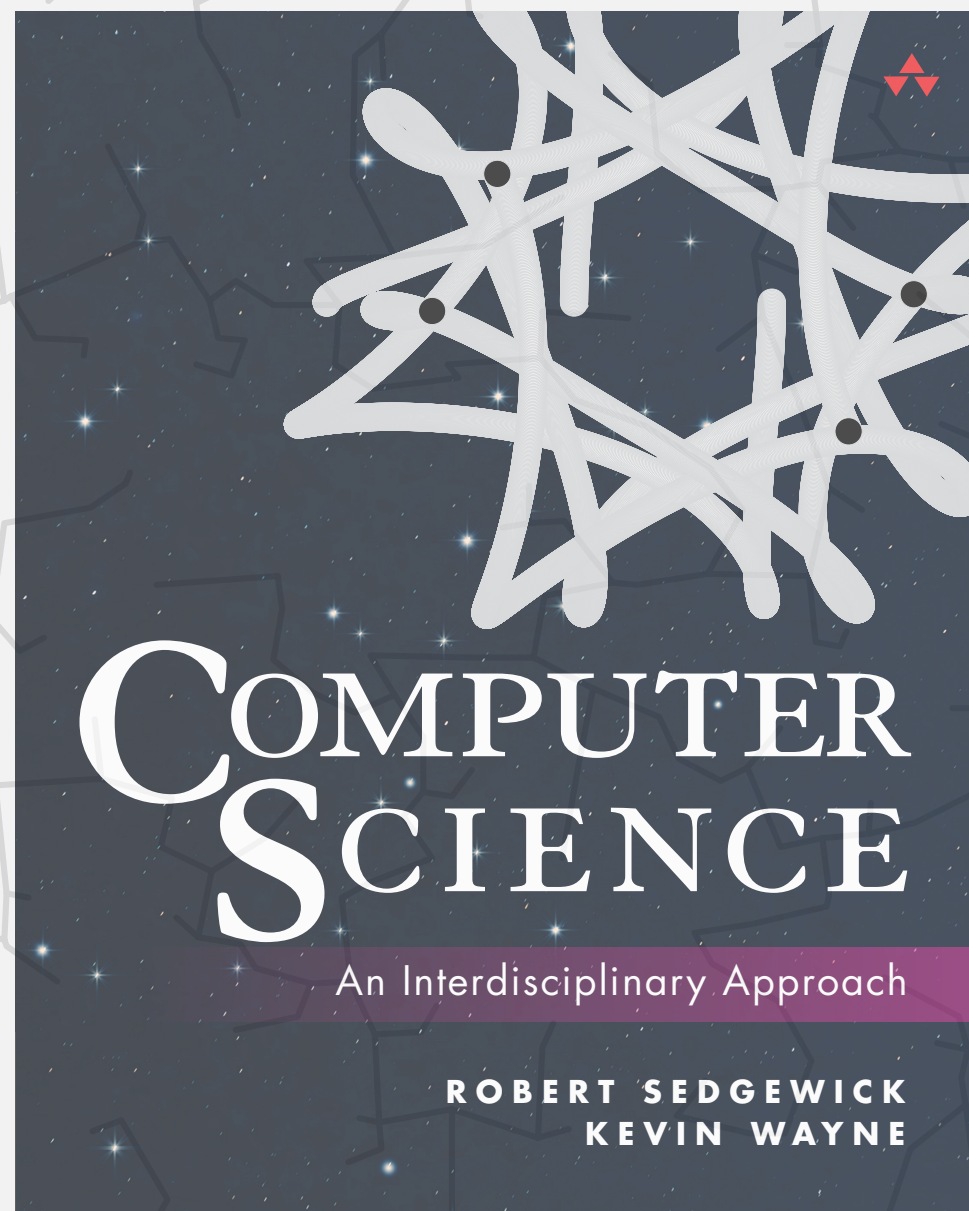


2. **Effective calculability.** A function is said to be effectively calculable if its values can be found by some purely mechanical process. Although it is fairly easy to get an intuitive grasp of this idea it is nevertheless desirable to have some definite, mathematically expressible definition. Such a definition was first given by Gödel at Princeton in 1934 (Gödel [2], 26) following in part an unpublished suggestion of Herbrand, and has since been developed by Kleene (Kleene [2]). We shall not be concerned much here with this particular definition. Another definition of effective calculability has been given by Church (Church [3], 356-358) who identifies it with λ -definability. The author has recently suggested a definition corresponding more closely to the intuitive idea (Turing [1], see also Post [1]). It was said above "a function is effectively calculable if its values can be found by some purely mechanical process." We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine. It is possible to give a mathematical description, in a certain normal form, of the structures of these machines. The development of the idea leads to the author's definition of a computable function, and an identification of computability with effective calculability. (We shall use the expression 'computable function' to mean a function calculable by a machine, and let 'effectively calculable' refer to the intuitive idea without particular identification with any one of these definitions. We do not restrict the values taken by a computable function to be natural numbers; we may for instance have computable propositional functions.) It is not difficult though somewhat laborious, to prove these three definitions equivalent (Kleene [3], Turing [2]). In the present paper we shall make considerable use of Church's identification of effective calculability with λ -definability, or, what comes to the same, of the identification with computability and one of the equivalence theorems. In most cases where we have to deal with an effectively calculable function we shall introduce the corresponding W, F, F, with so me such phrase as "the function f is effectively calculable, let F be a formula λ -defining it" or "let F be a formula such that $F(n)$ is convertible to \dots whenever n represents a positive integer". In such cases there is no difficulty in seeing how a machine could in principle be designed to calculate the values of the function concerned, and assuming this done the equivalence theorem can be applied. A statement as to what the formula F actually is may be omitted. We may introduce immediately on this basis a W, F, F, with the property that $\phi(n, m)$ conv 2, if n is the greatest positive integer for which m' divides n , if any, and is 1 if there is none. We also introduce D with the properties: $D(n, n)$ conv 2; $D(n, m)$ conv 1. There is another point to be made clear in connection with the point of view we are adopting. It is intended that all proofs that are given should be regarded no more critically than proofs in classical analysis. The subject matter, roughly speaking, is constructive systems of logic, but as the purpose is directed towards choosing a particular constructive system of logic for practical use, an attempt at this stage to put our theorems into constructive form would be putting the cart before the horse. Those computable functions which take only the values 0 and 1 are of particular importance since they determine and are determined by computable properties, as may be seen by replacing '0' and '1' by 'true' and 'false'. But besides this type of property we may have to consider a different type, which is roughly speaking, less constructive than the computable properties, but more so than the general predicates of classical mathematics. Suppose we have a computable function of the natural members taking natural numbers as values, then corresponding to this function there is the property of being a value of the function. Such a property we shall describe as 'axiomatic'; the reason for using this term is that it is possible to define such a property by giving a set of axioms, the property to hold for a given argument if and only if it is possible to deduce that it holds from the axioms. Axiomatic properties may also be characterized in this way. A property ψ of positive integers is axiomatic if and only if there is a computable property ϕ of two positive integers such that $\psi(x)$ is true if and only if there is a positive integer y such that $\phi(x, y)$ is true. Or again ψ is axiomatic if and only if there is a W, F, F, E such that $\psi(n)$ is true if and only if $E(n)$ conv 2. 3. **Number theoretic theorems.** By a number theoretic theorem I believe we there is no generally accepted meaning for this term, but it should be noticed that we are using it in a rather restricted sense. The most generally accepted meaning is probably this: suppose we take an arbitrary formula of the function calculus of first order and replace the function variables by primitive recursive relations. The resulting formula represents a typical number theoretic theorem in this [more general] sense. We shall mean a theorem of the form ' $\theta(x)$ vanishes for infinitely many natural numbers x ', where $\theta(x)$ is a primitive recursive function. (Primitive recursive functions of natural numbers are defined inductively as follows: "The class of primitive recursive functions is more restricted than the computable functions, but has the advantage that there is a process whereby one can tell of a set of equations whether it defines a primitive recursive function in the manner described above. If $\phi(x_1, \dots, x_n)$ is primitive recursive then $\phi(x_1, \dots, x_n) = 0$ is described as a primitive recursive between x_1, \dots, x_n ." We shall say that a problem is number theoretic if it has been shown that any solution of the problem may be put in the form of a proof of one or more number theoretic theorems. More accurately we may say that a class of problems is number theoretic if the solution of any one of them can be transformed (by a uniform process) into the form of proofs of number theoretic theorems. I shall now draw a few consequences from the definitions of 'number theoretic theorems', and in section 5 will try to justify confining our considerations to this type of problem. An alternative form for number theoretic theorems is 'for each natural number x there exists a natural number y such that $\phi(x, y)$ vanishes', where $\phi(x, y)$ is primitive recursive and conversely. In other words, there is a rule whereby given the function $\theta(x)$ we can find a function $\phi(x, y)$, or given $\psi(x, y)$ we can find a function $\theta(x)$, so that ' $\theta(x)$ vanishes infinitely often' is a necessary and sufficient condition for 'for each x there is y so that $\phi(x, y) = 0$ '. In fact given $\theta(x)$ we define $\phi(x, y) = \theta(y) + x(x, y)$ where $x(x, y)$ is the (primitive recursive) function with the properties $\alpha(x, y) = 1$ ($y < x$); $\alpha(x, y) = 0$ ($y > x$). If on the other hand we are given $\phi(x, y)$ we define $\theta(x)$ by the equations $\theta(0) = 3$; $\theta(x+1) = 2, 2, 3$ ($\theta(x)$) ($\phi(x, 0)$); $\theta(x) = 1, \alpha_0$ ($\theta(x)$); $\theta(x) = \phi(x, \alpha_0)$ ($\theta(x) - 1, \alpha_0$ ($\theta(x)$)) where $\alpha_0(x)$ is to be defined so as to mean the largest x' for which α_0 divides x' to be defined primitive recursively so as to have its usual meaning if x is a multiple of 3. The function $\delta(x)$ is to be defined by the equations $\delta(0) = 0$, $\delta(x+1) = 1$. It is easily verified that the functions so defined have the desired properties. We shall now show that questions as to the truth of statements of form 'does $f(x)$ vanish identically', where $f(x)$ is a computable function, can be reduced to questions as to the truth of number theoretic theorems. It is understood that in each case the rule for the calculation of $f(x)$ is given and that one is satisfied that this rule is valid, i.e., that the machine which should calculate $f(x)$ is circle free (Turing [1], 232). The function $f(x)$ being computable is general recursive in the Herbrand-Gödel sense, and therefore by a general theorem due to Kleene (Kleene [2], 177) *Suppose $\phi(x_1, \dots, x_n)$ and $\psi(x_1, \dots, x_n)$ are primitive recursive then $\phi(x_1, \dots, x_n)$ is primitive recursive if it is defined by one of the sets of equations (a) - (e). (a) $\phi(x_1, \dots, x_n) = h(x_1, \dots, x_n, \psi(x_1, \dots, x_n))$, ($1 \leq m < n$); (b) $\phi(x_1, \dots, x_n) = f(x_1, \dots, x_n)$; (c) $\phi(x_1, \dots, x_n) = a$, where $n = 1$ and a is some particular natural number; (d) $\phi(x_1, \dots, x_n) = x_1 + 1$ ($\psi = 1$); (e) $\phi(x_1, \dots, x_n) = f(\psi(x_1, \dots, x_n))$; $\phi(x_1, \dots, x_n) = h(x_1, \dots, x_n, \psi(x_1, \dots, x_n))$. LMC

Church-Turing thesis (modern interpretation). A TM can perform any computation (decide a language or compute a function) that can be described by any physically realizable computing device.

Spring 2012, Question 8 and Fall 2012, Question 9

	TRUE	FALSE	UNKNOWN
For any language that some TM decides, there exists a DFA that recognizes the same language.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
For any language that some RE describes, there exists a TM that decides the same language.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
For any function that some TM computes, there exists a Java program that computes the same function.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Most computer scientists believe that the Church-Turing thesis will eventually be proved true.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
No future computer can solve the halting problem.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>



<http://introc.cs.princeton.edu>

ALL QUESTIONS THEORY

- ▶ *REs, DFAs, and NFAs*
- ▶ *Universality and computability*
- ▶ ***P, NP, NP-complete***

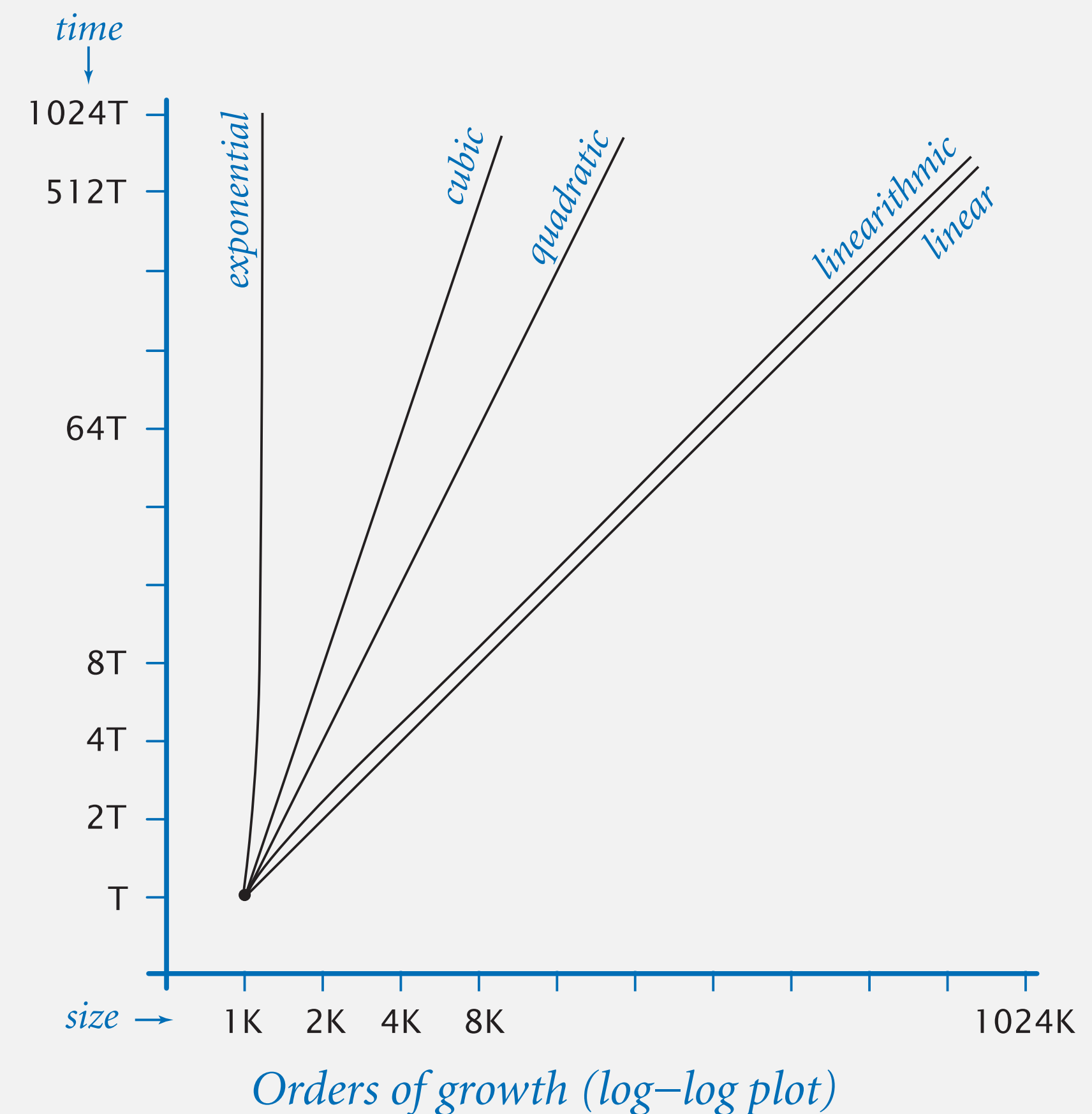
Big ideas (polynomial time vs. exponential time)

Polynomial-time algorithm. Running time $\leq a n^b$ for all inputs of size n .

Exponential-time algorithm. Running time $\geq 2^n$ for infinitely many inputs.

or $1.1^{\sqrt{n}}$ or $n!$

Relevance. "Efficient in practice."



Polynomial time vs. exponential time

Classify each algorithm.

	POLYNOMIAL	EXPONENTIAL
mergesort (to sort an array of n elements)	<input type="radio"/>	<input type="radio"/>
insertion sort (to sort an array of n elements)	<input type="radio"/>	<input type="radio"/>
recursive H-tree program (to draw an H-tree of order n)	<input type="radio"/>	<input type="radio"/>
smallest increase heuristic (to build a TSP tour of n points)	<input type="radio"/>	<input type="radio"/>
brute-force TSP algorithm (that tries all $n!$ permutations)	<input type="radio"/>	<input type="radio"/>

Big ideas (search problems)

Search problem. There exists a poly-time algorithm that checks whether a given solution solves a given instance of the problem.

FACTOR. Given an n -digit integer x , find a factor (other than 1 and x).

Q. How to show that FACTOR is a search problem?

A. Given a purported factor d , need a poly-time algorithm to check that

- d is not equal to 1 or x .
- d is a divisor of x .

grade-school division is n^2
(faster algorithms are known)

Instance. $x = 305753$

Factor? $d = 31$

Handwritten long division showing the division of 305753 by 31. The quotient is 9863. The digits of the quotient are color-coded: 9 (blue), 8 (red), 6 (green), 3 (black). The steps of the division are as follows:

$$\begin{array}{r} 9863 \\ 31 \overline{) 305753} \\ \underline{279} \\ 26753 \\ \underline{248} \\ 1953 \\ \underline{186} \\ 93 \\ \underline{93} \\ 0 \end{array}$$

Big ideas (search problems)

Search problem. There exists a poly-time algorithm that checks whether a given solution solves a given instance of the problem.

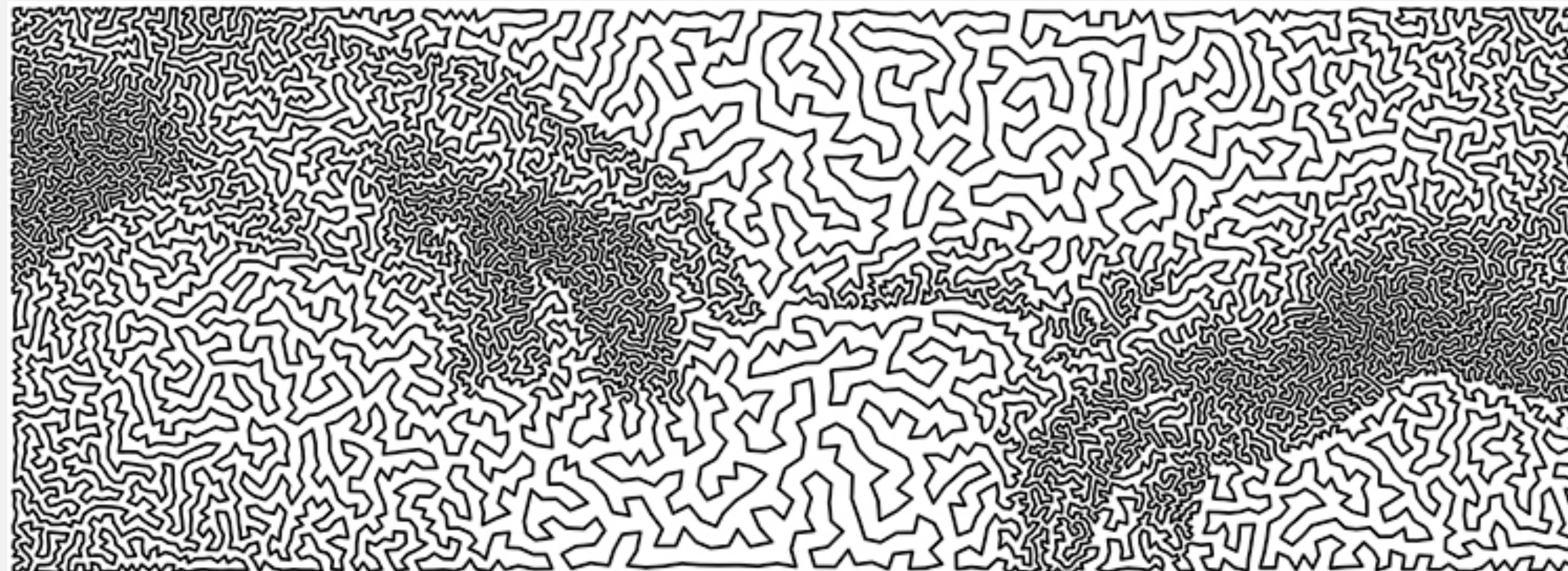
TSP. Given n points in the plane and an integer L , find a tour of length at most L .

Q. How to show that TSP is a search problem?

A. Given a purported tour x , need a poly-time algorithm to verify that

- x is a tour (i.e, a permutation of the n points).
- The length of x is at most L .

distance between two points
is Euclidean distance,
rounded to the nearest inch



Big ideas (P vs. NP)

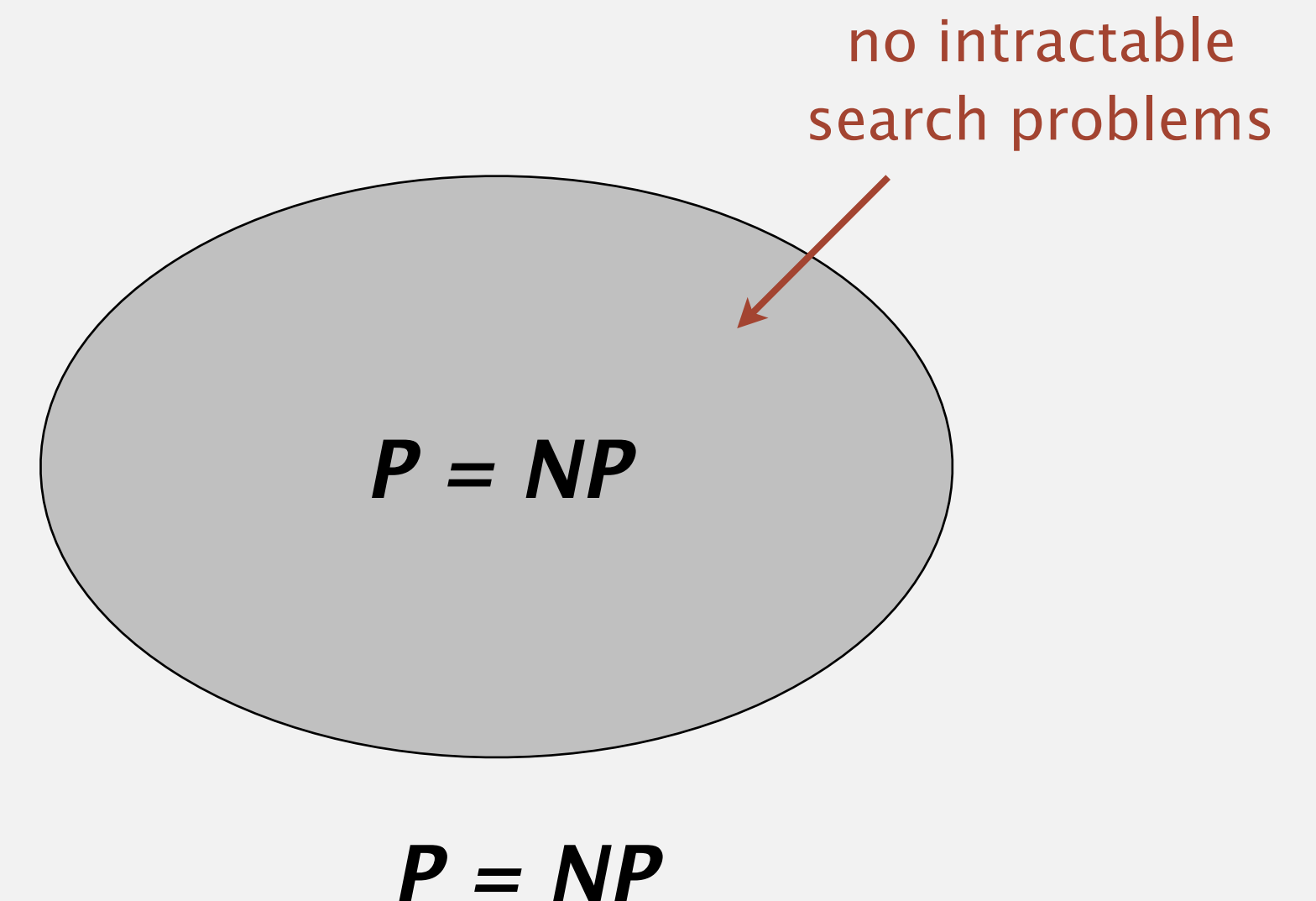
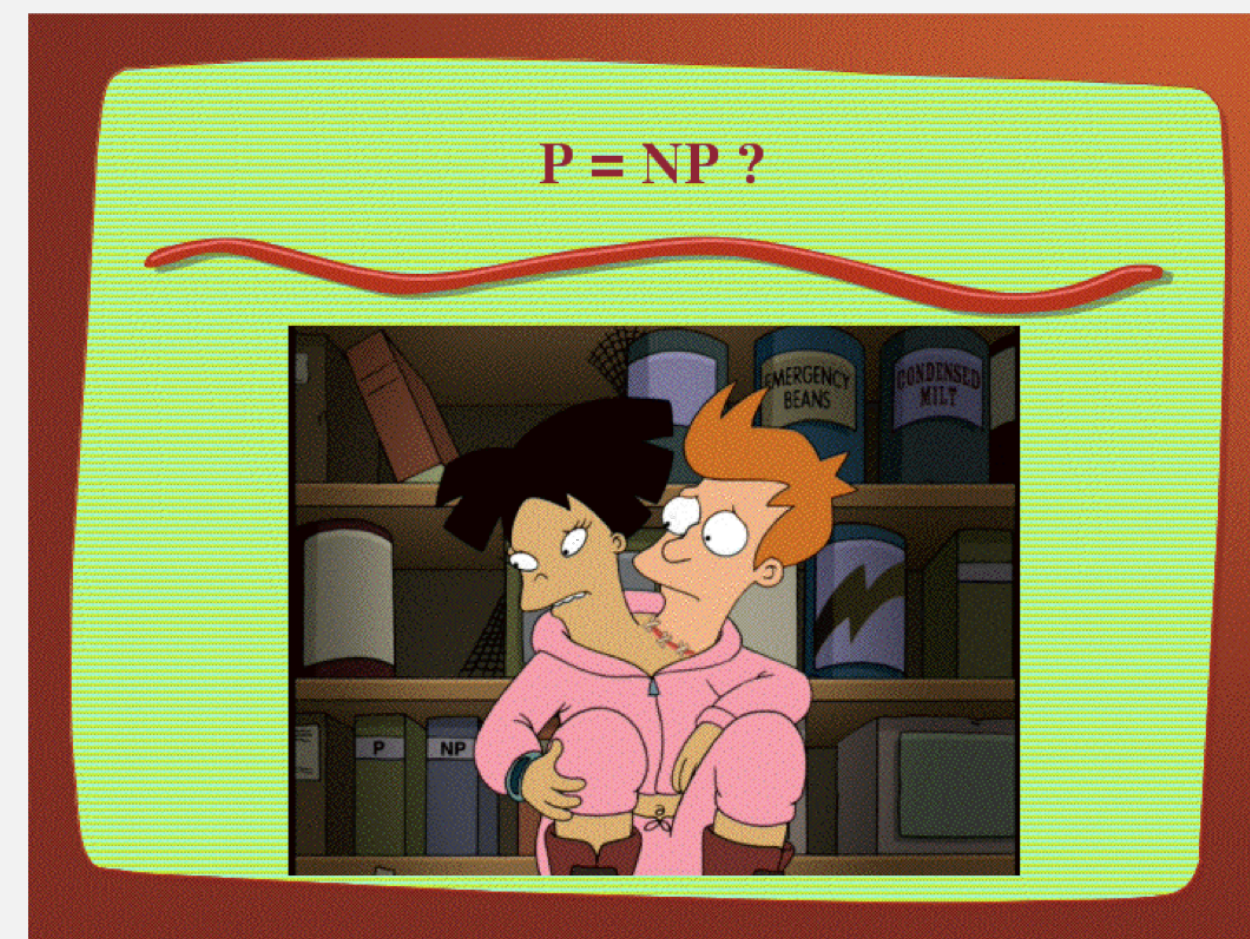
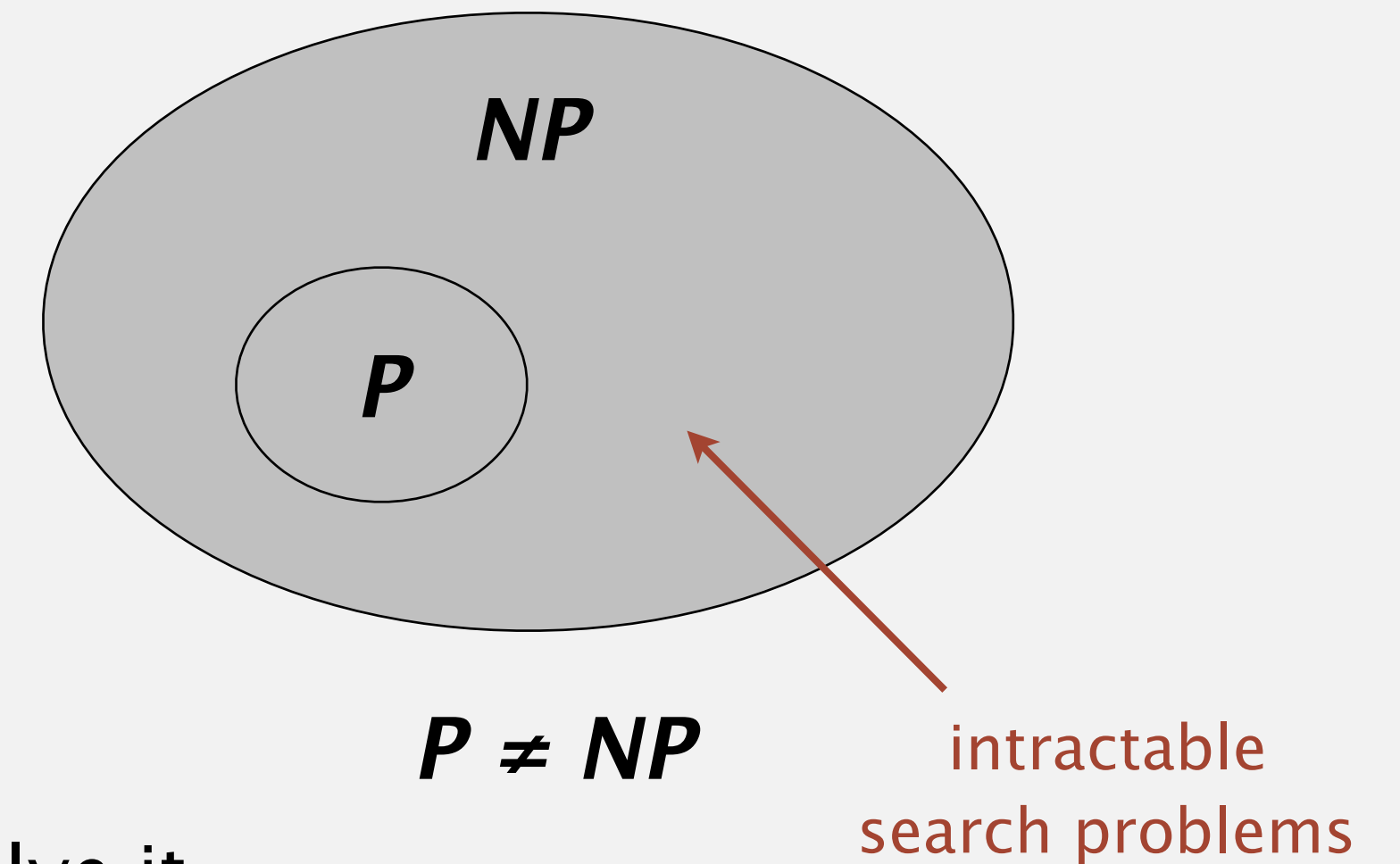
NP. The set of all search problems.

P. The set of all search problems that can be solved in polynomial time.

Relevance. **NP** contains problems that we aspire to solve in practice;
P contains problems that we can solve in practice.

Definition. A problem is **intractable** if there exists no poly-time algorithm to solve it.

Famous conjecture. $P \neq NP$ [There exist intractable search problems]

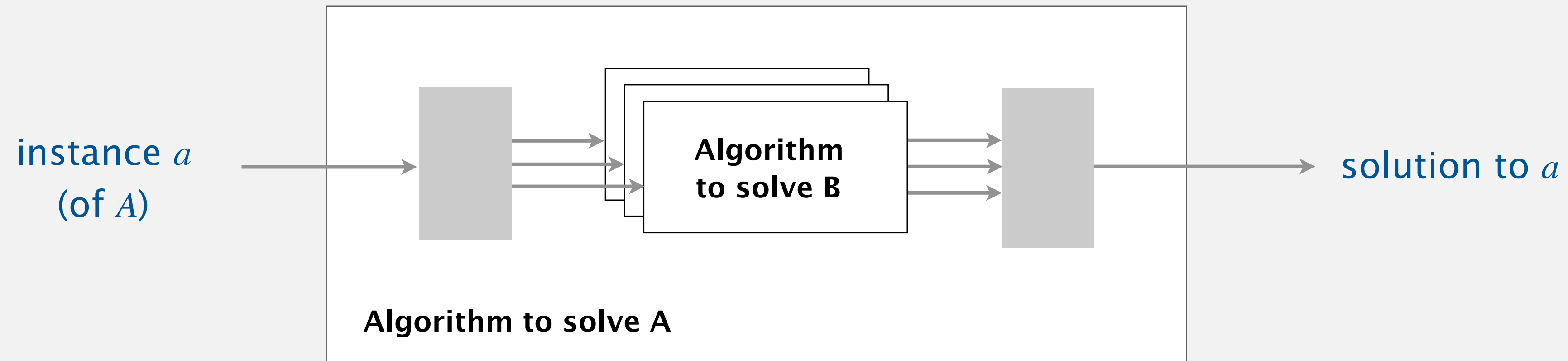


Fall 2014, Question 8

	TRUE	FALSE	UNKNOWN
TSP is in <i>NP</i> .	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Every problem in <i>P</i> is also in <i>NP</i> .	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Every problem in <i>NP</i> is also in <i>P</i> .	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Every problem in <i>NP</i> can be solved with an exponential-time algorithm.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The halting problem is in <i>NP</i> .	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
If $P = NP$, then there exists a poly-time algorithm for FACTOR.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Big ideas (poly-time reductions)

Definition. A problem A **poly-time reduces** to a problem B if there exists an algorithm for A that uses a polynomial number of calls to a subroutine for B , plus polynomial time outside of those subroutine calls.



**A poly-time reduces to B
(A can be solved by B)**

Intuition. Can solve B efficiently \Rightarrow can also solve A efficiently.

Contrapositive. Can't solve A efficiently \Rightarrow can't solve B efficiently either.

Definition. A search problem B is **NP-complete** if every search problem A poly-time reduces to B .

Spring 2015, Question 5a

Suppose that problem A poly-time reduces to problem B . Which of the following statements can we infer?

	TRUE	FALSE
If B is in P , then A is in P .	<input type="radio"/>	<input type="radio"/>
If A is in P , then B is in P .	<input type="radio"/>	<input type="radio"/>
If A is NP -complete, then B is NP -complete.	<input type="radio"/>	<input type="radio"/>
If A is NP -complete and B is in NP , then B is NP -complete.	<input type="radio"/>	<input type="radio"/>
A and B cannot both be NP -complete.	<input type="radio"/>	<input type="radio"/>