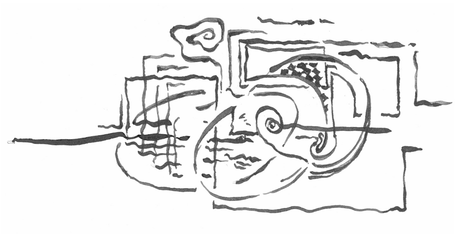
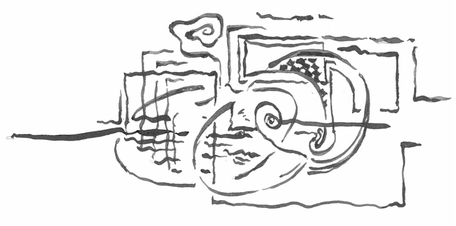
Algorithmic Adventures

From Knowledge to Magic



Book by Juraj Hromkovič, ETH Zurich
Slides by Tom Verhoeff, TU Eindhoven

Quotation



Discuss, commit errors, make mistakes,
but for God's sake think –
even if you should be wrong –
but think your own thoughts.

Gotthold Ephraim Lessing

How Large Is the Set of All Texts?

There are infinitely many texts, but what kind of infinity?

A text is a sequence of **symbols** from an enumerable **alphabet** A
(often: from a *finite* alphabet, cf. ASCII keyboard)

Each text can be encoded in a **tuple of natural numbers**,
by representing each symbol of A by a unique natural number

`Juraj & Tom` $\longrightarrow$ (74, 117, 72, 97, 106, 32, 38, 32, 84, 111, 109)

The number of all texts over A is equal to $|\mathbb{N}^*| = |\mathbb{N}|$ (Ch. 3)

How Large Is the Set of All Programs?

Every program is a text over some suitable enumerable alphabet A

Not every text over A is program: it must be **syntactically correct** according to the rules of the **programming language**

A **compiler** checks the syntactical correctness of a text as a program (but not *semantical* correctness: whether the text is an *algorithm*)

Construct an enumeration* of all programs from an enumeration of all texts over A by deleting all texts that are not syntactically correct:

$$P_0,\ P_1,\ P_2,\ \ldots,\ P_i,\ \ldots$$

where P_i denotes the i-th program

Every algorithm† appears in the sequence, not every P_i is an algorithm

*Each programming language gives rise to its own enumeration

†Provided the programming language is sufficiently expressive

Problem(c)

For real number c, **Problem(c)** is defined by

Input:	a natural number $n \in \mathbb{N}$
Output:	the number c up to n decimal digits after the decimal point

Algorithm A_c solves Problem(c) when

> for any given $n \in \mathbb{N}$, A_c outputs all digits of c before the decimal point and the first n digits of c after the decimal point

N.B. c is not an input of the problem, but a 'built-in' constant

E.g., $A_{\sqrt{2}}$ with input $n = 5$ must output 1.41421

Not All Problem(c) Are Algorithmically Solvable

Because $|\mathbb{R}| > |\mathbb{N}|$, there are more algorithmic tasks than algorithms*:

There exist $c \in \mathbb{R}$ such that Problem(c) is not algorithmically solvable

Real numbers having a finite representation are exactly the numbers that can be algorithmically generated

There exist real numbers that do not possess a finite representation and so are not computable (algorithmically generable)

For these unsolvable problems, c is not explicitly specifiable

Are there other (more interesting) algorithmically unsolvable tasks?

*Note that no algorithm can solve more than one Problem(c)

Decision Problem $(\mathbb{N}, M)$

For $M \subseteq \mathbb{N}$, **decision problem $(\mathbb{N}, M)$** is defined by

Input:	a natural number $n \in \mathbb{N}$
Output:	$\texttt{YES}$ if $n \in M$
	$\texttt{NO}$ if $n \notin M$

Example: for *primality testing* take $M := \{2, 3, 5, 7, 11, 13, 17, 19, \ldots\}$

Algorithm A solves decision problem $(\mathbb{N}, M)$ when

for any given $n \in \mathbb{N}$, A outputs $\texttt{YES}$ if $n \in M$ and $\texttt{NO}$ if $n \notin M$

$(\mathbb{N}, M)$ is called **decidable** when there exists an algorithm to solve it, and **undecidable** if no such algorithm exists

Not All Problems $(\mathbb{N}, M)$ Are Decidable

Because $|\mathcal{P}(\mathbb{N})| > |\mathbb{N}|$, there are more problems $(\mathbb{N}, M)$ than algorithms

There exist $M \subseteq \mathbb{N}$ such that $(\mathbb{N}, M)$ is undecidable

Define set $\boxed{DIAG = \{\, i \in \mathbb{N} \mid \text{program } P_i \text{ does } \textit{not} \text{ output } \mathrm{YES} \text{ on input } i \,\}}$

N.B. Each way of enumerating all programs, gives rise to its own set $DIAG$

Problem $(\mathbb{N}, DIAG)$ is undecidable, because

no program P_i implements an algorithm A that solves $(\mathbb{N}, DIAG)$:

$$
\begin{array}{ll}
 & A \text{ outputs } \mathrm{YES} \text{ on input } i \\
\Rightarrow & \qquad [\text{ by definition of "A solves } (\mathbb{N}, DIAG)\text{" }] \\
 & i \in DIAG \\
\Rightarrow & \qquad [\text{ by definition of } DIAG\,] \\
 & P_i \text{ does } \textit{not} \text{ output } \mathrm{YES} \text{ on input } i
\end{array}
$$

Comparing Problems for Algorithmic Solvability

By definition, the following statements are equivalent:

- Problem U_1 is *easier than or as hard as* problem U_2
- Problem U_1 is *no harder than* problem U_2
- $U_1 \leq_{\mathsf{Alg}} U_2$
- Algorithmic solvability of U_2 implies algorithmic solvability of U_1
  (Note the order of U_2 and U_1 here)
- It is *not* the case that:

  U_2 is algorithmically solvable and U_1 is not algorithmically solvable

How to Prove $U_1 \leq_{\textsf{Alg}} U_2$?

Question Can you prove $U_1 \leq_{\mathsf{Alg}} U_2$ without knowing about the algorithmic solvability of U_1 and U_2?

Answer Yes, via problem reduction:

Reduce algorithmic solvability of U_1 to that of U_2

Provide a solution for U_1 *in terms of* a *hypothetic* solution for U_2

U_1 can be algorithmically reduced to U_2 $\Rightarrow$ $U_1 \leq_{\mathsf{Alg}} U_2$

N.B. The converse implication does not necessarily hold

It is not necessary to know whether U_2 is solvable, and
if U_2 is solvable, it is not necessary to know how to solve U_2

Examples for Proving $U_1 \leq_{\text{Alg}} U_2$ by Algorithmic Reduction

Example 1 $\quad U_1 \;:\; ?_x : a \neq 0 : ax^2 + bx + c = 0$

$U_2 \;:\; ?_x :: x^2 + 2px + q = 0$

Solve U_1 by taking $p, q := \frac{b}{2a}, \frac{c}{a}$ in an algorithm for U_2, if it exists

Thus, $U_1 \leq_{\text{Alg}} U_2$ N.B. Also $U_2 \leq_{\text{Alg}} U_1$, by reduction $a, b, c := 1, 2p, q$

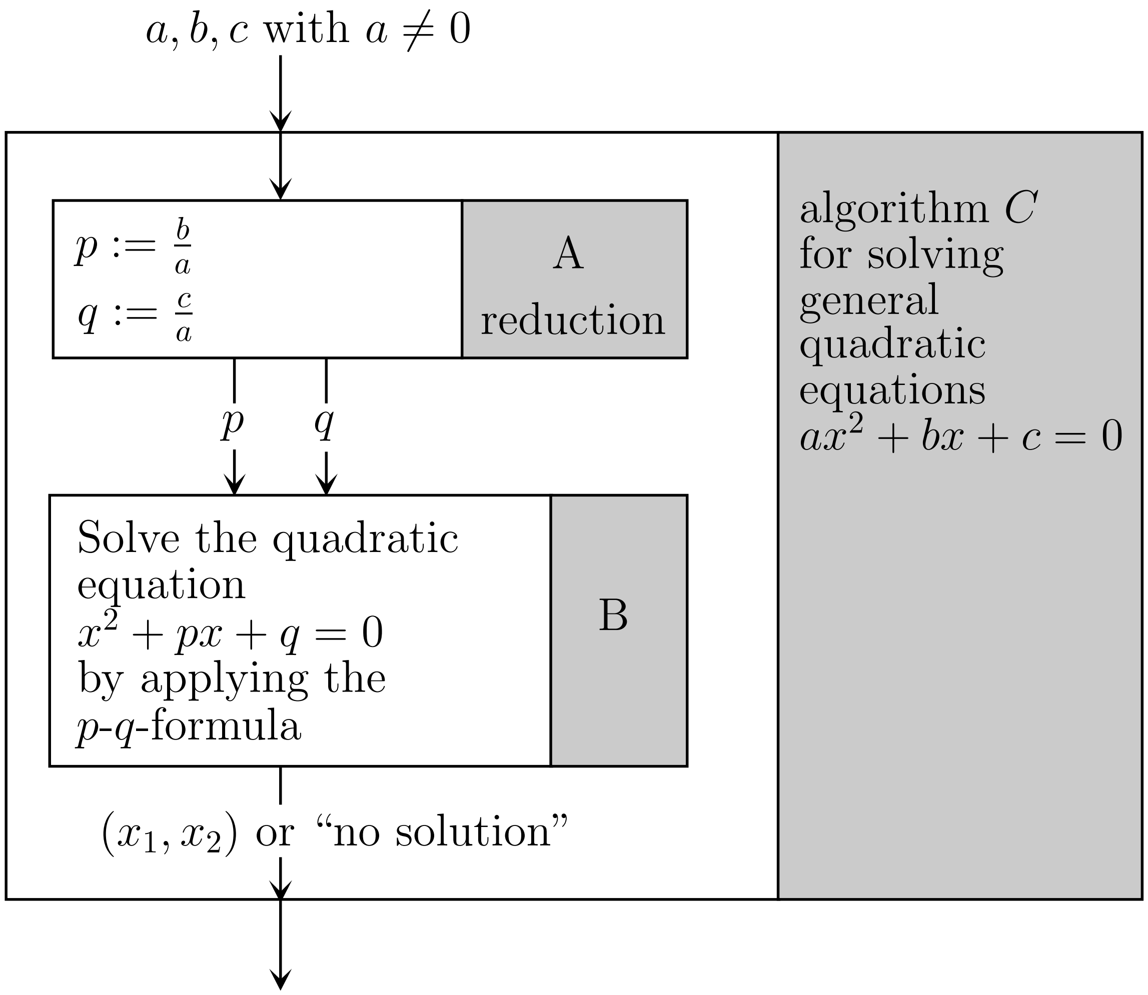
Example 2 $\quad U_1$: $?_x : a_5 \neq 0 : \sum_{i=0}^{5} a_i x^i = 0$ (5-th degree equation)

U_2: $?_x : b_6 \neq 0 : \sum_{i=0}^{6} b_i x^i = 0$ (6-th degree equation)

Solve U_1 by taking $b_i := a_{i-1} - a_i$ with $a_6 = a_{-1} = 0$ in an algorithm for U_2 and dropping result $x = 1$: $(x-1)\sum_{i=0}^{5} a_i x^i = \sum_{i=0}^{6} (a_{i-1} - a_i) x^i$

Thus, $U_1 \leq_{\text{Alg}} U_2$ (N.B. Also $U_2 \leq_{\text{Alg}} U_1$, but not by reduction)

Diagram for Problem Reduction



How to Use $U_1 \leq_{\textsf{Alg}} U_2$?

Assumption We know $U_1 \leq_{\mathsf{Alg}} U_2$

i.e. solvability of problem U_2 implies solvability of problem U_1

Question How can we use that knowledge?

Answer In two ways:

1. If you solve problem U_2, then you know that U_1 is solvable as well

   But you do not necessarily then also know *how* to solve U_1
   If you have a *reduction* of U_1 to U_2, you do know how to solve U_1

2. If you know U_1 is *not* solvable, then you know the same about U_2

Examples for Using $U_1 \leq_{\mathsf{Alg}} U_2$

1. We know $?_x : a \neq 0 : ax^2 + bx + c = 0$ $\leq_{\mathsf{Alg}}$ $?_x :: x^2 + 2px + q = 0$

   The second problem is solvable: $x = -p \pm \sqrt{p^2 - q}$ when $p^2 - q \geq 0$

   Hence, first problem is solvable: $x = \frac{-b}{2a} \pm \sqrt{\left(\frac{b}{2a}\right)^2 - \frac{c}{a}} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

2. We know $?_x : a_5 \neq 0 : \sum_{i=0}^{5} a_i x^i = 0$ $\leq_{\mathsf{Alg}}$ $?_x : b_6 \neq 0 : \sum_{i=0}^{6} b_i x^i = 0$

   The first problem is *not* solvable in radicals* (Abel, 1824)

   Hence, the second problem is *not* solvable in radicals

   Note that the reductions were also 'in radicals'

*'In radicals' means 'by using $+$, $-$, $\times$, $/$, and $\sqrt{\ }$ only'

Properties of $\leq_{\mathsf{Alg}}$

Relation $\leq_{\mathsf{Alg}}$ is transitive:

$$U_1 \leq_{\mathsf{Alg}} U_2 \leq_{\mathsf{Alg}} U_3 \;\Rightarrow\; U_1 \leq_{\mathsf{Alg}} U_3$$

Solvability propagates from *right to left* across a chain of the form

$$U_1 \;\leq_{\mathsf{Alg}}\; U_2 \;\leq_{\mathsf{Alg}}\; U_3$$

Unsolvability propagates from *left to right* across the chain

Algorithmic reducibility is also transitive:

> If you can reduce U_1 to U_2 and you can reduce U_2 to U_3,
> then you can reduce U_1 to U_3

Problems UNIV and HALT

More interesting problems:

UNIV (the **universal problem**)

Input: a program P and a natural number $i \in \mathbb{N}$

Output: $\texttt{YES}$, if P outputs $\texttt{YES}$ on input i

$\texttt{NO}$, if P outputs $\texttt{NO}$ or does not halt on input i

HALT (the **halting problem**)

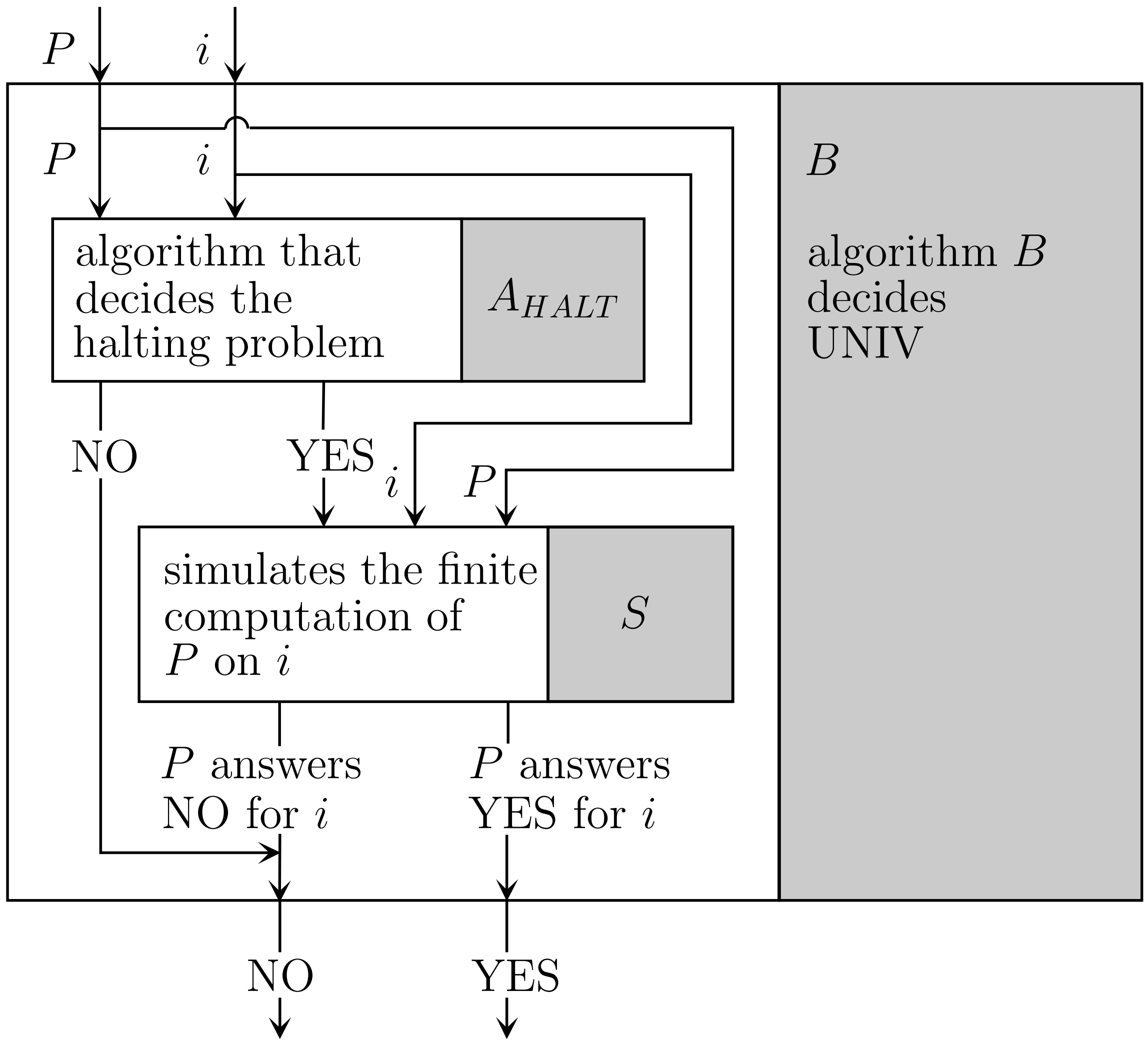
Input: a program P and a natural number $i \in \mathbb{N}$

Output: $\texttt{YES}$, if P halts on input i

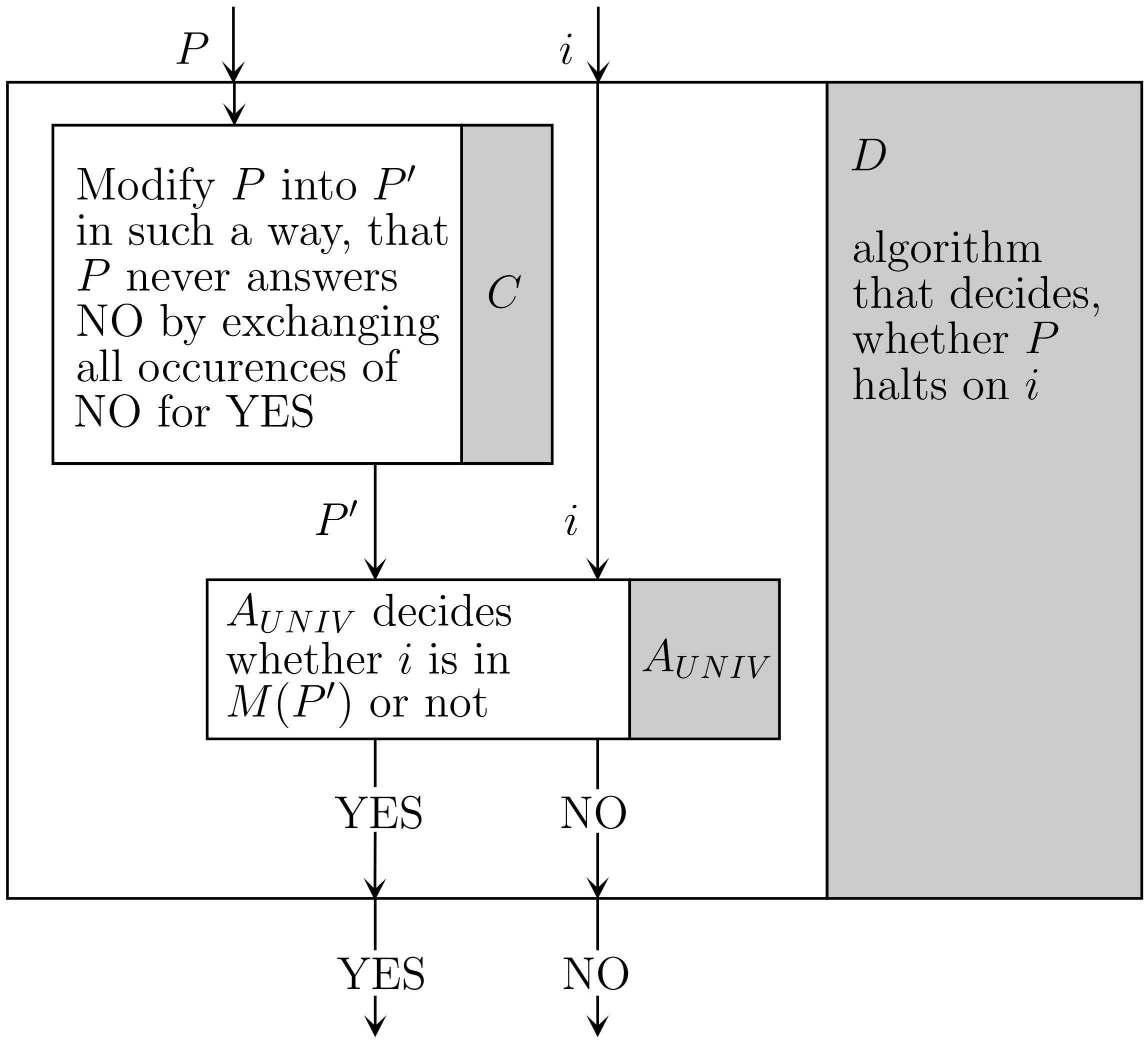
$\texttt{NO}$, if P does not halt on input i

N.B. Simulation of P will not work, because it need not terminate

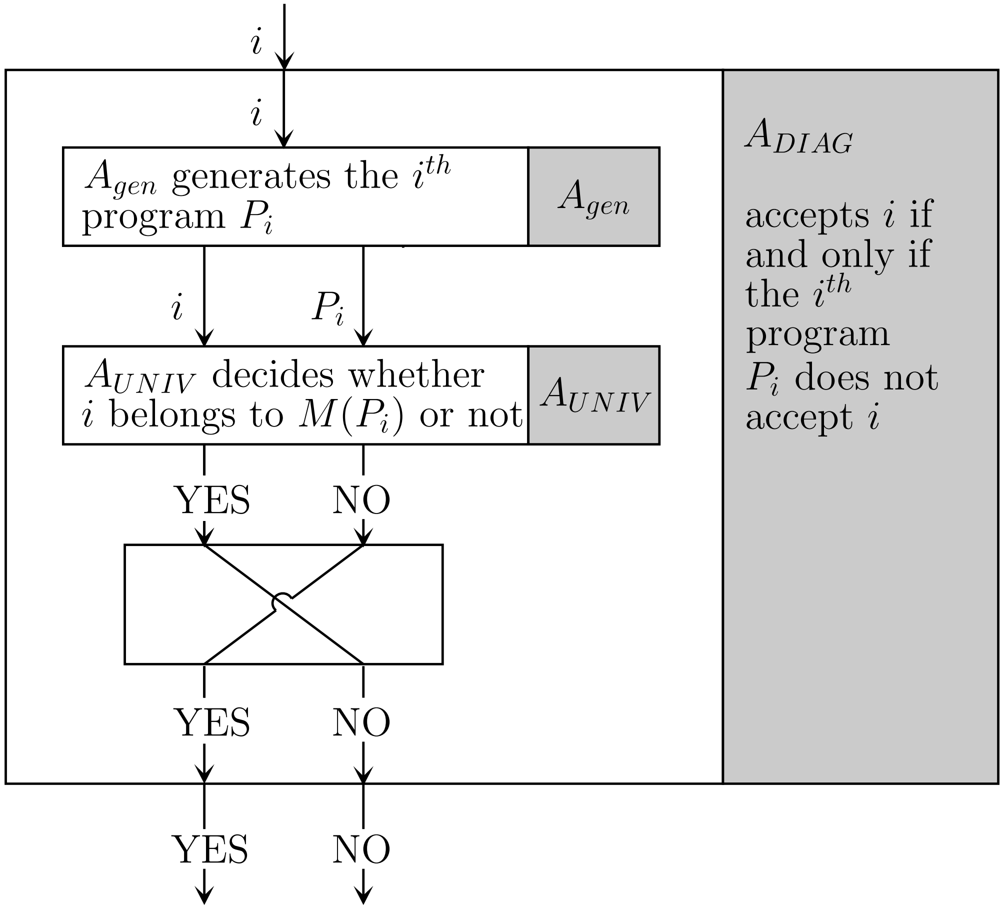
UNIV $\leq_{\mathsf{Alg}}$ HALT by Reduction

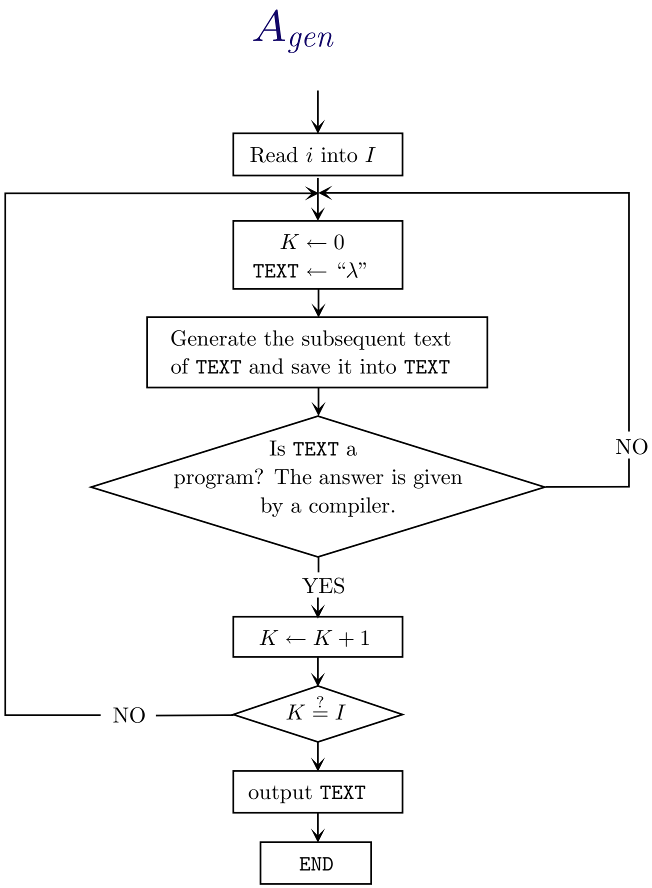


HALT $\leq_{\mathsf{Alg}}$ UNIV by Reduction



$(\mathbb{N}, DIAG) \leq_{\textsf{Alg}}$ UNIV by Reduction





Conclusion: UNIV and HALT are also not solvable by an algorithm

Summary

- *There exist tasks that cannot be automatically solved*.
- This claim is true independent of computer technologies.
- **Algorithmic reductions** help to compare problems for solvability.
- Among the algorithmically unsolvable problems, one can find:
  - Is a program correct?
  - Does a program avoid endless computations?
- **Syntactic** tasks, usually related to the correct *representation* of a program, are algorithmically solvable.
- **Semantic** questions, related to the *meaning* of a program, are not algorithmically solvable, unless trivial.