Undecidable Problems

Algorithmically Solvable Problems

Let us assume we have a problem P.

If there is an algorithm solving the problem P then we say that the problem P is **algorithmically solvable**.

If P is a decision problem and there is an algorithm solving the problem P then we say that the problem P is **decidable (by an algorithm)**.

If we want to show that a problem P is algorithmically solvable, it is sufficient to show some algorithm solving it (and possibly show that the algorithm really solves the problem P).

Algorithmically Unsolvable Problems

A problem that is not algorithmically solvable is **algorithmically unsolvable**.

A decision problem that is not decidable is **undecidable**.

Surprisingly, there are many (exactly defined) problems, for which it was proved that they are not algorithmically solvable.

Halting Problem

Let us consider some general programming language $\mathcal{L}$.

Futhermore, let us assume that programs in language $\mathcal{L}$ run on some idealized machine where a (potentially) unbounded amount of memory is available — i.e., the allocation of memory never fails.

Example: The following problem called the **Halting problem** is undecidable:

Halting problem

Input: A source code of a $\mathcal{L}$ program P, input data x.

Question: Does the computation of P on the input x halt after some finite number of steps?

Halting Problem

Let us assume that there is a program that can decide the Halting problem.

So we could construct a subroutine H, declared as

Bool H(String code, String input)

where $H(P, x)$ returns:

- true if the program P halts on the input x,
- false if the program P does not halt on the input x.

Remark: Let us say that subroutine $H(P, x)$ returns false if P is not a syntactically correct program.

Halting Problem

Using the subroutine H we can construct a program D that performs the following steps:

- It reads its input into a variable x of type String.
- It calls the subroutine $H(x, x)$.
- If subroutine H returns true, program D jumps into an infinite loop

```
loop: goto loop
```

In case that H returns false, program D halts.

What does the program D do if it gets its own code as an input?

Halting Problem

If D gets its own code as an input, it either halts or not.

- If D halts then $H(D, D)$ returns true and D jumps into the infinite loop. A contradiction!
- If D does not halt then $H(D, D)$ returns false and D halts. A contradiction!

In both case we obtain a contradiction and there is no other possibility. So the assumption that H solves the Halting problem must be wrong.

Semidecidable Problems

A problem is **semidecidable** if there is an algorithm such that:

- If it gets as an input an instance for which the answer is YES, then it halts after some finite number of steps and writes "YES" on the output.
- If it gets as an input an instance for which the answer is NO, then it either halts and writes "NO" on the input, or does not halt and runs forever.

It is obvious that for example HP (Halting Problem) is semidecidable.

Some problems are not even semidecidable.

Post's Theorem

The **complement** problem for a given decision problem P is a problem where inputs are the same as for the problem P and the question is negation of the question from the problem P.

Post's Theorem

If a problem P and its complement problem are semidecidable then the problem P is decidable.

Reduction between Problems

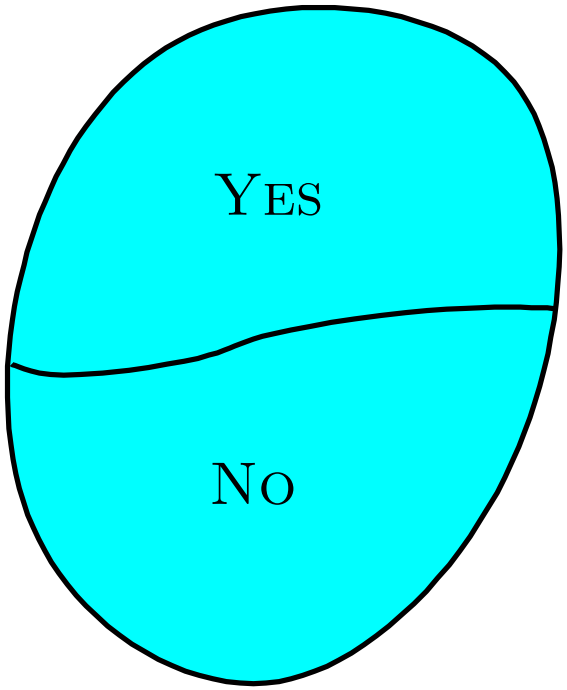
If we have already proved a (decision) problem to be undecidable, we can prove undecidability of other problems by reductions.

Problem P_1 can be **reduced** to problem P_2 if there is an algorithm Alg such that:

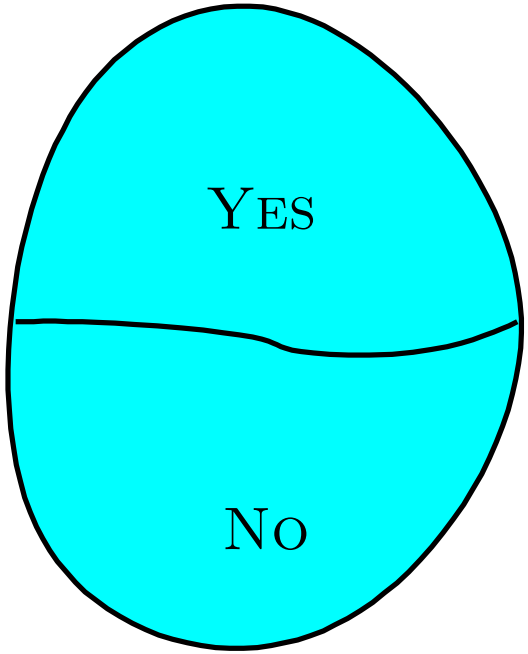
- It can get an arbitrary instance of problem P_1 as an input.
- For an instance of a problem P_1 obtained as an input (let us denote it as w) it produces an instance of a problem P_2 as an output.
- It holds i.e., the answer for the input w of problem P_1 is YES iff the answer for the input $Alg(w)$ of problem P_2 is YES.

Reductions between Problems

Inputs of problem P_1



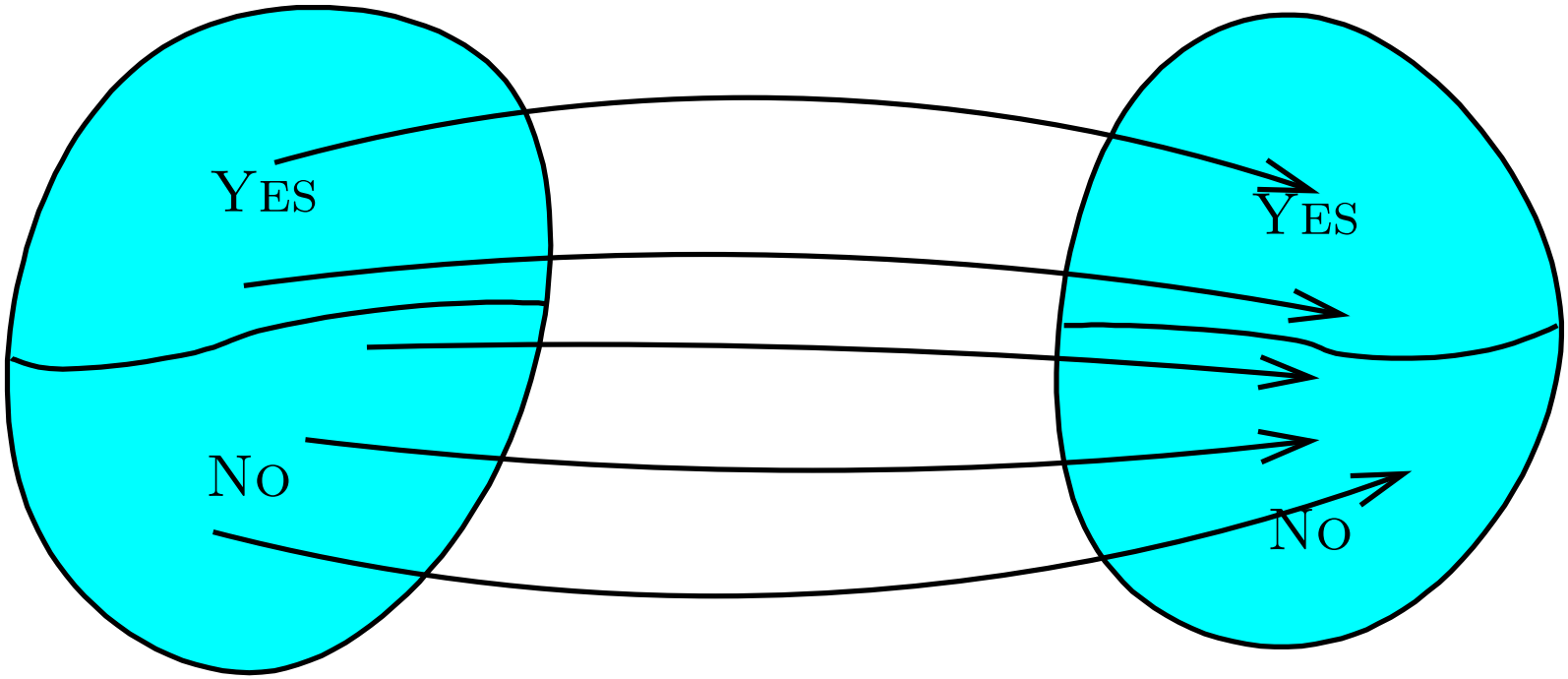
Inputs of problem P_2



Reductions between Problems

Inputs of problem P_1

Inputs of problem P_2



Alg

Reductions between Problems

Let us say there is some reduction Alg from problem P_1 to problem P_2.

If problem P_2 is decidable then problem P_1 is also decidable.

Solution of problem P_1 for an input x:

- Call Alg with x as an input, it returns a value $Alg(x)$.
- Call the algorithm solving problem P_2 with input $Alg(x)$.
- Write the returned value to the output as the result.

It is obvious that if P_1 is undecidable then P_2 cannot be decidable.

Other Undecidable Problems

By reductions from the Halting problem we can show undecidability of many other problems dealing with a behaviour of programs:

- Is for some input the output of a given program YES?
- Does a given program halt for an arbitrary input?
- Do two given programs produce the same outputs for the same inputs?
- ...

Halting Problem

For purposes of proofs, the following version of Halting problem is often used:

Halting problem

Input: A description of a Turing machine $\mathcal{M}$ and a word w.

Question: Does the computation of the machine $\mathcal{M}$ on the word w halt after some finite number of steps?

Other Undecidable Problems

We have already seen the following example of an undecidable problem:

Problem

Input: Context-free grammars $\mathcal{G}_1$ and $\mathcal{G}_2$.

Question: Is $\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$?

respectively

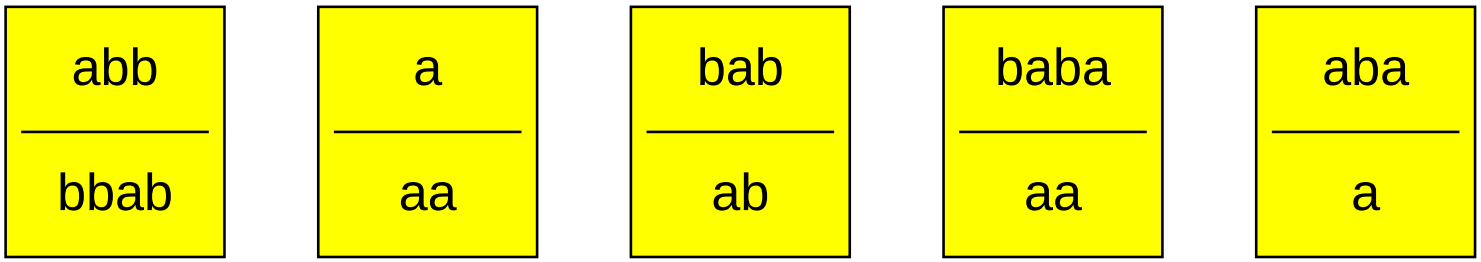
Problem

Input: A context-free grammar generating a language over an alphabet Σ.

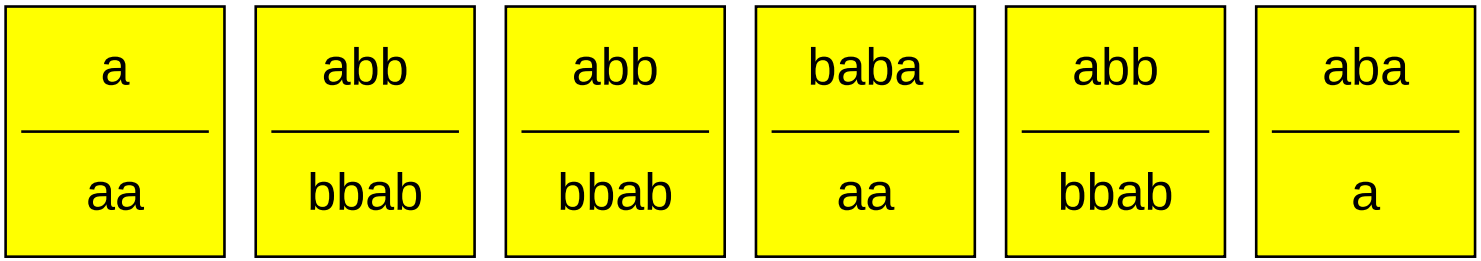
Question: Is $\mathcal{L}(\mathcal{G}) = \Sigma^*$?

Other Undecidable Problems

An input is a set of types of cards, such as:



The question is whether it is possible to construct from the given types of cards a non-empty finite sequence such that the concatenations of the words in the upper row and in the lower row are the same. Every type of a card can be used repeatedly.



In the upper and in the lower row we obtained the word aabbabbbabaabbaba.

Other Undecidable Problems

Undecidability of several other problems dealing with context-free grammars can be proved by reductions from the previous problem:

Problem

Input: Context-free grammars $\mathcal{G}_1$ and $\mathcal{G}_2$.

Question: Is $\mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2) = \emptyset$?

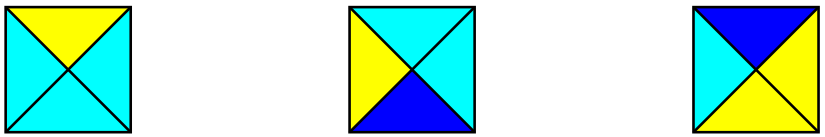
Problem

Input: A context-free grammar $\mathcal{G}$.

Question: Is $\mathcal{G}$ ambiguous?

Other Undecidable Problems

An input is a set of types of tiles, such as:

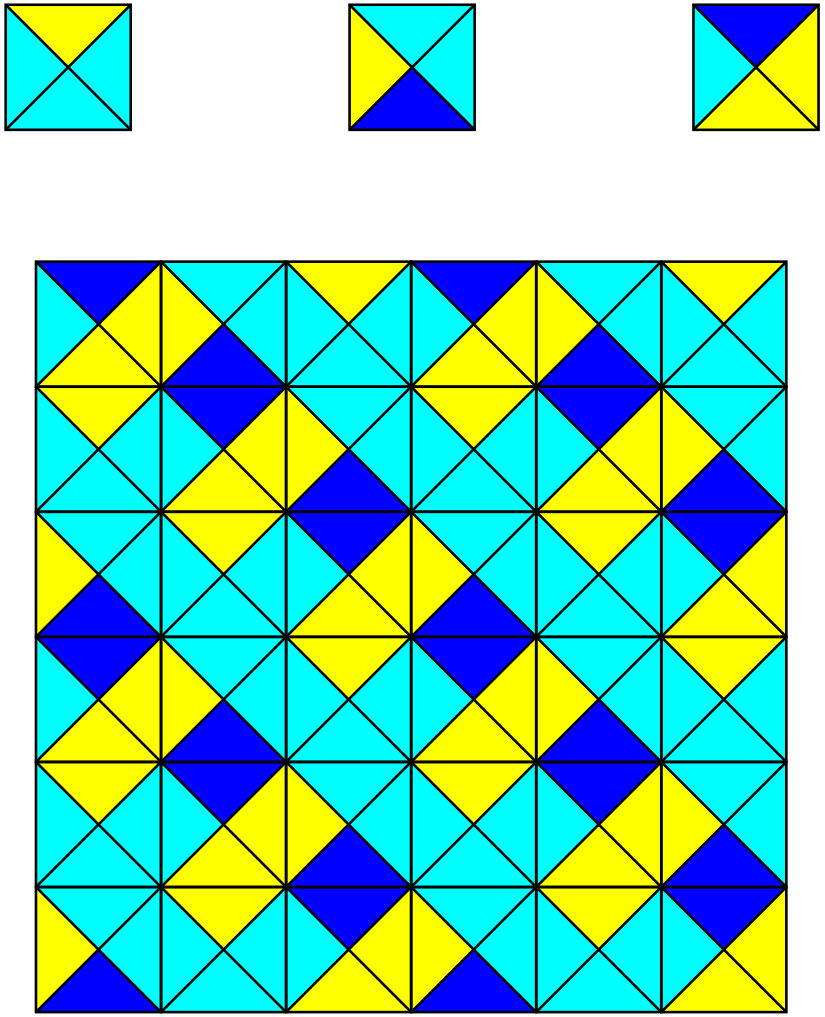


The question is whether it is possible to cover every finite area of an arbitrary size using the given types of tiles in such a way that the colors of neighboring tiles agree.

Remark: We can assume that we have an infinite number of tiles of all types.

The tiles cannot be rotated.

Other Undecidable Problems



Other Undecidable Problems

Problem

Input: A closed formula of the first order predicate logic where the only predicate symbols are $=$ and $<$, the only function symbols are $+$ and $*$, and the only constant symbols are 0 and 1.

Question: Is the given formula true in the domain of natural numbers (using the natural interpretation of all function and predicate symbols)?

An example of an input:

$$\forall x \exists y \forall z((x * y = z) \wedge (y + 1 = x))$$

Remark: There is a close connection with Gödel's incompleteness theorem.

Other Undecidable Problems

It is interesting that an analogous problem, where real numbers are considered instead of natural numbers, is decidable (but the algorithm for it and the proof of its correctness are quite nontrivial).

Also when we consider natural numbers or integers and the same formulas as in the previous case but with the restriction that it is not allowed to use the multiplication function symbol $*$, the problem is algorithmically decidable.

Other Undecidable Problems

If the function symbol $*$ can be used then even the very restricted case is undecidable:

Hilbert's tenth problem

Input: A polynomial $f(x_1, x_2, \ldots, x_n)$ constructed from variables $x_1, x_2, \ldots, x_n$ and integer constants.

Question: Are there some natural numbers $x_1, x_2, \ldots, x_n$ such that $f(x_1, x_2, \ldots, x_n) = 0$?

An example of an input: $5x^2y - 8yz + 3z^2 - 15$

I.e., the question is whether

$$\exists x \exists y \exists z (5 * x * x * y + (-8) * y * z + 3 * z * z + (-15) = 0)$$

holds in the domain of natural numbers.

Other Undecidable Problems

Also the following problem is algorithmically undecidable:

Problem

Input: A closed formula φ of the first-order predicate logic.

Question: Is $\models \varphi$?

Remark: Notation $\models \varphi$ denotes that formula φ is logically valid, i.e., it is true in all interpretations.

Rice's Theorem

Let P be an arbitrary prorperty of Turing machines.

The property P is:

- **nontrivial** — if there exists at least one machine that has the property P, and at least one machine that does not have the property P
- **input-output** — if in every pair of machines that halt on the same inputs and that give the same outputs for the same inputs, either both machines have the property P or both do not have it

Theorem

Every problem of the form

Input: A Turing machine $\mathcal{M}$.

Question: Does machine $\mathcal{M}$ have property P?

where P is a nontrivial input-output property, is undecidable.