

Formale Methoden der Informatik

Block 1: Computability and Complexity

Exercises (Examples)

Mantas Šimkus

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

SS 2013



Exercise 1

By providing a reduction from the **HALTING** problem, prove that the following problem is undecidable:

HELLO-WORLD

INSTANCE: A string I and a program Π that takes one string as input and outputs a string.

QUESTION: Does the program Π on input I return the string “Hello world!” as output?

Solution to Exercise 1

The reduction is defined as follows. Let (Π, I) be an arbitrary instance of **HALTING**. We build an instance (Π', I') of **HELLO-WORLD** by setting $I' = I$ and constructing Π' as follows:

```
String  $\Pi'$  (String  $S$ )  
Type val =  $\Pi(S)$ ; /* Type is the type of the output of  $\Pi^*$  /  
return "Hello world!";
```

In other words, for a instance $x = (\Pi, I)$, the instance $R(x)$ resulting from the reduction is (Π', I') . To prove the correctness of the reduction we have to show: (Π, I) is a positive instance of **HALTING** $\Leftrightarrow (\Pi', I')$ is a positive instance of **HELLO-WORLD**.

" \Rightarrow Assume (Π, I) is positive, i.e. Π terminates on I . Then by the construction of (Π', I') , we have that Π' returns "Hello world!" on I' . It follows that (Π', I') is a positive instance of **HELLO-WORLD**.

" \Leftarrow Assume (Π', I') is positive, i.e. Π' returns "Hello world!" on the input I' . Since Π' involves the call $\Pi(I')$ and since $I' = I$, we have that Π terminates on I , i.e. that (Π, I) is a positive instance of **HALTING**.

Solution to Exercise 1: Why does it work?

Why does the reduction R prove the undecidability of **HELLO-WORLD**?

Towards a contradiction, suppose **HELLO-WORLD** is decidable. Then there is an algorithm $\Pi_{hw}(\cdot)$ such that $\Pi_{hw}(x)$ returns *true* if x is a positive instance of **HELLO-WORLD**, and returns *false* otherwise.

Build a procedure Π_h , which takes instances of **HALTING**, as follows:

```
Bool  $\Pi_h(\text{String } \Pi, \text{String } I)$   
return  $\Pi_{hw}(R((\Pi, I)))$ ;
```

It is easy to see that Π_h is a decision procedure for **HALTING**:

- $\Pi_h(\Pi, I)$ returns *true* if Π terminates on I
- $\Pi_h(\Pi, I)$ returns *false* if Π does not terminate on I

We arrive at a contradiction: we know from the lecture that **HALTING** is undecidable.

Sanity test

Check that the problem instances that you are using in your solutions are compatible with the definition of a given problem:

- INSTANCE: A pair (Π, I) , where Π is a program that takes one string as input and returns a string, and I is a string.

In a proof:

- $(\Pi, I), (\Pi', I'), (\Pi, \text{"hello"})$ are **O.K.**
- $(\Pi, I, I'), (\Pi, I, k), \Pi$ are **not O.K.**

- INSTANCE: A program Π that takes one string as input and returns a string.

In a proof:

- Π, Π', Π_1, Π_2 are **O.K.**
- $(\Pi, I), (\Pi', I'), (\Pi, I, I'), (\Pi, I, k)$ are **not O.K.**

Exercise 2

Prove that the **HELLO-WORLD** problem is semi-decidable.

Solution to Exercise 2

Recall that a decision problem P is called **semi-decidable** if we can build a program Π such that:

- Π takes as input instances I of P ;
- if I is a “yes” instance, then Π returns *true*;
- if I is a “no” instance, then Π returns *false* or does not terminate;

To show that **HELLO-WORLD** is semi-decidable we can provide a procedure Π' that fulfills the above conditions:

- Π' that takes as input an instance of **HELLO-WORLD**, i.e. a program Π and an input I ,
- Π' simulates the run of Π on I .
- If the simulation reaches output “Hello world!”, then Π' returns *true*.
- If the simulation ends without output “Hello world!”, then Π' returns *false*. (Observe: termination guaranteed on positive instances only).

Solution to Exercise 2 (continued)

We argue that such an interpreter Π' is a semi-decision procedure for **HELLO-WORLD**. We distinguish the following cases:

- Case 1. Suppose that (Π, I) is a positive instance, i.e., Π on the string I as input returns “Hello world”. Then the simulation in Π' will encounter the call to print “Hello world” and return *true* by the construction of Π' .
- Case 2.1. Suppose that (Π, I) is a negative instance and that Π halts on input I . Then Π halts with output different from “Hello world”. Hence, the simulation in Π' will detect that the output is different from “Hello world”. Thus, Π' returns *false* by the construction of Π' .
- Case 2.2. Suppose that (Π, I) is a negative instance and that Π does not halt on input I . Then the simulation of this computation of Π on I by the interpreter Π' will not terminate either. Hence, Π' will run forever on the negative instance (Π, I) , which is a correct behavior for a semi-decision procedure.

Exercise 3

Give a formal proof that **3-COLORABILITY** is in NP, i.e. define the required certificate relation and discuss its polynomial balance and polynomial decidability.

Solution to Exercise 3

Define the relation

$$R = \{(G, \mu) \mid \mu \text{ is a 3-coloring for the graph } G\}.$$

- We argue that R is a certificate relation for **3-COLORABILITY**. Indeed, a graph G is a positive instance of **3-COLORABILITY** \Leftrightarrow there exists a proper 3-coloring μ of $G \Leftrightarrow (G, \mu) \in R$.
- R is polynomially balanced because a color assignment μ can be represented in space that is linear in the size of G .
- R is polynomially decidable. Indeed, given a graph G and an assignment μ of colors, checking if μ is a k -coloring of G is feasible in polynomial time in the size of G and μ .

Exercise 4

Formally prove that **VERTEX COVER** is NP-hard. For this you may use the well-known fact that **INDEPENDENT SET** is NP-complete.

Solution to Exercise 4

We reduce **INDEPENDENT SET** to **VERTEX COVER**.

Let (G, k) be an arbitrary instance of **INDEPENDENT SET**, i.e., G is an undirected graph and k is an integer. Let $G = (V, E)$. We construct the instance $(G, |V| - k)$ of **VERTEX COVER**. Clearly, this reduction is feasible in polynomial time.

It remains to prove the correctness. Indeed, by the proposition on page 18 in the lecture “4. NP-Completeness”, we have that (G, k) is a positive instance of **INDEPENDENT SET** iff $(G, |V| - k)$ is a positive instance of **VERTEX COVER**.

Exercise 5

We consider a polynomial time reduction from the **3-COLORABILITY** problem for directed graphs to **SAT**. Let $G = (V, E)$ be a directed graph (i.e. an instance of **3-COLORABILITY**) with $V = \{a_1, \dots, a_n\}$. We construct a formula φ_G (i.e. an instance of **SAT**) as follows.

We use propositional variables c_i^m , where $1 \leq i \leq n$ and $m \in \{0, 1, 2\}$, to indicate that the vertex a_i of G is colored with color m . Then φ_G is defined as $\varphi_G = \varphi_1 \wedge \varphi_2 \wedge \varphi_3$, where

$$\varphi_1 = \bigwedge_{1 \leq i \leq n} (c_i^0 \vee c_i^1 \vee c_i^2),$$

$$\varphi_2 = \bigwedge_{1 \leq i \leq n} (\neg(c_i^0 \wedge c_i^1) \wedge \neg(c_i^0 \wedge c_i^2) \wedge \neg(c_i^1 \wedge c_i^2)),$$

$$\varphi_3 = \bigwedge_{(a_i, a_j) \in E} (\neg(c_i^0 \wedge c_j^0) \wedge \neg(c_i^1 \wedge c_j^1) \wedge \neg(c_i^2 \wedge c_j^2)).$$

Prove the “ \Rightarrow ” direction of the correctness of the reduction, i.e. prove the following statement: if G is 3-colorable, then φ_G is satisfiable.

Solution to Exercise 5

To prove “ \Rightarrow ”, assume $G = (V, E)$ is 3-colorable. Let $\mu : V \rightarrow \{0, 1, 2\}$ be an assignment of colors that shows 3-colorability of G .

To show that φ_G is satisfiable, we have to define a truth assignment T that makes φ_G evaluate to *true*.

We take the truth assignment T as follows:

$$T(c_i^m) = \begin{cases} \text{true} & \text{if } \mu(a_i) = m, \\ \text{false} & \text{if } \mu(a_i) \neq m. \end{cases}$$

To show that φ_G evaluates to *true* under T , it suffices to make sure that $\varphi_1, \varphi_2, \varphi_3$ evaluate to *true* under T .

- μ assigns at least one color to each vertex of G . Then due to the construction of T , the formula φ_1 evaluates to *true* under T .
- μ assigns no more than one color to each vertex of G . Then due to the construction of T , also φ_2 evaluates to *true* under T .
- We deal with φ_3 next...

Solution to Exercise 5

- We know μ is a proper 3-coloring of G , i.e. for each $(a_i, a_j) \in E$, we have $(*) \mu(a_i) \neq \mu(a_j)$. Towards a contradiction, suppose φ_3 evaluates to *false* under T . Then there must exist $(a_i, a_j) \in E$ and some $m \in \{0, 1, 2\}$ such that the subformula $\neg(c_i^m \wedge c_j^m)$ evaluates to *false* under T . Hence, $(c_i^m \wedge c_j^m)$ evaluates to *true* under T . It follows that $T(c_i^m) = \text{true}$ and $T(c_j^m) = \text{true}$. Then due to the construction of T , we have $\mu(a_i) = m$ and $\mu(a_j) = m$. This contradicts $(*)$, and thus φ_3 evaluates to *true* under T .

Exercise 6

Consider the following problem:

MOST-FREQUENT

INSTANCE: A list L of letters from $\{a, \dots, z\}$, and a letter s from $\{a, \dots, z\}$.

QUESTION: Is s the most frequent element in L ?

Argue that **MOST-FREQUENT** can be solved using only logarithmic space.

Solution to Exercise 6

We only need one pointer to an element in the list and two counters. Both require only logarithmic space. The decision procedure for **MOST-FREQUENT** works in two phases:

- First, we traverse the input (with the pointer) and count (with the first counter) the number of occurrences of the letter s . At the end of this step, the number of occurrences of the letter s is stored in the first counter.
- In a loop over all letters from $\{a, \dots, z\}$, we do the following: If the current letter is s , then we do nothing. For any other letter, we traverse the input (with the pointer) and count (with the second counter) the number of occurrences of the current letter. If the number of occurrences of this letter is greater than the number stored in the first counter, we halt with “no”. Otherwise, we proceed to the next letter.

If the above loop ends without encountering a letter that occurs more often than s , then the procedure halts with “yes”.

Solution to Exercise 6 (continued)

Remark. Since the alphabet $\{a, \dots, z\}$ is fixed, the above loop can be “hard coded” in the program, i.e., 26 copies of the code to be carried out in the loop. If the alphabet were part of the input, we would simply use a second pointer to the current letter to be processed in the loop.

Exercise 7

Let $L = \{w \in \{0, 1\}^* \mid |w| \text{ is even}\}$, i.e. L is the set of all strings w such that (a) w is built using symbols 0 and 1, and (b) w is of even length.

Define a Turing machine M that decides L , i.e. define a tuple $M = (K, \Sigma, \delta, s)$ such that, for all $w \in \{0, 1\}^*$, we have:

- if $w \in L$, then $M(w) = \text{"yes"}$;
- if $w \notin L$, then $M(w) = \text{"no"}$.

Additionally, provide a high-level description of M .

Solution to Exercise 7

$M = (K, \Sigma, \delta, s)$ with $K = \{s, q\}$, $\Sigma = \{0, 1, \sqcup, \triangleright\}$ and a transition function δ defined as follows:

| $p \in K$ | $\sigma \in \Sigma$ | $\delta(p, \sigma)$ |
|-----------|---------------------|------------------------------------|
| s | \triangleright | $(s, \triangleright, \rightarrow)$ |
| s | 0 | $(q, 0, \rightarrow)$ |
| s | 1 | $(q, 1, \rightarrow)$ |
| s | \sqcup | $(\text{"yes"}, \sqcup, -)$ |
| q | 0 | $(s, 0, \rightarrow)$ |
| q | 1 | $(s, 1, \rightarrow)$ |
| q | \sqcup | $(\text{"no"}, \sqcup, -)$ |

Solution to Exercise 7 (continued)

High-level description of M :

- In state s : If the symbol is \triangleright , then move the head to the right without changing the state. If the symbol is 0 or 1, then move the head to the right and change the state to q . If the symbol is \sqcup , then stop in the state “yes”.
- In state q : If the symbol is 0 or 1, then move the head to the right and change the state to s . If the symbol is \sqcup , then stop in the state “no”.