Formale Methoden der Informatik

Block 1: Computability and Complexity

Exercises (Sample Solutions)

Mantas Šimkus

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

WS 2013

Exercise 1

Consider the following problem:

VARIABLE-VALUE (VV)

INSTANCE: A tuple (Π, I, x), where

a) Π is a program that takes as input a string,

b) I is a string,

c) x is a global variable in Π of type *Boolean*.

QUESTION: In the run of Π on I, does x ever get assigned the value *true*?

By providing a reduction from **HALTING** to **VARIABLE-VALUE**, prove that **VARIABLE-VALUE** is undecidable.

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', x) of **VV** as follows. We let $I' = I$, and build Π' as follows:

```
String Π'(String S)
x := false // x is a global variable not used in Π
Π(S); // Π is hardcoded
x := true
return 0;
```

In other words, for an instance $Y = (\Pi, I)$, the instance $R(Y)$ resulting from the reduction is (Π', I', x). To prove the correctness of the reduction we have to show:

(Π, I) is a positive instance of **HALTING** $\Leftrightarrow$ (Π', I', x) is a positive instance of **VV**.

Solution to Exercise 1 (continued)

"$\Rightarrow$" Assume (Π, I) is a positive instance of **HALTING**, i.e. Π terminates on I. Then the call $\Pi(S)$ in program Π' terminates on input I and also on input I' since $I' = I$ by the definition of the reduction. Thus the statement "$x := \mathit{true}$" is reached with the input I' to Π'. Hence, (Π', I', x) is a positive instance of **VV**.

"$\Leftarrow$" Assume (Π', I', x) is a positive instance of **VV**. That is, at some point in the run of Π' on I' the variable x gets assigned the value *true*. In Π' the only way to assign *true* to x is by executing "$x := \mathit{true}$" which comes *after* the call $\Pi(I')$. Thus, it must be the case that Π terminates on I' and thus also on input I since $I' = I$. Hence, (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 **VARIABLE-VALUE**?

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

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

```
Bool Π_h(String Π,String I)
return Π_vv(R((Π, 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:

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

Exercise 2

By providing a semi-decision procedure, prove that **VARIABLE-VALUE** is semi-decidable.

Solution to Exercise 2

We can write an interpreter Π_{int} that takes as input (Π, I, x), i.e. an instance of **VARIABLE-VALUE**, and simulates the run of Π on I:

- If the simulation reaches the point where the variable x gets value *true*, then Π_{int} returns *true*.
- If the simulation ends without ever assigning *true* to x, then Π_{int} returns *false*.

Solution to Exercise 2 (continued)

It can be seen as follows that such an interpreter Π_{int} is a semi-decision procedure for **VARIABLE-VALUE**. We distinguish the following cases:

- Case 1. Suppose that (Π, I, x) is a positive instance, i.e., in the run of Π on I, the variable gets assigned *true*. Then the simulation in Π_{int} will encounter this and return *true* by the construction of Π_{int}.
- Case 2.1. Suppose that (Π, I, x) is a negative instance and that Π halts on input I. Then Π halts without ever assigning *true* to x. Hence, the simulation in Π_{int} will terminate and conclude that x was never assigned *true*. Thus, Π_{int} returns *false* by the construction of Π_{int}.
- Case 2.2. Suppose that (Π, I, x) 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_{int} will not terminate either. Hence, Π_{int} will run forever on the negative instance (Π, I, x), which is a correct behavior for a semi-decision procedure.

Exercise 3

Consider the following problem:

> **EXIST-PAIR**
>
> INSTANCE: A program Π that takes as input a pair of natural numbers, and returns *true* or *false*. It is guaranteed that Π terminates on any input.
>
> QUESTION: Do there exist n_1, n_2 such that $\Pi(n_1, n_2) = \mathit{true}$?

By providing a semi-decision procedure, prove that **EXIST-PAIR** is semi-decidable.

Solution to Exercise 3

We define a procedure Π_{ep}, which enumerates all pairs n_1, n_2 and checks whether $\Pi(n_1, n_2) = \mathit{true}$ for the input program Π. If such a pair exists, the procedure outputs *true*.

```
String Π_ep (String Π)
n_1 := 0
while (true) do {
    for n_2 from 0 to n_1 do
        if Π(n_1, n_2) = true then return true
n_1 := n_1 + 1
}
```

Solution to Exercise 3 (continued)

The described procedure is a semi-decision procedure for **EXIST-PAIR**:

Indeed, assume a pair n_1, n_2 such that $\Pi(n_1, n_2) = \mathit{true}$. Since Π is guaranteed to terminate on any input, and since the enumeration covers all pairs of integers, our procedure will terminate with output *true*.

On the other hard, on negative instances of **EXIST-PAIR** the procedure will not terminate, which is an acceptable behavior of a semi-decision procedure.

Exercise 4

Prove that **EXIST-PAIR** is undecidable.

Hint: For your proof you may assume the availability of an interpreter for instances of **HALTING**. In particular, you have available a procedure Π_{int} that does the following:

1. Π_{int} takes as input a program Π, a string I, and a natural number n.
2. Π_{int} emulates the first n steps of the run of Π on I. If Π terminates on I within n steps, then Π_{int} returns *true*. Otherwise, Π_{int} returns *false*.

Solution to Exercise 4

We provide a reduction from **HALTING**, which is known to be undecidable. Let (Π, I) be an arbitrary instance of **HALTING**. We build an instance Π' of **EXIST-PAIR** by constructing Π' as follows:

String Π' (**Int** n_1,**Int** n_2)
return $\Pi_{int}(\Pi, I, n_1)$ // Π and I are 'hard-coded' in Π'

To prove the correctness of the reduction we have to show:

(Π, I) is a positive instance of **HALTING** $\Leftrightarrow$ Π' is a positive instance of **EXIST-PAIR**.

Solution to Exercise 4 (continued)

"$\Rightarrow$" Assume (Π, I) is a positive instance of **HALTING**, i.e. Π terminates on I. In particular, there is a number n, such that Π terminates on I within n steps. Hence, for such n, $\Pi_{int}(\Pi, I, n) = \mathit{true}$ by definition of Π_{int} and $\Pi'(n, 0) = \mathit{true}$ by definition of Π'. Clearly, here 0 can be replaced by any number. It follows that there exist n_1, n_2 (namely, $n_1 = n$ and $n_2 = 0$) such that $\Pi'(n_1, n_2) = \mathit{true}$. Thus Π' is a positive instance of **EXIST-PAIR**.

"$\Leftarrow$" Assume Π' is a positive instance of **EXIST-PAIR**, i.e. $\Pi'(n_1, n_2) = \mathit{true}$ for some n_1, n_2. By definition of Π', it must be the case that $\Pi_{int}(\Pi, I, n_1) = \mathit{true}$. Due to the definition of Π_{int}, Π terminates on I within n_1 steps. Thus (Π, I) is a positive instance of **HALTING**.

Reduction from **REACHABILITY** to **2-UNSAT**

Recall that the complement of **2-SAT** is as follows:

> **2-UNSAT**
>
> INSTANCE: Boolean formula φ in 2-CNF.
>
> QUESTION: Is φ unsatisfiable?

We consider a polynomial-time reduction from **REACHABILITY** to **2-UNSAT**. Let an arbitrary instance of **REACHABILITY** be given by the undirected graph $G = (V, E)$ and two vertices $a, b \in V$. In the reduction we use a propositional variable R_v for each vertex $v \in V$. The resulting instance $\varphi_{G,a,b}$ of **2-UNSAT** is as follows:

$$\varphi_{G,a,b} = \underbrace{R_a}_{(1)} \wedge \underbrace{\neg R_b}_{(2)} \wedge \underbrace{\bigwedge_{[c,d] \in E} (\neg R_c \vee R_d)}_{(3)} .$$

Exercise 5

Prove formally the "$\Rightarrow$" direction of the correctness of the reduction, i.e. prove the following statement: if b is reachable from a in G, then $\varphi_{G,a,b}$ is unsatisfiable.

Solution to Exercise 5

Suppose b is reachable from a in G. Then, by the definition of reachability, there is a sequence $v_1, \ldots, v_n$ of vertices from G with:

(a) $v_1 = a$, $v_n = b$, and

(b) $[v_i, v_{i+1}] \in E$ for all $1 \leq i < n$.

We apply induction to show that for all $1 \leq i \leq n$, the proposition R_{v_i} must be set to *true* in any model of $\varphi_{G,a,b}$ (in other words, $\varphi_{G,a,b} \models R_{v_i}$).

Base case: Suppose $i = 1$. By (1) in $\varphi_{G,a,b}$, trivially $\varphi_{G,a,b} \models R_{v_i}$ holds.

Solution to Exercise 5 (continued)

Induction step. Assume $i = n$ and $\varphi_{G,a,b} \models R_{v_i}$ holds for all $i \leq n$. We have to show that $\varphi_{G,a,b} \models R_{v_{i+1}}$ holds. Since $[v_i, v_{i+1}] \in E$, we have the clause $\neg R_{v_i} \vee R_{v_{i+1}}$ in (3). Assume a model M of $\varphi_{G,a,b}$. By the induction hypothesis, R_{v_i} is set *true* in M. Since, M must make $\neg R_{v_i} \vee R_{v_{i+1}}$ *true*, it must make $R_{v_{i+1}}$ *true*. That is, $\varphi_{G,a,b} \models R_{v_{i+1}}$.

To conclude the proof, assume that $\varphi_{G,a,b}$ is satisfiable. Then it has some model M. By the above argument, R_{v_n} is set to *true* in M. However, $\varphi_{G,a,b}$ has the clause $\neg R_b$ and $v_n = b$. Thus, in order for M to be a model of $\varphi_{G,a,b}$, R_n must be set to *false* in M. Contradiction.

Exercise 6

Prove the "$\Leftarrow$" direction of the correctness of the reduction, i.e. prove the following statement: if $\varphi_{G,a,b}$ is unsatisfiable, then b is reachable from a in G.

Hint: the above is equivalent to proving the following statement: if b is not reachable from a in G, then $\varphi_{G,a,b}$ is satisfiable.

Solution to Exercise 6

We show: if b is not reachable from a in G, then $\varphi_{G,a,b}$ is satisfiable.

Assume b is not reachable from a in G. Let V_r be the set of all vertices that are reachable from a in G. Note that $a \in V_r$.

We define a truth assignment for $\varphi_{G,a,b}$. For all propositions R_v of $\varphi_{G,a,b}$, we set $T(R_v) = \mathit{true}$ iff $v \in V_r$.

It remains to see that $\varphi_{G,a,b}$ evaluates to *true* under the assignment T:

1. Clearly, (1) and (2) evaluate to *true* because $a \in V_r$ and $b \notin V_r$, respectively. The latter is because, by the assumption, b is not reachable from a.
2. To show that (3) evaluates to *true*, assume any $[c, d] \in E$. We must show that the clause $(\neg R_c \vee R_d)$ evaluates to *true* in T. There are two possibilities:
   - $c \notin V_r$. Then $T(R_c) = \mathit{false}$ by the definition of T and the clause trivially evaluates to *true*.
   - $c \in V_r$. Since $[c, d] \in E$ and c is reachable from a, d is also reachable from a, and thus $d \in V_r$. Thus $T(R_d) = \mathit{true}$ and the clause evaluates to *true*.

Problem: a bank robber on the loose!

Suppose we are in a country that has cities connected by roads. We know that the robber is in city a and will try to run to city b.

There are n police cars available, and they have to be used to block the roads so that the robber cannot reach the city b. Your problem is to assign the available police cars to roads so that *all* possible paths from a to b are blocked.

The above problem can be formalized as follows:

EDGE-REMOVAL

INSTANCE: A tuple (G, a, b, n), where

- $G = (V, E)$ is an undirected graph,
- $a, b \in V$, and
- n is an integer.

QUESTION: Is it possible to remove n edges from G so that the resulting graph does not have a path from a to b?

Exercise 7

Give a proof that **EDGE-REMOVAL** is in NP, i.e. define a certificate relation and briefly discuss that it is polynomially balanced and polynomial-time decidable.

Solution to Exercise 7

We define a binary relation R such that:

$(e_1, e_2) \in R$ iff $e_1 = (G, a, b, n)$ is an instance of **EDGE-REMOVAL** and e_2 is a set of at most n edges from G whose removal from G makes b unreachable from a.

Clearly, R is a certificate relation for **EDGE-REMOVAL**, since the following equivalences hold: $e_1 = (G, a, b, n)$ is a positive instance of **EDGE-REMOVAL** $\Leftrightarrow$ there exists a set e_2 of at most n edges whose removal from G makes b unreachable from $a \Leftrightarrow (e_1, e_2) \in R$.

R is polynomially balanced because a subset of edges of a graph can be represented in space that is linear in the size of the considered graph.

Finally R is decidable in polynomial time because, given a graph G, a pair of vertices a, b and a set of edges D to be removed from G, one can check in polynomial time if the deletion makes b unreachable from a.

Exercise 8

Provide a reduction from 3-**COLORABILITY** to 4-**COLORABILITY**, and argue that your reduction is correct.

Hint: for the reduction it suffices to introduce one additional node to the input graph.

It is known that 3-**COLORABILITY** is NP-hard. Thus, your reduction shows that 4-**COLORABILITY** is NP-hard as well.

Solution to Exercise 8

Assume an arbitrary instance $G = (V, E)$ of 3-**COLORABILITY**. We create a new graph $G' = (V', E')$, where

- $V' = V \cup \{v\}$ where v is a fresh vertex, and
- $E' = E \cup \{[v, v'] \mid v' \in V\}$.

Intuitively, we add to G a new vertex v and connect it to each original vertex of G.

It is easy to see that G is a positive instance of 3-**COLORABILITY** iff G' is a positive instance of 4-**COLORABILITY**.

($\Rightarrow$) Suppose G can be properly colored with 3 colors (say, *red*,*blue* and *green*). Then the existing coloring can be extended to G' by coloring v with the additionally available color (say, *yellow*).

($\Leftarrow$) Suppose G' can be properly colored with 4 colors. Since e has an edge to every original node of G, all the original nodes must be properly colored with 3 colors only.

Exercise 9

Argue that the following problem is solvable in logarithmic space:

PREFIX

INSTANCE: A list $L = \langle s_1, \ldots, s_n \rangle$, where each s_i is either 0 or 1.

QUESTION: Does there exist $1 \leq j \leq n$ such that $\langle s_1, \ldots, s_j \rangle$ has the same number of 0s and 1s.

Solution to Exercise 9

We present next a procedure that decides **PREFIX**:

```
Boolean solvePREFIX (List L)

Integer i                 //pointer in the list
Integer c_0, c_1;         //counters

c_0 = 0; c_1 = 0
for i from 1 to length(L) do {
        if L(i) = 0 then  c_0 := c_0 + 1
        if L(i) = 1 then  c_1 := c_1 + 1
        if c_0 = c_1 then return true
}

return false
```

Solution to Exercise 9

The procedure uses 3 variables i, c_1, c_2. For a list $L = (s_1, \ldots, s_n)$, the variable i needs at most $\lceil \log n \rceil$ bits to be represented in binary. Observe that for a very large L, the size of i may well exceed any constant bound (in particular, 32- or 64-bit integers are not sufficient in general). The variables c_1, c_2 also only need logarithmic space. Indeed, the maximal value that we can encounter during summing is bounded by n. Thus to represent c_1, c_2 we need $2 \times \lceil log(n) \rceil$ bits.

Exercise 10

Let $L = \{w \in \{0,1\}^* \mid w \text{ has the form } 0^*1^*\}$, i.e. L is the set of all strings w that can be split into 2 words $w = w_1 w_2$ such that (a) w_1 is a sequence of 0s, and (b) w_2 is a sequence of 1s. 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 $M(w) = \text{"}yes\text{"}$;
- if $w \notin L$, then $M(w) = \text{"}no\text{"}$.

Additionally, provide a high-level description of M.

Solution to Exercise 10

$M = (K, \Sigma, \delta, s_0)$ with $K = \{s_0, s_1\}$, $\Sigma = \{0, 1, \sqcup, \triangleright\}$ and a transition function δ defined as follows:

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s_0	$\triangleright$	$(s_0, \triangleright, \rightarrow)$
s_0	0	$(s_0, 0, \rightarrow)$
s_0	1	$(s_1, 1, \rightarrow)$
s_0	$\sqcup$	$(\text{“yes”}, \sqcup, -)$
s_1	1	$(s_1, 1, \rightarrow)$
s_1	0	$(\text{“no”}, 0, -)$
s_1	$\sqcup$	$(\text{“yes”}, \sqcup, -)$

(note: $\delta(s_1, \triangleright)$ can be arbitrary)

Solution to Exercise 10 (continued)

High-level description of M: As long as the machine is in state s_0, it scans the word from left to right. If it reaches the end of the word, it outputs "yes" (it has scanned a word of the form 0^*). If it reads the symbol 1, it switches the state from s_0 to s_1. In state s_1 the machine ensures that the remainder of the word has no occurrences of 0s.