Formale Methoden der Informatik

Block 1: Computability and Complexity

Exercises (Sample Solutions)

Mantas Šimkus

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

WS 2012

Exercise 1

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

MODIFY-INPUT

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

QUESTION: Does the program Π on input I return a string I' such that $I' \neq I$, i.e. $\Pi(I) \neq I$?

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 **MODIFY-INPUT** by setting $I' = I$ and constructing Π' as follows:

```
String Π′ (String S)
eval(Π(S)); // Π is hardcoded in Π′
return S + "a";
```

In other words, for an 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** $\Leftrightarrow$ (Π', I') is a positive instance of **MODIFY-INPUT**.

Solution to Exercise 1 (continued)

"$\Rightarrow$" Assume (Π, I) is a positive instance of **HALTING**, i.e. Π terminates on I. Then the call $\Pi(I')$ in program Π' terminates since $I' = I$ by the problem reduction. Hence, the "return" statement in Π' is executed on input I' and, therefore, Π' returns $I' +$ "a" on input I'. Since $I' +$ "a" $\neq I'$, it follows that (Π', I') is a positive instance of **MODIFY-INPUT**.

"$\Leftarrow$" Assume (Π', I') is a positive instance of **MODIFY-INPUT**, i.e. Π' terminates on I' with output $\Pi'(I') \neq I'$. Since Π' involves the call $\Pi(I')$ and since $I' = I$, also Π terminates on I, i.e. (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 **MODIFY-INPUT**?

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

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

```
Bool Πh(String Π, String I)
return Πmi(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'), $(\Pi, "hello")$ 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

Prove that **MODIFY-INPUT** is semi-decidable. To this end, provide a semi-decision procedure and justify your solution.

Solution to Exercise 2

Write an interpreter Π_{int} that takes as input Π and I, i.e. an instance of **MODIFY-INPUT**, and simulates the run of Π on I:

- If the simulation reaches the point where a string I' with $I' \neq I$ is output, then Π_{int} returns *true*.
- If the simulation ends with an output I' such that $I' = I$, 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 **MODIFY-INPUT**. We distinguish the following cases:

- Case 1. Suppose that (Π, I) is a positive instance, i.e., Π outputs I' on input I with $I \neq I'$. Then the simulation in Π_{int} will encounter the output $I' \neq I$ and return *true* by the construction of Π_{int}.
- Case 2.1. Suppose that (Π, I) is a negative instance and that Π halts on input I. Then Π halts with an output $I' = I$. Hence, the simulation in Π_{int} will detect that the output I' is equal to I. Thus, Π_{int} returns *false* by the construction of Π_{int}.
- 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_{int} will not terminate either. Hence, Π_{int} will run forever on the negative instance (Π, I), which is a correct behaviour for a semi-decision procedure.

Exercise 3

Prove that the following problem is semi-decidable:

KEEP-SOME

INSTANCE: A program Π that takes one string as input and returns a string. It is guaranteed that Π terminates on any input.

QUESTION: Does there exist a string I such that $\Pi(I) = I$? That is, does there exist a string I such that Π does not modify I?

Provide a semi-decision procedure and justify your solution.

Solution to Exercise 3

For our construction of a semi-decision procedure we use another procedure Π_{int} that does the following:

1. Π_{int} takes as input a program Π and a string I.
2. Π_{int} simulates the run of Π on I, and returns the output of Π.

Π_{int} terminates on any instance of **KEEP-SOME**.

We define a semi-decision procedure Π_{ks} for **KEEP-SOME** as follows:

Solution to Exercise 3 (continued)

Boolean Π_{ks}(*String* Π)

$i := 0$
while (*true*) do {

let L be the set of all strings with length i

if there is a string $I \in L$ s.t. $\Pi_{int}(\Pi, I) = I$, then return *true*

$i := i + 1$
}

Solution to Exercise 3 (continued)

The procedure is correct. Indeed, if Π is a positive instance of **KEEP-SOME**, then there is a string I such that $\Pi(I) = I$. In particular, I has some length n. Since Π_{int} is a decision procedure, we are guaranteed that the procedure will reach the call $\Pi_{int}(\Pi, I)$ with output I and thus terminate with output *true*. If Π is a negative instance, then Π_{ks} does not terminate because the call to Π_{int} never returns I.

Exercise 4

Prove that the following problem is undecidable:

SOME-TRUE

INSTANCE: A program Π that takes as input a natural number and returns *true* or *false*. It is guaranteed that Π terminates on any input.

QUESTION: Does there exist a natural number k such that Π on k returns *true*?

Hint: For your proof you may assume the availability of an interpreter for instances of **HALTING**. In particular, you have available a decision 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**. Let (Π, I) be an arbitrary instance of **HALTING**. We build an instance Π' of **SOME-TRUE** by constructing Π' as follows:

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

In other words, for an instance $x = (\Pi, I)$, the instance $R(x)$ resulting from the reduction is 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 **SOME-TRUE**.

Solution to Exercise 4 (continued)

"$\Rightarrow$" Assume (Π, I) is a positive instance of **HALTING**, i.e. Π terminates on I. In particular, Π terminates on I within some n steps. Hence, $\Pi_{int}(\Pi, I, n) = true$ by definition of Π_{int} and $\Pi'(n) = true$ definition of Π'. That is, there is n such that $\Pi'(n) = true$. Thus Π' is a positive instance of **SOME-TRUE**.

"$\Leftarrow$" Assume Π' is a positive instance of **SOME-TRUE**, i.e. there exists a natural number n such that $\Pi'(n) = true$. By definition of Π', $\Pi_{int}(\Pi, I, n) = true$. By definition of Π_{int}, Π terminates on I within n steps. Thus (Π, I) is a positive instance of **HALTING**.

Exercise 5

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

Solution to Exercise 5

Define the relation

$$R = \{\langle (G, k), I\rangle \mid I \text{ is an independent set in } G \text{ with } |I| \geq k\}.$$

Clearly, R is a certificate relation for **INDEPENDENT SET**, since the following equivalences hold: (G, k) is a positive instance of **INDEPENDENT SET** $\Leftrightarrow$ there exists an independent set I in G with $|I| \geq k \Leftrightarrow \langle (G, k), I\rangle \in R$.

R is polynomially balanced because any set I of nodes from $G = (V, E)$ can be represented in space that is linear in the size of G. E.g. by a list whose length is $\leq |V|$.

Finally R is decidable in polynomial time because, given a graph G, an integer k, and a set of nodes I, one can check in polynomial time w.r.t. the size of (G, k) and I if I is an independent set in G. Likewise, one can check the condition $|I| \geq k$ in polynomial time.

Note that a "guess and check" procedure is obvious: guess a set of nodes and check in polynomial time whether it is an independent set of size $\geq k$.

Exercise 6

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

Solution to Exercise 6

The proof consists of two parts:

- (A) showing that **CLIQUE** is in NP, and
- (B) showing NP-hardness of **CLIQUE**, i.e. that for all problems $\mathcal{P}'$ in NP, $\mathcal{P}'$ is reducible to **CLIQUE**.

For the part (A), we define the relation

$$R = \{\langle (G, k), C \rangle \mid C \text{ is a clique in } G \text{ with } |C| \geq k\}.$$

We argue that R is a certificate relation for **CLIQUE**. Indeed, the following equivalences hold: (G, k) is a positive instance of **CLIQUE** $\Leftrightarrow$ there exists a clique C in G with $|C| \geq k \Leftrightarrow \langle (G, k), C \rangle \in R$.

R is polynomially balanced because any set C of nodes from G can be represented in space that is linear in the size of G.

Finally R is decidable in polynomial time because, given a graph G, an integer k, and a set of nodes C, one can check in polynomial time w.r.t. the size of (G, k) and C if C is a clique in G. One can check the condition $|C| \geq k$ in polynomial time as well.

Solution to Exercise 6 (continued)

For the part (B), we reduce **INDEPENDENT SET** to **CLIQUE**. Such a reduction suffices, because any problem $\mathcal{P}'$ in NP can be reduced to **CLIQUE** by composing (i) a reduction from $\mathcal{P}'$ to **INDEPENDENT SET** (which exists because of NP-completeness of **INDEPENDENT SET**), and (ii) the reduction from **INDEPENDENT SET** to **CLIQUE**.

Thus let (G, k) be an arbitrary instance of **INDEPENDENT SET**, i.e., G is an undirected graph and k is an integer. We construct the instance $(\bar{G}, k)$ of **CLIQUE**, where $\bar{G}$ is the complement of G. This reduction is feasible in polynomial time. It remains to prove the correctness.

- Assume G has an independent set I with $|I| \geq k$. We show that $\bar{G}$ has a clique C with $|C| \geq k$. Simply let $C = I$ and assume a pair $v_1, v_2 \in C$. Since C is an i.s. in G, by the definition of the complement graph, $[v_1, v_2] \in \bar{G}$. Thus, C is a clique in $\bar{G}$.
- Assume $\bar{G}$ has a clique C with $|C| \geq k$. We show that G has an i.s. I with $|I| \geq k$. Simply let $I = C$ and assume a pair $v_1, v_2 \in I$. Since I is a clique in $\bar{G}$, by the definition of the complement graph, $[v_1, v_2] \notin G$. Thus, I is an i.s. in G.

Exercise 7

We provide next a reduction from **2-COLORABILITY** to **2-SAT**. Let $G = (V, E)$ be an arbitrary undirected graph (i.e. an arbitrary instance of **2-COLORABILITY**), where $V = \{v_1, \ldots, v_n\}$. For the reduction we use propositional variables $x_1, \ldots, x_n$. Then the instance φ_G of **2-SAT**resulting from G is defined as follows:

$$\varphi_G = \bigwedge_{[v_i, v_j] \in E} (x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j).$$

Your task is to prove the "⇒" direction in the proof of correctness of the reduction, i.e. prove the following statement: if G is a positive instance of **2-COLORABILITY**, then φ_G is a positive instance of **2-SAT**.

Solution to Exercise 7

Suppose G is a positive instance of **2-COLORABILITY**. We have to show that φ_G is satisfiable. By assumption, there is a color assignment $f : \{0,1\} \to V$ such that $f(v_i) \neq f(v_j)$ for all $[v_i, v_j] \in E$. To show that φ_G is satisfiable, we define a truth assignment T such that, for all $i \in \{1,\ldots,n\}$, $T(x_i) = \mathit{true}$ if $f(v_i) = 1$, and $T(x_i) = \mathit{false}$ otherwise. We have to show that φ_G evaluates to *true* under T. Take an arbitrary edge $[v_i, v_j] \in E$. It remains to show that $(x_i \vee x_j)$ and $(\neg x_i \vee \neg x_j)$ both evaluate to *true* under T. Due to the assumption that f is a proper 2-coloring of G, we have $f(v_i) \neq f(v_j)$. Then due to the definition of T we have $T(x_i) \neq T(x_j)$. We are left with two possible cases:

- $T(x_i) = \mathit{true}$ and $T(x_j) = \mathit{false}$. Then trivially both clauses $(x_i \vee x_j)$ and $(\neg x_i \vee \neg x_j)$ evaluate to *true* under T.
- $T(x_i) = \mathit{false}$ and $T(x_j) = \mathit{true}$. Again, both clauses $(x_i \vee x_j)$ and $(\neg x_i \vee \neg x_j)$ evaluate to *true* under T.

Exercise 8

Provide a polynomial time algorithm for **2-COLORABILITY**. Argue why it only requires polynomial time.

Hint: We can assume that any instance G of the problem is a *connected* graph, i.e. there exists a path between any pair of nodes. In other words, G has no disconnected components.

Solution to Exercise 8

We provide the following procedure, which builds μ is stages. We assume in the procedure that V is a *list* of vertices and E is a *list* of edges in G.

Boolean *FindColoring*(*Graph G*)

0: if V is empty, then return *true*

1: Let e be the first element in V and let $\mu(e) := 0$

2: Choose the first edge $[e, e']$ in E such that e is colored and e' is not. If such an edge does not exist, return *true*

3: Color e' with the $\neg\mu(e)$, i.e. $\mu(e') = \neg\mu(e)$

4: If there exists an edge $[e_1, e_2]$ in E with $\mu(e_1) = \mu(e_2)$ then return *false*;

5: Go to 2.

Solution to Exercise 8 (continued)

We observe that with each iteration the number of edges where one endpoint is not colored decreases. That is, the number of iterations is bounded by the number of edges in E.

Furthermore, each single step requires linear time in the size of G.

Thus overall, the algorithm runs in time $\mathcal{O}(n^2)$, where n is the size of G.

Exercise 9

Argue that **1-COLORABILITY** can be solved in logarithmic space.

Solution to Exercise 9

We observe that a graph G is 1-colorable iff G has no edge. Thus checking 1-colorability reduces to checking whether a graph has no edges. The latter can be check in logarithmic space: one needs to traverse pairs of vertices and check whether they are related by an edge. Storing a vertex requires only logarithmic space, and we need to store only a constant number of vertices (that is, 2).

Exercise 10

Let $L = \{w \in \{0,1,2\}^* \mid w \text{ has an even number of occurrences of } 0\}$, i.e. L is the set of all strings w such that (a) w is built using symbols 0, 1 and 2, and (b) the number of occurrences of 0 in w is even. 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 $M(w) =$ "*yes*";
- if $w \notin L$, then $M(w) =$ "*no*".

Additionally, provide a high-level description of M.

Solution to Exercise 10

$M = (K, \Sigma, \delta, s)$ 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	$(s, 1, \rightarrow)$
s	2	$(s, 2, \rightarrow)$
s	$\sqcup$	(“yes”, $\sqcup, -$)
q	0	$(s, 0, \rightarrow)$
q	1	$(q, 1, \rightarrow)$
q	2	$(q, 2, \rightarrow)$
q	$\sqcup$	(“no”, $\sqcup, -$)

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

Solution to Exercise 10 (continued)

High-level description of M:

- In state s: If the symbol is $\triangleright$, 1 or 2, then move the head to the right without changing the state. If the symbol is 0, 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, then move the head to the right and change the state to s. If the symbol is 1 or 2, then move the head to the right without changing the state. If the symbol is $\sqcup$, then stop in the state "no".