Formale Methoden der Informatik
Block 1: Computability and Complexity

Exercises 1-10

SS 2012

Exercise 1 *Consider the problem* ***PROCEDURE NEG-ASSIGNMENT****, which is defined as follows:*

> ***PROCEDURE NEG-ASSIGNMENT***
>
> *INSTANCE: A triple* (Π, I, k)*, where (i)* Π *is a program that takes one string as input and outputs true or false, (ii)* I *is a string, and (iii)* k *is an integer variable used in program* Π*.*
> *QUESTION: Does variable* k *ever get assigned a negative value when the program* Π *is executed with input* I*?*

Prove that ***NEG-ASSIGNMENT*** *is undecidable. Prove the undecidability by providing a reduction from the* ***HALTING*** *problem to* ***NEG-ASSIGNMENT****, and arguing that your reduction is correct.*

Solution.

To prove the reduction, we have to show that

Π' halts on I <==> (Π, I, k) is a positive instance of NEG-ASSIGNMENT.

Let (Π', I') be an arbitrary instance of HALTING. Set $I = I' = S$, let be k an arbitrary integer occuring in Π and construct (Π, I, k) in the following way:

Procedure 1 NEG-ASSIGNMENT Procedure

```
function Π(String S)
    int k = 0
    call Π'(S)
    k = -1;
    return TRUE;
end function
```

" => "

Assume that (Π', I') is a positive instance of halting, so Π' halts on I because we set $I = I'$. Directly after the termination of Π', we set $k = -1$, so (Π, I, k) is a positive instance of NEG-ASSIGNMENT.

" <= "

Assume that (Π, I, k) is a positive instance of NEG-ASSIGNMENT, so it returns TRUE which means that k gets negative and the program terminates. But the only time when k gets set to a negative value is directly after the call of (Π', I'), where $I = I'$, so Π halts on I, so (Π', I') is a positive instance of HALTING.

Exercise 2 *Prove that the problem* ***NEG-ASSIGNMENT*** *from Exercise 1 is semi-decidable. To this end, provide a semi-decision procedure and justify your solution. Additionally, show that the co-problem of* ***NEG-ASSIGNMENT*** *is not semi-decidable.*

Solution. i)

Consider the following interpreter for NEG-ASSIGNMENT. For semi-decidability, the program must return TRUE for all positive instances (Π, I, k). The behaviour for negative instances (Π, I, k) is such that Π return FALSE or loops forever. So we have to distinguish the following cases:

- POSITIVE INSTANCE (Π', I')
  - When (Π', I') halts on input I, (Π', I') is a POSITIVE instance for our halting program, so the variable k gets assigned a negative value and therefore (Π, I, k) is a positive assignment of NEG-ASSIGNMENT and returns TRUE
- NEGATIVE INSTANCE (Π', I')
  - When (Π', I') is a NEGATIVE instance for our halting program and halts, the variable is NOT touched by our interpreter and remains positive, so (Π, I, k) is a NEGATIVE instance on I and returns FALSE
  - When (Π', I') is a NEGATIVE instance for our halting program but does not halt, the variable k cannot be NOT touched by our interpreter and remains positive, so (Π, I, k) is a NEGATIVE instance on I and loops forever, which is OK for a semi-decision procedure.

Procedure 2 Interpreter for NEG-ASSIGNMENT

```
function Π(String S)
    int k = 0
    rc = call Π'(S)
    if rc = TRUE then
        k = -1;
        return TRUE;
    else
        return FALSE;
    end if
end function
```

Solution. ii)

The opposite of NEG-ASSIGNMENT(k gets negative at least one time) is CO-NEG-ASSIGNMENT, which means that k is ALWAYS positive(FIXME!!!!! stimmt das so???)

Exercise 3 *Give a formal proof that* ***SUBSET SUM*** *is in* NP*, i.e. define a certificate relation and discuss that it is polynomially balanced and polynomial-time decidable.*
In the ***SUBSET SUM*** *problem we are given a finite set of integer numbers* $S = \{a_1, a_2, \ldots, a_n\}$ *and an integer number* t*. We ask whether there is a subset* $S' \subseteq S$ *whose elements sum is equal to* t*?*

Solution. 3 First we have to define a certificate relation with an arbitrary t:

$R = \{[(S,t), S'] | S' \subseteq S \text{ with } \sum(S') = t\}$

We argue that R is a certificate relation for SUBSET-SUM. (S,t) is a positive instance of SUBSET-SUM $\Leftrightarrow$ there exist a S' which sum is equal to t.

R is polynomially balanced because any subset of S can be represented in space that is linear in the size in S.

Finally R is decidable in polynomial time because, for a given subset of S we can easily check that a the sum is equal to t. This needs at most n computational steps. So it can be done in polynomial time.

Exercise 4 *Formally prove that* ***PARTITION*** *is* NP*-complete. For this you may use the fact that* ***SUBSET SUM*** *is* NP*-complete.*
In the ***PARTITION*** *problem we are given a finite set of integers* $S = \{a_1, a_2, \ldots, a_n\}$*. We ask whether the set* S *can be partitioned into two sets* S_1, S_2 *such that the sum of the numbers in* $S1$ *equals the sum of the numbers in* S_2*?*

Exercise 5 *Formally prove that* ***FREQUENCY ASSIGNMENT*** *is* NP*-complete. For this you may use the fact that a similar problem used in lectures is* NP*-complete.*
In the ***FREQUENCY ASSIGNMENT*** *problem we are given a set of transmitters* $T = \{t_1, t_2, \ldots, t_n\}$*,* k *frequencies, and the list of pairs of transmitters that interfer and therefore cannot use the same frequency. We ask whether there is an assignment of each transmitter to one of* k *frequencies such that there is no interference between the transmitters.*

Exercise 6 *Fomally prove that logical entailment is* $co - \mathsf{NP}$*-complete. The formal definition of entailment (* $\models$*) is this:* $\alpha \models \beta$ *if and only if, in every truth assignment in which* α *is true,* β *is also true.*

Exercise 7 *It is well known that the* ***k-COLORABILITY*** *problem is* NP*-complete for every* $k \geq 3$*. Recall that the instance of* ***k-COLORABILITY*** *is an undirected graph* $G = (V, E)$*. Suppose that we restrict this instance of* ***k-COLORABILITY*** *to trees. Can the restricted problem be solved with an algorithm that runs in polynomial time? If yes, provide such an algorithm.*

Solution.
By definition, a tree MUST NOT contain any cycles. By modifying DEPTH or BREADTH FIRST SEARCH, we get an algorithm for k-colorability that works in polynominally time. Consider a tree with n - nodes, where every parent-node has x child nodes. One extreme example would be a tree where every node has exactly 1 child. The

other extreme would be a tree with 1 parentnode and n-1 childnodes, directly attached to the parent.

Our algorithm has to process every level of the tree, starting at the root-node, coloring the node with one out of k colors. Afterwards, the next level has to be processed. If we take again the extreme with only 2 levels(1 parent, n-1 childs), we have to color (n-1) nodes and the program has finished. Otherwise, if every node has one child, only 1 node has to be colored with another color. This step then must be repeated n times.

Normaly, an average tree will be a mixture of the 2 extremes. This means that we have at most n levels, with $(n-1)$ nodes at each level, which means that the algorithm is $O(n^2)$.

```
for( every level of the tree )
{
   for( every node in this level )
   {
      color(node, color);
   }
   increment color(color);
}
```

function DFS(a)
  for each vertex $u \in V[G]$ **do**
    $i \leftarrow 0$ color[u] $\leftarrow$ k
    if $i + k \leq maxval$ **then**
      $i \leftarrow i + k$
    end if
  end for
end function

Exercise 8 *Provide a reduction of **N-Queens** problem to **SAT**. Give a proof sketch of the correctness of your reduction. Does this implies that the **N-Queens** is an* NP*-complete problem? Argue your answer.*

*In the **N-Queens** problem we are given n queens and an $n \times n$ chessboard. We ask whether we can place these n queens on the chessboard such that no two queens attack each other. Two queens attack each other if they are placed in the same row, or in the same column, or in the same diagonal.*

Exercise 9 *Consider the following problem:*

> ***N-SORTED-ELEMENTS***
>
> *INSTANCE: A non-empty list $L = (e_1, \ldots, e_n)$ of non-negative integers.*
> *QUESTION: Does the list L contain a sub-list of k consecutive sorted numbers in ascending order (from left to right)?*

*Argue that **N-SORTED-ELEMENTS** can be solved using only logarithmic space.*

Solution. 9 To show that problem is solvable in logarithmic space we have to take a look on the elements we need to solve the problem. We only need one pointer to an element in the list and a counter. Both require only logarithmic space. The pseudo algorithm for N-SORTED-ELEMENTS works as follows:

Procedure 3 Boolean N-SORTED-ELEMENTS

```
function BOOLEAN N-SORTED-ELEMENTS(List L, Integer k)
    int i ;
    int counter = 0;
    for i from 2 to length L do
        if L[i] > L[(i − 1)] then counter = counter + 1 then
            if counter ≥ k then return TRUE
            end if
        else counter = 0;
        end if
    end for
    return FALSE
end function
```

The procedure (see procedure 3) uses 2 variables i and a counter. For a list $L = (e1; ::: ; en)$, the variable i need log n bits to be represented. The variable counter also only needs logarithmic space. Such a value max needs only $log2^64xn) = log2^64 + logn = 64 + logn$ when we assume that the integers are 64-bit integer. So we can see that the procedure needs only logarithmic space to be represented.

Exercise 10 *Design a Turing machine that increments by one a value represented by a string of 0s and 1s.*

hline $p \in K$
$S \triangleright$
$S\ 0$