Exercise 1 *(15) Consider the following problem:*

EXISTS-HALTING
INSTANCE: A program Π*, which takes as input a string over the alphabet* $\{a, b, c, \ldots, z\}$*.* *QUESTION: Does there exist an input* I*, such that* Π *halts on* I*?*

Prove that the problem ***EXISTS-HALTING*** *is semi-decidable. For this, provide a procedure that shows the semi-decidability of the problem (i.e. a semi-decision procedure for* ***EXISTS-HALTING****) and argue that it is correct.*

Hint: *You may use the fact that the following problem is decidable:*

SPECIAL-HALTING

INSTANCE: A program Π*, a string* I *and a natural number* n*.*
QUESTION: Does the program Π *terminate on input* I *within* n *computational steps?*

Solution. From the assignment we know that there exists some decision procedure for **SPECIAL-HALTING**. In other words there exists a function $\Pi_{DS}(\Pi, I, n)$ which returns true for a yes instance and false for a no instance. With this function we can define a semi-decision procedure for **EXISTS-HALTING** as follows:

```
Boolean Π_i (Π)
    for(n = 1; true; n++)
        for each length(I) ≤ n
            if Π_DS(Π,I,n) then return true
```

It is easy to show that Π_i is a semi-decision procedure for **EXISTS-HALTING**. We can distinguish two cases:

Case 1: Suppose we have a yes instance of **EXISTS-HALTING**. So there exists some input I that Π halts on I. If this is the case our decision procedure will find this input because it tests for all combinations of I and n if Π halts on I with the help of our decision procedure for **SPECIAL-HALTING** and returns true.

Case 2: Suppose we have a no instance of **EXISTS-HALTING**. In this case there exists no input I so that Π halts on I. So our semi-decision procedure can not find an input. This means that Π_i will not terminate and run forever.

Exercise 2 *Show that for all programs* $p, wp(p, true) = false$ *if and only if* $sp(p, true) = false$.

Solution. We know the following:

$wp(p, true) = false$ holds for all programs p that don't terminate on any input

$sp(p, true) = false$ holds for all programs p that don't terminate on any input

So we can say that $wp(p, true) = false$ implies $sp(p, true) = false$ and vice versa.

Exercise 3 *Compute (not guess!) the waekest precondition of the following program for the postcondition $x = 4x_0$. What is the weakest liberal precondition for this program and postcondition ?*

```
y ← 3x;
while 2x ≠ y do
    x ← x + 1;
    y ← y + 1;
od
```

Solution. We first compute the weakest precondition:

$wp(y = 3x; while \quad 2x \neq y \quad do \quad x = x + 1; y = y + 1 \quad od, x = 4x_0)$
$wp(y = 3x, wp(while \quad 2x \neq y \quad do \quad x = x + 1; y = y + 1 \quad od, x = 4x_0))$

Compute the while loop:

$F_0 = 2x = y \wedge x = 4x_0$

$F_1 = 2x \neq y \wedge wp(x = x + 1; y = y + 1, 2x = y \wedge x = 4x_0)$
$F_1 = 2x \neq y \wedge 2(x + 1) = y + 1 \wedge x + 1 = 4x_0$
$F_1 = 2x \neq y \wedge 2x + 2 = y + 1 \wedge x = 4x_0 - 1$
$F_1 = 2x \neq y \wedge 2x + 1 = y \wedge x = 4x_0 - 1$
$F_1 = 2x + 1 = y \wedge x = 4x_0 - 1$

$F_2 = 2x \neq y \wedge wp(x = x + 1; y = y + 1, 2x + 1 = y \wedge x = 4x_0 - 1)$
$F_2 = 2x \neq y \wedge 2(x + 1) + 1 = y + 1 \wedge x + 1 = 4x_0 - 1$
$F_2 = 2x \neq y \wedge 2x + 3 = y + 1 \wedge x = 4x_0 - 2$
$F_2 = 2x \neq y \wedge 2x + 2 = y \wedge x = 4x_0 - 2$
$F_2 = 2x + 2 = y \wedge x = 4x_0 - 2$

Now we can guess F_i:

$F_i = 2x + i = y \wedge x = 4x_0 - i$

Now we have to proof F_i:

$F_{i+1} = 2x \neq y \wedge wp(x = x + 1; y = y + 1, 2x + i = y \wedge x = 4x_0 - i)$
$F_{i+1} = 2x \neq y \wedge 2(x + 1) + i = y + 1 \wedge x + 1 = 4x_0 - i$
$F_{i+1} = 2x \neq y \wedge 2x + 2 + i = y + 1 \wedge x + 1 = 4x_0 - i$
$F_{i+1} = 2x \neq y \wedge 2x + (i + 1) = y \wedge x = 4x_0 - (i + 1)$
$F_{i+1} = 2x + (i + 1) = y \wedge x = 4x_0 - (i + 1)$

$\exists i(i \geq 0 \wedge 2x + i = y \wedge x = 4x_0 - i)$
$\exists i(i \geq 0 \wedge i = y - 2x \wedge x = 4x_0 - i)$

$y - 2x \geq 0 \wedge x = 4x_0 - y + 2x$
$y - 2x \geq 0 \wedge y - x = 4x_0$

Now we can insert the weakest precondition from the while loop:

$wp(y = 3x, y - 2x \geq 0 \wedge y - x = 4x_0)$
$3x - 2x \geq 0 \wedge 3x - x = 4x_0$
$x \geq 0 \wedge 2x = 4x_0$
$x \geq 0 \wedge x = 2x_0$

So we finally get the weakest precondition $x \geq 0 \wedge x = 2x_0$

From the weakest precondition we can derive the weakest liberal precondition
$x = 2x_0 \vee x < 0$

Exercise 4 *State an algorithm that takes two finite Kripke structures $M_1 = (S_1, I_1, R_1, L_1)$ and $M_2 = (S_2, I_2, R_2, L_2)$ as input and returns a simulation relation from M_1 to M_2. In case M_2 simulates M_1, the relation computed by your algorithm must witness the simulation from M_1 to M_2. You may use pseudo code.*
Prove that your algorithm is complete, i.e., prove that your algorithm eventually computes a relation which is a simulation relation, and correct, i.e., prove that the computed relation witnesses the simulation from M_1 to M_2 in case that $M2$ simulates M_1.

Solution. We first define our algorithm:

(a) Add all pairs (s_1, s_2) in $S_1 \times S_2$ to H where $L(s_1) = L(s_2)$.

(b) Remove all pairs (s_1, s_2) from H when there is a State s_1' in S_1 with $(s_1, s_1') \in R_1$ and no state s_2' in S_2 with $(s_2, s_2') \in R_2$ and (s_1', s_2') not in H.

(c) Repeat (b) until H doesn't change anymore.

(d) Because we have to check if H witnesses the simulation from M_1 to M_2, we have to chekc if for each initial state $s_1 \in I_1$ if there is an initial state $s_2 \in I_2$ with $(s_1, s_2) \in H$.

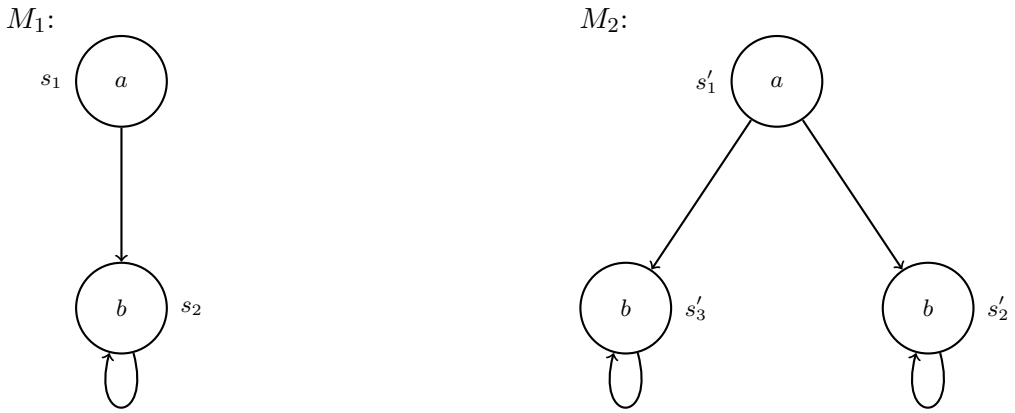
It is easy to see, that our algorithm terminates because we just remove elements from H but don't add new elements during computation (except point (a) where H is created). In the first step of our algorithm we just add pairs with the same label. So the first requirement for a simulation relation is guaranteed. In the second step we remove all elements where the second condition for a simulation relation doesn't hold. We repeat this step until the result doesn't change anymore. If we just remove elements the worst case would an empty result set but the algorithm would terminate in every case. In the last step we check the initial states. This step also terminates because in the worst case the number of initila states is equal to the number of states but in each case it's finite.

Exercise 5 *State two Kripke structures $M_1 = (S_1, I_1, R_1, L_1)$ and $M_2 = (S_2, I_2, R_2, L_2)$ and a relation H with the following properties:*

- *M_1 and M_2 are bisimilar,*
- *H is a simulation relation from M_1 to M_2 but not a bisimulation relation.*

Explain why M_1 and M_2 are bisimilar and why H is not a bisimulation relation.

Solution. Suppose we have the following two Kripke structures:



We can see that M_1 and M_2 are bisimilar because they have the same computation trees. We also know the following two simulation relations:

$$M_2 \text{ simulates } M_1 \; H = (s_1, s'_1), (s_2, s'_2)$$
$$M_1 \text{ simulates } M_2 \; H' = (s'_1, s1), (s'_2, s_2), (s'_3, s_2)$$

Now we can see that M_1 and M_2 simulate each other but there is no bisimulation relation because the tuple (s'_3, s_2) is not in H.