FORMAL METHODS EXAMS SOLUTIONS

- Block 1
- Block 2
- Block 3
- Block 4

Exam 20.10.2017 (FMI.175.pdf)
- Block 1
- Block 2
- Block 3
- Block 4

Exam 30.06.2017 (FMI.174.pdf)
- Block 1
- Block 2
- Block 3
- Block 4

Exam 05.05.2017 (FMI.173.pdf)
- Block 1
- Block 2
- Block 3
- Block 4

Exam 17.03.2017 (FMI.172.pdf)
- Block 1
- Block 2
- Block 3
- Block 4

Exam 27.01.2017 (FMI.171.pdf)
- Block 1
- Block 2
- Cx x Block 3
- Block 4

Exam 09.12.2016 (FMI.166.pdf)
- Block 1
- Block 1 (Cantor’s enumeration principle)
- Block 2
- Block 3
- Block 4

Exam 21.10.2016 (FMI.165.pdf)

Status

solved by student: solution was posted by a student, not approved by a tutor or professor
approved by tutor: solution was approved by a tutor
approved by professor: solution was approved by a professor

Exam 26.02.2021 (FMI 212)

Block 1

1.) Recall the **HALTING** problem which takes a program and a string as input, and consider the following variant thereof:

> **HALTING-X**
>
> INSTANCE: Two program Π_1, Π_2 that take a string as input.
>
> QUESTION: Does there exist at least one input string I such that both Π_1 and Π_2 halt on I.

(a) The following function f provides a polynomial-time many-one reduction from **HALTING** to **HALTING-X**: for a program Π and a string I let $f(\Pi, I) = (\Pi_1, \Pi_2)$ with

$$\begin{aligned} \Pi_1(\texttt{string } S) &= \texttt{call } \Pi(S)\texttt{; return;} \\ \Pi_2(\texttt{string } S) &= \texttt{if } (S \neq I)\ \{\texttt{while(true)}\{\}\}\texttt{; return;} \end{aligned}$$

Show that (Π, I) is a yes-instance of **HALTING** $\iff$ (Π_1, Π_2) is a yes-instance of **HALTING-X**.

(9 points)

Solution

(Proposed by student)

=> Suppose an arbitrary instance (Π, I) is a yes-instance of HALTING. Then Π halts on I. Since by construction Π_1 halts on S if Π halts on S, and Π halts on S if S = I; and Π_2 halts on S if S = I, (Π_1, Π_2) is a yes-instance of HALTING-X because both Π_1 and Π_2 halt on I.

<= Suppose an arbitrary instance (Π_1, Π_2) is a yes-instance of HALTING-X. Then both Π_1 and Π_2 halt on I. Since Π_1 halts on I and Π_1 only halts if Π halts on I, (Π, I) is a yes-instance of HALTING.

(b) Tick the correct statements (for ticking a correct statement a certain number of points is given; ticking an incorrect statement results in a substraction of the same amount; you cannot go below 0 points):

- Since **HALTING** is decidable, our reduction from (a) shows that **HALTING-X** is decidable.
- Since **HALTING** is undecidable, our reduction from (a) shows that **HALTING-X** is undecidable.
- Since **HALTING** is semi-decidable, our reduction from (a) shows that **HALTING-X** is semi-decidable.
- Since **HALTING** is not semi-decidable, our reduction from (a) shows that **HALTING X** is not semi-decidable.
- A reduction from **HALTING-X** to **HALTING** would show that **HALTING-X** is semi-decidable.
- A reduction from **HALTING-X** to **HALTING** would show that **HALTING-X** is undecidable.

(6 points)

Solution

(Proposed by student)

1. [_] False. HALTING *is not* decidable.
2. [X] True. A reduction from HALTING to HALTING-X shows that HALTING-X must be at least as hard as HALTING. Since HALTING is undecidable, HALTING-X must also be undecidable.
3. [_] False. A reduction from HALTING to HALTING-X shows that HALTING-X must be at least as hard as HALTING. While HALTING is semi-decidable, semi-decidability is *easier* than undecidability, hence the reduction only proves undecidability.
4. [_] False. HALTING *is* semi-decidable.
5. [X] True. Reducing from HALTING-X to HALTING would show that HALTING-X is not harder than HALTING. Since HALTING is semi-decidable, HALTING-X cannot be harder than semi-decidable and must therefore be semi-decidable as well.
6. [_] False. Reducing from HALTING-X to HALTING would show that HALTING-X is not harder than HALTING. HALTING-X could therefore be decidable as well.

Block 2

Block 3

Block 4

Exam 29.01.2021 (FMI 211)

Block 1

Recall the NP-complete problem **SAT** and its specialization **3SAT** which is also NP-complete:

3SAT

INSTANCE: A propositional formula φ in 3-CNF, i.e. of the form $\bigwedge_{i=1}^{n}(l_{i1} \vee l_{i2} \vee l_{i3})$.

QUESTION: Does there exists a truth assignment T that makes φ true?

Now consider the following further restriction:

3SATX

INSTANCE: A propositional formula φ in 3-CNF, where each variable occurs positively at most two times (i.e., at most two times a variable is not in the scope of negation).

QUESTION: Does there exists a truth assignment T that makes φ true?

The following function f provides a polynomial-time many-one reduction from **3SAT** to **3SATX**: for a formula $\varphi = \bigwedge_{i=1}^{n}(l_{i1} \vee l_{i2} \vee l_{i3})$ over variables V let

$$f(\varphi) = \Big(\bigwedge_{v \in V} \big((\neg v \vee \neg v \vee \neg \bar{v}) \wedge (v \vee v \vee \bar{v})\big) \wedge \bigwedge_{i=1}^{n} (l_{i1}^* \vee l_{i2}^* \vee l_{i3}^*)\Big)$$

where $l_{ij}^* = \neg v$ if $l_{ij} = \neg v$ and $l_{ij}^* = \neg \bar{v}$ if $l_{ij} = v$ (i.e., we replace each literal v in φ by $\neg \bar{v}$ for all $v \in V$).

It can be shown that φ is a yes-instance of **3SAT** $\iff$ $f(\varphi)$ is a yes-instance of **3SATX**. Provide a proof for the $\implies$ direction.

TODO

=> Assume φ is a positive Instance of 3SAT.

We can split our reduction into two parts M = μ1 ∧ μ2

μ1: Each Variable in φ is mapped to another variable v' with the following relation for all v ϵ V: v = ¬v'. With μ1 we guarantee that each of our variables must be either true or false and because of the relation of our mapping v' will always be negative but represent v.

μ2: We substitute each of our non-negative variables in φ, with ¬v' and the negative ones can stay the same. So assuming that φ is a positive instance of 3SAT, it must also be a positive instance of 3SATX.

Tick the correct statements (for ticking a correct statement a certain number of points is given; ticking an incorrect statement results in a subtraction of the same amount; you cannot go below 0 points):

- Since 3SATX is a special case of 3SAT, 3SATX must be NP-hard.
- Since 3SAT is NP-hard, our reduction from Exercise 1 (a) shows that 3SATX is in NP.
- ✓ Since 3SAT is NP-hard, our reduction from Exercise 1 (a) shows that 3SATX is NP-hard.
- ✓ Since 3SATX is a special case of SAT, 3SATX must be contained in NP.
- Since 3SAT is in NP, our reduction from Exercise 1 (a) shows that 3SATX is NP-hard.
- ✓ Since 3SATX is a special case of 3SAT, 3SATX must be contained in NP.

Block 2

We consider the theory $\mathcal{T}_A$ of arrays from the lecture.

i. What is the signature of this theory?
ii. What kinds of axioms are available in this theory? Please name them.
iii. Consider a $\mathcal{T}_A$-formula ψ and suppose that ψ is not valid. What is a counter-example to $\mathcal{T}_A$-validity of ψ and what properties has this counter-example to satisfy?

i.Σ_A ={(Array Read, Array Write), (Equality)}
ii.

- **Read-Over-Write 1**
- **Read-Over-Write 2**
- **Array congruence**
- **Reflexivity**
- **Symmetry**
- **Transitivity**

iii. TODO

Group A:

Consider the theory $\mathcal{T}_A$ of arrays and the following formula

$$\varphi: \quad \big(\forall j\ a[j] \doteq b\langle i \triangleleft v\rangle[j]\big) \rightarrow a[i] \doteq v\ .$$

If φ is $\mathcal{T}_A$-valid, then provide a proof in the semantic argument method (similarly to the proofs in the lecture and on the extra sheets). If φ is not $\mathcal{T}_A$-valid, then provide a counter-example.
Besides the equality axioms reflexivity, symmetry and transitivity, you have the following ones for arrays.

- $\forall a, i, j\ \big(i \doteq j \rightarrow a[i] \doteq a[j]\big)$ (array congruence)
- $\forall a, v, i, j\ \big(i \doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq v\big)$ (read-over-write 1)
- $\forall a, v, i, j\ \big(i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq a[j]\big)$ (read-over-write 2)

Solution:

Given: $\varphi : \forall j.(a[j] \doteq b\langle i \triangleleft v\rangle[j]) \rightarrow a[i] \doteq v$

Proof by contradiction:

1. $\mathcal{I} \not\models \varphi$	[assumption]
2. $\mathcal{I} \models (\forall j.(a[j] \doteq b\langle i \triangleleft v\rangle[j])$	[1., MP, $\rightarrow$]
3. $\mathcal{I} \not\models a[i] \doteq v$	[1., MP, $\rightarrow$]
4. $\mathcal{I}\{j \leftarrow i\} \models a[i] \doteq b\langle i \triangleleft v\rangle[i]$	[2., us $\forall$]
5. $\mathcal{I}\{j \leftarrow i\} \models b\langle i \triangleleft v\rangle[i] \doteq v$	[4., Semantics of Read-over-Write 1]
6. $\mathcal{I}\{j \leftarrow i\} \models a[i] \doteq v$	[4., 5., Symmetry & Transitivity]
7. $\mathcal{I}\{j \leftarrow i\} \models \bot$	[3., 6., contradiction]

Group B:

(b) Consider the theory $\mathcal{T}_A$ of arrays and the following formula

$$\varphi: \quad a[i] \neq v \rightarrow \big(\exists j\ a[j] \neq b\langle i \triangleleft v\rangle[j]\big) \ .$$

If φ is $\mathcal{T}_A$-valid, then provide a proof in the semantic argument method (similarly to the proofs in the lecture and on the extra sheets). If φ is not $\mathcal{T}_A$-valid, then provide a counter-example.

Besides the equality axioms reflexivity, symmetry and transitivity, you have the following ones for arrays.

- $\forall a, i, j\ \big(i \doteq j \rightarrow a[i] \doteq a[j]\big)$ (array congruence)
- $\forall a, v, i, j\ \big(i \doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq v\big)$ (read-over-write 1)
- $\forall a, v, i, j\ \big(i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq a[j]\big)$ (read-over-write 2)

Please be precise. In a proof indicate exactly why proof lines follow from some other(s) and name the used rule. If you use derived rules you have to prove them. **(11 points)**

Solution:

$\mathcal{I} \models \neg\phi$	by assumption
$\mathcal{I} \models a[i] \neq v$	by 1), by semantics of $\neg, \rightarrow$
$\mathcal{I} \not\models \exists j(a[j] \neq b\langle i \triangleleft v\rangle[j])$	by 1), by semantics of $\neg, \rightarrow$
$\mathcal{I} \models \forall(a[j] = b\langle i \triangleleft v\rangle[j])$	by 3), by semantics of $\neg$
$\mathcal{I} \models a[i] = b\langle i \triangleleft v\rangle[i]$	by 4), by $\forall$, for some $k \in \mathcal{U}$, $\mathcal{I}_1 = \mathcal{I}_2\{i \leftarrow j\}$
$\mathcal{I} \models b\langle i \triangleleft v\rangle[i] = v$	by $i = i$, read-over-write 1
$\mathcal{I} \models a[i] = v$	by 5) and 6), transitivity
$\mathcal{I} \models \bot$	contradiction between 2) and 7)

Block 3

a)

Let p be the following program:

$$
\begin{array}{l}
x := 0; z := 0; y := 0; \\
\textbf{while } y < n \textbf{ do} \\
\quad x := x + 2; \\
\quad z := z + 5; \\
\quad y := y + 1 \\
\textbf{od}
\end{array}
$$

Give a loop invariant for the **while** loop in p and prove the validity of the partial correctness triple $\{n > 1\}\ p\ \{z - x = 3 * n\}$.

Find invariant:
y[k+1] = y[k]+1 => y[k] = y[0]+k => y = k
z[k+1] = z[k]+5 => z[k] = z[0]+5k => z = 5k => z = 5y
x[x+1] = x[k]+2 => x[k] = x[0]+2k => x = 2k => x = 2y

Invariant: y <= n ∧ x = 2y ∧ z = 5y

Proof VC(p, B) ∧ A => wlp(p, B) as usual

b)

(b) Let p be the following program:

$$
\begin{array}{l}
n := 0 \\
\textbf{while } x > 0 \wedge y > 0 \textbf{ do} \\
\quad \textbf{if } n = 0 \textbf{ then} \\
\quad\quad y := y + 1; \\
\quad\quad x := x - 2; \\
\quad \textbf{else} \\
\quad\quad x := x + 1 \\
\quad\quad y := y - 2 \\
\quad n := 1 - n \\
\textbf{od}
\end{array}
$$

Provide a loop variant t for the **while** loop in p strong enough to prove the validity of the total correctness triple $[x \geq 0 \wedge y \geq 0]\ p\ [x \leq 0 \vee y \leq 0]$. You may assume the invariant to be *true*.

You are **not** required to write a proof here, just state a suitable variant.

(2 points)

t = x + y

c) (Approved by professor)

(c) Is the following theorem correct?

"For all assertions A, B and programs p, it holds that $[A]\ p\ [B]$ is valid if and only if $(VC(p,B)\ \wedge\ (A\ \Rightarrow\ wp\,(p,B))$). "

If it is, give an argument why. If not, what is wrong?

Be concise and write no more than 1-2 sentences. **(2 points)**

The direction
******
IF
* VC(p,B) holds
AND
* A => wp(p,B) holds
THEN

[A] p [B] is valid
****
Is valid.

However,
*****
IF
[A] p [B] valid
THEN
VC(p,B) holds AND A => wp(p,B) holds
******

Now this is not the case, for the following reason. In case "p" is a loop, your invariant I=wp(p,B). However, this invariant might be too strong/weak and you may fail proving A=>wp(p,B) and/or VC(p,B), even though [A] p [B].

So in general the THEOREM
****
[A] p [B] is valid IFF (VC(p,B) AND A => wp(p,B) holds)
*****
is not correct (only for loop-free p it is correct).
What holds for arbitrary p is only one direction, as explained above, that is:
IF (VC(p,B) AND A => wp(p,B) holds) then [A] p [B] is valid. *

d)

(d) Let n, m be integer-valued constants and A an assertion. Is there a state σ and a program p such that $\sigma \models \{true\}$ p $\{false\}$? If so, provide such a state σ and program p. If, not, explain why there exists no such σ.

(2 points)

We define an arbitrary σ since all states σ entail true and p = **while true do skip od**, then there is no σ', such that $\langle \sigma, p \rangle \rightarrow \sigma'$. Thus the partial correctness triple is satisfied.

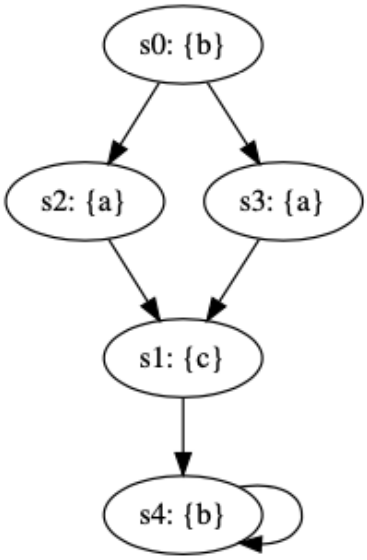
Block 4

A

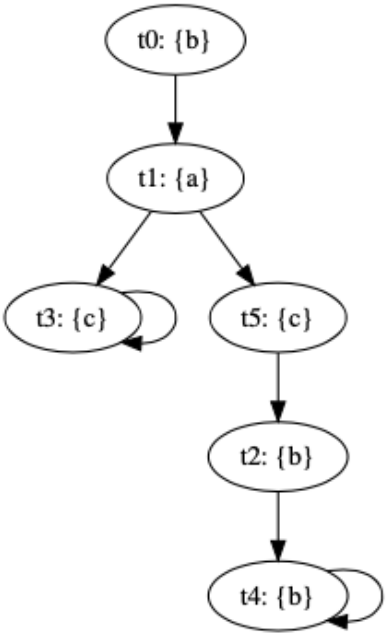
a)

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



Kripke structure M_2:



(4 points)

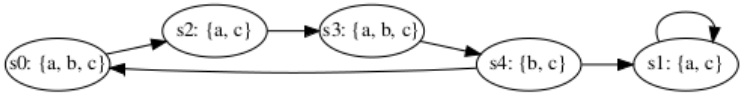
If we strictly use the algorithm given in the slides we get the following:

H_0={(s0,t0), (s0,t2), (s0,t4),(s1,t3),(s1,t5),(s2,t1),(s3,t1),(s4,t0),(s4,t2),(s4,t4)}
H_1={(s0,t0),(s1,t5),(s2,t1),(s3,t1),(s4,t2),(s4,t4)}
H_2={(s0,t0),(s1,t5),(s3,t1),(s4,t2),(s4,t4)}

H=H_2={(s0,t0),(s1,t5),(s3,t1),(s4,t2),(s4,t4)}

b)

Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?
Hint: If φ is a path formula, list the states s_i such that $M, s_i \models \mathbf{A}\varphi$.

G(a): CTL*, LTL, s1

E[(a) U (b)]: CTL*, CTL, s0,s2,s3,s4

F(a&b): CTL*, LTL, s0, s2, s3

AF(c): CTL*, CTL, s0,s2,s3,4,s1

X(b): CTL*, LTL, s2,s3

c)

Proof by student:

4.c) $\phi : G(a \Rightarrow Fb) \Rightarrow ((GFa) \Rightarrow (GFb))$

Proof by contradiction: we assume that there exists some model, s. t. the formula ϕ is falsified.

in the following proof, we use the MT... *Modus Tollens*, MP ... *Modus Ponens*

0.	$M, \pi \not\models \phi$	[Assumption]
1.	$M, \pi \not\models (G(a \Rightarrow Fb)) \Rightarrow ((GFa) \Rightarrow (Gfb))$	[0, MT]
2.	$M, \pi \not\models \neg(G(a \Rightarrow Fb)) \vee ((GFa) \Rightarrow (Gfb))$	[1., semantics of $\rightarrow$]
3.	$M, \pi \models (G(a \Rightarrow Fb)) \wedge \neg((GFa) \Rightarrow (Gfb))$	[2., MP, $\vee$]
4.	$M, \pi \models (G(a \Rightarrow Fb))$	[3. $\wedge$]
5.	$M, \pi \not\models ((GFa) \Rightarrow (GFb))$	[3.,MT, $\wedge$]
6.	$\forall i \geq 0; M, \pi^i \models (a \Rightarrow Fb)$	[4., MP, semantics of G]
7.	$\forall i, i \geq 0; M, \pi^i \models a \Rightarrow (\exists j, j \geq i, M, \pi^j \models b)$	[6., semantics of F]
8.	$\forall i, i \geq 0; M, \pi^i \not\models a$	[7., $\Rightarrow$]
9.	$\exists j, j \geq i \geq 0, M, \pi^j \models b$	[7., $\Rightarrow$]
10.	$M, \pi \not\models (\neg(GFa) \vee (GFb))$	[5., $\Rightarrow$]
11.	$M, \pi \not\models \neg((GFa) \wedge \neg(GFb))$	[10., $\vee$]
12.	$M, \pi \models ((GFa) \wedge \neg(GFb))$	[11., $\neg$]
13.	$M, \pi \models GFa$	[12., $\wedge$]
14.	$M, \pi \not\models GFb$	[12., $\wedge$]
15.	$\forall i, i \geq 0, M, \pi^i \not\models b$	[14., semantics of G and F]
16.	$\forall i, i \geq 0, M, \pi^i \models \neg b$	[15., $\neg$]
17.	$\forall j, j \geq i \geq 0, M, \pi^j \not\models \neg b$	[9., $\exists$]
18.	$\forall j, j \geq i \geq 0, M, \pi^j \models b$	[17., $\neg$]
19.	$M, \pi \models \bot$	[17., 19., contradiction]

The initial assumption (0.) must be therefore false and the path formula must be a valid for all models, i.e. a tautology.

Alternative solution:

(i)

$\varphi = \boldsymbol{G}(a \Rightarrow \boldsymbol{F}b) \Rightarrow ((\boldsymbol{GF}a) \Rightarrow (\boldsymbol{GF}b))$

First of all we summarize the following definitions and equivalencies which are needed for this proof:
$\boldsymbol{G}(a \Rightarrow \boldsymbol{F}b) \Leftrightarrow$ for all $i \geq 0$, $M, \pi^i \models a \Rightarrow \boldsymbol{F}b$
$\boldsymbol{GF}a \Leftrightarrow$ for all $i \geq 0$, $M, \pi^i \models \boldsymbol{F}a$
$\boldsymbol{F}a \Leftrightarrow$ there exists a $k \geq 0$ such that $M, \pi^k \models a$

$\boldsymbol{GF}a \Leftrightarrow$ for all $i \geq 0$ there exists a $k \geq i$ such that $M, \pi^k \models a$
$\boldsymbol{GF}b \Leftrightarrow$ for all $i \geq 0$ there exists a $k' \geq i$ such that $M, \pi^{k'} \models b$

To prove that φ is a tautology we assume that the following holds: $\boldsymbol{G}(a \Rightarrow \boldsymbol{F}b)$

Since we have to prove $(\boldsymbol{GF}a) \Rightarrow (\boldsymbol{GF}b)$ under the assumption that $\boldsymbol{G}(a \Rightarrow \boldsymbol{F}b)$ holds, we assume $\boldsymbol{GF}a$ also holds and show that $\boldsymbol{GF}b$ follows from this assumption.

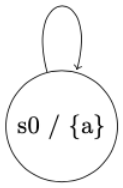
From $\boldsymbol{GF}a$ it follows that there is a k for any i for which $M, \pi^k \models a$ holds. From $\boldsymbol{G}(a \Rightarrow \boldsymbol{F}b)$ we know that for all i, $M, \pi^i \models a \Rightarrow \boldsymbol{F}b$ holds. Since it holds for any i it follows that $M, \pi^k \models a \Rightarrow \boldsymbol{F}b$. We can follow that there is a $k' \geq k$ for all i for which the following holds: $M, \pi^{k'} \models b$. Due to transitivity of $k' \geq k$ and $k \geq i$ we can write $k' \geq i$.
Now we have the following: For all i there exists a $k' \geq i$ such that $M, \pi^{k'} \models b$ which is the exact definition of $\boldsymbol{GF}b \Leftrightarrow$ for all $i \geq 0$ there exists a $k' \geq i$ such that $M, \pi^{k'} \models b$.

We have shown that $\boldsymbol{GF}b$ is true under the assumption that $(\boldsymbol{GF}a) \Rightarrow (\boldsymbol{GF}b)$ and $\boldsymbol{GF}a$ holds. We can follow that φ is a tautology.

(ii)

$\psi = ((\boldsymbol{GF}a) \Rightarrow (\boldsymbol{GF}b)) \Rightarrow \boldsymbol{G}(a \Rightarrow \boldsymbol{F}b)$

We have to provide a counterexample where $(\boldsymbol{GF}a) \Rightarrow (\boldsymbol{GF}b)$ holds and $\boldsymbol{G}(a \Rightarrow \boldsymbol{F}b)$ does not hold.
$(\boldsymbol{GF}a) \Rightarrow (\boldsymbol{GF}b)$ trivially holds if both $\boldsymbol{GF}a$ and $\boldsymbol{GF}b$ does not hold. Then we have to provide a counterexample where $\boldsymbol{G}(a \Rightarrow \boldsymbol{F}b)$ does not hold.

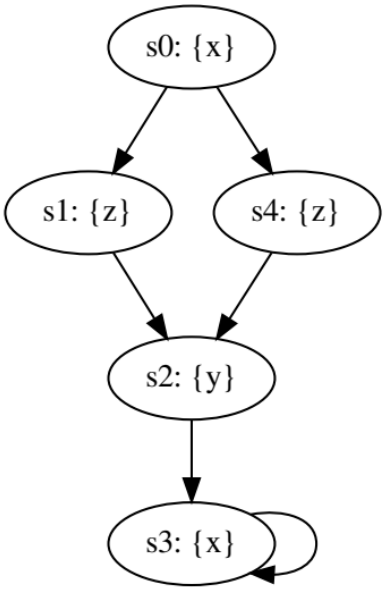


B

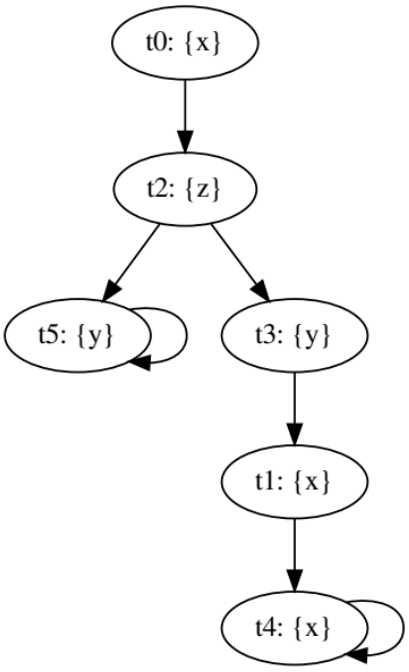
a)

Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



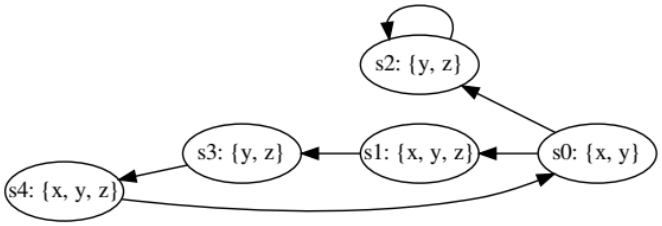
Kripke structure M_2:



H = {(s0,t0),(s1,t2),(s4,t2),(s2,t3),(s3,t1),(s3,t4)}

b)

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?
Hint: If φ is a path formula, list the states s_i such that $M, s_i \models \mathbf{A}\varphi$.

φ	CTL	LTL	CTL*	States s_i
$\mathbf{E}[(z)\ \mathbf{U}\ (x)]$	☐	☐	☐	
$\mathbf{X}(x)$	☐	☐	☐	
$\mathbf{AF}(y)$	☐	☐	☐	
$\mathbf{G}(z)$	☐	☐	☐	
$\mathbf{F}(x \wedge z)$	☐	☐	☐	

ϕ	**CTL**	**LTL**	**CTL***	**States s_i**
E[(z) **U** (x)]	X		X	s0, s1, s3, s4
X(x)		X	X	s3,s4
AF(y)	X		X	s0,s1,s2,s3,s4
G(z)		X	X	s2
F(x □ z)		X	X	s1,s3,s4

c)

LTL tautologies

Prove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π in M, or find a Kripke structure M and path π in M, for which the formula does not hold and justify your answer.

i. $((\mathbf{GF}x) \Rightarrow (\mathbf{GF}y)) \Rightarrow \mathbf{G}(x \Rightarrow \mathbf{F}y)$

ii. $\mathbf{G}(x \Rightarrow \mathbf{F}y) \Rightarrow ((\mathbf{GF}x) \Rightarrow (\mathbf{GF}y))$

i) is falsifiable

ii) is a Tautology

Exam 09.12.2020 (FM 205)

Block 1

A

1.) An undirected graph (V, E) is called a *mirror graph* if the following conditions hold:

- V can be partitioned into two sets of equal size $V' = \{v'_1, \ldots, v'_n\}$ and $V'' = \{v''_1, \ldots, v''_n\}$;
- for all $i \in \{1, \ldots, n\}$ the edge $[v'_i, v''_i]$ is in E;
- for all $i, j \in \{1, \ldots, n\}$, $[v'_i, v'_j] \in E$ iff $[v''_i, v''_j] \in E$;
- no other edges are contained in E.

In words, a mirror graph has each vertex from V' connected to a clone from V'' and the graph over V' is mirrored in V''.

Consider the following new problem **3COLMG** and recall the NP-complete problem **3COL** defined below:

3-COLORABILITY-MIRROR GRAPH (3COLMG)

INSTANCE: A mirror-graph $G = (V, E)$.

QUESTION: Does there exists a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

3-COLORABILITY (3COL)

INSTANCE: An undirected graph $G = (V, E)$.

QUESTION: Does there exists a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

(a) The following function f provides a polynomial-time many-one reduction from **3COL** to **3COLMG**: for a directed graph $G = (\{v_1, \ldots, v_n\}, E)$, let

$$f(G) = \big(\{v'_1, v''_1, \ldots, v'_n, v''_n\}, \{[v'_i, v''_i] \mid i = 1 \ldots n\} \cup \{[v'_i, v'_j], [v''_i, v''_j] \mid [v_i, v_j] \in E\} \big)$$

Show that G is a yes-instance of **3COL** $\iff$ $f(G)$ is a yes-instance of **3COLMG**.

(9 points)

1.a) First of all we provide a prove for the $\Rightarrow$ direction, which means G is a yes-instance of 3COL $\Rightarrow$ $f(G)$ is a yes-instance of 3COLMG. For simplicity we define $f(G) = G^* = (V^*, E^*)$.

Suppose $G = (V, E)$ is a positive instance of 3-COLORABILITY with $V = \{v_1, v_2, ..., v_n\}$, so there exists a function: $\psi : \{v_1, v_2, ..., v_n\} \to \{0, 1, 2\}$ where $\psi(v_i) \neq \psi(v_j)$ for each $[v_i, v_j] \in E$.

We construct a function ψ^* for the graph $f(G)$ as follows:
$\psi^*(v_i^{'}) = \psi(v_i)$
$\psi^*(v_i^{''}) = (\psi(v_i) + 1)\ mod\ 3$ for any index $i \in \{1, ..., n\}$

As we can see, for each vertex $v_i^{'}$, ψ^* assigns the same color as ψ vor vertices v_i. For each vertex $v_i^{''}$, ψ^* assigns a different color than ψ for vertices v_i. If we just look at the second case for vertices of type $v_i^{''}$ the function ψ^* is bijective for this case.
Now we have to show, that ψ^* assigns a different color to adjacent vertices of $f(G)$, which means the following:

(a) $\psi^*(v_i^{'}) \neq \psi^*(v_i^{''})$ for each $[v_i^{'}, v_i^{''}] \in E^*$

(b) $\psi^*(v_i^{'}) \neq \psi^*(v_j^{'})$ for each $[v_i^{'}, v_j^{'}] \in E^*$

(c) $\psi^*(v_i^{''}) \neq \psi^*(v_j^{''})$ for each $[v_i^{''}, v_j^{''}] \in E^*$

Case a It is easy to see, that ψ^* assigns a different color to $v_i^{'}$ and $v_i^{''}$. ψ^* assigns the color $\psi(v_i)$ to $v_i^{'}$ and color $(\psi(v_i) + 1)\ mod\ 3$ to $v_i^{''}$, which is different.

Case b Suppose $\psi^*(v_i^{'}) = \psi^*(v_j^{'})$ for any $[v_i^{'}, v_j^{'}] \in E^*$. Then there would be any $[v_i, v_j] \in E$ where $\psi(v_i) = \psi(v_j)$. We assumed that G is 3COL which leads us to a contradiction.

Case c Suppose $\psi^*(v_i^{''}) = \psi^*(v_j^{''})$ for any $[v_i^{''}, v_j^{''}] \in E^*$. Then there would be any $[v_i, v_j] \in E$ where $\psi(v_i) = \psi(v_j)$. We assumed that G is 3COL which leads us to a contradiction.

Now we have to prove the $\Leftarrow$ direction, which means $f(G)$ is a yes-instance of 3COLMG $\Rightarrow$ G is a yes-instance of 3COL. We assume $f(G)$ is a yes-instance of 3COLMG which means that the graph $f(G)$ is 3-colorable. The graph G is a sub-graph of graph $f(G)$, so G is also 3-colorable.

(b) Tick the correct statements (for ticking a correct statement a certain number of points is given; ticking an incorrect statement results in a substraction of the same amount; you cannot go below 0 points):

- Given that **3COL** is in NP, our reduction shows that **3COLMG** is also in NP.
- Given that **3COL** is in NP, our reduction shows that **3COLMG** is NP-hard.
- Given that **3COL** is NP-hard, our reduction shows that **3COLMG** is also NP-hard.
- Given that **3COL** is NP-hard, our reduction shows that **3COLMG** is in NP.
- Since **3COLMG** is a special case of **3COL**, **3COLMG** must be contained in NP.
- Since **3COLMG** is a special case of **3COL**, **3COLMG** must be NP-hard.

(6 points)

B

1.) An undirected graph (V, E) is called a *distance2-graph* if the there exists a vertex in V such that all vertices are reached via a path of length at most 2.

For example, the graph $(\{a_1, b_1, c_1, a_2, b_2, c_2, x\}, \{[a_1, a_2], [b_1, b_2], [c_1, c_2], [a_2, x], [b_2, x], [c_2, x]\})$ is a distance2-graph (since each vertex is reached from x via a path of length at most 2), while $(\{a, b, c, d\}, \{[a, b], [b, c], [c, d]\})$ is not.

Consider the following new problem **3COLD2** and recall the NP-complete problem **3COL** defined below:

3-COLORABILITY-DISTANCE2-GRAPH (3COLD2)

INSTANCE: A distance2-graph $G = (V, E)$.

QUESTION: Does there exists a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

3-COLORABILITY (3COL)

INSTANCE: An undirected graph $G = (V, E)$.

QUESTION: Does there exists a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

(a) The following function f provides a polynomial-time many-one reduction from **3COL** to **3COLD2**: for a directed graph $G = (\{v_1, \ldots, v_n\}, E)$, let

$$\begin{aligned} f(G) \quad = \big(\quad & \{v'_1, v''_1, \ldots, v'_n, v''_n\} \cup \{x\}, \\ & \{[v'_i, v''_i], [v''_i, x] \mid i = 1 \ldots n\} \cup \\ & \{[v'_i, v'_j] \mid [v_i, v_j] \in E\}\big) \end{aligned}$$

Show that G is a yes-instance of **3COL** $\iff$ $f(G)$ is a yes-instance of **3COLD2**.

(b) Tick the correct statements (for ticking a correct statement a certain number of points is given; ticking an incorrect statement results in a substraction of the same amount; you cannot go below 0 points):

- Since **3COLD2** is a sub-problem of **3COL**, **3COLMG** must be NP-hard.
- Since **3COLD2** is a sub-problem of **3COL**, **3COLD2** must be contained in NP.
- Given that **3COL** is NP-hard, our reduction shows that **3COLD2** is also NP-hard.
- Given that **3COL** is NP-hard, our reduction shows that **3COLD2** is in NP.
- Given that **3COL** is in NP, our reduction shows that **3COLD2** is also in NP.
- Given that **3COL** is in NP, our reduction shows that **3COLD2** is NP-hard.

We do not get any information from 3COLD2 being a subproblem of 3COL, thus 1 and 2 are false.
3 is correct since 3COLD2 needs to be at least as hard as 3COL, if we can reduce 3COL to 3COLD2.
4 is false because 3COLD2 could be harder than NP.
5 is false as we do not get an upper limitation on complexity of 3COLD2 by reducing 3COL to 3COLD2.
6 is false because 3COLD2 could be in NP but not NP-hard.

Block 2

c)

(b) Consider the function P, defined as follows.

Algorithm 1: The function P

Input: x, y, two non-negative integers
Output: The computed non-negative integer value for x, y

```
1 if x == 0 then
2     return y + 1;
3 else if y == 0 then
4     return P(x − 1, 1);
5 else return P(x − 1, P(x, y − 1));
```

Show, using well-founded induction, that

$$\forall x \forall y \left((x \in \mathbb{N}_0 \wedge y \in \mathbb{N}_0) \rightarrow \mathrm{P}(x, y) > y \right)$$

(10 points)

(c) Suppose $\mathrm{P_C}$ is a correct implementation of P in the `C` programming language with x and y of type unsigned integers of size 64 bit (i.e., of type `uint64_t`). Is

$$\mathrm{P}(x', y') = \mathrm{P_C}(x', y')$$

true for all integers x', y' satisfying $0 \leq x', y' \leq$ `UINT64_MAX`, where `UINT64_MAX` is the largest value for a variable of type `uint64_t`?
If so, then prove this fact. Otherwise provide a counterexample with an exact explanation of what is computed and what is happening. **(2 points)**

Counterexample: (approved by professor**)**
We define x' = 0 and y' = UINT64_MAX.

Then by definition of P, P(x',y') = UINT64_MAX +1.

But for uint, the C standard says that integer overflow means "wrapping around". So P_c(x',y') = 0, since in line 2, the computation “y +1” where y= UINT64_MAX would result in such a wrap around and therefore 0 would be returned.
For signed integers, the overflow behaviour is undefined by the C standard.

Block 3

A

3.) (a) Let p be the following program:

$$
\begin{array}{l}
y := n; \\
\textbf{while } y > 0 \textbf{ do} \\
\quad x := x - 4 * y + 2; \\
\quad y := y - 1 \\
\textbf{od}
\end{array}
$$

Give a loop invariant for the **while** loop in p and prove the validity of the partial correctness triple $\{n > 0 \wedge x = 2 * n^2 + 1\}\ p\ \{x = 1\}$.

(9 points)

3.a.

At first we have to give a loop invariant.

$$\begin{gathered}
y[N+1] = y[N] - 1 \\
y[N] = y[N-1] - 1 \\
y[N] = y[0] - \textstyle\sum_{k=0}^{N-1} 1 \\
\boldsymbol{y[N] = y[0] - N}
\end{gathered}$$

$$\begin{gathered}
x[N+1] = x[N] - 4 * y[N] + 2 \\
x[N+1] = x[N] - 4 * (y[0] - N) + 2 \\
x[N] = x[N-1] - 4 * (y[0] - (N-1)) + 2 \\
x[N] = x[N-1] - 4 * y[0] + 4 * (N-1) + 2 \\
x[N] = x[0] - 4 * \textstyle\sum_{k=0}^{N-1} y[0] + 4 * \sum_{k=0}^{N-1} k - 1 + 2 * \sum_{k=0}^{N-1} 1 \\
x[N] = x[0] - 4y[0]N + 4 * \frac{(N-1)N}{2} + 2N \\
x[N] = x[0] - 4y[0]N + 2(N-1)N + 2N \\
x[N] = 2n^2 + 1 - 4Nn + 2N^2 - 2N + 2N \\
x[N] = 2n^2 + 1 - 4Nn + 2N^2 \\
x[N] = 2 * (n^2 - 2Nn + N^2) + 1 \\
x[N] = 2 * (n - N)^2 + 1
\end{gathered}$$

As we have seen $y[N] = y[0] - N$, so it follows that $N = y[0] - y[N]$ and therefore $N = n - y$.
Now we have the following:

$$\begin{gathered}
x[N] = 2 * (n - (n - y))^2 + 1 \\
x[N] = 2 * (n - n + y)^2 + 1 \\
\boldsymbol{x[N] = 2y^2 + 1}
\end{gathered}$$

Finally we take the invariant $\boldsymbol{I : x = 2y^2 + 1 \wedge y \geq 0}$.

We set
$I : x = 2y^2 + 1 \wedge y \geq 0$,
$A : n > 0 \wedge x = 2n^2 + 1$,
$b : y > 0$,
$B : x = 1$,
$q :$ **while** $y > 0$ **do** $x := x - 4y + 2; y := y - 1$ **od** and
$r : x := x - 4y + 2; y := y - 1$

To prove the hoare triple $\{n > 0 \wedge x = 2n^2 + 1\}\ p\ \{x = 1\}$ we have to show that $VC(p, B) \wedge (A \rightarrow wlp(p, B))$. It is sufficient to calculate $VC(q, B)$ because the verification condition of the assignment $y := n$ always evaluates to true.

$$\begin{gathered}
VC(q, B) = (I \wedge \neg b) \rightarrow B \wedge \\
(I \wedge b) \rightarrow wlp(r, I) \wedge \\
VC(r, I).
\end{gathered}$$

$(I \wedge \neg b) \rightarrow B$

$$x = 2y^2 + 1 \wedge y \geq 0 \wedge \neg(y > 0) \rightarrow x = 1 \Leftrightarrow$$
$$x = 2y^2 + 1 \wedge y \geq 0 \wedge y \leq 0 \rightarrow x = 1 \Leftrightarrow$$
$$x = 2y^2 + 1 \wedge y = 0 \rightarrow x = 1 \Leftrightarrow$$
$$x = 2 * 0 + 1 \rightarrow x = 1 \Leftrightarrow$$
$$x = 1 \rightarrow x = 1 \checkmark$$

$\boldsymbol{wlp(r, I)}$

$$wlp(x := x - 4y + 2; y := y - 1, x = 2y^2 + 1) =$$
$$wlp(x := x - 4y + 2, wlp(y := y - 1, x = 2y^2 + 1)) =$$
$$wlp(x := x - 4y + 2, x = 2(y - 1)^2 + 1) =$$
$$x - 4y + 2 = 2(y - 1)^2 + 1 =$$
$$x - 4y + 2 = 2(y^2 - 2y + 1) + 1 =$$
$$x - 4y + 2 = 2y^2 - 4y + 2 + 1 =$$
$$\boldsymbol{x = 2y^2 + 1}$$

$\boldsymbol{(I \wedge b) \rightarrow wlp(r, I)}$

$$(x = 2y^2 + 1 \wedge y \geq 0 \wedge y > 0) \rightarrow x = 2y^2 + 1$$
$$x = 2y^2 + 1 \rightarrow x = 2y^2 + 1 \checkmark$$

$\boldsymbol{VC(r, I)}$

$$VC(x := x - 4y + 2; y := y - 1, x = 2y^2 + 1 \wedge y \geq 0) = \top$$

$\boldsymbol{wlp(p, B)}$

$$wlp(p, B) =$$
$$wlp(y := n, wlp(\textbf{while } y > 0 \textbf{ do } x := x - 4y + 2; y := y - 1 \textbf{ od}, B)) =$$
$$wlp(y := n, x = 2y^2 + 1 \wedge y \geq 0) =$$
$$\boldsymbol{x = 2n^2 + 1 \wedge n \geq 0}$$

$\boldsymbol{A \rightarrow wlp(p, B)}$

$$n > 0 \wedge x = 2n^2 + 1 \rightarrow x = 2n^2 + 1 \wedge n \geq 0$$
$$x = 2n^2 + 1 \rightarrow x = 2n^2 + 1 \checkmark$$
$$n > 0 \rightarrow n \geq 0 \checkmark$$

We have shown that $VC(p, B) \wedge (A \rightarrow wlp(p, B))$ is valid. It follows that the given hoare triple $\{n > 0 \wedge x = 2n^2 + 1\}\ p\ \{x = 1\}$ is also valid.

3.b.

{x=0 & y=0} p {x=y}

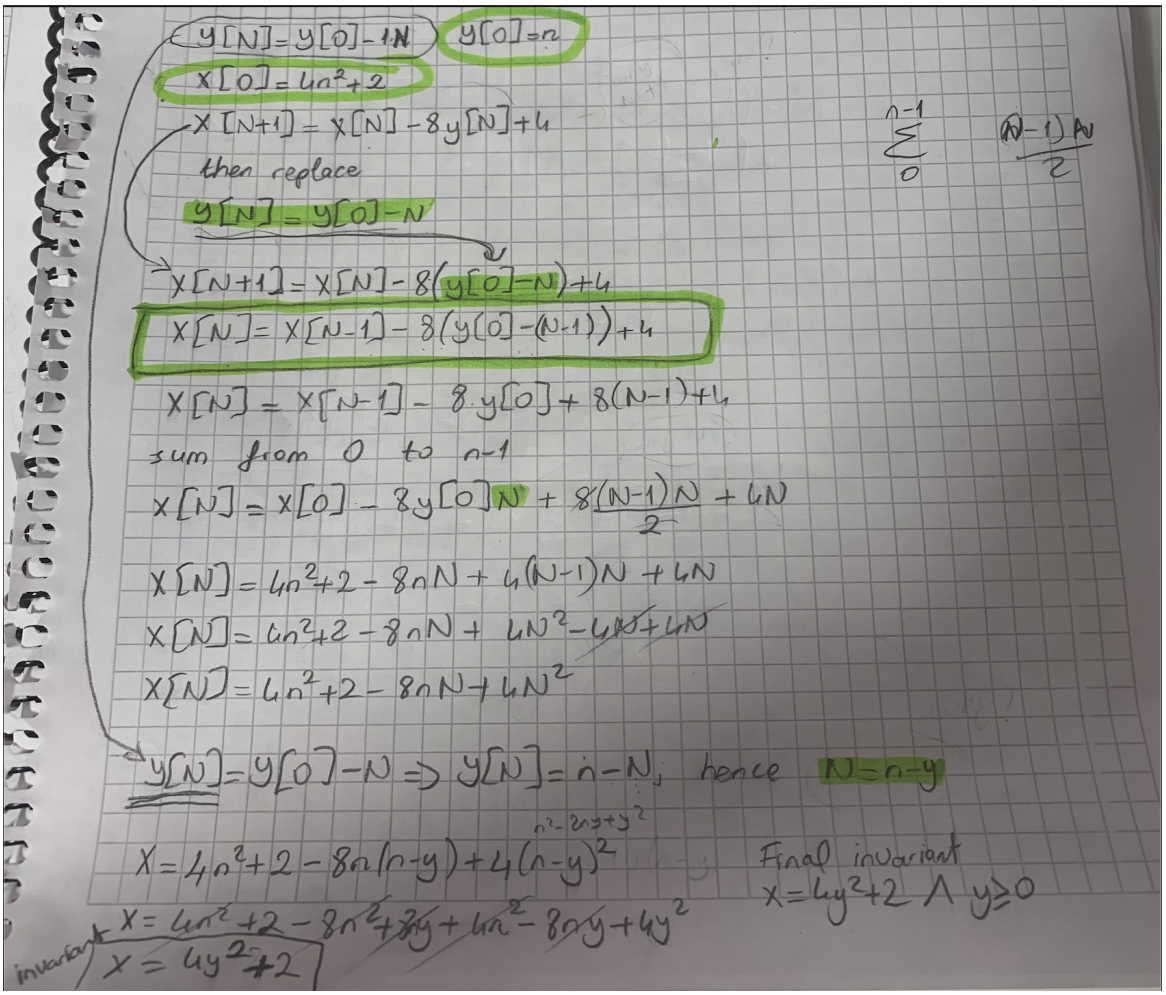
3.c.

We have to show that there exists no such σ for which $\sigma \models [N \geq 0]$ $\boldsymbol{abort}$ $[B]$ holds.
$\sigma \models [A]\ p\ [B]$ is defined as follows:

$$\sigma \models [A]\ p\ [B] \Leftrightarrow$$
$$[\sigma \models A \Rightarrow (\forall \sigma' \langle p, \sigma \rangle \rightarrow \sigma' \Rightarrow \sigma' \models B)] \wedge$$
$$[\sigma \models A \Rightarrow (\exists \sigma' \langle p, \sigma \rangle \rightarrow \sigma')]$$

Hence $\langle \boldsymbol{abort}, \sigma \rangle$ is not defined, such a state σ' cannot exists and it follows that $\sigma \models [N \geq 0]$ $\boldsymbol{abort}$ $[B]$ does not hold.

B



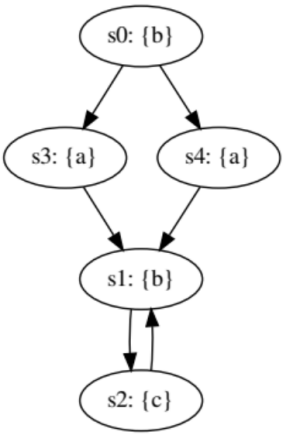
(Lösung von Professorin)

Block 4

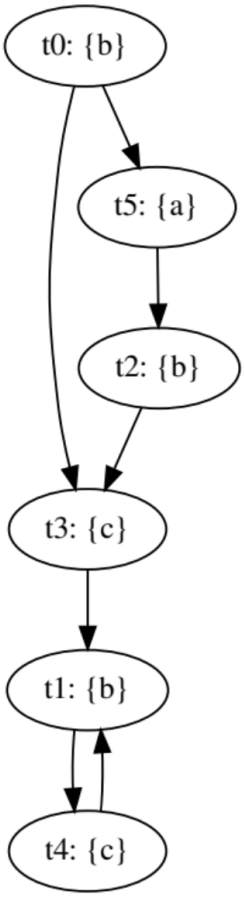
4.a)

Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



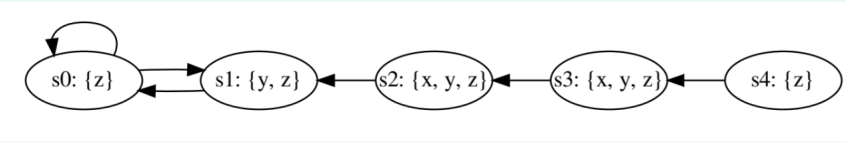
Kripke structure M_2:



H= {(s0,t0), (s3,t5), (s4,t5), (s1,t2), (s1,t1), (s2,t3), (s2,t4)}

4.b)

Consider the following Kripke structure M:



For each of the following formulae φ

$\mathbf{AX}(y), \mathbf{EG}(z), \mathbf{F}(y), \mathbf{G}(x \wedge z), \mathbf{E}[(z)\ \mathbf{U}\ (x)]$

i. write below whether the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

```
AX(y) is CTL and CTL* true in states s2,s3,s4.

EG(z) is CTL and CTL* true in all states. (s0, s1, s2, s3, s4)

F(y) is LTL and CTL* true in states s1,s2,s3,s4.

G(x & z) is LTL and CTL* false in all states.

E[(z) U (x)] CTL and CTL* is true in states s2,s3,s4.
```

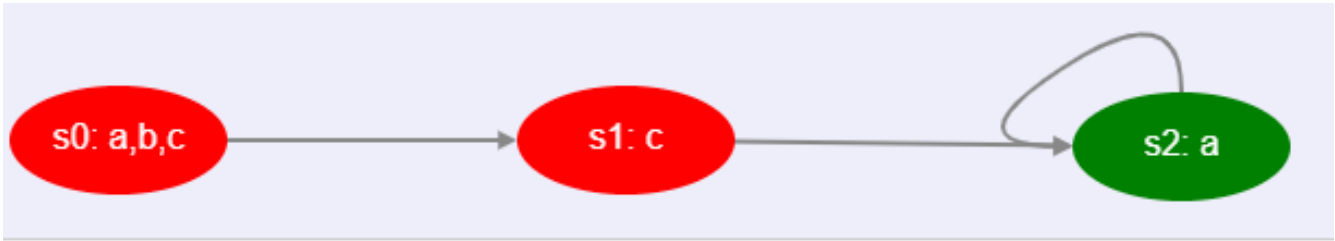
4.c) LTL tautologies

Prove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π in M, or find a Kripke structure M and path π in M, for which the formula does not hold and justify your answer.

i. $((\neg a\ \mathbf{U}\ b)\ \mathbf{U}\ \neg c) \Leftrightarrow (\neg a\ \mathbf{U}\ (b\ \mathbf{U}\ \neg c))$

ii. $(\mathbf{FG}a) \Rightarrow\ \mathbf{G}(a \vee\ \mathbf{XF}a)$

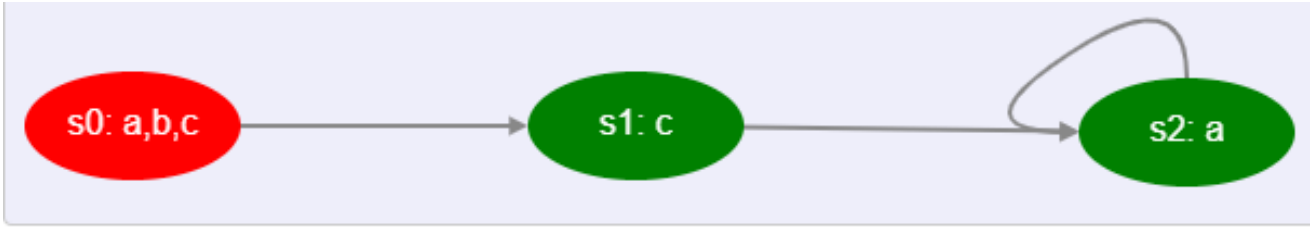
i) is not Tautology



(!a U b) U !c | Check | (!a U b) U !c | (!a U (b U !c))

ccepts NuSMV CTL or LTL specifications.

```
(!a U b) U !c is true in states s2.
```



(!a U (b U !c))

Check

(!a U b) U !c

(!a U (b U !c))

Accepts NuSMV CTL or LTL specifications.

```
(!a U (b U !c)) is true in states s1,s2.
```

S1 is different so this is a counterexample?
It was easier to construct by replacing x=~a, y=b, z=~c to only have positive attributes and then the states are (y)-(x)-(z)

ii) is Tautology
Solution by student:
Suppose M,p|=FGa for arbitrary path p and structure M. Then M,p|=Ga or M,p|=XFGa by semantics of F.
Case 1. Suppose M,p|=Ga. Then M,p|=G(a OR XFa) holds by semantics of OR.
Case 2. Suppose M,p|=XFGa.
Then there is a state s (other than the first one) and from s on always holds a.
Then there is a single state t (other than the first one) for which holds a. This means M,p|=XFa.
As from s on a holds always, that state t can also always be found.
Consequently even M,p|=GXFa. Then by semantics of OR holds M,p|=G(a OR XFa)

Another solution by student:
Suppose M,p|=FGa for arbitrary path p and structure M. Then there is some state sx that has a (M,sx|=a) and all states that follow it have a.
Let sy be an arbitrary state.
Case 1: sy (strictly) before sx. Then the next state or some state after that has a. So XFa. Then a OR XFa by semantics of OR.
Case 2: sy=sx. Then M,sy|=a because M,sx|=a. Then a OR XFa.

Case 3: sx before sy. Then sy follows sx. Then M,sy|=a. Then a OR XFa.
In every case a OR XFa. As sy was chosen arbitrarily, G(a OR XFa).

Exam 16.10.2020 (FMI 204)

Block 1

1.a.)

1.) Consider the following decision problem:

> **HALTING AFTER LINE-FLIP (HALF)**
>
> INSTANCE: A tuple (Π, I), where Π is a program that takes a string as input; I a string.
>
> QUESTION: Do there exist two consecutive lines of code in Π, such that when the two lines are flipped (i.e., the order of the two lines is reversed) in Π, the resulting program (a) is syntactically correct and (b) halts on I?

(1) By providing a suitable many-one reduction from the **HALTING** problem, prove that **HALF** is undecidable.

Corrected by Student who said this was accepted at his exam (sorry I removed the comment by accident and cant restore it)

Let (Π,I) be an arbitrary instance of HALTING.

ManyToOne Reduction from HALTING to HALF:

```
String Π'(String S) {
  String s = "s";
  String t = "t";
  Π(I);
  return 0;
}
```

```
String Π'(String S) {
  String s = "s";
  String t = "t";
  Π(I);
  String x = "x";
 Π(I);
  return 0;
}
```

(Π,I) is a positive instance of HALTING ⇔ (Π',I) is a positive instance of HALF

Assuming (Π,I) is a positive instance of HALTING -> (Π',I) halts and there are 2 consecutive lines that can be flipped and it is still syntactically correct. (flipping s & t)

Assuming (Π',I) is a positive instance of HALF -> (Π,I) is also a positive instance because b) indicates that positive instances of HALF halt.

1.b.)

(2) Is **HALF** semi-decidable? Explain your answer. **(15 points)**

1b Semi-decidable:

Solution by student.

Consider an interpreter Πi that does (in a manner that shares computation power) for every pair of consecutive lines:
Swap them to get the resulting Program Π'. Then

- If Π' is syntactically incorrect, Πi returns false.
- If Π' is syntactically correct
    - and returns, then Πi returns true.
    - But runs forever, then Πi cannot say anything about Π'. Here it is important that the computation power is shared or rather that the simulation for all possible Π' is scheduled to avoid simulating a single Π' infinitely long and never get to the other ones.

This behavior makes Πi a semi-decision procedure and HALF semi-decidable.

Other solution by student: (not from exam, questionable)
This is not a complete solution, only an example for the previous argument

Example instance (Π,"forever") with Π being the following program (for simplicity, ignore argc… and say the first command line parameter argv[1] is the necessary input string. So on the CLI we enter './a.out "forever"')

```
1 #include <stdio.h>
2 int main(int argc, char *argv[]) {
3         int dummy1;
4         int dummy2;
5         int test;
6         test = 0;
7         while (test == 0) { printf("%s\n", argv[1]); }
8         test = 2;
9         return test;
10 }
```

Swapping the two consecutive lines 7 and 8 gives a syntactically correct program that terminates on the input "forever". Therefore, (Π,"forever") is a positive instance of HALF.

Block 2

2.a.)

2.) (a) Let φ be the first-order formula

$$\forall x \forall y \left[\left((r(x,y) \wedge p(x)) \rightarrow p(y) \right) \wedge \left(r(x,y) \rightarrow (p(y) \rightarrow p(x)) \right) \right] .$$

i. Is φ valid? If yes, present a proof. If no, give a counter-example and prove that it falsifies φ.

ii. Replace r in φ by $\doteq$ (equality) resulting in ψ. Is ψ E-valid? Argue formally!

(5 points)

2.b.) (see also EXTRASHEETS)

Show the following:

$$\varphi^{uf} \text{ is satisfiable iff } FC^E \wedge flat^E \text{ is satisfiable.}$$

FC^E and $flat^E$ are obtained from φ^{uf} by Ackermann's reduction. (Hint: FC^E is the same for φ^{uf} and $\neg\varphi^{uf}$.)

Solution to the exercise

We make precise here on which formula FC^E and $flat^E$ depend. Both are computed according to AR. First observe that

$$\varphi^{uf} \text{ is valid} \quad \text{iff} \quad FC^E(\varphi^{uf}) \rightarrow flat^E(\varphi^{uf}) \text{ is valid.}$$

Taking the above hint into account, we get

$$\begin{array}{lll} \neg\varphi^{uf} \text{ is valid} & \text{iff} & FC^E(\neg\varphi^{uf}) \rightarrow flat^E(\neg\varphi^{uf}) \text{ is valid} \\ & \text{iff} & FC^E(\varphi^{uf}) \rightarrow flat^E(\neg\varphi^{uf}) \text{ is valid.} \end{array}$$

Since φ^{uf} and $flat^E(\varphi^{uf})$ have the same propositional structure (only some arguments of literals change during the computation of $flat^E(\varphi^{uf})$ from φ^{uf}), the equality $flat^E(\neg\varphi^{uf}) = \neg flat^E(\varphi^{uf})$ holds. Consequently,

$$\neg\varphi^{uf} \text{ is valid} \quad \text{iff} \quad FC^E(\varphi^{uf}) \rightarrow \neg flat^E(\varphi^{uf}) \text{ is valid.} \qquad (*)$$

Then we have:

			Explanation
φ^{uf} is sat	iff	$\neg\varphi^{uf}$ is *not* valid	Ψ is sat iff $\neg\Psi$ is not valid
	iff	$FC^E(\varphi^{uf}) \rightarrow \neg flat^E(\varphi^{uf})$ is *not* valid	from $(*)$ above
	iff	$\neg\neg(FC^E(\varphi^{uf}) \rightarrow \neg flat^E(\varphi^{uf}))$ is *not* valid	$\neg\neg\Psi \equiv \Psi$
	iff	$\neg(FC^E(\varphi^{uf}) \rightarrow \neg flat^E(\varphi^{uf}))$ is sat	Ψ is sat iff $\neg\Psi$ is not valid
	iff	$FC^E(\varphi^{uf}) \wedge flat^E(\varphi^{uf})$ is sat	basic propositional manipulations

Block 3

3.a.)

3.) (a) Let p be the following program:

$x := 1;$
$y := -2;$
$z := 2;$
while $x < N$ **do**
  $x := x - 2 * y - 2 * z + 1;$
  $y := y - 2$
  $z := z + 2$
od

Provide a formal proof of the partial correctness triple $\{N \geq 1\}\ p\ \{z = 2 * N\}$.

Note: Give an appropriate loop invariant for the **while** loop in p, to be further used in proving the partial correctness of the above Hoare triple.

(10 points)

Find Invariant :
Y[n] = y[0] - 2n => Y[n] = -2 - 2n
Z[n] = z[0] + 2n => Z[n] = 2n + 2

X[n] = x[0] - 2 * ∑y - 2 * ∑z + 1n
X[n] = x[0] - 2 * ∑(-2 -2n) - 2 * ∑(2n + 2) + 1n
Replacing ∑ with i=0 until n-1 and (a0 +an)*n/2
X = 1 - 2 * (-2 +(-2-2(n-1)))*n/2 - 2 * (2+(2(n-1) +2))*n/2 + 1n
X = 1 -2 * (-2 + (-2 -2n +2))*n/2 - 2 * (2+ (2n -2 +2))*n/2 +1n
X = 1 - 2 * (-2-2n)*n/2 - 2 * (2n +2) *n/2 +1n
X = 1 - 2 * (-n -n²) - 2*(n² +n) + 1n
X = 1 + 2n + 2n² - 2n² - 2n +1n
X = n+1 -> n = x -1

Replace n in z
Z = 2 * (x-1) + 2>
Z = 2x

I : z = 2x ∧ x <= N

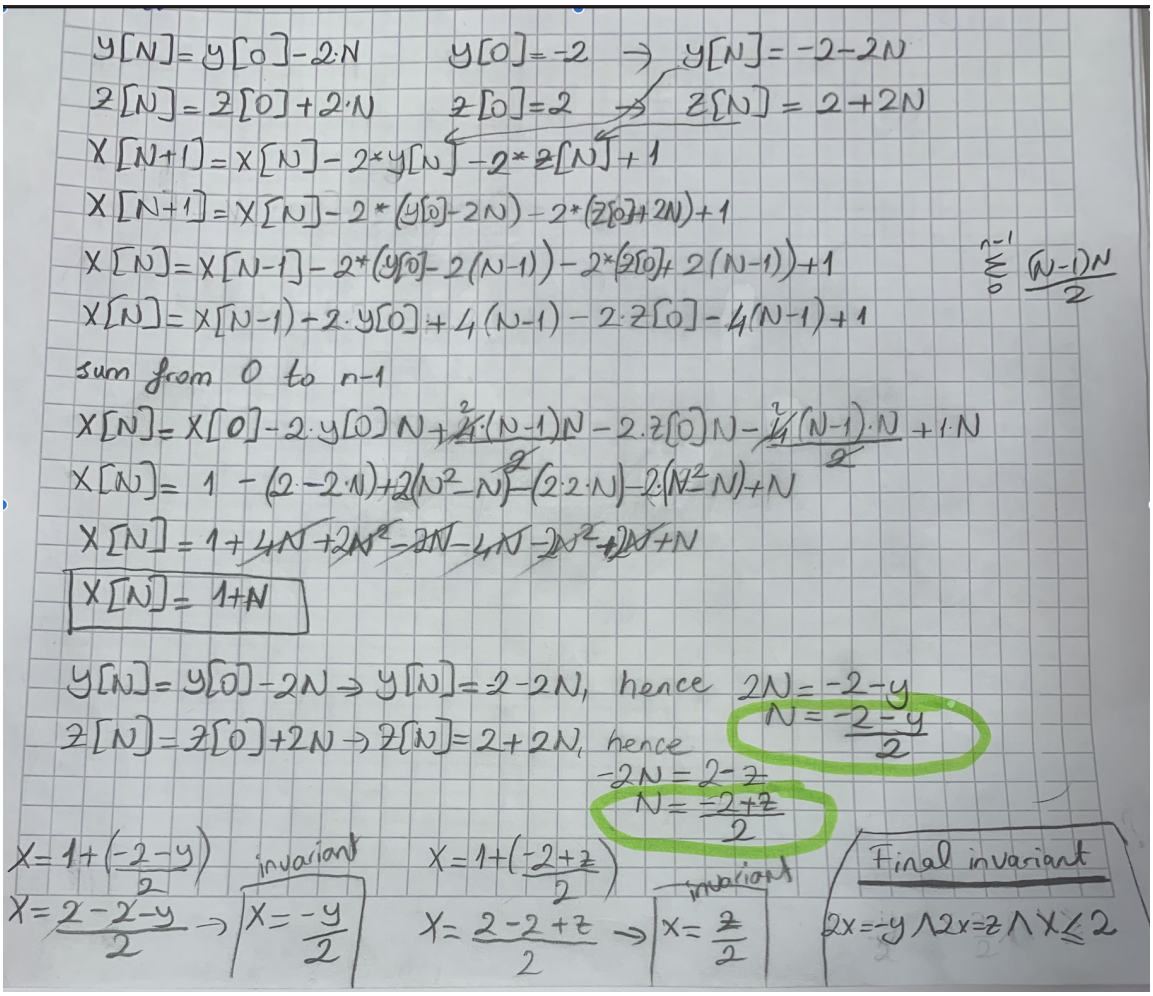
Now show VC(p,B) ∧ (A =>wlp(p,B)) …

(other student - idk if this is correct)
I : z = 2x ∧ **y = -2x** ∧ x <= N
(I ∧ !b) => B
z = 2x ∧ y = -2x ∧ x <= N ∧ x >= N => z = 2N
x <= *N* ∧ *x* >= *N* → *x* = *N*
z = 2N ∧ y = -2x ∧ x = N => z = 2N → true

(I ∧ b) => wlp(p, I)
wlp(p, I):
z+2 = 2(x-2y-2z +1) ∧ y-2 = -2*(x-2y-2z+1) ∧ x-2y -2z +1 <= N
z = 2x -4y -4z ∧ y-2 = -2x +4y +4z -2 ∧ x - 2y -4x + 1 <= N
z = 2x ∧ y = -2x ∧ x+1 <= N

(I ∧ b) => wlp(p, I)
z = 2x ∧ y = -2x ∧ x <=N ∧ x < N => z =2x ∧ y = -2x ∧ x+1 <=N
x <=N ∧ x < N → x < N
z = 2x ∧ y = -2x ∧ x < N => z = 2x ∧ y = -2x ∧ x+1 <= N → true

VC(p, I) = correct



(Die Professorin hat die Lösung gecheckt.)

3.a.
Let $I : y = -2x \wedge z = 2x \wedge x \leq N$ be the loop invariant, $A : N \geq 1$, $B : z = 2 * N$ and $b : x < N$. We show that the hoare triple $\{N \geq 1\}\ p\ \{z = 2 * N\}$ holds by proving the following formula:

$VC(p, z = 2 * n) \wedge (N \geq 1 \rightarrow wlp(p, z = 2 * N)$

wlp(p, z = 2 * N)

$$wlp(p, z = 2 * N) \equiv$$
$$wlp(x := 1, wlp(y := -2, wlp(z := 2, wlp(\textbf{while } x < N \textbf{do } x := x - 2y - 2z + 1; y := y - 2; z := z + 2; \textbf{od}, z = 2 * N)))) \equiv$$
$$wlp(x := 1, wlp(y := -2, wlp(z := 2, I))) \equiv$$
$$wlp(x := 1, wlp(y := -2, wlp(z := 2, y = -2x \wedge z = 2x \wedge x \leq N))) \equiv$$
$$wlp(x := 1, wlp(y := -2, y = -2x \wedge 2 = 2x \wedge x \leq N)) \equiv$$
$$wlp(x := 1, -2 = -2x \wedge 2 = 2x \wedge x \leq N) = -2 = -2 \wedge 2 = 2 \wedge 1 \leq N \equiv$$
$$\top \wedge \top \wedge 1 \leq N$$

$wlp(p, z = 2 * N) = 1 \leq N$

We have to show that $A \rightarrow wlp(p, B)$ which is $N \geq 1 \rightarrow 1 \leq N$ in our case.
We can omit verification conditions for the assignments $x := 1$, $y := -2$ and $z := 2$ because they all are evaluating to true per definition. Now we have to show that the following holds:

VC(while ... od, B)

$$VC(\textbf{while } x < N \textbf{do } x := x - 2y - 2z + 1; y := y - 2; z := z + 2; \textbf{od}, z = 2 * N) =$$
$$(I \wedge \neg b \rightarrow B) \wedge$$
$$(I \wedge b \rightarrow wlp(x := x - 2y - 2z + 1; y := y - 2; z := z + 2; , I) \wedge$$
$$VC(x := x - 2y - 2z + 1; y := y - 2; z := z + 2; , I)$$

I ∧ ¬b → B

$$I \wedge \neg b \rightarrow B \equiv$$
$$y = -2x \wedge z = 2x \wedge x \leq N \wedge \neg(x < N) \rightarrow z = 2 * N \equiv$$
$$y = -2x \wedge z = 2x \wedge x = N \rightarrow z = 2 * N \equiv$$
$y = -2x \wedge z = 2 * N \rightarrow z = 2 * N$ which is true in our case.

wlp(x := x − 2y − 2z + 1; y := y − 2; z := z + 2; , y = −2x ∧ z = 2x ∧ x ≤ N)

$$wlp(x := x - 2y - 2z + 1; y := y - 2; z := z + 2; , y = -2x \wedge z = 2x \wedge x \leq N) \equiv$$
$$wlp(x := x - 2y - 2z + 1, wlp(y := y - 2, wlp(z := z + 2, y = -2x \wedge z = 2x \wedge x \leq N))) \equiv$$
$$wlp(x := x - 2y - 2z + 1, wlp(y := y - 2, y = -2x \wedge z + 2 = 2x \wedge x \leq N)) \equiv$$
$$wlp(x := x - 2y - 2z + 1, y - 2 = -2x \wedge z + 2 = 2x \wedge x \leq N) \equiv$$
$$y - 2 = -2(x - 2y - 2z + 1) \wedge z + 2 = 2(x - 2y - 2z + 1) \wedge (x - 2y - 2z + 1) \leq N \equiv$$
$$y - 2 = -2x + 4y + 4z - 2 \wedge z + 2 = 2x - 4y - 4z + 2 \wedge (x - 2y - 2z + 1) \leq N \equiv$$
$$y = -2x + 4y + 4z \wedge z = 2x - 4y - 4z \wedge x - 2y - 2z + 1 \leq N \equiv$$
$$-3y = -2x + 4z \wedge 5z = 2x - 4y \wedge x - 2y - 2z + 1 \leq N$$

$(I \wedge b \rightarrow wlp(x := x - 2y - 2z + 1; y := y - 2; z := z + 2; , I)$

$$(I \wedge b \rightarrow wlp(x := x - 2y - 2z + 1; y := y - 2; z := z + 2; , I) \equiv$$
$$y = -2x \wedge z = 2x \wedge x \leq N \wedge x < N \rightarrow -3y = -2x + 4z \wedge 5z = 2x - 4y \wedge x - 2y - 2z + 1 \leq N \equiv$$
$$y = -2x \wedge z = 2x \wedge x \leq N \wedge x < N \rightarrow 6x = -2x + 8x \wedge 10x = 2x + 8x \wedge x + 4x - 4x + 1 \leq N \equiv$$
$$y = -2x \wedge z = 2x \wedge x \leq N \wedge x < N \rightarrow 6x = 6x \wedge 10x = 10x \wedge x + 1 \leq N \equiv$$
$$y = -2x \wedge z = 2x \wedge x \leq N \wedge x < N \rightarrow x = x \wedge x = x \wedge x < N \equiv$$
$$y = -2x \wedge z = 2x \wedge x \leq N \wedge x < N \rightarrow x < N \text{ which is true in our case.}$$

$VC(x := x - 2y - 2z + 1; y := y - 2; z := z + 2; , I)$

Verification conditions of program composition and assignments are always evaluating to true, so $VC(x := x - 2y - 2z + 1; y := y - 2; z := z + 2; , I) = \top$

We have shown that the following holds in our case: $VC(p, z = 2 * n) \wedge (N \geq 1 \rightarrow wlp(p, z = 2 * N)$
We can conclude that $\{N \geq 1\}\ p\ \{z = 2 * N\}$ is a valid partial correctness triple for program p.

3.b.)

For the Hoare Triple to be totally correct: x=2
(because then the post condition [x=2] is fulfilled AND the program terminates)

For the Hoare Triple to be partially, but not totally correct: abort
(because abort is not allowed in totally correct triples >> it would lead to an undefined state which does not ensure termination)

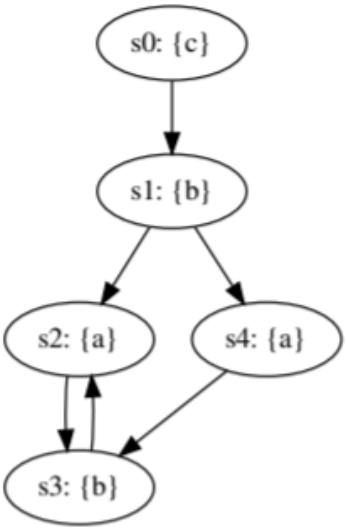
For the Hoare Triple to be not partially correct: x=0
(because this is a wrong state >> x=0 does not respect the post condition {x=1})

Block 4

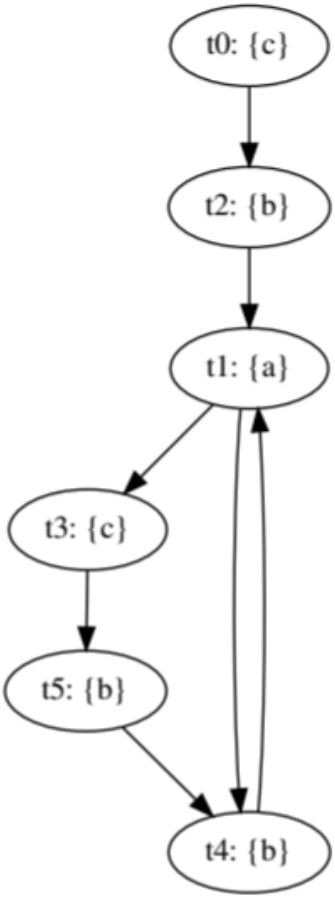
4.a.)

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



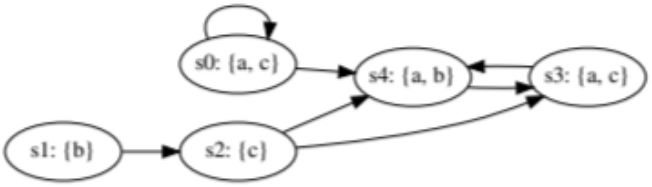
Kripke structure M_2:



(4 points)

4.a.) H = { (s0,t0), (s1,t2), (s2,t1), (s4,t1), (s3,t4) }

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{G}(a)$	☐	☐	☐	
$c\ \mathbf{U}\ b$	☐	☐	☐	
$\mathbf{AF}(a \wedge b)$	☐	☐	☐	
$\mathbf{EF}(a \wedge b)$	☐	☐	☐	
$\mathbf{X}(a)$	☐	☐	☐	

(5 points)

4.b.)

	CTL	LTL	CTL*	States
G(a)		X	X	s0,s3,s4
c U b		X	X	s1,s2,s3,s4
AF(a^b)	X		X	s1,s2,s3,s4
EF(a^b)	X		X	s0,s1,s2,s3,s4
X(a)		X	X	s0,s2,s3,s4

4.c.)

(c) **LTL tautologies**

Prove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π in M, or find a Kripke structure M and path π in M, for which the formula does not hold and justify your answer.

i. $\mathbf{X}(p\ \mathbf{U}\ q) \Leftrightarrow (\mathbf{X}p)\ \mathbf{U}\ (\mathbf{X}q)$

ii. $\mathbf{FGF}p \Leftrightarrow \mathbf{GFG}p$

(6 points)

4.c.i.) tautology - prove both directions

4c.ii.) no tautology - show a counterexample

Exam 24.7.2020 (fmi203.pdf)

Block 1

1.) Consider the following decision problem:

HALTING ON SAME INPUT (HSI)
INSTANCE: A tuple (Π_1, Π_2), where Π_1, Π_2 are programs that take a string as input.
QUESTION: Does there exist a string I such that both programs Π_1 and Π_2 halt on I?

(1) Provide the following many-one reductions:

- from the **HALTING** problem to **HSI**
- from the **HSI** problem to **EXISTS-HALTING** (see below)

EXISTS HALTING (EH)
INSTANCE: A program Π that takes a string as input.
QUESTION: Does there exist a string I such that Π halts on I?

Reduction from HALTING to HSI

Let (Π',I') be an arbitrary instance of HALTING.

```
String Π1(String S) {
  return Π'(I');          // Π' and I' are both hard-coded in Π1 and S is ignored
}
```

We define Π2 = Π1

To be proved: (Π',I') is a yes-instance of HALTING ⇔ (Π1, Π2) is a yes-instance of HSI

Proof:

"=>": assume (Π',I') is a yes-instance of HALTING:
Since Π' halts on I' we know that there exists at least one input I' for which Π' halts. Because we defined that Π2 = Π1 we know that Π1 and Π2 both halt on any arbitrary input S by construction. This means there are infinitely many S for which both halt.
Therefore (Π1, Π2) is a yes-instance of HSI.

"<=": assume (Π',I') is a no-instance of HALTING:
Since Π' never halts on I' we know that by construction of Π1 and Π2 that both never halt on any input S as well.
Therefore (Π1, Π2) is a no-instance of HSI.

(this reduction is similar to the reduction from HALTING to REACHABLE-CODE)

Reduction from HSI to EXISTS-HALTING
(Let (Π',I') be an arbitrary instance of HALTING.)

```
Boolean Π(String S) {
  Π1(S);
  Π2(S);
  return true;
}
```

To be proved: (Π1, Π2) is a yes-instance of HSI ⇔ (Π) is a yes-instance of EXISTS-HALTING
Proof:
"=>": assume (Π1, Π2) is a yes-instance of HSI:
Since we know that there does indeed exist an input I such that both Π1 and Π2 halt on I, also Π halts on I by construction of Π above.
Therefore there exists an input for which Π halts.
Therefore (Π) is a yes-instance of EXISTS-HALTING.
"<=": assume (Π1, Π2) is a no-instance of HSI:
Since there does not exist any input I such that both Π1 and Π2 halt on I we can conclude by construction of Π above that either Π1 or Π2 runs forever for any input S. Therefore there does not exist any input I such that Π halts.
Therefore (Π) is a no-instance of EXISTS-HALTING.

(I am not entirely sure if my solution is either 100% correct or the shortest one possible)

(2) Given that **HALTING** is undecidable and **EXISTS-HALTING** is semi-decidable, what can be concluded for **HSI** given the two reductions from (1)? **(15 points)**

Because of the reduction from HALTING to HSI we can conclude that HSI is undecidable. Because of the reduction from HSI to EXISTS-HALTING we can even further conclude that HSI must indeed be semi-decidable as well. EXISTS-HALTING being semi-decidable means that there exists a semi-decision procedure for EXISTS-HALTING. Together with the proposed reduction it implies that **HSI is semi-decidable**.
This implies that HSI is a semi-decidable problem.

Alternative Solution

1.)

(1) We provide a many-to-one reduction from HALTING to HSI. Suppose we have an instance (Π, I) of HALTING. We construct an instance (Π_1, Π_2) of HSI by setting $\Pi_1 = \Pi_2 = \Pi$.

$\Rightarrow$ Suppose we have an arbitrary positive instance (Π, I) of HALTING. That means program Π halts on input I. In this case (Π_1, Π_2) is also a positive instance, because there exist a string I such that both programs Π_1 and Π_2 halt on I. In this special case they halt at least on input I.

$\Leftarrow$ Suppose is an arbitrary negative instance of (Π_1, Π_2) of HSI. That means there exists no input I so that Π_1 and Π_2 halt on input I. Due to the fact that we set $\Pi_1 = \Pi_2 = \Pi$, this means there exists no input I on which Π halts $\rightarrow$ It is a negative instance of HALTING either.

We provide a many-to-one reduction from HSI to EH. Suppose we have an instance (Π_1, Π_2) of HSI. We construct an instance (Π) of EH by setting $\Pi = \Pi_1$.

$\Rightarrow$ Suppose we have an arbitrary positive instance (Π_1, Π_2) of HSI. That means there exists a string I such that both programs Π_1 and Π_2 halt on I. We set $\Pi = \Pi_1$ so there exists a string I such that Π halts on I. Due to the fact that in our assumption there exists a string I so that both programs Π_1 and Π_2 halt on I, also Π_1 halts on this I. We set $\Pi = \Pi_1$ so also Π halts on this I.

$\Leftarrow$ Suppose we have an arbitrary negative instance (Π) of EH. That means there doesn't exists a string I such that program Π halts on I. Due to the fact we set $\Pi_1 = \Pi_2 = \Pi$ it is also a negative instance of HSI.

(2)

- We provided a one-to-many reduction from HALTING to HSI. HALTING is undecidable so alse HSI is undecidable.
- We provided a one-to-many reduction from HSI to EXISTS-HALTING, so each problem instance of HSI could be solved by reducing this instance to an instance of EXISTS-HALTING. Given that EXISTS-HALTING is semi-decidable also HSI must be semi-decidable.

Block 2

2.) (a) Recall from the lecture that arrays are represented functionally in $\mathcal{T}_A$. For instance, $write(a, i, e)$ is denoted by $a\langle i \lhd e\rangle$. Similarly, $read(a, k)$ is denoted by $a[k]$. Consider

$$\varphi: \quad \big(a\langle i \lhd e\rangle\langle j \lhd f\rangle[k] \doteq g \wedge j \neq k \wedge (i \doteq k \rightarrow j \doteq k)\big) \rightarrow a[k] \doteq g.$$

If φ is $\mathcal{T}_A$-valid then provide a proof using the semantic argument method from the lecture. If φ is not $\mathcal{T}_A$-valid then provide a counter-example. Besides the equality axioms reflexivity, symmetry and transitivity, you have the following ones for arrays.

i. $\forall a, i, j \ \big(i \doteq j \rightarrow a[i] \doteq a[j]\big)$ (array congruence)

ii. $\forall a, v, i, j \ \big(i \doteq j \rightarrow a\langle i \lhd v\rangle[j] \doteq v\big)$ (read-over-write 1)

iii. $\forall a, v, i, j \ \big(i \neq j \rightarrow a\langle i \lhd v\rangle[j] \doteq a[j]\big)$ (read-over-write 2)

Please be precise. In a proof indicate exactly why proof lines follow from some other(s) and name the used rule. If you use derived rules you have to prove them.

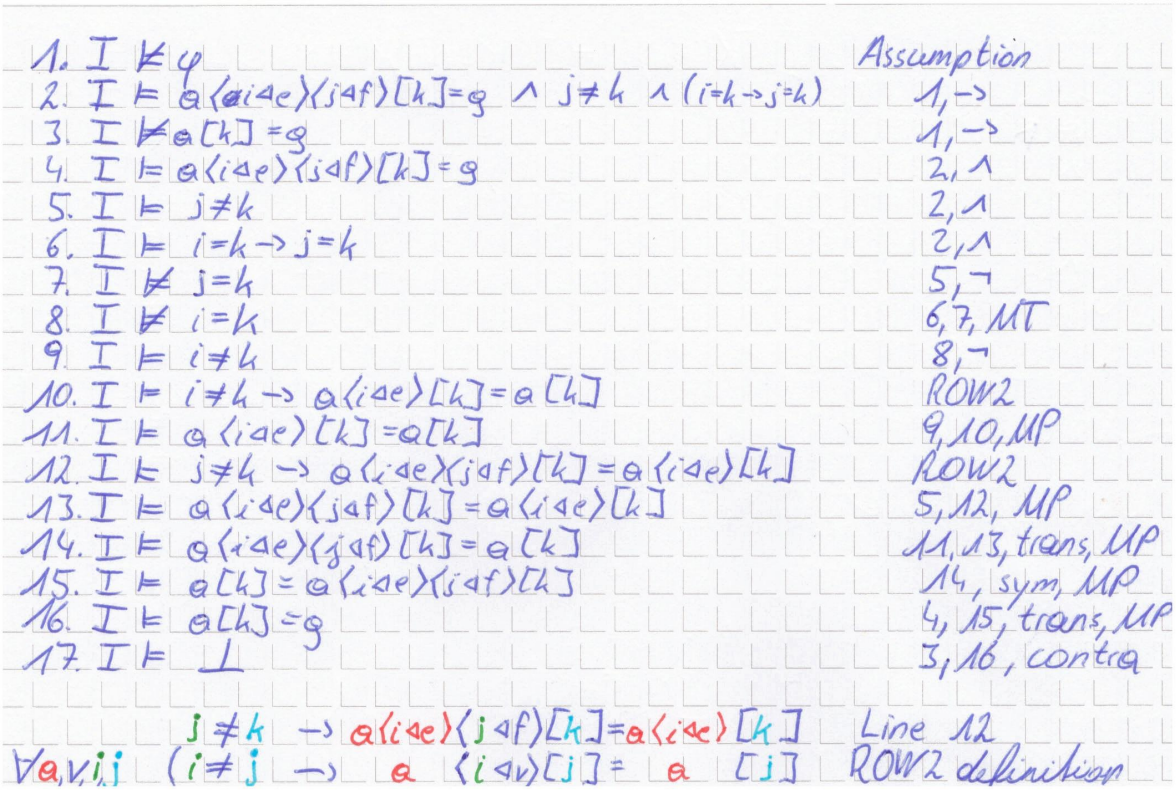
(12 points)

Prove by contradiction, that means LHS must hold and RHS must NOT hold.

1 I ⊨ phi
2 I ⊦ a(i<e)(j<f) [k] = g & j != k & (i=k ⇒ j=k)
3 I ⊨ a[k] = g
4 I ⊦ a(i<e)(j<f) [k] = g | 2, semantic of conjunction
5 I ⊦ j != k | 2, semantic of conjunction
6 I ⊦ (i=k ⇒ j=k) | 2, semantic of conjunction
7 I ⊦ i != k | 5,6 We know that j != k from 5 so i != k
| must hold for the entailment to hold and so
| also for LHS as a whole to hold (Modus
| Tollens)
8 I ⊦ a(i<e) (j<f) [k] = a(i<e)[k] |
5, 4, read over write 2 (Modus Ponens)
9 I ⊦ a(i<e) [k] = a[k] | 7, 8 read over write 2
10 I ⊦ a(i<e)(j<f) [k] = a[k] | 5,7,8,9 We know that i!=k and j!=k so it is
| trivial that any write access to i and j will not
| affect k therefore a(i<e)(j<f) [k] = a[k]
11 I ⊦ a[k] = g | 10,4 transitivity
12 ⊥ | 11,3 contradiction

Like the solution from before, but more precise I think

I tried to make more precise what was discussed for the previous solution. Especially the part marked red is interesting. Happy about feedback :)



Note that Modus Ponens (MP) has been mentioned in the exercise sheet as well as the PDF about the semantic argument method. So using it should be OK.
Note 2: Prof. Egly used ROW2 similarly (without referencing a line) in the sample solutions for the exercise sheet. As far as I see it, referencing lines becomes necessary when applying MP only:

1.	$\mathcal{I} \not\models a\langle i \lhd e\rangle[j] \doteq e \rightarrow i \doteq j \vee a[j] \doteq e$	assumption
2.	$\mathcal{I} \models a\langle i \lhd e\rangle[j] \doteq e$	1., $\rightarrow$
3.	$\mathcal{I} \not\models i \doteq j \vee a[j] \doteq e$	1., $\rightarrow$
4.	$\mathcal{I} \not\models i \doteq j$	3., $\vee$
5.	$\mathcal{I} \not\models a[j] \doteq e$	3., $\vee$
6.1	$\mathcal{I} \not\models i \neq j$	ROW2, $\rightarrow$
6.2	$\mathcal{I} \models i \doteq j$	6.1., $\neg$
6.3	$\mathcal{I} \models \bot$	4, 6.2., contradiction
7.1	$\mathcal{I} \models a\langle i \lhd e\rangle[j] \doteq a[j]$	ROW2, $\rightarrow$
7.2	$\mathcal{I} \models a[j] \doteq a\langle i \lhd e\rangle[j]$	7.1, symm., modus ponens
7.3	$\mathcal{I} \models a[j] \doteq e$	7.2, 2., trans., modus ponens
7.4	$\mathcal{I} \models \bot$	5., 7.3, contradiction

Some definitions

Definition

Given a theory $\mathcal{T} = (\Sigma, \mathcal{A})$.

1. A $\mathcal{T}$-interpretation $\mathcal{I}$ is a Σ-structure which satisfies $\mathcal{T}$'s axioms, i.e.,
$$\mathcal{I} \models \varphi \qquad \text{for all } \varphi \in \mathcal{A}$$
2. A Σ-formula φ is valid in the theory $\mathcal{T}$, or $\mathcal{T}$-valid, if every $\mathcal{T}$-interpretation satisfies φ. Notation: $\mathcal{T} \models \varphi$
3. A Σ-formula φ is satisfiable in the theory $\mathcal{T}$, or $\mathcal{T}$-satisfiable, if some $\mathcal{T}$-interpretation satisfies φ.

Definition

A theory $\mathcal{T} = (\Sigma, \mathcal{A})$ is

1. complete, if for every closed Σ-formula φ, $\mathcal{T} \models \varphi$ or $\mathcal{T} \models \neg\varphi$;
2. consistent, if there is at least one $\mathcal{T}$-interpretation;
3. decidable, if $\mathcal{T} \models \varphi$ is decidable for every Σ-formula φ.

Formulas φ_1 and φ_2 are equivalent in $\mathcal{T}$ or $\mathcal{T}$-equivalent if $\mathcal{T} \models \varphi_1 \leftrightarrow \varphi_2$, i.e., $\mathcal{I} \models \varphi_1$ iff $\mathcal{I} \models \varphi_2$ holds for all $\mathcal{T}$-interpretations $\mathcal{I}$.

Example of a complete theory: Presburger arithmetic

Example of an incomplete theory: Group theory

Block 3

3.) Let p be the following IMP program:

$$\begin{array}{l}\textbf{while } y < 5 \textbf{ do}\\ \quad z := z + 4 * x + 6;\\ \quad x := x + 2;\\ \quad y := y + 1\\ \textbf{od}\end{array}$$

where x, y, z are program variables. For each Hoare triple below, prove/disprove its total correctness. If the Hoare triple is correct, prove its total correctness by providing a formal proof. If the Hoare triple is not correct, provide a counterexample and justify your answer.

(3a) Hoare triple: $[z = 2 \wedge x = 1 \wedge y = 1]\ p\ [z = 90]$.

(3b) Hoare triple: $[z = 2 \wedge x = 1 \wedge y = 5]\ p\ [z = 2]$.

(15 points)

3.a.)

y[n] = y[0]+n => n =y-1
x[n] = x[0]+2n => n =(x-1)/2 = y-1 => x=2y-1
bc
z[n] = z[0] + 4*x[0] + 4* sum(x[j], j=1 to n-1) + 6n
z[n] = 2 + 4*1 +4r + 6n
z[n] = 6 + 4*((n-1) + 2 (n^2-n)/2) +6n
z[n] = 6 + 4n - 4 + $4n^2$ - 4n + 6n
Z[n] = $4n^2$ + 6n + 2

Substitution n=y-1
z = 4*(y-1)² + 6*(y-1) + 2 = $4y^2$ - 2y
y <= 5 is also a part of the invariant

Invariant:
I: I & y<=5
Variant:
t = 5-y

wp(p,B) = I
// p is the while loop in this case and the weakest precondition of it is I.
//This is different if there are assignments in the code before the loop, see fmi195 for example.

Step 1)
A => wp(p,B)
z=2 & x=1 & y=1 ⇒ z=$4y^2$-2y & y <= 5
z=2 & x=1 & y=1 ⇒ 2=4-2 & 1 <= 5

z=2 & x=1 & y=1 ⇒ 2=2 & 1 <= 5 Valid!

Step 2)
VC(while do … od, B)

1)
I & -b ⇒ B
z=4y²-2y & y<=5 & y>=5 ⇒ z=90
((y<=5 & y>=5) ⇒ y=5)
z=4y²-2y & y=5 ⇒ z=90
z=4*25 - 2*5 & y=5 ⇒ z=90
z=90 & y=5 ⇒ z=90
Valid!

2)
I & b ⇒ t >= 0
z=4y²-2y & y<=5 & y<5 ⇒ 5-y >= 0
(y<=5 & y<5 ⇒ y<5)
z=4y²-2y & y<5 ⇒ 5-y >= 0
Valid! Because y<5 and so 5-y >= 0

wp(I & t < t0) = wp(z=4y²-2y & y<=5 & 5-y<t0)
= z+4x+6 = 4*(y+1)² - 2*(y+1) & y+1<=5 & 5-(y+1)<t0
= z+4x+6 = 4y²+8y+4-2y-2 & y<=4 & 4-y<t0
= z+4x = 4y²+6y-4 & y<=4 & 4-y<t0
(remember x=2y-1)
= z + 8y - 4 = 4y²+6y-4 & y<=4 & 4-y<t0
= z=4y²-2y & y<=4 & 4-y<t0

3)
I & b & t=t0 ⇒ wp(p,I & t<t0)
z=4y²-2y & y<=5 & y<5 & t0=5-y ⇒ 4y²-2yz= & y<=4 & 4-y<t0
z=4y²-2y & y<5 & t0=5-y ⇒ z=4y²-2y & y<5 & 4-y<5-y
Valid!

4)
VC(p,I & t<t0) = true
Valid! Because p is the loop body which only consists of assignments and VC(*assignment*,B) is always true.

We proved total correctness.

---

t3.b.)

The while loop is skipped, therefore it is **totally correct** because the precondition is also the postcondition at the same time. I think it is trivial, still we can prove it via hoare calculus.

(not sure if correct)

[A] skip [A]
_____ rule for skip
[A] skip [B]
_____
[A] p [B]

Another approach, using operational semantics:

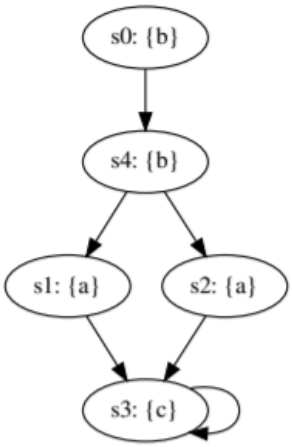
By definition of total correctness, the triple [z=2 & x=1 & y=5] p [z=2] is valid only if for all states σ with σ |= z=2 & x=1 & y=5, there exists a state σ' such that <p, σ> --> σ' and σ' |= z=2. Hence, let σ be an arbitrary state that satisfies z=2 & x=1 & y=5, i.e. σ |= z=2 & x=1 & y=5. Since <”y < 5”, σ> --> false, by operational semantics for while loops <p, σ> --> σ. Thus there exists a state σ' s.t. <p, σ> --> σ', i.e. σ'=σ. As σ |= z=2 (by semantics of assertions), the given triple is valid w.r.t. total correctness.

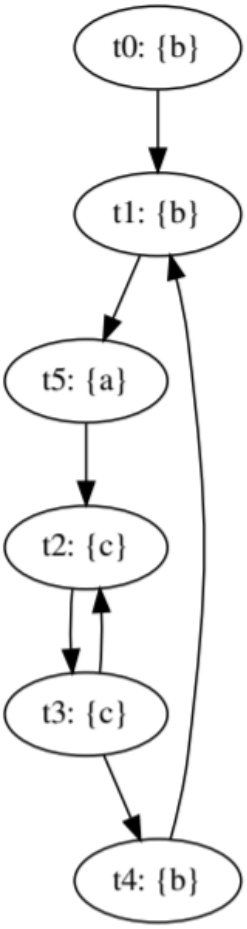
Block 4

a)

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1: **Kripke structure M_2:**



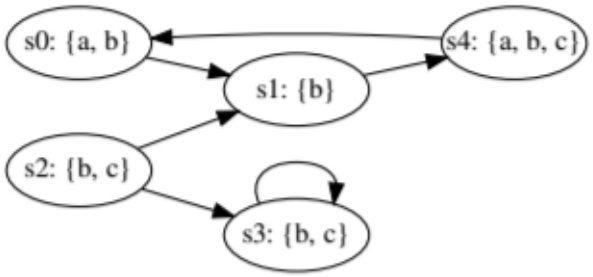


(4 points)

H = {(s0, t0), (s4,t1), (s1,t5), (s2,t5), (s3,t2), (s3,t3)}

b)

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{G}(c)$	☐	☐	☐	
$((a)\ \mathbf{U}\ (a))$	☐	☐	☐	
$\mathbf{A}[(a \wedge c)\ \mathbf{U}\ (c)]$	☐	☐	☐	
$\mathbf{EG}(a \wedge c)$	☐	☐	☐	
$\mathbf{E}[(b)\ \mathbf{U}\ (c)]$	☐	☐	☐	

(5 points)

	CTL	LTL	CTL*	States
G(c)		X	X	s3
a U a		X	X	s4,s0
A[(a & c) U c]	X		X	s3,s2,s4
EG(a & c)	X		X	-
E[b U c]	X		X	s2,s3,s4,s0,s1

c)

(c) **LTL tautologies**

Prove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π in M, or find a Kripke structure M and path π in M, for which the formula does not hold and justify your answer.

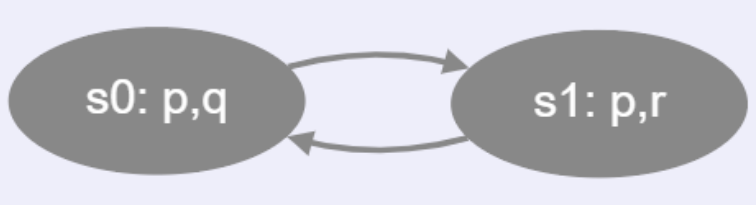
i. $\mathbf{GFG}p \Leftrightarrow \mathbf{FG}p$
ii. $p\ \mathbf{U}\ (q \wedge r) \Leftrightarrow (p\ \mathbf{U}\ q) \wedge (p\ \mathbf{U}\ r)$
iii. $p\ \mathbf{U}\ (q \vee r) \Leftrightarrow (p\ \mathbf{U}\ q) \vee (p\ \mathbf{U}\ r)$

(6 points)

i) Tautology
By student:
FGp means there is some state so that all following states satisfy p (p stabilizes).
=> Use equivalence $G\phi \equiv \phi \wedge X(G\phi)$ with ϕ=FGp: $GFGp \equiv FGp \wedge X(GFGp)$ and the conjunction implies the single conjunct FGp.

<= Assume $M,\pi \models FGp$ for all paths π. Suppose for a contradiction $M,\pi \models \neg GFGp$. Then by duality of F and G holds $M,\pi \models F\neg FGp$. So by definition there is k>=0: $M,\pi^k \models \neg FGp$. But as π^k is some path in M, this contradicts the initial assumption. Hence, it holds $M,\pi \models GFGp$.

ii)
Counterexample:
RHS holds but LHS doesn't hold



iii) Tautology
By student, using the same equivalences as i):
$pU(q \vee r) = (q \vee r) \vee (p \wedge X(pU(q \vee r))) = q \vee r \vee p \wedge X(pU(q \vee r))$
(note that the previous temporal property is also useful for part b) is equivalent to
$(pUq) \vee (pUr) = q \vee p \wedge X(pUq) \vee r \vee p \wedge X(pUr)$
$= q \vee r \vee p \wedge X(pUq) \vee p \wedge X(pUr)$
$= q \vee r \vee p \wedge (X(pUq) \vee X(pUr))$
$= q \vee r \vee p \wedge X(pUq \vee pUr)$
$= q \vee r \vee p \wedge X(pU(q \vee r))$

Exam 9.6.2020 (fmi202.pdf)

Block 1

1.) Consider the following decision problem:

> **HALTING AFTER LINE-REMOVAL (HALR)**
>
> INSTANCE: A tuple (Π, I), where Π is a program that takes a string as input; I a string.
>
> QUESTION: Does there exist a line of code in Π, such that when the line is removed from Π, the resulting program (a) is syntactically correct and (b) halts on I?

(1) By providing a suitable many-one reduction from the **HALTING** problem, prove that **HALR** is undecidable.

(2) Is **HALR** semi-decidable? Explain your answer. **(15 points)**

Solved by student (feedback welcome)

a) We construct a Π' from an arbitrary instance (Π,I) by setting I'=I and Π' like the following:

```
String Π'(String s) {
    while(true){};
    return Π(s); // Π is hardcoded. We could also hardcode I here, but we
just defined I'=I above.
}
```

To prove the reduction is correct, we have to show: **(Π,I) is a positive instance of HALTING <=> (Π',I') is a positive instance of HALR**.

"=>" We assume (Π,I) is a positive instance of HALTING. By the construction of Π' only the line "while(true){}" can be removed so that Π' halts and is syntactically correct. Since we set I'=I, Π(I') terminates by assumption and therefore (Π',I') is a positive instance of HALR.

"<=" We assume (Π',I') is a positive instance of HALR. By the construction of Π', this means that the line "while(true){}" had been removed, or else Π' would not halt (and be syntactically correct). Since we know Π' halts on I' and Π' contains Π(I'), we know that Π(I') halts. And by having set I=I', we know that Π(I) halts i.e. (Π,I) is a positive instance of HALTING.

Alternative "<=" We assume (Π,I) is a negative instance of HALTING i.e. Π does not terminate on I. We have to remove "while(true)" in Π' to be syntactically correct (and halt). But since Π' contains Π(I) (via I'=I), which does not halt by assumption, Π' can't halt either -> (Π',I') is a negative instance of HALR.

b) Yes, HALR is semi-decidable.

We can construct a semi-decision procedure using an interpreter Π_{Int} (that returns true if the program terminates after the given amount of steps, false otherwise):

```
Boolean test(Π, I) {
  steps = 1;
  anyValid = false;
  while(true) {
    for line = 1 to line <= |Π| {
      Π' = removeLine(Π, line);
      if (validSyntax(Π')) {
        anyValid = true;
        if (Π_Int(Π', I, steps)) {
          return true;
        }
      }
    }
    if (!anyValid) {
      return false;
    }
    steps++;
  }
}
```

Here we use a kind of diagonalization argument so that we can test multiple lines to remove and don't run into an endless loop if the resulting program Π' does not terminate.

To summarize:

- If there is a line in Π so that its removal yields a syntactically correct Π', which eventually terminates, then "test" does find it and return true.
- If there does not exist a line so that its removal yields a syntactically correct Π', "test" returns false.
- If there exists a line (or more) that yields a syntactically correct Π' on its removal, but none of the resulting Π' terminate, then "test" runs forever.

This describes semi-decidability and therefore HALR is semi-decidable.

Block 2

2.) We consider a slightly restricted and simplified form M of the Ackermann-Péter function, which we discussed in the exercise part.

Algorithm 1: The function M

Input: x, y, two *positive* integers
Output: The computed positive integer value for x, y

```
1 if x == 1 then
2 |  return 2y;
3 else if y == 1 then
4 |  return x;
5 else return M(x − 1, M(x, y − 1));
```

(a) Let $\mathbb{N}$ denote the natural numbers *without* 0. Use well-founded induction to show

$$\forall x \forall y \left((x \in \mathbb{N} \wedge y \in \mathbb{N}) \rightarrow \mathrm{M}(x, y) \geq 2y\right).$$

(11 points)

Solved by a student. please feedback!

We know that : (NxN,⊑) is a well-founded with least element (1,1). Let P(x,y): M(x,y) terminates on inputs (x,y) ∈ NxN and returns a positive integer value.

The proof is by well-founded induction on (NxN,⊑).

Base case: The least element is (1,1). Then P(1,1) is true because M(1,1)=2 (line2).
Induction Hypothesis: we pick an arbitrary non-minimal element (x,y) and assume that P(x,y) is true for all (x',y') ⊏ (x,y).
Induction Step: We want to show that P(x,y) is true. Since x>=1 and y >=1, we have the following cases:

Case 1: x = 1. Then M(1,y) = 2y (line2) and P(x,y) is true because P(x,y) terminates.
Case 2: x≠ 1 and y =1. Then x ⊏ (x,y). By Induction Hypothesis P(x) is true and M(x,1) = x (line4). Thus P(x,1) is true.
Case 3: x≠ 1 and y≠ 1. Then (x,y-1) ⊏ (x,y) and for all z ∈ N, (x-1,z) ⊏ (x,y). For all z ∈ N, P(x-1,z) is true by IH. Therefore, M(x-1, M(x,y-1)) terminates and computes a positive integer value, thus P(x,y) is true.

We conclude that P(x,y) is true for all (x,y) ∈ N x N.

Alternative Solution

We know that $(\mathbb{N} \times \mathbb{N}, \sqsubseteq)$ is a well-founded and lexicographically ordered set with least element $(1, 1)$, since 0 is not included in this exercise.

Let $P(x, y)$ denote $M(x, y) > 2y$. The proof is by lexicographic well-founded induction on $(\mathbb{N} \times \mathbb{N}, \sqsubseteq)$.

1 Base case

The least element is $(1, 1)$. $P(1, 1)$ is true because $M(1, 1) = 2$. Since $x == 1$ the program returns $2y$ which is 2 in our case.

2 Induction hypothesis

We pick an arbitrary non-least element (x, y) and assume that $P(x', y')$ is true for all $(x', y') \sqsubset (x, y)$.

3 Induction step

We show that $P(x, y)$ is true. Distinguish the following three cases:

3.1 Case 1: $x = 1$

Since $M(1, y)$ returns $2y$ in line 2 it trivially holds that $M(1, y) \geq 2y$. $P(x, y)$ is true in this case.

3.2 Case 2: $x \neq 1 \wedge y = 1$

$M(x, 1)$ returns x in line 4. Since $x \neq 1$ and $x \in \mathbb{N}$ it holds that $x \geq 2$. It holds that $M(x, 1) \geq 2$, so $P(x, y) \geq 2y$ in this case, because $y = 1$ and $M(x, y) \geq 2$.

3.3 Case 3: $x \neq 1 \wedge y \neq 1$

Since $(\mathbb{N} \times \mathbb{N}, \sqsubseteq)$ is well-founded it holds that $(x, y - 1) \sqsubset (x, y)$ and for all $z \in \mathbb{N}$, $(x - 1, y) \sqsubset (x, y)$. Then $P(x - 1, y)$ is true due to the induction hypothesis, which means that $M(x - 1, y) \geq 2y$ and so it holds that $M(x, y) \geq 2y$. $P(x, y)$ is also true in this case.

We can conclude that $P(x, y)$ is true for all $(x, y) \in \mathbb{N} \times \mathbb{N}$.

(b) Suppose $\texttt{M}_\texttt{C}$ is an implementation of M in the `C` programming language with x and y of type unsigned integers of size 32 bit (i.e., of type `uint32_t`). Is

$$\mathrm{M}(x', y') = \texttt{M}_\texttt{C}(x', y')$$

true for all integers x', y' satisfying $1 \leq x', y' \leq$ `UINT32_MAX`, where `UINT32_MAX` is the largest value for a variable of type `uint32_t`?

If so, then prove this fact. Otherwise provide a counterexample with an exact explanation of what is computed and what is happening. **(4 points)**

(solved by Student, feedback welcome)

Counterexample:

let M(x,y) be +

$x,y \in \mathbb{N}$

x = 1

y = UINT32_MAX

M(x,y) = UINT32_MAX +1 (there can be no overflow)

Mc(x,y) = 1 (overflow)

M(x,y) != Mc(x,y)

(Another solution by Student, feedback welcome)

Counterexample:

let M(x,y) be +

$x,y \in \mathbb{N}$

x = 1

y = (UINT32_MAX/2) +1

M(x,y) = UINT32_MAX+1 (overflow)

Mc(x,y) = 0

M(x,y) != Mc(x,y)

Block 3

3.) Let p be the following IMP program:

$$\begin{array}{l}\textbf{while } y < 10 \textbf{ do}\\ \quad x := x + z + 1;\\ \quad z := z + 2;\\ \quad y := y + 1\\ \textbf{od}\end{array}$$

where x, y, z are program variables. For each Hoare triple below, prove/disprove its total correctness. If the Hoare triple is correct, prove its total correctness by providing a formal proof. If the Hoare triple is not correct, provide a counterexample.

(3a) Hoare triple: $[x = 0 \wedge y = 0 \wedge z = 0]\ p\ [x = 100]$.

(3b) Hoare triple: $[x = 0 \wedge y = 20 \wedge z = 0]\ p\ [x = 100]$.

(15 points)

Solved by student:

a)

Using invariant generation ap-proach (Or just look at the values for each iteration..probably easier in this example):

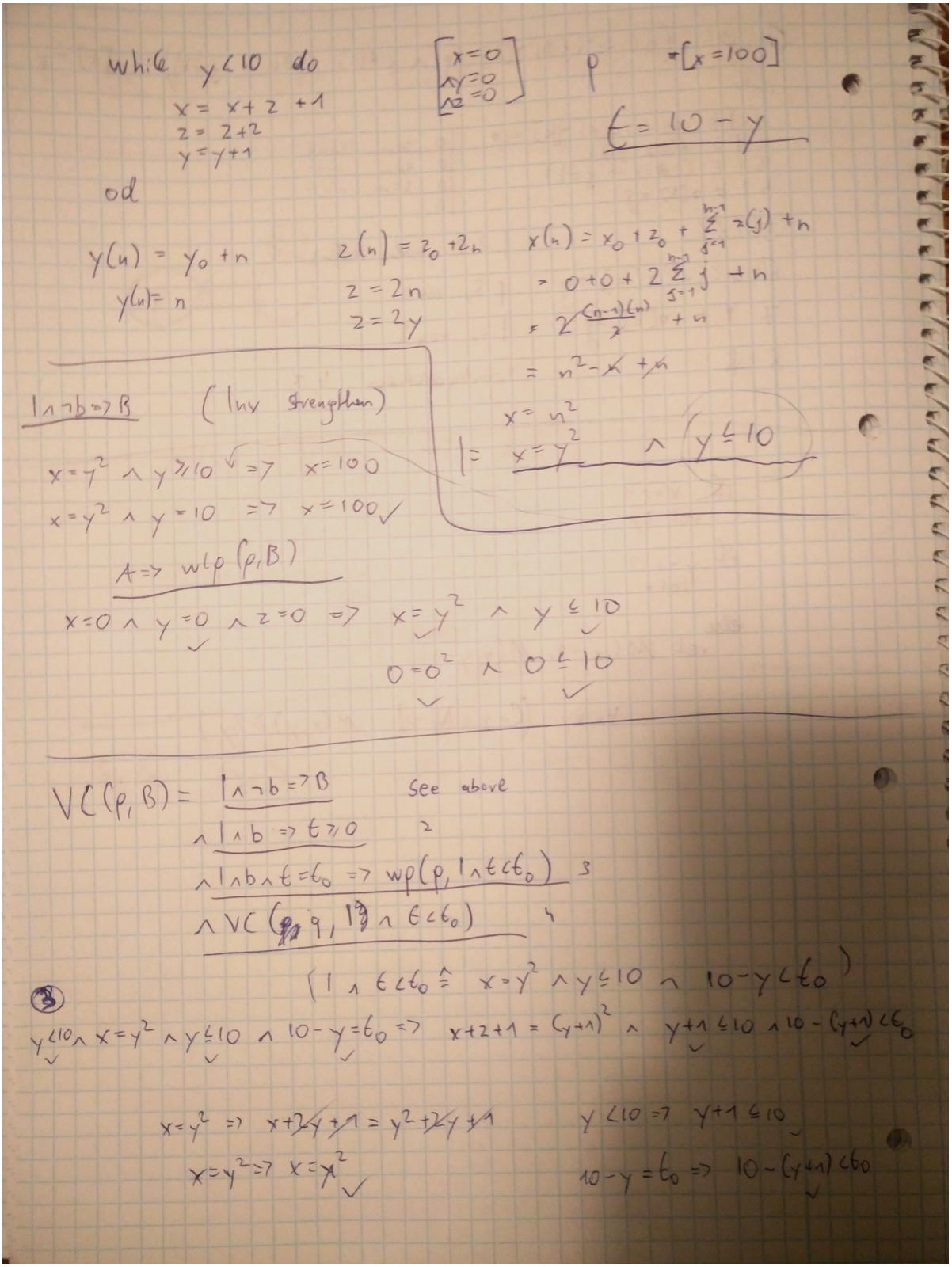
x[c+1] = x[c] + z[c] + 1;
z[c+1] = z[c] + 2;
y[c+1] = y[c] + 1;

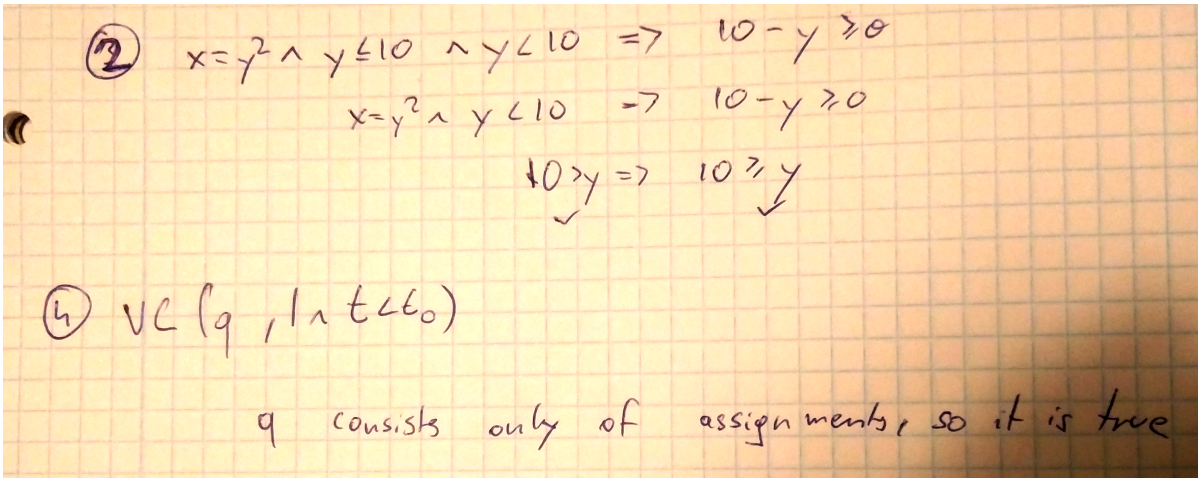
z[c] = z[0] + 2*c;
y[c] = y[0] + c;
x[c] = x[0] + z[0] + sum(z[j], j=1 to c-1) + c;
x[c] = 0 + 0 +
Variant:
10-y

Using wp and VC we can prove that the tripel is valid.

b)
Not valid! The precondition is actually already the counterexample, since setting σ(x)=0, σ(y)=20, σ(z)=0 causes the loop body to never be executed and x remains 20.

Second Solution (with full proof) - please correct me if i am wrong

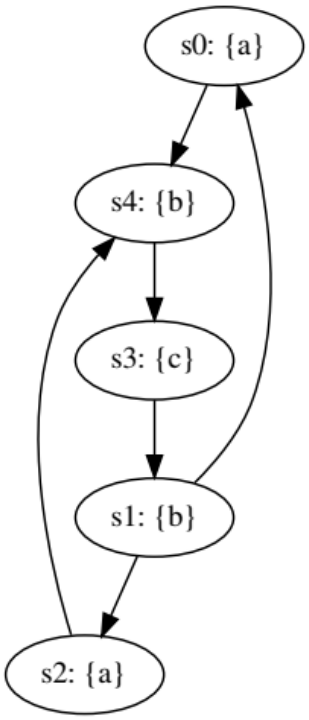



② $x=y^2 \wedge y \leq 10 \wedge y < 10 \Rightarrow 10-y \geq 0$
$x=y^2 \wedge y < 10 \rightarrow 10-y \geq 0$
$10 > y \Rightarrow 10 \geq y$
④ $VC(q, l \wedge t < t_0)$
q consists only of assignments, so it is true

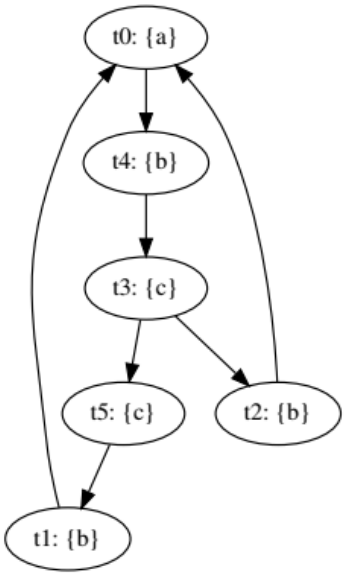
Block 4

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



Kripke structure M_2:



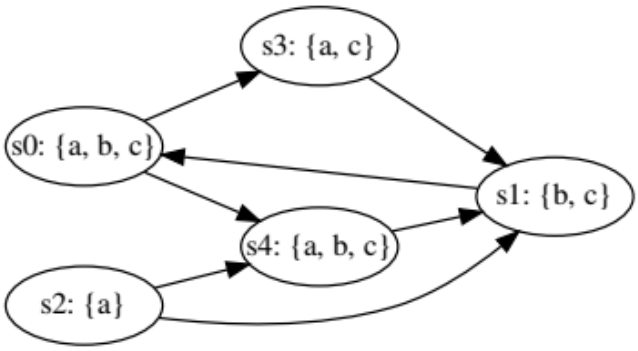
(5 points)

Solved by student:

H = {(s0, t0), (s4, t4), (s3, t3), (s1, t2), (s2, t0)} - Correct. Verified by exam correction. (5/5)

what about ~~H = {(s0, t0), (s4, t4), (s3, t3), (s1, t2), (s0, t0)}~~ ? —see comments

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{X}(a \wedge b)$	☐	☐	☐	
$\mathbf{AX}(b)$	☐	☐	☐	
$\mathbf{EG}(a)$	☐	☐	☐	
$\mathbf{EX}(a \wedge b)$	☐	☐	☐	
$\mathbf{E}[(a \wedge b \wedge c)\ \mathbf{U}\ (c)]$	☐	☐	☐	

(5 points)

Solved by student (correct. verified by exam correction):

	CTL	LTL	CTL*	States
X(a ∧ b)		X	X	s1
AX(b)	X		X	s1, s2, s3, s4
EG(a)	X		X	-
EX(a ∧ b)	X		X	s0, s1, s2
E[(a ∧ b ∧ c) U (c)]	X		X	s0, s1, s3, s4

(c) **LTL tautologies**

Prove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π in M, or find a Kripke structure M and path π in M, for which the formula does not hold and justify your answer.

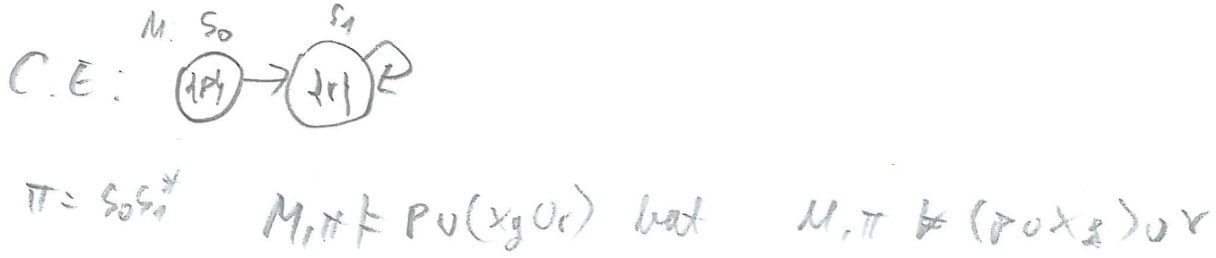
i. $(p\ \mathbf{U}\ (\mathbf{X}q))\ \mathbf{U}\ r \Leftrightarrow p\ \mathbf{U}\ ((\mathbf{X}q)\ \mathbf{U}\ r)$

ii. $((\mathbf{F}p) \Rightarrow q) \Rightarrow (p \Rightarrow \mathbf{F}q)$

(5 points)

i)

Counter example (correct. See exam correction - pencil):



f

(p U (Xq)) U r = false

P U ((Xq) U r) = true

ii)

Solved by student:

We look at an arbitrary structure M with state s and path π starting at s.

"=>" We assume M,π ⊨ ((Fp) => q)

It follows:

M,π ⊨ (¬(Fp) ∨ q)

M,π ⊨ (G¬p ∨ q)

This means either 1) for all i≥0 $M,\pi^i \vDash \neg p$ or 2) M,π ⊨ q.

Proof by contradiction: We assume the right side does not hold.

M,π ⊭ (p => Fq)

M,π ⊨ ¬(p => Fq)

M,π ⊨ (p ∧ ¬Fq)

M,π ⊨ (p ∧ G¬q)

This means it must hold that 3) $M,\pi \vDash p$ and 4) $M,\pi \vDash G\neg q$ (i.e. for all $j \geq 0$ $M,\pi^j \vDash \neg q$)

In case 1) holds, it contradicts with 3) and in case 2) holds, it contradicts with 4). Our assumption is therefore wrong and the right side must hold. **The implication is therefore valid.**

Exam 27.1.2020 (fmi201.pdf)

Block 1

1.) Consider the following decision problem:

REMOVE VERTEX 3-COL (RV3COL)

INSTANCE: An undirected graph $G = (V, E)$.

QUESTION: Does there exists a vertex $x \in V$ such that the removal of x from G yields a 3-colorable graph.
Formally, does there exist $x \in V$ such that the graph $(V \setminus \{x\}, E')$ with $E' = \{(u, v) \in E \mid u \neq x \text{ and } v \neq x\}$ is 3-colorable.

Show NP-hardness of **RV3COL** by providing a polynomial-time many-one reduction from the standard **3-COL** problem. Prove the correctness of your reduction.

Recall that **3-COL** is defined as follows:

3-COL

INSTANCE: An undirected graph $G = (V, E)$.

QUESTION: Does there exist a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

Many-one reduction from 3COL (G = (V,E)) to RV3COL (G' = (V', E'')):

V' = V u $\{v_1, v_2, v_3, v_4\}$
E' = E u $\{(v_1,v_2),(v_1,v_3),(v_1,v_4),(v_2,v_3),(v_2,v_4),(v_3,v_4)\}$

=> By adding the subgraph with the four vertices, we force the RV3COL algorithm to always remove one of the four added vertices. So if the remaining, original graph from the 3COL problem is a positive instance of 3COL the reduction results in a positive RV3COL instance.
<= Also if the original graph is a negative instance of the 3COL problem, the reduction results in a negative RV3COL instance. (proof by contraposition)

2. Option <= Let G' = (V', E') be a positive instance of RV3COL.
Then there exists a vx ∈ V' which when removed makes G = (V' \ {vx}, E' \ {(v1, v2) ∈ E' | v1 = vx v v2 = vx}) by definition a positive instance of 3COL (comments requested)

Alternative Solution

1.) Suppose $G = (V, E)$ is an arbitrary instance of 3-COL. We construct an instance $G^{'}(V^{'}, E^{'})$ as follows:

$V^{'} = V \cup \{v_{new}\}$
$E^{'} = E \cup \{[v_{new}, v_{new}]\}$

$\Rightarrow$ Suppose $G = (V, E)$ is a positive instance of 3-COL. So there exists a function $\varphi : V \to \{v_1, v_2, ..., v_n\}$ with $\varphi(v_i) \neq \varphi(v_j)$ for each $[v_i, v_j] \in E$. In $G^{'}$ we only added one additional vertex v_{new} with a self-loop $([v_{new}, v_{new}])$.
We define $\varphi^{'}$ as follows:
$\varphi^{'}(v_i^{'}) = \varphi(v_i)$
$\varphi^{'}(v_{new}) = 0$

We assumed $G = (V, E)$ is 3-colorable, so if we remove vertex v_{new} from $G^{'}$ the resulting graph is also 3-colorable.

$\Leftarrow$ Suppose $G^{'} = (V^{'}, E^{'})$ is a positive instance of RV3COL. So there exists a Graph $G_*^{'} = (V_*^{'}, E_*^{'})$ which is 3-colorable. Graph $G_*^{'}$ is defined as follows:

$V_*^{'} = V\ \{x | x \in V^{'}\}$
$E_*^{'} = \{(u, v) \in E | u \neq x \text{ and } v \neq x\}$

Due to the self loop we constructed in $E^{'} = E \cup \{[v_{new}, v_{new}]\}$ it is ensured that the removed vertex x must be the vertex v_{new}. Assume that the resulting instance $G = (V, E)$ is not 3-colorable. Then there exists $\varphi(u_i, u_j)$ for any $[u_i, u_j] \in E$ where $\varphi(u_i) = \varphi(u_j)$. Due to the construction of $\varphi^{'}$ there must be a an $[u_i^{'}, u_j^{'}] \in E^{'}$ so that $\varphi^{'}(u_i^{'}) = \varphi^{'}(u_j^{'})$. Since $G^{'}$ is remove vertex 3-colorable and $G_*^{'}$ is 3-colorable this would lead to a contradiction. $\Rightarrow G = (V, E)$ must also be 3-colorable.

Block 2

2.) In the exercise part, we considered the Ackermann-Péter function which is as follows.

Algorithm 1: The Ackermann-Péter function AP

Input: x, y, two non-negative integers
Output: The computed non-negative integer value for x, y

```
if x == 0 then
    return y + 1;
else if y == 0 then
    return AP(x − 1, 1);
else return AP(x − 1, AP(x, y − 1));
```

Show, using well-founded induction, that

$$\forall x \forall y \left((x \in \mathbb{N}_0 \wedge y \in \mathbb{N}_0) \rightarrow \text{AP}(x, y) > y\right)$$

Base case: x = 0 and y = 0
Induction hypothesis: $\forall$ x $\forall$ y (x $\in$ N $\wedge$ y $\in$ N) $\rightarrow$ AP(x, y) > y

Induction step: see slides ind-proofs, but use P(x,y) as “AP(x,y)>y for x and y from natural numbers”.tr

I think base case x=0 for $\forall$ y y $\in$ N is sufficient. 0 and y are N and AP(0, y) = y + 1 > y
Then assume x,y are N and AP(x,y) > y, show x+1,y are N and AP(x+1, y) > y:
Since x,y are N, so is x+1;
Case 1: y = 0 then AP(x+1, 0) = 2 which > 0
Case 2: y > 0 then AP(x+1, y) = AP(x , AP(x+1, y-1)); Since the output of ap is non-neg integer and assumption AP(x, y')> y' $\forall$ y' (here y' = AP(x+1, y-1)), but that only gives us AP(x+1,y)>y-1
You can prob. Argue that for y=0 you actually have AP > y+1 so that cancely the > y-1 to > y

Other student: Please give feedback
The real argument for the last part could be:
y' = AP(x+1, y-1) that means: y' > y-1 that means: y' >= y
AP(x+1, y) = AP(x, y') > y' according to the induction hyp

AP(x+1,y) > y' and since y' >= y we also know that AP(x+1,y) > y

Other Student: For the last step do the following
set z=AP(x,y-1) > y-1.
Then clearly z >=y (as z>y-1)
And thus AP(x-1,z)>z>=y.
Therefore AP(x,y)>y holds! (as AP(x,y)>=AP(x-1,z)>y)

Block 3

3.) (a) Let p be the following IMP program:

$$\begin{array}{l}\textbf{while } y < n \textbf{ do}\\ \quad x := x + 4 * y + 2;\\ \quad y := y + 1\\ \textbf{od}\end{array}$$

Prove the total correctness of the following Hoare triple:

$$[n = 10 \wedge x = 0 \wedge y = 0]\ p\ [x = 200].$$

(10 points)

Solved by student:

Invariants:
y≤n

To get the invariant concerning x, we use the invariant calculation from the lecture. I used c as the program counter, since n is already used by p:

x[c+1] = x[c] + 4*y[c] + 2
y[c+1] = y[c] + 1
y[c] = y[0] + c
y = c

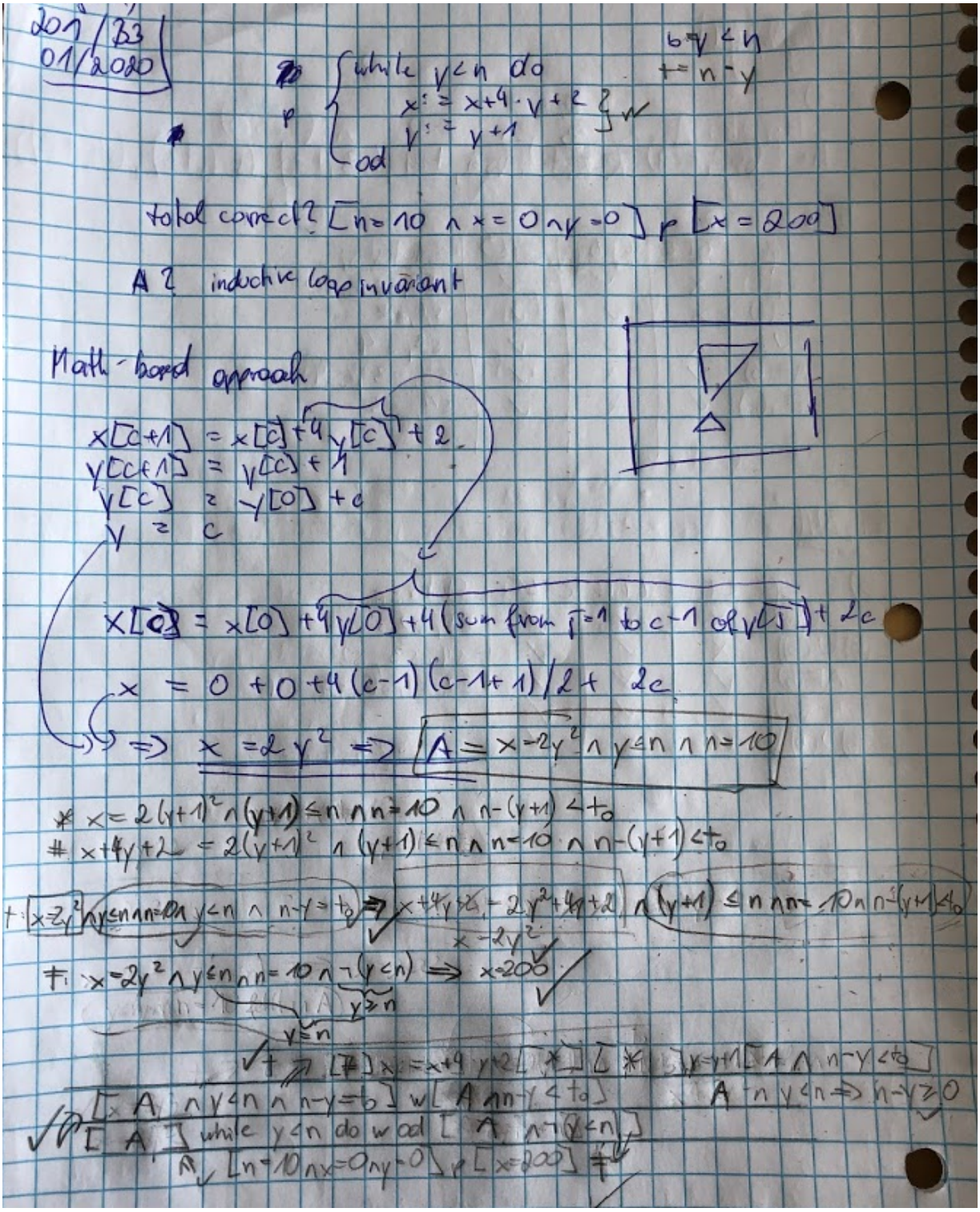
To get a closed form of x, we need to consider that y is updated by the loop as well, so we need to use a sum formula. We sum up to only c-1 since we use y before it's incremented.
x[c] = x[0] + 4*y[0] + 4*(sum from j=1 to c-1 of y[j]) + 2*c
x = 0 + 0 + 4*(c-1)*(c-1+1)/2 + 2*c
Simplifying and substituting y=c we get:
$x = 2*y^2$

Variant:
10 - y

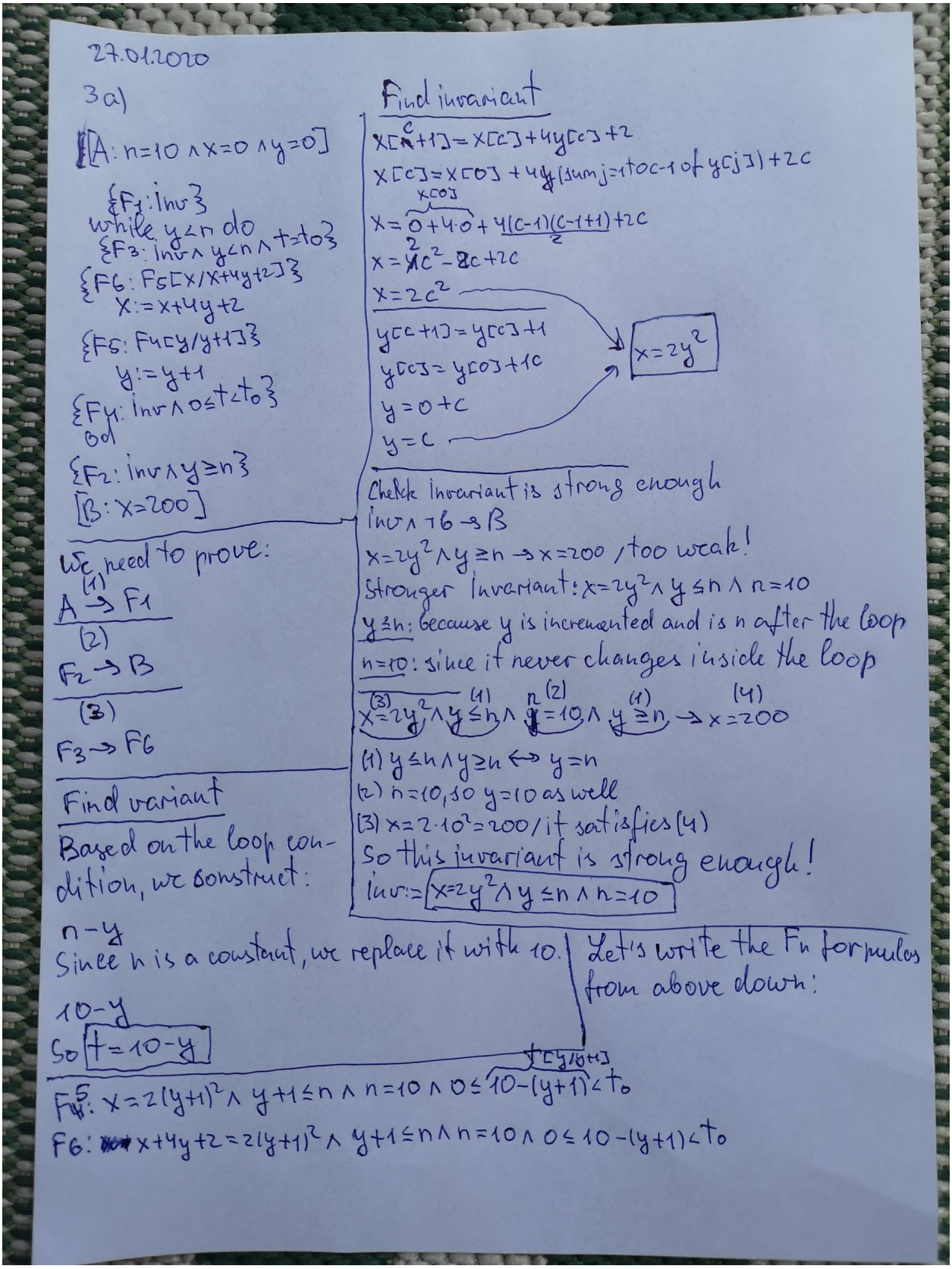
Proof total correctness using wp and VC as usual.

Addition from another student:
Complete inductive loop invariant A should be A = $x = 2*y^2 \land y \leq n \land n = 10$
And as variant b we used b = n - y

This way the last implication $A \land -b \Rightarrow x = 200$ checks out.



Another more detailed solution by another student (please provide feedback!)



27.01.2020 за page 2

Now we prove (1), (2), (3)

(1) $A \to F_1$

$n \overset{(1)}{=} 10 \wedge x = 0 \wedge y = 0 \to \underbrace{x = 2y^2}_{(3)} \wedge \underbrace{y \le n}_{(2)} \wedge \underbrace{n = 10}_{(1)}$

(1) is in premise

(2) $y \le n$ is $0 \le 10$ which is true

(3) is $0 = 2 \cdot 0^2$, so true

RHS all true ✓

---

(2) $F_2 \to B$

$x = 2y^2 \wedge y \le n \wedge n = 10 \wedge y \ge n \to x = 200$

We proved it before, when we where checking if Inv is strong enough.

---

(3) $F_3 \to F_6$

$x = 2y^2 \wedge y \le n \wedge n = 10 \wedge y < n \wedge 10 - y = t_0 \to \underbrace{x + 4y + 2 = 2(y+1)^2}_{(1)} \wedge \underbrace{y + 1 \le n}_{(2)} \wedge$

$\wedge \underbrace{n = 10}_{(3)} \wedge \underbrace{0 \le 10 - (y+1) < t_0}_{(4)}$

(1) can be simplified to. We have $x = 2y^2$ in premise, so: $2y^2 + 4y + 2 = 2(y+1)^2$

it is the same as $2(y+1)^2 = 2(y+1)^2$, which is true ✓

(2) in premise $y < n$, so $y + 1 \le n$ should hold ✓

(3) is in premise

(4) in premise $t_0 = 10 - y$, so let's replace t_0

$0 \le 10 - (y+1) < 10 - y$, then open $(y+1)$: $0 \le 10 - y - 1 < 10 - y$

"$10 - y - 1$" is clearly less than "$10 - y$"

I am not sure how to prove $0 \le 10 - y - 1$ which is actually $0 \le y \le 9$

Let's assume it is true :)

(b) Consider the following proof rule:

$$\frac{\{A\}\ x := a_1\ \{B\} \quad \{B \wedge x \leq a_2\}\ p; x := x + 1\ \{B\}}{\{A\}\ x := a_1; \textbf{while}\ x \leq a_2\ \textbf{do}\ p; x := x + 1\ \textbf{od}\ \{B \wedge x = a_2 + 1\}}$$

where x is a variable, a_1, a_2 are arithmetic expressions, p is a program, and A, B are assertions.

Is this proof rule sound? If yes, give a formal proof. Otherwise, give a counterexample.

(5 points)

Solved by student:

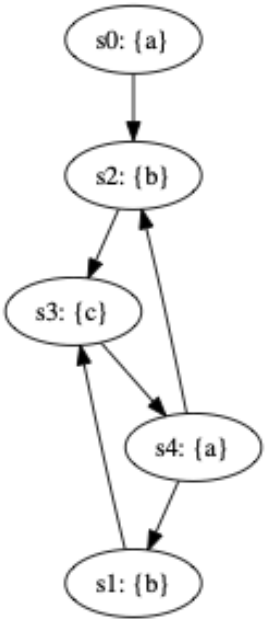
Not sound!
Counterexample:
a1 = 2; a2 = 0;
The loop is never executed and x stays at 2. This violates the Postcondition: 2 = 0 + 1
We conclude that the rule is not sound.

Note how this rule is the same as the "for" Rule in fmi196, with a modified Postcondition, making the rule not sound.

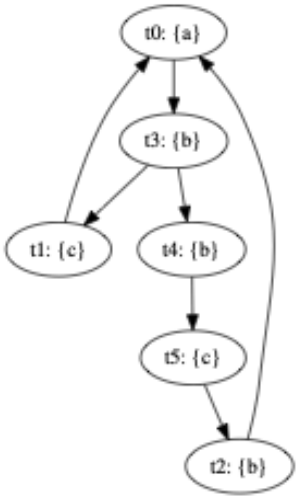
Block 4

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:

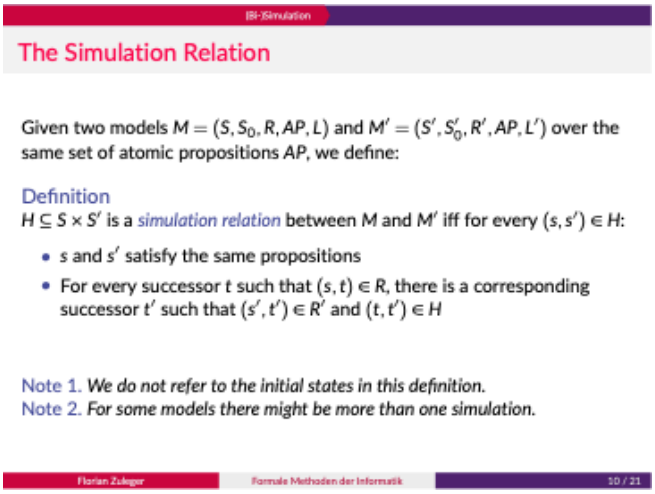


Kripke structure M_2:



(4 points)

H={(s0,t0),(s2,t3),(s3,t1),(s4,t0),(s1,t3),~~(s2,t3)~~}



IBI-Simulation

The Simulation Relation

Given two models $M = (S, S_0, R, AP, L)$ and $M' = (S', S_0', R', AP, L')$ over the same set of atomic propositions AP, we define:

Definition

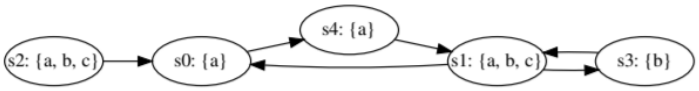
$H \subseteq S \times S'$ is a *simulation relation* between M and M' iff for every $(s, s') \in H$:

- s and s' satisfy the same propositions
- For every successor t such that $(s, t) \in R$, there is a corresponding successor t' such that $(s', t') \in R'$ and $(t, t') \in H$

Note 1. *We do not refer to the initial states in this definition.*
Note 2. *For some models there might be more than one simulation.*

Florian Zuleger | Formale Methoden der Informatik | 10 / 21

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and
ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

Should be correct based on a corrected exam by a prof.

	CTL	LTL	CTL*	States
G(a)		X	X	-
X(a^b^c)		X	X	s3,s4
AF(b)	X		X	s2,s0,s4,s1,s3
A[(a^c) U (c)]	X		X	s2,s1
E[(b^c) U (b)]	X		X	s2,s1,s3

(c) **LTL tautologies**

Prove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π in M, or find a Kripke structure M and path π in M, for which the formula does not hold and justify your answer.

i. $\mathbf{F}p \Leftrightarrow ((\mathbf{XG}\neg p) \Rightarrow p)$

ii. $(\mathbf{X}p)\ \mathbf{U}\ q \Leftrightarrow \mathbf{X}(p\ \mathbf{U}\ q)$

(6 points)

solved by student:

2021 4c)

i) We show $Fp \Leftrightarrow ((X G \neg p) \Rightarrow p)$

1) $Fp \Rightarrow ((X G \neg p) \Rightarrow p)$?

Assume arbitrary KS K and path $\pi = s_0 s_1 \ldots$ s.t. $K, \pi \models Fp$ ①

We know $\exists i \geq 0$ s.t. $K, \pi^i \models p$

Now assume $K, \pi \not\models (X G \neg p) \Rightarrow p$ towards a contradiction

Clearly: ① $K, \pi \models X G \neg p$ and ② $K, \pi \not\models p$

① $\Leftrightarrow \forall j \geq 1 \quad K, \pi^j \models \neg p$

② $\Leftrightarrow K, \pi^0 \models \neg p$

And thus $K, \pi \models \neg p$ which contradicts ①
(it leaves no choices for i)

2) $((X G \neg p) \Rightarrow p) \Rightarrow Fp$

Assume arbitrary KS K and Path $\pi = s_0 s_1 \ldots$ s.t. $K, \pi \models (X G \neg p) \Rightarrow p$ ①

There are 2 Cases:

1) $K, \pi \models X G \neg p$ and thus $\forall i \geq 1 \quad K, \pi^i \models \neg p$

also because of ① $K, \pi^0 \models p$

Obviously, it holds that $\exists i \geq 0$ s.t. $K, \pi^i \models p$ since $\exists j \geq 0$ s.t. $K, \pi^j \models Fp$

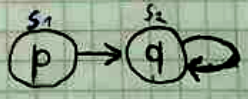
(namely we choose $j = 0$)

2) $K, \pi \not\models X G \neg p$ and thus $\neg(\forall i \geq 1 \quad K, \pi^i \models \neg p)$.

Therefore $\exists i \geq 1 \quad K, \pi^i \not\models \neg p$ From which we can follow that $\exists i \geq 0 \quad K, \pi^i \models p$ which means $K, \pi \models Fp$

ii) $(Xp) U q \Leftrightarrow X(p U q)$ is not a tautology!

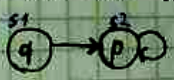
Counterexample:



in s_1:

- RHS holds
- LHS doesn't hold

[In fact, the 2 formulas are even uncomparable as this additional example shows:

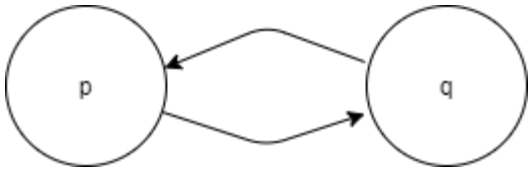


]

Other solution for ii:

Please provide feedback!

Not a tautology. Counterexample:8



LHS doesn’t hold on both nodes (only on the one with q)
RHS holds on both nodes

Exam 10.12.2019 (fmi196.pdf)

Block 1

INDEPENDENT DOMINATING SET (IDS)

INSTANCE: A directed graph $G = (V, E)$.

QUESTION: Does there exists a set $S \subseteq V$ of vertices, such that

(1) for each $(u, v) \in E$, $\{u, v\} \not\subseteq S$;

(2) for each $v \in V$ either $v \in S$ or there exists an $(u, v) \in E$, such that $u \in S$.

(a) The following function f provides a polynomial-time many-one reduction from **IDS** to **SAT**: for a directed graph $G = (V, E)$, let

$$f(G) = \bigwedge_{(u,v)\in E} (\neg x_u \vee \neg x_v) \wedge \bigwedge_{v\in V} (x_v \vee \bigvee_{(u,v)\in E} x_u).$$

It holds that G is a yes-instance of **IDS** $\iff$ $f(G)$ is a yes-instance of **SAT**. Prove the $\Longleftarrow$ direction of the claim.

(10 points)

(b) Given that **SAT** is NP-complete, what can be said about the complexity of **IDS** from the above reduction? NP-hardness of **IDS**, NP-membership of **IDS**, neither of them, or both (NP-completeness of **IDS**)

(5 points)

a)

Proof:

"$\Leftarrow$": Assume that $f(G)$ is a yes-instance of **SAT**:

Consider the propositional atoms x_u and x_v which represent the vertices of an edge. x is $true$ iff is is in S. To proof a CNF-formula true we have to proof all conjuncted subformulas true.

Now consider since $\bigwedge_{(u,v)\in E}(\neg x_u \vee \neg x_v)$ holds and tells us that only one of the vertices u, v is allowed to be in S at the same time, this implies $\forall (u, v) \in E, \{u, v\} \not\subseteq S$.

For the second part we know that $\bigwedge_{v\in V}(x_v \vee \bigvee_{(u,v)\in E} x_u)$ only evaluates to true if either $v \in S$ or any $u \in S$ which is connected by an edge to v. This implies
$\forall v \in V : v \in S \vee \exists (u, v) \in E : u \in S$.

Therefore we are done.

We did not show NP-hardness of IDS. This would require a reduction **from** SAT **to** IDS and not the other way around. However, what we proved is NP-membership. Informally we could say that with reducing IDS to SAT we showed that SAT is at least as hard as IDS. Since we know that SAT is NP-complete we know that IDS is at most as hard as any NP-complete problem. Therefore one knows that IDS is in NP (="NP-membership") since NP-completeness requires a problem to be NP-hard and in NP.

(need confirmation of my proof of Block 1)

Proof was discussed during the Q&A Session WS 2020/21:

($\Longleftarrow$): Suppose $f(G)$ is a yes-instance of **SAT**. Hence, there exists a truth-assignment T to the variables in $f(G)$ making this formula true. We construct a set $S = \{v \in V \mid T(x_v) = \mathit{true}\}$ and show that S satisfies (1) and (2).

(1): Towards a contradiction suppose (1) is violated by S, i.e. there exists $(u, v) \in E$ such that $u, v \in S$. By construction $T(x_u) = T(x_v) = \mathit{true}$. But then the conjunct $(\neg x_u \vee \neg x_v)$ in the first part of the formula $f(G)$ evalutes to false under T and thus T cannot make $f(G)$ true. Contradiction. Hence, S satisfies (1).

(2): Towards a contradiction suppose (2) is violated by S, i.e. there exists a vertex $v \in V$ such that $v \notin S$ and for all $(u, v) \in E$, $u \notin S$. By construction, we have $T(x_v) = \mathit{false}$ and, for all u with $(u, v) \in E$, $T(x_u) = \mathit{false}$. Since $f(G)$ has as a conjunct $x_v \vee \bigvee_{(u,v)\in E} x_u$, this subformula then evaluates to false under T and so does $f(G)$. Contradiction. Hence, S satisfies (2).

Our reduction shows NP-membership of **IDS**.

T
From the slides:

($\Longleftarrow$): Suppose $f(G)$ is a yes-instance of **SAT**. Hence, there exists a truth-assignment T to the variables in $f(G)$ making this formula true. We construct a set $S = \{v \in V \mid T(x_v) = \mathit{true}\}$ and show that S satisfies (1) and (2).

(1): Towards a contradiction suppose (1) is violated by S, i.e. there exists $(u, v) \in E$ such that $u, v \in S$. By construction $T(x_u) = T(x_v) = \mathit{true}$. But then the conjunct $(\neg x_u \vee \neg x_v)$ in the first part of the formula $f(G)$ evalutes to false under T and thus T cannot make $f(G)$ true. Contradiction. Hence, S satisfies (1).

(2): Towards a contradiction suppose (2) is violated by S, i.e. there exists a vertex $v \in V$ such that $v \notin S$ and for all $(u, v) \in E$, $u \notin S$. By construction, we have $T(x_v) = \mathit{false}$ and, for all u with $(u, v) \in E$, $T(x_u) = \mathit{false}$. Since $f(G)$ has as a conjunct $x_v \vee \bigvee_{(u,v)\in E} x_u$, this subformula then evaluates to false under T and so does $f(G)$. Contradiction. Hence, S satisfies (2).

Our reduction shows NP-membership of **IDS**.

Conclusion. If such a problem reduction from B to A exists, then problem A must be at least as hard as problem B. $\Longrightarrow$ This problem reduction proves the same hardness result for A as for B.

Hardness and Completeness

Remark

From now on, unless explicitly stated otherwise, we only consider polynomial-time many-one reductions in this lecture and write $\mathcal{P} \leq_{\mathrm{R}} \mathcal{P}'$ to denote that problem $\mathcal{P}$ can be reduced to $\mathcal{P}'$. Note that $\leq_{\mathrm{R}}$ orders problems with respect to their difficulty as it is reflexive and transitive.

Definition

Let C be a complexity class and let $\mathcal{P}$ be a problem.

$\mathcal{P}$ is called C-hard if any problem $\mathcal{P}' \in \mathrm{C}$ is reducible to $\mathcal{P}$.

$\mathcal{P}$ is called C-complete if $\mathcal{P}$ lies in C and $\mathcal{P}$ is C-hard, i.e.:

completeness = membership and hardness

Important Observation

If problem $\mathcal{P}$ is known as C-hard and $\mathcal{P} \leq_{\mathrm{R}} \mathcal{P}''$ then $\mathcal{P}''$ is also C-hard.

Block 2

(a) Let φ be the first-order formula

$$\forall x \forall y \left[\big(r(x,y) \to (p(x) \to p(y))\big) \wedge \big(r(x,y) \to (p(y) \to p(x))\big) \right] .$$

i. Is φ valid? If yes, present a proof. If no, give a counter-example and prove that it falsifies φ.

ii. Replace r in φ by $\doteq$ (equality) resulting in ψ. Is ψ E-valid? Argue formally!

(5 points)

Part a)

i) No, it is **not valid**. Counterexample: U={0,1}, I(r) = U², I(p) = {0}

Other suggestion for a counterexample:
U={0,1}, I(r(x,y)) = 1, I(p(x)) = 1, I(p(y)) = 0

ii) **Yes** it is. One can argue via interpretations (assume there is an interpretation that falsifies the formula,...) using the calculus for T_E validity or show that such an interpretation cannot exist using the semantic.

Part b)

(b) Show the following:

$$\varphi^{EUF} \text{ is satisfiable iff } FC^E \wedge \mathit{flat}^E \text{ is satisfiable.}$$

FC^E and flat^E are obtained from φ^{EUF} by Ackermann's reduction.
(Hint: FC^E is the same for φ^{EUF} and $\neg\varphi^{EUF}$.) **(10 points)**

Proof is given in hw1.pdf of the supplementary material.

Block 3

3.) (a) Let p be the following program:

$x := 3;$
$y := 1;$
while $y \geq N$ **do**
  $x := x - 4 * y + 2;$
  $y := y - 1$
od

Give a loop invariant for the **while** loop in p and prove the validity of the partial correctness triple $\{N < 0\}\ p\ \{x = 2 * N * N - 4 * N + 3\}$.

(10 points)

Solution by student:

Invariants:
(y≤1) … not needed
y ≥ N-1

To get the invariant concerning x, we use the invariant calculation from the lecture:

x[n+1] = x[n] - 4*y[n] + 2
y[n+1] = y[n] - 1

y[n] = y[0] - n
y = 1 - n
n = 1 - y

*For the closed form of x we have to consider that y is also updated by the loop, so we have to use a sum formula. Note that we use the sum up to n-1, as y is used before it's updated by the loop. We add an additional "-4*y[0]" to account for the initial value of y.*

x[n] = x[0] - 4*y[0] - 4*(sum from j=1 to n-1 of y[j]) + 2*n
~~x = 3 - 4 - 4*(n-1)*(n-1+1)/2 + 2n~~
Corrected sum calculation (by same student):
x[n] = x[0] - 4*y[0] - 4*((n-1)*1 - (n-1)*(n-1+1)/2) + 2n
x = 3 - 4 - 4*(n-1) + 2*(n^2-n) + 2n
This simplifies to:
x = 2*n² - 4*n + 3
Substituting n=1-y we get:

x = 2*(1-y)² - 4*(1-y) + 3
x = 2*y² + 1

We can now prove the partial correctness as usual with wlp and VC.

Other student, corrected closed form (Note: y[2..n-2] values will be negative, thus their sum is negative, so the sum has to be added not subtracted , thus + 4∑):

$$x[n] = x[0] - 4y[0] + 4\sum_{i=1}^{n-2} y[i] + 2n$$

This equation correctly can be simplified to $x = 2y^2 + 1$

Full solution from another student. Feedback please =)

10.12.2019 3a

A: $N<0$

$\{F8 = F[x/3]\}$
$x := 3;$
$\{F7 = F1[y/1]\}$
$y := 1;$
$\{F1: inv\}$
while $y \geq N$ do
$\{F6: inv \wedge y \geq N\}$
$\{F5: F4[x/x-4y+2]\}$
$x := x - 4y + 2;$
$\{F4: F3[y/y-1]\}$
$y := y - 1;$
$\{F3: inv\}$
od
$\{F2: inv \wedge y < N\}$
B: $x = 2N^2 - 4N + 3$

Find invariant

$y[c+1] = y[c] - 1$
$y[c] = y[0] - 1c$
$y = 1 - c$ $\quad 1 - c = y$
$-c = y - 1$
$c = -y + 1$
$c = 1 - y$

$x[c+1] = x[c] - 4y[c] + 2$
$x[c] = x[0] - 4\frac{(c-1)(c-1+1)}{2} + 2c$
$x = 3 - 4 - 2(c^2 - c) + 2c$
$x = -1 - 2c^2 + 2c + 2c$
$x = -2c^2 + 4c - 1$
$x = -2(1-y)^2 + 4(1-y) - 1$
$x = -2(1 - 2y + y^2) + 4 - 4y - 1$
$x = -2y^2 + 4y - 2 + 4 - 4y - 1$
$x = -2y^2 + 1$

We need to change -1 to +1 it has something to do with the sum of closed form

Let's strengthen the invariant right away.

$inv := x = 1 + 2y^2 \wedge y \geq N - 1 \wedge N < 0$

*this is $y \geq N-1$, because y should hold after the loop is done

We need to prove (1) $A \to F8$ (2) $F_2 \to B$ (3) $F6 \to F5$
So (2) and (3) are like from VC: F5 is wlp(Rloop, inv)

F4: $x = 1 + 2(y-1)^2 \wedge y - 1 \geq N - 1 \wedge N < 0$
F5: $x - 4y + 2 = 1 + 2(y-1)^2 \wedge y - 1 \geq N - 1 \wedge N < 0$
F3: $x = 1 + 2 \cdot 1^2 \wedge 1 \geq N - 1 \wedge N < 0$
F8: $3 = 1 + 2 \wedge 1 \geq N - 1 \wedge N < 0$

(1) $A \to F8$
$N < 0 \to 3 = 3 \wedge 1 \geq N - 1 \wedge N < 0$ ✓
* is in premise

(2) $F_2 \to B$
$x = 1 + 2y^2 \wedge y \geq N - 1 \wedge N < 0 \wedge y < N \to$
$\to x = 2N^2 - 4N + 3$

By taking a look at (1) and (2), we can see y actually is $N-1$. Let's say $N = -1$, so $y \geq -1 - 1$ and $y < -1 - 1$. This was just an example. Next insert x from premise to (4): $1 + 2y^2 = 2N^2 - 4N + 3$ We said before, $N - 1 = y$, so $N = y + 1$, Let's insert it as well: $1 + 2y^2 = 2(y+1)^2 - 4(y+1) + 3$ this simplifies to $1 + 2y^2 = 1 + 2y^2$, which holds ✓

$F6 \to F5$ (3)

$x = 1+2y^2 \wedge y \geq N-1 \wedge \underbrace{N<0}_{(1)} \wedge \underbrace{y \geq N}_{(2)} \to \underbrace{x - 4y + 2 = 1+2(y-1)^2}_{(3)} \wedge \underbrace{y-1 \geq N-1}_{(2)} \wedge \underbrace{N<0}$

(1) is in premise ✓

(2) both are the same, so (2) is in premise ✓

(3) * Replace x: $1+2y^2-4y+2 = 1+2(y-1)^2$

$2y^2-4y+3 = 1+2(y^2-2y+1)$

$2y^2-4y+3 = 1+2y^2-4y+2$ ✓

We've proven $VC(p, B)$ is true, $A \Rightarrow wlp(p, B)$ is true, so the Hoare triple is valid with respect to partial correctness.

(b) We add **for** loops with the following syntax to the IMP language.

$$\textbf{for } v := e_1 \textbf{ until } e_2 \textbf{ do } c \textbf{ od},$$

where v is a variable, e_1 and e_2 are arithmetic expressions and c is a program. The informal semantics of the **for** loop is as follows.

- v is initialized to e_1;
- in every loop iteration, c is executed and then v is incremented, i.e., $v := v + 1$;
- the loop terminates when $v > e_2$.

Stated differently, the above **for** loop is equivalent to

$$v := e_1; \textbf{while } v \leq e_2 \textbf{ do } c; v := v + 1 \textbf{ od}.$$

Prove the soundness of or provide a counterexample to the following proof rule.

$$\frac{\{P\}\ v := e_1\ \{I\} \quad \{I \wedge v \leq e_2\}\ c; v := v + 1\ \{I\}}{\{P\}\ \textbf{for } v := e_1 \textbf{ until } e_2 \textbf{ do } c \textbf{ od}\ \{I \wedge v > e_2\}}$$

(5 points)

Solved by student:

To prove a rule is sound, assume its premises are provable and show that the conclusion is provable.

We assume:
1) {P} v:=e1 {I}
2) {I ∧ v ≤ e2} c; v:=v+1 {I}

Applying the Rule of While on 2 we get:

3) {I} while v ≤ e2 do c; v:=v+1 {I ∧ v > e2}

Using Rule of Composition on 1 and 3:

{P} v:=e1; while v ≤ e2 do c; v:=v+1 {I ∧ v > e2}

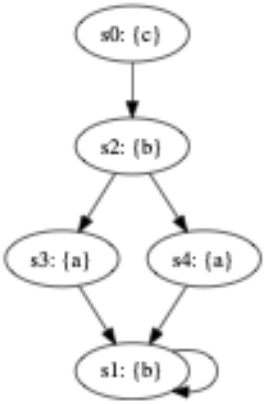
This is by definition equivalent to “for”, so we can derive:

{P} for v:=e1 until e2 do c od {I ∧ v > e2} … so we conclude that the rule is sound.

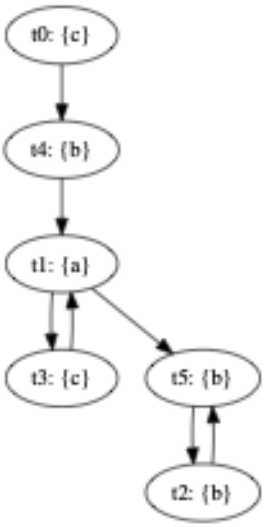
Block 4

4.) (a) Provide a simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



Kripke structure M_2:



(5 points)

~~H={(s0,t0),(s2,t4),(s3,t1),(s4,t1), (s1,t5)}~~

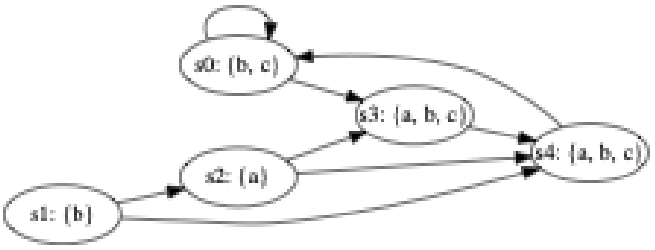
The for every successor t such that (s, t) belongs to R, there is a corresponding successor t' such that (s', t') belongs to R' and (t, t') belongs to H

Let s = s1 and s' = t5
Let t = s1. (s1, s1) belongs to R, there should be a successor t'=t2 such that (t5, t2) belongs to R' and (s1, t2) belongs to H.

H={(s0,t0),(s2,t4),(s3,t1),(s4,t1), (s1,t5), (s1, t2)}

Feedback?

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{F}(a)$	☐	☐	☐	
$\mathbf{X}(b \wedge c)$	☐	☐	☐	
$\mathbf{AG}(b \wedge c)$	☐	☐	☐	
$\mathbf{AX}(a)$	☐	☐	☐	
$\mathbf{E}[(b)\ \mathbf{U}\ (a)]$	☐	☐	☐	

(5 points)

	CTL	LTL	CTL*	States
F(a)		X	X	s3,s4,s2,s1
X(b^c)		X	X	s0,s4,s2,s3
AG(b^c)	X		X	s0,s3,s4
AX(a)	X		X	s1,s2,s3
E[(b) U (a)]	X		X	s2,s3,s4, s1,s0

(c) **LTL tautologies**

Prove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π in M, or find a Kripke structure M and path π in M, for which the formula does not hold and justify your answer.

i.

$$p \Rightarrow \top \ \mathbf{U} \ (\bot \ \mathbf{U} \ p)$$

ii.

$$q \wedge \mathbf{FG}p \Rightarrow q \ \mathbf{U} \ (\mathbf{G}p)$$

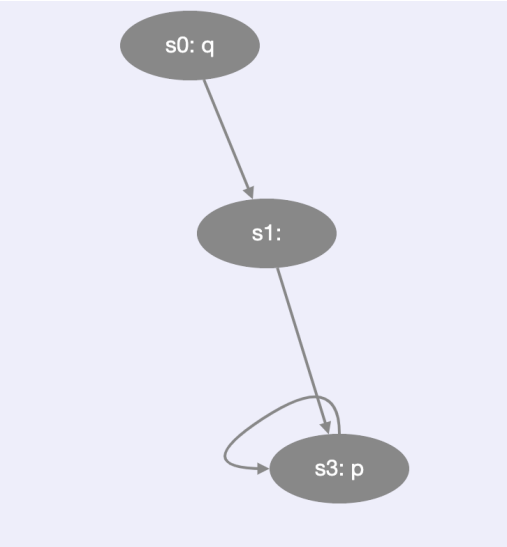
i.) **tautology**
p implies true U (false U p) this can be solved by following a few rules from the slides

true U (false U p) can be rewritten as F (false U p)
F (false U p), can be rewritten as F (p), as false will be always be false the validity only follows p

Now we have p implies Fp, which can be proven by contraposition

Assume M satisfies p, then the start state s0 has to satisfy p (s0 sat p) (0)

Now assume M does not satisfy true U (false U p) (or Fp), then for all paths pi from initial state s0 and pi is defined per pi = s0,s1,s2,s3,..,sn
For every k>=0, it holds that pi(k) not sat p. This contradicts 0, as pi(0) satisfies p, as it is a state formula, s0 sat p
=> tautology
v
ii.) **not a tautology**, see counterexample



Exam 18.10.2019 (fmi195.pdf)

Block 1

1.) Consider the following decision problem:

> **REACHING LINE in LESS STEPS (RLLS)**
>
> INSTANCE: A tuple $(\Pi_1, \Pi_2, I, n_1, n_2)$, where Π_1, Π_2 are programs that take a string as input, I a string, and n_1, n_2 integers.
>
> QUESTION: Does Π_1 when applied to I reach line n_1 (of the source code of Π_1) in strictly less computation steps than Π_2 when applied to I reaches line n_2 (of the source code of Π_2)?

Remark: If neither Π_1 reaches line n_1 nor Π_2 reaches line n_2, we have a negative instance of RLLS.

(1) By providing a suitable many-one reduction from the **HALTING** problem, prove that **REACHING LINE in LESS STEPS** is undecidable.

(2) Is **REACHING LINE in LESS STEPS** semi-decidable? Explain your answer.

(15 points)

Solution by student: (Please provide some feedback)

October 18 2013 - Block 1

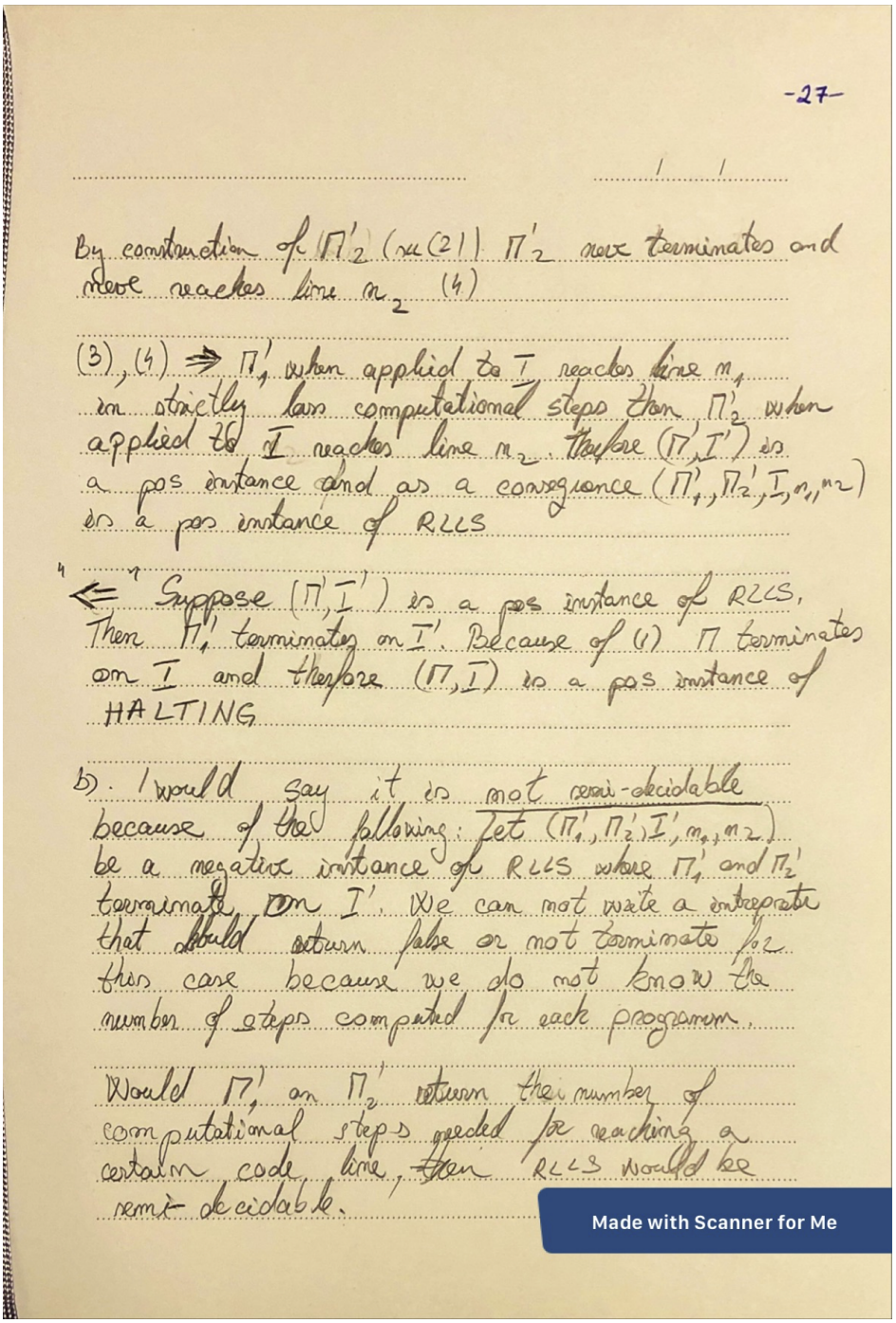
a) Let (Π, I) be an arbitrary instance of HALTING
Be $(\Pi_1', \Pi_2', I', m_1, m_2)$ a instance of RLLS
Constructed as follows:
- Let (Π', I') be a mapping of $(\Pi_1', \Pi_2', I', m_1, m_2$
where $\Pi_1', \Pi_2', m_1, m_2, I$ are hard coded

String Π' (String I')
$\Pi_1'(I)$ | I is hard coded ; I' is ignored
$\Pi_2'(I)$

(1) where $\Pi_1' = \Pi$, m_1 any line in Π_1' and Π defined
(2) Π_2' defined as follows:
String Π_2 (String I)
while TRUE do {}
return TRUE // line for $m_2 = 2$
where $m_2 = 2$

(Π, I) is a pos instance if Π terminates on I and Π_2 does not on I
To prove the correctness of the reduction we have to show that:
(Π, I) is a positive instance of HALTING $\Leftrightarrow$
(Π', I') is a positive instance of RLLS

"$\Rightarrow$" Suppose (Π, I) is a positive instance of HALTING. Then Π terminates on I.
By construction of Π_1' (see (1)) $\Rightarrow$ Π_1' reaches line m_1 (3)



Different approach by student

(PI, I) is instance of Halting, (PI1, PI2, n1, n2, I') is an instance of RLLS.
let PI1=PI and I'=I, n1 = number of last line of PI, PI2=never terminating while loop, n2 = number of last line in PI2, which is never reached because of the never terminating loop

(PI, I) is a positive instance of Halting =>

PI2 never stops, n2 is never reached, as PI terminates on I, PI1 terminates as well, ie reaches n1 with less computational steps than PI2, n2 (which is never reached by construction), therefore (PI1, PI2, n1, n2, I') is a positive instance of RLLS

<= (PI1, PI2, n1, n2, I') is a positive instance of RLLS

by construction PI2 never stops (n2 is never reached). As the tuple is a positive instance of RLLS, PI1 reaches n1 with less computational steps than PI2, n2. As n1 is set to be the last line of PI1 (return statement, when the program terminates), it is guaranteed that PI1 terminates, as PI1 = PI and I=I', PI terminates as well, which means PI is a positive instance of halting

Block 2 (missing)

2.) (a) Consider φ: $a[i] \doteq e \rightarrow a\langle i \triangleleft e\rangle \doteq a$. If φ is $\mathcal{T}_A^{=}$-valid then provide a proof using the semantic argument method from the lecture. If φ is not $\mathcal{T}_A^{=}$-valid then provide a counter-example. Besides the equality axioms, you have the following ones for arrays.

i. $\forall a, i, j \ \left(i \doteq j \rightarrow a[i] \doteq a[j]\right)$ (array congruence)

ii. $\forall a, v, i, j \ \left(i \doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq v\right)$ (read-over-write 1)

iii. $\forall a, v, i, j \ \left(i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq a[j]\right)$ (read-over-write 2)

iv. $\forall a, b \left(\left(\forall j \ a[j] \doteq b[j]\right) \leftrightarrow a \doteq b\right)$ (extensionality)

Please be precise. In a proof indicate exactly why proof lines follow from some other(s). If you use derived rules you have to prove them. Recall that a counter-example has to satisfy all axioms and falsifies φ.

(11 points)

(b) First define the concept of a theory and of a $\mathcal{T}$-interpretation. Then use them to define:

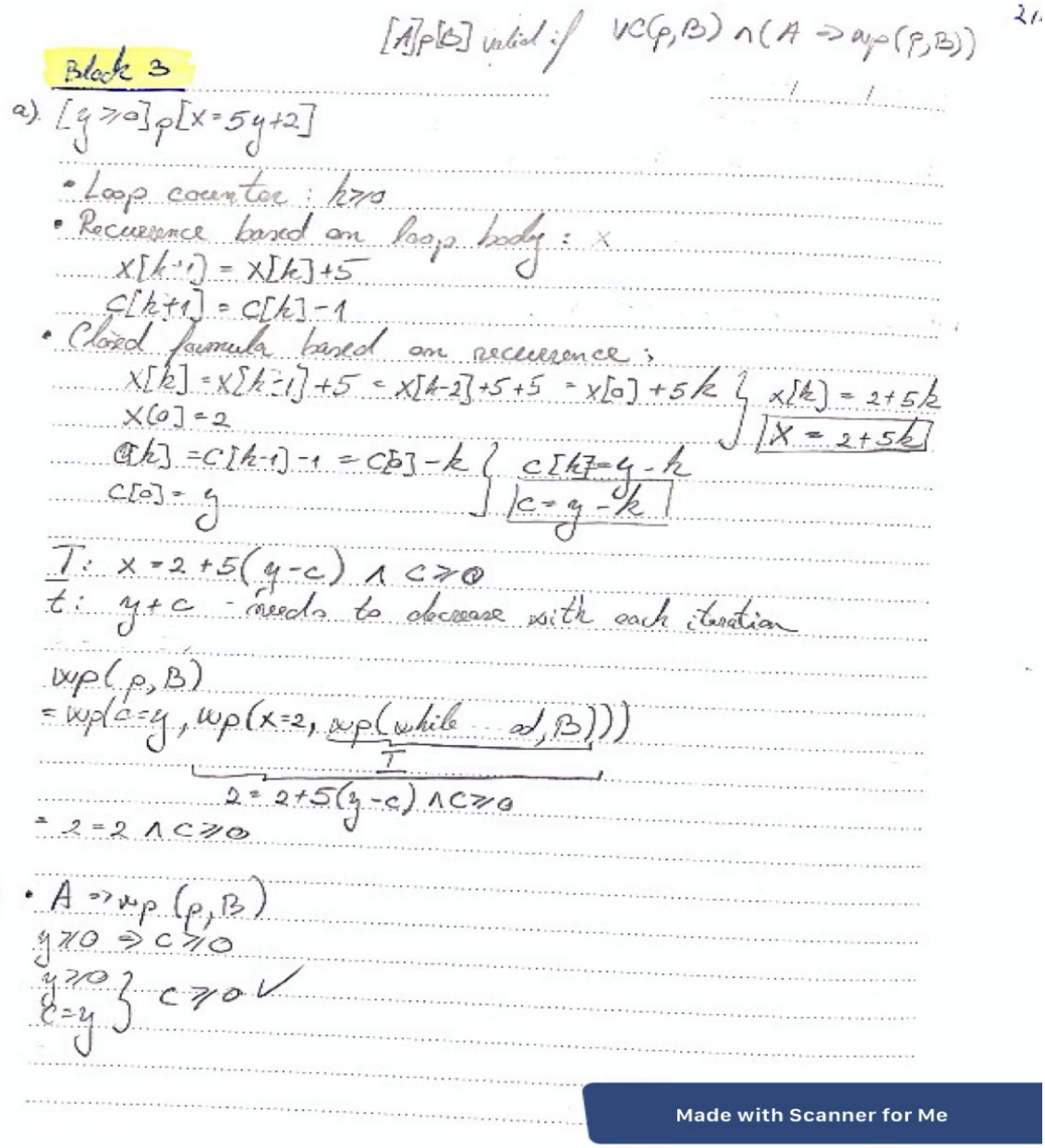
i. the $\mathcal{T}$-satisfiability of a formula;

ii. the $\mathcal{T}$-validity of a formula.

Additionally define the completeness of a theory $\mathcal{T}$. **(4 points)**

Block 3

\


[A]p[B] valid if VC(p,B) ∧ (A ⇒ wp(p,B))
Block 3
a) [y ≥ 0] p [x = 5y + 2]
• Loop counter : k ≥ 0
• Recurrence based on loop body : x
x[k+1] = x[k] + 5
c[k+1] = c[k] - 1
• Closed formula based on recurrence :
x[k] = x[k-1] + 5 = x[k-2] + 5 + 5 = x[0] + 5k
x[0] = 2
c[k] = c[k-1] - 1 = c[0] - k
c[0] = y
x[k] = 2 + 5k
x = 2 + 5k
c[k] = y - k
c = y - k
I: x = 2 + 5(y - c) ∧ c ≥ 0
t: y + c - needs to decrease with each iteration
wp(p, B)
= wp(c = y, wp(x = 2, wp(while ... od, B)))
I
2 = 2 + 5(y - c) ∧ c ≥ 0
= 2 = 2 ∧ c ≥ 0
(1) • A ⇒ wp(p, B)
y ≥ 0 ⇒ c ≥ 0
y ≥ 0
c = y
c ≥ 0 ✓
Made with Scanner for Me

VC for wp

$VC(\text{while } a\ b) = (I \wedge \neg b) \Rightarrow B \wedge (I \wedge b) \Rightarrow t \geq 0 \wedge (I \wedge b \wedge t = t_0) \Rightarrow wp(p, I \wedge t < t_0)$
$\wedge VC(p, I \wedge t < t_0)$

(2) $(I \wedge \neg b) \Rightarrow B$ ✓

$x = 2 + 5(y - c) \wedge c \geq 0 \wedge \neg(c > 0) \Rightarrow x = 5y + 2$

$c \geq 0 \wedge c \leq 0 \Rightarrow c = 0$
$x = 2 + 5(y - c)$
$\Big\} \; x = 5y + 2$

(3) $(I \wedge b) \Rightarrow t \geq 0$ ✓

$x = 2 + 5(y - c) \wedge c \geq 0 \wedge c > 0 \Rightarrow y + c \geq 0$

$y \geq 0$
$c > 0$
$\Big\} \; y + c \geq 0$

(4) $I \wedge b \wedge t = t_0 \Rightarrow wp(p, I \wedge t < t_0)$ ✓

$x = 2 + 5(y - c) \wedge c \geq 0 \wedge c > 0 \wedge t = t_0 \Rightarrow wp(x = x + 5, \underbrace{wp(c = c - 1, I \wedge t < t_0)}_{x = 2 + 5(y - c - 1) \wedge c - 1 \geq 0 \wedge y + c - 1 < t_0})$

$x + 5 = 2 + 5(y - c + 1) \wedge c - 1 \geq 0 \wedge y + c - 1 < t_0$

$\begin{cases} x = 2 + 5(y - c) & | + 5 \\ x + 5 = 2 + 5(y - c + 1) & \checkmark \end{cases}$

$c > 0 \Rightarrow c - 1 \geq 0$ ✓

$y + c = t_0 \Rightarrow y + c - 1 < t_0$ ✓

(5) $VC(p, I \wedge t < t_0)$

$= VC(c = c - 1, I \wedge t < t_0) \wedge \underbrace{VC(x = x + 5, wp(\ldots, I \wedge t < t_0))}_{\text{true}}$

$\underbrace{\qquad\qquad}_{\text{true}}$

$VC(c = y, x = 2, \text{while } \ldots \text{ od}, B)$

$= VC(\ldots) \wedge \text{true}$
$VC(\ldots) \wedge \text{true}$

(6) $VC(\text{while } x = x + 5;\ c = c - 1 \text{ od}, B)$

$= \text{true}$ (see (2), (3), (4), (5))

(6), (1) $\Rightarrow$

$VC(p, B) \wedge (A \Rightarrow wp(p, B))$

true

b). Let σ be a state s.t $\sigma(x) < a$ and $\sigma(y) < a$. Then $\sigma \models$ true but $\sigma \not\models x = x + y$ since the loop is never executed Therefore $\{true\}\ q\ \{x = x+y\}$ is not partial correct

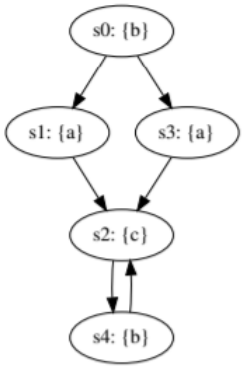
c). $\vdash \{A\}\ p\ \{B\}$ iff $\sigma \models A \Rightarrow (\forall \sigma' \langle p, \sigma \rangle \to \sigma' \Rightarrow \sigma' \models B)$

weacest precondition P for which $\{P\}\ q\ \{x = x+y\}$ is valid is false.

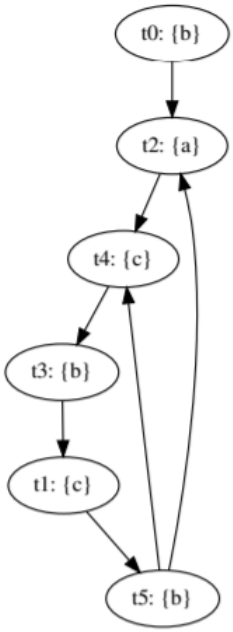
Block 4

4.) (a) Provide a simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



Kripke structure M_2:

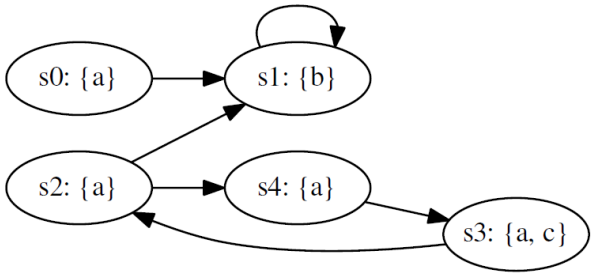


(5 points)

H={(s0,t0),(s1,t2),(s3,t2),(s2,t4),(s4,t3)}

Other solution:
H={(s0,t0),(s1,t 2),(s3,t2),(s2,t4),(s4,t3),(s2,t1),(s4,t5)}

4b) Consider the following Kripke structure M



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{G}(a)$	☐	☐	☐	
$((a \wedge c)\ \mathbf{U}\ (a))$	☐	☐	☐	
$\mathbf{AF}(a \wedge c)$	☐	☐	☐	
$\mathbf{AX}(b)$	☐	☐	☐	
$\mathbf{E}[(c)\ \mathbf{U}\ (b)]$	☐	☐	☐	

Formula number	CTL	LTL	CTL*	States
1		x	x	-
2		x	x	s0, s2, s3, s4
3	x		x	s3, s4
4	x		x	s0, s1
5	x		x	s1

4c) LTL tautologies

(c) **LTL tautologies**

Prove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π in M, or find a Kripke structure M and path π in M, for which the formula does not hold and justify your answer.

i.

$$\mathbf{F}(\mathbf{F}p \wedge \mathbf{G}q) \Rightarrow \mathbf{F}p \wedge \mathbf{FG}q$$

ii.

$$\mathbf{F}(\mathbf{F}p \wedge \mathbf{G}q) \Leftarrow \mathbf{F}p \wedge \mathbf{FG}q$$

4c i) F(Fp and Gp) => Fp and FGq

i I think this is a tautology, but I cannot prove it nicely. =)

Student:

F(Fp and Gp) => Fp and FGq

Meaning of F(Fp and Gp) : There exists a i>= 0 such that M,Pi^i |= Fp and M,Pi^i |= Gp

Meaning of FP and FGq: There exists a k>=0 such that M,Pi^k |= p and there exists a j>=0 such that M,Pi^j |= Gq

Assume M,Pi |/= Fp and FGq

|/=FGq ⇔ |= not (Fp and FGq) ⇔ |= not Fp or not FGq ⇔ |= G not p or G not (Gq)

⇔ |= G not p or G F not q

|= G not p contradicts with "**There exists a i>= 0 such that M,Pi^i |= Fp** and M,Pi^i |= Gq"

|= G F not q contradicts with "**There exists a i>= 0 such tha**t M,Pi^i |= Fp and **M,Pi^i |= Gq**"

Therefore F(Fp and Gp) => Fp and FGq **is true**

Another student:

Let M be an arbitrary Kripke structure with starting state s0. Let π be some path -> s0, s1, ...

Let's try to prove this is a tautology by contradiction. Let's assume LHS is true and RHS is false. Thus for LHS it is true that □ i (i > 0) for wh=ich M, π^i |= Fp (0) and M, π^i |= Gq (1). Since we assumed RHS is false, it holds that !□ j (j >= 0) for which M, π^j |= p (2) and M, π^j |= Gq (3). (0) and (1), as well as (2) and (3) clearly contradict each other.

4c ii) F(Fp and Gq) <= Fp and FGq

ii is not a tautology.
Counterexample: s0(p) -> s1(q)selflooped
RHS is true, since “p in future” holds, actually right now in s0, and “globally q in future” holds in s1. But LHS is false, since “p in future” holds in s0, but “globally q” holds only in the next statement.

Exam 28.6.2019 (fmi194.pdf)

Block 1

1.) Consider the following decision problem:

> **EXACTLY-ONE-HALTS (EOH)**
>
> INSTANCE: A tuple (Π_1, Π_2, I), where Π_1, Π_2 are programs that take a string as input, and I is a string.
> QUESTION: Does either Π_1 halt on I or Π_2 halt on I (but not both halt on I)?

(a) By providing a suitable many-one reduction from the **HALTING** problem, prove that **EXACTLY-ONE-HALTS** is undecidable.

(10 points)

Solved by Student:
Let (Pi, I') be an arbitrary instance of Halting and Pi1 = Pi && I = I' && Pi2 hardcoded never terminates (endless loop)
Positive instance of halting ⇔ Positive instance of EOH
" =>" assume (Pi, I') is a positive instance of halting, i.e. Pi halts on I'. As Pi1 = Pi & I = I' we know that Pi1 must halt on I. Due to construction of Pi2 we know it never halts -> pos instance of EOH
"<=" assume (Pi1, Pi2, I) is a positive instance of EOH. Due to our construction we know that Pi2 never halts on I'. Therefore Pi1 must halt on I'. As we have Pi = Pi1 and I = I' we have that Pi halts on I' and therefore pos instance of HALTING

(b) Is **EXACTLY-ONE-HALTS** semi-decidable? Explain your answer.

(5 points)

Not semi-decidable; We can not build an interpreter for this. Interpreter has to return true on all positive instances and might not terminate on negative instances.
We have two programs and one is supposed not to halt. If we call Pi1 before Pi2 and Pi1 does not halt we would never reach any statement after the execution of Pi1.
Same with Pi2 first.. Therefore it would not return/terminate on a positive instance of EOH
→ not semi-decidable;
Note: even in parallel execution one of the calls has to run forever to determine if a program is not halting.

Block 2

2.) (a) Consider $\mathcal{T}_{PA}^{+}$, which is Peano arithmetic with the axioms

$\forall x\ \neg(x + 1 \doteq 0)$	(zero)
$\forall x\ \forall y\ (x + 1 \doteq y + 1 \rightarrow x \doteq y)$	(successor)
$F[0] \wedge (\forall x\ (F[x] \rightarrow F[x + 1])) \rightarrow \forall x\ F[x]$	(induction)
$\forall x\ (x + 0 \doteq x)$	(plus zero)
$\forall x\ \forall y\ (x + (y + 1) \doteq (x + y) + 1)$	(plus successor)
$\forall x\ (x \cdot 0 \doteq 0)$	(times zero)
$\forall x\ \forall y\ (x \cdot (y + 1) \doteq (x \cdot y) + x)$	(time successor)

together with the following additional axioms:

$\forall x\ (x^0 \doteq 1)$	(exp zero)
$\forall x\ \forall y\ (x^{y+1} \doteq x^y \cdot x)$	(exp succ)
$\forall x\ \forall z\ (exp_3(x, 0, z) \doteq z)$	(exp_3 zero)
$\forall x\ \forall y\ \forall z\ (exp_3(x, y + 1, z) \doteq exp_3(x, y, x \cdot z))$	(exp_3 succ)

Show by induction that φ: $\forall x \forall y\ (exp_3(x, y, 1) \doteq x^y \cdot 1)$ is $\mathcal{T}_{PA}^{+}$-valid. Use the semantic argument method from the lecture to formally prove the formula in the base case and in the step case. In order to sImplify the proof, you may use the formulas (L): $\forall x\ (1 \cdot x \doteq x)$ and (A): $\forall x \forall y \forall z\ (x \cdot (y \cdot z) \doteq (x \cdot y) \cdot z)$ as additional lemmas.

Please be precise and indicate exactly why proof lines follow from some other(s). Moreover, recall that equality handling is performed using equality axioms.

Hint: You need to prove a stronger formula from which φ follows.

(12 points)

See Block 2 from 03.05.19

Suggestion: Stronger formula φ' with general z instead of 1. z=1 is just a specific case, so if we can prove φ' for all z, then φ with z=1 follows. z is also a free variable.

(b) Apply the Ackermann reduction to the formula

$$\varphi^{EUF}: F(F(x_1)) \doteq G(x_2, G(x_1, x_3, x_4), F(x_2)) \rightarrow p(x_1, y)$$

and obtain the validity-equivalent formula φ^E. **(3 points)**

Solution 15.
First, we replace the predicate (as described in the lecture) and obtain:

$$\varphi' : \quad F(F(x_1)) \doteq G(x_2, G(x_1, x_3, x_4), F(x_2)) \rightarrow H_p(x_1, y) \doteq x_p.$$

Label-ling function occurrences inside-out yields:

$$\varphi'' : \quad F_2(F_1(x_1)) \doteq G_2(x_2, G_1(x_1, x_3, x_4), F_3(x_2)) \rightarrow H_p(x_1, y) \doteq x_p.$$

The propositional skeleton is:

$$flat^E : \quad f_2 \doteq g_2 \rightarrow h_p \doteq x_p.$$

FC^E is the conjunction of the following functional constraints:

$$\begin{aligned} x_1 \doteq f_1 &\rightarrow f_1 \doteq f_2 \\ x_1 \doteq x_2 &\rightarrow f_1 \doteq f_3 \\ f_1 \doteq x_2 &\rightarrow f_2 \doteq f_3 \\ (x_2 \doteq x_1 \wedge g_1 \doteq x_3 \wedge f_3 \doteq x_4) &\rightarrow g_1 \doteq g_2 \end{aligned}$$

Note that there are no constraints for h_p since it only occurs once in φ'. Finally, φ^E is $FC^E \rightarrow flat^E$.

192

Same as in HW from SS2019 (Solution in VoWi):
First, we replace the predicate (as described in the lecture) and obtain:
φ′:F(F(x1))=G(x2,G(x1,x3,x4),F(x2))→Hp(x1,y) =xp
Label-ling function occurrences inside-out yields:
φ″:F2(F1(x1))=G2(x2,G1(x1,x3,x4),F3(x2))→Hp(x1,y)=xp.
The propositional skeleton is:
flatE: f2=g2→hp=xp
FCE is the conjunction of the following functional constraints:
x1=f1→f1=f2
x1=x2→f1=f3
f1=x2→f2=f3
(x2=x1∧g1=x3∧f3=x4)→g1=g2
Note that there are no constraints for hp since it only occurs once in φ′. Finally, φE is FCE→flatE.

Solution by student. Needs to be approved

June 28 2019 - Block 2

a) φ: $\forall x \forall y \, (exp_3(x,y,1) \doteq x^y \cdot 1)$

Let φ': $\forall x \forall y \forall z \, (exp_3(x,y,z) \doteq x^y \cdot z)$

with φ as special case of φ' with $z = 1$

Therefore if φ' holds than φ holds.

We prove φ' by stepwise induction over y.

Let $F[y]$ denote

$$\forall x \forall z \, (exp_3(x,y,z) \doteq x^y \cdot z)$$

Base case: $y = 0$; $F[0]: \forall x \forall z (exp_3(x, 0, z) \doteq x^{0} \cdot z) \Rightarrow$
$\Rightarrow F[0]: \forall x, \forall z (exp_3(x, 0, z) \doteq z)$ (see exp Basis)
We have to prove the formula

$$F[0]: \forall x \forall z (exp_3(x, 0, z) \doteq z)$$

$F[0]: \forall x \forall z (exp_3(x, 0, z) \doteq z)$
$\forall x \forall z (exp_3(x, 0, z) = z)$ (exp_3 Basis) $\Rightarrow F[0]$ holds

Induction Hypothesis (IH): Assume $F[y]: \forall x \forall z (exp_3(x, y, z) \doteq x^{y} \cdot z)$ holds

Induction Step: We show that for all y
$F[y+1]: \forall x \forall z (exp_3(x, y+1, z) \doteq x^{y+1} \cdot z)$
holds.

Without loss of generality, we will assume that the following applies for all x, y and not write the universal quantifier in the next proof
Let
ψ : $exp_3(x, y+1, z) \doteq x^{y+1} \cdot z$;

Proof by contradiction. Assume there exists a $\mathfrak{I}_{PA}^{+}$ - interpretation structure $\mathfrak{I}$, with $\mathfrak{I} \not\models \psi$.

1. $I \not\models exp_3(x, y+1, z) \doteq x^{y+1} \cdot z$ assumption
2. $I \models exp_3(x, y+1, z) \doteq exp_3(x, y, x \cdot z)$ exp_3 succ
3. $I \not\models exp_3(x, y, z) \doteq x^y \cdot z$ IH
4. $I \not\models exp_3(x, y, x \cdot z) = x^y \cdot x \cdot z$ 2, 3
5. $I \not\models exp_3(x, y+1, z) = x^y \cdot x \cdot z$ 2, 4 transitivity
6. $I \not\models exp_3(x, y+1, z) = x^{y+1} \cdot z$ 5, exp succ
7. $I \not\models \bot$ 1, 7 contradiction

The assumption is false. Therefore φ_1 holds.
Therefore φ' is J^+_{PA} - valid. Since φ is a special case of φ', φ is also J^+_{PA} - valid.

b) $\varphi^{EUF} : F(F(x_1)) = G(x_2, G(x_1, x_3, x_4), F(x_2)) \Rightarrow p(x_3, y)$,
$\varphi^E = ?$

Replace predicat:

$$\varphi' : F(F(x_1)) \doteq G(x_2, G(x_1, x_3, x_4), F(x_2)) \Rightarrow Hp(x_4, y) \doteq xp$$

Labeling the functions:

$$\varphi'' : F_2(F_1(x_1)) \doteq G_2(x_2, G_1(x_1, x_3, x_4), F_3(x_2)) \Rightarrow Hp(x_4, y) \doteq xp$$

$\varphi^E : FC^E \Rightarrow flat^E$

$flat^E : /_2 \doteq g_2 \Rightarrow hp \doteq xp$

FC^E: $x_1 \doteq f_1 \to f_1 \doteq f_2 \wedge$

$x_1 \doteq x_2 \to f_1 \doteq f_3 \wedge$

$f_1 \doteq x_2 \to f_2 \doteq f_3 \wedge$

$x_2 \doteq x_1 \wedge g_1 \doteq x_3 \wedge f_3 = x_2 \to g_1 \doteq g_2$

$\varphi^E : FC^E \to Mat^E$

Block 3

Note that all programs within this exercise are programs over the integers, that is, every program variable can only take integer values.

(a) Show that the Hoare triple $\{a \geq 0 \wedge b \geq 0\}\ p\ \{a - b * c - d = 0\}$ is valid with respect to partial correctness where p is the following program:

$c := 0;$
$d := a;$
while $b \leq d$ **do**
  $d := d - b;$
  $c := c + 1;$
od

(8 points)

Solved by Student
I.. Invariant
b...loop condition

B… postcondition
A…

precondition

Invariant:
D[n] = d0 -nb
C[n] = c0 +n
C[n] = n
D[n] a-cb because D[n] = a-nb, d0 -nb = a-nb and C[n] = n
Invariant: d = a-cb
I
Step 1
A ⇒ I [d/a][c/0] (or equivalently A => wlp(c := 0, wlp(d := a, wlp(loop, B))))
(a>= 0 & b>= 0) ⇒ d = a-cb | plugin initial values
(a>= 0 & b>=0) ⇒ a = a - 0*b
Trivially true

Step 2
VC(while..)
I & !b ⇒ B &
I & b ⇒ wlp(p,i) &
VC(p,I)

I & !b ⇒ B
d = a-cb & b>d ⇒ a-bc-d = 0 tautology (we can ignore b>d)

I & b ⇒ wlp(p,I)
Lhs ⇒ (d-b) = a- (c+1)*b
⇒ (d-b) = a-(bc+b)
⇒ d-b = a-bc-b | +b both sides
⇒ d = a-bc
Tautology

VC(p,I) true
All true -> valid

(b) Is the Hoare triple from Exercise 3a totally correct? If yes, provide a variant, otherwise provide a counterexample. In both cases, justify your answer!

(2 points)

Solved by Student

~~Yes: variant: n-c~~

~~Program is already partially correct (proven before) only thing missing for TC is termination. Due to pre,post and loopcondition, there is no way the program can be stuck in an endless loop.~~

~~I.e.: d is always starting with a, and c is initialized with 0 → even without entering the loop, the multiplication in the postcondition returns 0 and we have a-d → a-a = 0 so it is always true;~~

~~Variant properties: decreases strictly with each iteration (yes), stays positive before and within loop (yes)~~

Counterexample: b = 0; a = 1;

(c) Let p be a program and let $A, B \in \text{Assn}$. Is the following Hoare rule sound/admissible? If yes, provide a proof. Otherwise, provide a counterexample and argue why it is a counterexample.

$$\frac{\{A\}\ \textbf{skip};\ p\ \{B\}}{\{A\}\ p\ \{B\}}$$

(5 points)

Solved by Student - please confirm (I doubt my hoare calculus is correct, bc I am not sure when I should use the consequence here...)
If premise can be derived from known rules it is sound/admissible;

{A} skip {C} {C} => {A}
_____ cons
{A} skip {A} {A} ⇒ {C} {C} p {B}
_____skip _____ cons
{A} skip {C} {C} p {B}
_____ seq
{A} skip; p {B}
_____
{A} p {B}

It is admissible because we have {C} => {A} and {A} => {C} which is {A} ⇔ {C} so if A holds, C also holds. And {A} p {B} therefore becomes valid.

What about this?:

_____skip
{A} skip {A} {A} p {B}
_____ seq
{A} skip; p {B}
_____
{A} p {B}

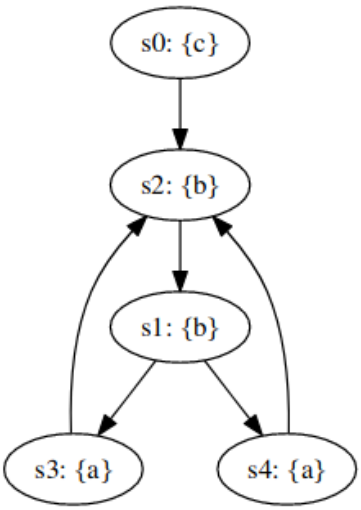
Another student attempt, in a similar way as in the solved exercise sheets (no idea if correct, feedback welcome!):

Since Hoare Logic is sound, it suffices to show that it {A} skip;p {B} is provable, then so is {A} p {B}. Hence, assume |- {A} skip;p {B}. By the Hoare rule for sequential composition, there exists assertion C s.t. 1) |- {A} skip {C} and 2) |- {C} p {B}. By the Hoare rule for skip, we know that 3) A => C must hold. We use the rule of consequence for 2) and 3) and arrive at |- {A} p {B}.

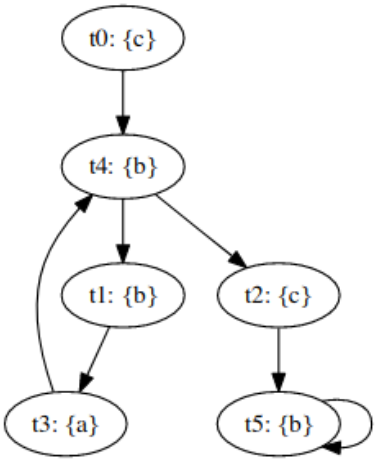
Block 4

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



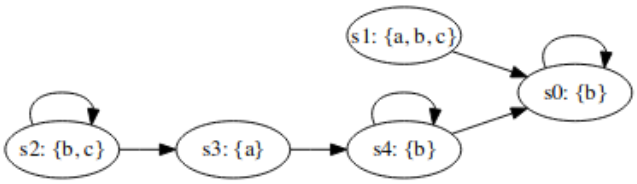
Kripke structure M_2:



(4 points)

H = {(s0,t0),(s2,t4),(s1,t1),(s3,t3),(s4,t3)}

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and
ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{F}(a)$	☐	☐	☐	
$((b \wedge c)\ \mathbf{U}\ (c))$	☐	☐	☐	
$\mathbf{AX}(a \wedge b)$	☐	☐	☐	
$\mathbf{EG}(c)$	☐	☐	☐	
$\mathbf{E}[(a \wedge b \wedge c)\ \mathbf{U}\ (a)]$	☐	☐	☐	

(5 points)

phi	CTL	LTL	CTL*	States
F(a)		X	X	s1,s3
((b&c) U (c))		X	X	s1,s2
AX(a&b)	X		X	----
EG(c)	X		X	s2
E[(a&b&c) U (a)]	X		X	s1,s3

(c) **LTL tautologies**

Provide a Kripke structure, which satisfies all of the following CTL* formulas:

$$\neg q$$
$$\mathbf{AG}p \Rightarrow q$$
$$\mathbf{EF}p$$
$$\mathbf{EFG}\neg p$$

(6 points)

Provide a Kripke structure, which satisfies all...

Solution:
p(selflooped) → q&!p → !q(selflooped)

!q: for all paths starting at state s lead to !q
AGp ⇒ q = !(AGp) v q = EF !p v q:
For all paths, where every state has p is followed by q = there exist a state in the future where !p OR q
EFp: there exists a path, where at some state we have p
EFG !P: there exists a path, where at some state all following states !p

Other solution:

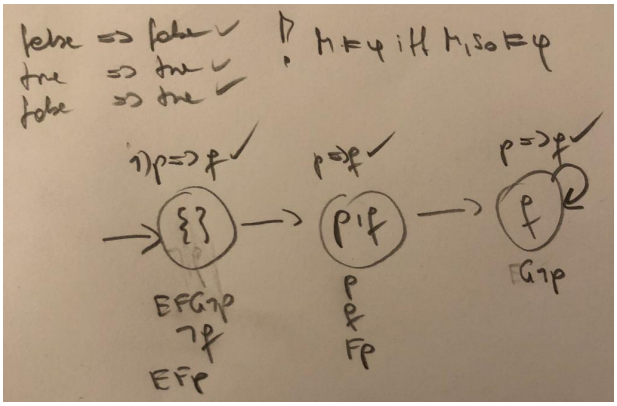
~~-> (s0: {p,q}) -> (s1: {!q,!p}) -> selfloop s1~~

~~!q: is contained in the set of atomic propositions~~
~~AG p > q: on all paths, always where p is also q (s0)~~
~~EF: there exists a path where at some point p (s0)~~
~~EFG !p: There exists a path, where at some point, on all states forever will be !p (s1 selfloop)~~

Other solution:
(s0: {p}) --> (s1: {}) --> selfloop s1

!q: true (s0, s1)
AGp => q: (AGp false, implication always true) (s0, s1)
EFp: true (s0)
EFG!p: true (s1)

Status: Solved by student



Solved by student:

S = (s0, s1)
R = {(s0, s1), (s1, s1)}
L(s0) = p
L(s1) = q

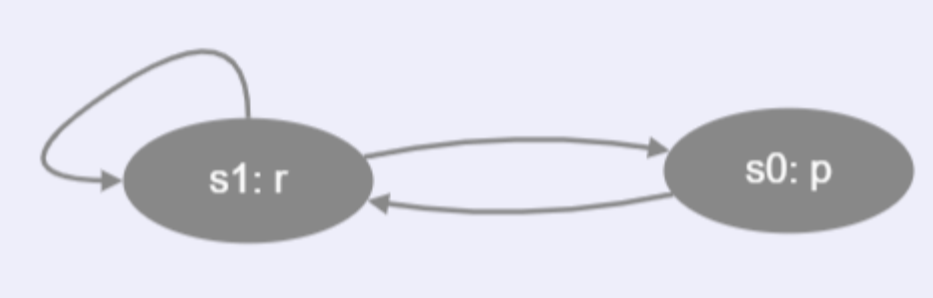
* not q - satisfied by s0
* EFp - satisfied by s0
* EFG(not p) - satisfied by s1
* AGp => q = not(AGp) or q = not(not(EF) not(p)) or q = EGp or q - satisfied by s1

Solved by another student:

->(s0:) -> (s1: p) -> (s2, q) -> selfloop s2

Solved by another student:

Every state is satisfying every given CTL* formula. Note: s1 could also be empty


s1: r
s0: p

Exam 03.05.2019 (fmi193.pdf)

Block 1-

1.) An undirected graph is called a *non-2-degree graph* if no vertex in the graph has exactly two edges to other vertices.

Examples: $(\{a, b, c, d\}, \{[a, b], [b, c], [b, d]\})$ is non-2-degree, while $(\{a, b, c, d\}, \{[a, b], [b, c], [c, d]\})$ or $(\{a, b, c\}, \{[a, b], [b, c], [a, c]\})$ are not.

Consider the following problem:

> **3-COLORABILITY-N2D**
>
> INSTANCE: A non-2-degree graph $G = (V, E)$.
>
> QUESTION: Does there exists a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

Use the fact that the standard version of the **3-COLORABILITY** problem is NP-complete to prove that **3-COLORABILITY-N2D** is NP-complete as well. Give a brief argument for NP-membership and show NP-hardness by providing a many-one reduction from the **3-COLORABILITY** problem. Prove the correctness of your reduction.

Recall that **3-COLORABILITY** is defined as follows:

> **3-COLORABILITY**
>
> INSTANCE: An undirected graph $G = (V, E)$.
>
> QUESTION: Does there exists a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

Solution: see slides below

(
Solved by student (pls check)
G = arbitrary pos instance of 3-colorability;
Construct G* as non-2-degree by using G as base:
For each vertex with degree 2 add edge + new vertex;
Assign color to the new vertex by using one of the 2 remaining colors;
As we added new vertices with different colors, we preserve the 3-colorability of the base;
(See 2-colorability to 3-colorability reduction)
)

(
Solution by other student (pls check):
NP-membership: 3COL-N2D is in NP, since it is a special case of 3COL which is in NP (i.e., each instance of 3COL-N2D is also an instance of 3COL, thus 3COL-N2D cannot be harder than 3COL).

NP-hardness: We provide a polynomial-time reduction from 3COL. Let G = (V; E) be an arbitrary instance of 3COL. We construct an instance G' = (V'; E') of 3COL-N2D by setting V' =
; v1; v2 | v E V) and
E' = E U {[v, v1], [v, v2]}. By definition, each such G' is a Non-2-degree graph thus the reduction yields the correct objects for the problem 3COL-N2D. (Because every vertex has at least an edge and when we add two more edges,it will become a N2D graph) We next show the correctness of the reduction.

The rest is the same as in 3 COL NT
)

Exercise 6

An undirected graph is called a *non-2-degree graph* if no vertex in the graph has exactly two edges to other vertices.
Examples: Graph $(\{a, b, c, d\}, \{[a, b], [b, c], [b, d]\})$ is non-2-degree, while graphs $(\{a, b, c, d\}, \{[a, b], [b, c], [c, d]\})$ or $(\{a, b, c\}, \{[a, b], [b, c], [a, c]\})$ are not.

Consider the following problem:

3-COLORABILITY-N2D (3COL-N2D)

INSTANCE: A non-terminal graph $G = (V, E)$.

QUESTION: Does there exist a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

Use the fact that the standard version of the **3-COLORABILITY** problem is NP-complete, to prove that **3COL-N2D** is NP-complete as well.

Solution to Exercise 6

Recall that **3-COLORABILITY** is defined as follows:

3-COLORABILITY (3COL)

INSTANCE: An undirected graph $G = (V, E)$.

QUESTION: Does there exist a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

To show NP-completeness of **3COL-N2D** we need to prove NP-membership and then NP-hardness for the problem.

NP-membership: **3COL-N2D** is in NP, since it is a special case of **3COL** which is in NP (i.e., each instance of **3COL-N2D** is also an instance of **3COL**, thus **3COL-N2D** cannot be harder than **3COL**).

Solution to Exercise 6 (continued)

NP-hardness: We provide a polynomial-time reduction from **3COL**. Let $G = (V, E)$ be an arbitrary instance of **3COL** with $U \subseteq V$ being the vertices that have degree 2 in G. We construct an instance $G' = (V \cup U', E')$ of **3-COLORABILITY-N2D** with $U' = \{u' \mid u \in U\}$ and $E' = E \cup \{[u, u'] \mid u \in U\}$. By definition, each such G' is a non-2-degree graph, thus the reduction yields the correct objects for the problem **3COL-N2D**. We next show the correctness of the reduction.

Assume G is a positive instance of **3COL**. Then there exists a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$. Consider now $\mu^* : V \cup U' \to \{0, 1, 2\}$ defined as follows: for each $v \in V$, $\mu^*(v) = \mu(v)$ and for each $u \in U$: $\mu^*(u') = (\mu(u) + i)\ mod\ 3$.
We have to show that for any $[x, y] \in E'$, $\mu^*(x) \neq \mu^*(y)$. We have the following cases: (1) $x, y \in V$: then by construction and assumption that G is 3-colorable, $\mu^*(x) \neq \mu^*(y)$; otherwise $x \in U$ and $y = x'$; by definition of μ^*, we then have $\mu^*(x) \neq \mu^*(y)$ as well. Hence, G' is 3-colorable and thus a positive instance of **3COL-N2D**.

Solution to Exercise 6 (continued)

Assume G' is a positive instance of **3COL-N2D**. Then G' is 3-colorable and since G is a subgraph of G' it is clear that also G is 3-colorable. Thus G is a positive instance of **3COL**.

Block 2

2.) (a) Consider $\mathcal{T}_{PA}^{+}$, which is Peano arithmetic with the axioms

$$\forall x\ \neg(x+1 \doteq 0) \qquad \text{(zero)}$$
$$\forall x\ \forall y\ (x+1 \doteq y+1 \to x \doteq y) \qquad \text{(successor)}$$
$$F[0] \wedge (\forall x\ (F[x] \to F[x+1])) \to \forall x\ F[x] \qquad \text{(induction)}$$
$$\forall x\ (x+0 \doteq x) \qquad \text{(plus zero)}$$
$$\forall x\ \forall y\ (x+(y+1) \doteq (x+y)+1) \qquad \text{(plus successor)}$$
$$\forall x\ (x \cdot 0 \doteq 0) \qquad \text{(times zero)}$$
$$\forall x\ \forall y\ (x \cdot (y+1) \doteq (x \cdot y)+x) \qquad \text{(time successor)}$$

together with the following additional axioms:

$$\forall x\ (x^0 \doteq 1) \qquad \text{(exp zero)}$$
$$\forall x\ \forall y\ (x^{y+1} \doteq x^y \cdot x) \qquad \text{(exp succ)}$$
$$\forall x\ \forall z\ (exp_3(x,0,z) \doteq z) \qquad (exp_3 \text{ zero})$$
$$\forall x\ \forall y\ \forall z\ (exp_3(x,y+1,z) \doteq exp_3(x,y,x \cdot z)) \qquad (exp_3 \text{ succ})$$

Show that $\forall x \forall y \forall z\ (exp_3(x,y,z) \doteq x^y \cdot z)$ is $\mathcal{T}_{PA}^{+}$-valid. Perform an induction proof and use the semantic argument method from the lecture to formally prove the formula in the base case and in the step case. In order to simplify the proof, you may use the formulas $(L)\colon \forall x\ (1 \cdot x \doteq x)$ and $(A)\colon \forall x \forall y \forall z\ (x \cdot (y \cdot z) \doteq (x \cdot y) \cdot z)$ as additional lemmas.

Please be precise and indicate exactly why proof lines follow from some other(s). Moreover, recall that equality handling is performed using equality axioms.

(a) Check HW of Block 2 or extra sheet on stepwise induction over list; (discussed the exam with him; pointed me to assignments/extra sheet for solution)

Solved by student (please confirm):

We have to perform stepwise induction over lists;
Further, we will perform the induction over y:
Base Case [F0]
For the basecase we have (according to induction axiom) F[0] to prove validity, we will assume an interpretation where F[0] does not hold;
So let x and z be arbitrary values (so we can omit the forall operator; but assume on both sides x=x and z=z) and y = 0:

1 I !⊨ exp3(x,0,z) = x^0 z	assumption
2 I ⊨ exp3(x,0,z) = 1 * z	1, exp zero
3 I ⊨ exp3(x,0,z) = z	2. Apply (L)
4 I ⊨ z = z	3, exp3 zero
5 ⊥	contradiction because z = z

Induction Hypothesis: Let's assume F[y] is true;
Induction Step: we have to prove that F[y+1] holds given the IH holds, we will work towards a contradiction to prove that F[y+1] holds. We do this by assuming there is no

interpretation where F[y+1] holds.
Again we assume arbitrary values for x and z but on both sides x=x and z=z

1 I $\models$ exp3(x, y+1,z) = $x^{y+1} * z$	assumption
2 I $\models$ exp3(x, y+1, z) = $x^y * xz$	1, exp succ, (A)
3 I $\models$ exp3(x,y, xz) = $x^y * xz$	2, exp3 succ
4 I $\models$ $x^y * xz = x^y * xz$	3, IH
5 $\bot$	contradiction equality axioms

(b) Show the soundness of the following variant of the resolution rule.

$$\frac{C \vee p \vee q \quad D \vee \neg p \quad E \vee \neg q}{C \vee D \vee E}$$

(b) C v p v q !p v D

_____ transitivity of v

C v q v D E v !q

_____ transitivity of v

C v D v E

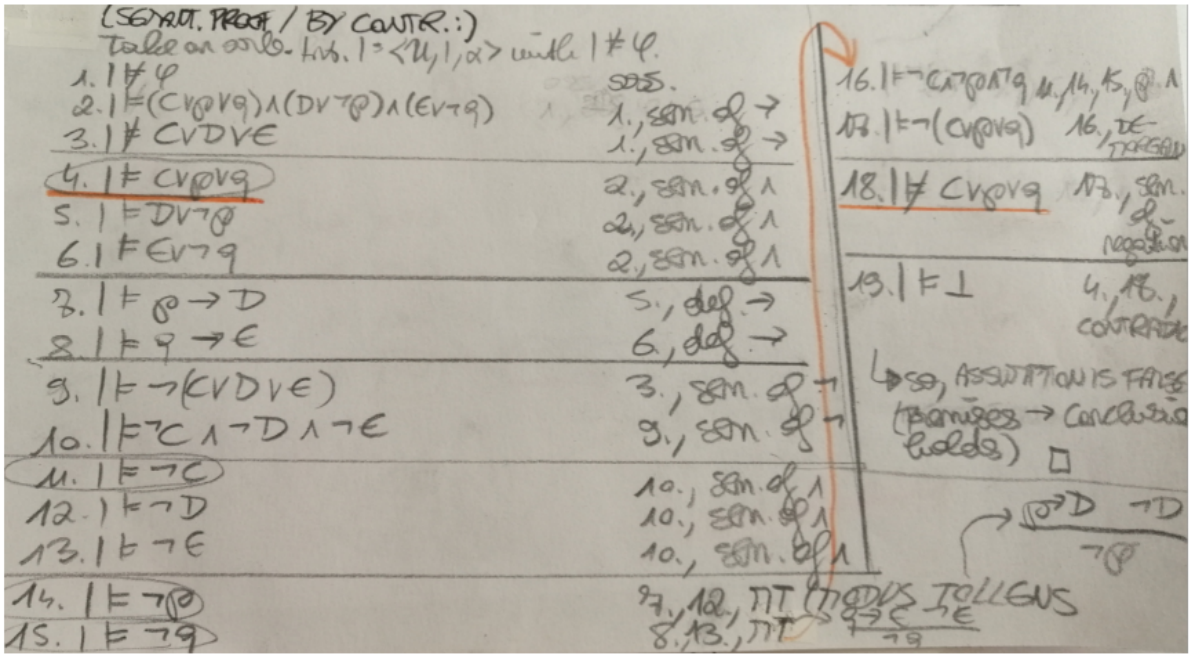
Alternative: find assignment where only C v D v E remains…

Solved by student (no plan if this is a proof.)

(C v p v q) ^ (D v !q) ^ (E v !q) ⇒ (C v D v E)
≡ ! [(C v p v q) ^ (D v !q) ^ (E v !q)] v (C v D v E) … implication def.
≡ (!C ^ !p ^ !q) v (!D ^ q) v (!E ^ q) v (C v D v E) … de morgan laws
≡ (!C v !D v !E) v (C v D v E) … some crazy resolution stuff
≡ true

Other solution: simply use resolution.
C1 = C ∨ p ∨ q; C2 = D ∨ ¬p; C3 = E ∨ ¬q
r1 = res(C1, C2, p) = C ∨ D ∨ q
r2 = res(r1, C3, q) = C ∨ D ∨ E

Proof by contradiction (semantic argument method) (solved by student):



Block 3

3.) Note that all programs within this exercise are programs over the integers, that is, every program variable can only take integer values.

(a) Show that the Hoare triple $\{x = 6 \wedge y = 9\}\ p\ \{y = 0\}$ is valid with respect to partial correctness where p is the following program:

while $x > 0$ **do**
  $x := x - 2;$
  $y := y - 3;$
od

Solved by Student:
This is my actual solution of the exam + corrections; i got a copy of the old exam and just adapted my solution according to the marking of the tutor; (basically it was correct, but i missed to strengthen the invariant (-4p), which i did here). So i assume it is completely correct now - if not please say so

Invariant: 2y = 3x & x >= 0;
How?

X[n] = x0 -2n | resolve recurrence using index notation
Y[n] = y0 - 3n;

X = 6 - 2n |rewrite without index + start values;
Y = 9 - 3n

N = 3- x/2 | apply some simple math to get n (from x)

Y = 9 - 3(3-x/2) |apply to y
Y = 9 - 9 + 3(x/2) | do some math
Y = 3(x/2)
y/3 = x/2
⇒ **2y = 3x**

We get the x> = 0 intuitively by looking at the loop condition.

Step 1:
Check if Invariant can be established by precondition (note the “x>=0” needed later)
P ⇒ I: x=6 & y = 9 ⇒ 2y=3x & x>= 0 (obviously this is true)

Step 2:
Using WLP/VC
1) I & !b ⇒ B &
2) I & b ⇒ wlp(p,I) &
3) VC(p,I)

Solving implication 1):
Invariant and negated condition implicate postcondition (without the strengthened invariant, it would not be possible to do this):
I & !b ⇒ B: 2y = 3x & x>= 0 & x<= 0 ⇒ y = 0
x>= 0 & x<= 0 → x = 0;
2y = 3*0 ⇒ y = 0

Solving implication 2):
Invariant and condition implicate wlp(p,I)
I & b ⇒ wlp(p,I)
wlp(p,I) := 2(y-3) = 3(x-2)& (x-2) >= 0
Rewrite as: 2y - 6 = 3x-6 & (x-2) >=0
Use simple math: 2y = 3x & (x-2) >= 0

So we have:

I & b ⇒ 2y = 3x & (x-2) >= 0
2y = 3x & x> 0 ⇒ 2y = 3x & (x-2) >= 0
x > 0 and later x-2 >= 0 is sound due to properties of integers (they let that argument count - not sure though)

Solving 3)
VC(p,I) = true;

Everything evaluates to true -> VALID;

(b) Let p be the program below. Is the Hoare triple $[n > 10 \wedge 2 < a < 10]\ p\ [n \leq 0]$ valid with respect to total correctness? If yes, prove its validity using the Hoare calculus. Otherwise, provide a counterexample – that is, a state that does not satisfy the correctness assertion – and argue why it is a counterexample.

```
while n > 0 do
  n := n − a;
  if n = 0 then
    abort
  fi
od
```

Solved by student:
It's not valid, because it's possible to reach "abort".

Not Valid: Counterexample.
n = 12
a = 6

(c) Is the following alternative rule for assignment sound/admissible? If yes, provide a proof. Otherwise, provide a counterexample and argue why it is a counterexample.

$$\vdash \{true\}\ x := e\ \{x = e\}$$

Solved by student:
Counterexample by setting e to an arithmetic expression: x+1

{true} x := x+1 {x = x+1} … here the postcondition is obviously false. True ⇒ False

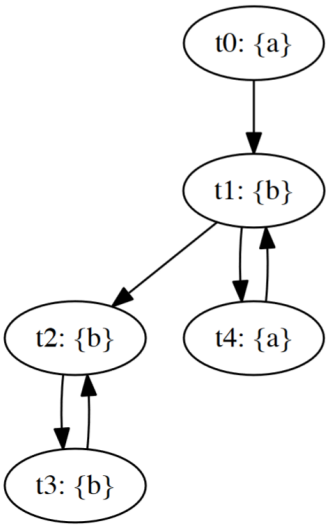
Other counterexample (solved by student): integer division
e = x/2
If x is odd it will be rounded to the nearest integer; ie: x = 5 ⇒ after the execution x = 2; while e will evaluate to 2.5; postcondition is therefore false, as 2 != 2.5;

Block 4

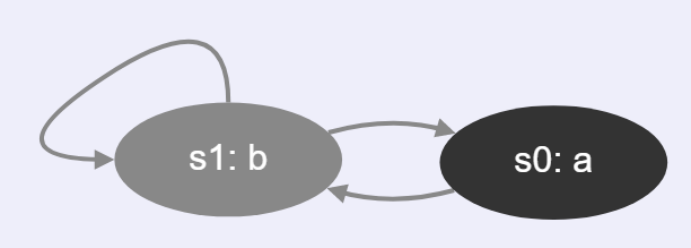
4.) (a) Consider M_1 shown below with initial state t_0.
Provide a Kripke structure M_2 such that

- $M_1 \leq M_2$ and
- M_2 has at most 3 states.

Furthermore provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$.



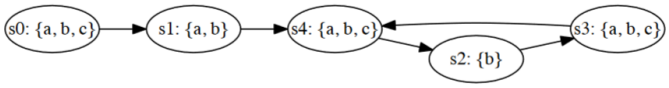
Solved by student:



with initial state s0

H = {(t0, s0), (t1, s1), (t4, s0), (t2, s1), (t3, s1)}

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{X}(c)$	☐	☐	☐	
$((a)\ \mathbf{U}\ (c))$	☐	☐	☐	
$\mathbf{AG}(b \wedge c)$	☐	☐	☐	
$\mathbf{EX}(c)$	☐	☐	☐	
$\mathbf{E}[(a \wedge c)\ \mathbf{U}\ (c)]$	☐	☐	☐	

Solved by student

phi	CTL	LTL	CTL*	States
X(c)		x	x	s1, s2, s3
((a) U (c))		x	x	s0,s1,s4,s3
AG(b ^ c)	x		x	-
EX(c)	x		x	s1,s2,s3

E[(a ^ c) U (c)]	**x**		**x**	**s0,s4,s3**

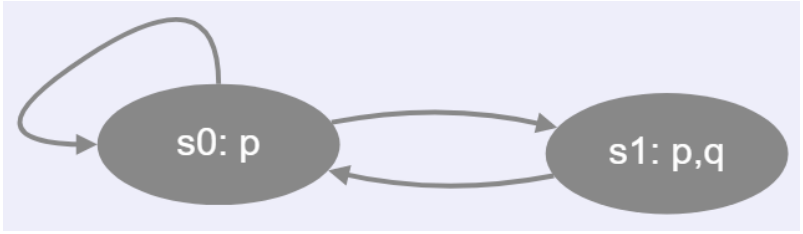
(c) **LTL tautologies**

Prove or disprove (e.g. by providing a counter-example) the following CTL formula:

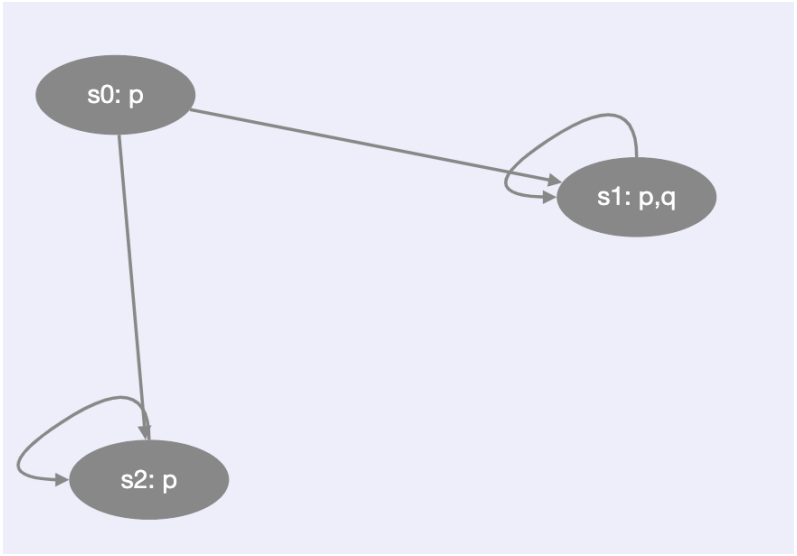
$$(\mathbf{AG}p \wedge \mathbf{EF}q) \Rightarrow (\mathbf{AF}(p \Rightarrow q))$$

Solved by student:

Counterexample:



F1.



Exam 15.03.2019 (fmi192.pdf)

Block 1

(a)

1.) Consider the following decision problem:

> **AT-MOST-ONE-HALTS (AMOH)**
>
> INSTANCE: A tuple (Π_1, Π_2, I), where Π_1, Π_2 are programs that take a string as input, and I is a string.
> QUESTION: Does either $\Pi_1(I)$ halt, $\Pi_2(I)$ halt, or none of the two halt?

(a) By providing a suitable many-one reduction from the **CO-HALTING** problem, prove that **AT-MOST-ONE-HALTS** is undecidable. Recall that **CO-HALTING** is given as follows

> **CO-HALTING**
> INSTANCE: A (source code of a) program Π, an input string I.
> QUESTION: Does $\Pi(I)$ run forever (i.e., does Π not terminate on I)?

(10 points)

concept solution by Student (pls check)
Everything is an endless loop:

We construct P1 = P2 = P and input I.
Further P' as instance of AMOH:

```
P'(String I){
        P1(I)
        P2(I)
        }
```

=> assume P is pos instance of co-Halting ie. P does not halt on I. As P1 = P2 = P nothing halts. Therefore it is a positive instance of AMOH (would not matter if P2 halts or not; because P1 does not therefore we fulfilled the properties of AMOH)
<= P1 and P2 pos instance of AMOH ie. neither P1 nor P2 halt. As P1 = P2 = P we know that P does not halt on I → positive instance of co-Halting

Other solution by Student (pls check)
Arbitrary Instance of CO-HALTING (P,I)

Construct Instance of AMOH with (P1,P2,I') where P1 = P, I'=I and P2 is a program that halts immediately
=> assume P is pos instance of CO-HALTING, i.e. P does not halt on I. As P1=P and I'=I, also P1 does not halt on I'. Therefore this is also a positive instance of AMOH (P2 does not matter, since it is sufficient that P1 does not halt)
<= assume P is pos instance of AMOH, i.e. P1 and/or P2 does not halt on I'. Since by construction P2 always halts, P1 must not halt on I'. As P1=P and I'=I, also P does not halt on I. Therefore this is also a positive instance of CO-HALTING

(b) Recall that **CO-HALTING** is not even semi-decidable. Given a reduction from **CO-HALTING** to **AT-MOST-ONE-HALTS**, what can we say about semi-decidability of **AT-MOST-ONE-HALTS**?

Solved by Student: I would say it is not semi-decidable; as we can not build an interpreter...

Block 2

2.) (a) Show that $a[i] \doteq e \rightarrow a\langle i \triangleleft e\rangle \doteq a$ is $\mathcal{T}_A^{=}$-valid using the semantic argument method from the lecture. Besides the equality axioms, you have the following ones for the arrays.

i. $\forall a, i, j \ \left(i \doteq j \rightarrow a[i] \doteq a[j]\right)$ (array congruence)
ii. $\forall a, v, i, j \ \left(i \doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq v\right)$ (read-over-write 1)
iii. $\forall a, v, i, j \ \left(i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq a[j]\right)$ (read-over-write 2)
iv. $\forall a, b \left(\forall j \ \left(a[j] \doteq b[j]\right) \leftrightarrow a \doteq b\right)$ (extensionality)

Please be precise and indicate exactly why proof lines follow from some other(s).

a) See Block 2 of fmi 191

(b) First define the concept of a $\mathcal{T}$-interpretation. Then use it to define the following:
i. the $\mathcal{T}$-satisfiability of a formula;
ii. the $\mathcal{T}$-validity of a formula.
Additionally define the completeness of a theory $\mathcal{T}$. **(4 points)**

(b) See definition in slides:
Given a Theory T = (Σ,A):
T-Interpretation: A T-interpretation I is a Σ-structure which satisfies T's axioms ie.: I ⊨ φ for all φ ∈ A
T-Satisfiability: A Σ-formula φ is satisfiable in the theory T or T-satisfiable, if some T-interpretation satisfies φ
T-Validity: if every T-interpretation satisfies φ. Notation T ⊨ φ

Completeness: if for every closed Σ-formula φ: T ⊨ φ or T ⊨ ⌐φ ie. Presburger arithmetic (incomplete example: group theory)

Block 3

,

Invariant: a = b*b (for short b^2)
As b[n] = 0 + n → b = n apply to postcondition→ $a = b^2$
P = {n>0}
b = {b != n}
B = {$a = n^2$ }

First test if invariant is strong enough: Precondition ⇒ I
n>0 ⇒ $a=b^2$
Using a0 and b0:
n>0 ⇒ $0 = 0^2$
(ANYTHING implying true is true)

Show partial correctness using wlp/vc:
(1) I & !b ⇒ B &
(2) I & b ⇒ wlp(p,I) & (3) VC(p,I)

Implication (1):
$a=b^2$ & b=n ⇒ $a=n^2$
Tautology this is true;

Implication (2)
wlp(p,I): use values of assignment in I →
$a+2b+1 = (b+1)(b+1)$ //rhs is a binomial formula
$a+2b+1 = b^2 +2b+1$ | -2b |-1
$a=b^2$
= wlp(p,I)

I & b!=n ⇒ $a=b^2$
$a=b^2$ & b!= n ⇒ $a=b^2$
Tautology this is true;

Part (3)
VC(p,I) = true

(1) & (2) & (3) = true & true & true
This is valid with rp to partial correctness;

(b) Let p be the program from exercise 3a. Is the Hoare triple $[true]\ p\ [a = n * n]$ valid with respect to total correctness? If yes, prove its validity using the Hoare calculus. Otherwise, provide a counterexample, that is, a state that does not satisfy the correctness assertion.

HT not valid.
Counterexample: set n < 0; we will never leave the loop (because b starts at b0 := 0;) ;
it will not terminate

(c) Is the following Hoare triple valid with respect to total correctness? If yes, prove its validity using the Hoare calculus. Otherwise, provide a counterexample, that is, a state that does not satisfy the correctness assertion.

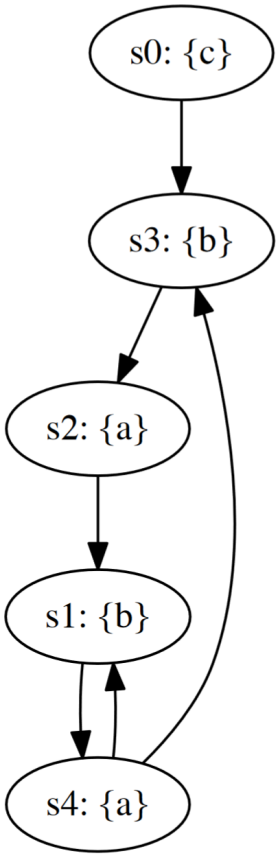
$$[n \geq 0]\ \textbf{if}\ n > 0\ \textbf{then}\ m := 2 * n\ \textbf{else abort endif}\ [m = 2 * n]$$

HT not valid.
Counterexample: n = 0; we enter the else branch, which performs abort; then the postcondition is not satisfied → not a valid HT w.r.t. total correctness

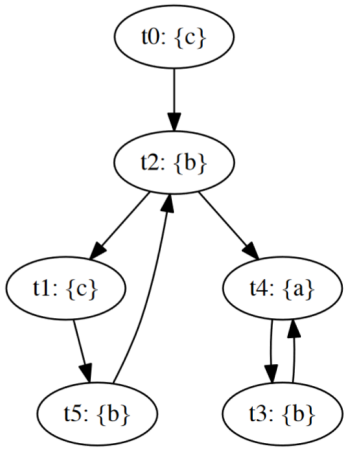
Block 4

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



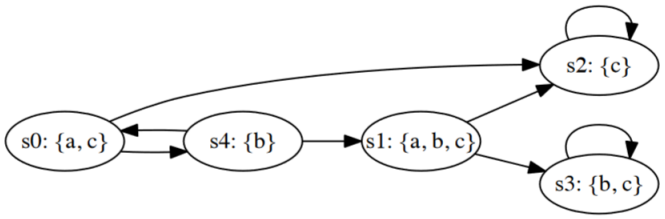
Kripke structure M_2:



Solved by student:

H = { (s0,t0) , (s3, t2) , (s2,t4) , (s1,t3) , (s4,t4) , (s3, t3) }

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{G}(c)$	☐	☐	☐	
$\mathbf{F}(a)$	☐	☐	☐	
$\mathbf{AF}(b)$	☐	☐	☐	
$\mathbf{EX}(a \wedge b)$	☐	☐	☐	
$\mathbf{E}[(a \wedge b)\ \mathbf{U}\ (a)]$	☐	☐	☐	

Solved by student

phi	CTL	LTL	CTL*	States
G(c)		x	x	s2,s1,s3
F(a)		x	x	s0,s4,s1
AF(b)	x		x	s4,s1,s3
EX(a ^ b)	x		x	s4
E[(a ^ b) U (a)]	x		x	s0,s1

(c) **LTL tautologies**

Prove or disprove (e.g. by providing a counter-example) the following LTL formulas:

i. $p \ \mathbf{U} \ (\neg q \ \mathbf{U} \ r) \Leftrightarrow \neg(p \ \mathbf{U} \ (q \ \mathbf{U} \ r))$

ii. $\mathbf{GF}p \Rightarrow \mathbf{G}(\neg p \ \mathbf{U} \ p)$

(i) would say this is **not a tautology**; as the first one basically says we have p until we have anything else than q until r… the negation in front of the whole term changes the whole expression;
counterexample : state p with selfloop -> r

(ii) this is a **tauology**: for every i at pi[i] there exists a k >= 0 & i<= k where pi = p ⇔ For all paths there exists a k st pi[k] is p and for all i s.t 0<i<k pi[i] = !p;

New try on a proof:
“sat” is used for satisfy
Assume M sat GFp and every path pi starting from s0, pi=s0,s1,..,sn
holds pi sat Fp, such that E i >= 0, pi(i) sat p (0)

Towards a contradiction it is assumed, that M not sat G (not p U p).
For every path pi from s0, it holds that pi not sat G (not p U p).
s.t. for all k >= i >= 0 pi(k) not sat (not p) and pi(i) not sat (p).
simplified there is no i, s.t. pi(i) sat p.
this contradicts the assumption (0).
therefore it is a tautology.

Solution above formalized:

Assume that $M, \pi \models GFp$

Proof by Contradiction:

1. $M, \pi \not\models G(\neg pUp)$
2. $M, \pi \not\models GFp$
3. $M, \pi \models GFp$

Since $\neg pUp \iff GFp$ we can rewrite 1. to 2. and because of our assumption 2. contradicts 3. Therefore $GFp \iff G(\neg pUp)$ holds.

Exam 25.01.2019 (fmi191.pdf)

Block 1

(a) Consider the following decision problem:

> **AT-LEAST-ONE-HALTS (ALOH)**
>
> INSTANCE: A tuple (Π_1, Π_2, I), where Π_1, Π_2 are programs that take a string as input, and I is a string.
>
> QUESTION: Does either $\Pi_1(I)$ halt, $\Pi_2(I)$ halt, or do both halt?

By providing a suitable reduction from the **HALTING** problem, prove that **AT-LEAST-ONE-HALTS** is undecidable.

Solved by Student (pls check)
Let (P,I) be an arbitrary instance of the Halting problem.
We construct P2 (of the input for P') in a way it will never terminate (endless loop)
Further, we set P1 = P.
FInally P' as instance of ALOH:

```
P'(){
        P(I) // I is hardcoded
        P2(I)
}
```

=> (P,I) is a positive instance of Halting ie. P halts on I. As P2 runs forever and P1=P (which halts) it is a positive instance of ALOH.

<= Assume (P1, P2, I) is a positive instance of ALOH. As we constructed P2 to never terminate, P1 must terminate on I. As P1 = P we have that (P, I) is a positive instance of Halting

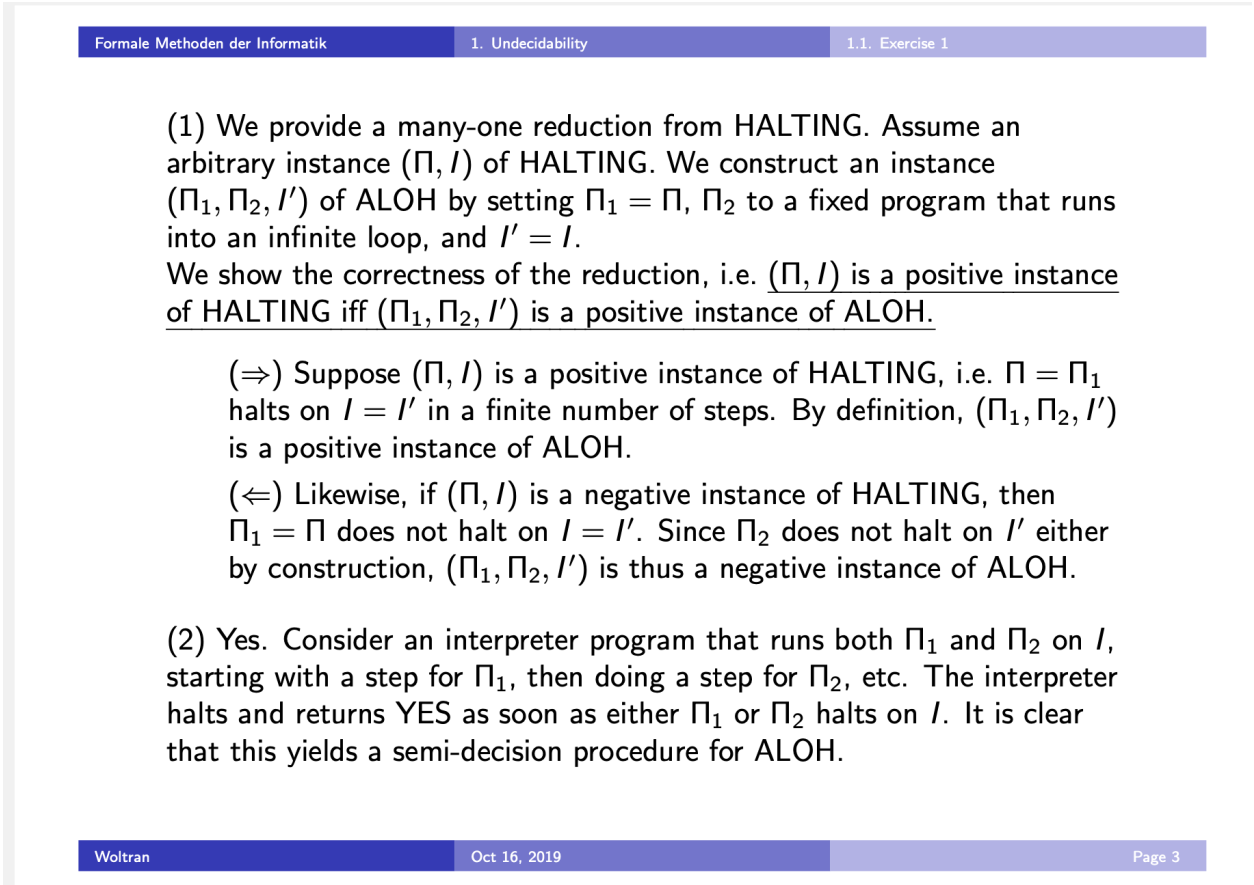
Solution (Checked from TA)

```
H (P, I) is an instance of HALTING
ALOH (P1, P2, I) is an instance of AT-LEAST-ONE-HALTS

H (P, I) {
      return ALOH (P, P, I)
}
```

Being P the same, either both halt or none, hence for (b) we have the same decidability of Halting.

> TA: [This solution] should be fine (and is probably the easiest one).



Formale Methoden der Informatik | 1. Undecidability | 1.1. Exercise 1

(1) We provide a many-one reduction from HALTING. Assume an arbitrary instance (Π, I) of HALTING. We construct an instance (Π_1, Π_2, I') of ALOH by setting $\Pi_1 = \Pi$, Π_2 to a fixed program that runs into an infinite loop, and $I' = I$.
We show the correctness of the reduction, i.e. (Π, I) is a positive instance of HALTING iff (Π_1, Π_2, I') is a positive instance of ALOH.

($\Rightarrow$) Suppose (Π, I) is a positive instance of HALTING, i.e. $\Pi = \Pi_1$ halts on $I = I'$ in a finite number of steps. By definition, (Π_1, Π_2, I') is a positive instance of ALOH.

($\Leftarrow$) Likewise, if (Π, I) is a negative instance of HALTING, then $\Pi_1 = \Pi$ does not halt on $I = I'$. Since Π_2 does not halt on I' either by construction, (Π_1, Π_2, I') is thus a negative instance of ALOH.

(2) Yes. Consider an interpreter program that runs both Π_1 and Π_2 on I, starting with a step for Π_1, then doing a step for Π_2, etc. The interpreter halts and returns YES as soon as either Π_1 or Π_2 halts on I. It is clear that this yields a semi-decision procedure for ALOH.

Woltran | Oct 16, 2019 | Page 3

(b) Is **AT-LEAST-ONE-HALTS** semi-decidable? Explain your answer.

Solved by Student:
Yes. It is possible to build an interpreter that terminates as soon as P1 or P2 terminates (whichever is first, or both at the same time). By simulating P1/P2 alternately step-by-step. ~~No. For semi-decidability we need ALOH to terminate on all positive instances and it can terminate on negative instances but doesn't have to. If only P1 of ALOH halts and not P2, it's a positive instance of ALOH but doesn't terminate.~~ This is wrong, ALOH can terminate when P1 halts.

Block 2

2.) (a) Show that $a[i] \doteq e \rightarrow a\langle i \triangleleft e\rangle \doteq a$ is $\mathcal{T}_A^{=}$-valid using the semantic argument method from the lecture. Besides the equality axioms, you have the following ones for the arrays.

i. $\forall a, i, j \ \big(i \doteq j \rightarrow a[i] \doteq a[j]\big)$ (array congruence)

ii. $\forall a, v, i, j \ \big(i \doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq v\big)$ (read-over-write 1)

iii. $\forall a, v, i, j \ \big(i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq a[j]\big)$ (read-over-write 2)

iv. $\forall a, b \, \big(\forall j \ \big(a[j] \doteq b[j]\big) \leftrightarrow a \doteq b\big)$ (extensionality)

Besides the axioms, you are allowed to use *modus ponens* and *modus tollens*.

Solved by Student

φ = a[i] = e → a(i <e) = a
To proof that φ is T_A=-valid we assume an interpretation I where φ is not valid towards a contradiction:

~~1. I ⊭ φ assumption~~
~~2. I ⊨ a[i] = e 1, semantics of →~~
~~3. I ⊭ a(i<e) = a 1, semantics of →~~
~~4. I ⊭ ∀ja(i<e)[j] = a[j] 3, apply extensionality~~
~~5. I ⊭ a(i<e)[i] = a[i] 4, due to the ∀we only need to find one case f. contradiction: $j \in U$; I1 = I {i← j} - aka i = j~~
~~6. I1 ⊭ e = a[i] 5, read-over-write 1~~
~~7. I1 ⊭ a[i] = e 6, transitivity symmetry~~
~~8. Contradiction 7 & 2~~

Another solution by student
φ = a[i] = e → a(i <e) = a

We prove that φ is T_A=-valid by contradiction. We assume φ does not hold, thus LHS (a[i] = e) holds and RHS (a(i <e) = a) doesn't hold.

1. I $\models \varphi$	assumption
2. I $\vdash$ a[i] = e	1, semantics of $\rightarrow$
3. I $\models$ a(i<e) = a	1, semantics of $\rightarrow$
4. I $\models \forall j$ (a(i<e)[j] = a[j])	3, apply extensionality

5. I1 $\models$ a(i<e)[k] = a[k] 4, $\forall$, for some k $\ni$ U; I1 = I {j$\leftarrow$ k} (informal explanation: we get rid of "for all j", k is some concrete number. If it works for this concrete case, it should work for all j as well.)

6. I1 $\vdash$ a(i<e)[k] $\neq$ a[k]	5, semantics of negation
7. I1 $\vdash$ i = k	6, read-over write 2 and modus tollens
8. I1 $\vdash$ a[i] = a[k]	7, array congruence
9. I1 $\vdash$ a(i<e)[k] = e	7, read-over-write 1
10. I1 $\vdash$ e = a[k]	2, 8, symmetry and transitivity
11. I1 $\vdash$ a(i<e)[k] = a[k]	9,10, transitivity
12. I1 $\vdash \bot$	6,11, contradiction

(b) Let $f(x_1, x_2) = x_1 \oplus x_2$ and $f(x_1, \ldots, x_{n+1}) = f(x_1, \ldots, x_n) \oplus x_{n+1}$ for $n > 2$.
(i) What is the number of clauses in a satisfiability-equivalent CNF version of $f(x_1, \ldots, x_n)$.
(ii) What is the number of clauses in a logically equivalent CNF version of $f(x_1, \ldots, x_n)$.
Explain and justify your answers in detail. **(3 points)**

Tseitin Transformation - see sample solution exercises of Block 2 SS19 Ex. 2.3:
https://vowi.fsinf.at/images/6/69/TU_Wien-Formale_Methoden_der_Informatik_VU_%28Egly%29_-_Fminf-ex-block2_sol.pdf

So, i) We get two clauses from encoding equivalences of the form li $\leftrightarrow$ xi , four clauses from encoding an XOR operator, and one clause as the final clause representing the output of the function, i.e., 1 + 2 × n + 4 × (n − 1) = 6n − 3.
ii) ~~x1 xor x2 $\Leftrightarrow$ (x1 v x2) ^ (x1 v -x2), so for every new term x we get 2 clauses (x ^ -x) -> 2*n (not sure though)~~
The formula is exponential in size. 2^(n)
$\rightarrow$ Actually it is 2^(n-1)
https://www.wolframalpha.com/input/?i=a++XOR+b++XOR+c+XOR+d

Block 3

3.) (a) Consider the program p:

$r := 0;$
$i := 0;$
$s := 1;$
while $i < n$ **do**
  $r := r + s;$
  $s := s + 2;$
  $i := i + 1$
od

Let $p' =$ **while** $i < n$ **do** ... **od** be the while loop of the program p. Give an inductive invariant for p', such that the Hoare triple $\{true\}\ p'\ \{true\}$ is valid with respect to partial correctness. **(2 points)**

3.a.)
Trivial solution: `I = true;`
~~Other solution: I = i ≤ n~~

(Or use invariant from (b))

(b) Let p be the program from exercise 3a. Show that the Hoare triple $\{n \geq 0\}\ p\ \{r = n*n\}$ is valid with respect to partial correctness. The program p is given again for your convenience:

$r := 0;$
$i := 0;$
$s := 1;$
while $i < n$ **do**
  $r := r + s;$
  $s := s + 2;$
  $i := i + 1$
od

3.b.)
Solved by Student (please check)
Invariant: i = i0 + n → i = n, s = s0 + 2n → s = 1+2n → s = 1+2i

For the r part I just wrote down the first few loop runs by hand and then quickly found $r = i^2$. So the full Invariant is $s = 1 + 2i \ \& \ r = i * i \ \& \ i <= n$ (The last part is derived from the loop condition and the fact that each loop increments by 1)

Annotation calculus (This is the old method by Prof. Salzer, I believe there is a new one since WS2018 but I haven't looked into that yet):

{n >=0}
F10: {Inv}[s/1][i/0][r/0]
r:=0
F9: {Inv}[s/1][i/0]
i:=0
F8: {Inv}[s/1]
s:= 1
F1: {Inv}
While i < n do
  F2: {Inv & i < n & t = t0}
  F7: {Inv & t < t0}[i/i+1][s/s+2][r/r+s]
  r = r + s
  F6: {Inv & t < t0}[i/i+1][s/s+2]
  s = s + 2
  F5: {Inv & t < t0}[i/i+1]
  i = i + 1
  F3: {Inv & t < t0}
od
F4: {Inv & !p}
{r = n * n}

Now we need to show:

Pre -> F10: { n >= 0 } → {s = 1 + 2i & r = i * i & i <= n}[s/1][i/0][r/0]

F2 -> F7: { s = 1 + 2i & r = i * i & i <= n & i < n & t = t0} → { s = 1 + 2i & r = i * i & i <= n & t < t0}[i/i+1][s/s+2][r/r+s]

F4 -> Post: { s = 1 + 2i & r = i * i & i <= n & i >= n } → {r = n * n}

{ n >= 0 } → {s = 1 + 2i & r = i * i & i <= n}[s/1][i/0][r/0]
{ n >= 0 } → {1 = 1 + 2*0 & 0 = 0 * 0 & 0 <= n}
{ n >= 0 } → {0 <= n} Tautologie

{ s = 1 + 2i & r = i * i & i <= n & i < n & t = t0} → { s = 1 + 2i & r = i * i & i <= n & t < t0}[i/i+1][s/s+2][r/r+s]
Skip t0 (not sure, but I believe it's only necessary for absolute correctness)
{ s = 1 + 2i & r = i * i & i <= n & i < n} → { s + 2 = 1 + 2 (i+1) & r + s= (i+1) * (i+1) & (i+1) <= n}

{ s = 1 + 2i & r = i * i & i <= n & i < n} → { s + 2 = 1 + 2i + 2 & r + s= i² +2i +1 & i < n}
{ s = 1 + 2i & r = i * i & i <= n & i < n} → { s = 1 + 2i & r + 1 + 2i = i² +2i +1 & i < n}
{ s = 1 + 2i & r = i * i & i <= n & i < n} → { s = 1 + 2i & r = i² & i < n} Tautologie

{ s = 1 + 2i & r = i * i & i <= n & i >= n } → { r = n * n }
i <= n & i >= n <==> i = n
{ s = 1 + 2i & r = i * i & i = n } → { r = n * n }
{ s = 1 + 2n & r = n * n & i = n } → { r = n * n } Tautologie

Alternative solution by student (new semantics: wlp & VC):

Note: Using wlp you have to memorize the implications for the while loop; for partial correctness it is wlp and for total wp & the ones for total correctness include the use of a variant.
For partial correctness no variant needed.

I… Invariant; b… loop condition; B...Postcondition; A… Precondition

Here: partial correctness.

Step 1: Finding invariant
Write everything in index notation:
R[n+1] = r[n] + s
S[n+1] = s[n] +2
I[n+1[] = i [n] + 1

So how to get to the n?
R[n] = r0 + n*s
S[n] = s0 + 2n
I[n] = i0 + n

So we know i[n] = n as i0 = 0
So we can put “i” in s[n] = 1+2i
And from the postcondition we know r = n²
So we can use the “i” again in the postcondition and the whole invariant would be:
r=i² & s = 1+2i

Step 2: VC for while loop
…

Alternative invariant calculation (by another student):

(Note that the variable n used here is not the same as the variable n in the program. We should probably pick another auxiliary variable to model the program counter, but I stayed with n to stay consistent with the other approach.)

I think the problem in the above approach is that r[n+1] uses the variable s incorrectly.
Instead of:
r[n+1] = r[n] + s
It should be:
r[n+1] = r[n] + **s[n]**

Note that I used s[n] instead of s, since s is also updated by the loop and used by r before (!) it's incremented.

To now get a closed form of r[n+1] we **can't just use *n*s like above**, since s does not stay the same for all iterations of the loop. Instead we need to use the **sum of all s[j] for *j=1 to n-1***.
We use n-1 because r[n] uses the value of s of the previous iteration, i.e., n-1, as mentioned above.

With s[n] = 1 + 2n we get:
r[n] = r[0] + s[0] + sum (j=1 to n-1) s[j]
r[n] = r[0] + s[0] + (n-1) + 2*(n-1)*(n-1+1)/2

Which simplifies to:
r[n] = 0 + 1 - 1 + n^2 = n^2
r = n^2

Substituting i = n:
r = i^2

A less formal way to infer this invariant is to just write down the values of each variable for multiple iterations, to quickly see that r is always the square of i.

With this invariant you still have to prove if the precondition establishes the invariant AND that the invariant & !b is strong enough to establish the postcondition.
By looking at the loop condition i<n we can see that it might be necessary to strengthen the invariant to **r=i^2 & s=1+2i & i<=n**
Why? Invariant needs to stay true before, within and after the loop, the loop condition implies, that after the last run i=n as i<n requires the program to stop as soon as i is equal to n
(if you cannot solve I&!b=>B, check if you can still strengthen the invariant to get to the solution)

Step 2: Precondition establishes invariant:
n>=0 => r=i² & s=1+2i & i<= n

Put in initial values:
n>=0 => 0 = 0² & 1 = 1+2*0 & 0<= n
This is trivially true.

Step 3: wlp & VC
To prove the partial correctness of a while loop, you need the VC and wlp rules from the slides. Just recursively apply the rules.
For partial correctness VC(while loop):
(1) I & !b => B &
(2) I & b => wlp(p,I) &
(3) VC(p, I)

(1)
r=i² &s = 1+2i & i<=n & !(i<n) => r = n² | apply negation
R = i² & s= 1+2i & i<=N & i>=n => r = n² | i<= n & i>= n is same as i=n
R = i² & s=1+2i & i =n => r=n²
Tautology this is true

(2)
r=i² & s=1+2i & i<=n & i<n => wlp(p,I) |lhs simplified
R = i² & s=1+2i & i<n => wlp(p,I) |

Calculate wlp(p,I):
(i.e. just put in the values they would have in the next iteration into the Invariant:)
r-> r+s
i-> i+1
s-> s+2
(r+s) = (i+1)² & (s+2) = 1+2(i+1) & (i+1)<=n |crazy math stuff aka algebra

R+s = i² + 2i + 1 & s+2 = 1+2i +2&(i+1)<=n | for easier reading i will split this

wlp(p,I) =
R+s = i² + 2i +1 &=
S+2 = 1+ 2i+2 &
(i+1)<=n

More algebra:
-2 on both sides of s, we get s = 1+ 2i which we can put into the equation for r, then -s on both sides and we get:

wlp(p,I) =
R = i² &
S = 1+2i &
(i+1) <= n
R & s part are a tautology in rhs and lhs
And if we have i<n on one side and (i+1) <=n on the other side, this is trivially true due the properties of integers (if it was less than n before adding +1, it now has to be either equal or still less than n)

(3) VC(p,I)
According to the rules, this is true. If p would contain more than assignments, we would have to do the whole procedure again; (same in wlp(p,I))

If all evaluates to true, the program is partial correct.
(For total correctness (TC), we would have to check the appropriate VC for TC and include variant rules.)

(c) Is the following Hoare triple valid with respect to total correctness? If yes, prove its validity using the Hoare calculus. Otherwise, provide a counterexample, that is, a state that does not satisfy the correctness assertion.

$$[n \geq 0]\ \textbf{if}\ n = 0\ \textbf{then abort else}\ m := n\ [m = n]$$

3.c.)
Solved by Student:
Nope: we can enter the abort if n=0 due to the precondition n>=0.

Solved by (other) Student:
Counterexample:
Say m:=1 and n:=0. ~~Then the if branch has "abort", so the program terminates, and after it, the postcondition [m=n] would be [1=0], which is false~~. Therefore, this HT is not valid w.r.t total correctness.

Block 4

4.) (a) Show that the simulation relation for Kripke structures is transitive, i.e., if $A \leq B$ and $B \leq C$ for some Kripke structures A, B, C, then $A \leq C$.

(5 points)

Solved by student:

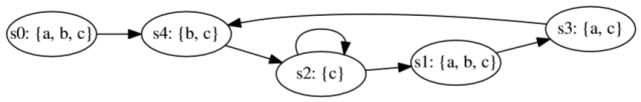
As we know, for every state $a \in A$ there is $(a, b) \in R$, $with\ A \leq_R B$.

Further, for every state $b \in B$ there is $(b, c) \in R'$ with $B \leq_{R'} C$.

Therefore, we can construct R" so that $\forall (a, b) \in R$ and $\forall\ b = b'$, $(b', c') \in R'$we add $(a, c') \in R''$. (Note that this means we could add multiple (a,c) for one (a,b) relation, as there could be (a,b1), (a,b2), … contained in R.)

Thus the simulation relation is transitive.

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

Recall that for an LTL formula φ it holds that $M, s \models \varphi \iff M, s \models \mathbf{A}\varphi$.

φ	CTL	States s_i
$\mathbf{X}(a)$	☐	
$((b)\ \mathbf{U}\ (a))$	☐	
$\mathbf{AF}(b \wedge c)$	☐	
$\mathbf{A}[(a)\ \mathbf{U}\ (b)]$	☐	
$\mathbf{EF}(c)$	☐	

b) **Solved by student** (please check)

phi	CTL	States
X(a)		s1
b U a		s0, s1, s3

AF(b ^ c)	x	s0, s1, s3, s4
A[a U b]	x	s0, s4, s1, s3
EF(c)	x	s0, s1, s2, s3, s4

(c) **LTL tautologies**

Prove or disprove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π:

i.

$$(\mathbf{GF}p) \wedge (\mathbf{GF}q) \Rightarrow \mathbf{G}(p\ \mathbf{U}\ q)$$

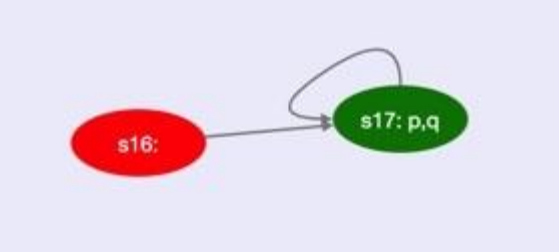
ii.

$$((\mathbf{G}\neg p)\ \mathbf{U}\ p) \wedge \neg p \Rightarrow (\mathbf{G}q) \vee (\neg q\ \mathbf{U}\ r)$$

Solved by student:

i. is <u>not a tautology</u>.

Counterexample (solved by student): ((GFp) and (GFq) holds, but G(pUq) does not hold):



Other counterexample (solved by student):
~~q with self-loop~~
~~(can occur on right side if q is immediately true, but cannot hold on the left side because there p has to hold in some future state for sure)~~

ii. Is a <u>tautology</u>. Proof (maybe):
((G !p) U p) and !p can never hold at the same time, because: <u>assume !p holds on every path</u>. Therefore p can not hold. This implies that ((G !p) U p) can not hold, because there must exist a k >= 0 such that M, phi(k) satisfies p. This contradicts with the assumption that !p holds on every path and therefore p cannot hold. Therefore the premise is always false. Since the premise is false, the implication is always true.

Exam 11.12.2018 (fmi186.pdf)

Block 1

(a)

INDEPENDENT DOMINATING SET (IDS)

INSTANCE: A directed graph $G = (V, E)$.

QUESTION: Does there exists a set $S \subseteq V$ of vertices, such that

(1) for each $(u, v) \in E$, $\{u, v\} \not\subseteq S$;

(2) for each $v \in V$ either $v \in S$ or there exists an $(u, v) \in E$, such that $u \in S$.

The following function f provides a polynomial-time many-one reduction from **IDS** to **SAT**: for a directed graph $G = (V, E)$, let

$$f(G) = \bigwedge_{(u,v)\in E} (\neg x_u \vee \neg x_v) \wedge \bigwedge_{v\in V} (x_v \vee \bigvee_{(u,v)\in E} x_u).$$

It holds that G is a yes-instance of **IDS** $\iff$ $f(G)$ is a yes-instance of **SAT**. Prove the $\implies$ direction of the claim.

(1.a.) Suppose G=(V,E) has an IDS S,V has i vertices |V|=i. We define a truth assignment T by setting the vertices in S to true and the vertices vi ∈{V/S} (i.e. the set V without S) to false, so that every vertex in the set V can only be signed to true or false, but not both.
To show that f(G) is evaluated true under T, it's sufficient to say the sub-clauses evaluate to true under T.
1. **if (u,v)** ∈**E**, ¬(xu∧xv), which means they must not belong to the S at same time. According to the definition of the S and the construction of the T this must be true.
2. **all vi** ∈**V**,
case 1: if there is no edge connected to it, vi must be in S. Then vi is signed with true, because the (clause?) and the whole formula is evaluated to be true.
case 2: if there are some edges connected to it, vi must belong to S or the vertices that connect with it must belong to S.
Both cases correspond to the definition of S.

(1.b.)

Given that **SAT** is NP-complete, what can be said about the complexity of **IDS** from the above reduction? NP-hardness of **IDS**, NP-membership of **IDS**, neither of them, or both (NP-completeness of **IDS**)

(1.b.) Solved by student

I would argue that IDS lies in NP, but is not necessarily NP-hard. For NP-hard we would have to show that any problem in NP is reducible to IDS which we didn't. However, as IDS is reducible to SAT it means we have an algorithm within NP (the SAT algorithm) which is able to solve it. Therefore it can't be more difficult than NP as we could always reduce it to SAT and then use the SAT algorithm.

More detailed explanation from other student

From the slides: "If this problem solving strategy works, we say that problem A is reduced to problem B. => Problem A is not harder than problem B."

So yeah it should be in NP, since a problem in P is also in NP. All problems in P are considered to be in NP, but NOT the other way around.

But we can NOT show NP hardness with that reduction, we would need a reduction from SAT to IDS to show IDS to be NP hard. (because SAT is NP complete)

Block 2

(a) The topic of this exercise is *translation validation* (discussed in the fifth lecture of the second block). Given a statement in a source program of the form

$$z = (y_1 * x) + (y_2 * x) \tag{S}$$

and the result of the compiler optimization of the form

$$u_1 = y_1 + y_2, \quad u_2 = u_1 * x, \tag{O}$$

the goal is to check the correctness of the translation.

i. Formulate the verification condition.

_____ (VC)

ii. Formulate the abstract verification condition (using uninterpreted functions).

_____ (AVC)

iii. Prove the correctness of the translation process (using the semantic proof method from the third lecture (on First-order Logic and Theories), or present a counter-example for (AVC). The symbols u_1, u_2, x, y_1, y_2, z are all free variables!
Hint: Do **not** use Ackermann!

(6 points)

a)

i) $u_1 = y_1 + y_2 \ \wedge \ u_2 = u_1 * x \wedge z = u_2 \rightarrow z = (y_1 * x) + (y_2 * x)$

ii) $u_1 = P(y_1, y_2) \ \wedge u_2 = M(u_1, x) \ \wedge z = u_2 \rightarrow z = P(M(y_1, x), M(y_2, x))$

iii) The EUF formula is not valid. The equivalence relies on the distributivity of + and * which is lost as soon as UFs are used. The following Interpretation provides a **counterexample:**
Let $U = \{1, 2\}$be a domain
Let $f$be a function with $f(x, y) = 1$for every x and y
Let g be a function with $g(x, y) = 2$for every x and y
Let αbe a variable assignment with $\alpha(x) = 1$for every variable x
Let I(P) be f and I(M) be g
Then the formula is not satisfied under the interpretation $J =< U, I, \alpha >$

Proof:

$J(P(y_1, y_2)) = J(P)(J(y_1)J(y_2)) = f(\alpha(y_1), \alpha(y_2)) = f(1, 1) = 1$

$J(M(u_1, x)) = J(M)(J(u_1)J(x)) = g(\alpha(u_1), \alpha(x)) = g(1, 1) = 2$

$J(P(M(y_1, x), M(y_2, x))) = J(P)(J(M(y_1, x)), J(M(y_2, x))) =$

$p(J(M)(J(y_1), J(x)), J(M)(J(y_2), J(x))) = f(g(\alpha(y_1), \alpha(x)), g(\alpha(y_2), \alpha(x))) = f(g(1, 1), g(1, 1)) = 1$

Therefore

$u_1 = 1 \wedge u_2 = 2 \wedge z = u_2 \rightarrow z = 1$

This leads to a contradiction of the transitivity axiom

(b) Let φ be the first-order formula

$$\forall x \forall y \left[\left(r(x,y) \rightarrow (p(x) \rightarrow p(y)) \right) \wedge \left(r(x,y) \rightarrow (p(y) \rightarrow p(x)) \right) \right] .$$

i. Is φ valid? If yes, present a proof. If no, give a counter-example and prove that it falsifies φ.

ii. Replace r in φ by $\doteq$ (equality) resulting in ψ. Is ψ E-valid? Argue formally!

Same as in fmi196 Block 2 and fmi152 Block 2

Block 3

(a) Show that the assertion

$\{F \ : \ y > 0\}$

$x := 0;$

$i := y;$

while $i > 0$ **do**

$x := x + 2;$

$i := i - 1$

od

$\{G \ : \ x = 2 * y\}$

is correct with respect to partial correctness.

Solved by student (check please):

$wlp(x := 0;\ i := y;\ while\ i > 0\ do\ x := x + 2;\ i := i - 1,\ x = 2 * y)$

$= wlp(x := 0;\ i := y,\ wlp(while\ i > 0\ do\ x := x + 2;\ i := i - 1,\ x = 2 * y))$

$wlp(while\ i > 0\ do\ x := x + 2;\ i := i - 1,\ x = 2 * y) = \exists j\ (j > 0 \wedge F_j)$

F_0: $i \leq 0 \ \wedge \ x = 2y$

F_1: $i > 0 \ \wedge \ wlp(x := x + 2;\ i := i - 1;\ x = 2y)$

$= i > 0 \ \wedge \ x = 2y\,[i/i - 1][x/x + 2]$

$= i > 0 \ \wedge \ x = 2y - 2$

F_j: $\ i > 0 \ \wedge \ x = 2y - 2j$

F_{j+1}: $i > 0 \ \wedge \ wlp(x := x + 2;\ i := i - 1;\ x = 2y - 2j)$

$= i > 0 \ \wedge \ x = 2y - 2j\ [i/i - 1][x/x + 2]$

$= i > 0 \ \wedge \ x = 2y - 2j - 2$

$= \exists j\ (j > 0 \ \wedge \ i > 0 \ \wedge \ x = 2y - 2j)$

$= i > 0 \ \wedge \ x = 2y - 2$

$= wlp(x := 0;\ i := y,\ i > 0 \ \wedge \ x = 2y - 2)$

$= (i > 0 \ \wedge \ x = 2y - 2)\ [i/y][x/0]$

$= y > 0 \ \wedge \ y = 1$

Solved by student (Prof. Kovac new approach)

Check via VC: Inv: i>=0 ^ x = 2 (y - i)

Solved by student:

(b) Consider the program q below.

```
y := 1;
while x > 0 do
  if y > 0 then
    x := x − 1;
    y := y − 1
  else
    y := y + 5
od
```

Find a loop variant t that is positive at the start of each loop iteration, and strictly decreases with each loop iteration.

Solved by student:

t = 5x - (y + x + 1)

Idea towards a formal explanation (please double check):
The requirement is that $t_{n+j} - t_n > 0 \ for \ all \ j > 0$
There are two cases:

(y > 0)

In this case $x_{n+1} - x_n = 1$and $y_{n+1} - y_n = 1$. Therefore

$5x_{n+1} - (x_{n+1} + y_{n+1} + 1) - 5x_n - (x_n + y_n + 1) = 5(x_n + 1) - (x_n + 1 + y_n + 1 + 1) - 5x_n - (x_n +$

(y <= 0)

In this case $x_{n+1} - x_n = 0$and $y_{n+1} - y_n = 5$. Therefore

$5x_{n+1} - (x_{n+1} + y_{n+1} + 1) - 5x_n - (x_n + y_n + 1) = 5$

Hoare Rules for $\vdash \{A\}\ p\ \{B\}$

$$\frac{}{\{B[x/a]\}\ x := a\ \{B\}} \qquad \frac{}{\{A\}\ \textbf{skip}\ \{A\}} \qquad \frac{}{\{\text{true}\}\ \textbf{abort}\ \{B\}}$$

$$\frac{\{A\}\ p_1\ \{C\} \quad \{C\}\ p_2\ \{B\}}{\{A\}\ p_1; p_2\ \{B\}} \qquad \frac{\{A \wedge b\}\ p_1\ \{B\} \quad \{A \wedge \neg b\}\ p_2\ \{B\}}{\{A\}\ \textbf{if}\ b\ \textbf{then}\ p_1\ \textbf{else}\ p_2\ \{B\}}$$

$$\frac{???}{\{A\}\ \textbf{while}\ b\ \textbf{do}\ p\ \textbf{od}\ \{B\}}$$

$$\frac{A \Rightarrow A' \quad \{A'\}\ p\ \{B'\} \quad B' \Rightarrow B}{\{A\}\ p\ \{B\}}$$

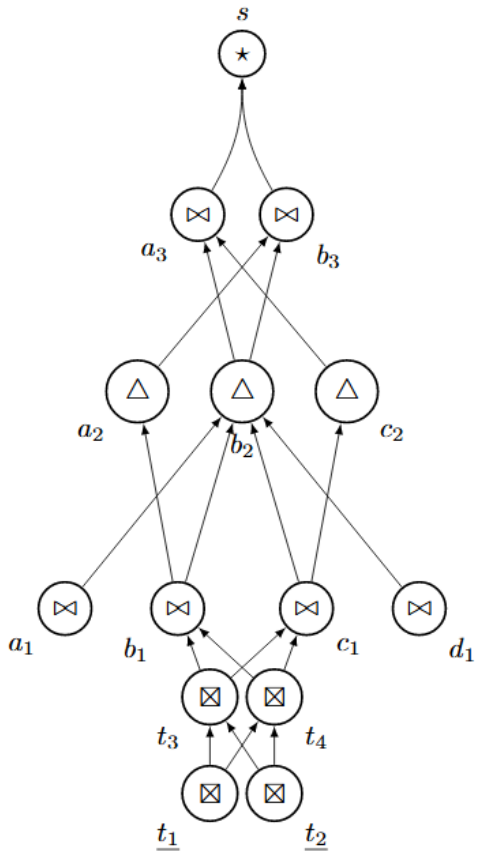
Block 4

4.) (a) The Kripke structure $M_1 = (S, S_0, R, AP, L)$ is depicted below, with $S_0 = \{t_1, t_2\}$ and $AP = \{\boxtimes, \bowtie, \triangle, \star\}$.

i. Find a Kripke structure M_2 with the smallest number of states and transitions, for which $M_1 \leq M_2$, and write down a witnessing simulation relation.

ii. Show that there exists no Kripke structure M_3 with fewer states than M_2, for which $M_1 \leq M_3$.

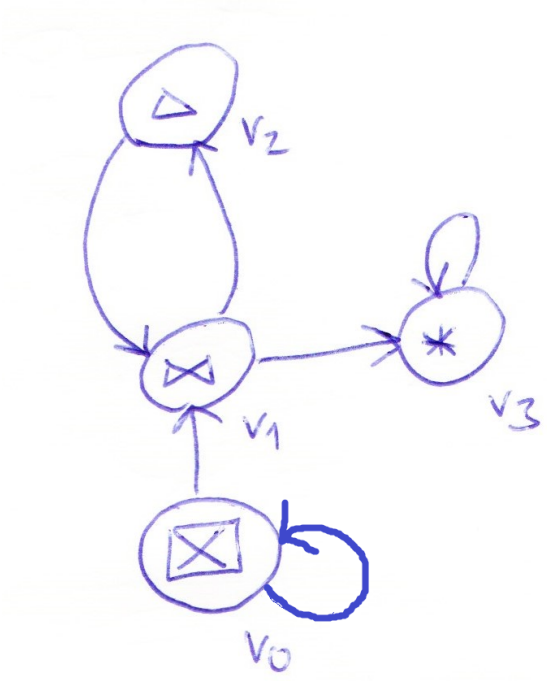
iii. Does $M_2 \leq M_1$ hold?



A christmas tree…

Confirmed by prof.:

i)
Minimal M2:



H = { (t1,r1) , (t2,r1) , (t3,r2) , (t4,r2) , (b1,r3) , (c1,r3) , (a2,r4) , (b2,r4) , (c2,r4) , (a3, r5) , (b3, r5) , (s,r6) }

H = {(t1, v0), (t2, v0), (t3, v0), (t4, v0), (b1, v1), (c1, v1), (a2, v2), (b2, v2), (c2, v2), (a3, v1), (b3, v1), (s, v3)}. You can also add (a1, v1) and (d1, v1) to H, but it is not necessary because these states are not reachable in M1. You can add them because the exercise does not require H to be minimal.

ii) There are no state iterations in M1 so the minimum height of the tree structure in M1 is the shortest path between the nodes t1 and s which is 6.
The amount of nodes of M2 is 6, so there exists no smaller path for M2.

iii) Let M3 be an arbitrary Kripke structure such that |S3| < |S2|, where S3 denotes the states in M3 and S2 denotes the states in M2. Since |S2| = 4, |S3| is at most 3. Because |AP| = 4 in M1 , not all AP of M1 are reachable in M3. Therefore M3 cannot simulate M1.

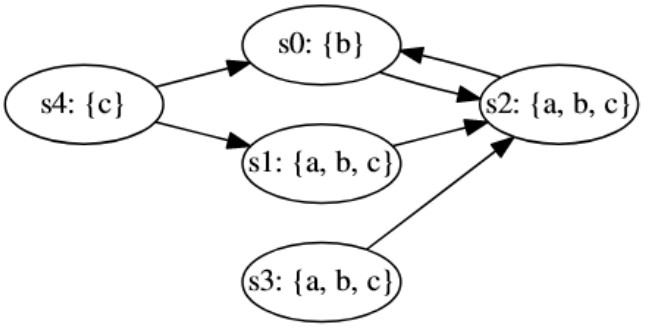
iii) solved by student

No, because the states a1 and d1 are not simulated in M2.

No, because (t4, v1)

Solved by other student:
Counter example: v0*

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, and
ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

Recall that for an LTL formula φ it holds that $M, s \models \varphi \iff M, s \models \mathbf{A}\varphi$.

φ	CTL	States s_i
$\mathbf{X}(a)$	☐	
$\mathbf{AG}(b \wedge c)$	☐	
$\mathbf{AX}(a \wedge c)$	☐	
$\mathbf{A}[(b)\ \mathbf{U}\ (c)]$	☐	
$\mathbf{E}[(b \wedge c)\ \mathbf{U}\ (a)]$	☐	

Confirmed by Prof.

phi	CTL	States si

X(a)		s0,s1,s3
AG(b^c)	x	-
AX(a^c)	x	s0,s1,s3
A[(b) U (c)]	x	s0,s1,s2,s3,s4
E[(b^c) U (a)]	x	s1,s2,s3

(c) **LTL tautologies**

Prove or disprove that the following formulas are tautologies, i.e., they hold for every Kripke structure M and every path π:

i.

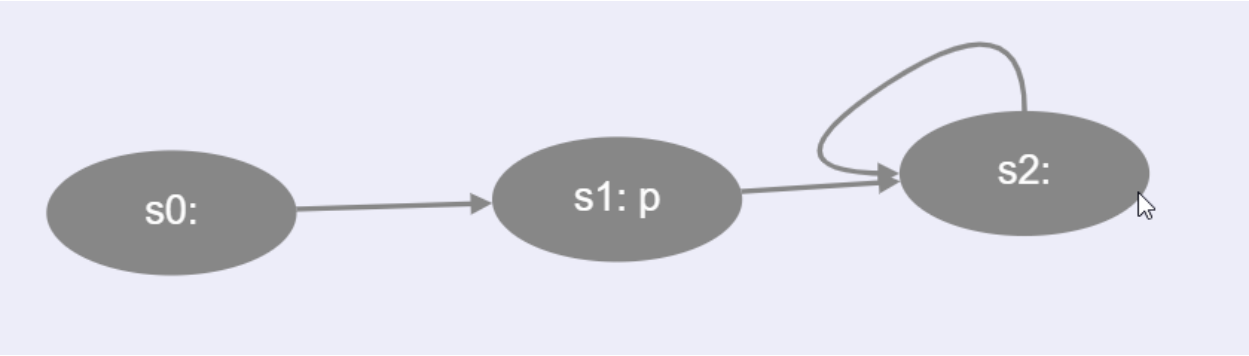
$$\mathbf{XFX}p \iff \mathbf{XF}p$$

ii.

$$(\mathbf{G}\neg p)\ \mathbf{U}\ p \iff p$$

Solved by student:

1. No, the following Kripke structure is a **counterexample**



Way to the solution:

XFp means $for\ all\ paths\ \pi.\ \pi^1 \mid = Fp$which means

$all\ paths\ \pi.\ there\ exists\ a\ k \geq 1\ such\ that\ \pi^k \mid = p$

FXp means $for\ all\ paths\ \pi.\ there\ exists\ a\ k \geq 0.\ \pi^k \mid = Xp$which means

$for\ all\ paths\ \pi.\ there\ exists\ a\ k \geq 0\ such\ that\ \pi^{k+1} \mid = p$

Therefore XFXp means $for\ all\ paths\ \pi.\ there\ exists\ a\ k\ \geq 1\ such\ that\ \pi^{k+1} \mid = p$
If one chooses k=1 now, the first formula is fulfilled whereas the second one is not (yet).

2. Is actually a tautology

=> direction: $G\neg p$means $for\ all\ i\ \geq 0,\ M, \pi^{i} \mid = \neg p.\ G\neg pUp$means

1. $there\ exists\ a\ k\ \geq 0\ such\ that\ \pi^{k} \mid = p$and
2. $for\ all\ 0 \leq j < k.\ M, \pi^{j}.\ for\ all\ i\ \geq 0,\ M, \pi^{i} \mid = \neg p$

any index $j \geq 0$would contradict 1.Therefore k must be 0 which means p is fulfilled everywhere

<= direction: if p is fulfilled, condition 1 of $G\neg pUp$is fulfilled for every k. Therefore there is no j<k and condition 2 is fulfilled trivially.

Exam 19.10.2018 (fmi185.pdf)

Block 1

Consider the following decision problem:

TRANSFORMED INPUT

INSTANCE: A tuple (Π_1, Π_2, I), where Π_1, Π_2 are programs and I is a string. Π_1 takes a string as input and outputs a string, and it is guaranteed that Π_1 terminates. Π_2 takes a string as input.

QUESTION: Does $\Pi_2(\Pi_1(I))$ hold, i.e. does Π_2 hold on the string that is delivered by Π_1 when called with string I?

1. By providing a suitable reduction from the **HALTING** problem, prove that the problem **TRANSFORMED INPUT** is undecidable.
2. Is **TRANSFORMED INPUT** semi-decidable? Explain your answer.

Block1:
Solution by student:

1)
Let (Π,I) be an arbitrary instance of HALTING.
We construct an instance (Π1,Π2,I') of TRANSFORMED INPUT:
Let Π2 = Π and I' = I.
Further Π1:

```
String Π1(String S) {
        return S;
}
```

Proof:
(Π,I) is a positive instance of HALTING ⇔ (Π1,Π2,I') is a positive instance of TRANSFORMED INPUT

=>: Assume Π halts on I.
Due to our construction we have that Π1(I') returns I', therefore the input of Π2 is I'.
Further, because we set Π2 = Π and I' = I we have that Π2 halts on I'.
It follows that Π2(Π1(I')) halts and (Π1, Π2, I') is a positive instance of TRANSFORMED INPUT.

<=: Assume (Π1, Π2, I') is a positive instance of TRANSFORMED INPUT, i.e. Π2(Π1(I')) halts.
Again, due to our construction we have that Π1(I') returns I' and the input of Π2 is I'.
I.e. Π2 halts on I'. Because of Π2=Π and I'=I it follows that Π halts on I.
Hence (Π, I) is a positive instance of HALTING.

2)
We create a semi-decision procedure Πint:

```
Bool Πint(String Π1, String Π2, String I) {
        String Out = simulate(Π1, I);
        simulate(Π2, Out);
        return true;
}
```

Proof that it is a correct **semi-decision procedure**:

- Case 1: Assume (Π1, Π2, I) is a pos instance of TI: Then Π2 will halt on String Out and Πint returns true.
- Case 2: Assume (Π1, Π2, I) is a neg. Instance of TI: Then Π2 will not halt on String Out and Πint will also not terminate, which is correct behavior for a semi-decision procedure.

Block 2

Exercise 2.1

The topic of this exercise is *translation validation* (discussed in the fifth lecture of the second block). Given a statement in a source program of the form

$$z = (x_1 + y_1) * (x_2 + y_2) \tag{S}$$

and the result of the compiler of the form

$$u_1 = x_1 + y_1; \quad u_2 = x_2 + y_2; \quad z = u_1 * u_2\,. \tag{O}$$

The goal is to check the correctness of the translation.

1. Formulate the verification condition.

   _____ (VC)

2. Formulate the abstract verification condition (using uninterpreted functions).

   _____ (AVC)

3. Prove the correctness of the translation process (using the semantic proof method from the third lecture (on First-order Logc and Theories), or present a counter-example for (AVC).

2.1) **status student (with doubts)**

2.1.1) z = (x1+y1) * (x2+y2) -> u1=x1+y1 ∧ u2=x2+y2 ∧ z=u1*u2

2.1.2) z = M(P(x1,y1), P(x2,y2)) -> u1=P(x1,y1) ∧ u2=P(x2,y2) ∧ z= M(u1,u2)

2.1.3)

Towards a contradiction assume there exists a T-interpretation I which does not satisfy (AVC).

1. I ⊨ AVC — assumption
2. I ⊨ z=M(P(x1,y1), P(x2,y2)) — 1. Semantics of impl
3. I ⊨ u1=P(x1,y1) ∧ u2=P(x2,y2) ∧ z=M(u1,u2) — 1. Semantics of impl
4. I ⊨ u1=P(x1,y1) — 3. Sem. of ∧
5. I ⊨ u2=P(x2,y2) — 3. Sem. of ∧
6. I ⊨ z=M(u1,u2) — 3. Sem. of ∧
7. I ⊨ z=M(P(x1,y1),P(x2,y2)) — 4+6 sym.+trans., 5+6 sym.+trans.
8. I ⊨ ⊥ — 2,7 contradiction

Therefore (AVC) is valid.

Exercise 2.2

Prove: During the run of a SAT solver, the implication graph G_k at step k is acyclic.
Hints:

1. Perform a proof by induction on k.
2. Consider the following events that can occur:
   a) making a decision,
   b) unit propagation (one step of BCP),
   c) a clause is unsatisfiable,
   d) backtracking.

2.2) Same as **exercise 14, sample solution from SS18**:

Let $P(n)$ be the property that G_n is acyclic.

Base case, $n = 0$: G_0 is the empty graph, hence it is acyclic. Therefore $P(0)$ holds.

Induction Hypothesis: Let n be an integer ≥ 0 and suppose $P(0), \dots P(n)$ is true.

Step: Consider $P(n+1)$ which we have to show to be true. By (IH) it holds that $G_i = (V_i, E_i)$ is acyclic $(0 \leq i \leq n)$. We continue by a case distinction wrt all possible steps of the SAT solver.

(i) "Making a decision": wlog. let the decision be $X = f@d$. Then $V_{n+1} = V_n \cup \{X\}$ and $E_{n+1} = E_n$. Since G_n is acyclic, $G_{n+1} = (V_{n+1}, E_{n+1})$ also is acyclic.

(ii) "A clause is unsatisfiable": wlog. let the unsatisfiable clause be $(\ell_1 \vee \dots \vee \ell_m)$, then $V_{n+1} = V_n \cup \{\kappa\}$ and $E_{n+1} = E_n \cup \{(\ell_j, \kappa) \mid 1 \leq j \leq m\}$. Note that no new edge has been added other than those pointing to κ. Hence G_{n+1} only contains a cycle if G_n contains a cycle. By (IH) G_n is acyclic and hence also G_{n+1} is acyclic.

(iii) "Unit propagation": wlog. let the unit clause be $(\ell_1 \vee \dots \vee \ell_m)$ and let ℓ_m be the unassigned variable. Then $V_{n+1} = V_n \cup \{\ell_i\}$ and $E_{n+1} = E_n \cup \{(\ell_j, \ell_m) \mid 1 \leq j \leq m-1\}$. Since all added edges point to the newly added node ℓ_m, it follows by (IH) that G_{n+1} is acyclic.

(iv) "Backtracking": since backtracking only removes a part of the implication graph $G_n = (V_n, E_n)$, we have that $G_{n+1} = (V_n \setminus V', E_n \setminus E')$ for some sets $V' \subset V_n, E' \subset E_n$. Removing edges can not make a graph cyclic. Therefore, by (IH) follows that G_{n+1} is acyclic.

Since $P(n+1)$ holds for any step under the assumption that (IH) holds, it therefore follows that $P(n)$ holds for any $n \geq 0$. Consequently, the implication graph G_k at step k is acyclic.

Block 3

Exercise 3.1

Show that the following correctness assertion is true with respect to total correctness:

$$
\begin{array}{l}
\{F: \ x > 0 \ \wedge y > 0\} \\
r := y; \\
n := x; \\
\text{while } n > 0 \text{ do} \\
\quad r := r + 2; \\
\quad n := n - 1; \\
\text{od} \\
\{G: \ r = 2 * x + y\}
\end{array}
$$

Use the formula $r = 2 * x + y - 2 * n \ \wedge 0 \leq n$ as loop invariant and choose an appropriate variant.

{F: x>0∧y>0}
{F8: Inv [n/x][r/y]} as
r:=y;
{F7: Inv[n/x]} as
n:=x;
{F1: Inv} wht
While n>0 do
    {F3: Inv∧n>0∧t=t0} wht
    {F6: F4[n/n-1][r/r+2]} as
    r:=r+2;
    {F5: F4[n/n-1]} as
    n:=n-1
    {F4: Inv∧0≤t<t0} wht
Od
{F2: Inv∧n≤0} wht
{G: r=2x+y}

Proof:

1) F-> F8
   x>0∧y>0 -> (r=2x+<-2n∧0≤n)[n/x][r/y]
   y=2x+y-2x∧0≤x
   True because x>0 implies 0≤x.
2) F3 -> F6
   r=2x+y-2n∧0<=n∧n>0∧n=t0 -> Rhs
   Rhs:

(r=2x+y-2n ∧ 0<=n ∧ 0<=n<t0)[n/n-1][r/r+2]
r+2=2x+y-2n+2 ∧ 0<=n-1 ∧ 0<=n-1<t0
r=2x+y-2n ∧ 0<=n-1<n |removed duplicate expr. + t0=n

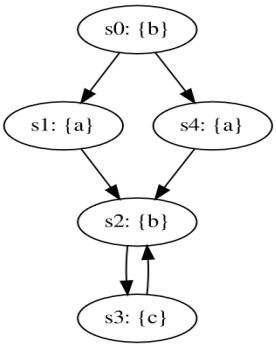
r=2x+y-2n is already in premise -> true, n-1<n is true, 0<=n-1 is implied by n>0 in premise -> true.

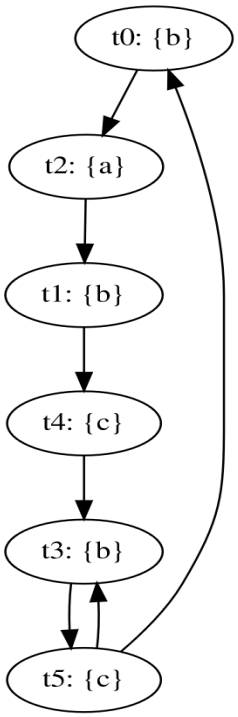
3) F2-> G
r=2x+y-2n ∧ n<=0 ∧ n>=0 -> r=2x+y
True because n=0 in premise.

Block 4

Provide a simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

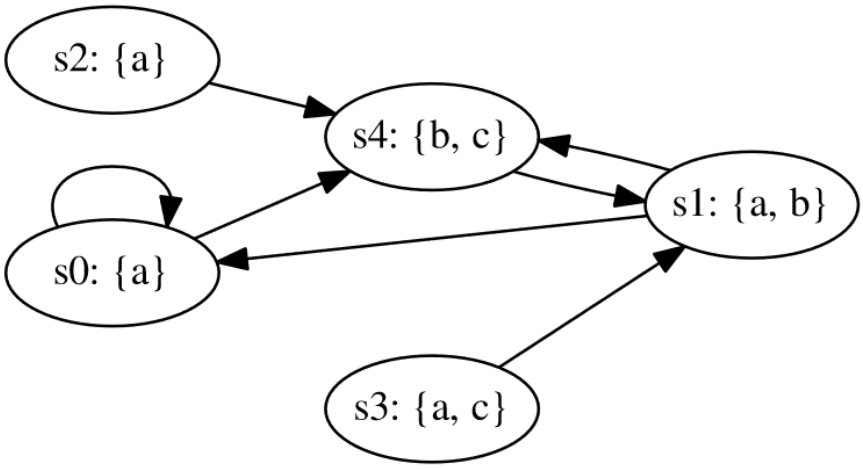
Kripke structure M_1: **Kripke structure M_2:**





Solved by student:

H={ (s0,t0) , (s1,t2) , (s4,t2) , (s2,t1) , (s3,t4) , (s2,t3) , (s3,t5)) }



For each of the following formulae φ,

1. check the respective box if the formula is in CTL, LTL, and/or CTL*, and
2. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

Note that by a CTL* formula, we here mean a CTL* *state* formula *or* a CTL* *path* formula. Also recall that for an LTL formula φ we have $M, s \models \varphi$ if and only if $\pi \models \varphi$ for *all* paths in M that start at s.

Solved by student

phi	CTL	LTL	CTL*	States si
G(a)		x	x	-
G(b)		x	x	-
G(c)		x	x	-
G(a ^ b)		x	x	-

Lets get crazy and solve them all (the ones satisfying no states are omitted)
All LTL:
F(a): s0,s1,s2,s3,s4 | F(b): s1,s2,s3,s4 | F(c): s2,s3,s4 | F(a∧b): s1,s2,s3,s4 | F(a∧c): s3 | F(b∧c): s2,s4 | X(a):s3,s4 | X(b): s2,s3,s4 | X(c): s2 | X(a∧b): s3,s4 | X(b∧c): s2 | aUa: s0,s1,s2,s3 | bUc: s3,s4 | cUa: s0,s1,s2,s3,s4 | a∧bUc: s3,s4 | a∧cUc: s3,s4 |

b∧cUc: s3,s4 | a∧b∧cUb: s1,s4

All CTL (Except A, which should be the same as the LTL formulae above):
A[aUc]: s2,s4,s3 | A[bUa]: s2,s1,s4,s3 | A[(a∧b)Ua]: s0, s1,s2,s3 | A[(a∧c)Ua]: s1,s2,s3 | A[(b∧c)Ua]: s1,s2,s3,s4 | A[(a∧b∧c)Uc]: s3,s4
EF(a): s0,s1,s2,s3,s4(all) | EF(b): all | EF(c): all | EF(a∧b): all | EF(a∧c): s3 | EF(b∧c): all | EX(a): s0,s1,s3,s4 | EX(b): all |
EX(c):s0,s1,s2 | EX(a∧b):s3,s4 | EX(b∧c): s0,s1,s2 |
E[aUa]: s0,s1,s2,s3 | E[bUa]: all | E[cUc]: s3,s4 | E[a∧bUc]:s1,s3,s4 |
E[a∧cUa]:s0,s1,s2,s3 | E[b∧cUc]: s3,s4 | E[a∧b∧cUb]: s1,s4 | EG(a):s0,s1,s3 | EG(b): s1,s4 |

Exercise 4.3

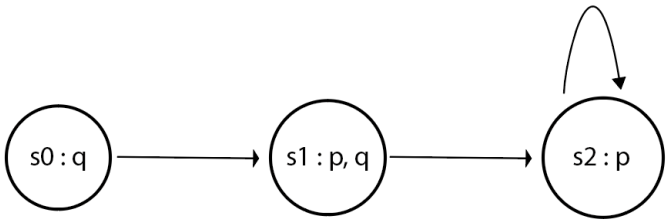
LTL tautologies

Prove or disprove that the following formulas are tautologies, i.e., they hold for every Kripke structure M:

1. $$(\mathbf{G}p)\ \mathbf{U}\ (p \wedge q) \rightarrow \mathbf{F}(q\ \mathbf{U}\ p)$$

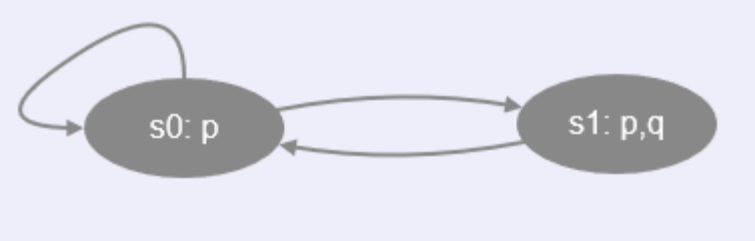
2. $$\mathbf{G}p \wedge \neg\mathbf{G}q \rightarrow p\ \mathbf{U}\ \neg q$$

Solved by student:
~~1 - disprove by counterexample~~



Other opinion: 1 is valid. Let M be an arbitrary Kripke structure and π an arbitrary path in M. If
Gp U (p ^ q) does not hold in π , then the implication holds in π since the left hand side is false.
Therefore it is left to show that if Gp U (p ^ q) holds F(q U p) holds as well. If Gp U (p ^ q) holds in π, then there exists a π^k such that π^k |= p ^ q. Trivially π^k |= p and therefore π^k |= q U p. Since π^k is a subpath of π we have that π |= F(q U p). Therefore the implication is valid.

2 - ~~NO, we disprove by **counterexample** (status student, NOT verified by kripke builder (webservice errors)):~~



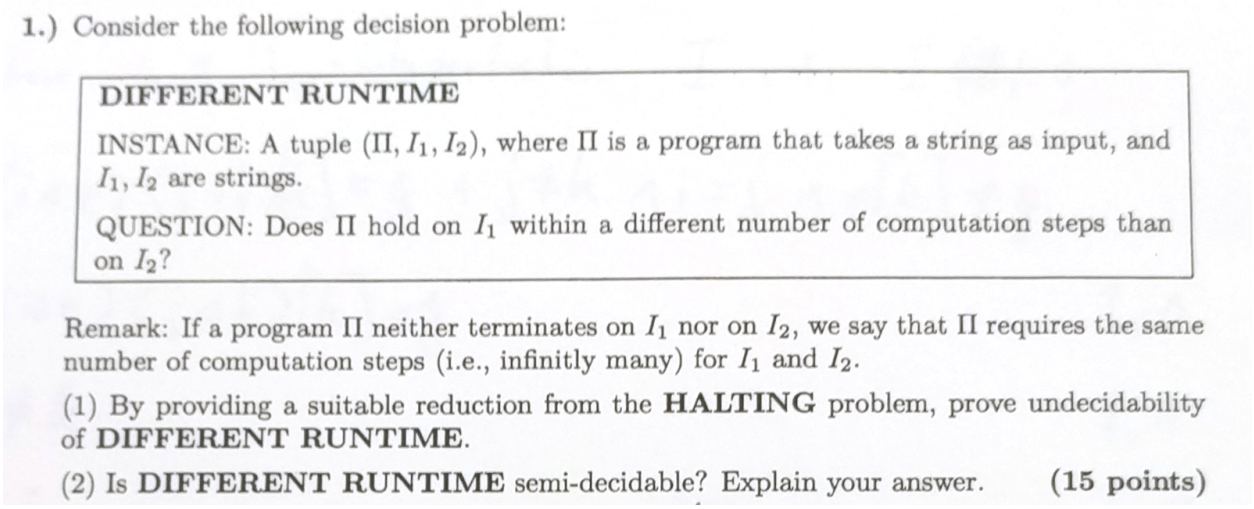
~~G(p) holds and ¬G(q) also holds~~
~~But pU¬q does not hold~~

Don't think the above is right:
G(p) holds in s0 and s1
¬G(q) = F(¬q) and holds only in s0
pU¬q also only holds in s0
So in this example in s0 the implication holds and s1 is not a counterexample…

Other opinion: I think the proof that it actually is a **tautology** looks something like this: Gq means $for\ all\ i\ \geq\ 0,\ M\ \pi^i\ |\ =\ q$. Therefore $\neg Gq$ means $there\ exists\ a\ i\ \geq\ 0\ such\ that\ M, \pi^i|\ =\ \neg q$. The definition for $pU\neg q$is $there\ exists\ a\ k\ \geq\ 0\ such\ that\ M, \pi^k|\ =\ \neg q\ and\ for\ all\ 0\ \leq\ j\ <\ k,\ M, \pi^j|\ =\ p$. The first part is guaranteed by $\neg Gq$and the second part is guaranteed by Gp

Exam 29.06.2018 (FMI.184.pdf)

Block 1



1.) Consider the following decision problem:

DIFFERENT RUNTIME

INSTANCE: A tuple (Π, I_1, I_2), where Π is a program that takes a string as input, and I_1, I_2 are strings.

QUESTION: Does Π hold on I_1 within a different number of computation steps than on I_2?

Remark: If a program Π neither terminates on I_1 nor on I_2, we say that Π requires the same number of computation steps (i.e., infinitly many) for I_1 and I_2.

(1) By providing a suitable reduction from the **HALTING** problem, prove undecidability of **DIFFERENT RUNTIME**.

(2) Is **DIFFERENT RUNTIME** semi-decidable? Explain your answer. **(15 points)**

Similar to Exam 26.01.2018 Block 1

Status: CONFIRMED by Prof.

(1)

(Π, I) arbitrary instance of HALTING

(Π', I1, I2) instance of DIFF RUNTIME, let I1=I, I2 ≠ I1

```
Π' (I') {
        //Pi, I1, I2 hardcoded
        if( I' == I1)
                Π(I1)
        else
                while(true){}
}
```

(2)

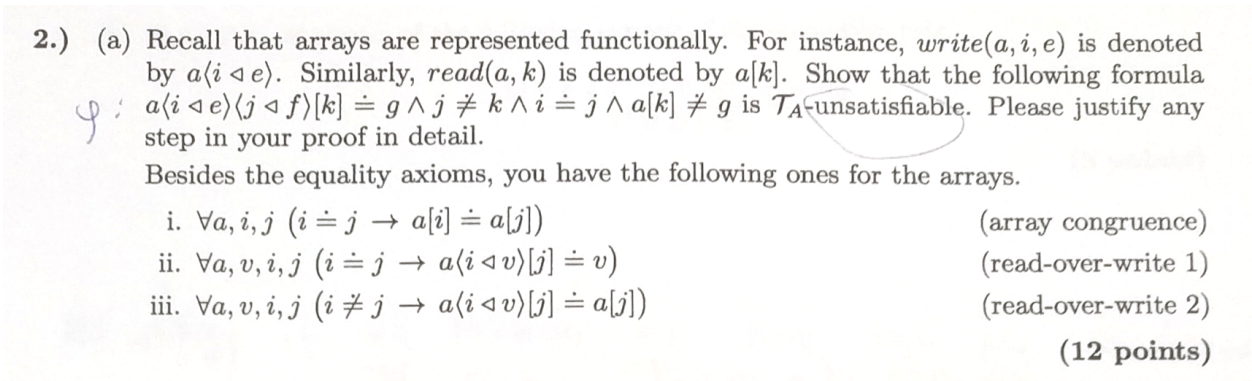
Let Pi_int be an interpreter that takes as input a source code Pi and strings I1, I2. Pi_int simulates the run of Pi(I1), Pi(I2) step by step alternating between the two program calls. If one of the program runs halts but the other is not finished in the same amount of steps,

return true, else false. Pi_int returns true for every positive instance of DIFF-RUNTIME, therefore it is semi-decidable.

We argue that such an interpreter Pi_int is a semi-decision procedure for DIFF-RUNTIME. We distinguish the following cases:

- Case 1:
  Suppose that (Pi', I1, I2) is a positive instance of DIFF-RUNTIME i.e. Pi' takes a different number of computation steps with input I1, than with I2.
  Then Pi_int will return true as soon as one program finishes but the other one needs more steps.
- Case 2.1:
  Suppose (Pi', I1, I2) is a negative instance and Pi'(I1), Pi'(I2) halt with the same number of computation steps. Then Pi_int returns false.
- Case 2.2:
  Suppose (Pi', I1, I2) is a negative instance and both Pi'(I1), Pi'(I2) don't terminate. Then the simulation of this computation by Pi_int will not terminate either. Hence Pi_int will run forever on the negative instance (Pi', I1, I2), which is a correct behavior for a semi-decision procedure.

Block 2



2.) (a) Recall that arrays are represented functionally. For instance, $write(a, i, e)$ is denoted by $a\langle i \triangleleft e\rangle$. Similarly, $read(a, k)$ is denoted by $a[k]$. Show that the following formula $a\langle i \triangleleft e\rangle\langle j \triangleleft f\rangle[k] \doteq g \wedge j \neq k \wedge i \doteq j \wedge a[k] \neq g$ is $\mathcal{T}_A$-unsatisfiable. Please justify any step in your proof in detail.

Besides the equality axioms, you have the following ones for the arrays.

i. $\forall a, i, j \ (i \doteq j \rightarrow a[i] \doteq a[j])$ (array congruence)

ii. $\forall a, v, i, j \ (i \doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq v)$ (read-over-write 1)

iii. $\forall a, v, i, j \ (i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq a[j])$ (read-over-write 2)

(12 points)

Similar to [Exam 4.5.2018 - Block 2]

φ = a⟨i ◁ e⟩⟨j ◁ f⟩[k] =g ∧ j≠k ∧ i=j ∧ a[k]≠g

1.	I ⊧ φ	assumption
2.	I ⊧ a⟨i ◁ e⟩⟨j ◁ f⟩[k] =g	1. Semantics of ∧
3.	I ⊧ j≠k	1. Semantics of ∧
4.	I ⊧ i=j	1. Semantics of ∧
5.	I ⊧ a[k]≠g	1. Semantics of ∧
6.	I ⊧ a⟨i ◁ e⟩⟨j ◁ f⟩[k] = a⟨i ◁ e⟩[k]	2., 3. Read-over-write 2
7.	I ⊧ a⟨i ◁ e⟩[k] = a[k]	3. Read-over-write 2
8.	I ⊧ a[k]=g	7., 6., 2. Transitivity
9.	I ⊧ ⊥	8., 5. contradiction

(b) Show the soundness of the following variant of the resolution rule.

$$\frac{C \vee p \vee q \quad D \vee \neg p \quad E \vee \neg q}{C \vee D \vee E}$$

(3 points)

It should suffice to give an informal argument like the following:

If all premises are true then the conclusion is true in the same interpretation.
As not both p, ¬p and both q, ¬q can be true, at least C or D or E have to be true.

Other solution: simply use resolution.
C1 = C ∨ p ∨ q; C2 = D ∨ ¬p; C3 = E ∨ ¬q
r1 = res(C1, C2, p) = C ∨ D ∨ q
r2 = res(r1, C3, q) = C ∨ D ∨ E

Block 3

3.) Prove that the following correctness assertion is true regarding total correctness. Use the invariant $x * m \leq n < y * m \wedge m > 0$. Describe the function computed by the program, if we consider m and n as the inputs and x as the result.

Some annotation rules you might need:
$\{F\}v := e \mapsto \{F\}v := e\{\exists v'(F[v/v'] \wedge v = e[v/v'])\}$
if e then $\{F\} \cdots$ else $\{G\} \mapsto \{(e \Rightarrow F) \wedge (\neg e \Rightarrow G)\}$if e then $\{F\} \cdots$ else $\{G\}$
$\{F\}$if e then $\cdots$ else $\mapsto \{F\}$if e then $\{F \wedge e\} \cdots$ else $\{F \wedge \neg e\}$
while e do $\cdots$ od $\mapsto \{\ Inv\ \}$while e do $\{\ Inv \wedge e \wedge t{=}t_0\ \} \cdots \{\ Inv \wedge (e \Rightarrow 0 \leq t < t_0)\ \}od\{\ Inv \wedge \neg e\ \}$

$\{\ m > 0 \wedge n \geq 0\ \}$

$x := 0;$

$y := n + 1;$

while $x + 1 \neq y$ do

$z := (x + y)/2;$

if $z * m > n$ then

$y := z$

else

$x := z;$

fi

od

$\{\ x * m \leq n < (x + 1) * m\ \}$

(15 points)

Solution status student (with doubts, please check)
{m>0∧n>=0}
{F10: Inv[y/n+1][x/0]} as
x:=0;

{F9: Inv[y/n+1]} as
y:=n+1;
{F1: Inv} wht
While x+1 != y do
    {F3: Inv ∧ x+1!=y ∧ t=t0} wht
    F8: F7[z/(x+y)/2]} as
    z:=(x+y)/2;
    {F7: (zm>n -> F6) ∧ (zm<=n ->F5)} fi
    If zm>n then
        {F6: F4[y/z]} as
        y:=z
    Else
    {F5: F4[x/z]} as
        x:=z
    Fi
    {F4: Inv ∧ (x+1!=y -> 0<=t<to} wht
Od
{F2: Inv ∧ x+1=y} wht
{xm<=n<(x+1)m}

Variant t: y-x-1

Proofs: 2) F3 ⇒ F8
F3: x*m <= n <ym ∧ m>0 ∧ x+1≠y ∧ y-x-1=t0
F6: x*m <=n <zm ∧ m>0 ∧ (x+1≠z ⇒ 0<= z-x-1 < t0 |F4[y/z]
F5: z*m<=n <ym ∧ m>0 ∧ (z+1≠y ⇒ 0<= y-z-1 < t0 |F4[x/z]
F8: ((z*m>n ⇒ x*m <=n <zm ∧ m>0 ∧ (x+1≠z ⇒ 0<= z-x-1 < t0) ∧ (z*m<=n ⇒ z*m<=n <ym ∧ m>0 ∧ (z+1≠y ⇒ 0<= y-z-1 < t0))[z/(x+y)/2]
(splitting F8 up and applying [z/(x+y)/2])

F8a: ((x+y)/2)*m>n ⇒ x*m <=n < ((x+y)/2)*m ∧ m>0 ∧ (x+1≠(x+y)/2 ⇒ 0<= (x+y)/2 -x -1 < t0)
F8b: ((x+y)/2)*m<=n ⇒ (x+y)/2)*m <=n < y*m ∧ m>0 ∧ ((x+y)/2 +1≠y ⇒ 0<= y -(x+y)/2 -1 < t0))

F3 ⇒ F8a:
x*m <= n <ym ∧ m>0 ∧ x+1≠y ∧ y-x-1=t0 ⇒ (((x+y)/2)*m>n ⇒ x*m <= n < ((x+y)/2)*m ∧ m>0 ∧ (x+1≠(x+y)/2 ⇒ 0<= (x+y)/2 -x -1 < y-x-1))

If (x+y)/2)*m>n is true it follows that the same expression on rhs is true. x*m <= n is in premise. m>0 is in premise
Remains to show: F3 ⇒ (x+1≠(x+y)/2 ⇒ 0<= (x+y)/2 -x -1 < y-x-1):
y/2-x/2-1 < y-x-1 is true

0<= (x+y)/2 -x -1
x+1 <= (x+y)/2
2x+2 <= x+y
x+2 <= y
from the premise we have y>x (bc: xm<=n<ym and m>0) and y≠x+1, therefore y>= x+2 holds

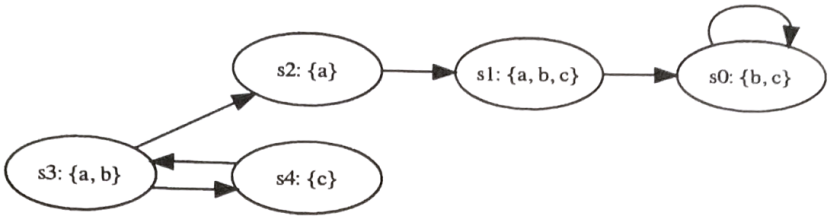
F3⇒ F8b: should be the same

Proofs 1) +3) should be easy to prove.

Block 4

H = {(s0, t0), (s1, t4), (s4, t4), (s3, t3), (s2, t5), (s3, t2)}
H = {(s3, t2), (s2, t5)} müsste auch passen, da auf der Folie mit der Definition (4-3/p10) extra steht, dass Startzustände nicht berücksichtigt werden

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and
ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

Note that by a CTL* formula, we here mean a CTL* *state* formula *or* a CTL* *path* formula. Also recall that for an LTL formula φ we have $M, s \models \varphi$ if and only if $\pi \models \varphi$ for *all* paths in M that start at s.

φ	CTL	LTL	CTL*	States s_i
$\mathbf{G}(c)$				
$((a)\ \mathbf{U}\ (a))$				
$\mathbf{AAF}(a \wedge c)$				
$\mathbf{AX}(a \wedge b)$				
$\mathbf{E}[(b \wedge c)\ \mathbf{U}\ (c)]$				

(5 points)

Status: confirmed by Prof.

φ	CTL	LTL	CTL*	states
G(c)		x	x	s0, s1
((a) U (a))		x	x	s1, s2, s3

AAF(a ^ c)			x	s2, s1
AX(a ^ b)	x		x	s2, s4
E[(b ^ c) U (c)]	x		x	s0, s1, s4

c)

(c) **LTL tautologies**

Prove or disprove (e.g. by providing a counter-example) the following LTL formulas:

i. $$\neg(p \ \mathbf{U} \ q) \rightarrow (\neg \mathbf{G} p \vee \neg \mathbf{F} q)$$

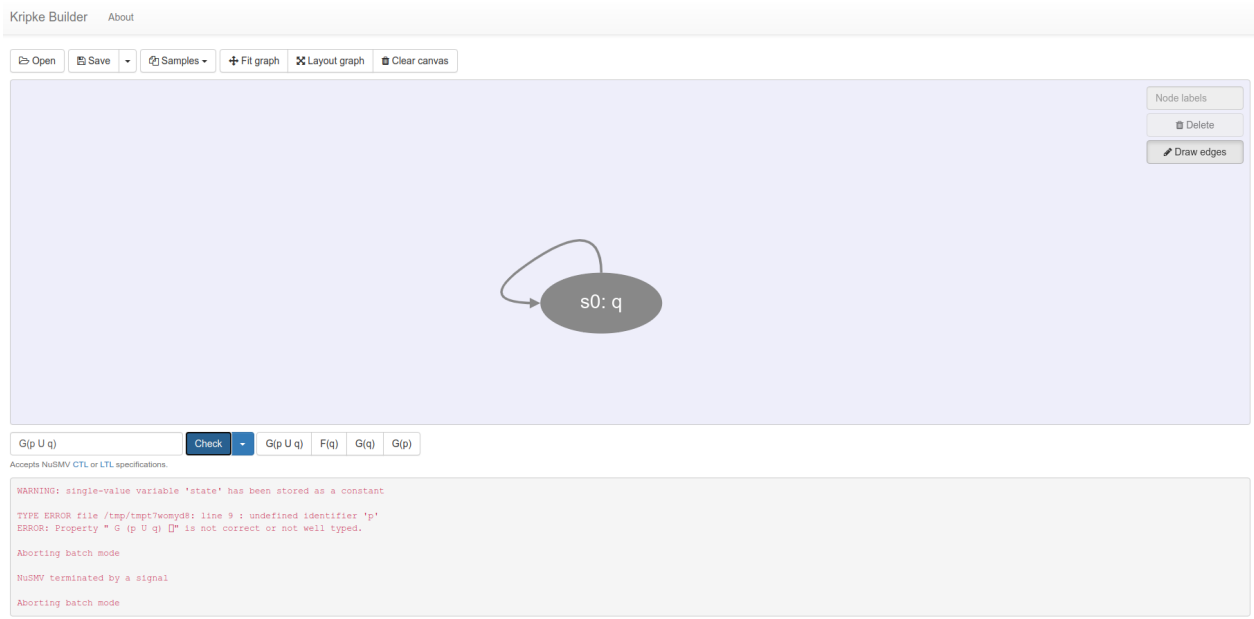
ii. $$\mathbf{G}(p \ \mathbf{U} \ q) \rightarrow \mathbf{G} p$$

iii. $$\mathbf{F}(p \ \mathbf{U} \ q) \rightarrow \mathbf{F} q$$

Solved by student. A have an exam of a friend of mine, so i and iii are tautologies, ii not, this is for sure. The explanations are not so great though, so I came up with my own.

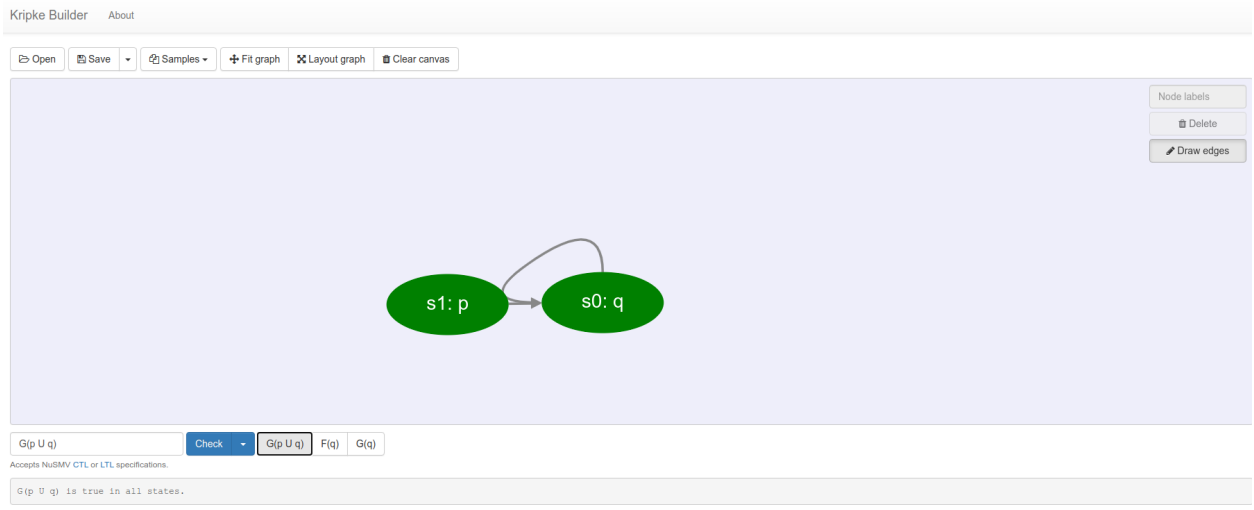
i. Is a tautology!
Let's first try to interpret the statements:
!(p U q) means there is no i, i>=0, so that state si |= q and no 0 <= k < i so that sk |= p
!Gp means p is not globally, so there is an i, i>=0, so that si |= !p, meaning !Gp ←→ F!p
!Fq means there is no i, i>=0, so that state si |= q, meaning !Fq ←→ G!q
Formal arguing:
Let M be an arbitrary Kripke structure with starting state s0. Let π be some path -> s0, s1, ...
Let's try to prove this is a tautology by contradiction. Let's assume LHS is true and RHS is false. Thus, for LHS it holds that !□ i (i >= 0) for which M, π^i |= q (0). In order for the RHS to be false, both (!Gp) and (!Fq) sides of the disjunction have to NOT hold. For !Fq to NOT hold, it should be true that □ j (j >= 0) for which M, π^j |= q (1). The claims (0) and (1) clearly contradict each other, therefore, it is a tautology.

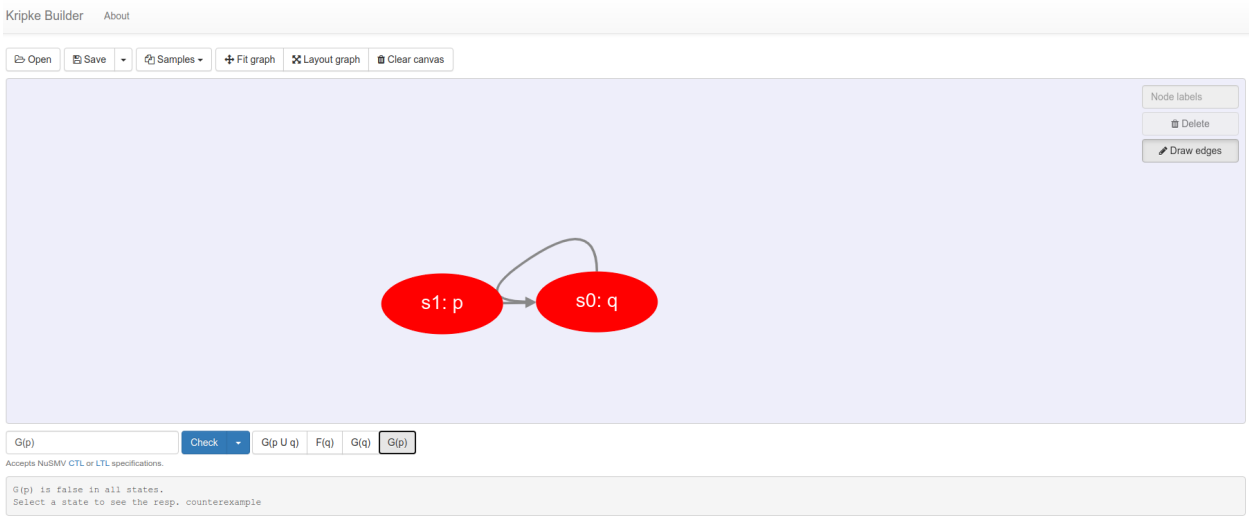
ii. No, it is not. A **counterexample**:
(s0:q)selflooped
We take the only case when the whole statement is false: RHS is true and LHS is false.
So LHS, G(p U q), holds, because globally is q. Since p before q is optional, it is fine there is no p at all.
RHS does not hold, because p is clearly not globally true. It is in fact never true.



Another Student:

Counterexample: (s1: p) -> (s0:q) -> self loop to (s0)





iii. Yes, it is a tautology.

Formal arguing:

Let M be an arbitrary Kripke structure with starting state s0. Let π be some path -> s0, s1, ... Let's try to prove this is a tautology by contradiction. Let's assume LHS is true and RHS is false. Thus, for LHS it holds that □ i (i >= 0) for which M, π^i |= q (0). In order for the RHS to be false, Fq should NOT hold. For Fq to NOT hold, it should be true that !□ j (j >= 0) for which M, π^j |= q (1). The claims (0) and (1) clearly contradict each other, therefore, it is a tautology.

Exam 04.05.2018 (FMI.183.pdf)

Block 1

1.) An undirected graph is called a *non-terminal graph* if each vertex in the graph has at least two edges to other vertices. Examples: $(\{a,b,c\},\{[a,b],[b,c],[a,c]\})$ is non-terminal, while $(\{a,b,c\},\{[a,b],[b,c]\})$ or $(\{a,b,c,d\},\{[a,b],[b,c],[a,c]\})$ are not.

Consider the following problem:

> **3-COLORABILITY-NT**
>
> INSTANCE: A non-terminal graph $G=(V,E)$.
>
> QUESTION: Does there exists a function μ from vertices in V to values in $\{0,1,2\}$ such that $\mu(v_1)\neq\mu(v_2)$ for any edge $[v_1,v_2]\in E$.

Use the fact that the standard version of the **3-COLORABILITY** problem is NP-complete, to prove that **3-COLORABILITY-NT** is NP-complete as well. Give a brief argument for NP-membership and show NP-hardness by a reduction from **3-COLORABILITY**.

Recall that **3-COLORABILITY** is defined as follows:

> **3-COLORABILITY**
>
> INSTANCE: An undirected graph $G=(V,E)$.
>
> QUESTION: Does there exists a function μ from vertices in V to values in $\{0,1,2\}$ such that $\mu(v_1)\neq\mu(v_2)$ for any edge $[v_1,v_2]\in E$.

Hint: When reducing from **3-COLORABILITY** to **3-COLORABILITY-NT**, replace (certain) vertices by a triangle.

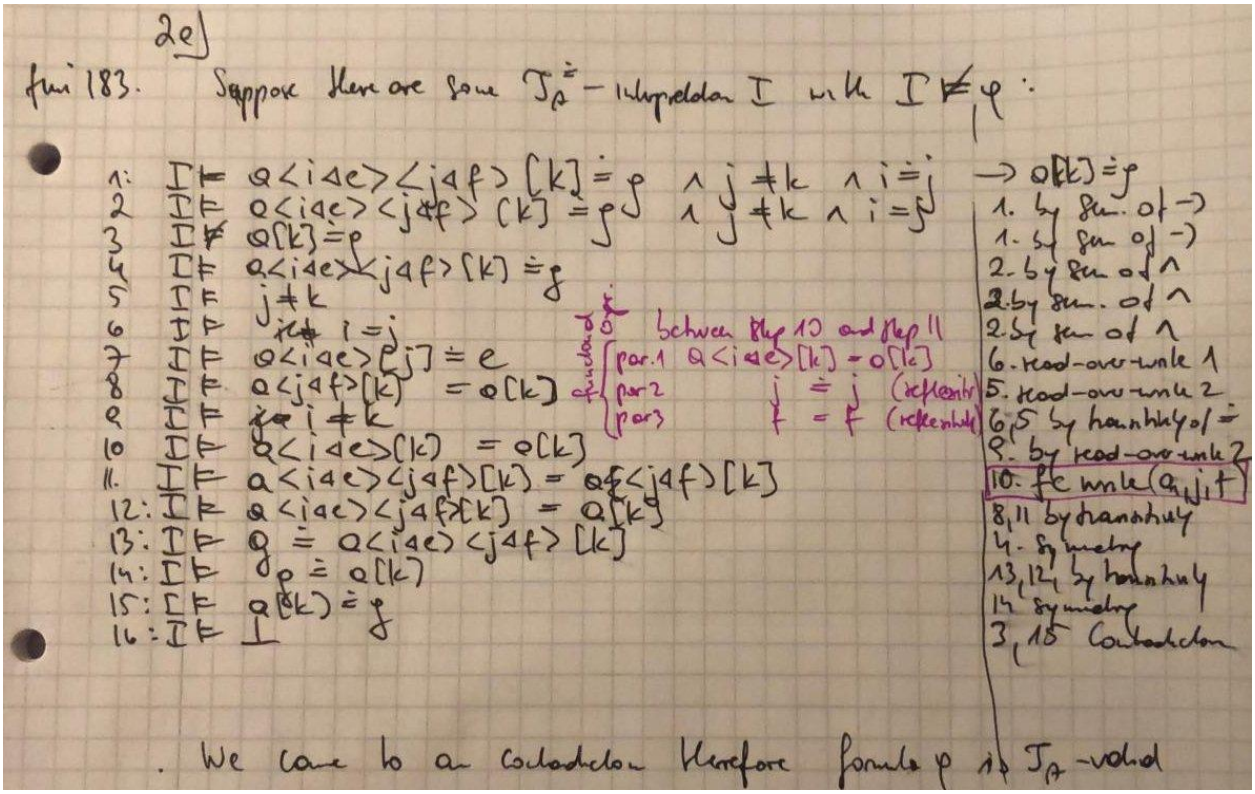
(15 points)

Replace vertices with only one edge (or zero edges) to other nodes with a triangle which itself is a pos instance of 3-col.
Certificate Relation?

Block 2

a) Just apply read-over-write 2 two times and use transitivity.

Status: solved by student



φ = a⟨i ◁ e⟩⟨j ◁ f⟩[k] =g ∧ j≠k ∧ i=j → a[k]=g

1.	I ⊨ φ	assumption
2.	I ⊨ ¬ {a⟨i ◁ e⟩⟨j ◁ f⟩[k] =g ∧ j≠k ∧ i=j} ∨ a[k]=g	1. Semantics of →
3.	I ⊧ a⟨i ◁ e⟩⟨j ◁ f⟩[k] =g ∧ j≠k ∧ i=j ∧ a[k]≠g	2. Semantics of ¬
4.	I ⊧ a⟨i ◁ e⟩⟨j ◁ f⟩[k] =g	3. Semantics of ∧
5.	I ⊧ j≠k	3. Semantics of ∧
6.	I ⊧ i=j	3. Semantics of ∧
7.	I ⊧ a[k]≠g	3. Semantics of ∧
8.	**I ⊧ a⟨i ◁ e⟩⟨j ◁ f⟩[k] = a⟨i ◁ e⟩[k]**	**4. Read-over-write 2**
9.	**I ⊧ a⟨i ◁ e⟩[k] = a[k]**	**8. Read-over-write 2**
10.	I ⊧ a[k]=g	9., 8., 4. Transitivity, Symm.
11.	I ⊧ ⊥	10., 7. contradiction

Status: solved by student

2.b. SOLUTION MISSING?

Block 3

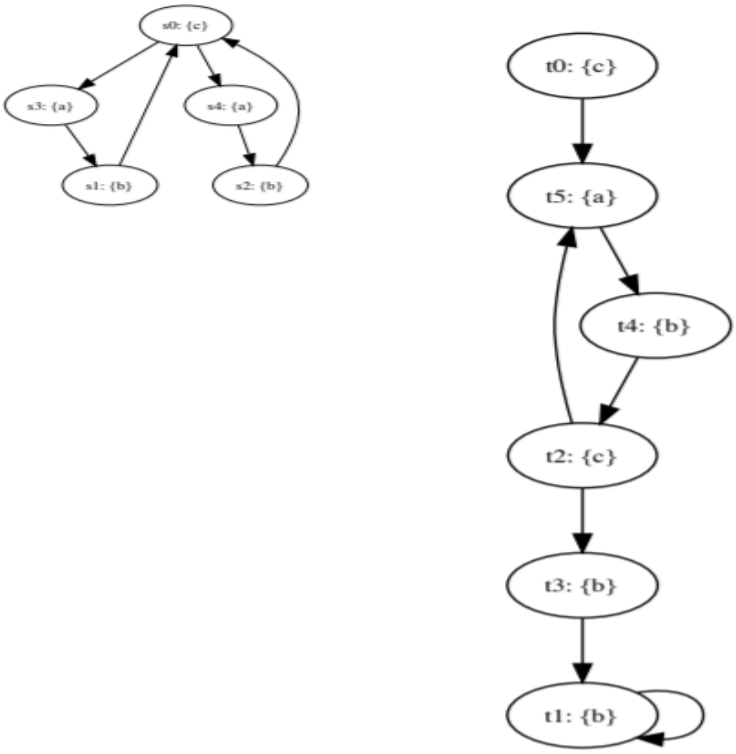
See Block 3 - fmi161 and Block 3 - 27.1.2017

Block 4

a)

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1: **Kripke structure M_2:**



(4 points)

Solved by student:
H={(s0,t0),(s3,t5),(s4,t5),(s1,t4),(s0,t2),(s2,t4)}

b) Verified by Kripke Builder:

G (b)	LTL, CTL*	s0, s1
AF(c)	CTL, CTL*	s4,s2,s3,s0,s1
((c) U (a))	LTL, CTL*	s4,s2,s0
EG (c)	CTL, CTL*	s4, s3
E[(a & b) U (c)]	CTL, CTL*	s3, s4, s0

(c) **LTL tautologies**

Prove or disprove (e.g. by providing a counter-example) the following LTL formulas:

i. $$(p \ \mathbf{U} \ \neg q) \rightarrow (\neg \mathbf{G} q)$$

ii. $$(\mathbf{FG} p) \rightarrow (\mathbf{GF} p)$$

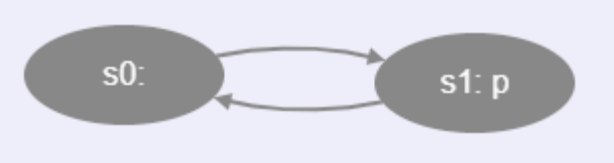
iii. $$\mathbf{FX} p \rightarrow \mathbf{XG} p$$

(6 points)

c) **i) True, is a tautology**, … i>=0 s.t. M,Pi_i |= !q for both

ii) True, is a tautology: Let π be a path that satisfies FGp. This means there is some k≥0 such that for all j ≥ k, π^j |= p. From this follows that for all i ≥ 0 there exists some j ≥ i (e.g., max(i,k)) such that π^j |= p. This means π |= GFp.

iii) Counterexample:



Exam 16.03.2018 (FMI.182.pdf)

Block 1

1.) Consider the following problem:

> **EXACT HITTING SET (EHS)**
>
> INSTANCE: A collection $\mathcal{C}$ of sets of elements.
>
> QUESTION: Does there exist a set S of elements, such that for each $C \in \mathcal{C}$, $|S \cap C| = 1$, i.e. each set in $\mathcal{C}$ contains exactly one element from S?

Example: Consider $\mathcal{C} = \{\{a, b, d\}, \{b, c\}, \{c, d\}\}$. $S = \{a, c\}$ witnesses that $\mathcal{C}$ is a positive instance of **EHS**. On the other hand, $\mathcal{C}' = \{\{a, b, d\}, \{a, b, c\}, \{c, d\}\}$ is a negative instance.

By providing a suitable reduction from the **1-IN-3-SAT** problem, prove that **EXACT-HITTING-SET** is an NP-hard problem. Argue formally that your reduction is correct.

Recall that **1-IN-3-SAT** is defined as follows:

> **1-IN-3-SAT**
>
> INSTANCE: Boolean formula φ in 3-CNF.
>
> QUESTION: Does there exists a satisfying truth assignment T on φ, such that in each clause of φ, exactly one literal is true in T?

Hint: For each variable v in φ, use two elements v and $\neg v$ in your definition of $\mathcal{C}$.

(15 points)

Solved by Student:

We construct C in a way that each clause in phi is a set of elements in C.
If a clause in phi contains a negated variable we change that to a new variable *nv* (negatedVariable) in C

Additionally for every variable you need the additional set that includes the positive and negative instance of said variable. E.g. {{a, an}, {b, bn}, …} so that the variables can not be true and false at the same time.
C = {{v_a, $v_{\neg a}$} | ∀ a ∈ Var} ∪ {{v_x, v_y, v_z} | ∀ clauses in phi where x, y, z are the 3 variables or negated variables}

Eg:

phi = (a v b v e) ∧ (a v -d v e) ∧ (-e v b v c)

T = { a, -b, -c, d, -e }

C = { { a, b, e }, { a, dn, e }, { en, b, c }, {a, an}, {b, bn}, {c, cn}, {d, dn} }

S = {a, bn, cn, d, en }

Iff Phi is a positive instance of 1-IN-3-SAT C is also a positive instance of EXACT HITTING SET and vice versa.

Formal proof => and <= is missing.

Block 2

2.) (a) Show that $b[j] \doteq f \rightarrow b\langle j \triangleleft f\rangle \doteq b$ is $\mathcal{T}_A^{=}$-valid.

Besides the equality axioms, you have the following ones for the arrays.

i. $\forall a, i, j \ \left(i \doteq j \rightarrow a[i] \doteq a[j]\right)$ (array congruence)

ii. $\forall a, v, i, j \ \left(i \doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq v\right)$ (read-over-write 1)

iii. $\forall a, v, i, j \ \left(i \neq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq a[j]\right)$ (read-over-write 2)

iv. $\forall a, b \left(\forall j \ \left(a[j] \doteq b[j]\right) \leftrightarrow a \doteq b\right)$ (extensionality)

(12 points)

(b) Consider the clauses $C_1, \ldots, C_6$ in `dimacs` format (in this order, shown in the box; recall that `0` indicates the end of a clause) which are given as input to a SAT solver.

- Apply CDCL using the convention that if a variable is assigned as a decision, then it is assigned 'true'. Select variables as decisions in *increasing* order of their respective integer IDs in the `dimacs` format, starting with variable `1`.
- When the *first* conflict occurs, draw the complete implication graph, mark the first UIP, give the derivation of the learned asserting clause that corresponds to the first UIP, and stop CDCL. You do *not* have to solve the formula!

```
-1 4 0
-4 5 0
-2 -4 6 0
-3 -6 7 0
-7 9 0
-5 -6 -7 -9 0
```

(3 points)

2.a. **Proof by contradiction (solved by student):**

Suppose there is a $T_A^{=}$-Interpretation I =⟨U, I,α⟩

1: I ⊨ b[j] = f → b⟨j ◁ f⟩ = b

2: I ⊨ b[j] = f — From 1 left side of implication

3: I ⊭ b⟨j ◁ f⟩ = b — From 1 right side of implication

4: I ⊨ ∀i ((b⟨j ◁ f⟩[i] = b[i]) — From 3 apply extensionality

5: $I_1 \not\models (b\langle j \triangleleft f\rangle[j] = b[j]$ From 4 take only the j case: $j \in U$, $I_1 = I\{j \leftarrow i\}$
6: $I_1 \not\models f = b[j]$ From 5 because of read-over-write 1
7: $I_1 \not\models b[j] = f$ From 6 because of reflexivity(equality axiom)
Conflict at 7 and 2. Therefore it is $T_A^=$-valid.
It is done similarly in FMI.181. Not sure mine is also correct

Alternative solution by student:
Proof by contradiction: Suppose there is a $T_A^=$-Interpretation $I = \langle U, I, \alpha\rangle$
1: $I \not\models b[j] = f \rightarrow b\langle j \triangleleft f\rangle = b$
2: $I \models b[j] = f$ From 1 left side of implication
3: $I \not\models b\langle j \triangleleft f\rangle = b$ From 1 right side of implication
4: $I \not\models \forall i\ ((b\langle j \triangleleft f\rangle[i] = b[i])$ From 3 apply extensionality
5: $I_1 \not\models (b\langle j \triangleleft f\rangle[k] = b[k]$ From 4 , ∀, for some $k \in U$: $I_1 = I\{i \leftarrow k\}$
6: $I_1 \not\models j \neq k$ 5, read over write 2, modus tollens
7: $I_1 \models j = k$ 6, ¬ (negation)
8: $I_1 \models b[j] = b[k]$ 7, array congruence
9: $I_1 \models f = b[k]$ 2, 8, transitivity
10: $I_1 \models b\langle j \triangleleft f\rangle[k] = f$ 7, read over write 1
11: $I_1 \models (b\langle j \triangleleft f\rangle[k] = b[k]$ 9, 10, transitivity
12: contradiction between 11 and 5

2.b.
Too much work to get the graph here, but my conflict was at f_9, my UIP was at f_7 and my conflict clause was c_7: $\neg f_5 \vee \neg f_6 \vee \neg f_7$. Would be great if someone could verify that

Checked (other student): → got the same result, also see Block 2 from 5.5.2017

Block 3

3.) Show that the following correctness assertion is true with respect to total correctness. Describe the function computed by the program if we consider k as its input and m as its output.

Hints: Use the formula $m^2 \leq k < n^2 \wedge 0 \leq m < n \leq k+1$ as loop invariant. Depending on how you choose the variant, use one of the following annotation rules:

while e do $\cdots$ od $\mapsto \{\, Inv \,\}$while e do $\{\, Inv \wedge e \wedge t{=}t_0 \,\} \cdots \{\, Inv \wedge 0 \leq t < t_0 \,\}od\{\, Inv \wedge \neg e \,\}$
while e do $\cdots$ od $\mapsto \{\, Inv \,\}$while e do $\{\, Inv \wedge e \wedge t{=}t_0 \,\} \cdots \{\, Inv \wedge (e \rightarrow 0 \leq t < t_0) \,\}od\{\, Inv \wedge \neg e \,\}$

$\{\, F\colon k \geq 0 \,\}$
$m := 0;$
$n := k + 1;$
while $m + 1 \neq n$ **do**
  $l := (m + n)/2;$
  if $l^2 \leq k$ **then**
    $m := l$
  else
    $n := l$
  fi
od
$\{\, G\colon m^2 \leq k < (m+1)^2 \,\}$

(15 points)

Solution by professor (dvsw-solutions.pdf, Exercise 10, SS18):

x=k, y=m, z=n, l=t

$\{ F_1 : x \geq 0 \}$
$\{ F_{15} : Inv[z/x+1][y/0] \}$ (as↑)
$y := 0;$
$\{ F_{14} : Inv[z/x+1] \}$ (as↑)
$z := x + 1;$
$\{ F_3 : Inv \}$ (wht″)
while $y + 1 \neq z$ **do**
  $\{ F_4 : Inv \wedge y + 1 \neq z \wedge s = s_0 \}$ (wht″)
  $t := (y + z)/2;$
  $\{ F_{11} : Inv \wedge y + 1 \neq z \wedge s = s_0 \wedge t = (y + z)/2 \}$ (as↓)
  if $t^2 \leq x$ **then**
    $\{ F_{12} : Inv \wedge y + 1 \neq z \wedge s = s_0 \wedge t = (y + z)/2 \wedge t^2 \leq x \}$ (if↓)
    $\{ F_9 : (Inv \wedge 0 \leq s < s_0)[y/t] \}$ (as↑)
    $y := t;$
    $\{ F_7 : Inv \wedge 0 \leq s < s_0 \}$ (fi↑)
  else
    $\{ F_{13} : Inv \wedge y + 1 \neq z \wedge s = s_0 \wedge t = (y + z)/2 \wedge t^2 > x \}$ (if↓)
    $\{ F_{10} : (Inv \wedge 0 \leq s < s_0)[z/t] \}$ (as↑)
    $z := t$
    $\{ F_8 : Inv \wedge 0 \leq s < s_0 \}$ (fi↑)
  fi
  $\{ F_5 : Inv \wedge 0 \leq s < s_0 \}$ (wht″)
od
$\{ F_6 : Inv \wedge y + 1 = z \}$ (wht″)
$\{ F_2 : y^2 \leq x < (y + 1)^2 \}$

We choose the invariant $Inv \equiv y < z \leq x+1 \wedge y^2 \leq x < z^2$ and the bound function $s := z - y$. We have to prove the following implications:

$$F_1 \colon x \geq 0 \Rightarrow F_{15} \colon Inv[z/x+1][y/0]$$
$$F_{12} \colon Inv \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x \Rightarrow F_{9a} \colon Inv[y/t]$$
$$F_{12} \colon Inv \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x \Rightarrow F_{9b} \colon 0 \leq s[y/t] < s$$
$$F_{13} \colon Inv \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \Rightarrow F_{10a} \colon Inv[z/t]$$
$$F_{13} \colon Inv \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \Rightarrow F_{10b} \colon 0 \leq s[z/t] < s$$
$$F_6 \colon Inv \wedge y+1 = z \Rightarrow F_2 \colon y^2 \leq x < (y+1)^2$$

$F_1 \Rightarrow F_{15}$:

$$x \geq 0 \Rightarrow Inv[z/x+1][y/0]$$
$$x \geq 0 \Rightarrow 0 < x+1 \leq x+1 \wedge 0^2 \leq x < (x+1)^2$$

The conditions on the right-hand side are obviously true. Note that $0 < x+1$ and $0^2 \leq x$ hold because of $x \geq 0$.

$F_{12} \Rightarrow F_{9a}$:

$$Inv \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x$$
$$\Rightarrow Inv[y/t]$$

$$y < z \leq x+1 \wedge y^2 \leq x < z^2 \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x$$
$$\Rightarrow t < z \leq x+1 \wedge t^2 \leq x < z^2$$

The conditions on the right-hand side also occur on the left-hand side except $t < z$, which we therefore have to prove. Since $y < z$ (first condition on the left-hand side) holds, the value of $t = (y+z)/2$ is at most $(z-1+z)/2 = z-1$, hence $t < z$ holds.

$F_{12} \Rightarrow F_{9b}$:

$$Inv \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x$$
$$\Rightarrow 0 \leq s[y/t] < s$$

$$y < z \leq x+1 \wedge y^2 \leq x < z^2 \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x$$
$$\Rightarrow 0 \leq z - t < z - y$$

The conclusion can be written as $y < t \leq z$. In the proof of the implication $12 \Rightarrow 9a$ we show $t < z$, hence we also have $t \leq z$. Moreover, in the proof of implication $13 \Rightarrow 10a$ we show $y < t$ using only premises also occurring in formula 12.

$F_{13} \Rightarrow F_{10a}$:

$$\begin{aligned} & Inv \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \\ \Rightarrow\ & Inv[z/t] \end{aligned}$$

$$\begin{aligned} & y < z \leq x+1 \wedge y^2 \leq x < z^2 \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \\ \Rightarrow\ & y < t \leq x+1 \wedge y^2 \leq x < t^2 \end{aligned}$$

The conditions on the right-hand side also occur on the left-hand side except $y < t \leq x+1$, which we therefore have to prove. The condition $t \leq x+1$ holds, since $t < z$ (see argument above) and $z \leq x+1$ (second condition on the left-hand side). To show $y < t$, note that $y < z$ and $y+1 \neq z$, i.e., $z \geq y+2$. Therefore the value of $(y+z)/2$ is at least $(y+y+2)/2 = y+1$, hence $y < t$.

$F_{13} \Rightarrow F_{10b}$:

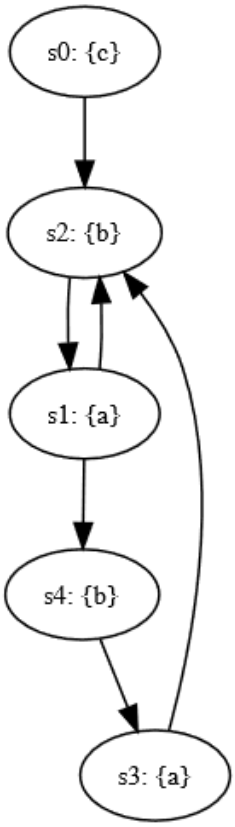
$$\begin{aligned} & Inv \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \\ \Rightarrow\ & Inv[z/t] \end{aligned}$$

$$\begin{aligned} & y < z \leq x+1 \wedge y^2 \leq x < z^2 \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \\ \Rightarrow\ & 0 \leq t-y < z-y \end{aligned}$$

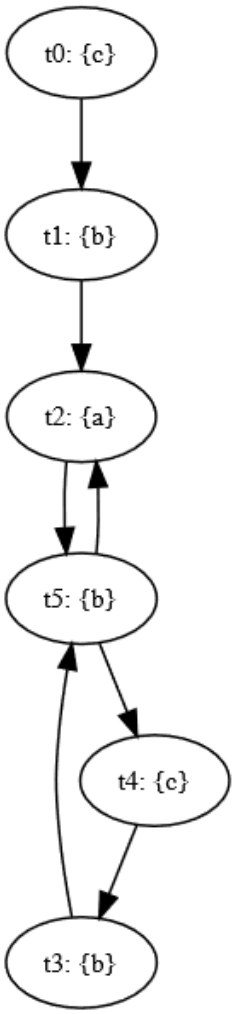
Block 4

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



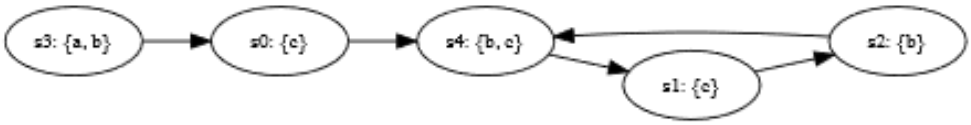
Kripke structure M_2:



(4 points)

Solved by student: H={(s0, t0), (s2, t1), (s1, t2), (s4, t5), (s3, t2), (s2, t5)}

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{F}(a \wedge b)$	☐	☐	☐	
$\mathbf{AX}(b)$	☐	☐	☐	
$\mathbf{AX}(b \wedge c)$	☐	☐	☐	
$\mathbf{EF}(c)$	☐	☐	☐	
$\mathbf{E}[(a \wedge b)\ \mathbf{U}\ (b)]$	☐	☐	☐	

(5 points)

Solved by student, verified by Kripke Builder:

F (a & b)	**LTL**, CTL*	s3
AX (b)	CTL, CTL*	s0, s2, s1
AX (b & c)	CTL, CTL*	s0, s2
EF (c)	CTL, CTL*	s0, s1, s2, s3, s4
E[(a & b) U (b)]	CTL, CTL*	s3, s4, s2

(c) **LTL tautologies**

Prove or disprove the following LTL formulas:

i. $(\neg \mathbf{G} q) \rightarrow (p\ \mathbf{U}\ \neg q)$

ii. $(\mathbf{GF} p) \rightarrow (\mathbf{FG} p)$

iii. $\mathbf{FX} p \rightarrow \mathbf{XF} p$

(6 points)

Solved by student:

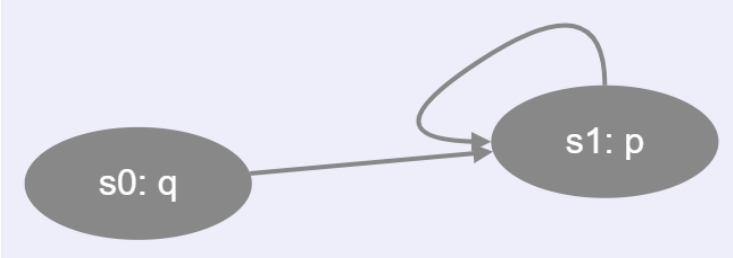
i) False:

¬(Gq) ≡ F¬q

$M,\pi \models F\neg q \Leftrightarrow$ there exists a k>=0 such that $M,\pi^k \models \neg q$

$M,\pi \models (p\ U\ \neg q) \Leftrightarrow$ there exists a k>= 0 such that $M,\pi^k \models \neg q$ and for all 0<=j<k, $M,\pi^j \models p$

Thus the implication would only hold in the other direction.



$M,\pi^0 \models F\neg q$ but $M,\pi^0 \not\models p\ U\ \neg q$

ii) False:

Counterexample which holds for G F p but not for F G p

iii) True:

$M,\pi \models FXp \Leftrightarrow$ there exists a k>=0 such that $M,\pi^k \models Xp$

$M,\pi^k \models Xp \Leftrightarrow M,\pi^{k+1} \models p$

$M,\pi \models XFp \Leftrightarrow M,\pi^1 \models Fp$

$M,\pi^1 \models Fp \Leftrightarrow$ there exists a k>=1 such that $M,\pi^k \models p$

Exam 26.01.2018 (FMI.181.pdf)

Block 1

1.) Consider the following decision problem:

> **HALTING IN LESS STEPS**
>
> INSTANCE: A tuple (Π_1, Π_2, I), where Π_1, Π_2 are programs that take a string as input, and I a string.
>
> QUESTION: Does Π_1 hold on I in strictly less computation steps than Π_2 does on I?

Remark: If a program Π terminates on an input I and a program Π' does not terminate on I, we say that Π holds on I in strictly less computation steps than Π'. In case both programs do not terminate, it is not the case that Π holds on I in strictly less computation steps than Π' on I.

(1) By providing a suitable reduction from the **HALTING** problem, prove that **HALTING IN LESS STEPS** is undecidable.

(2) Is **HALTING IN LESS STEPS** semi-decidable? Explain your answer. **(15 points)**

CONFIRMED by Prof.
student attempts (1):
For an instance of HALTING with program π and input I, build an instance for HALTING IN LESS STEPS as π1 = π, π2 = [program that never halts] I' = I.

=> assuming π and I is a yes-instance of HALTING, then π halts on I. Because [program that never halts] is defined to need more than finite steps we have a yes-instance for HALTING IN LESS STEPS.

<= assuming π1, π2 and I is a yes-instance of HALTING IN LESS STEPS, then π1 halts in less steps than π2. Either both halt or π2 does not halt while π1 does halt. (why? because only a finite number of steps is strictly less than infinite steps) direct consequence -> π = π1, I = I is an instance of HALTING.

(2)
is wrong in this picture (the reduction only looks at instances where Pi2 does not halt). We need to **provide a Semi-Decision Algorithm:**

We can create an Interpreter that executes one Step from π1 and then one from π2 until one of them halts (Or an interpreter that runs both in parallel and aborts π1 as soon as π2 holds should also be a possibility).

One should probably argue like in the example slides and list the distinct cases for the interpreter function:
Case 1: pos instance of HLS
Case 2.1: neg instance of HLS and Pi2 halts on I.
Case 2.2: neg instance of HLS and Pi2 does not halt on I (and because neg. instance also Pi1 does not halt)

Block 2

2.) (a) Show that $a[i] \doteq e \rightarrow a\langle i \triangleleft e\rangle \doteq a$ is $\mathcal{T}_A^{=}$-valid.
Besides the equality axioms, you have the following ones for the arrays.

i. $\forall a, i, j \ \left(i \doteq j \rightarrow a[i] \doteq a[j]\right)$ (array congruence)
ii. $\forall a, v, i, j \ \left(i \doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq v\right)$ (read-over-write 1)
iii. $\forall a, v, i, j \ \left(i \not\doteq j \rightarrow a\langle i \triangleleft v\rangle[j] \doteq a[j]\right)$ (read-over-write 2)
iv. $\forall a, b \left(\forall j \ \left(a[j] \doteq b[j]\right) \leftrightarrow a \doteq b\right)$ (extensionality)

(12 points)

(b) Answer the questions about the CDCL algorithm and justify your answers in detail.

i. Suppose that CDCL is applied to solve a propositional formula φ in CNF, and a clause C is learned. Does $\varphi \equiv \varphi \wedge C$ hold?
ii. Consider a run of CDCL on a given CNF F, and suppose that the run has terminated. What is the current decision level in CDCL at the time when CDCL terminates?

(3 points)

(a) Same Example as presented in slides fminf_sat3.pdf. Just perform the extensionality axiom on the right hand side of the implication and you lead to the formula below.

b)
i) Yes, because C just avoids truth-assignments which would lead to a conflict. CDCL does not affect soundness or completeness. Each learnt clause can be inferred from the original clauses and other learnt clauses.
ii) If F is unsat decision level will be 0, else DL < amount of distinct variables -1 (from a comment at Block 2 of 30.6.2017)

Block 3

3.) Consider the following modified while-rule:

$$\frac{\{\, Inv \wedge e \,\}\, p \,\{\, Inv \,\}}{\{\, Inv \wedge e \,\}\ \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \{\, Inv \wedge \neg e \,\}}\ \text{mw}$$

(a) Show that this rule is admissible regarding partial correctness. **(5 points)**

(b) Show that the Hoare calculus for partial correctness is no longer complete, if we replace the regular while-rule by the modified one. **(10 points)**

See 4.12.2015 Block 2

OR

Solution for similar exercise by professor (dvsw-solutions.pdf, Exercise 6):

Consider the following modified while-rule:

$$\frac{\{\, Inv \,\}\, p \,\{\, Inv \,\}}{\{\, Inv \,\}\ \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \{\, Inv \wedge \neg e \,\}}\ \text{mw}$$

Solution

(a) We have to show that the conclusion $\{\, Inv \,\}\ \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \{\, Inv \wedge \neg e \,\}$ is true whenever the premise $\{\, Inv \,\}\, p\, \{\, Inv \,\}$ is true, for all formulas Inv, e and programs p. This can be done by deriving the conclusion from the premise using rules that are already known to be admissible, like the regular rules of Hoare calculus.

$$\dfrac{\dfrac{(Inv \wedge e) \Rightarrow Inv \quad \{\, Inv \,\}\, p\, \{\, Inv \,\}}{\{\, Inv \wedge e \,\}\, p\, \{\, Inv \,\}}\ \text{lc}}{\{\, Inv \,\}\ \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \{\, Inv \wedge \neg e \,\}}\ \text{wh}$$

The formula $(Inv \wedge e) \Rightarrow Inv$ is a tautology. Therefore, by the correctness of the Hoare calculus, the conclusion $\{\, Inv \,\}\ \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \{\, Inv \wedge \neg e \,\}$ is true whenever the premise $\{\, Inv \,\}\, p\, \{\, Inv \,\}$ is true.

(b) To show the incompleteness of the modified Hoare calculus we give a counter-example, which consists of a concrete correctness assertion that is true with respect to partial correctness, but which cannot be derived in the modified calculus. Consider the assertion

$$\{\, x \geq 0 \,\}\ \mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\ \{\, x \geq 0 \wedge x \not> 0 \,\}\ .$$

This assertion is true as can e.g. be shown by deriving it in the regular calculus:

$$\dfrac{\dfrac{\overset{\text{valid}}{x \geq 0 \wedge x > 0 \Rightarrow x - 1 \geq 0} \quad \{\, x - 1 \geq 0 \,\}\, x := x - 1\, \overset{\text{as}}{\{\, x \geq 0 \,\}}}{\{\, x \geq 0 \wedge x > 0 \,\}\, x := x - 1\, \{\, x \geq 0 \,\}}\ \text{lc}}{\{\, x \geq 0 \,\}\ \mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\ \{\, x \geq 0 \wedge x \not> 0 \,\}}\ \text{wh}$$

But this assertion cannot be derived in the modified calculus as we will show by contradiction. Suppose it can be derived. Then the derivation must have the form

$$\dfrac{x \geq 0 \Rightarrow F \quad \dfrac{\dfrac{\text{some derivation of } \{\, F \,\}\, x := x - 1\, \{\, F \,\}}{\{\, F \,\}\, x := x - 1\, \{\, F \,\}}}{\{\, F \,\}\ \mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\ \{\, F \wedge x \not> 0 \,\}}\ \text{mw} \quad (F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)}{\{\, x \geq 0 \,\}\ \mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\ \{\, x \geq 0 \wedge x \not> 0 \,\}}\ \text{lc}$$

for some suitable invariant F. Note that lc and mw are the only rules that can have our counter-example as conclusion. Moreover, mw has to be applied once since it is

the only rule that can introduce a while statement. (In fact, the logical consequence rule can be applied several times, but the effect of several applications can always be achieved with just one application.)

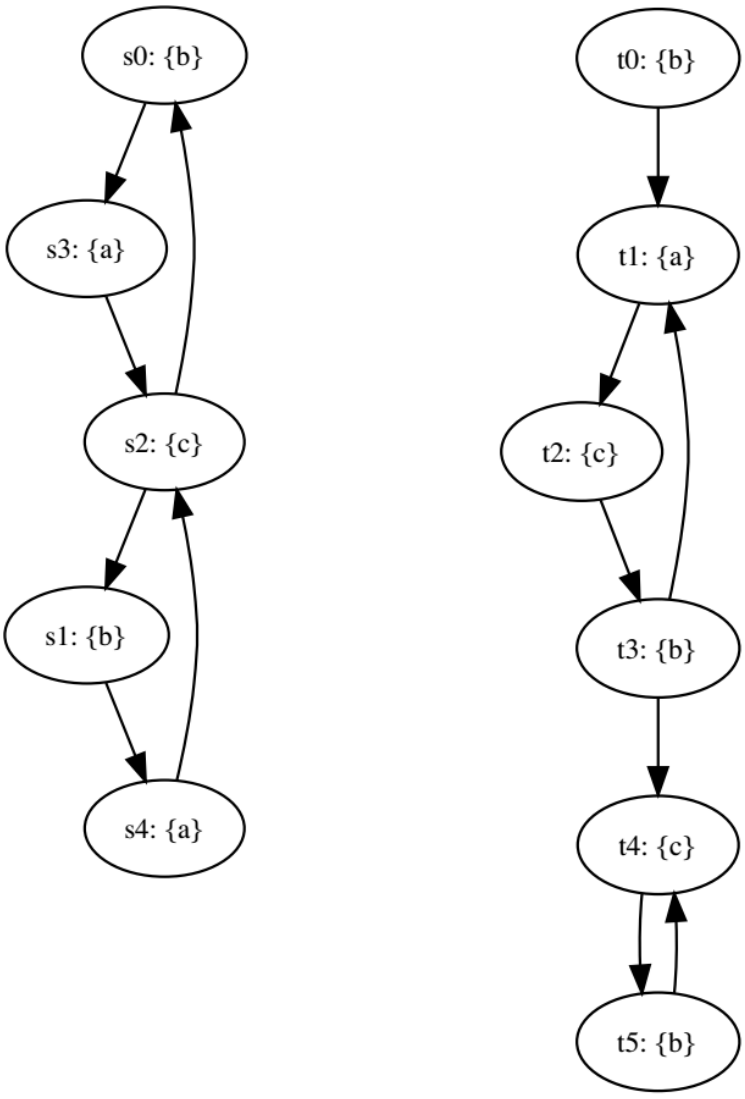
Now, if such a derivation indeed exists, then the formulas $x \geq 0 \Rightarrow F$ and $(F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)$ are valid and the correctness assertion $\{\, F \,\}\, x := x - 1 \,\{\, F \,\}$ is true (since it can be derived). We show that this cannot happen simultaneously, hence the derivation does not exist. This means that the calculus is incomplete, since we have found a true correctness assertion that cannot be derived.

Consider a state σ with $\sigma(x) = 0$. Since $x \geq 0 \Rightarrow F$ is supposed to be valid and $x \geq 0$ is true for σ, F must also be true for σ. Since $\{\, F \,\}\, x := x - 1 \,\{\, F \,\}$ is true, we conclude that F is also true for σ', where $\sigma'(x) = -1$ (state after executing the assignment). But the implication $(F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)$ does not hold for σ': The premise is true since $\sigma'(x) \not> 0$, but the conclusion $x \geq 0 \wedge x \not> 0$ is not true, since σ' does not satisfy $x \geq 0$.

Block 4

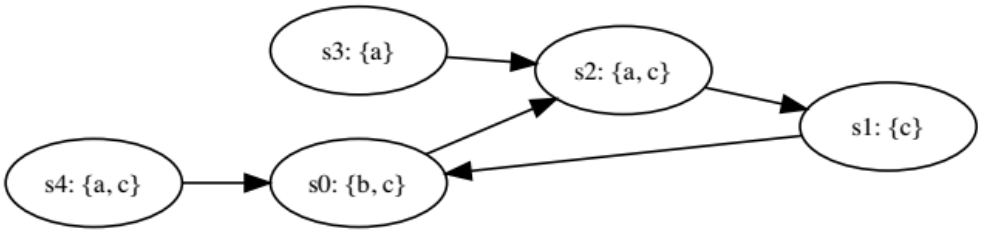
4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1: **Kripke structure M_2:**



Solved by student: H = {(s0,t0), (s3, t1), (s2, t2), (s0, t3), (s1,t3), (s4, t1)}

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{G}(c)$	☐	☐	☐	
$((a)\ \mathbf{U}\ (a))$	☐	☐	☐	
$\mathbf{AF}(b)$	☐	☐	☐	
$\mathbf{EG}(b)$	☐	☐	☐	
$\mathbf{E}[(b)\ \mathbf{U}\ (a)]$	☐	☐	☐	

(5 points)

Solved by student, verified by Kripke Builder:

G(c)	LTL, CTL*	s4, s0, s2, s1
((a) U (a))	LTL, CTL*	s4, s2, s3
AF(b)	CTL, CTL*	s0, s4, s1, s2, s3
EG(b)	CTL, CTL*	-
E[(b) U (a)]	CTL, CTL*	s4, s2, s3, s0

(c) Consider the LTL formulas $\mathbf{GF}(p\mathbf{U}q)$ and $\mathbf{GF}q$. Prove that they are equivalent or give an example showing that they are not equivalent.

(6 points)

Solved by student:

I think they are **equivalent**.

"=>": Assume we have an M, such that M |= GF(pUq). This means for every initial state s of M and every path π starting at s and every i >= 0, M,π^i |= F(pUq). This further means that there is some k >= i, such that M,π^k |= pUq. This means that there is some n>=k, such that M,π^n |= q (and also some k<=j<n such that M,π^j |= p, but this doesn't matter in this case). This means that M,π^n |= Fq and also M,π^k |= Fq and also M,π^i |= Fq and as such M,π |= GFq. As π was chosen arbitrarily, M |= GFq also holds.

"<=": Assume we have an M, such that M |= GFq holds. This means for every initial state s of M and every path π starting at s and every i >= 0, M,π^i |= Fq. This means that there is some k >= i such that M,π^k |= q. But then also M,π^k |= pUq and by the definition of "F" M,π^k |= F(pUq) and M,π^i |= F(pUq) and M,π |= GF(pUq). Since π was chosen arbitrarily, M |= GF(pUq) also holds.

Exam 12.12.2017 (FMI.176.pdf)

Block 1

1.) Consider the following problem:

> **3-COLORABILITY-MIRROR**
>
> INSTANCE: A pair (G, x) with $G = (V, E)$ an undirected graph and $x \in V$ a vertex.
>
> QUESTION: Is it true that $G^* = (V^*, E^*)$ is 3-colorable, where G^* is defined via vertices $V^* = V \cup \{v' \mid v \in V\}$ and edges $E^* = E \cup \{[u', v'] \mid [u, v] \in E\} \cup \{[x, x']\}$?

By providing a suitable reduction from the standard **3-COLORABILITY** problem, prove that **3-COLORABILITY-MIRROR** is an NP-hard problem. Argue formally that your reduction is correct.

Recall that **3-COLORABILITY** is defined as follows:

> **3-COLORABILITY**
>
> INSTANCE: An undirected graph $G = (V, E)$.
>
> QUESTION: Does there exists a function μ from vertices in V to values in $\{0, 1, 2\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

Hint: If there is a valid coloring μ for a graph G then there is also also a valid coloring μ' for G which "swaps" colors (i.e. $\mu(v) \neq \mu'(v)$ for each vertex v in G).

Status: solved by student

=> Let G be an arbitrary positive instance of 3-colorability, with μ(v1) ≠ μ(v2) for any edge [v1, v2] ∈ E. G* contains G as a subgraph by construction, so half of G* is already 3-colored. The other half can not simply use μ as a coloring function because of the edge [x, x'] ∈ E*. Let therefore μ' be a "swaping coloring function" with μ'=(μ(v)+1 % 3) (we swap/rotate the initial coloring by one). μ'(v1') ≠ μ'(v2') for every v'∈V* and μ'(v')≠μ(v) for every v'∈V* and v ∈ V. Use μ' for the second half of G* and the 3 coloring property is still preserved for G and further G*.

<= Let G* be an arbitrary positive instance of 3-colorability-mirror. G* contains by definition a subgraph G=(V,E) which has also to hold that it is 3-colorable (otherwise G* would not be 3-colorable). From that we know G is a positive instance of 3-colorability.

Block 2

2.) (a) Use a semantic argument to prove the $\mathcal{T}_A$-validity of the following Σ_A-formula, or provide a counterexample (i.e., a falsifying $\mathcal{T}_A$-interpretation):

$$a\langle i \triangleleft e\rangle[j] = e \rightarrow i = j \vee a[j] = e$$

(8 points)

Status: solved by student

φ = a<i ◁ e>[j] = e →i = j ∨a[j] = e

1.	I ⊭ φ	assumption
2.	I |= a<i ◁ e>[j] = e	1. Semantics of →
3.	I ⊭ i = j ∨a[j] = e	1. Semantics of →
4.	I |= i≠j ∧a[j] ≠ e	3. Semantics of ¬
5.	I |= i≠j	4. Semantics of ∧
6.	I |= a[j] ≠ e	4. Semantics of ∧
7.	**I |= a<i ◁ e>[j]=a[j]**	**5. Read over write 2**
8.	I |= a[j] = e	2., 7., Symmetry, Transitivity
9.	I |= ⊥	6., 8., contradiction

Formula is T_A-valid.

(b) Is the following Σ_A-formula $\mathcal{T}_A$-valid? Justify your answer.

$$a\langle i \triangleleft e\rangle[j] = e \rightarrow i = j$$

(1 point)

(c) Is the following Σ_A-formula $\mathcal{T}_A$-valid? Justify your answer.

$$a\langle i \triangleleft e\rangle[j] = e \rightarrow a[j] = e$$

(1 point)

b) No, this implication only holds in the other direction (read-over-write 1). i ≠ j would still allow the given implication.

c) No, if i = j, then a[j] could be anything, therefore the implication does not hold.

(d) Let ψ be a propositional formula in CNF and let $S := \{C \in \psi \mid \ell \in C\}$ be the set of all clauses C in ψ that contain ℓ, where ℓ is the literal of some arbitrary but fixed variable. Assume that literal $\neg\ell$ does not occur in any clause in ψ.
Let ψ' be the CNF obtained from ψ by removing all clauses in S: $\psi' := \psi \setminus S$.
Give a detailed proof of the following statement:

ψ is satisfiable if and only if ψ' is satisfiable.

(5 points)

d) **Status: solved by student- not sure if formal enough** (it is very similar to 171 - Block 2)

=> assume Ψ is sat, this means that in each clause of Ψ there is at least one literal to make the conjunction over all clauses true.
Case 1: If every clause in Ψ contains the literal L, then Ψ' = {} and the conjunction of an empty set is always true.
Case 2: If Ψ contains clauses which do not have L in them, they need some other literal L' to make them true (because of the assumption Ψ is sat). In Ψ' there are only these clauses left with the literal L' which makes them true under Ψ. So Ψ' is also sat.

<= assume Ψ' is sat.
Case 1: Ψ' = {} the conjunction of an empty set is always true. This means that Ψ consists only of clauses where L is present. Let L = true and this makes Ψ sat.

Case 2: Ψ' ≠ {} so we have some clauses C' with literals L'' which have to be true to make Ψ' sat. Because Ψ' is a subformula of Ψ and Ψ additionally only has clauses where we have the literal L (and the negation of L is never in this set), then let this literal L = true. This makes Ψ sat.

Block 3

3.) Let π be the program $x := x - y;\ y := x + y;\ x := y - x$.

(a) Specify a correctness assertion stating that this program swaps that values of the variables x and y. **(1 point)**

(b) Prove the correctness assertion using weakest preconditions. **(5 points)**

(c) Prove the correctness assertion using strongest postconditions. **(9 points)**

a) **{x=x0 and y=y0}** x=x-y;y=x+y;x=y-x **{x=y0 and y=x0}**
b)

i) Calculate wp(x=x-y;y=x+y;x=y-x, x=y0 and y=x0) = wp(x=x-y, wp(y=x+y, wp(x=y-x, x=y0 and y=x0)
ii) show **{x=x0 and y=y0} => wp**

c)

i) Calculate sp(x=x0 and y=y0, x=x-y;y=x+y;x=y-x) = sp(sp(sp(x=x0 and y=y0, x=x-y), y=x+y), x=y-x)
ii) show sp => **{x=y0 and y=x0}**

Block 4

Same as Exam 20.10.2017
Block 4 - 175 - 20.10.2017

Exam 20.10.2017 (FMI.175.pdf)

Block 1

Solved by student:

Assuming G is a positive instance of the 2-COLORABILITY problem, then there is a coloring function μ, such that μ(vi) != μ(vj) for all (vi, vj) of E. Now let these colors be the values “true” and “false”. This means that μ assigns every vertex either the value “true” or the value “false”. We can represent this using variables of the form xi, that represent the truth-value of the vertex vi.
Obviously, since μ is a valid 2-coloring, (xi or xj) and (not xi or not xj) is valid for all (vi, vj) of E, and hence PhiG is satisfiable.

Solution from *Exercises(Examples) SS2018:*

Solution to Exercise 5

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 $f : V \to \{0, 1\}$ 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 as follows. For all $i \in \{1, \ldots, n\}$,

- $T(x_i) = \mathit{true}$ if $f(v_i) = 1$, and
- $T(x_i) = \mathit{false}$ if $f(v_i) = 0$.

It remains to show that φ_G evaluates to *true* under T.

Solution to Exercise 5 (cnt'd)

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.

Block 2

a) **Solved by student:** (not sure if formal enough)

For φ^{EUF} to be E-satisfiable there has to be an assignment making all conjuncts true. If at least one conjunct is not satisfiable, the whole formula becomes E-unsatisfiable.

The conjuncts of φ^{EUF} are listed in the following.
C1: x=y
C2: f(x) = g(y)
C3: z = g(f(y))
C4: z != g(f(x))
C5: P(g(f(y)), x)

From C1 it follows that x equals y.

Using this information in combination with C2 leads to f(x) = g(x).
The functional consistency axiom now says that the functions f and g need to be equal.
Substituting z in C4 with the z given by C3 leads to formula (VI): g(f(y)) != g(f(x)).
As we stated above x = y and g(x) = f(x).
So the formula (VI) can be simplified to f(f(x)) != f(f(x)). This is obviously a contradiction.

Using the first four clauses it was possible to show that there cannot be a valid assignment for this EUF-formula. Therefore it is E-unsatisfiable.

b) **Solved by student:**

$$\varphi^{EUF}:\ x=x\ \wedge\ f(x)=g(y)\ \wedge\ z=g(f(y))\ \wedge\ z\neq g(f(x))\ \wedge\ P(g(f(y)),x)$$

After removing the uninterpreted predicate:

$$\psi^{EUF}:\ x=x\ \wedge\ f(x)=g(y)\ \wedge\ z=g(f(y))\ \wedge\ z\neq g(f(x))\ \wedge\ Fp(g(f(y)),x)=x_p$$

Calculating the flat:

$$flat^E(\psi^{EUF}):=\ x=y\ \wedge\ f_1=g_1\ \wedge\ z=g_2\ \wedge\ z\neq g_3\ \wedge\ f_3=x_p$$

Calculate functional constraints:

$$FC^E(\psi^{EUF}):\ FC^E_F(\psi^{EUF})\ \wedge\ FC^E_G(\psi^{EUF})\ \wedge\ FC^E_{Fp}(\psi^{EUF})$$

$$FC^E_F(\psi^{EUF}):\ x=y\ \rightarrow\ f_1=f_2$$

$$FC^E_G(\psi^{EUF}):\ y=f_2\rightarrow g_1=g_2\ \wedge\ y=f_1\rightarrow g_1=g_3\ \wedge\ f_2=f_1\rightarrow g_2=g_3$$

$FC^E_{Fp}(\psi^{EUF})$: // I assume this term is empty. Confirmation / correction would be appreciated.

The functional constraints therefore are:

$$FC^E(\psi^{EUF}):\ (x=y\ \rightarrow\ f_1=f_2)\ \wedge\ (y=f_2\rightarrow g_1=g_2)\ \wedge\ (y=f_1\rightarrow g_1=g_3)\ \wedge\ (f_2=f_1\rightarrow g_2=g_3)$$

The equality logic formula ψ^E is SAT iff ψ^{EUF} is SAT:

$$\psi^E:\ FC^E(\psi^{EUF})\ \wedge\ flat^E(\psi^{EUF})$$

$$\psi^E:\ ((x=y\ \rightarrow\ f_1=f_2)\ \wedge\ (y=f_2\rightarrow g_1=g_2)\ \wedge\ (y=f_1\rightarrow g_1=g_3)\ \wedge\ (f_2=f_1\rightarrow g_2=g_3))\ \wedge\ (x=y\ \wedge\ f_1=g_1\ \wedge\ z=g_2\ \wedge\ z\neq g_3\ \wedge\ f_3=x_p)$$

Block 3

A & b see [144 - Block 3]
c) Characterize all programs p that satisfy wp(p, true) = true, i.e. specify a condition such that the equation holds exactly when p satisfies the condition.

p must be a program that terminates, i.e. not run into an abort or endless loop.

Argument (please check)

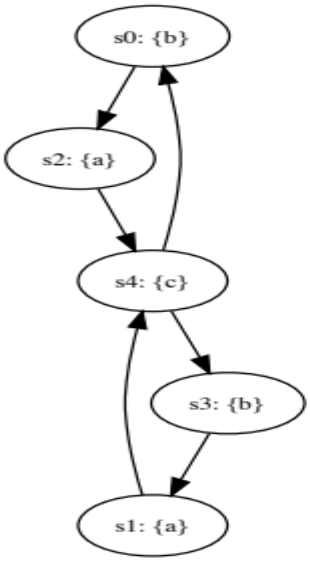
- True ist the weakest formula, it is implied by everything.
- By definition, Sin p Sout is totally correct, iff Sin => wp(p, Sout).
- As everything implies “true”, Sin => true and thus p is totally correct.

Block 4

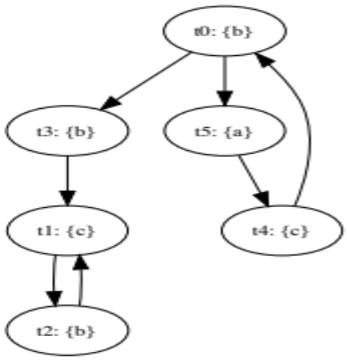
a)

(a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



Kripke structure M_2:



Solved by student:

H: {(s_0, t_0), (s_2, t_5), (s_4, t_4), (s_3, t_0), (s_1, t_5)}

b) **Solved by student:**

	CTL	LTL	CTL*	states
F(b ∧c)		X	X	$\{s_0,s_1,s_2,s_3,s_4\}$
X(a ∧ b ∧c)		X	X	{}
b U b		X	X	$\{s_1,s_2,s_4\}$
EG c	X		X	$\{s_0,s_1\}$
E ((a ∧ b) U a)	X		X	$\{s_0,s_2,s_3,s_4\}$

c) **Solved by student:**

i) Mark all states where p is valid
ii) Unmark all states that do not have a marked successor
iii) Repeat II until nothing changes anymore. (fixpoint)

Now all marked states satisfy p and have a successor satisfying p. So on every marked state a path starts satisfying p.

Alternative solution (maybe a very inefficient one, but efficiency is not required):

1) Mark all states satisfying p
2) Run through all paths between s1 and s2, for each s1, s2 in S and s1, s2 satisfying p and s1 != s2
    a) If there is no state on the path between s1 and s2 not satisfying p, then add s1 to the result list
3) Repeat 2) until all states have been tried -> Result-List contains all s satisfying EGp

Exam 30.06.2017 (FMI.174.pdf)

Block 1

1.) Consider the following problem:

> **LOOPS-OR-HALTS (LOH)**
>
> INSTANCE: A tuple (Π_1, Π_2, I_1, I_2), where I_1, I_2 are strings, and Π_1, Π_2 are programs that take a string as input.
>
> QUESTION: Does *at least one* of the following conditions hold? The conditions are:
>
> (a) Π_1 does not halt on I_1
>
> (b) Π_2 halts on I_2

Provide a reduction from **co-HALTING** to **LOOPS-OR-HALTS**. Argue formally that your reduction is correct.

HINT: Recall that **co-HALTING** is defined as follows:

> **co-HALTING**
>
> INSTANCE: A pair (Π, I), where I is a string and Π is a program that takes a string as input.
>
> QUESTION: Is it true that Π does not halt on I?

Status: solved by student (please approve if correct)

- Let (Π_1,I_1) be an arbitrary instance of co-HALTING.
- Let (Π_2,I_2) be a positive instance of co-HALTING (Π_2 does not halt on I_2).
- We construct an instance of LOH as follows: (Π_1,Π_2,I_1,I_2)

Due to the construction, condition (b) can never be true (as Π_2 does not halt on I_2), hence it remains to show that (Π_1,I_1) is a positive instance of co-HALTING $\Leftrightarrow$ (Π_1,Π_2,I_1,I_2) is a positive instance of LOH.

$\Rightarrow$ If (Π_1,I_1) is a positive instance of co-HALTING, Π_1 does not halt on I_1 and therefore condition (a) is fulfilled and (Π_1,Π_2,I_1,I_2) is a positive instance of LOH

⇐ If (Π_1,Π_2,I_1,I_2) is a positive instance of LOH, condition (a) has to be true (as condition (b) cannot be true due to our construction). Therefore $\Pi_1$does not halt on I_1 and (Π_1,I_1) is a positive
instance of co-HALTING.

Block 2

(a) Let φ^E be any E-formula with Boolean variables $b_1, \ldots, b_n$. Construct an E-formula ψ^E without any Boolean variable by replacing each b_i $(i = 1, \ldots, n)$ by an equality e_i of the form $v_i \doteq w_i$, where $v_1, w_1, \ldots, v_n, w_n$ are new distinct term variables (identifiers). Prove: ψ^E is E-satisfiable if φ^E is E-satisfiable. **(11 points)**

Solution by professor:

Tuwel -> block 2 -> supplementary material folder -> extra-sheet4-1a.pdf

(b) Answer the following questions to the CDCL algorithm and justify your answers in detail.

i. Suppose that CDCL learns a unit clause C. Given C, to which decision level does CDCL backtrack?

ii. Consider a run of CDCL on a given CNF F, and suppose that the run has terminated. What is the current decision level in CDCL at the time when CDCL terminates?

Student solution:

i) To the second highest decision level in the learned clause (without erasing that decision). Then, the clause should get unit.

Algorithm 2.2.2: ANALYZE-CONFLICT

Input:
Output: Backtracking decision level + a new conflict clause

```
1. if current-decision-level = 0 then return -1;
2. cl := current-conflicting-clause;
3. while (¬STOP-CRITERION-MET(cl)) do
4.     lit := LAST-ASSIGNED-LITERAL(cl);
5.     var := VARIABLE-OF-LITERAL(lit);
6.     ante := ANTECEDENT(lit);
7.     cl := RESOLVE(cl, ante, var);
8. add-clause-to-database(cl);
9. return clause-asserting-level(cl);        ▷ 2nd highest decision level in cl
```

ii) If CNF F is unsatisfiable, then -1 *0*, otherwise TODO

Not a solution but a thought towards the remaining TODO - discussion welcome

If we are at decision level x with the first conflict clause, in the worst case we have to backtrack n-times to the second highest DL. under the assumption F will become SAT at some point, in the worst case we backtrack to the n^{th} second highest DL from the first conflict clause.

Block 3

OBJ

Solution by student:

Counter example:
{F}: {x = 1}
{G}: {x = 2}
p: x:= x+1
q: x:= x+1

It remains to show that {F} p {G} and therefore {F} q {G} holds (via Hoare calculus or WLP).

(b) Characterize all programs p that satisfy $\mathrm{wp}(p, \mathrm{true}) = \mathrm{false}$, i.e., specify a condition such that the equation holds exactly when p satisfies the condition. **(4 points)**

Solved by student:
Solution 1:t/images/f/f4/TU_Wien-Formale_Methoden_der_Informatik_VU_%28Egly%29_-_Sample_Solutions_SS18_Block_3_Additional_Exercises.pdf – 更改 | 移除

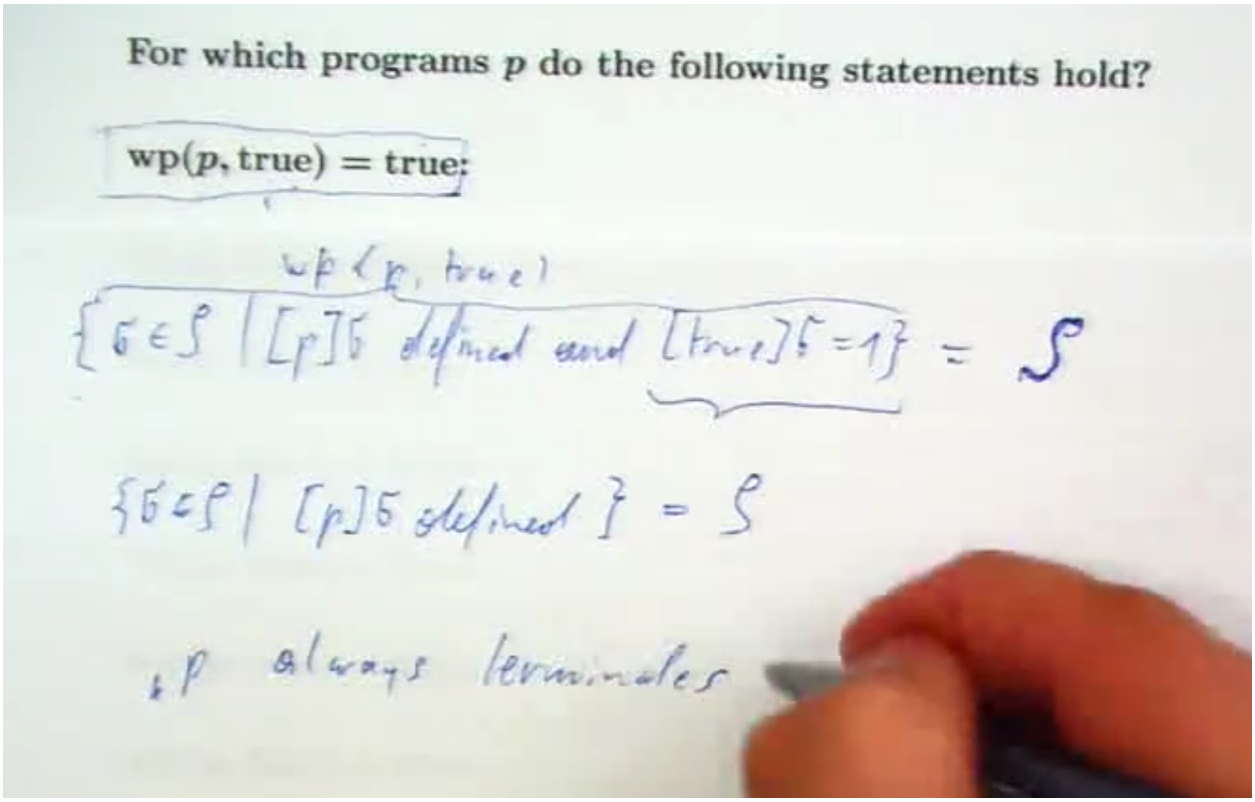
Programs p which do not terminate, i.e. p hits an abort statement or runs into an infinite loop on whichever input is supplied.
More formally:
wp(p, true) = false means that there are no such F-states such that the execution of p terminates and we end up in a state where true holds, i.e. any state.
Therefore programs which do not terminate satisfy the condition.

See also Exercise 6 (“A program p satisfies this property iff it does not terminate for any input.”, Similar questions also answered in this exercise)

Solution 2: (informal argument from Solution 1 but a bit more formal)
The weakest precondition (call it F') for any program p and postcondition G (wp(p,G)=F') must be implied by any other correct precondition F
So for every {F}p{G} where after executing p G holds, F => F'
Since we have given that the wp(p,true) = false, we have to find a precondition F that satisfies F => false.
Obviously the only possibility for F is false, therefore wp(p,true) = false can only hold for programs p that are totally correct if and only if the weakest precondition is false.
Only two cases satisfy this requirement, the ABORT statement, and endless loops.
Total correctness requires termination, so for programs that don't terminate (or abort) the precondition must be false
For all other programs, some F can be found (actually i think for every terminating program {true} p {true} is valid) such that F =/=> false (e.g. true =/=> false) and therefore false can not be the weakest precondition.

Solution 3: Solution by Prof. Salzer when showing example during lecture

Formal Methods in Computer Science - Block 3 (Recap1 vom 2018-05-09)
https://oc-presentation.ltcc.tuwien.ac.at/engage/theodul/ui/core.html?id=7ed14ec4-91b3-442b-af24-5a4c60df63cb (ab 01:23:49)



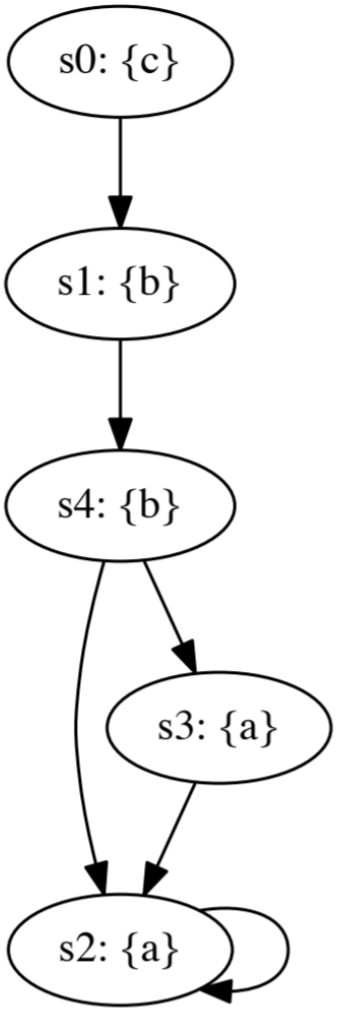
(c) Characterize all programs p that satisfy $\text{wlp}(p, \text{true}) = \text{false}$, i.e., specify a condition such that the equation holds exactly when p satisfies the condition. **(4 points)**

Alternative solution (same argumentation as solution 2 above):
wlp(p,true) = false can only hold for programs for which false ({false} p {true}) is the only possible precondition. However unlike for total correctness, {true} abort {true} is partially correct, and also an endless loop ({true} while 1 do skip od {true}) is also partially correct
Therefore for any program a precondition F can be found (again, true is a possible precondition) such that F =/=> false
Conclusion: There is NO such program p

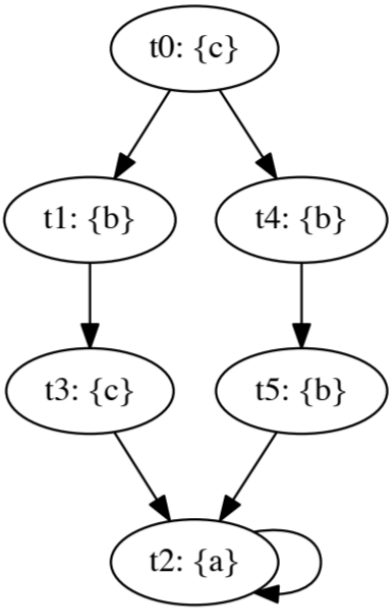
Block 4

(a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



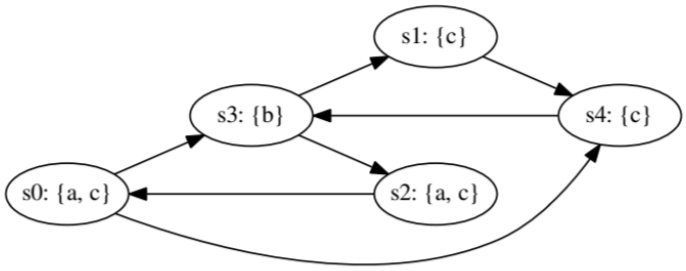
Kripke structure M_2:



solved by student:

H = {(s_0, t_0), (s_1, t_4), (s_2, t_2), (s_3, t_2), (s_4, t_5)}

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and

ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

φ	CTL	LTL	CTL*	States s_i
$\mathbf{G}(c)$	☐	☐	☐	
$\mathbf{F}(a)$	☐	☐	☐	
$\mathbf{AX}(c)$	☐	☐	☐	
$((a \wedge c)\ \mathbf{U}\ (b))$	☐	☐	☐	
$\mathbf{EF}(a)$	☐	☐	☐	

solved by student:

phi	CTL	LTL	CTL*	states
G(c)		*	*	-
F(a)		*	*	s0, s2
AX(c)	*		*	s1, s2, s3
[(a & c) U (b)]		*	*	s3
EF(a)	*		*	s0 ,s1, s2, s3, s4

Given a graph, write a C program such that CBMC can determine whether the given graph is 3-colorable. Augment the given code corresponding to the following subtasks. The 2-dimensional array `graph` encodes the adjacency matrix.

```
#define TRUE  1
#define FALSE 0

#define RED   0
#define GREEN 1
#define BLUE  2

#define N     4 // Number of nodes in the graph

int graph[N][N] = { { 0, 1, 0, 1 }, { 1, 0, 0, 0 }, ... };
int coloring[N];

int nondet_int();
```

i. Write a loop that nondeterministically guesses a coloring for the graph. A coloring assigns to every node of the given graph either the color red, green, or blue.

ii. Write a loop that checks whether the coloring assigns to every node in the graph a color that is different to the colors of its neighbors. Furthermore, ensure that CBMC reports a 3-coloring of the graph in case there exists one.

Approach by student:

```
// each node gets exactly one color
for(int i=0; i < N; ++i) {
  coloring[i] = nondet_int() % 3;
}

int three_colorable = 1;

for(int i=0; i < N; ++i) {
  for(int j= i+1; j < N, ++j) {
    if( graph[i][j] == 1 && coloring[i] == coloring[j]) {
      // nodes are connected and same colour
           three_colorable = 0;
     }
    }
  }
// report if 3-colorable (assert that we did not find a solution)
assert(!three_colorable);
```

Exam 05.05.2017 (FMI.173.pdf)

Block 1

HALTING-NO-INPUT
INSTANCE: A program Π where Π takes no input.
QUESTION: Does Πterminate?

By providing a reduction from HALTING, prove that HALTING-NO-INPUT is undecidable. Argue formally that your reduction is correct.

SOLUTION: Let (π, I) be an arbitrary instance of halting. We build an instance (π`) of HNI by constructing $\pi$` as follows:

```
π'() {
   π(I);
   return;
}
```

To prove the reduction we have to show that (π, I) is a positive instance of Halting ⇔ (π') is a positive instance of HNI.

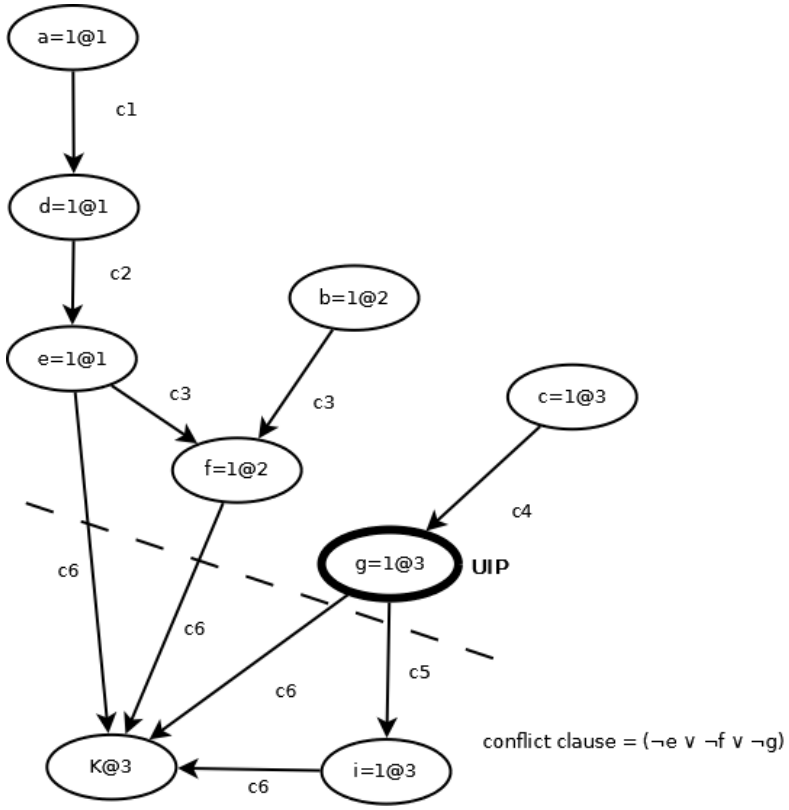
=> Assume (π, I) is a positive instance of Halting. This means that π terminates and π' reaches the return statement. Therefore π' is a positive instance of HNI.

<= Assume (π') is a positive instance of HNI. This means that π' reaches the return statement. This means that π has to terminate and therefore has to be a positive instance of Halting.

Block 2

a) Same as 174/2/a (proof by prof in supplementary materials: extra-sheet4-1a.pdf)

b) **Solution suggested by student:**



Block 3

a) { x=1 } x=2 { x=1 and x=2} x=1 and x=2 is obviously wrong

(b) Use weakest preconditions to compute a description of all states from which the following program will terminate.

$y := 3x;$
while $2x \neq y$ do
  $x := x + 2;$
  $y := y + 1$
od

(10 points)

First of all we have to calculate the wp of the while loop:

wp(while…, G) = F0 v F1 v F2 v F3 …. = $\exists i$ (i>=0 $\wedge$ Fi)

F0 = ¬e $\wedge$ G

Fi+1 = e $\wedge$ wp(p,Fi) (i>=0)

F0: 2x = y

F1: 2x + 3 = y

F2: 2x + 6 = y

F3: 2x + 9 = y

Guess Fi: 2x + 3i = y

Prove by Induction:

Base case:

F0: 2x + 3*(0) = y

F0: 2x = y Fine

Induction Step Fi+1:

Fi+1 = e ∧ wp(p,Fi)

Fi+1 = 2x != y ∧ 2x + 3 * (i+1) = y Fine

wp(while..,G) = ∃ *i* (i>=0 ∧ 2x + 3i = y)

= ∃ *i* (i>=0 ∧ i = $\frac{y-2x}{3}$)

= $\frac{y-2x}{3}$ >= 0 ∧ ∃ *i* (i = $\frac{y-2x}{3}$)

= $\frac{y-2x}{3}$ >= 0

Danach muss nur noch y:=3x berücksichtigt werden und wir sind fertig:

wp(y:=3x, $\frac{y-2x}{3}$ >= 0) = $\frac{x}{3}$ >= 0

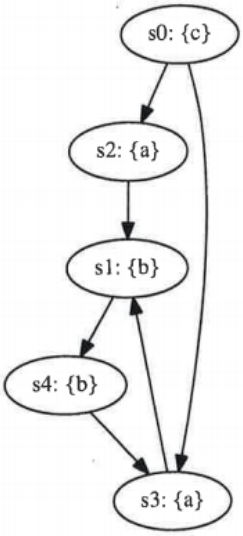
Somit ist unsere Weakest precondition:

{F} = {$\frac{x}{3}$ >= 0}, x muss also ein Vielfaches von 3 sein (0, 3, 6, 9, …)

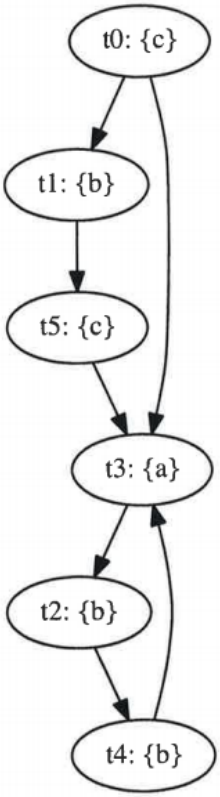
Block 4

a)

Kripke structure M_1:



Kripke structure M_2:



$H = \{(s_0, t_0), (s_2, t_3), (s_1, t_2), (s_4, t_4), (s_3, t_3)\}$

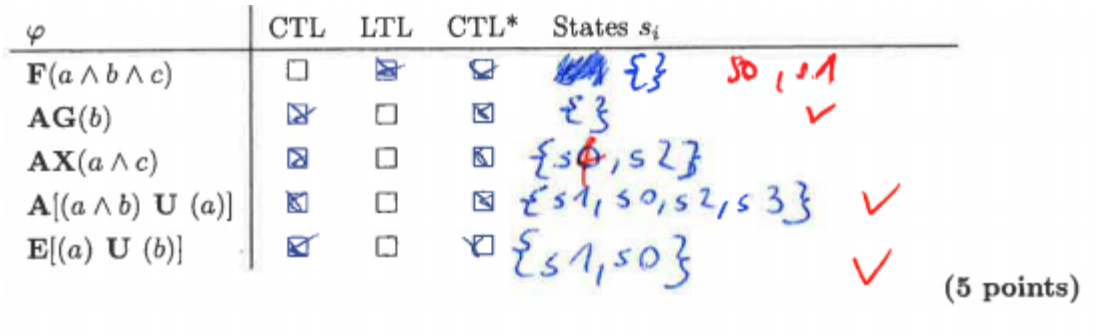
4

(4 points)

b)

phi	CTL	LTL	CTL*	states
F(a&b&c)		*	*	s0,s1
AG(b)	*		*	{}
AX(a&c)	*		*	s2,s4
A[(a & b) U (a)]	*		*	s0,s1,s2,s3
E[(a) U (b)]	*		*	s0,s1

)



c)

Status: **solved by student**

Ga -> Fb = a U (b v !a)

i) If K,s0 |= Ga -> Fb then K,s0 |= a U (b v !a)

Let p = s0,s1… be some path starting at s0. Since K,s0 |= Ga -> Fb we know that there can be the following cases:

- There is some state si for i >= 0 such that K,si |= b holds and for every j **<=** i we know that at least K,sj |= a holds.
- There is some state si for i >= 0 such that K,si |= !a holds and for every j **<** i we know that K,sj |= a holds.

Now consider the second formula where it clearly holds at **either** i>= 0 K,s0 |= b and hence K,s0 |= Fb **or** there is some state si for i >= 0 such that K,si |= !a and for every state j < i K, sj |= a holds and hence the formula defaults to true.
Since p was chosen arbitrarily K,s0 |= a U (b v !a).

ii) If K,s0 |= a U (b v !a) then K,s0 |= Ga -> Fb

Let p = s0,s1… be some path starting at s0. Since K,s0 |= a U (b v !a) we know that there can be the following cases:

- There is some state si for i >= 0 such that K,si |= b holds and for every j **<=** i we know that at least K,sj |= a holds.
- There is some state si for i >= 0 such that K,si |= !a holds and for every j **<** i we know that K,sj |= a holds.

Now consider the second formula where it clearly holds that either at some point i >= 0 K, si |= !a holds and hence K, s0 |!= Ga can't hold or there is some i >= such that K,si |= Fb holds.
Since p was chosen arbitrarily K,s0 |= Ga -> Fb.

Status: solved by student, verified with Kripke Builder
Proof that the following LTL formulae are not equivalent:

(Fa) & (XGa) != Fa

Let's suppose (Fa) & (XGa) = Fa is true and the following Kripke structure is given:

(S0: b) -> (S1: a) -> (S2: b) -> ...

Fa is obviously true for S0 but Fa & XGa does not hold in S0. Hence the equivalence of the 2 formulae can not hold.

Exam 17.03.2017 (FMI.172.pdf)

Block 1

Status: **solved by student**, please verify

Let G = (V, E) be an arbitrary instance of the 2-COLORABILITY problem. We construct an instance of the 3-COLORABILITY problem as follows:

G' = (V', E') where V' will contain a new vertex w, i.e. V' = V u {w} and E' will contain new edges from all vertices in V to w, i.e. E' = E u {[vi n, w] | vi el V}. Later on, we will adapt the coloring of the 2-COLORABILITY problem μ s.t. The new coloring μ' will assign the 3rd color only to the new vertex w.

It remains to see that G = (V, E) is a positive instance of 2-COLORABILITY iff G' = (V', E') is a positive instance of 3-COLORABILITY:

=> : Assume that G is a positive instance of 2-COLORABILITY, then there exists a μ that assigns different colors to adjacent vertices. Based on the construction of G', we construct a valid 3-coloring for G' by constructing a μ' that assigns the same color to the same vertices as in V plus the new (3rd) color to the newly introduced vertex w. Due to the fact that no other vertex in V had the new color, and that μ was a valid 2-coloring for G, it must be the case the μ' is a valid 3-coloring for G'. Hence, G' is a positive instance of 3-COLORING.

=> : Assume that G is a negative instance of 2-COLORABILITY, then there does not exist a valid μ that assigns different colors to adjacent vertices. By construction of G', it is impossible that G' has a valid coloring either. Assume there exist a 3-coloring μ' for G'. Then, it must be the case that all vertices from V have only the first two colors assigned (remember, we added edges from all vertices in V to the new vertex w). Hence, there is a 2-coloring for the subset of vertices and edges coming from G, which result in a valid 2-coloring for G. However, this violates the assumption that there exists a valid 3-coloring for G'. Therefore G' is also a negative instance of 3-COLORABILITY.

---

alternative <=: Assume that G is a positive instance of 3-COLORABILITY. As by the construction of G' and μ', only a single vertex has the new color (3) assigned. This means that removing this vertex and the edges involving it, leaves a graph in which all adjacent vertices get a different color from {1, 2} assigned by μ which makes G_2 a positive instance of 2-COLORABILITY by definition.

Block 2

(a) Clarify the logical status of each of the following formulas. If one is $\mathcal{T}^E_{cons}$-valid or $\mathcal{T}^E_{cons}$-unsatisfiable, then prove it using semantics. If one is $\mathcal{T}^E_{cons}$-satisfiable but not $\mathcal{T}^E_{cons}$-valid, then present a satisfying and a falsifying interpretation. Argue formally that the formula evaluates to true resp. false under the constructed interpretations.

i. φ_0: $cons(car(x), cdr(x)) \doteq cons(y, z) \wedge cons(car(x), cdr(x)) \not\doteq x \rightarrow x \not\doteq cons(y, z)$

ii. φ_1: $\neg atom(x) \wedge car(x) \doteq y \wedge cdr(x) \doteq z \wedge x \not\doteq cons(y, z)$

iii. φ_2: $car(x) \doteq y \wedge cdr(x) \doteq z \wedge x \not\doteq cons(y, z)$

Besides the equality axioms, the following axioms of $\mathcal{T}^E_{cons}$ may be helpful.

- $\forall x, y\ car(cons(x, y)) \doteq x$ (left projection)
- $\forall x, y\ cdr(cons(x, y)) \doteq y$ (right projection)
- $\forall x\ \neg atom(x) \rightarrow cons(car(x), cdr(x)) \doteq x$ (construction)
- $\forall x, y\ \neg atom(cons(x, y))$ (atom)

a) Status: **solved by student**, **please verify**

Clarify the logical status of the following formulas. If one is T^E_{cons}-valid or T^E_{cons}-unsatisfiable, then prove it using semantics. If one is T^E_{cons}-satisfiable but not T^E_{cons}-valid, then present a satisfying and a falsifying interpretation. Argue formally that the formula evaluates to true resp. false under the constructed interpretations.

i. φ_0 = cons(car(x), cdr(x)) = cons(y, z) ∧ cons(car(x), cdr(x)) ≠ x → x ≠ cons(y, z)

Proof by contradiction: Suppose there exists a T^E_{cons} interpretation with I |≠ φ_0:

1.	I |≠ φ	assumption
2.	I |= cons(car(x), cdr(x)) = cons(y, z) ∧ cons(car(x), cdr(x)) ≠ x →	1, semantics of
3.	I |≠ x ≠ cons(y, z)	1, semantics of →
4.	I |= cons(car(x), cdr(x)) = cons(y, z) ∧	2, semantics of
5.	I |= cons(car(x), cdr(x)) ≠ x ∧	2, semantics of
6.	**I |= cons(y, z) ≠ x**	**4, 5, transitivity**
7.	**I |= x ≠ cons(y, z)**	**6, symmetry of ≠**
8.	I |= ⊥	3, 7, contradiction

The assumption is false, therefore φ_0 is T^E_{cons}-valid.

Alternatively:
if you don't want to check the transitivity this way (from the 6th clause on), you can use 4 and 3, to generate a 6th clause: I |= cons(car(x), cdr(x))=x and then use this 6th clause for the contradiction in combination with clause 5.

ii. φ_1 = ¬atom(x) ∧ car(x) = y ∧ cdr(x) = z ∧ x ≠ cons(y, z)

Suppose there exists a T^E_{cons}-interpretation with I |= φ_1:

1.	I |= φ	assumption
2.	I |= ¬atom(x)	1, semantics of ∧
3.	I |= car(x) = y	1, semantics of ∧
4.	I |= cdr(x) = z	1, semantics of ∧
5.	I |= x ≠ cons(y, z)	1, semantics of ∧
6.	I |= cons(car(x), cdr(x)) = x	2, construction
7.	I |= cons(y, z) = x	6, 3, 4, substitution
8.	I |= cons(y, z) ≠ cons(y, z)	5, 7, transitivity
9.	I |= ⊥	8, contradiction

The assumption is false, therefore φ_1 is T^E_{cons}-unsatisfiable.

iii. φ_2 = car(x) = y ∧ cdr(x) = z ∧ x ≠ cons(y, z)

Status: student solution; incorrect but leaving for the conversation in the sidebar

1.	I |= φ	
2.	I |= car(x) = y	1, semantics of ^
3.	I |= cdr(x) = z	1, semantics of ^
4.	I |= x ≠ cons(y,z)	1, semantics of ^
5.	I |= y = car(cons(y,z))	2, left projection
6.	I |= car(x) = car(cons(y,z))	2,5, substitution
7.	I |= x = cons(y,z)	6, functional congruence <- this is wrong!
8.	I |=⊥	4,7 contradiction

The assumption is false, therefore φ_2 is T^E_{cons}-unsatisfiable.

Correct solution:

- provide satisfying solution, i.e. in case x is an atom
- provide falsifying solution, i.e. in case x is not an atom

b) **Status: solved by student**

To show that the resolution rule is sound, one has to show that the resolvent is a logical consequence of the resolved clauses, i.e. $\{C_1, C_2\} \models R$ where R is the resolvent of the

clauses C_1 and C_2. R is defined as $C_1 \setminus \{l\} \cup C_2 \setminus \{\neg l\}$ where l and ¬l are dual literals.

Towards a contradiction, assume there is a model I which satisfies two arbitrary clauses with dual literals C_1 and C_2 but does not satisfy their resolvent R:

1.	$I \vDash \{C_1, C_2\}$	assumption
2.	$I \nvDash R$	assumption
3.	$I \vDash C_1$	semantics $\wedge$
4.	$I \vDash C_2$	semantics $\wedge$
5.	$l \vee \neg l$	lem.
6.	Case 1: l	
7.	$I \vDash C_2 \setminus \{\neg l\}$	semantics $\wedge$, 4, def. C_2
8.	$I \vDash C_1 \setminus \{l\} \vee C_2 \setminus \{\neg l\}$	pr. $\vee$, 7
9.	Case 2: ¬l	
10.	$I \vDash C_1 \setminus \{l\}$	semantics $\wedge$, 1, def. C_1
11.	$I \vDash C_1 \setminus \{l\} \vee C_2 \setminus \{\neg l\}$	pr. $\vee$, 10
12.	$I \vDash C_1 \setminus \{l\} \vee C_2 \setminus \{\neg l\}$	us. $\vee$, 6-11
13.	$I \vDash C_1 \setminus \{l\} \cup C_2 \setminus \{\neg l\}$ semantics $\vee$	def. /
14.	$I \vDash R$	def. R
15.	$I \vDash \bot$	contradiction, 2, 14

See also https://math.stackexchange.com/questions/1482745/binary-resolution-rule-proof ?

Block 3

3.) Verify that the following program doubles the value of x, i.e., that x contains two times its initial value when the program terminates. For which inputs does it terminate? Choose appropriate pre- and postconditions and show that the assertion is totally correct.
Hint: Use $y = 2x_0 + x$ as a starting point for the invariant, where x_0 denotes the initial value of x. You may have to extend the formula to prove termination.

Remember the annotation rule
while e do $\cdots$ od $\mapsto \{ Inv \}$while e do $\{ Inv \wedge e \wedge t = t_0 \} \cdots \{ Inv \wedge (e \Rightarrow 0 \leq t < t_0) \}od\{ Inv \wedge \neg e \}$

$y := 3x;$
while $2x \neq y$ do
    $x := x + 1;$
    $y := y + 1;$
od

Status: solved by student

For which inputs does the program terminate?

The program terminates for $x \geq 0$ only; it enters an infinite loop when $x < 0$.

Choose appropriate pre- and postconditions and show that the assertion is totally correct.

We choose { F1 : $x \geq 0 \wedge x = x_0$ } as precondition and { F2 : $x = 2x_0$ } as postcondition; we show that { F1 } p { F2 } is totally correct by giving an annotation calculus proof:

{ $F_1 : x \geq 0 \wedge x = x_0$ }
{ $F_9 : Inv[y/3x]$ }
y := 3x;
{ $F_3 : Inv$ }
while $2x \neq y$ do
    { $F_4 : Inv \wedge 2x \neq y \wedge t = t_0$ }
    { $F_8 : (Inv \wedge ((2x \neq y) \Rightarrow 0 \leq t < t_0))[y/y+1][x/x+1]$ }
    x := x + 1;
    { $F_7 : (Inv \wedge ((2x \neq y) \Rightarrow 0 \leq t < t_0))[y/y+1]$ }
    y := y + 1
    { $F_5 : Inv \wedge ((2x \neq y) \Rightarrow 0 \leq t < t_0)$ }
od
{ $F_6 : Inv \wedge \neg(2x \neq y)$ }
{ $F_2 : x = 2x_0$ }

where Inv = ($y = 2x_0+x$) (hint from specification) and t = y-2x (since the loop condition is $2x \neq y$ and y is bigger when the loop condition holds, we know that the term y-2x gets smaller in every loop iteration until it reaches 0).

It remains to prove the following implications:

(1) $F_1 \Rightarrow F_9 :: (x \geq 0 \wedge x = x_0) \Rightarrow 3x = 2x_0+x$

is true since $x = x_0$ is true (premise), hence $3x = 2x + x = 3x$ is also true.

(2) $F_4 \Rightarrow F_8 :: (y = 2x_0+x \land 2x \neq y \land y-2x = t_0) \Rightarrow (y+1 = 2x_0+(x+1) \land (2(x+1) \neq (y+1) \Rightarrow 0 \leq$
$(y+1)-2(x+1) < t_0))$

We show that this implication is true by showing that the LHS implies both conjuncts of the RHS:

(a) $(y = 2x_0+x \land 2x \neq y \land y-2x = t_0) \Rightarrow y+1 = 2x_0+(x+1)$

is true since $y = 2x_0+x$ is true (premise), hence $y+1 = 2x_0+x+1$ is true.

(b) $(y = 2x_0+x \land 2x \neq y \land y-2x = t_0) \Rightarrow (2(x+1) \neq (y+1) \Rightarrow 0 \leq (y+1)-2(x+1) < t_0)$
is equivalent to:
$(y = 2x_0+x \land 2x \neq y \land y-2x = t_0 \land 2(x+1) \neq (y+1)) \Rightarrow 0 \leq (y+1)-2(x+1) < t_0$

Since $y-2x = t_0$ is true (premise), we can conclude that $(y+1)-2(x+1) = y-2x-1 < t_0$ is true as well. However, with the given premises, we cannot conclude that $0 \leq (y+1)-2(x+1)$ is true; we have to adapt Inv.

We adapt Inv and add $2x \leq y$ as conjunct: $Inv' = y = 2x_0 + x \land 2x \leq y$. We reprove already proved implications:

(1) $F_1 \Rightarrow F_9 :: (x \geq 0 \land x = x_0) \Rightarrow 3x = 2x_0+x \land 2x \leq 3x$

(a) $(x \geq 0 \land x = x_0) \Rightarrow 3x = 2x_0+x$

Since $x = x_0$, $3x = 2x_0+x = 2x+x = 3x$ is true as well.

(b) $(x \geq 0 \land x = x_0) \Rightarrow 2x \leq 3x$

Since $x \geq 0$, $2x \leq 3x$ is always true (only negative numbers would make it false).

(2) $F_4 \Rightarrow F_8 :: (y = 2x_0+x \land 2x \leq y \land 2x \neq y \land y-2x = t_0) \Rightarrow (y+1 = 2x_0+(x+1) \land 2(x+1) \leq y+1$
$\land (2(x+1) \neq (y+1) \Rightarrow 0 \leq (y+1)-2(x+1) < t_0))$

(a) $(y = 2x_0+x \land 2x \leq y \land 2x \neq y \land y-2x = t_0) \Rightarrow y+1 = 2x_0+(x+1)$

see above.

(b) $(y = 2x_0+x \land 2x \leq y \land 2x \neq y \land y-2x = t_0) \Rightarrow 2(x+1) \leq y+1$

$2(x+1) \leq y+1$ is equivalent to $2x+1 \leq y$ as well as $2x < y$; this inequality can be derived from the premises $2x \leq y \land 2x \neq y$.

(c) $(y = 2x_0+x \land 2x \leq y \land 2x \neq y \land y-2x = t_0) \Rightarrow (2(x+1) \neq (y+1) \Rightarrow 0 \leq (y+1)-2(x+1) < t_0)$ is equivalent to:
$(y = 2x_0+x \land 2x \leq y \land 2x \neq y \land y-2x = t_0 \land 2(x+1) \neq (y+1)) \Rightarrow 0 \leq (y+1)-2(x+1) < t_0$

$0 \leq (y+1)-2(x+1)$ is equivalent to $2x+1 \leq y$. We know that $2x \leq y$ and $2x \neq y$ are true (premises), hence $2x < y$ is true and therefore $2x+1 \leq y$.

(3) $F_6 \Rightarrow F_2 :: (y = 2x_0+x \land 2x \leq y \land 2x = y) \Rightarrow x = 2x_0$

Since $2x = y$ is true, we know that $2x = 2x_0+x$ is true as well, which let us conclude that $x = 2x_0$.

We have shown that all implications resulting from the annotation calculus proof are true, hence we can conclude that the assertion $\{ F_1 \}\ p\ \{ F_2 \}$ is totally correct.

Solution from class exercises

Exercise 8

Verify that the following program doubles the value of x. For which inputs does it terminate? Choose appropriate pre- and postconditions and show that the assertion is totally correct. Use $y = 2x_0 + x$ as a starting point for the invariant, where x_0 denotes the initial value of x.

$y := 3x;$
while $2x \neq y$ **do**
  $x := x + 1;$
  $y := y + 1;$
od

Solution

$\{\, 1\colon x = x_0 \wedge x \geq 0 \,\}$
$\{\, 7\colon Inv[y/3x] \,\}$ as↑
$y := 3x;$
$\{\, Inv\colon y = 2x_0 + x \wedge y \geq 2x \,\}$ wht″
while $2x \neq y$ **do**
  $\{\, 4\colon Inv \wedge 2x \neq y \wedge t = t_0 \,\}$ wht″
  $\{\, 9\colon (Inv \wedge 0 \leq t < t_0)[y/y+1][x/x+1] \,\}$ as↑
  $x := x + 1;$
  $\{\, 8\colon (Inv \wedge 0 \leq t < t_0)[y/y+1] \,\}$ as↑
  $y := y + 1;$
  $\{\, 5\colon Inv \wedge 0 \leq t < t_0 \,\}$ wht″
od
$\{\, 6\colon Inv \wedge 2x = y \,\}$ wht″
$\{\, 2\colon x = 2x_0 \,\}$

We choose $y = 2x_0 + x \wedge 2x \leq y$ as invariant Inv and $y - 2x$ as variant t. It remains to

show that the three implications $1 \Rightarrow 7$, $4 \Rightarrow 9$, and $6 \Rightarrow 2$ are valid.

$$1 \Rightarrow 7$$
$$x = x_0 \wedge x \geq 0 \Rightarrow Inv[y/3x]$$
$$x = x_0 \wedge x \geq 0 \Rightarrow 3x = 2x_0 + x \wedge 2x \leq 3x$$

$3x = 2x_0 + x$ holds because of the first premise and $2x \leq 3x$ because of the second one.

$$4 \Rightarrow 9$$
$$Inv \wedge 2x \neq y \wedge t = t_0 \Rightarrow (Inv \wedge 0 \leq t < t_0)[y/y+1][x/x+1]$$
$$Inv \wedge 2x \neq y \wedge t = t_0 \Rightarrow (y = 2x_0 + x \wedge 2x \leq y \wedge 0 \leq y - 2x < t_0)[y/y+1][x/x+1]$$
$$Inv \wedge 2x \neq y \wedge t = t_0 \Rightarrow y + 1 = 2x_0 + x + 1 \wedge 2(x+1) \leq y + 1 \wedge 0 \leq y + 1 - 2(x+1) < t_0$$
$$Inv \wedge 2x \neq y \wedge t = t_0 \Rightarrow y = 2x_0 + x \wedge 2x + 1 \leq y \wedge 0 \leq y - 2x - 1 < t_0$$

$y = 2x_0 + x$ is part of the invariant (first premise). $2x + 1 \leq y$ holds since $2x \leq y$ is part of the invariant (first premise) and $2x \neq y$ holds because of the second premise. $0 \leq y - 2x - 1$ is the same as $2x + 1 \leq y$, which we just showed to be true. $y - 2x - 1 < t_0$ is true since t_0 is the same as $y - 2x$ (third premise) and $t_0 - 1$ is obviously smaller than t_0.

$$6 \Rightarrow 2$$
$$Inv \wedge 2x = y \Rightarrow x = 2x_0$$

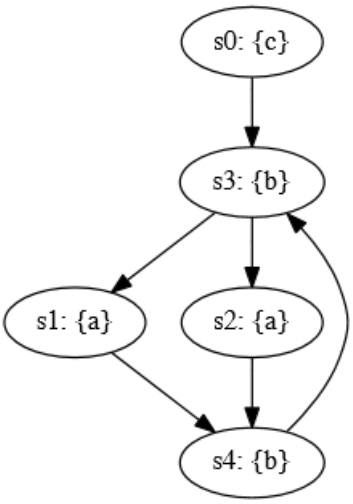
Solving the two equalities $y = 2x_0 + x$ (from Inv) and $2x = y$ for x we obtain the conclusion $x = 2x_0$.

Block 4

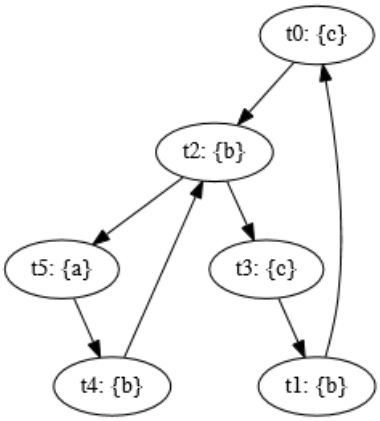
a) Status: solved by student

Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:

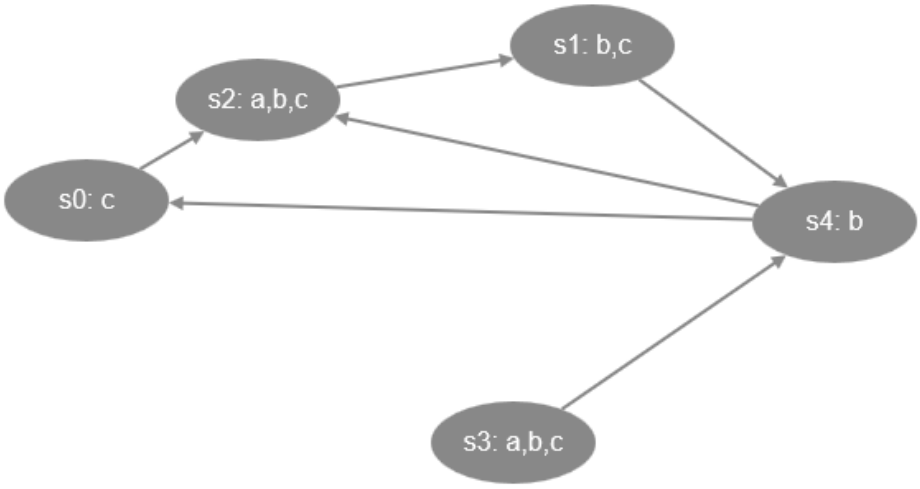


Kripke structure M_2:



H: {(s0, t0), (s3, t2), (s1, t5), (s2, t5), (s4, t4)}

b) Status: solved by student



phi	CTL	LTL	CTL*	States
((b & c) U a)		OK	OK	{s2, s3}
AX(a)	OK		OK	{s0}

EG(b)	OK		OK	{s1, s2, s3, s4}
EF(a)	OK		OK	{s0, s1, s2, s3, s4}
E(a U a)	OK		OK	{s2, s3}

c) Let K = (S, T, L) be a Kripke structure and let p, q be atomic propositions. Give an algorithm that computes the set of all states s ∈ S that satisfy A[p U q].

Definitions:

- M,s |= Ap iff for all paths π starting at state s, M,π |= p
- M,PI |= p U q iff there exists a k >= 0 such that M,π^k |= q and for all 0 <= j < k, M,π^j |= p

Solution: status student

```
checkStates()
{
	for all states
	{
		if labeled q
			add to list
	}
	do {
		added = 0;
		for all states not in list
		{
			if labeled p and all successors are in list
			{
				add to list
				added = 1
			}
		}
	} while (added == 1);
}
```

Alternative solution: **status student** Implement DFS with "visited" set of nodes and recall that algorithm for every node in Kripke structure.

Alternative Solution by student:

No algorithm is **construction** with a programming language but described:

Step 1 : Mark all states where q is satisfied
Step 2 : Mark all states where p is satisfied and when there exists an already marked successor
Step 3 : Iterate over all marked states and check whether all successors are marked.
Step 4 : Repeat step 3 until there are no changes anymore
Step 5 : Return all marked states

Exam 27.01.2017 (FMI.171.pdf)

Block 1

Status: solved by student

Let G be an arbitrary instance of the 3-COLORABILITY problem. We construct an instance (G_1, G_2) of PROB as follows: G_1 = G and G_2 is a graph with 3 vertices that are interconnected (hence it is trivially 3-colorable).

It remains to show that this is a valid reduction, i.e. G is a positive instance of 3-COLORABILITY iff (G_1, G_2) is a positive instance of PROB:

=>: Assume G is a positive instance of 3-COLORABILITY. Then, by the simple construction of our reduction, we know that G_1 must be a positive instance of the 3-COLORABILITY problem, i.e. there exists a 3-coloring for it. Due to the fact that the answer of the PROB instance (G_1, G_2) is a disjunction of the propositions “G_1 is 3-colorable” and “G_2 not 3-colorable”, and that we know that G_1 is indeed 3-colorable, we know that the left disjunct makes the whole disjunction true, hence (G_1, G_2) is a positive instance of PROB.

<=: Assume G is a negative instance of 3-COLORABILITY. Then, by the simple construction of our reduction, we know that G_1 is a negative instance of the 3-COLORABILITY problem, i.e. there exists no 3-coloring for it. Therefore, the left disjunct of the PROB question is false. However, by construction of G_2, we know that it has a (trivial) 3-coloring, therefore the right disjunct is false as well. Due to the fact that both disjuncts are false, we know that the whole disjunction if false. Hence, (G_1, G_2) must be a negative instance of PROB.

Alternative <=:
Assume (G_1, G_2) is a positive instance of PROB. Due to our construction we have that G2 can only be 3-colorable, therefore G1 must be 3-colorable. Since we set G=G1 it follows that G is 3-colorable. Hence, G is a positive instance of 3-COLORABILITY.

Block 2

a) **Status: solved by student**
Semantic Proof / Proof by Contradiction:

Suppose, there exists a T-interpretation such that I |≠ φ.

1. I |≠ φ	assumption
2. I |= f(f(f(a))) ≈ f(f(b))	1, semantics of →
3. I |≠ a ≈ b	1, semantics of →
4. I |= f(f(f(a))) ≈ f(b)	2, Axiom 2 for f(f(b))
5. I |= f(f(a)) ≈ f(b)	4, Axiom 2 for f(f(f(a)))
6. I |= f(a) ≈ f(b)	5, Axiom 2 for (f(f(a))
7. I |= a ≈ b	6, Axiom 1 for ≈
8. I |= ⊥	3, 7, contradiction

The assumption is false, therefore φ is T-valid.

I'm not sure if the usage of Axiom 2 in the proof above is permissible -- I'd rather construct the proof as follows:

1. I ⊭ f(f(f(a))) ≈ f(f(b)) → a ≈ b	assumption (towards contradiction)
2. I ⊨ f(f(f(a))) ≈ f(f(b))	semantics →, 1
3. I ⊭ a ≈ b	semantics →, 1
4. I ⊨ f(f(b)) ≈ f(f(f(b)))	Axiom 2 -- x = f(b)
5. I ⊨ f(f(f(a))) ≈ f(f(f(b)))	transitivity, 2+4
6. I ⊨ f(f(a)) ≈ f(f(b))	Axiom 1 -- x=f(f(a)), y=f(f(b)) + us →, 5
7. I ⊨ f(a) ≈ f(b)	Axiom 1 -- x=f(a), y=f(b) + us →, 6
8. I ⊨ a ≈ b	Axiom 1 -- x=a, y=b + us →, 7
9. I ⊨ ⊥	contradiction 3 + 8

b)

(b) Let ψ be a propositional formula in conjunctive normal form, and let C be a non-tautological clause containing a literal p such that the literal $\neg p$ does not occur in ψ. Give a detailed proof of the following statement:

$\psi \wedge C$ is satisfiable if and only if ψ is satisfiable

Status: solved by student

We have to show Ψ ∧ C is sat. ⇔ Ψ is sat.

=> : Assume Ψ ∧ C is satisfiable, then by semantics of conjunction it trivially follows that Ψ must be satisfiable as well.

<=: Assume Ψ is satisfiable; it remains to show that C is satisfiable as well. Since C contains a pure literal p, we can safely assign p = T without affecting the satisfiability of Ψ (the assignment p = T could make Ψ unsatisfiable iff Ψ contains a clause which relies on the literal ¬p to become true; however, this can be precluded by our initial assumption that Ψ does not contain ¬p at all). Since C is a disjunction, it suffices to make one literal true to make C true. Hence, C is satisfiable and therefore Ψ ∧ C as well.

Cx x Block 3

(the solution for this can be found under the January 2016 exam)
Status: no solution, but some ideas for a discussion

a) We deduce (as') from (as) and vice versa using the Hoare calculus. When we can show that both rules can be deduced from each other, at least one is redundant:

   (as') => (as): We apply the logical consequence rule on (as') and get (as) and two implications we need to prove:

$$\frac{G[v/e] \Rightarrow F \quad \{F\} v := e \{\exists v'(F[v/v'] \wedge v = e[v/v'])\} \quad \exists v'(F[v/v'] \wedge v = e[v/v']) \Rightarrow G}{\{G[v/e]\}\ v := e\ \{G\}}\ lc$$

   We choose F = G[v/e] which makes the first implication trivially true. It remains to that second implication:

   ∃ v'(F[v/v'] ∧ v = e[v/v']) ⇒ G is true which is equivalent to ∃ v'(G[v/e][v/v'] ∧ v = e[v/v']) ⇒ G because of our choice for F. G[v/e][v/v'] is equivalent to G[v/e[v/v']], which is again equivalent to G[v/v] under the condition v = e[v/v']. We get G ∧ ∃ v'(v = e[v/v']) ⇒ G which is obviously true (G occurs as conjunct in the premise).

   (as) => (as'): We apply the logical consequence rule on (as) and get (as') and two implications we need to prove:

$$\frac{F \Rightarrow G[v/e] \quad \{G[v/e]\}\ v := e\ \{G\} \quad G \Rightarrow \exists v'(F[v/v'] \wedge v = e[v/v'])}{\{F\}\ v := e\ \{\exists v'(F[v/v'] \wedge v = e[v/v'])\}}\ lc$$

   We choose G = ∃ v'(F[v/v'] ∧ v = e[v/v']) w-hich makes the second implication

trivially true. It remains to show the first implication; we get F ⇒ (∃ v'(F[v/v'] ∧ v = e[v/v']))[v/e] which corresponds to F ⇒ ∃ v'(F[v/v'][v/e] ∧ e = e[v/v'][v/e]) after doing the replacements on the RHS of the implication. Moving the second replacements into the first, we get F ⇒ ∃ v'(F[v/v'[v/e]] ∧ e = e[v/v'[v/e]]). Since e is equal to e after doing the replacements, and e was chosen arbitrarily, we may assume that the replacements do not have any effect at all. Under this assumption, we get F ⇒ F ∧ ∃ v'(e = e[v/v'[v/e]])

b) Idea: start with (as'') and try to deduce axioms / valid implications only (maybe making use of the axioms as, as')? (actually the other way around: start with admissible axioms and deduce (as''))

Other students approach:

Considering v does not occur in F and e, (as') can be simplified to (as'') as follows: Since v does not occur in F, F[v/v'] is equal to F and v:=e[v/v'] is equal to v:=e for any v'.

Therefore we are left with {F} v:=e {∃ v'(F ∧ v:=e)} and moreover {F} v:=e {F ∧ v:=e} which is (as'').

c) We construct a counterexample that shows the rule cannot be sound. Let F, and v := e be as follows:

{ F: x = 1 } x := x+1 { G: F[v/e]: x+1 = 1 }

x+1 = 1 is equivalent to x = 0 -- we've found one example which shows that, when starting in the chosen F-state and executing the chosen program, we do not arrive at a (expected) G-state. Hence, the (xx) rule cannot be sound.

Alternative:

Same counter example as above: {x=1}x:=x+1{x=0} this is obviously false, but now I think we need to show that it is derivable and thus we derive something which is not true (which contradicts the soundness)

$$\dfrac{x \Rightarrow F \qquad \dfrac{\{F\}\ x:=x+1\{G[x/x+1]\}}{\{F\}\ x:=x+1\{G[x/x+1]\}}\ (xx) \qquad G \Rightarrow x=0}{\{x=1\}x:=x+1\{x=0\}}\ (lc)$$

All the implications are valid or admissible under the given rules but the derived counter example is nevertheless false.

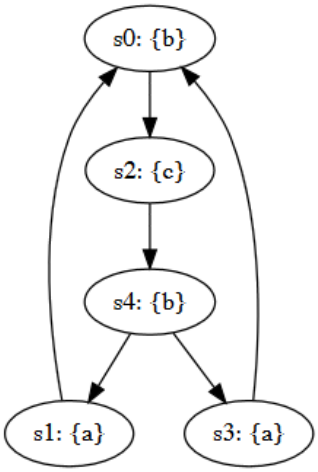
d) ???
Take a program {x:=0} x:=x+1 {x:=1}. This program can be verified using (as) but not using (as'') as "v occurs in F and in e". Hence the Hoare calculus is not complete with (as'') alone.

Block 4

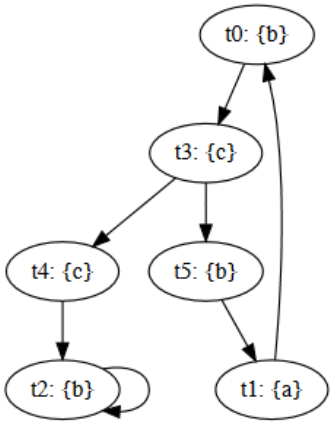
a) **Status: solved by student**

Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:

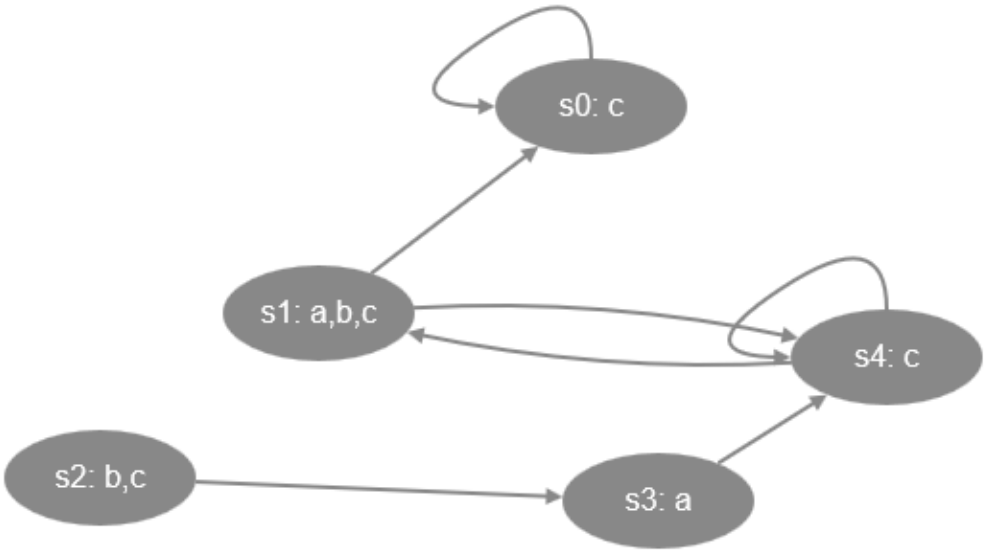


Kripke structure M_2:



H: {(s0, t0), (s2, t3), (s4, t5), (s1, t1), (s3, t1)}

b) **Status: solved by student**



phi	CTL	LTL	CTL*	States
X(a)		OK	OK	{s2}
EX(a)	OK		OK	{s2, s4}
(b & c) U c		OK	OK	{s0, s1, s2, s4}
EF(a)	OK		OK	{s1, s2, s3, s4}

c) **status: solved by student**

I didn't do the full proof, but this is the idea:

1. Show that ACTL is sublogic of ACTL*
    a. Show that CTL is sublogic of CTL*
    b. Say that both ACTL and ACTL* have only universal path quantifiers
2. Show that M' simulates M
    a. Provide a reasoning for simulation and show it makes sense
3. Use theorem from the lecture which is provided as hint
4. Conclude: Since it holds for ACTL*, then it holds for ACTL as well since ACTL is a sublogic - proved in (1).

Solution 2: solved by student: I hope this is correct.

1. Proof that ACTL is sublogic of ACTL*: I'm not quite sure if this is required, as it has been defined in the lecture that it is.
2. Proof that a simulation can be constructed

a. **Definition** H <= SxS' is a simulation relation between M and M' iff for every (s,s') ∈ H:
    i. (S1) s and s' satisfy the same propositions
    ii. (S2) for every successor t such that (s,t) ∈ R there is a successor t' such that (s',t') ∈ R' and (t,t') ∈ H
b. **Definition** H <= SxS' iff
    i. H is a simulation relation and
    ii. (S3) for every initial state s e S0, there is a corresponding s' ∈ S0' such that (s,s') ∈ H
c. We can construct such an H = { (s,s') | s ∈ S and s' ∈ S' and L(s) = L(s') } and then removing all (s,s') where (s,t) ∈ R but not (s',t') ∈ R' , i.e. adding an entry for all states with the same labels and removing those entries that do not lead to a valid successor state. It remains to be shown that H satisfies S1, S2 and S3
    i. (S1): This is trivially true due to the construction of H
    ii. (S2): By the definition of R' (s,t) ∈ R → (s',t') ∈ R' is fulfilled. Also by Construction of H (t,t') ∈ H is also fulfilled
    iii. (S3) Also trivially fulfilled by H, as every initial state of M' corresponds to an initial state of M

3. As we know that M' simulates M, and we know that ACTL is a sublogic of ACTL*, If M' |= phi, then M |= phi must trivially hold

Exam 09.12.2016 (FMI.166.pdf)

https://www.logic.at/lvas/fminf/angaben/fmi166.pdf

Block 1

Status : solved by student

We construct the following semi-decision procedure:

```
Bool PROB (String Π) {
      while true {
            String I = GenerateNextInput();
            String O = Π(I);
            If IsValidString(O) {
                  return true;
            }
      }
}
```

whereas `GenerateNextInput()` generates the next possible input string from the alphabet Σ = {0, 1} and `IsValidString()` checks if a given string is valid, i.e. that it can be constructed from Σ.

It remains to show that PROB is indeed a semi-decision procedure, i.e. that it terminates with a positive answer on yes instances, and either returns a negative answer on no instances or does not terminate at all.

Case (1):
Assume that (Π) is a positive instance of PROB, i.e. that there exist strings I_1 and I_2 s.t. Π outputs I_2 on I_1. Then, `GenerateNextInput()` will at some point return a string which causes Π to return a valid output string (remember, Π always terminates). This will cause `IsValidString()` to return true (showing that there exists a string I_2), and therefore terminating the semi-decision procedure with a positive answer.

Case (2):
Assume that (Π) is a negative instance of PROB, i.e. that there do not exist strings I_1 and I_2 s.t. Π outputs I_2 on I_1. Then, `GenerateNextInput()` will never return a valid output string, causing the outer infinite loop to run forever and PROB to never terminate.

Block 1 (Cantor's enumeration principle)

I think that the solution above is incorrect. It does not check whether the two strings are equal, it just checks whether the output string can be constructed from the alphabet Σ = {0,1} - which it always does by our definition of "strings".

Similar to the slides for REACHABLE-CODE, we use Cantor' enumeration principle to explore the strings - we cannot start by some input instance I_1 and try all possible I_2 (or vice versa), we have to explore them "simultaneously".

When a single string enumeration looks like S_1, Cantor's enumeration of two strings could look like S: (see diagram)
S_1={0, 1, 00, 01, 10, 11, 000, …}
$S=S_1 \times S_1$={(0,0), (0,1), (1,0), (00, 0), (1,1), (0,00), (0,01), (1,00), (00,1), (01,0), ...}

Then we construct a decision procedure that takes a program Π as input:

```
Π_D(Π):
  for each (I_1,I_2) in S:
    if Π(I_1)==I_2:
      return true
```

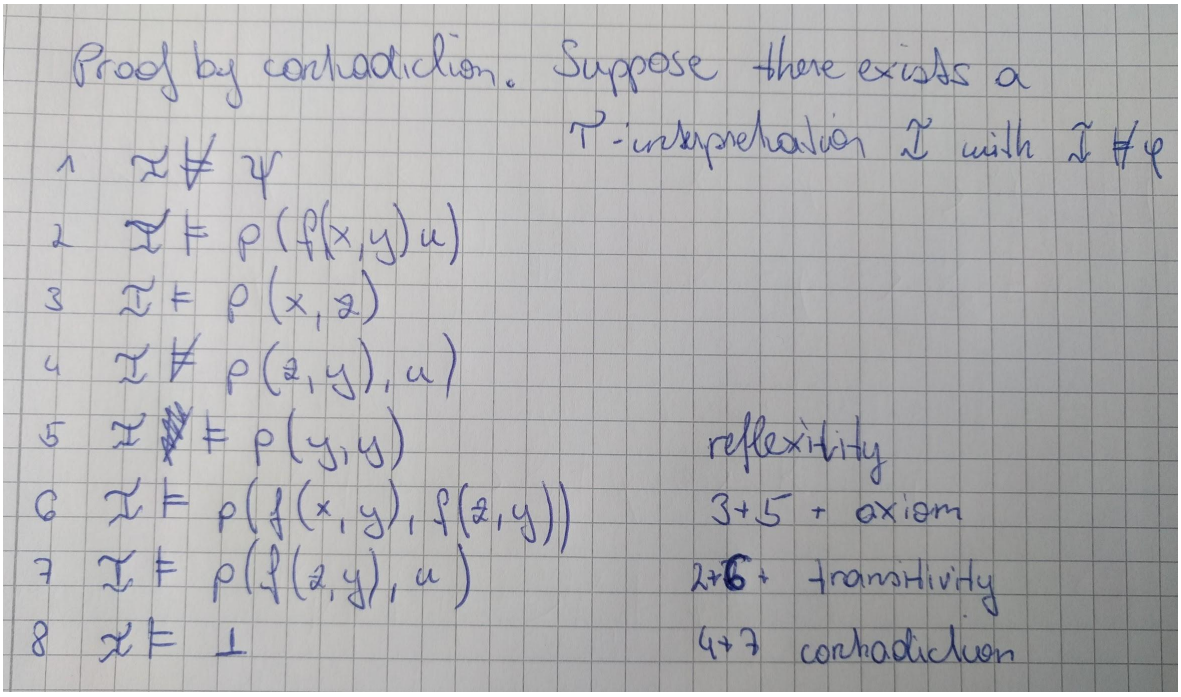
Case 1: Assuming Π is a positive instance of PROB, then there exists a string pair $(I_1,I_2)\in S$ for which $\Pi(I_1)=I_2$. Because S is countably infinite, this pair will be found in a finite amount of time. Π_D will terminate and return true.

Case 2: Assuming Π is a negative instance of PROB, Π_D will continue to iterate over S and never terminate, which is correct behavior for a semi-decision procedure.

Block 2

a) Status student

I don't know if it's correct, but that would be my solution.

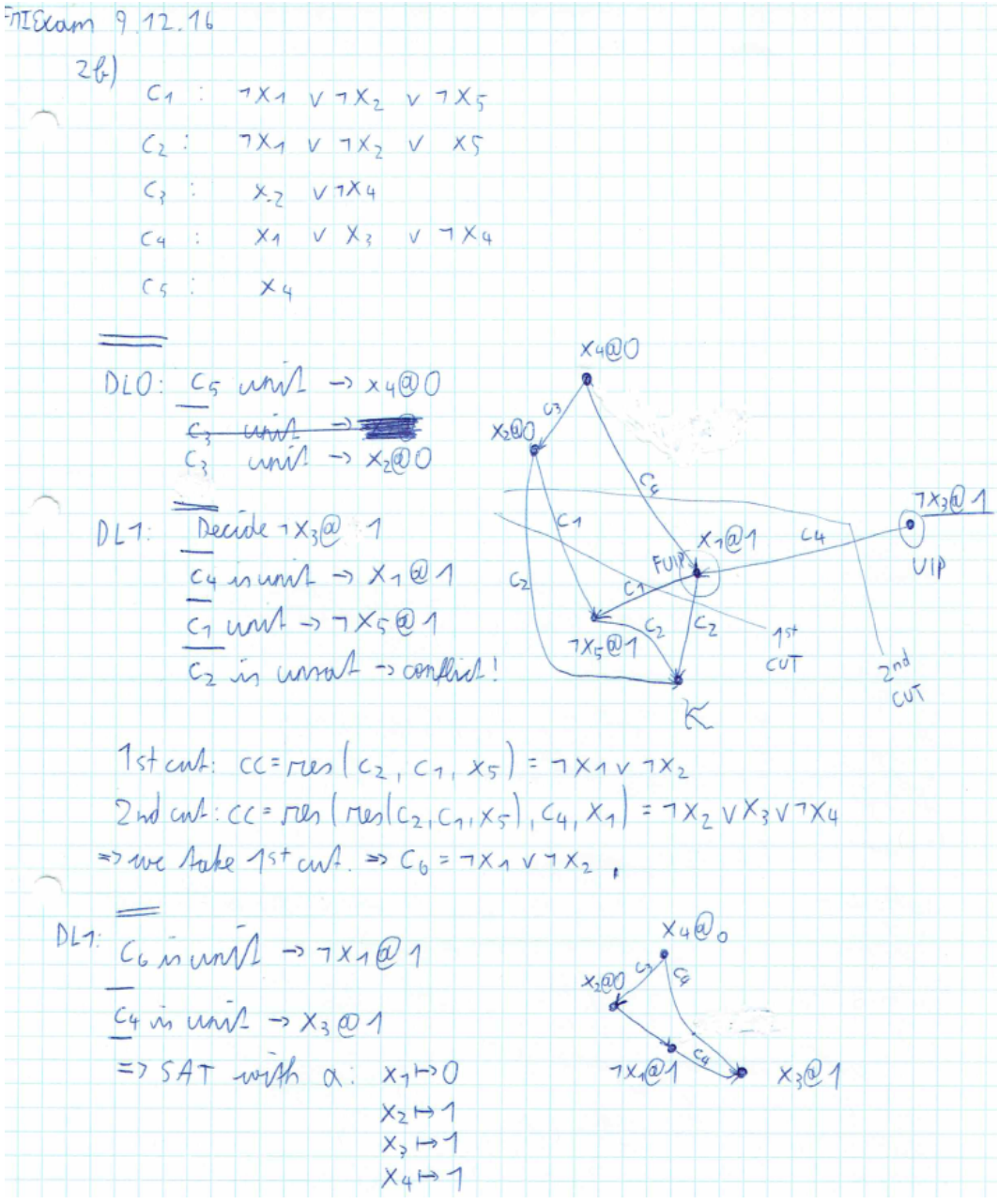


Before step 7 you have to use **symmetry on 6 -> I entails p(f(z,y), f(x,y)** (see comments)

Alternative Solution: Status student (different approach, please check for correctness)

b)	I ⊨ psi	assumption
c)	I ⊨ p(f(x,y), u)	1., semantics of ∧ and impl
d)	I ⊨ p(x, z)	1., semantics of ∧ and impl
e)	I ⊭ p(f(z,y), u)	1., semantics of impl
f)	I ⊭ p(f(z,y), f(x,y)) f(z,y), u)	2,4 symmetry + trans (i.e. p(u, f(x,y)) ∧ p(-> p(f(z,y), f(x,y))
g)	I ⊨ p(z, x) ∧ p(y, z)	5 axiom modus tollens
h)	I ⊨ p(x, z)	6 semantics of ∧, and symmetry
i)	I ⊨ falsum	3,7 contradiction

j) Status: solved by student



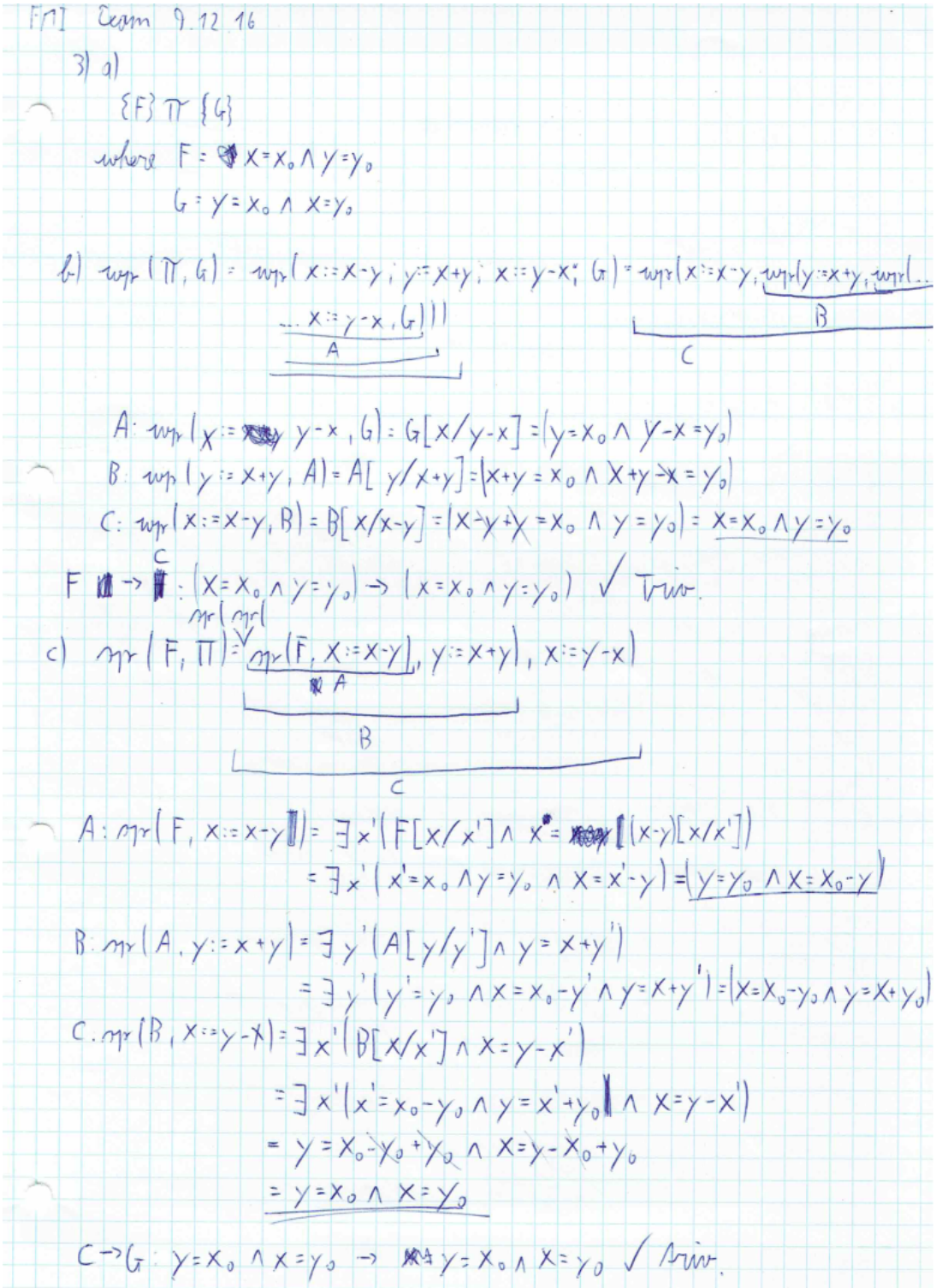
The formula is satisfiable.

- Example interpretation which evaluates the formula to true:
  x1 = 0, x2 = 1, x3 = 1, x4 = 1, x5 = 0
- Example interpretation which evaluates the formula to false:
  x1 = 1, x2 = 1, x3 = 0, x4 = 1, x5 = 1

The empty clause CANNOT be derived from the given CNF as this would mean it is unsatisfiable (has been mentioned in the consolidation lectures).

Block 3

Status: Student



Typeset solution:

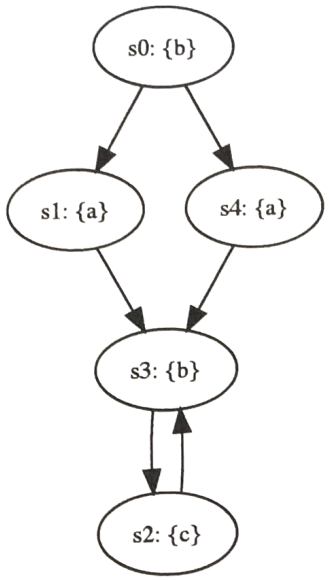
Π : x := x-y; y := x+y; x := y-x

a) { F } Π { G } where F: $x = x_0 \wedge y = y_0$ and G: $x = y_0 \wedge y = x_0$

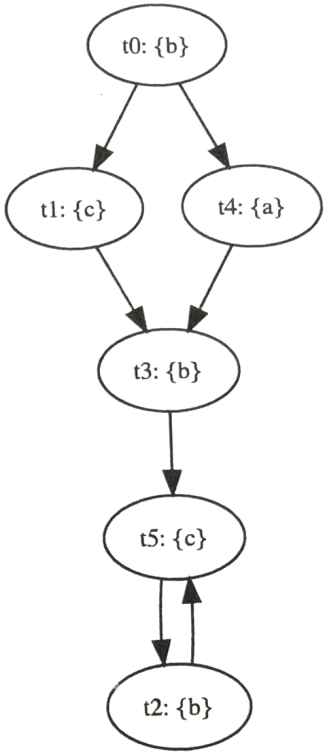
b) wp(x := x-y; y := x+y; x := y-x, $x = y_0 \wedge y = x_0$) =
= wp(x := x-y, wp(y := x+y; x := y-x, $x = y_0 \wedge y = x_0$)) =
= wp(x := x-y, wp(y := x+y, wp(x := y-x, $x = y_0 \wedge y = x_0$))) =
= wp(x :=

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



Kripke structure M_2:



(4 points)

x-y, wp(y := x+y,)) =
= wp(x := x-y, x+y-x = $y_0 \wedge$ x+y = x_0) =
= x+y-x = $y_0 \wedge$ x-y+y = x_0) =
= **$y = y_0 \wedge x = x_0$**

c) sp($x = x_0 \wedge y = y_0$, x := x-y; y := x+y; x := y-x) =
= sp(sp($x = x_0 \wedge y = y_0$, x := x-y), y := x+y; x := y-x)) =

= sp(∃ x'(x' = x_0 ∧ y = y_0 ∧ x = x'-y), y := x+y; x := y-x) =
= sp(x+y = x_0 ∧ y = y_0 ∧ ∃ x'(x' = x+y), y := x+y; x := y-x) =
= sp(sp(x+y = x_0 ∧ y = y_0, y := x+y), x := y-x)) =
= sp(∃ y'(x+y' = x_0 ∧ y' = y_0 ∧ y = x+y'), x := y-x) =
= sp(x+y-x = x_0 ∧ y-x = y_0 ∧ ∃ y'(y' = y-x), x := y-x) =
= sp(y = x_0 ∧ y-x = y_0, x := y-x) =
= ∃ x'(y = x_0 ∧ y-x' = y_0 ∧ x = y-x') =
= y = x_0 ∧ y-(y-x) = y_0 ∧ ∃ x'(x'=y-x) =
= **y = x_0 ∧ x = y_0**

Block 4

a) Status student

Let φbe an ACTL specification. Then
If $K_1 \preccurlyeq K_2$ then $K_2 \models \varphi \Rightarrow K_1 \models \varphi$
If $K_2 \preccurlyeq K_3$ then $K_3 \models \varphi \Rightarrow K_2 \models \varphi$
Hence
$K_3 \models \varphi \Rightarrow K_2 \models \varphi \Rightarrow K_1 \models \varphi$
As a logical consequence $K_3 \models \varphi \Rightarrow K_1 \models \varphi$
Therefore $K_1 \preccurlyeq K_3$

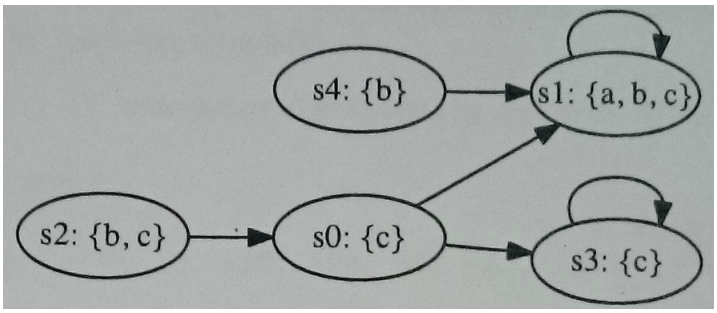
Alternative solution (not complete, from the exam)
1 -deleted?

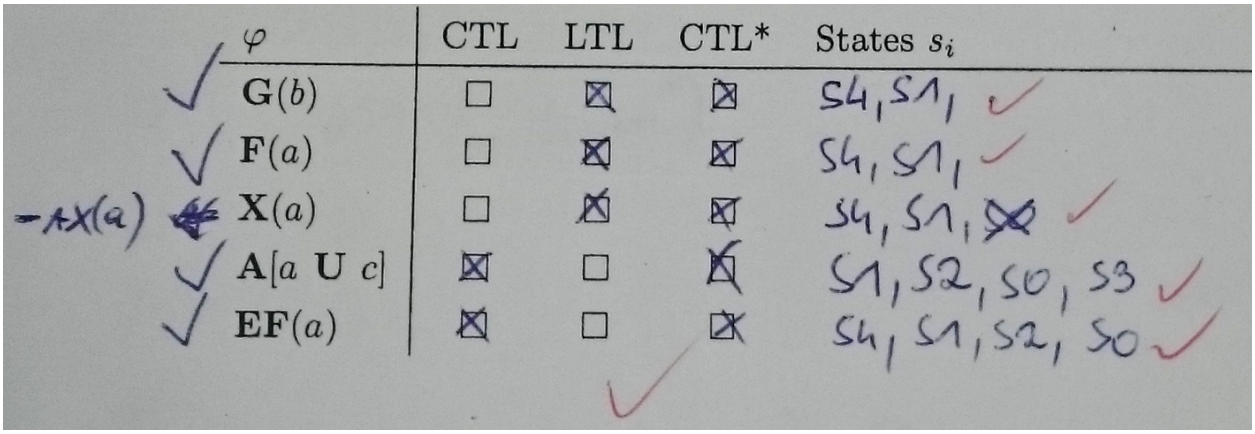
Alternative solution: Structural induction

Base Case: consider all initial (starting) states
Induction step: consider two cases how relations can come in place

- from one of the initial states -> prove accordingly
- from one of the successor states -> prove accordingly

b) **approved by professor**





φ	CTL	LTL	CTL*	States s_i
$\mathbf{G}(b)$	☐	☒	☒	s_4, s_1
$\mathbf{F}(a)$	☐	☒	☒	s_4, s_1
$\mathbf{X}(a)$	☐	☒	☒	s_4, s_1
$\mathbf{A}[a\ \mathbf{U}\ c]$	☒	☐	☒	s_1, s_2, s_0, s_3
$\mathbf{EF}(a)$	☒	☐	☒	s_4, s_1, s_2, s_0

c) **status student**

```
int sum=0, isinitialized=0;
for(int i=0; i<N; i++){
      if(nondet_bool()){
            sum+=values[i];
            isinitialized=1;
      }
}
//if there is a solution with sum 0 -> report
assert(!(sum==0 && isinitialized==1));
```

Exam 21.10.2016 (FMI.165.pdf)

https://www.logic.at/lvas/fminf/angaben/fmi165.pdf

Block 1 - fmi165

LOOPS-HALTS

INSTANCE: A tuple (Π1; Π2; I1; I2), where I1; I2 are strings and Π1;Π2 are programs that take a string as input.

QUESTION: Is it true that Π1 does not halt on I1 and Π2 halts on I2?
By providing a reduction from an undecidable problem, prove that LOOPS-HALTS is undecidable. Argue formally that your reduction is correct.

Solution (by student with help of tutor, but not approved!):

Let $x = (\Pi\ ,\ I\ )$ be an arbitrary instance of HALTING, then we construct

$R(x) = (\Pi_1, \Pi_2, I_1, I_2)$ as follows:

We construct Π' as the transformation from x to R(x):

$String\ \Pi'(String\ I\ )$

$\Pi_2(I\ );$

$\Pi_1(I_1);$ // *has to be second one, otherwise* Π_2 *can never be reached*

}

$\Pi_1$with the body { while (true) do {}}

Π_2:= Π

Let I_1 be any string. E.g., let I_1 = “Hello World”

Let I_2:=I

We claim that: Π_1 does not halt on I_1 and Π_2 halts on I_2 ⇔ Π halts on I_1

- " ⇒ " Suppose Π_1 does not halt on I_1 and Π_2 halts on I_2: by our construction of Π, Π_1 is executed after Π_2, Π_2 has to halt on I_2⇒ Π halts on I , because Π_2= Π and I_2=I
- " ⇐ " Suppose Π halts on I: Π_2 also halts on I_2, since Π_2= Π and I_2=I . Since Π_2 halts, Π will continue with Π_1, which does not halt by construction.

Alternative solution
(student with no tutor/professor input)

Let $(\Pi'\ ,\ I'\ )$be an arbitrary instance of HALTING. Let $\Pi'\ = \Pi_1 = \Pi_2$ and Π_{LH}be the function LOOPS_HALTS. Let $I_1$and I_2be hardcoded in Π_{LH}. Let 0 == does_not_halt and 1 == halts

Bool Π_{LH}()
{
    If (0 ==$\Pi_1(I_1)$)

```
            If (1 == Π2(I2))
                    return true;
        return false;
}
```

Π_{LH} returns true $\Leftarrow\Rightarrow\Pi_1$does not halt and $\Pi_2$halts

$\Rightarrow$Suppose Π_{LH}returns true. Thus we have that $\Pi_1$does not halt on $I_1$and $\Pi_2$halts on I_2.

$\Leftarrow$ Suppose $\Pi_1$does not halt on $I_1$and $\Pi_2$halts on I_2. Due to the construction Π_{LH} also halts and returns true.

QED

Additional solution2: (student without tutor/prof)

Co-halting and halting are both undecidable:

Let $(\Pi_1, I1)$be an arbitrary instance of Co-HALTING.

Let $(\Pi_2, I2)$be a pos. instance of HALTING i.e. it halts on I2.

We set: Π =Π_1, I=I1

=> (Π , I) doesn't halt

=> Due to construction of Π follows, (Π_1, I1) doesn't halt.

=> (Π_1, Π_2, I1, I2) is a positive instance of loophalts.

<= (Π_1, Π_2, I1, I2) is a positive instance of loophalts:

=> (Π_1, I1) doesn't halt:

=> Due to construction of Π follows, (Π , I) doesn't halt.

=> (Π , I) doesn't halt

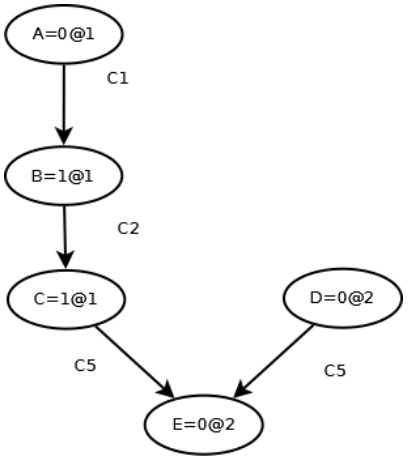
Block 2 - fmi165

status student

a)

Sorry for a messy solution, but should be a valid solution? Or at least better than nothing :)

The result is in each case satisfiable… comments / improvements are welcomed!



Alternative solution (?)

Alternative solution:
https://www.informatik-forum.at/forum/index.php?thread/116560-exam-21-10-2016-2a/

b) **status student**
From the test we know:

- ϕ is a set of clauses and clauses are sets of literals (additionally i define atoms to be literals without negation signs)
- C_1 ∈ ϕ and C_2 ∈ ϕ
- l ∈ C_1 and ¬l ∈ C_2
- ϕ' = (ϕ ∪ {R}) \ {C_1}

Assumption to make: resolvent R ⊂ C_1

Additional definitions:

- C_1' := C_1 \ {l}
- C_2' := C_2 \ {¬l}

From the available information I gather that the resolution looks like this:

$$\frac{C'_1 \cup \{l\}, C'_2 \cup \{\neg l\}}{R}$$

From this we can see that R = C_1' ∪ C_2'. Now we use the assumption that R is a strict subset of C_1 (resp C_1' ∪ {l}): C_1' ∪ C_2' ⊂ C_1' ∪ {l}
This is only possible if C_1' = C_2' => C_2' ⊆ C_1'

Now the question is: ϕ ⇔ ϕ' ?

From before we know that C_1 and C_2 differ only in the atom l. From this we can draw the following conclusions (with the assumption that C_1 and C_2 are the only clauses in ϕ since we don't know anything about the other clauses and their satisfiability doesn't matter):

1. If C_1 holds then C_2 either holds or not. From this we can conclude that
    a. If C_1 holds and C_2 holds ϕ clearly holds then also ϕ' holds, because some other atom in C_1 and C_2 other than l is true.
    b. If C_1 holds and C_2 does **not** hold then ϕ does not hold, then ϕ' does **not** hold either, because the truth value of C_1 does not matter since C_2 is already false and does not change in ϕ'.
2. If C_1 does **not** hold, C_2 definitely holds since they both contain l with flipped truth values. From this we can conclude that
    a. ϕ does **not** hold and ϕ' does **not** hold either, because all literals in C_1 need to be false in order to make C_1 false, and removing a literal does not change that.

Therefore I conclude that ϕ ⇔ ϕ'.
EDIT: I didn't show ϕ' => ϕ:

1. If C_1' holds, then extending the CNF (conjunction of disjunctions) clause by a literal does not change its truth value.
2. If C_1' does **not** hold, it doesn't matter, because then the implication looks like this: false => ϕ which is true by default.

Please correct me if I'm wrong :).

Block 3 - fmi165

a) **status student, please check**

{Inf & e} p {Inf & e} (Inf & e) => Inf
——————————————————————— lc
{Inf & e} p {Inf}
——————————————————————— wh
{Inf} while e do p od {Inf & !e}

To show: (Inf & e) => Inf
This is a tautology and therefore valid

b) **status student please check**

Take a program, {x >= 0} while x=1 do x:=x-1 od { x >= 0 }

1) Show, that the program is correct

$$\frac{\{x>= 0 \ \& \ x=1\} \Rightarrow \{x-1 >= 0\} \qquad \{x-1 >= 0\}\ x=x-1\ \{x >= 0\}}{\{x >= 0 \ \& \ x = 1\}\ x=x-1\ \{x >= 0\}}\ lc$$

$$\frac{\{x >= 0 \ \& \ x = 1\}\ x=x-1\ \{x >= 0\}}{\{x >= 0\}\ \text{while } x=1 \text{ do } x=x-1 \text{ od } \{x >= 0 \ \& \ x!=1\}}\ wh$$

We need to show that {x>= 0 & x=1} => {x-1 >= 0}. This is trivially true.

2) Show that the program is no longer correct if we attempt to prove it with the logical consequence and the modified while rule:

$$\frac{\{x>= 0\} \Rightarrow F \qquad \dfrac{\{F \ \& \ x = 1\}\ x=x-1\ \{F \ \& \ x = 1\}}{\{F\}\ \text{while } x=1 \text{ do } x=x-1 \text{ od } \{F \ \& \ x != 1\}}\ mw \qquad \{F \ \& \ x != 1\} \Rightarrow \{x >= 0 \ \& \ x!=1\}}{\{x >= 0\}\ \text{while } x=1 \text{ do } x=x-1 \text{ od } \{x >= 0 \ \& \ x!=1\}}\ lc$$

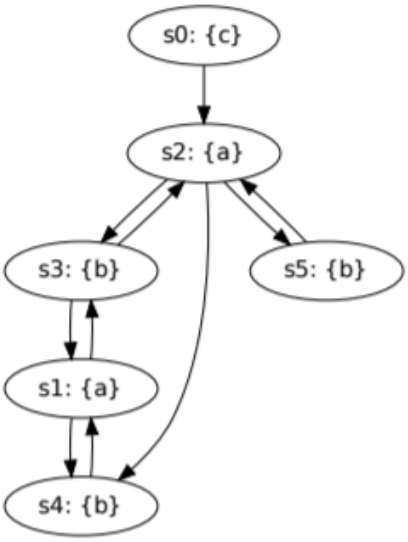
We need to show that it is impossible for F to exist such that it satisfies all Implications.

Block 4 - fmi165

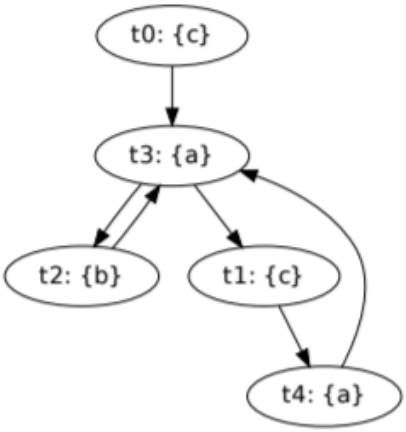
a) **Status student, Status approved by professor**

4.) (a) Provide a non-empty simulation relation H that witnesses $M_1 \leq M_2$, where M_1 and M_2 are shown below. The initial state of M_1 is s_0, the initial state of M_2 is t_0:

Kripke structure M_1:



Kripke structure M_2:

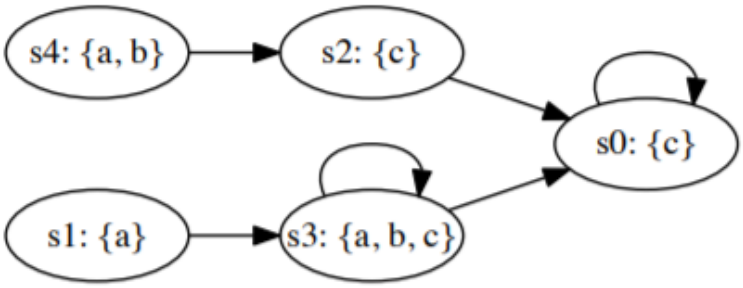


(4 points)

H={(s0, t0), (s2, t3), (s3, t2), (s1, t3), (s4, t2), (s5, t2)}

b) **status student, Status approved by professor**

(b) Consider the following Kripke structure M:



For each of the following formulae φ,

i. check the respective box if the formula is in CTL, LTL, and/or CTL*, and
ii. list the states s_i on which the formula φ holds; i.e. for which states s_i do we have $M, s_i \models \varphi$?

G(c)= s0, s2, s3 → LTL, CTL*
X(b) = s1 → LTL, CTL*
c U b = s3, s4 → LTL, CTL*
AF(a) = s1, s3, s4 → CTL, CTL*
EG (a)= s1, s3 → CTL, CTL*

c) see 17.10.2014 c)

Exam 01.07.2016 (FMI.164.pdf)

http://www.logic.at/lvas/fminf/angaben/fmi164.pdf

Block 1 - fmi164

Status: solved by student, approved by professor

Choose SAT as the NP-complete problem. To show: **SAT ≤ SAT-UNSAT**
Let φ be an arbitrary instance of SAT. φ is a propositional formula. We construct an instance of SAT-UNSAT (φ_1,φ_2) as follows: φ_1=φ and φ_2=true.

⇒: Let φ be a positive instance of SAT. By the construction from above, φ_1 is also satisfiable, therefore yielding a positive instance of SAT-UNSAT.

⇐: Let (φ_1,φ_2) be a positive instance of SAT-UNSAT. This means, either φ_1 is sat or φ_2 is unsat. φ_2 however, cannot be unsat, since we set it to true. Therefore, φ_1 is sat (otherwise (φ_1,φ_2) would not be a positive instance). Since, $\varphi_1 = \varphi$, we get a positive instance for SAT.

Alternative solution
(please let me know if it's also correct, I know it's not as neat as the one above, but for some reason I came to this one instead):
We set φ1=φ and φ2=¬φ

⇒ same as above

⇐ Assume (φ1, φ2) is positive instance of SAT-UNSAT, i.e. φ1 is sat or φ2 is unsat.
Case φ1 is sat: by construction φ is therefore sat, yielding a positive instance of SAT.
Case φ2 is unsat: by construction we have: ¬φ is unsat, which is equivalent to φ is valid.
Because φ is valid it naturally is also satisfiable, yielding a positive instance of SAT.

Block 2 - fmi164

Status student

a) Does the program terminate?
Never terminates, since both x and y are declared as unsigned. Therefore, the condition of the while loop will never be false.

Does termination depend on the integer representation?

Status student suggestion:
Yes, it depends on the integer representation, for x and y being 64 bit signed integers the program would terminate, as x becomes negative in the first iteration and the loop condition will not be true anymore.

Status student alternative suggestion:
Here's a working example that terminates with 64bit integers (the code in the exam does not terminate - I tested both):

```
#include <inttypes.h>
```

```
#include <stdio.h>
int main() {
	int64_t x = 10;
	int64_t y = (1 << 16);
	printf("X: %" PRId64 "\tY: %" PRId64 "\n", x, y);
	while (x >= 0 && y >= 0) {
		y = y * y;
		x = x - y;
		printf("X: %" PRId64 "\tY: %" PRId64 "\n", x, y);
	}
	return x;
}
```

b)
To show: φ is sat ⇔ φ' is sat. Both φ and φ' are in CNF.

⇒: Assume φ is sat. φ' is φ minus one clause. Taking away a clause from a sat formula in CNF leaves the formula still satisfiable → φ' is sat.

⇐: Assume φ' is sat. Since φ' := φ\{C}, we only have to show that C is satisfiable, since we know that all other clauses in φ' already are satisfiable. There is a literal L in C, such that for **every** resolvent R of C with a C', R contains both x and ¬x for some x in φ (this was stated in the exercise description). Since x and ¬x is in R, R is true under every assignment.
In other words, every time we resolve C with another clause from C' upon variable L, the result is a valid clause. Therefore, we can always find a formula φ'' that was built from φ' using resolution that is satisfiable.

(TODO: Is the second part of the proof enough?)

Alternative 1 (status student, should be correct) for
⇐: Assume φ'.
φ' = { C', X} X … remaining clauses, have to be satisfiable, since φ' is satisfiable
φ = {C, C', X}

Showing C' sat -> C' and C sat
To show that a formula is sat, it is also possible to show that the formula is not unsatisfiable, λ sat ⇔ not (λ unsat).
C' and C sat ⇔ not (C' and C unsat)
The resolvent of C' and C is never empty, so it is not possible to derive the empty clause, which would mean C and C' are unsat. We have shown that C' and C can't be unsatisfiable, so C' and C are satisfiable, which means φ is satisfiable.

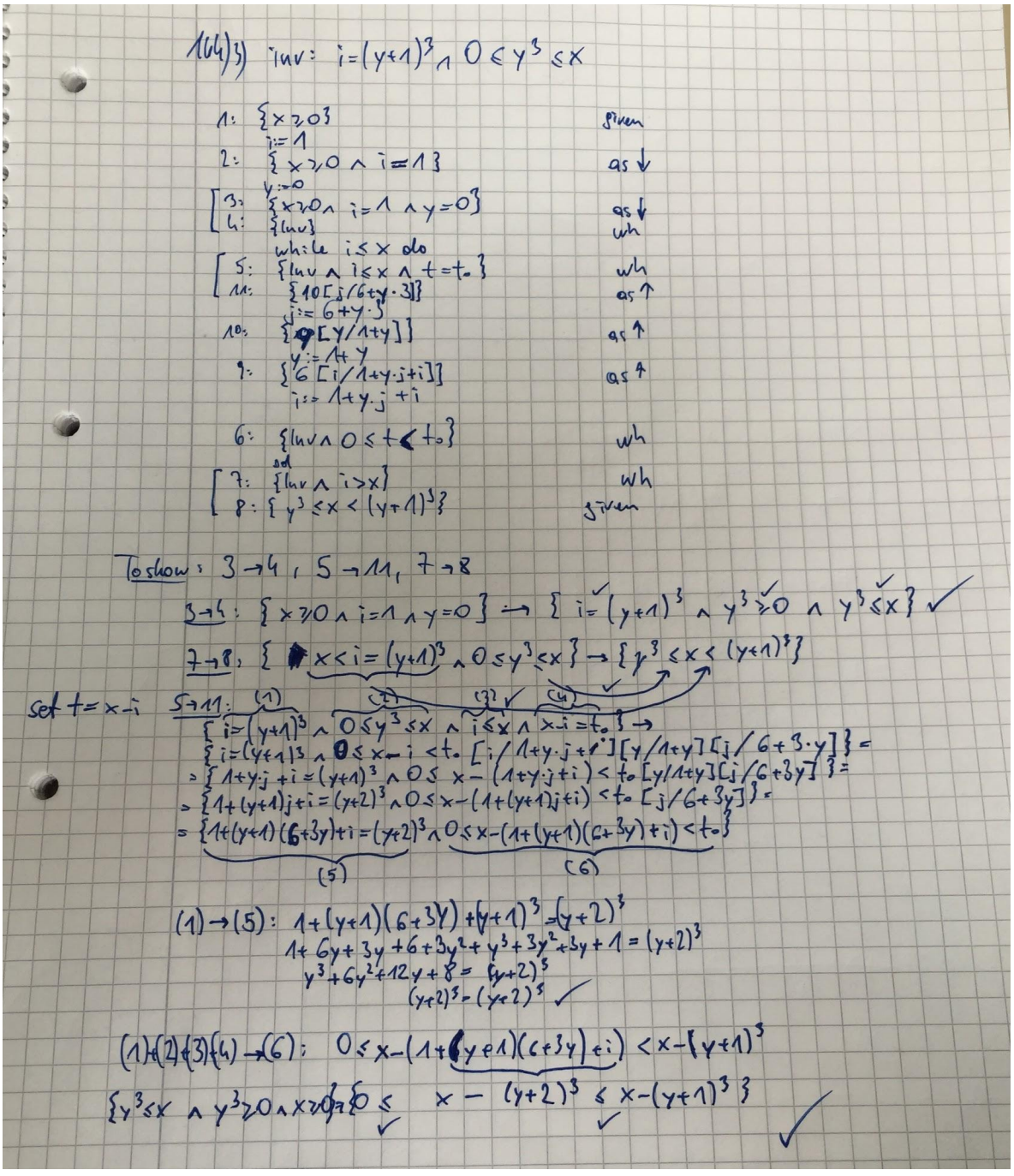
Alternative 2 (status student, not sure) for

⇐: Assume φ'.
Like first solution: Since φ' is satisfiable C' and X are satisfiable, we have to show that C is satisfiable. Since R contains x and ¬x, l e C, ¬l e C', there are 4 cases, where the x and -x are coming from:

Case	C	C' is sat	Construction of I' to make C sat
1	l v x v ¬x	¬l	C is always sat
2	l	¬l v x v ¬x	I'(l)=1
3	l v x	¬l v ¬x	I'(x)=I(x), I'(l)=¬I'(x)
4	l v ¬x	¬l v x	I'(x)=I(x), I'(l)=¬I'(x)

We have to show: C' sat -> C sat
Which is the same like showing: I e Mod(C') -> there exists a I' e Mod(C). So we construct an I' so that C is satisfiable. For this see last column in the table.

Block 3 - fmi164



Status: solved by student

Alternative solution that should be complete; however, due to the fact that this example is rather cumbersome, please check correctness thoroughly. I tried to give detailed arguments why the implications hold -- I hope that this is not required for the exam.

Describe what p does:

P calculates the integer cubic root of the given input value x (cf. postcondition).

Proof that following correctness assertion regarding total correctness:

We prove that $\{ F_1 \}\ p\ \{ F_2 \}$ is totally correct using annotation calculus:

$\{ F_1 : x \geq 0 \}$
$\{ F_{11} : F_{10}[i / 1] \}$ (as ↑)
i := 1;
$\{ F_{10} : Inv[y / 0] \}$ (as ↑)
y := 0;
$\{ F_3 : Inv \}$ (wht)
while $i \leq x$ do
  $\{ F_4 : Inv \wedge i \leq x \wedge t = t_0 \}$ (wht)
  $\{ F_9 : F_8[j / 6 + y*3] \}$ (as ↑)
  j := 6 + y*3;
  $\{ F_8 : F_7[y / 1 + y] \}$ (as ↑)
  y := 1 + y;
  $\{ F_7 : F_5[i / 1 + y*j + i] \}$ (as ↑)
  i := 1 + y*j + i
  $\{ F_5 : Inv \wedge (i \leq x \Rightarrow 0 \leq t < t_0) \}$ (wht)
od
$\{ F_6 : Inv \wedge \neg(i \leq x) \}$ (wht)
$\{ F_2 : y^3 \leq x < (y+1)^3 \}$

We choose Inv: $i = (y+1)^3 \wedge 0 \leq y^3 \leq x$ (according to the hint in the specification) and t: x - i (considering that i grows towards x and the difference gets smaller on each iteration: $0 \leq x - i$).

To show total correctness of { F1 } p { F2}, it remains to show the following three implications:

(1) $F_1 \Rightarrow F_{11}$

$x \geq 0 \Rightarrow (i = (y+1)^3 \wedge 0 \leq y^3 \leq x)[y / 0][i / 1]$
$x \geq 0 \Rightarrow (1 = (0+1)^3 \wedge 0 \leq 0^3 \leq x$
$x \geq 0 \Rightarrow (1 = 1 \wedge 0 \leq x)$

The implication in the last line is obviously true.

(2) $F_4 \Rightarrow F_9$

$(i = (y+1)^3 \wedge 0 \leq y^3 \leq x \wedge i \leq x \wedge x-i = t_0) \Rightarrow$
$(i = (y+1)^3 \wedge 0 \leq y^3 \leq x \wedge (i \leq x \Rightarrow 0 \leq x-i < t_0))[i/1+y*j+i][y/1+y][j/6+y*3]$

$(i = (y+1)^3 \wedge 0 \leq y^3 \leq x \wedge (i \leq x \Rightarrow 0 \leq x-i < t_0))[i/1+y*j+i][y/1+y][j/6+y*3] =$
$= (1+y*j+i = (y+1)^3 \wedge 0 \leq y^3 \leq x \wedge (1+y*j+i \leq x \Rightarrow 0 \leq x-(1+y*j+i) < t_0))[y/1+y][j/6+y*3]$
$= (1+(1+y)*j+i = (y+1+1)^3 \wedge 0 \leq (y+1)^3 \leq x \wedge (1+(y+1)*j+i \leq x \Rightarrow 0 \leq x-(1+(y+1)*j+i) < t_0))[j/6+y*3]$
$= 1+(1+y)*(6+y*3)+i = (y+1+1)^3 \wedge 0 \leq (y+1)^3 \leq x \wedge (1+(y+1)*(6+y*3)+i \leq x \Rightarrow 0 \leq x-(1+(y+1)*(6+y*3)+i) < t_0)$

We show (2) by implying the LHS on each conjunct of the RHS and show that all of these smaller implications evaluate to true.

(a) $(i = (y+1)^3 \wedge 0 \leq y^3 \leq x \wedge i \leq x \wedge x-i = t_0) \Rightarrow (1+(1+y)*(6+y*3)+i = (y+1+1)^3$

Given that $i = (y+1)^3$ is true, we can show that $1+(1+y)*(6+y*3)+i = (y+1+1)^3$ is true by simplifying the equation:

(b)
$1+(1+y)*(6+y*3)+(y+1)^3 = 1+6+3y+6y+3y^2+y^3+3y^2+3y+1 = y^3+6y^2+12y+8 = (y+2)^3$

(c) $(i = (y+1)^3 \wedge 0 \leq y^3 \leq x \wedge i \leq x \wedge x-i = t_0) \Rightarrow 0 \leq (y+1)^3 \leq x$

Given that $i \leq x$ and $i = (y+1)^3$ and $0 \leq y^3 \leq x$, we may conclude $0 \leq (y+1)^3 \leq x$.

(d) $(i = (y+1)^3 \wedge 0 \leq y^3 \leq x \wedge i \leq x \wedge x-i = t_0 \wedge 1+(y+1)*(6+y*3)+i \leq x) \Rightarrow 0 \leq x-(1+(y+1)*(6+y*3)+i) < t_0$

We split the implication again in two parts, one for each conjunct on the RHS:

(i) LHS $\Rightarrow 0 \leq x-(1+(y+1)*(6+y*3)+i)$

Given $i = (y+1)^3$, $i \leq x$ and $0 \leq y^3 \leq x$ are true, one can show the RHS:

$x-(1+(y+1)*(6+y*3)+i) = x-(1+6y+3y^2+6+3y+i) = x-1-6y-3y^2-6-3y-i$
$= x-1-6y-3y^2-6-3y-(y+1)^3 = x-1-6y-3y^2-6-3y-(y^3-3y^2-3y-1) =$
$= x-1-6y-3y^2-6-3y-y^3+3y^2+3y+1 = x-(y^3+6y+6)$

From the premises, we know $0 \le y^3 \le (y+1)^3 \le x$, hence $0 \le x-y^3$ and $0 \le x-(y+1)^3$. Since $y^3 \le y^3+6y+6 \le (y+1)^3$, we may conclude that $0 \le x-(y^3+6y+6)$, therefore $0 \le x-(1+(y+1)*(6+y*3)+i)$.

(ii) LHS $\Rightarrow x-(1+(y+1)*(6+y*3)+i) < t_0$

RHS: $x-(1+(y+1)*(6+y*3)+i) = x-i-(1+6y+3y^2+6+3y) = x-i-(3y^2+9y+7)$.

Since $x-i = t_0$ is true, it remains to show that $(3y^2+9y+7) > 0$. However, since $0 \le y^3$, we know that $y \ge 0$; therefore, we may conclude that $3y^2+9y+7 > 0$ holds.

(3) $F_6 \Rightarrow F_2$

$i = (y+1)^3 \wedge 0 \le y^3 \le x \wedge i > x \Rightarrow y^3 \le x < (y+1)^3$

From $0 \le y^3 < x$, $i = (y+1)^3$ and $i > x$, we may conclude $y^3 \le x$ and $x < (y+1)^3$, hence $y^3 \le x < (y+1)^3$ is true as well.

We have shown the implications (1), (2) and (3), therefore $\{ F_1 \}\ p\ \{ F_2 \}$ is totally correct.

Solution

$\{\,1\colon a \geq 0\,\}$
$\{\,7\colon Inv[c/0][b/1]\,\}$ as↑
$b := 1;$
$\{\,6\colon Inv[c/0]\,\}$ as↑
$c := 0;$
$\{\,Inv\colon b = (c+1)^3 \wedge 0 \leq c^3 \leq a\,\}$ wht‴
while $b \leq a$ **do**
  $\{\,3\colon Inv \wedge b \leq a \wedge 0 \leq t = t_0\,\}$ wht‴
  $\{\,10\colon Inv \wedge (b \leq a \Rightarrow 0 \leq t < t_0)[b/b+cd+1][c/c+1][d/3c+6]\,\}$ as↑
  $d := 3 * c + 6;$
  $\{\,9\colon Inv \wedge (b \leq a \Rightarrow 0 \leq t < t_0)[b/b+cd+1][c/c+1]\,\}$ as↑
  $c := c + 1;$
  $\{\,8\colon Inv \wedge (b \leq a \Rightarrow 0 \leq t < t_0)[b/b+cd+1]\,\}$ as↑
  $b := b + c * d + 1$
  $\{\,4\colon Inv \wedge (b \leq a \Rightarrow 0 \leq t < t_0)\,\}$ wht‴
od
$\{\,5\colon Inv \wedge a < b\,\}$ wht‴
$\{\,2\colon c^3 \leq a < (c+1)^3\,\}$

Proof of the implications:

$1 \Rightarrow 7$: $a \geq 0$ obviously implies $Inv[c/0][b/1] = (1 = (0+1)^3 \wedge 0 \leq 0^3 \leq a)$

$5 \Rightarrow 2$: The third conjunct of the invariant, $c^3 \leq a$, and the negated loop condition together yield $c^3 \leq a < b$. Using $b = (c+1)^3$ we obtain formula 2.

We split implication $3 \Rightarrow 10$ into two parts, one for partial correctness and one for termination:

$$Inv \wedge b \leq a \Rightarrow Inv[b/b+cd+1][c/c+1][d/3c+6]$$
$$Inv \wedge b \leq a \wedge 0 \leq t = t_0 \Rightarrow (b \leq a \Rightarrow 0 \leq t < t_0)[b/b+cd+1][c/c+1][d/3c+6]$$

Partial correctness: We start by simplifying the right-hand side.

$$\begin{aligned}
&Inv[b/b+cd+1][c/c+1][d/3c+6]\\
&= (b = (c+1)^3 \wedge 0 \leq c^3 \leq a)[b/b+cd+1][c/c+1][d/3c+6]\\
&= (b + (c+1)(3c+6) + 1 = (c+2)^3 \wedge 0 \leq (c+1)^3 \leq a)\\
&= (b + 3c^2 + 9c + 6 + 1 = ((c+1)+1)^3 \wedge 0 \leq (c+1)^3 \leq a)\\
&= (b + 3c^2 + 9c + 7 = (c+1)^3 + 3(c+1)^2 + 3(c+1) + 1 \wedge 0 \leq (c+1)^3 \leq a)\\
&= (b + 3c^2 + 9c + 7 = (c+1)^3 + 3c^2 + 6c + 3 + 3c + 3 + 1 \wedge 0 \leq (c+1)^3 \leq a)\\
&= (b = (c+1)^3 \wedge 0 \leq (c+1)^3 \leq a)
\end{aligned}$$

So we have to prove the implication

$$b = (c+1)^3 \wedge 0 \leq c^3 \leq a \wedge b \leq a \Rightarrow b = (c+1)^3 \wedge 0 \leq (c+1)^3 \leq a$$

We consider the conjuncts on the right-hand side in turn:

- $b = (c+1)^3$: occurs also on the left-hand side.
- $0 \leq (c+1)^3$: follows from $0 \leq c^3$.
- $(c+1)^3 \leq a$: follows from $b \leq a$ and $b = (c+1)^3$.

Termination: As variant t we choose $a-b$ (alternatively: $a-c^3$). We start by simplifying the right-hand side.

$$\begin{aligned}&(b \leq a \Rightarrow 0 \leq a-b < t_0)[b/b+cd+1][c/c+1][d/3c+6]\\ &= ((b \leq a \Rightarrow 0 \leq a-b) \wedge (b \leq a \Rightarrow a-b < t_0))[b/b+cd+1][c/c+1][d/3c+6]\end{aligned}$$

The first implication is obviously true, it remains to prove the second one. In fact, we prove a stronger conclusion (without the assumption $(b \leq a)[b/\cdots][c/\cdots][d/\cdots]$):

$$\begin{aligned}&a-b < t_0[b/b+cd+1][c/c+1][d/3c+6]\\ &= a-(b+(c+1))(3c+6)+1) < t_0\end{aligned}$$

So we have to prove the implication

$$Inv \wedge b \leq a \wedge 0 \leq a-b = t_0 \Rightarrow a-(b+(c+1)(3c+6)+1) < t_0$$

Using $a-b = t_0$ the right-hand side becomes

$$a-(b+(c+1))(3c+6)+1) < a-b$$

and further

$$(b+(c+1))(3c+6)+1 > b$$

Because of *Inv* both, b and c, are non-negative, hence the inequality holds.

Function computed by the program: Integer cubic root, $i = \lfloor \sqrt[3]{a} \rfloor$.

Block 4 - fmi164

Status student

a)

```
int set_size = 0;
int covered_vertices[N];      // assume, that covered_vertices is
                              // initialized with zeros
for (int i=0; i<N; i++){
```

```
      a = nondet_bool();
      if (a==1) {
            covered_vertices[i]=1;
            set_size+=1;
            for(int j=0; j<N; j++){
                  if(edge[i][j] == 1) {
                        covered_vertices[j]=1;
                  }
            }
            if(set_size == K) break;
      }
}
int has_uncovered_vertices = 0;
for (int i=0; i<N; i++){
      if (covered_vertices[i] == 0)
            has_uncovered_vertices = 1;
}
assert(has_uncovered_vertices == 0); // break if dominating set found
```

b)
Status: approved by professor
All formulas are valid CTL* formulas.

CTL: **AG**(b): s_0, s_2
CTL: **EG**(b): s_0, s_2, s_3
CTL: **EF**(a): s_1,s_3
LTL: **G**(b): s_0,s_2
CTL: **E**[(a $\wedge$ b) **U** (c)]: s_1, s_4

c)
Status: solved by student

Φ_1 = EFAG(p)
Φ_2 = EFG(p)
Φ_3 = EGF(p)

Following implications have to be shown:
1.) $\Phi_1 \models \Phi_2$ and $\Phi_2 \models \Phi_1$
2.) $\Phi_1 \models \Phi_3$ and $\Phi_3 \models \Phi_1$
3.) $\Phi_2 \models \Phi_3$ and $\Phi_3 \models \Phi_2$

1.)
EFAG(p) $\models$ EFG(p)

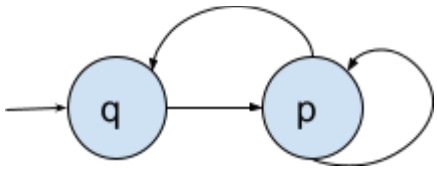
Implication holds

Proof: Assume K, $s_0 \models$ EFAG(p), we show that K, $s_0 \models$ EFG(p)
(K is a Kripke Structure and s_0 an initial state of K)
We know there exists a path π where FAG(p) holds. Therefore for some state s_i on this path π AG(p) has to hold. It follows that G(p) holds for all paths starting from s_i and therefore there has to be a path where FG(p) holds. Hence K, $s_0 \models$ EFG(p) is true.

More verbose proof of EFAG(p) $\models$ EFG(p):
$M,s_0 \models EFAG(p)$
$M,\pi \models FAG(p)$ for a path $\pi = (s_0,...)$
$\exists\, k \geq 0: M,\pi^k \models AG(p)$
Let μ be an arbitrary path starting in the starting state of π^k.
$M,\mu \models G(p)$
Since p holds globally on the computation path μ, it is implied that it holds from a subpath onwards.
$\exists\, i \geq 0: M,\mu^i \models G(p)$
$M,\mu \models FG(p)$
Since μ is a subpath of π^k, the formula also holds on π^k and also on π.
$M, \pi \models FG(p)$
$M, s_0 \models EFG(p)$.

EFG(p) $\models$ EFAG(p)
Implication does not hold

Counterexample:



K, $s_0 \models$ EFG(p) but K, $s_0 \not\models$ EFAG(p)
(EFG(p): there has to be (at least) one path in the future, for which p holds globally)

2.)
EFAG(p) $\models$ EGF(p)
Implication holds

Proof: Assume K, $s_0 \models$ EFAG(p), we show that K, $s_0 \models$ EGF(p)
(K is a Kripke Structure and s_0 an initial state of K)

We know there exists a path π where FAG(p) holds. Therefore for some state s_i on this path π AG(p) has to hold. It follows that G(p) holds for all paths starting from s_i and therefore there has to be a path where GF(p). Hence K, $s_0 \models$ EGF(p) is true.

EGF(p) $\models$ EFAG(p)
Implication does not hold

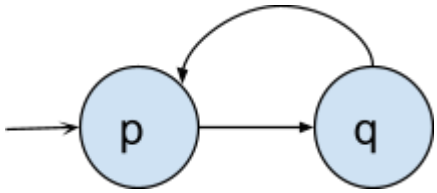
Counterexample:
Same as in 1.)
K, $s_0 \models$ EGF(p) but K, $s_0 \not\models$ EFAG(p)

3.)
EFG(p) $\models$ EGF(p)
Implication holds

Proof: Assume K, $s_0 \models$ EFG(p), we show that K, $s_0 \models$ EGF(p)
(K is a Kripke Structure and s_0 an initial state of K)
We know there exists a path π where FG(p) holds. Therefore for some state s_i a path π_i exists where G(p) holds. It follows that for π_i GF(p) also holds. Hence we have a path s.t GF(p) holds and thus K, $s_0 \models$ EGF(p) is true.

EGF(p) $\models$ EFG(p)
Implication does not hold

Counterexample:



K, $s_0 \models$ EGF(p) but K, $s_0 \not\models$ EFG(p)

Comment:
I would write: K, $\pi^i \models$ G(p)
this can be written as (by semantics of G):
$\forall k \geq i$ M,$\pi^k \models$ p which implies $\exists\ h \geq i$: M,$\pi^h \models$ p and $\forall\ k \geq h$: M,$\pi^k \models$ p
and this can be written as M,$\pi^k \models$ GF(p).

….I'm absolutely not sure if this solution is correct, comments are very welcome

Exam 06.05.2016 (FMI.163.pdf)

http://www.logic.at/lvas/fminf/angaben/fmi163.pdf

Block 1 - fmi163

Solved by student

Consider the following problems:

GROUPS

INSTANCE: A pair (P, D), where

- $P = \{p_1, \ldots, p_n\}$ is a set of elements (intuitively, P is a set of persons), and
- $D \subseteq P \times P$ is a binary relation over P (intuitively, $(p, p') \in D$ means that p dislikes p').

QUESTION: Is it possible to divide the person of P into 5 groups such that none of the groups contains persons p, p' such that p dislikes p'? That is, does there exist a partitioning of P into sets $G_1, \ldots G_5$ such that $(p, p') \notin D$ holds for all $1 \leq i \leq 5$ and all pairs $p, p' \in D_i$?

GENERALIZED-COLORING (GC)

INSTANCE: A pair (G, k), where $G = (V, E)$ is a directed graph and k is an integer.

QUESTION: Does there exist a k-coloring of G? That is, does there exist a coloring function $\mu : V \to \{1, \ldots, k\}$ such that $\mu(v) \neq \mu(v')$ for all edges $(v, v') \in E$?

Provide a polynomial time reduction from **GROUPS** to **GENERALIZED-COLORING**, and prove the correctness of your reduction.

Reduction:

Let (P, D) be an arbitrary Instance of GROUPS.
We construct an Instance (G, k) of GEN-COL as follows:

- Set k to 5 *// Number of Groups in (P, D)*
- Construct the directed graph G=(V, E) as following:
  - For each p ∈ P add a vertex v to V
  - For each entry $[p_1, p_2]$ in the binary relation D add an edge $[v_1, v_2]$ to E
- Set the coloring of each vertex v e V to its corresponding partitioned set. Therefore: μ: V → $\{G_1, \ldots, G_5\}$

The above reduction is feasible in polynomial-time.

Prove correctness of reduction:

(P, D) positive Instance of GROUPS ⇔ (G, k) positive Instance of GEN-COL

"⇒":
Assume (P, D) is a positive Instance of GROUPS, i.e. there exists a partitioning into groups G_1, …, G_5 s.t. p, p' ∉ D ∀ groups.
By construction of the graph G we have an edge $[v_1,v_2]$ for every entry p_1, p_2 ∈ D.
Since p_1, p_2 denotes p_1 "dislikes" p_2 and (P, D) is a correct partitioning we know that $p_1$and p_2 do not belong to the same group. From construction of μ (which gives each vertex v the color of its corresponding group) we know that each pair of vertices $[v_1,v_2]$ ∈ E has a different coloring.
Therefore ∀ $[v_1,v_2]$ ∈ E it is guaranteed that $\mu(v_1) \neq \mu(v_2)$ and (G, k) is a positive instance of GEN-COL.

"⇐":
Assume (G,k) is a positive Instance of GEN-COL, i.e. there exists $\mu: V \rightarrow \{1, \ldots, yk\}$ s.t. $\mu(v_1) \neq \mu(v_2)$ ∀ $[v_1,v_2]$ ∈ E.
Since there exists a correct k-coloring of G and k is set to 5 by the reduction, we have 5 groups of vertices with different colorings.
For each pair of vertices $[v_1,v_2]$ ∈ E of G it is assured that v_1 and v_2 have a different coloring. Therefore between any two vertices v_1 and v_2 of the same color there does not exist a connecting edge. (which would mean v_1 "dislikes" v_2).
Thus there exists a partitioning into k=5 groups s.t. no two elements v_1, v_2 in V "dislike" each other. Therefore (G,k) is a positive Instance of GROUPS. ∎

Example:
P = {P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}}
D = {
(P_1 dislikes P_5), (P_8 dislikes P_2),
(P_3 dislikes P_7), (P_4 dislikes P_3),
(P_2 dislikes P_7), (P_5 dislikes P_{10}),
(P_8 dislikes P_{10}), (P_5 dislikes P_8),
(P_1 dislikes P_8), (P_8 dislikes P_1)
}

Block 2 - fmi163

a) **Solved by student**
=>
Assume that phi(x1,..,xn) is valid, i.e. every interpretation over every variable assignment evaluates to true. Take an arbitrary interpretation I. We argue that you

can create a model I' for phi(c1, …, cn) from this interpretation. We set I'=I and keep the variable assignment alpha (since x1,...,xn do not occur in phi(c1,...,cn) this does not matter). Given the variable assignment alpha of I, we extend I' the following way: I'(ci) = alpha(xi).
Since I is a model, I' does not change the value of xi and I' ⊧ xi <-> ci, by the equivalence replacement lemma I' is a model.

<=
Assume that phi(c1,...,cn) is valid, i.e. every interpretation is a model. Take an arbitrary interpretation I. We argue that you can construct a model I' for phi(x1,...,xn). Set I'=I (the definitions for c1,..,cn do not matter, as they are newly introduced constants). Now create a variable assignment alpha for I' the following way: alpha(xi) = I(ci).
Since I is a model, I' does not change the value of ci and I' ⊧ xi <-> ci, by the equivalence replacement lemma I' is a model.

I don't know what he wants to hear on the second part of the question, but I guess the last part of the slides. Given that you replace predicates, constants and boolean variables, you can construct an equisat propositional formula that does not interpret the function symbols, besides functional congruence.

b) https://www.informatik-forum.at/showthread.php?112972-Solution-for-6-5-2016-2a-b

Block 3 - fmi163

Solution seems very similar to Exercise 2gh in dvsw-solutions.pdf, i got
wp(p, x=y)=(x+y)>0 ^ x>=0
Anyone else?

Von WS14 if then und else ist vertauscht sonst ganz gleich :-)

Exercise 1 Let p be the following program:

$x := x + y;$
if $x < 0$ then
  abort
else
  while $x \neq y$ do
    $x := x + 1;$
    $y := y + 2$
  od
fi

Exercise 2 Let p be the program of exercise 1. Show that $\{ x = 2y \wedge y > 2 \}\, p\, \{ x = y \}$ is totally correct using weakest preconditions.

Solution

We show that the precondition implies the weakest precondition of the program with respect to the postcondition.

$\mathrm{wp}(x := x + y; \mathsf{if}\ \cdots,\ x = y) = \mathrm{wp}(x := x + y,\ \mathrm{wp}(\mathsf{if}\ \cdots,\ x = y))$

$\mathrm{wp}(\mathsf{if}\ \cdots,\ x = y)$
$= ((x < 0 \wedge \mathrm{wp}(\mathsf{abort}, x = y)) \vee (x \geq 0 \wedge \mathrm{wp}(\mathsf{while}\ \cdots,\ x = y)))$
$= ((x < 0 \wedge \mathrm{false}) \vee (x \geq 0 \wedge \mathrm{wp}(\mathsf{while}\ \cdots,\ x = y)))$
$= (\mathrm{false} \vee (x \geq 0 \wedge \mathrm{wp}(\mathsf{while}\ \cdots,\ x = y)))$
$= (x \geq 0 \wedge \mathrm{wp}(\mathsf{while}\ \cdots,\ x = y))$

$\mathrm{wp}(\mathsf{while}\ \cdots,\ x = y) = \exists i \geq 0\, F_i$
$F_0 : \neg(x \neq y) \wedge x = y$
$F_1 : x \neq y \wedge \mathrm{wp}(x := x + 1; y := y + 2,\ x = y)$
$= (x \neq y \wedge x + 1 = y + 2)$
$= (x = y + 1)$
$F_i : x = y + i$ (guess)
$F_{i+1} : x \neq y \wedge \mathrm{wp}(x := x + 1; y := y + 2,\ F_i)$
$= x \neq y \wedge (x + 1) = (y + 2) + i$
$= x \neq y \wedge x = y + i + 1$
$= (x = y + i + 1)$ (proof)
$= \exists i \geq 0\, (x = y + i)$
$= x \geq y$

$= (x \geq 0 \wedge x \geq y)$

$= \mathrm{wp}(x := x + y,\ x \geq 0 \wedge x \geq y)$
$= (x + y) \geq 0 \wedge (x + y) \geq y$
$= (x + y) \geq 0 \wedge x \geq 0$

It remains to show that the precondition implies the weakest precondition.

$$x = 2y \wedge y > 2 \Rightarrow \mathrm{wp}(p, x = y)$$
$$\Rightarrow (x + y) \geq 0 \wedge x \geq 0$$

The two conjuncts of the conclusion can be proven separately:

$$x = 2y \wedge y > 2 \Rightarrow (x + y) \geq 0$$
$$x = 2y \wedge y > 2 \Rightarrow x \geq 0$$

The first implication is valid, since $x = 2y$ and $y > 2$ imply $(x + y) = 3y > 6 \geq 0$. The second implication holds since $x = 2y$ and $y > 2$ imply $x > 4$, which implies the conclusion $x \geq 0$.

Block 4 - fmi163

4a) solved by student

H={(s0,t0),(s4,t2),(s3,t2),(s1,t4),(s2,t3),(s1,t1),(s0,t5)}

> Q:*Do I really need the last relation (s0, t5)?*
> *A: Yes! how else would you simulate the move from s1->s0 ? also this condition needs to be met: For every successor t such that(s,t) ∈ R, there is a corresponding successor t' such that(s',t') ∈ R' and (t,t') ∈ H (from slide 9/40 in Simulation and Abstraction)*
> *A2: I don't think duplicates should be present in H. The goal is to map all states from M1 to a corresponding state in M2, such that all transitions are correct in respect to the labels. In the concrete example: the transition s0->s3->s1->s0 in M1 would also simulate the transition t0->t2->t4->t5 in M2, regardless of the mapping in H, because the labels correspond in those transitions. In my final solution, I have the following: {(s0,t0),(s4,t2),(s2,t3),(s1,t1),(s3,t2)}*
> *RE: For your simulation, check the following moves in M1: s0, s3, s1. Your simulation in M2 would be: t0, t2, t1. This is not possible since there is no edge from t2 to t1.*
> → ***you do need the "duplicates"!!***

4b) solved by student

For each of the following three equivalences, determine whether the equivalence holds. If so, prove correctness. Otherwise, disprove by giving a Kripke structure M and a path π such that (M, π) satisfies one side of the equivalence but not the other.

i. true U p ≡ F p ∧ ¬G ¬p

Equivalence holds

Proof:1

We have to show that for every Kripke structure K and every state s in K it holds that K,s ⊧ true U p if and only if K,s ⊧ F p ∧ ¬G ¬p.

Let π = s0,s1,... be <u>some path</u> starting at s_0.

<u>1) "=>" - Direction</u>

If K,s ⊧ **true U p** then K,s ⊧ **F p ∧ ¬G ¬p**

from π ⊧ true U p we know that on some state $s_{i\geq0}$ p holds.

Therefore π ⊧ Fp holds.

Since p holds on state s_i, ¬p does not hold on at least one state.

Therefore π ⊧ ¬G¬p also holds. (¬G¬p means: ¬p must not hold on every state)

<u>2) "<=" - Direction</u>

If K,s ⊧ F p ∧ ¬G ¬p then K,s ⊧ true U p

From π ⊧ Fp we know on some state s_i≥0 p has to hold.

Therefore we also know ¬p does not hold on at least one state (namely on s_i).

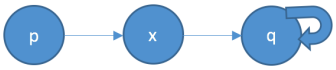
Hence π ⊧ ¬G ¬p = true.

Therefore π ⊧ true U p holds.

ii. p U q ≡ p ∧ F q

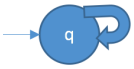
Equivalence does not hold

Counter-Example:



Holds for $p \wedge F q$ but not for $p U q$

Alternative Counter-Example:



Holds for $p U q$ but not for $p \wedge F q$

iii. GFGF q ≡ GF q

Equivalence holds

Proof:
We have to show that for every Kripke structure K and every state s in K it holds that
K,s $\models$ Fq if and only if K,s $\models$ GFq.
Let π = s0,s1,... be some path starting at s_0.

1) "=>" - Direction
If K,s $\models$ **GFGFq** then K,s $\models$ **GF q**

Since for every path GFGFq has to hold, $\pi \models$ **GFGFq**

from $\pi \models$ **GFGFq** we know that for every state $s_{i\geq 0}$ in π **FGFq** holds.
Therefore for some state $s_{i\geq 0}$ **GFq** holds.
Therefore for every state starting from $s_{i\geq 0}$ **Fq** holds.
Hence finally for some state starting from $s_{i\geq 0}$ **q** holds.

Since **q** holds for some state $s_{i\geq 0}$ holds, it holds that $\pi \models$ **Fq.**
And since for every state it has to hold that at some state $s_{i\geq 0}$ **q** holds, it also holds that $\pi \models$ **GFq.**

2) "<=" - Direction
If K,s $\models$ **GF q** then K,s $\models$ **GFGFq**

Since for every path GFq has to hold $\pi \models$ **GFq.**
Applying GF pairwise arbitrarily often will not change the statement,
that q holds at some state $s_i \geq 0$.
It follows that $\pi \models$ **GFGFq**

4c) solved by student

```
int sum_s1=0;
int sum_s2=0;

for(int i=0; i<N; i++) {
      if(nondet_bool()) {
            sum_s1 += values[i];
      } else {
            sum_s2 += values[i];
      }
}

assert(sum_s1!=sum_s2);
```

Exam 18.03.2016 (FMI.162.pdf)

http://www.logic.at/lvas/fminf/angaben/fmi162.pdf

Block 1 - fmi162

Status: Solved by student

Let $x = (\Pi)$ be an arbitrary instance of SMALLER.

We first define $STRINGS$ as the enumeration of strings over the alphabet $\{0, 1\}$ i.e. $STRINGS = \{0, 1, 00, 01, 10, 11, ...\}$..

We then construct $\Pi_D(String\ \Pi)$ as follows:

```
1. Π_D(String Π):
2.   for each I in STRINGS:
3.     if |Π(I)| < |I|:
4.       return true
```

We claim that Π_D is a semi-decision procedure for SMALLER, i.e. that

A. If x is a positive instance of SMALLER, Π_D returns true
B. If x is a negative instance of SMALLER, Π_D returns false or does not terminate

Proof:

A. If x is a positive instance of SMALLER, then there exists a string $I \in STRINGS$ for which $|I| \leq |\Pi(I)|$. Because Π_D iterates over all possible $STRINGS$, and $STRINGS$ is countably infinite, if such an I exist, it will be found in a finite amount of steps, and Π_D will terminate at line 4 ⇒ Π_D returns true.
B. If x is a negative instance of SMALLER, then Π_D will continue to iterate over $STRINGS$ and never halt ⇒ Π_D does not terminate.

Alternative semi-decision procedure without the definition of infinite STRINGS list

Not that defining such a list wouldn't be fine, but for people like me this is probably an easier solution to remember.

```
1. Π_D(String Π):
2.     strings = { "0", "1"};
3.     Do {
4.         stringsIt = {};
5.         For (String n : strings) {
6.             if |n| < |Π(n)|:
7.                 Return true;
8.             Else {
9.                 stringsIt += (n + "0");
10.                stringsIt += (n + "1");
11.            }
12.        }
13.        strings = stringsIt;
14.    } while (true);
```

Block 2 - fmi162

a) https://tuwel.tuwien.ac.at/pluginfile.php/603025/mod_folder/content/0/hw1.pdf?forcedownload=1
(see tuwel supplementary hw1.pdf)

Solution to Exercise 1

We make precise here on which formula FC^E and $flat^E$ depend. Both are computed according to AR. First observe that

$$\varphi^{EUF} \text{ is valid} \quad \text{iff} \quad FC^E(\varphi^{EUF}) \rightarrow flat^E(\varphi^{EUF}) \text{ is valid.}$$

Taking the above hint into account, we get

$$\begin{aligned} \neg\varphi^{EUF} \text{ is valid} \quad &\text{iff} \quad FC^E(\neg\varphi^{EUF}) \rightarrow flat^E(\neg\varphi^{EUF}) \text{ is valid} \\ &\text{iff} \quad FC^E(\varphi^{EUF}) \rightarrow flat^E(\neg\varphi^{EUF}) \text{ is valid.} \end{aligned}$$

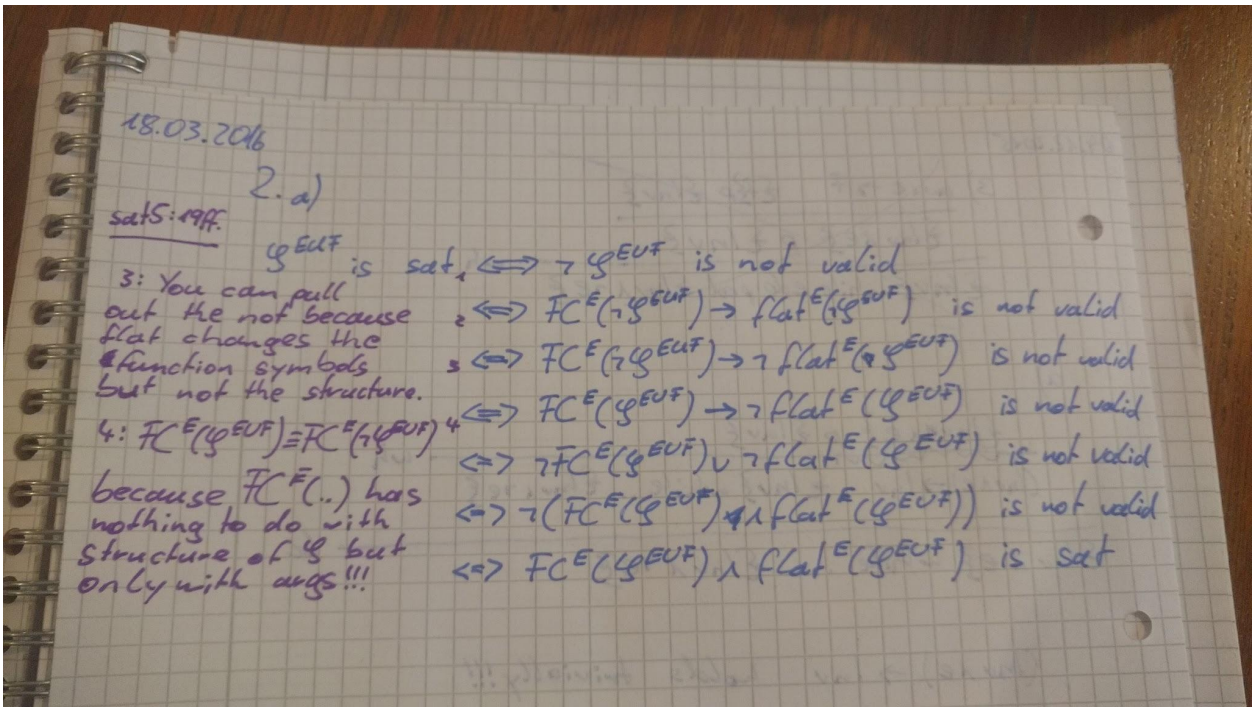
Since φ^{EUF} and $flat^E(\varphi^{EUF})$ have the same propositional structure (only some arguments of literals change during the computation of $flat^E(\varphi^{EUF})$ from φ^{EUF}), the equality $flat^E(\neg\varphi^{EUF}) = \neg flat^E(\varphi^{EUF})$ holds. Consequently,

$$\neg\varphi^{EUF} \text{ is valid} \quad \text{iff} \quad FC^E(\varphi^{EUF}) \rightarrow \neg flat^E(\varphi^{EUF}) \text{ is valid.} \qquad (*)$$

For space reasons, we omit the superscript EUF in the following. Then we have:

			Explanation
φ is sat	iff	$\neg\varphi$ is *not* valid	Ψ is sat iff $\neg\Psi$ is not valid
	iff	$FC^E(\varphi) \rightarrow \neg flat^E(\varphi)$ is *not* valid	from $(*)$ above
	iff	$\neg\neg(FC^E(\varphi) \rightarrow \neg flat^E(\varphi))$ is *not* valid	$\neg\neg\Psi \equiv \Psi$
	iff	$\neg(FC^E(\varphi) \rightarrow \neg flat^E(\varphi))$ is sat	Ψ is sat iff $\neg\Psi$ is not valid
	iff	$FC^E(\varphi) \wedge flat^E(\varphi)$ is sat	basic propositional manipulations

Effectively the same (was accepted in the exam):



b)

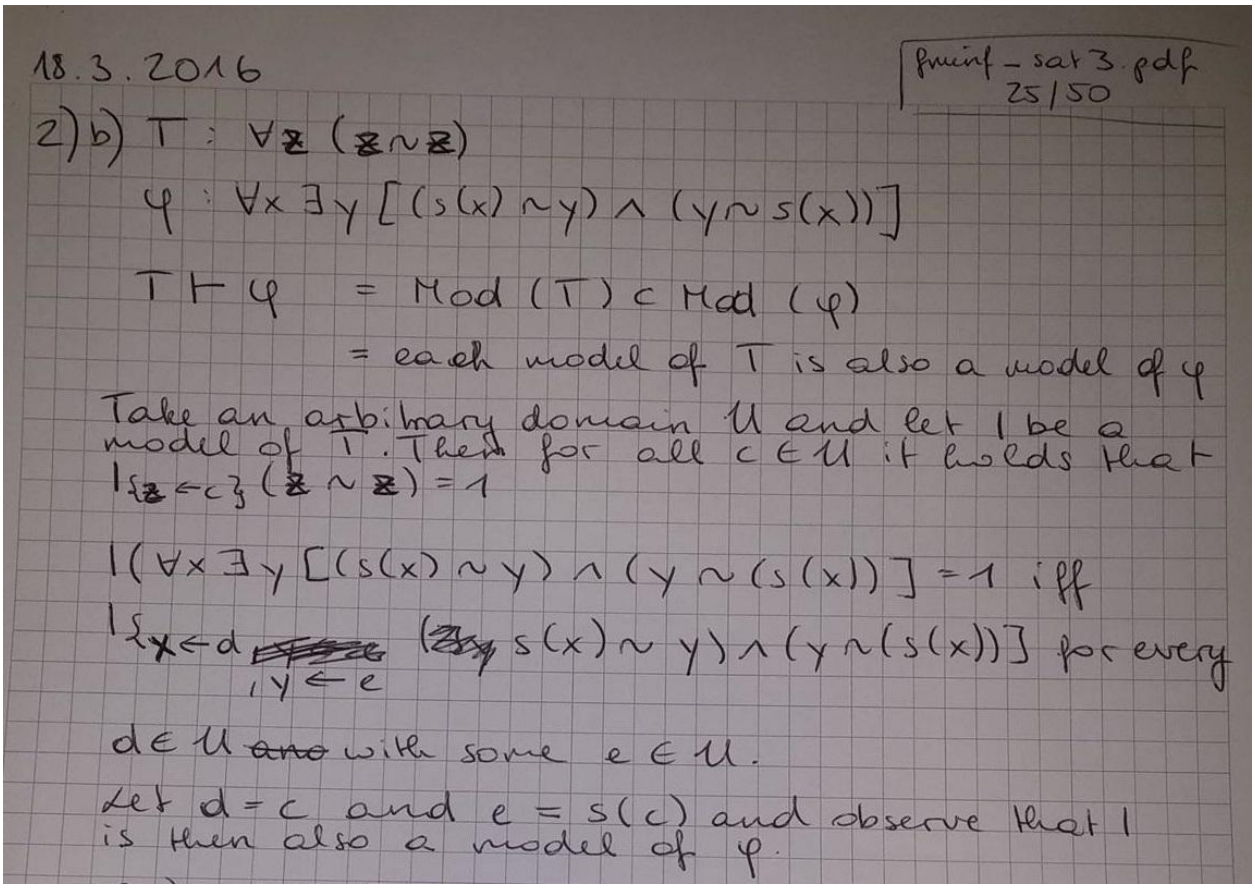
(b) Let $\varphi : \forall x \exists y[(s(x) \sim y) \wedge (y \sim s(x))]$, where $\sim/2$ is a binary predicate written in infix notation. Let T be a theory which forces $\sim/2$ to be reflexive. Show by purely semantical means that $T \models \varphi$ holds. (Hint: show that $Mod(T) \subseteq Mod(\varphi)$.) **(5 points)**

Since ~ is reflexive, it means that x~y is equal to y~x (since reflexivity means that the element is related to itself).

Proof by contradiction:
Let there be an interpretation I that is a model of T but is not a model of ϕ. Then it must hold that we can find two arbitrary x,y where:
Case 1: x~y holds, but not y~x. Since this violates the reflexivity of ~, this cannot be the case and I is thus not a model of T.
Case 2: y~x holds, but not x~y. Since this violates the reflexivity of ~, this cannot be the case and I is thus not a model of T.
Case 3: neither x~y nor y~x holds. By the definition of ~, it means that both x~y and y~x evaluate to false and I can thus not be a model of T.

Since we cannot find a case where I is a model of T, but is not a model of ϕ, (T entails ϕ) holds.

(ALTERNATIVE LÖSUNG) → genau wie in den Folien



A symbolic proof of the version shown above -- THIS IS NOT A VALID SOLUTION BECAUSE THE **SPEC DEMANDS A PROOF WITH SEMANTICAL MEANS**. However, if someone tries this as exercise, it may be helpful.

1.	Let x be an arbitrary element of an arbitrary domain U	assumption
2.	Pick an element y from U s.t. y = s(x)	assumption
3.	y ~ y	reflexivity of ~
4.	y ~ s(x)	equiv.repl.theorem, 3
5.	s(x) ~ y	equiv.repl.theorem, 3
6.	(s(x) ~ y) ∧ (y ~ s(x))	proof rule ∧, 4+5
7.	∃ y[(s(x) ~ y) ∧ (y ~ s(x)]	proof rule ∃, 2-6
8.	∀ x ∃ y[(s(x) ~ y) ∧ (y ~ s(x)]	proof rule ∀, 1-7

c) **Status: solved by student**

To show that the resolution calculus is sound, one has to show that the resolvent is a logical consequence of the resolved clauses. Let R be the resolvent and C1 and C2 the resolved clauses of a formula in CNF. We have to show that $C_1, C_2 \models R$, i.e. that $Mod(\{C_1, C_2\}) \subseteq$

Mod(R). Recall that R is defined as R = $C_1 \setminus \{cl\} \cup C_2 \setminus \{\neg cl\}$ where cl and ¬cl are dual literals.

Towards a contradiction, assume that there is a model which is a model of the premises, but not a model of the consequence, i.e. $I \in Mod(\{C1, C2\})$ and $I \notin Mod(R)$.

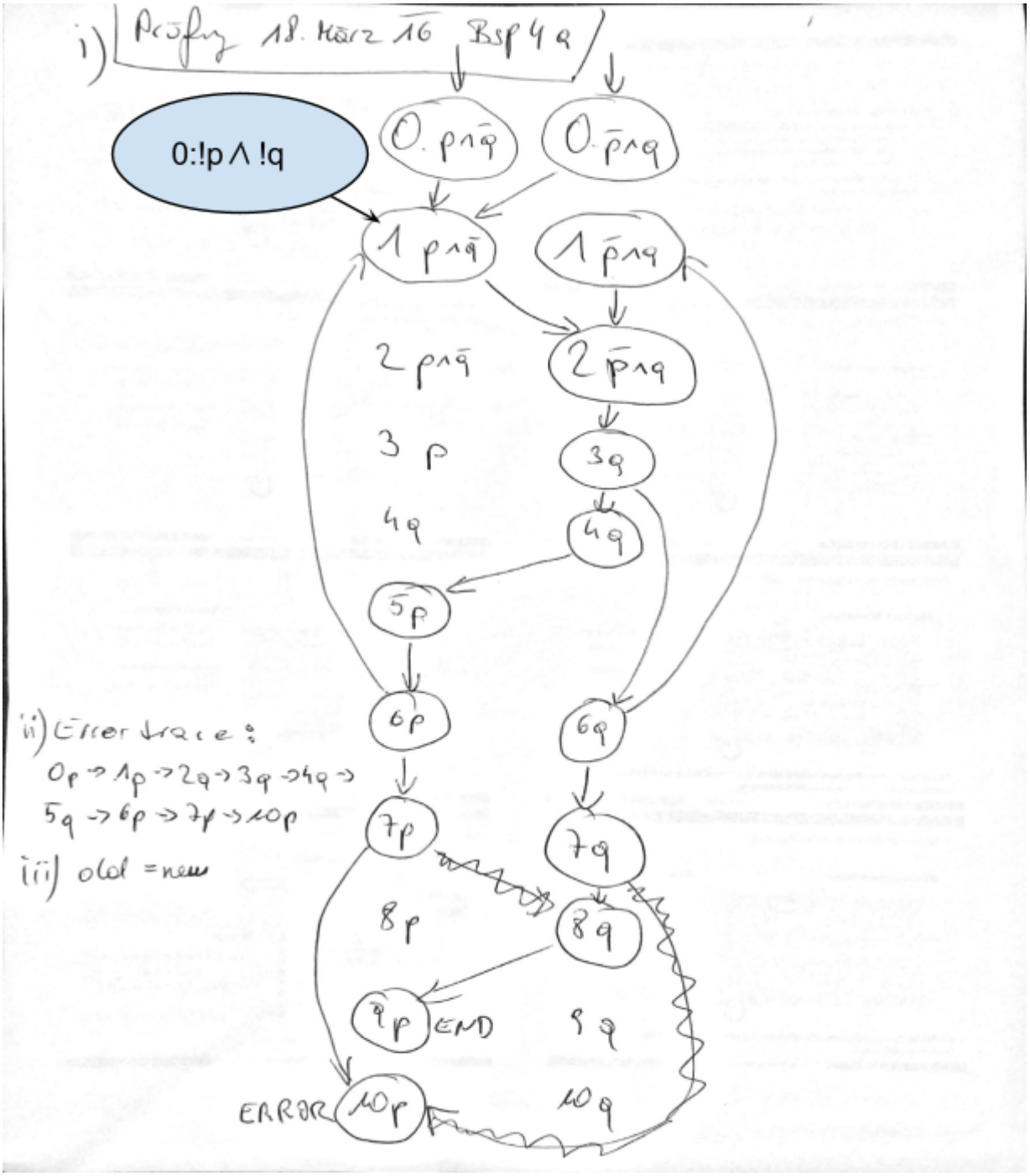
1.	$I \not\models R$	assumption
2.	$I \models C_1$	assumption, semantics of $\wedge$
3.	$I \models C_2$	assumption, semantics of $\wedge$
4.	$I \models cl$ or $I \models \neg cl$	lem
5.	Case 1: $I \models cl$	
6.	$I \models C_2 \setminus \{\neg cl\}$	semantics of $\vee$, 3
7.	$I \models C_1 \setminus \{cl\}$ or $I \models C_2 \setminus \{\neg cl\}$	proof rule $\vee$, 6
8.	Case 2: $I \models \neg cl$	
9.	$I \models C_1 \setminus \{cl\}$	semantics of $\vee$, 2
10.	$I \models C_1 \setminus \{cl\}$ or $I \models C_2 \setminus \{\neg cl\}$	proof rule $\vee$, 6
11.	$I \models C_1 \setminus \{cl\}$ or $I \models C_2 \setminus \{\neg cl\}$	use rule $\vee$, 5-10
12.	$I \models C_1 \setminus \{cl\} \vee C_2 \setminus \{\neg cl\}$	semantics of $\vee$, 11
13.	$I \models C_1 \setminus \{cl\} \cup C_2 \setminus \{\neg cl\}$	prop. $\vee$, 12
14.	$I \models R$	def R
15.	$I \models \bot$	contradiction, 1, 12

Block 3 - fmi162

??

Block 4 - fmi162

a) https://www.informatik-forum.at/showthread.php?113168-Exam-18-03-2016-4a



b)

F(a) -> LTL CTL* -> {S1,S3}
X(b) -> LTL CTL* -> {S1,S2,S4}
A[cUa] -> CTL CTL* -> {S1,S3}
EX(c) -> CTL, CTL* -> {S0,S3}

c) (by student)

i)

Does not hold!

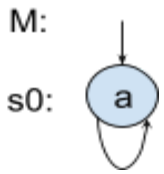
Counterexample:

Φ: b

Ψ: a

M |= E(Φ**U**Ψ)

M |≠ EF(Φ)



Therefore the left side holds, but the right side doesn't.

ii) TAUTOLOGY

Φ**U**Ψ is defined, that Φ has to hold until Ψ holds. So Ψ has to hold at some state on the path which is equvalent with FΨ. As we just showed, that Φ**U**Ψ -> FΨ holds, therefore also E(Φ**U**Ψ) -> EFΨ holds. E(Φ**U**Ψ) ≡ E(Φ**U**Ψ) trivially holds, and adding EFΨ to one side, does not interfere with the equivalence as Mod(E(Φ**U**Ψ))⊆Mod(EFΨ) or E(Φ**U**Ψ) -> EFΨ or E(Φ**U**Ψ) |= EFΨ.

iii) DOES NOT HOLD.

Suggestion of a counter-example:

ϕ = b

(s0, a) -> (s1, a) -> (s2, a) -> (s3, b)
\-> (s4, a) -> (s5, a)

~~Holds for **AXF(b)** ~~for state s0 but not for AXAF(b) for state s0.

AXF(b) does NOT hold for state s0! (Therefore not a valid counterexample!)

Because:

For all paths starting in S0:

That would be

1) S0,S1,S2,S3...

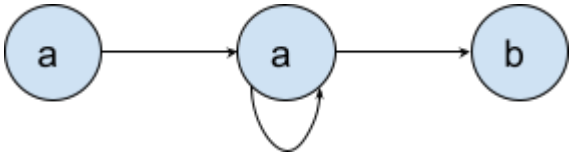
2) S0,S1,S4,S5...

it has to hold that from the next on some state b has to hold.

So beginning from S1 you have to reach on every path that was considered in the beginning that at some state b holds.

(

Suggestion of another Counterexample by X I:



Explanation why it's wrong:

AXF(b): So you select all paths, there is the path that stays forever in the second a therefore it does not hold.
AXAF(b): So you select all paths, and all paths from the next node include also the path that stays forever in a -> does not hold.
)

IT HOLDS:
Informal explanation: This holds as the additional A on the right side does not have any effect on the formula. AX already considers the next state on all possible paths and now considering again all paths does not add any new paths that have not been considered before.

Formal Proof:
I tried:
$M \vDash AXF\ \varphi$

1. Let s be an arbitrary state from S_0
   $\Leftrightarrow M, s \vDash AXF\ \varphi$ (by semantics of $\vDash$)
2. Let π be an arbitrary path starting in s
   $\Leftrightarrow M, \pi \vDash XF\ \varphi$ (by semantics of A)
3. $\Leftrightarrow M, \pi^1 \vDash F\ \varphi$ (by semantics of X)
4. We observe that the path formula $F\ \varphi$ must hold on every path originating from the starting state in π^1
5. $\Leftrightarrow M, \pi^1 \vDash AF\ \varphi$ (by the semantics of $\vDash$ and A and 4.)
6. $\Leftrightarrow M, \pi \vDash XAF\ \varphi$ (by semantics of X)
7. Because we chose π arbitrarily, and π originates from a starting state $s \in S_0$
   $\Leftrightarrow M, s \vDash AXAF\ \varphi$ (by semantics of A)
8. Because we chose s arbitrarily (in 1.)
   $\Leftrightarrow M \vDash AXAF\ \varphi$ (by semantics of A $\vDash$)

Exam 29.01.2016 (FMI.161.pdf)

http://www.logic.at/lvas/185291/angaben/fmi161.pdf

Block 1 - fmi161

Status: **solved by student**, **approved by tutor, approved by professor**

Let (Π', I') be an arbitrary instance of HALTING. That means Π' halts on I. We construct an instance of SOME-HALTS as follows:

- Π1 = Π2 = Π'

- I = I'

->

Assume (Π', I') is a positive instance of HALTING. For any input I', Π' halts on I'. By construction of SOME-HALTS we have that Π1 = Π2 = Π' and I = I'. Because Π' halts on any I' also Π1 and Π2 halt on I. Hence (Π1, Π2, I) is a positive instance of SOME-HALTS.

<-
Assume (Π1, Π2, I) is a positive instance of SOME-HALTS. That means Π1 halts on I or Π2 halts on I. By construction we know that Π' = Π1 = Π2 and I' = I. That means, because (Π1, Π2, I) is a positive instance and either Π1 or Π2 or both halt on I also Π' has to halt on I'. Hence (Π', I') is a positive instance of HALTING.

Block 2 - fmi161

a)

Status: student
substitution in function parameters is not allowed, only x = y -> f(x) = f(y)!
This procedure is called "substitution axiom for f or g" - fmi_sat3.pdf - S37

i)

1. $I \vDash f(x, y) = x \land g(x) \neq g(z) \land h(y) = g(x) \land f(f(x, y),y) = z$
2. $I \vDash f(x, y) = x$ (1, semantics of $\land$)
3. $I \vDash g(x) \neq g(z)$ (1, semantics of $\land$)
4. $I \vDash f(f(x, y),y) = z$ (1, semantics of $\land$)
5. $I \vDash y = y$ (reflexivity)
6. **$I \vDash f(f(x,y),y) = f(x,y)$ (2,5, substitution axiom for f)**
7. $I \vDash f(f(x,y),y) = x$ (2,6, transitivity)
8. $I \vDash z = x$ (4,7, transitivity)
9. $I \vDash x = z$ (8, symmetry)
10. $I \vDash g(x) = g(z)$ (9, substitution axiom for g)
11. $I \vDash \bot$ (3,10 contradiction)

i) alternate solution; status student

1. $I \vDash f(x, y) = x$ (1, semantics of $\land$)
2. $I \vDash g(x) \neq g(z)$ (1, semantics of $\land$)
3. $I \vDash f(f(x, y),y) = z$ (1, semantics of $\land$)
4. $I \vDash g(f(x,y)) \neq g(f(f(x,y),y))$ (1, 2, 3, substitution)
5. $I \vDash g(f(f(x,y),y)) \neq g(f(f(x,y),y))$ (1, 4, substitution)
6. $I \vDash \bot$ (5, contradiction)

ii)

1. $I \models x = y \wedge f(x) \neq f(y)$
2. $I \models x = y$ (1, semantics of $\wedge$)
3. $I \models f(x) \neq f(y)$ (1, semantics of $\wedge$)
4. $I \models f(x) = f(y)$ (2, substitution for f)
5. $I \models \bot$ (3,4 contradiction)

b)

added by student, approved by tutor (Lonsing)

- BC = base case, SC = step case
- BC: let formula be T, then the equivalent formula is also T
- BC: let formula be F, then the equivalent formula is ¬T
- BC: let formula be an atom, then the equivalent formula is also an atom
- SC: let ϕ1 and ϕ2 be formulas that can be translated to equivalent ones and ϕ be of the form ϕ1 & ϕ2 then the translated formula is also ϕ1 & ϕ2
- SC: let ϕ1 and ϕ2 be formulas that can be translated to equivalent ones and ϕ be of the form ϕ1 v ϕ2 then the translated formula is ¬(¬ϕ1 & ¬ϕ2) because ¬(¬ϕ1 & ¬ϕ2) is equivalent to (¬¬ϕ1 v ¬¬ϕ2) which is equivalent to (ϕ1 v ϕ2)
- SC: let ϕ be of the form ∀ϕ1 then the translated formula is also of the form ∀ ϕ1
- SC: let ϕ be of the form ∃ϕ1. Then the translated formula is ¬∀(¬ϕ1) because ¬∀(¬ϕ1) is equivalent to ∃(¬ ¬ϕ1) which is equivalent to ∃ ϕ1

Block 3 - fmi161

Status: student

Agree with the solution? 2 Yes vs. 1 No

(A) Status: Hint by professor.

Showing that as and as' are equal by lc rule.
First we show that from as' and lc we can infer as (i.e. as'->as)

$$\frac{\{F[v/e]\}\ v:=e\ \{\text{ex. } v' :(F[v/e][v/v'] \ \&\ v=e[v/v'])\},\ \text{ex. } v' :(F[v/e][v/v'] \ \&\ v=e[v/v']) \Rightarrow F}{\{F[v/e]\}\ v:=e\ \{F\}}\ lc$$

It remains to show that ex. v' :(F[v/e][v/v'] & v=e[v/v']) ⇒ F is valid.
ex. v' :(F[v/e][v/v'] & v=e[v/v']) ⇒ Fx
ex. v' :(F[v/e[v/v']] & v=e[v/v']) ⇒ F
ex. v' :(F[v/v] & v=e[v/v']) ⇒ F
ex. v' :(F & v=i-y**e[v/v']) ⇒ F

F & ex. v':(v=e[v/v']) ⇒ F (holds; *comment: because the conclusion is part of the premise*)

Secondly we need to show as -> as'
{F[v/e]} v:=e {F} , {F} ⇒ ex. v' :(F[v/e][v/v'] & v=e[v/v'])
——— lc
{F[v/e]} v:=e {ex. v' :(F[v/e][v/v'] & v=e[v/v'])}

It remains to show that F ⇒ ex. v' :(F[v/e][v/v'] & v=e[v/v']) is valid.
F ⇒ ex. v' :(F[v/e][v/v'] & v=e[v/v'])
F ⇒ ex. v' :(F[v/v] & v=e[v/v'])
F ⇒ ex. v' :(F & v=e[v/v'])
F ⇒ F & ex. v' :(v=e[v/v']) // from previous step since v' does not occur in F
F ⇒ ex. v' :(v=e[v/v']) (holds)

(A) **Status: Teacher Approved**

Show that $\{G[v/e]\}\, v := e\, \{G\}$ is equivalent to

$\{F\}\, v := e\, \{\exists v'.(F[v/v'] \wedge v = e[v/v'])\}$

1) $\{G[v/e]\}\, v := e\, \{\exists v'.(G[v/e][v/v'] \wedge v = e[v/v'])\}$

$\{\exists v'.(G[v/e[v/v']] \wedge v = e[v/v'])\}$

$\{\exists v'.(G[v/v] \wedge v = e[v/v'])\}$

$G \wedge \{\exists v'.(v = e[v/v'])\} \rightarrow G$

stronger → weaker
more restrictive → less restrictive

2) $\{F\}\, v := e\, \{\exists v'.(F[v/v'] \wedge v = e[v/v'])[v/e]\}$

$F \rightarrow \{\exists v'.(F[v/v'] \wedge e = e[v/v'])\}$

choose $v' = v$

$F \rightarrow F \wedge e = e$

$$\frac{\dfrac{F \rightarrow \exists v'.(\ldots)}{\{F\}\, v := e\, \{\exists v'.(F[v/v'] \wedge v = e[v/v'])\}}\,\text{1c} \qquad \dfrac{\{\exists v'.(\ldots)[v/e]\}\, v := e\, \{\exists v'.(\ldots)\}}{\{G[v/e]\}\, v := e\, \{G\}}\,\text{2c}}{\{G[v/e]\}\, v := e\, \{G\}}\,\text{M} \qquad G \rightarrow G\,\text{W}$$

2) $\{\underbrace{\exists v'.(F[v/v'] \wedge v = e[v/v'])}_{G}[v/e]\}$

1) $\{\underbrace{G[v/e]}_{F}\}\, v := e\, \{\exists v'.(G[v/e][v/v'] \wedge v = e[v/v'])\}$

(B) (not sure)

Is it possible to deduce with lc rule?? From the as axiom

F[v/e] -> F, {F} v := e {F & v = e}, F & v = e -> F
---------- (lc)
{F[v/e]} v := e {F}

F & v = e -> F trivially true
F[v/e] -> F … i think with the assumption from the axiom, that v does not occur in F and e, this is also true. When you replace v by e in F, but v does not occur in F, it will just be extended but F still holds. But can you assume this at this point?

(B) alternative solution based on discussion (forward reasoning)
We need to show that we can derive as'' from existing rules and valid premises.
By first applying the lc rule with as', we can get:

$$\frac{\{F\}\ v{:=}e\{\exists v'(F[v/v'] \wedge v{=}e[v/v']\} \quad \exists v'(\ldots) \Rightarrow F \wedge v{=}e}{\{F\}v{:=}e\{F \wedge v{=}e\}}\ lc$$

- It remains to show that the premises are valid:
- We know that $\{F\}v{:=}\ e\{\exists v'(\ldots)\}$ is an admissible rule.
- We know from the assumption, that v does not occur in F and not in e.
  Therefore, on the left side, only $\exists v'(F \wedge v = e)$ remains. since v' does not occur anywhere anymore, it leaves $F \wedge v = e$.
  $F \wedge v = e \Rightarrow F \wedge v = e$ is valid

(B) alternative solution based on discussion (backward reasoning)
We need to show that we can derive as'' from existing rules and valid premises.
By first applying the lc rule with as, we can get:

$$\frac{\{F\} \Rightarrow \{(F \wedge v{=}e)[v/e]\}, \{(F \wedge v{=}e)[v/e]\}v{:=}e\{F \wedge v{=}e\}}{\{F\}v{:=}e\{F \wedge v{=}e\}}\ lc$$

- F -> (F ^ v=e)[v/e] = F -> (F ^ e=e) = F -> F (holds)

(C)

Find a counterexample:
Counterexample: {x=0}x:=x+1{x=1}
Since x is in the precondition, as'' can not be applied. Lc does not help either:

```
x=0→F          {F}x:=x+1{G}        G→x=1
-------------------------------------(lc)
             {x=0}x:=x+1{x=1}
```

```
However, with (as) it can be derived:

     F→G[x/x+1] {G[x/x+1]}x:=x+1{G}
     ------------------------------(as)
x=0→F          {F}x:=x+1{G}        G→x=1
-------------------------------------(lc)
             {x=0}x:=x+1{x=1}
```

```
With F: x=0 and G: x=1
```

Complete: every true correctness assertion is derivable.
Since the shown correctness assertion is true, but not derivable with as'', the Hoare calculus is not complete without as and as'.

(D) (not sure)
In as'' it is important that v does not occur in F and e, because as we have seen in (C) the postcondition will be invalid if that happens.
In as' it basically means that there is an original value, s.t. the precondition was valid with this original value and the assignment basically means replacing the original value with the formula of the assignment. But as we can choose v' (because of the exists) we can choose it s.t. it will not occur in F or e and so it's not that important here. *(i am really not sure about the second part about as')*

D alternative, status student:
I would argue, that from as' you can derive as'' (if v' does not occur in F or e) and you have covered both cases (if v' does or does not occur in F or e). With as'' you only have covered the case where v' does not occur in F or e.

Block 4 - fmi161

a) approved by professor

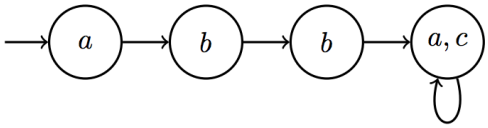
4.) Model Checking

Let $M = (S, I, R, L)$ be a Kripke structure over a set of propositional symbols AP.

Let $AP' \subseteq AP$ be a subset of AP. We define $M' = (S', I', R', L')$ as follows:

- $S' = S$, $I' = I$, $R' = R$, and
- $L'(s) = L(s) \cap AP'$, where $s \in S$.

(a) Consider the concrete instance M over $AP = \{a, b, c\}$ below. Draw M' over $AP' = \{a, b\}$ according to the definition above.



(1 point)

-> (a) -> (b)->(b)->(a)
U // reflexive arrow

b) solved by student, approved by tutor

(b) Given any M, M' and AP, AP' according to the definitions above, prove that for any ACTL formula φ over propositions from AP' the following holds:

$$M \models \varphi \text{ if and only if } M' \models \varphi$$

Hint: Use the semantics of ACTL. You can either use induction on the structure of the formula (structural induction) or induction on the formula length.
We recall the definition of ACTL formulae over AP:

- $p \in AP$ is an ACTL formula,
- if φ and ψ are ACTL formulae, then $\neg\varphi$, $\varphi \wedge \psi$, $\mathbf{AX}\,\varphi$, and $\mathbf{A}\,[\varphi\,\mathbf{U}\,\psi]$ are ACTL formulae.

(6 points)

2 4b

$\iff$ means iff.

2.1 Induction Hypothesis

Let $\neg\phi, \phi \wedge \psi, AX\phi$ and $A[\phi U \psi]$ be ACTL formulae over AP'. Then $M \models \phi \iff M' \models \phi$ and $M \models \psi \iff M' \models \psi$.

2.2 Induction base

Let ϕ be an ACTL formula consisting just of $p \in AP'$. Remind that $AP' \subseteq AP$. Therefore, also $p \in AP$. According to the above definitions it must follow that $M' \models \phi \iff M \models phi$.

Case 1: $\phi = p, p \in AP'$
Assume $M \models p$.
$M \models p \iff$
$p \in L(s)$ where s in I (by semantics of ACTL) $\iff$
$p \in L'(s)$ where s in I' (by definition of L') $\iff$
$M' \models p$ (by semantics of ACTL).

2.3 Induction Step

Case 2: $\phi = \phi 1 \wedge \phi 2$
Let $\phi 1$ and $\phi 2$ be some arbitrary ACTL formulae according to the ind. hyp. Because $M, \pi \models \phi 1 \iff M, \pi \models \phi 2$. By induction assumption $M, \pi \models \phi 1 \wedge \phi 2$ hence $M, \pi \models \phi$

Case 3: $\phi = \phi 1 \vee \phi 2$ is similar to the $\wedge$ - case

Case 4: $\phi = \neg\psi$
Assume $M \models \neg\psi$.
$M \models \neg\psi \iff$
$M \not\models \psi \iff$
$M' \not\models \psi \iff$ by ind. hyp.(This is important as it isn't obvious!)
$M' \models \neg\psi$

Case 5: $\phi = A[\phi_1 U \phi_2]$
Assume $M \models \phi = A[\phi_1 U \phi_2] \iff$
for initial state $s_0 \in I$ there exists a path $\pi = s_0, s_1, ...$ s.t. $M, \pi \models \phi_1$ U $\phi_2 \iff$
for initial state $s_0 \in I \exists k \geq 0$ s.t. $M, \pi^k \models \phi_2$ and $0 \leq i < k$ s.t. $M, s_i \models \phi_1$
$\iff$ for initial state $s_0 \in I$ there exists a path $\pi = s_0, s_1, ...$ s.t. $M', \pi \models \phi_1$ U $\phi_2 \iff$
for initial state $s_0 \in I \exists k \geq 0$ s.t. $M', \pi^k \models \phi_2$ and $0 \leq i < k$ s.t. $M', s_i \models \phi_1$

c) approved by professor
Initial state is a.

(“”)	(a)	<->	(b)	(a,b)
	U		U	// reflexive arrow

d) student
Basically, the definition designs M^ in a way, s.t. M^ simulates M (M ≤ M’). As ACTL is a subset of ACTL* we could therefore directly apply the hint and get “If M^ |= ϕ, then M |= ϕ”.

Seems to be too easy that way, and probably too informal.

Correction: (I don’t have time for too many details, but this sketch should help)
This simulation is valid for any M’ that was created from M according to the specified rules:
H = { (s,s’) for s ∈ S and s’ ∈ S’ | L(s)=L(s’) }

H is a valid simulation relation because:
TODO: Elaborate why it follows from construction of M’ and H that:

1. predicates are equal
2. successors match
3. initial states are equal (not needed for the simulation relation but to show that H is a simulation from M to M’)

Exam 04.12.2015 (FMI.156.pdf)

http://www.logic.at/lvas/185291/angaben/fmi156.pdf

Block 1

1.) Consider the following problem:

PROB

INSTANCE: A triple (P, K, m), where

- $P = \{p_1, \ldots, p_n\}$ is a set of elements (intuitively, a set of persons),
- $K \subseteq P \times P$ is a binary relation over P (intuitively, $(p, p') \in K$ means that p knows p'),
- m is an integer (intuitively, the number of seats at a dinner table).

QUESTION: Is it possible to sit m persons at the dinner table so that none of the persons knows another person at the table? That is, does there exist $S \subseteq P$ such that $|S| = m$ and $(p, p') \notin K$ for all $p \in S$ and $p' \in S$ with $p \neq p'$?

Provide a polynomial time reduction from **PROB** to **CLIQUE**, and prove the correctness of your reduction.

Recall that **CLIQUE** is defined as follows:

CLIQUE

INSTANCE: A pair (G, k), where $G = (V, E)$ is an undirected graph and k is an integer.

QUESTION: Does there exist a set $C \subseteq V$ such that (i) $|C| \geq k$ and (ii) for all $v, v' \in C$ with $v \neq v'$ we have $[v, v'] \in E$.

(15 points)

added by student

let x=(P,K,m) be an arbitrary instance of PROB. We construct a reduction R that maps instances of PROB to instances of CLIQUE by constructing a pair (G,k) as follows:
let k=m and G be a graph (V,E) with V=P and (pi,pj) in E ⇔ (pi,pj) not in K (i.e. E is the complement of K)

We have to show that x is a positive instance of PROB ⇔ R(x) is a positive instance of clique

“=>”: assuming that x is a positive instance of PROB. This means that there exists a subset S of P with |S|=m and for every pair (pi,pj) in S there does not exist a pair in K. If there is no pair in K for those elements, there is (by construction of R(x)) an element for each pair (pi,pj) of S also in E. Since |S|=k, by construction |E|≥k=m holds which means that R(x) is a positive instance of CLIQUE

“<=”: assuming that R(x) is a positive instance of CLIQUE. This means there exists a subset C of V, s.t. |C|≥k and for every pair (vi,vj) in C, (vi,vj) is also element of E. By construction of R(x) this means that for every pair (vi,vj) in E there DOES NOT exist a pair (pi,pj) in K. Since |C|≥k there are at least k=m elements in P (let’s denote it as a subset S) where each pair (pi,pj) is not element of K and therefore x is also a positive instance of PROB.

In the lecture there was a reduction from INDEPENDENT SET to CLIQUE which is basically the same.

Block 2

2.) (a) Suppose `a, b, c` are (unsigned) integers in a programming language like C and `a` ≤ `b`. Which problem can occur with a C statement `c=(a+b)/2`? What is a simple solution to the problem? **(3 points)**

a) Solution1 (from the solutions from WS_14):

Exercise 11 Bonus: Integer Overflow

We have argued in the slides of the third SAT lecture (`fminf_sat3.pdf`, page 28) that most of the binary search algorithms are broken. Implement the algorithm in a correct way which avoids that the program crashes for large arrays.

Solution:

The problem is line 6 where $(l + h)$ already can lead to an integer overflow:

$$m = (l + h)/2;$$

One possible solution: replace line 6 by

$$m = l + ((h - l)/2);$$

Solution2 (manually considering the lowest bits of a and b):
a+b can lead to an integer overflow.
Solution: c = (a/2) + (b/2) + (a&b&1)

(see:

b) first step: converting ϕ to ψ^e1 which is in NNF:ψ^e1: a=b & a!=c & (a!=d v e=f v g=h) & g=i & h=i & (b=c v g!=i v i=j). this formula is basically the same as from 16th october.

And then ?

Block 3

Status: approved by professor

{Inv & e} p {Inv}
——————————————————————— while
(Inv & e)->Inv {Inv}while...{Inv & !e}
————————————————lc
{Inv & e}while e do ... od {Inv & !e}

(Inv & e)->Inv is a tautology
The solution was to use the fact that the precondition can trivially be strengthened

(B)
We have to show that not everything is derivable (using a logical consequence) with the modified while loop. So we have to find a counterexample:
We choose the invariant e ≤ x and the program

{e = 1} (Program precondition)
x = 1;
{e = 1 & x = 1} as ↓ (Assignment with forward reasoning of the form F & !e)
{e ≤ x (Invariant) & e < 2*x (our modified precondition)}
while e < 2*x do
{... rest does not matter for what we show …}
x = x + 1
od
{true} (don’t care about the postcondition for what we show)

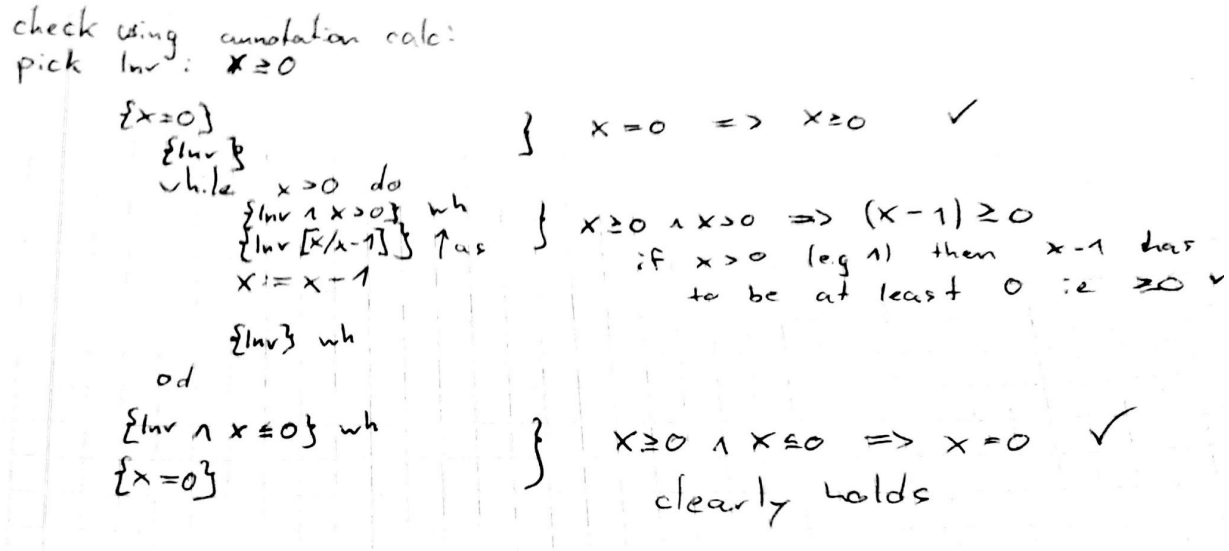
So we see the logical consequence after the assignment x = 1; which is as follows:
e = 1 & x = 1 => e ≤ x & e < 2*x
but obviously, when e = 1 and x = 1, e cannot be < 2*x as e = x. So in the modified variant this is not complete anymore.

(B) Alternative

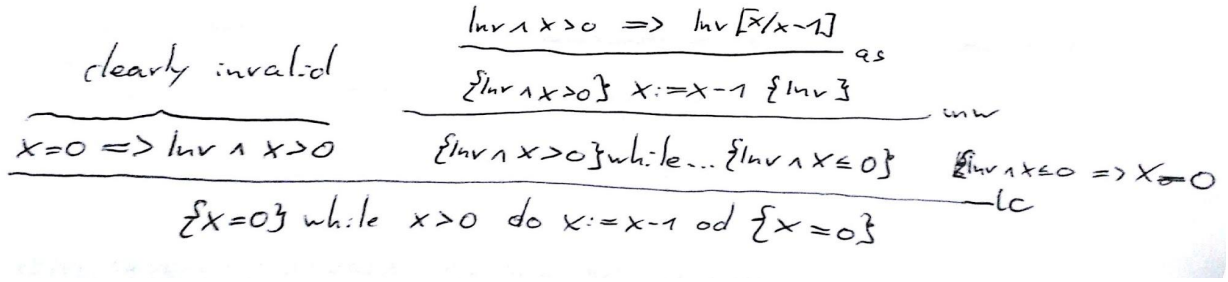
Consider the counter-example:$\{x = 0\} while\ x > 0\ do\ x := x - 1\ od\ \{x = 0\}$

Use $Inv = x \geq 0$

It is easy to show that this is a correct assertion:



Now we first apply lc to get it into the form s.t. we can apply mw. It's easy to see that this immediately leads to an invalid implication, although the rule applications are correct.



I would probably argue in more detail during the exam, but the intuition seems right.

Block 4

added by student

a) K1≤K2 does not hold because regarding s2 there must exist an a-state in K2 that has a b-successor and a c successor. K1≥K2 holds with H={(s1',s1),(s2',s2),(s3',s3),(s4',s4),(s5',s6),(s6',s6)}. K1=K2 does not hold again because of s2

b) G(b) LTL + CTL*: s0, s1, s2, s3
X(c) LTL + CTL*: s0, s4
AX(a & b & c) ~~LTL~~ +CTL + CTL*: none
EF(a & b & c) CTL + CTL*: s4,s2,s3

c) None

(C)

Status: Solved by student, comments welcome

Basically I just unfold the semantics of the operators:
Let φ be a state formula, and ψ be a path formula. The semantics of $\models$over a Kripke structure M are defined as:

- $M \models \varphi \Leftrightarrow \forall s \in S_0 \colon M, s \models \varphi$
- $M \models \psi \Leftrightarrow \forall s \in S_0, \forall \pi(\pi^0 = s)\ M, \pi \models \psi$ ($\pi^0 = s$, states that the path starts in s)

(i)d

$M \models AX\ AF\ p$

1. Let s be an arbitrary state from S_0
   $\Leftrightarrow M, s \models AX\ AF\ p$ (by semantics of $\models$)
2. Let π be an arbitrary path starting in s
   $\Leftrightarrow M, \pi \models X\ AF\ p$ (by semantics of A)
3. $\Leftrightarrow M, \pi^1 \models AF\ p$ (by semantics of X)
4. Because we know that $F\ p$ holds for all computation paths originating from the starting state of π^1 we can safely assume that
5. $\Rightarrow M, \pi \models XF\ p$ (by semantics of X and 4.)
6. Because we chose π arbitrarily, and every π originates from a state in S_0
   $\Rightarrow M \models XF\ p$ (by semantics of $\models$)

(ii)

$M \models AF\ AF\ p$

1. Let s be an arbitrary state from S_0
   $\Leftrightarrow M, s \models AF\ AF\ p$ (by semantics of $\models$)
2. Let π be an arbitrary path starting in s
   $\Leftrightarrow M, \pi \models F\ AF\ p$ (by semantics of A)
3. $\Leftrightarrow \exists k \geq 0,\ M, \pi^k \models AF\ p$ (by semantics of F)
4. Let μ be an arbitrary path starting in the starting state of π^k
   $\Leftrightarrow M, \mu \models F\ p$ (by semantics of A)
5. $\Leftrightarrow \exists j \geq 0, \mu^j \models p$ (by semantics of F)

6. We observe that, any such μ^j must lie on a path that originates from $s \in S_0$. This means that the existence of a state that lies on a path originating from $s \in S_0$, where p holds, is guaranteed (by $\exists$). Because we chose s and π arbitrarily, this holds for all paths originating from $s \in S_0$. We can therefore safely assume
7. $\Rightarrow \exists k \geq 0\ M, \pi^k \vDash p$ (namely where $\pi^k = \mu^j$)
8. $\Leftrightarrow M, \pi \vDash F\ p$ (by semantics of F)
9. $\Leftrightarrow M \vDash F\ p$ (by semantics of $\vDash$)

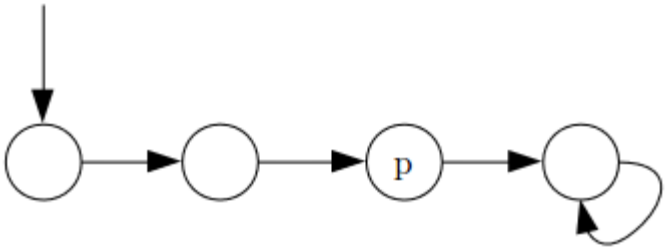
(iii)

By first checking the semantics of both statements, we can directly derive a structure from it.

π is chosen arbitrarily, starting from any $s \in S_0$

1. $M \vDash XFX\ p \Leftrightarrow M, \pi \vDash XFX\ p \Leftrightarrow M, \pi^1 \vDash FX\ p \Leftrightarrow \exists k \geq 1\ M, \pi^k \vDash X\ p \Leftrightarrow M, \pi^{k+1} \vDash p$
   So the statement holds e.g. for $k = 1\ \ M, \pi^2 \vDash p$
2. $M \vDash XXXF\ p \Leftrightarrow M, \pi^3 \vDash F\ p\ \Leftrightarrow \exists k \geq 3\ M, \pi^k \vDash p$
   In contrast to the one before, this statement can only hold for $k \geq 3$

This is basically enough proof to show that $M \vDash XFX\ p$ does not imply $M \vDash XXXF\ p$, but a Kripke structure could therefore look like this:



Exam 16.10.2015 (FMI.155.pdf)

http://www.logic.at/lvas/185291/angaben/fmi155.pdf

kBlock 1

DIFFERENT (DIFF)

INSTANCE: A pair (Π_1, Π_2) of programs that take one string as input and return a string as output. It is guaranteed that Π_1 and Π_2 terminate on any input string.

QUESTION: Does there exist a string I on which the programs Π_1 and Π_2 produce a different output, i.e. $\Pi_1(I) \neq \Pi_2(I)$?

Note: we consider only strings that are built from symbols 0 and 1.

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

Correction:

To show that DIFF is semi-decidable, we construct a semi-decision procedure as follows:

```
Boolean decide(Program Π1, Program Π2) {
      for(int i=0; true, i++) {
            s = getStringByNumber(i);
            String out1 = Π1(s);
            String out2 = Π2(s);
            If (out1 != out2) {
                  return True;
            }
      }
}
```

Since the set of all possible strings is countably infinite, it is possible to assign a unique number to sed in parameter i. decide is a valid semi-decision procedure:

Case1: Let (Π_1, Π_2) be a positive instance of DIFF. From the definition of DIFF it follows that there exists a string I so that Π_1(I) != Π_2(I). Since it is guaranteed that Π_1 and Π_2 terminate on any input and the for loop in the decision function iterates over all possible strings, it will eventually find this I and return True.

Case2: Let (Π_1, Π_2) be a negative instance of DIFF. From the definition of DIFF it follows that there exists no string I so that Π_1(I) != Π_2(I). In this case the function decide will iterate all

Strings without finding any String for which $\Pi_1(I)$!= $\Pi_2(I)$. Since the number of Strings to test is infinite, decide will not terminate. This is acceptable behavior for a semi-decision procedure.

Status: student

Block 2

(a) When is a Theory complete, name a complete one and an incomplete one

A theory is complete, if every closed formula under the theory is either true or false

- complete example: Pressburger arithmetic, Peano arithmetic
- incomplete example: Group Theory

Status: student

(b)

Simplification: Replacement of pure literals that are not part of simple contradictory cycles with true. These are e=f and a!=d

ψ^e1: a=b & a!=c & (true v true v g=h) & g=i & h=j & (b=c v g!=i v i=j)

ψ^e1 can be simplified to ψ^e2: a=b & a!=c & g=i & h=j & (b=c v g!=i v i=j)

here i=j, j=h and i!=g/i=g are not part of simple contradictory cycles, i!=g and i=g cannot be removed since they are not pure literals, i=j and j=h however can be removed:

ψ^e3: a=b & a!=c & g=i & true & (b=c v g!=i v true) which can be simplified to

ψ^e4: a=b & a!=c & g=i. However in the resulting graph, ALL clauses are neither pure literals nor part of simple contradictory cycles and ψ^e5 is therefore TRUE

What about Bt?

Block 3

wp (y:=x; if …, ($y = -2x_0$))

wp (y:=x; ($y \geq 0$ & wp (P, $y = -2x_0$)) || $y < 0$ & wp(y:=y-3, ($y=-2x_0$)), ($y=-2x_0$)) → wp of **if**

Now the wp of P:

First get F_0 then F_1 and F_2 until you can figure out how a general formula for F_i should look like

In this example we got

$F_i = x - (i-1) > 0 \ \& \ (x - i) =< 0 \ \& \ (y-3i) = -2x_0$

Replace back:

wp (y:=x; (y ≥ 0 & (x - (i-1) > 0 & (x - i) =< 0 & (y-3i) = $-2x_0$) || y <0 & wp(y:=y-3, (y=$-2x_0$)), (y=$-2x_0$))

Now for the y:=y-3 part we have y-3 = $-2x_0$, so:

wp (y:=x; (y ≥ 0 & (x - (i-1) > 0 & (x - i) =< 0 & (y-3i) = $-2x_0$) || y <0 & y-3 = $-2x_0$, (y=$-2x_0$))

Now we have the sequential composition, first we need the wp of the second part, then the y:=x part:

wp (y:=x; wp((y ≥ 0 & (x - (i-1) > 0 & (x - i) =< 0 & (y-3i) = $-2x_0$) || y <0 & y-3 = $-2x_0$, (y=$-2x_0$), (y=$-2x_0$))

At the end you get a logical formula, and you have to prove that the precondition implies it, like:

$x=x_0 \rightarrow$ *formula after solving the wp(...)-s*

Block 3 alternate version:

Def: $\{F\}\ p\ \{G\}$ totally correct $\Leftrightarrow F \Rightarrow wp(p,G)$

$wp(p,G) = wp(y:=x;\ if\ldots,\ y=-2x_0)$ // sequential composition: $wp(p;q,G) = wp(p, wp(q,G))$

$= wp(y:=x, \underbrace{wp(if\ldots, y=-2x_0)}_{}))$ // conditional rule see exam-sheet

$(y>0 \wedge \underbrace{wp(P, y=2x_0)}) \vee (y<0 \wedge wp(y:=y-3x, y=-2x_0))$

$wp(while\ldots, y=-2x_0)$ // while ~~rule~~ rule see exam-sheet

$F_0: x \leq 0 \wedge y = -2x_0$

$F_1: x>0 \wedge wp(x:=x-1, y:=y-3, y=-2x_0 \wedge x\leq 0)$
$wp(x:=x-1, \underbrace{wp(y:=y-3, y=-2x_0 \wedge x\leq 0)}_{})$ // ass-rule: G[y/e]

$\underbrace{wp(x\leq 0 \wedge y-3 = -2x_0)}_{}$

$x>0 \wedge x-1 \leq 0 \wedge y-3 = -2x_0$
$= \underbrace{x>0 \wedge x \leq 1}_{} \wedge y = -2x_0+3$

$= \underline{x=1 \wedge y=-2x_0+3}$

$F_2: x>0 \wedge wp(x:=x-1, y:=y-3, x=1 \wedge y=-2x_0+3)$
$wp(x:=x-1, \underbrace{wp(y:=y-3, x=1 \wedge y=-2x_0+3)}_{})$

$\underbrace{wp(x:=x-1, x=1 \wedge y-3=-2x_0+3)}_{}$
$x>0 \wedge x-1=1 \wedge y=-2x_0+6$
$\underline{x=2 \wedge y=-2x_0+6}$ // x=... is more restrictive than $x>0$ ⇒ drop it

$F_i: x=i \wedge y=-2x_0+3i$ (GUESS)

$F_{i+1}: x>0 \wedge wp(x:=x-1, y:=y-3, x=i \wedge y=-2x_0+3i)$
$wp(x:=x-1, \underbrace{wp(y:=y-3, x=i \wedge y=-2x_0+3i)}_{})$

$\underbrace{wp(x:=x-1, x=i \wedge y-3=-2x_0+3i)}_{}$

$x>0 \wedge x-1=i \wedge y-3 = -2x_0+3i$
$= x=i+1 \wedge y=-2x_0+3i+3$
$= x=i+1 \wedge y=-2x_0+3(i+1)$ (PROOF!)

$\exists i (i\geq 0 \wedge \underline{x=i} \wedge y=-2x_0+3i)$
$= \exists i (x\geq 0 \wedge x=i \wedge y=-2x_0+3x)$
$= y=-2x_0+3x \wedge x\geq 0 \wedge \underbrace{\exists i (x=i)}_{}$
$= y=-2x_0+3x \wedge x\geq 0$ TRUE always

$= wp(y:=x, wp(if\ldots, y=-2x_0))$
$= wp(y:=x, y>0 \wedge wp(P, y=-2x_0) \vee (y<0 \wedge wp(y:=y-3x, y=-2x_0)))$
$= wp(y:=x, y>0 \wedge y=-2x_0+3x \wedge x>0) \vee (y<0 \wedge y=-2x_0+3x))$

$= x>0 \wedge (x=-2x_0+3x \wedge x>0) \vee (x<0) \wedge (x=-2x_0+3x)$
$= (x=-2x_0+3x) \wedge \underbrace{x>0 \vee (x<0)}_{}$ TRUE always
$= x=-2x_0+3x$

Show: $F \Rightarrow wp(p,G)$ $\quad x=x_0 \Rightarrow \underbrace{-2x_0+3x}_{x_0=x} \Leftrightarrow x=x_0 \Rightarrow x=x_0$ ✓

I verified the alternative solution -- here a typesetted variant:

$wp(y := x; \text{if } y \geq 0 \text{ then P else } y := y - 3x \text{ fi}, y = -2x_0) =$
$= wp(y := x; wp(\text{if } y \geq 0 \text{ then P else } y := y - 3x \text{ fi}, y = -2x_0)) =$
$= wp(y := x; (y \geq 0) \wedge wp(P, y = -2x_0) \vee (y < 0) \wedge wp(y := y - 3x \text{ fi}, y = -2x_0)) =$
$= wp(y := x; (y \geq 0) \wedge wp(P, y = -2x_0) \vee (y < 0) \wedge (y-3x = -2x_0))$

$wp(P, y = -2x_0) = wp(\text{while } x > 0 \text{ do } x := x-1; y := y-3, y = -2x_0) = \exists i(i \geq 0 \wedge F_i)$

- F_0: $\neg(x > 0) \wedge y = -2x_0$
  $\Leftrightarrow x \leq 0 \wedge y = -2x_0$

- F_1: $(x > 0) \wedge wp(x := x-1; y := y-3, F_0)$
  $\Leftrightarrow x > 0 \wedge F_0[y/y-3][x/x-1]$
  $\Leftrightarrow x > 0 \wedge x-1 \leq 0 \wedge y-3 = -2x_0$
  $\Leftrightarrow x=1 \wedge y = 3-2x_0$

  since $x > 0$ and $x-1 \leq 0$ is equivalent to $x \leq 1$, we may conclude that $x = 1$.

- F_2: $(x > 0) \wedge wp(x := x-1; y := y-3, F_1)$
  $\Leftrightarrow x > 0 \wedge F_1[y/y-3][x/x-1]$
  $\Leftrightarrow x-1=1 \wedge y-3 = 3-2x_0$
  $\Leftrightarrow x=2 \wedge y = 6-2x_0$

- Guess F_i: $x = i \wedge y = 3i-2x_0$

- Proof F_i by induction: $F_{i+1} = (x > 0) \wedge wp(x := x-1; y := y-3, F_i)$
  $\Leftrightarrow x > 0 \wedge F_i[y/y-3][x/x-1]$
  $\Leftrightarrow x > 0 \wedge x-1 = i \wedge y-3 = 3i-2x_0$
  $\Leftrightarrow x > 0 \wedge x = i+1 \wedge y = 3i+3-2x_0$
  $\Leftrightarrow x = i+1 \wedge y = 3(i+1)-2x_0$

  since $i \geq 0$ and $x = i+1$, $x > 0$ is redundant.

$wp(P, y = -2x_0) = \exists i(i \geq 0 \wedge F_i) = \exists i(i \geq 0 \wedge x = i \wedge y = 3i-2x_0)$

Since $i \geq 0 \wedge x = i$, we know that $x \geq 0$, which allows us to eliminate the existential quantifier:

$\exists i(i \geq 0 \wedge x = i \wedge y = 3i-2x_0) \Rightarrow x \geq 0 \wedge \exists i(i \geq 0 \wedge y = 3i-2x_0) \Rightarrow$ **$x \geq 0 \wedge y = 3x-2x_0$**

= wp(y := x; (y ≥ 0) ∧ (x ≥ 0) ∧ (y = 3x-2x_0) ∨ (y < 0) ∧ (y-3x = -2x_0)) =
= (x ≥ 0) ∧ (x ≥ 0) ∧ (x = 3x-2x_0) ∨ (x < 0) ∧ (x-3x = -2x_0) =
= (x ≥ 0) ∧ (x = x_0) ∨ (x < 0) ∧ (x = x_0)
It remains to show that the precondition implies the WP we calculated:
(x ≥ 0) ⇒ (x ≥ 0) ∧ (x = x_0) ∨ (x < 0) ∧ (x = x_0)

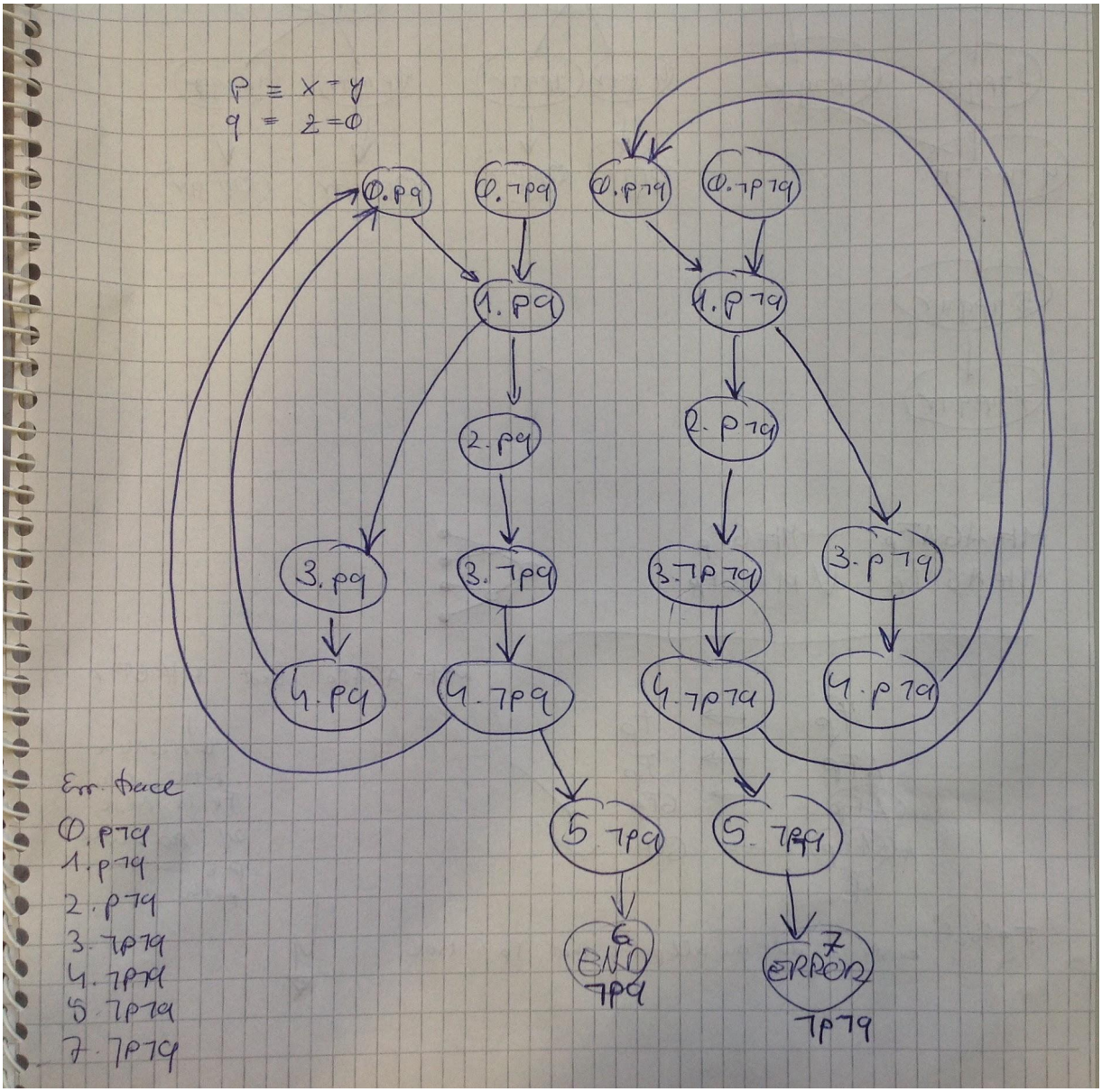
Since (x ≥ 0) is true (premise), and (x ≥ 0) ∨ (x < 0) is always true, the implication is true as well.

Block 4

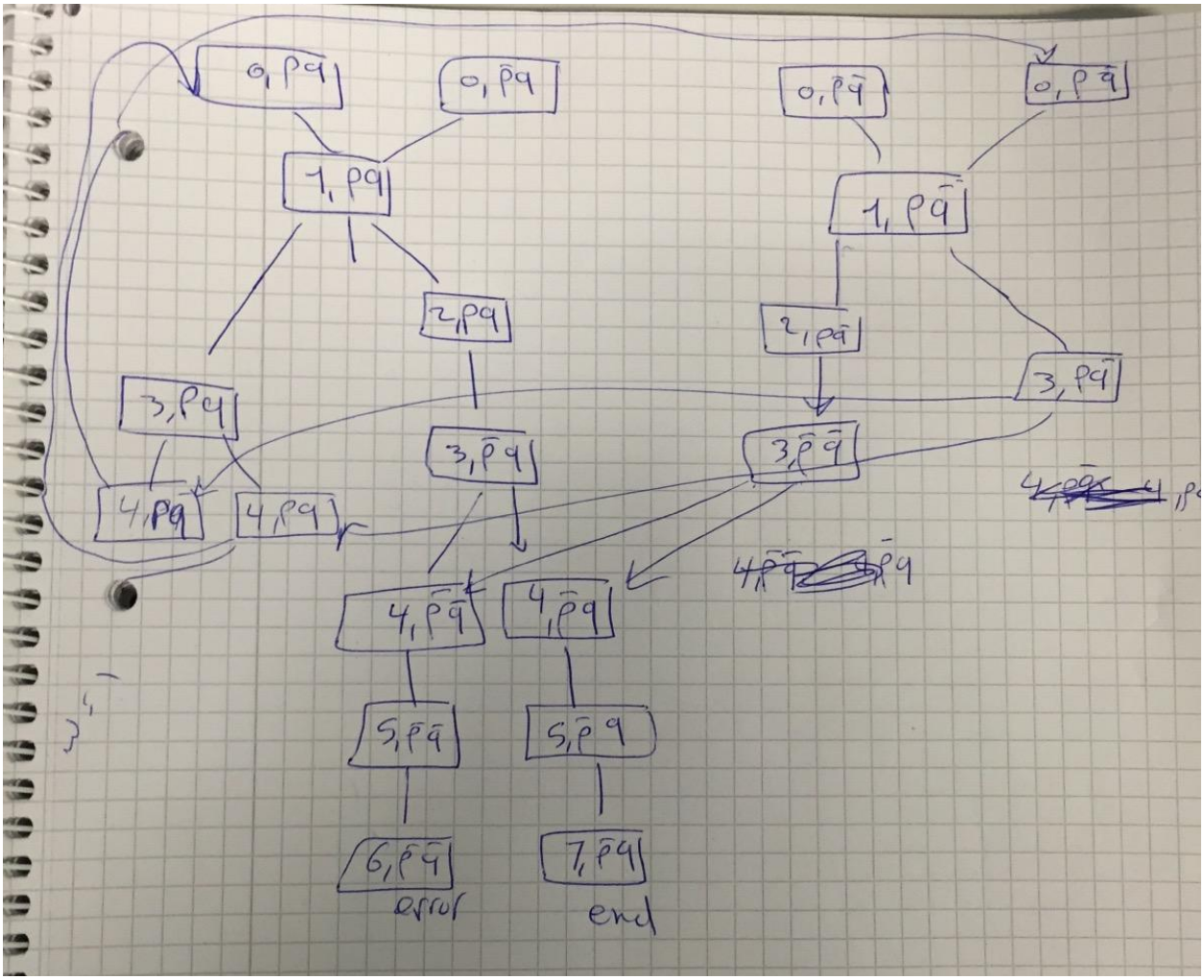
by student

a)

i.



We have another Solution



ii. an example error trace would be

0 p !q

1 p !q

2 p !q

3 !p !q

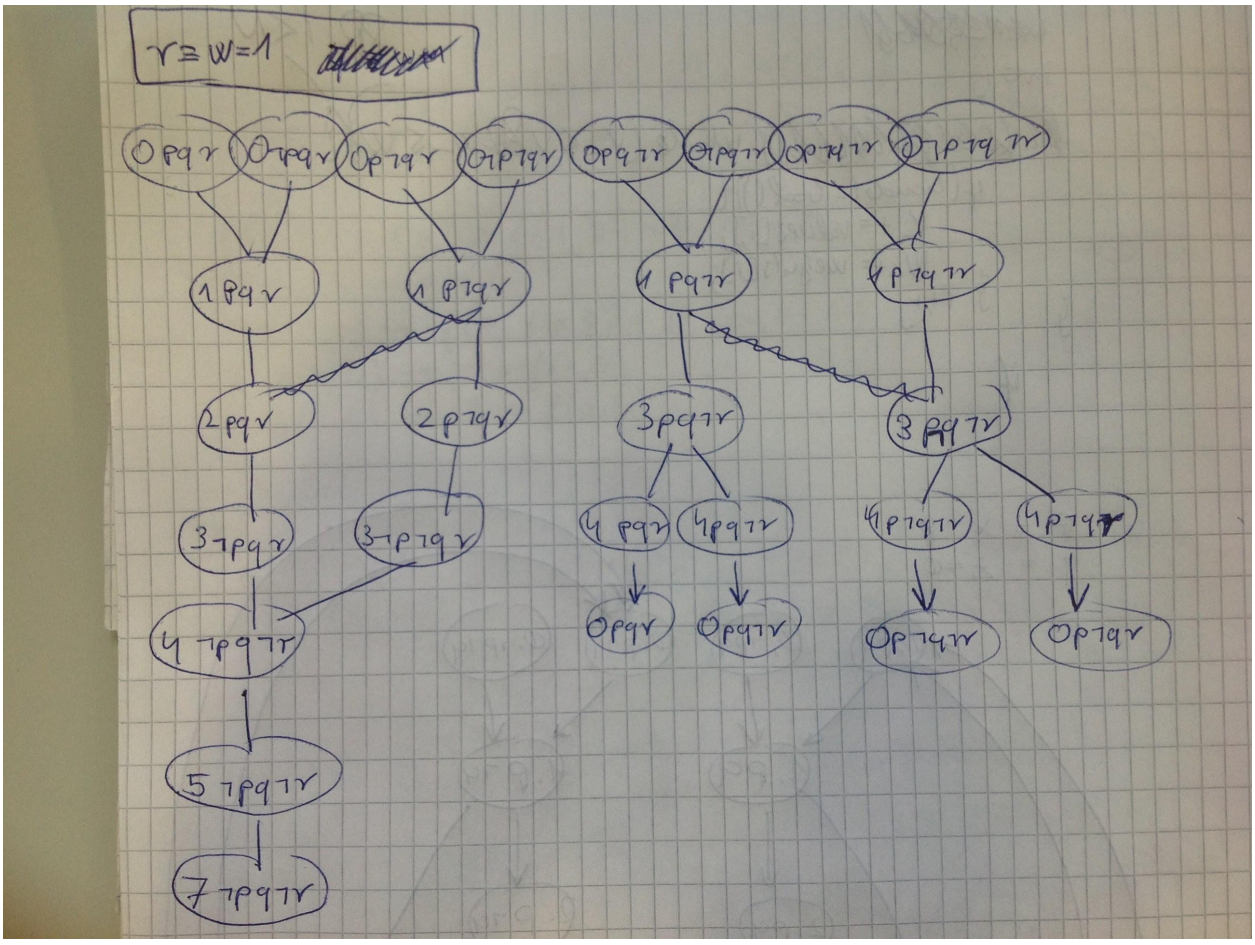
4 !p !q

5 !p !q

6 !p !q (this is the error state)

iii. A predicate would be y: w = 1.

You don’t have to draw the new refined LTS, but we just couldn’t resist:



b)

	CTL	LTL	CTL*	States
G(c)	X	OK	OK	s2, s4
X(a & c)	X	OK	OK	s4
AG (a)	OK	~~OK~~	OK	<nowhere>
EF (a)	OK	X	OK	s0,s1,s2,s3,s4

c)
AGAFp ⇒ GFp

i. Assume AGAFp ⇒ NOT(GFp)
By semantics we know that on an arbitrary path Π = s0, s1, … Is a state reachable from where on Fp true is.
We also know by semantics that there exists a path in Π, where Fp false is. This contradicts with the first sentence. Since Π was chosen arbitrarily, the assumption is in general true.

Alternative solution (direct proof):
$M \vDash AG\ AF\ p \Leftrightarrow M, \pi \vDash G\ AF\ p$ (we choose π arbitrarily starting from any $s \in S_0$)

$\Leftrightarrow \forall k \geq 0\ M, \pi^k \vDash AF\ p$ (semantics of G)
$\Leftrightarrow M, \mu \vDash Fp$ (semantics of A, we choose μ as an arbitrary state starting from the starting state of π^k)

Because μ is any path that originates from any state in π, μ includes all states $k \geq 0\ \pi^k$ and therefore ⇒ $M, \pi \vDash GF\ p \Leftrightarrow M \vDash GF\ p$ (by semantics of G and ⊨)

ii. for the other way round, the argumentation is basically the same, but reversed
We have an arbitrary path pi = s0, s1… . GF p implies that globally, so everything, there will be p at some state in the future. This means there is no path, where this is not the case. Now we assumed that AGAF p is not entailed. AGAF p means that on all paths, everywhere, p will hold at some point in the future. But it follows from GF p that this is the case. so it can't be that AGAF is not entailed as pi is chosen arbitrarily.

iii.
FGp ⇒ AFAGp

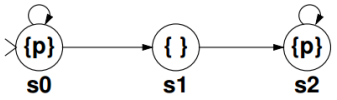
~~Like above, proof by semantics and contradiction.~~
~~FGp ⇒ NOT(AFAGp)~~

Is it enough to draw a kripke structure where this does not hold? E.g. use the kripke structure from http://www.inf.ed.ac.uk/teaching/courses/propm/papers/CTL.pdf page 31. But why does AFAGp does not hold on that graph? Doesn't that mean, in all possible paths there will be a point in the future, from that AG p holds. But don't we end up at s2 at some point in all cases, so AG p would be true? Why does this not hold then? Does the possibility of staying infinitely long in s0 (so we have an infinite path s0) where AG p does not hold break this?

FG on the other hand is LTL, so it is not seen as a tree of paths but basically as a single path, and if you see it as a single path at some point you can go to s2 and then Gp holds ???

Counterexample:
Explanation: AF means, eventually on all possible paths-> so there exists one path, where we stay for ever in s0! And in this path AGp does not hold, because after you have been infinitly long in s0 you select again all possible paths from that point and can go to s1 where p does not hold.

The LTL property $\mathbf{F}\,\mathbf{G}\,p$ is not expressible in CTL:



$\mathcal{K} \models \mathbf{F}\,\mathbf{G}\,p$ but $\mathcal{K} \not\models \mathbf{AF}\,\mathbf{AG}\,p$

Exam 03.07.2015 (FMI.154.pdf)

http://www.logic.at/lvas/185291/angaben/fmi154.pdf

Block 1

1.) Consider the following problem:

> **HALTING-ON-PAIR**
>
> INSTANCE: A tuple (Π, I_1, I_2), where I_1, I_2 are strings and Π is a program that takes two strings as input.
>
> QUESTION: Is it the case that Π halts on the pair (I_1, I_2)?

By providing a reduction from an undecidable problem, prove that **HALTING-ON-PAIR** is undecidable. Argue formally that your reduction is correct. **(15 points)**

added by student - Halting on pair (Π,I1,I2)

Let Π' be an arbitrary instance for halting that takes a single string I' as input. We construct an instance of halting-on-pair as follows:

```
Π(I1,I2) {
  Π'(I1);
  return;
}
```

To show that Halting-on-pair is undecidable, we have to show that the following equivalence holds:

Π(I1,I2) is a positive instance of Halting-on-pair ⇔ (Π',I1) is a positive instance of halting

“=>” suppose Π(I1,I2) is a positive instance of halting-on-pair. This means that Π halts on (I1,I2). By the construction of Π, this also means that Π’ halts on I1 and therefore (Π’,I1) is a positive instance of halting.
“<=” suppose (Π’,I1) is a positive instance of halting. In this case by the construction of Π the return statement is reached and Π halts which means that (Π,I1,I2) is a positive instance of halting-on-pair

More detailed added by student (please approve if correct):
Let (Π, I) be an arbitrary instance of the halting problem, i.e. Π as a program that takes one string as input and I is an input for Π. From this, we construct an instance (Π1, I1, I2) of HALTING-ON-PAIR by setting I = I1 = I2 and building Π1 from this as follows:

```
void Π1(String s1, String s2) {
        call(Π(s1));
        call(Π(s2));
}
```

It remains to show that the following equivalence holds:

$$\Pi \text{ halts on } I \Leftrightarrow \Pi 1 \text{ halts on } I1 \text{ and } I2$$

“⇒”: Suppose Π halts on I. Due to the construction of Π1, Π1 also halts on I since I = I1 = I2.
“⇐”: Suppose Π1 halts on I1 and I2. Since I1 = I2 = I and Π1 involves running of Π on I, we have that Π halts on I.

Block 2

a) Prove by contradiction, i.e. there exists an interpretation I that does not satisfy ϕ:
(1) I |/= ϕ (assumption
(2) I |= f(f(f(a))) = f(b) (by (1) and by semantics of ->)
(3) I |/= a=b (by (1) and semantics of ->)
(4) I |= a!=b (by (3))
(5) I |= f(f(a)) = f(f(f(a))) (by applying f-idempotency ~~on (2)~~)
(6) I |= f(a) = f(f(a)) (by applying f-idempotency ~~on (5)~~)
(7) I |= f(a) = f(f(f(a))) (by replacing equivalent terms of (5) and (6))
(8) I |= f(a) = f(b) (by replacing equivalent terms of (2) and (7))
(9) I |= a = b (by applying f-injectivity to (8))
(10) F ((4) and (9))

Not sure if necessary but alternative for (5)-(7):

(5) I |= f(a) = f(f(a)) (f-idempotency)
(6) I |= a = f(a) (5, f-injectivity)
(7) I |= a = f(f(a)) (5,6, transitivity)
(8) I |= f(a) = f(f(f(a)) (7, substitution)

<u>yet another proof by student (please check):</u>

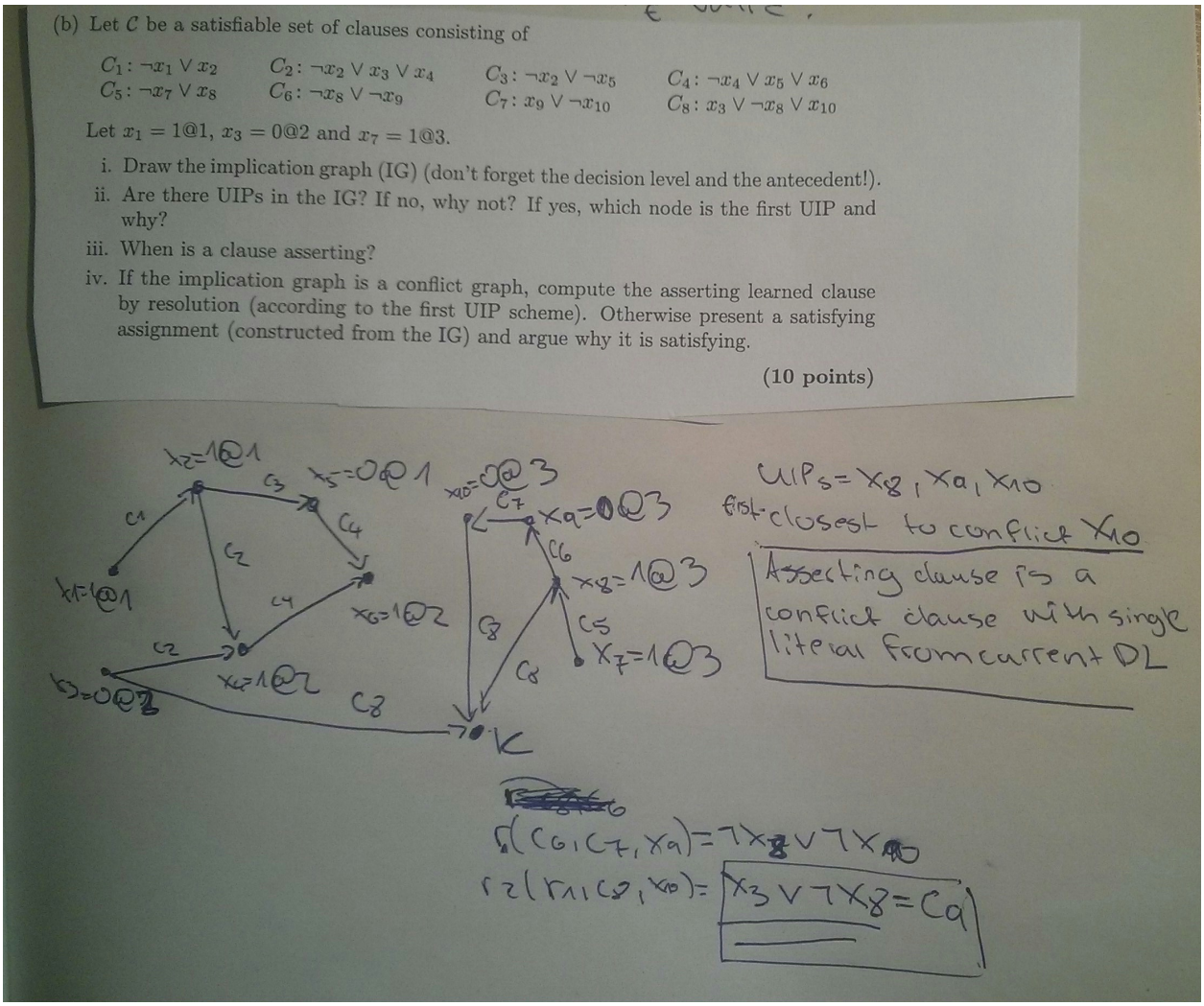
phi is valid means (not phi) is unsatisfiable

(1) f(f(f(a))) = f(b)
(2) a != b
(3) f(a) = f(b) | (1) + f-idempotency applied two times
(4) a=b | (3) + f-injectivity
x | contradiction of (2) and (4)

b)

- I received a conflict in the last clause c8 (-x3,x8,-x10)
- “A unique implication point (UIP) is an internal node in the IG that all paths from the decision node (at the current dl) to the conflict node go through it. The first UIP is the UIP closest to the conflict.” - therefore I think the UIPs are x8@3 and x7@3 and x8@3 is closer to the conflict node.
- “An asserting clause (AC) is a conflict clause with a single literal from the current decision level. Backtracking to the right level makes it a unit clause.”

Student s’olution:



Block 3

Added by student (please confirm)

a) Proof by counterexample

Let F:{x=0}, p:{x=1}, q:{skip} and G:{x=1}. Obviously {x=0}x=1;skip{x=1} holds. If we set H:{x≤0}, the precondition also holds, but the postcondition becomes false:

{x=0 & x≤0}x=1;skip{x=1 & x≤0}

b)

The required states can be computed using the weakest precondition:

wp(while … od,x=0) = Exists i ≥0 Fi

F0: 3x=2y & x=0 => x=0 & y=0

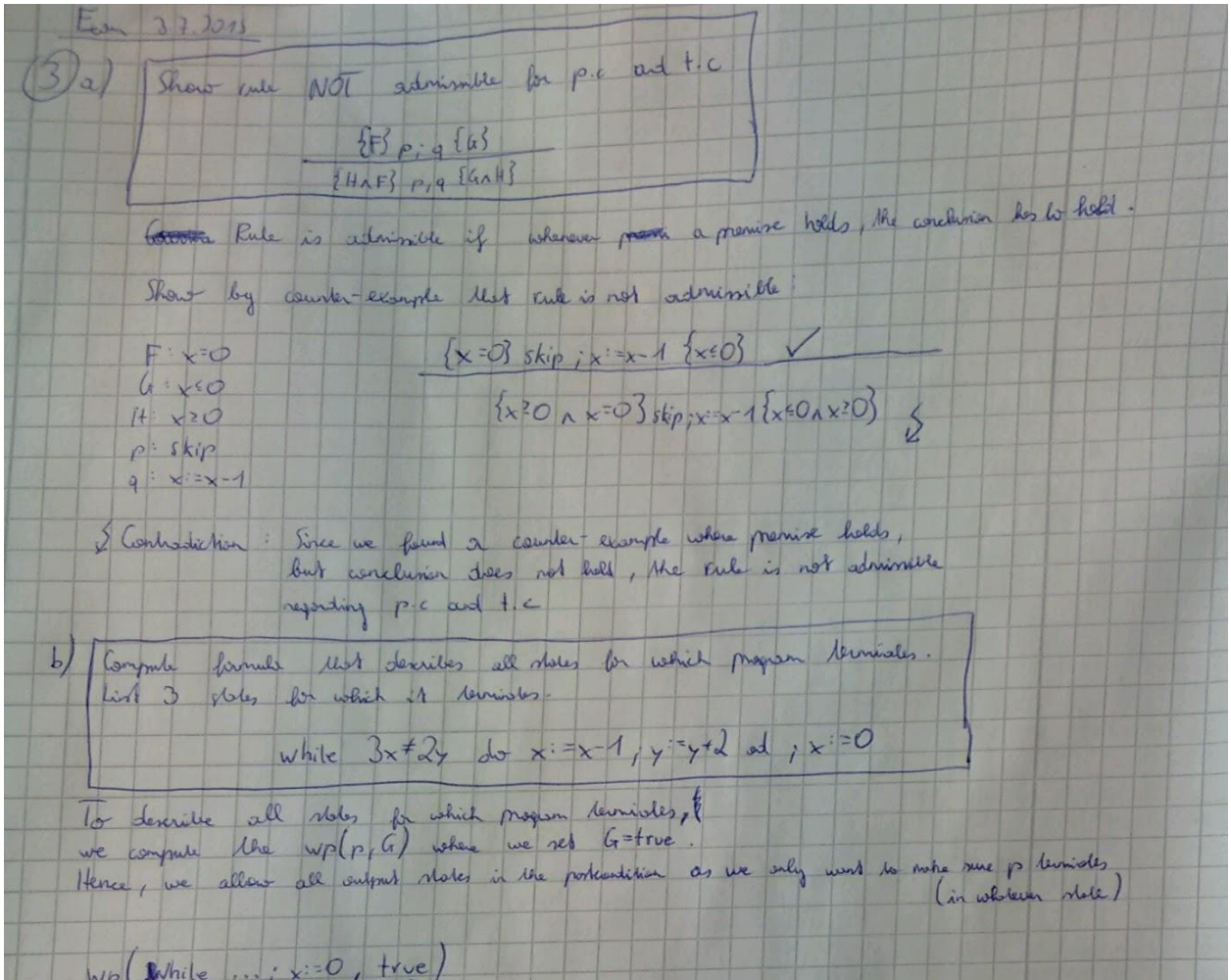
F1: 3x!=2y & wp(x=x-1;y=y+2,F0) = 3x!=2y & x-1=0 & x+2=0 => 3x!=2y & x=1 & y=-2
Guess: Fi: Exists i≥0 & x=i & y=-2i
=> Fi+1 = 3x!=2y & wp(x=x-1,y=y+2;Fi) = 3x!=2y & x-1=i & x+2=-2i = x=(i+1) & y=-2(i+1) (Proof)

=> wp(while...od,x=0) = Exists i≥0 x=i & y=-2i = Exists i≥0 x=i & y=-2x = x≥0 & y=-2x (the requested formula)

3 states:
{x=0,y=0}
{x=1,y=-2}
{x=2,y=-4}

alternative Version:

Solution
Status: **solved by student, partly approved by professor**
3a: approved by professor



13b: Set Postcondition G=true approved by professor (rest of the computation solved by student)

Comment to solution of 3b (by student):
In my opinion the very last part of this example is false. As soon as you get rid of the exists quantifier you lose information. It is easiest to see if you try to run the code with the defined states. The program will not terminate. In my opinion it is enough to calculate the line with the exists statement.

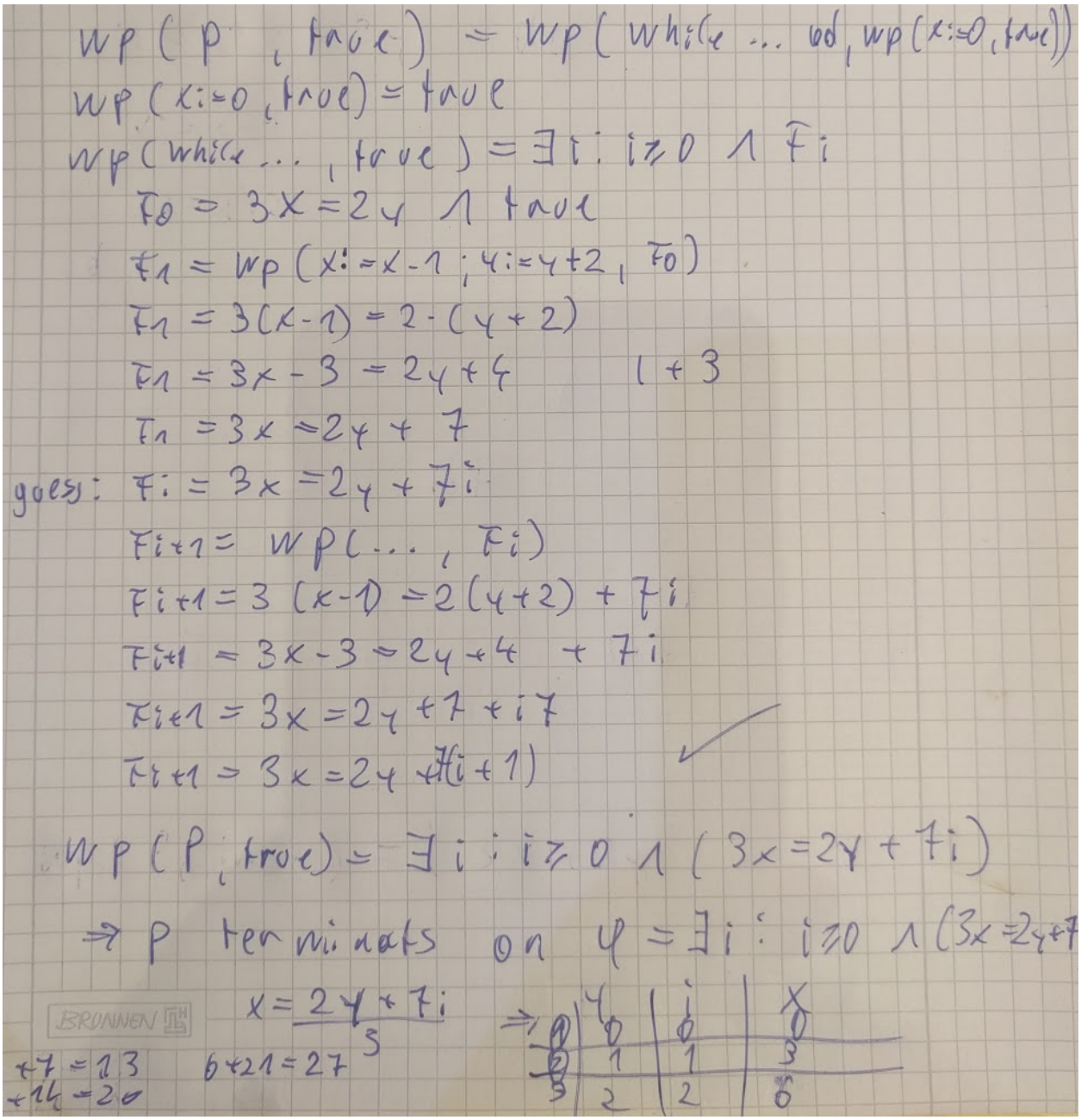
As mentioned in the comment the provided states in the solution are not correct. If you doublecheck them by searching for a correct i you'll see that you cannot find a valid integer for it.

$\exists$ i(i≥0 $\wedge$ 3x=2y+7i)
x=3,y=3: 9=6+7i ➜ i=3/7 ↯
x=2,y=3: 6=6+7i ➜ i=0 ✓
x=0,y=0: 0=0+7i ➜ i=0 ✓

The simplification step from
$\exists$ i(i≥0 $\wedge$ 3x=2y+7i) to x = 2y/3 is therefore losing too much information.

We would suggest to just stop at
$\exists$ i(i≥0 $\wedge$ 3x=2y+7i) and create valid states from that.

Other valid states:
x=3,x=6: i=0 ✓
x=7,y=0: i=3 ✓



Block 4

Added by student (please confirm)

a) H={(s0,t0),(s1,t0),(s2,t4),(s3,t2),(s4,t4),(s5,t0)}

b)

i) EG(a): CTL+CTL*: s1,s2

ii) EX(a & b): CTL+CTL*: s1

iii) EF(a & b): CTL+CTL*: s1,s2

iv) X(b & c): LTL+CTL*: s0

v) F(a & b): LTL+CTL*: ~~none~~, s2

c)

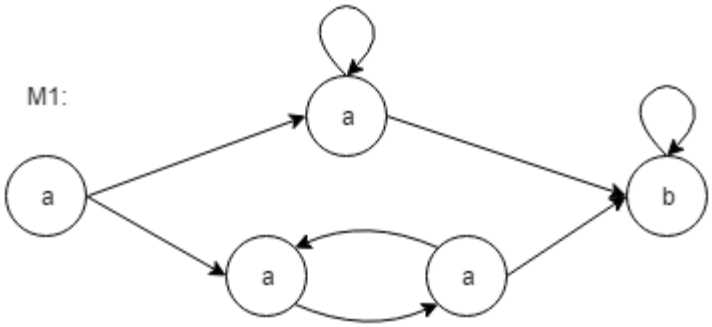
Make 2 structures like M1:

A
/ \
B B
| \ /
C D

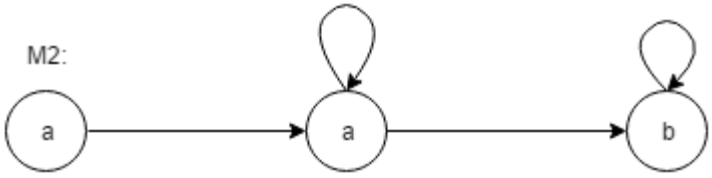
M2:
A
|
B
/ \
C D
And the formula G(B -> (XC or XD)), valid for M2, not valid for M1, traces are equivalent

Remark to this solution: ϕ is required to be a CTL formula so it should be written as: AG(B -> (AX(C) or AX(D)))

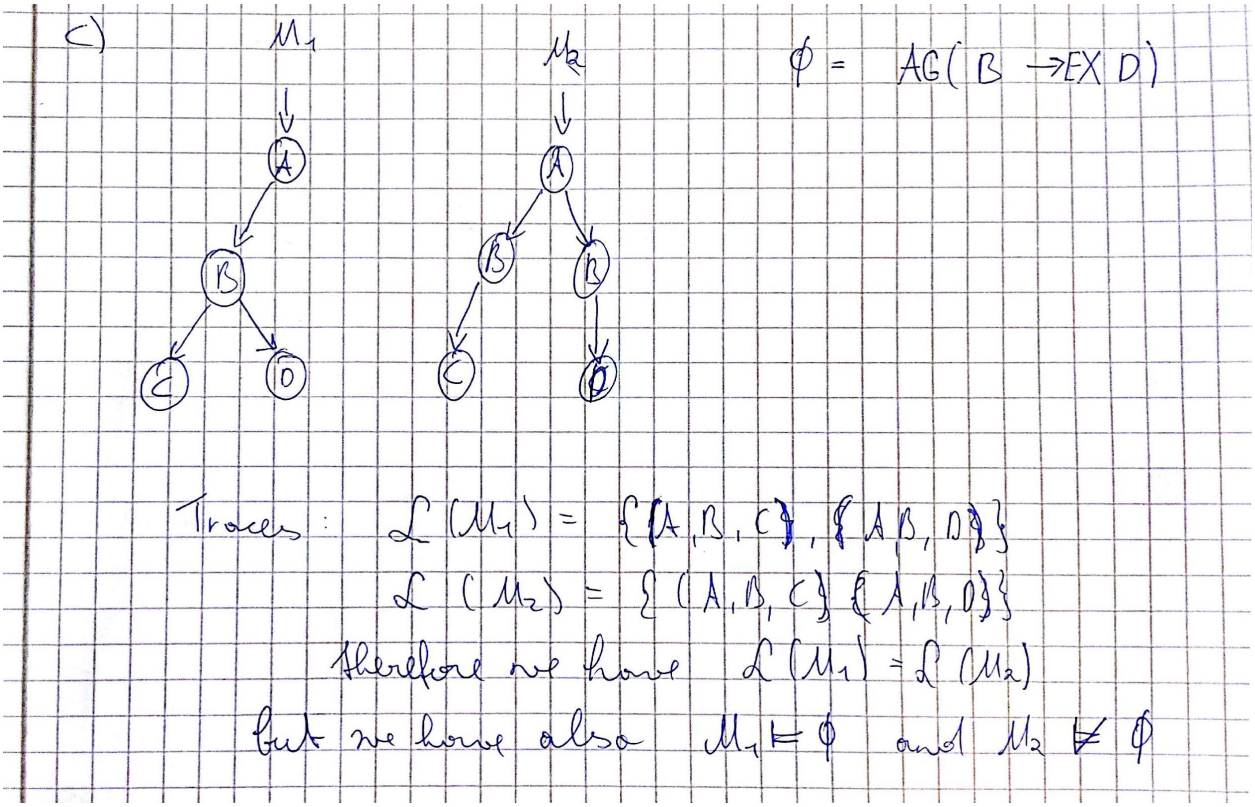
Other solution to c):



Φ: AX AG EX(b)



And another approach to c):



Exam 08.05.2015 (FMI.153.pdf)

http://www.logic.at/lvas/185291/angaben/fmi153.pdf

Block 1

PROB

INSTANCE: An undirected graph $G = (V, E)$ such that G has a vertex v_0 that is connected to precisely all other vertices of G, i.e. there exists $v_0 \in V$ such that (a) $[v_0, v] \in E$ for all $v \in V \setminus \{v_0\}$, and (b) $[v_0, v_0] \notin E$.

QUESTION: Is it the case that G is 4-colorable? That is, does there exist a function μ from vertices in V to values in $\{1, 2, 3, 4\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

Provide a reduction from **PROB** to 3-**COLORABILITY**, and explain the intuition behind your reduction. **(15 points)**

Let G(V, E) be an arbitrary instance of PROB. We construct an instance G'(V', E') of 3-COLORABILITY as follows:

- V' = V \ {v0}, because if we would keep v0 we would need a fourth color
- E' = [v1, v2] element of E, we exclude all [v0, v] from E, where v is an arbitrary node
- my' = assigns a subset {1, 2, 3} of my to vertices from V', s. t. my'(v1) != my'(v2). Hence following also holds my(v1) != my(v2) for v1 and v2 from V' for a reduced color set {1, 2, 3}.

We provide a proof by contradiction. We show that if G(V, E) is a pos. instance of PROB <-> G'(V', E') is a pos. instance of 3-COLORABILITY.

->

Assume G(V, E) is a positive instance of PROB. That means there exists v0 ∈ V such that (a) [v0, v] ∈ E for all v ∈ V \ {v0}, and (b) [v0, v0] not∈ E. By construction of G' we have 3-colorability, s. t. for every vertex of an arbitrary edge [v1, v2] from E' the assignment function my' holds, s. t. my(v1) != my(v2). It follows that G' is a pos. instance of 3-COLORABILITY.

Let's assume that there exists an arbitrary edge [v1, v2] from E' s. t. v1 and v2 have the same coloring. That cannot be the case, because by construction of E', we define that v0 is

not part of any edge [v1, v2] of E'. Hence the wrong coloring must be present already in G, but by definition of my that cannot be the case, therefore there exists no arbitrary edge [v1, v2] from E' with the same coloring.

<-

Assume G'(V', E') is a positive instance of 3-COLORABILITY, that means by definition of my', there exists no [v1, v2] from E' with same coloring. Each v from V' contains one of the following colors {1, 2, 3} assigned by my'.
By construction of G' it follows that G is a positive instance of PROB. V from G contains an additional vertex v0 s. t. every vertex v from V is connected to v0. That means that for every vertex v from V there exists an edge [v0, v] element of E. By definition of my from G, my assigns four colors {1, 2, 3, 4} to vertices v from V s.t. my(v1) != my (v2) for every v1 and v2 in V. Because my' defined 3 colorability and by adding v0 to V, we simply assign the fourth color to v0, thus we have 4 colorability. Hence G is a positive instance of PROB.

Block 2

a)

ϕ-EUF is sat iff FC^E and flat^e is sat

(a) Show the following:

$$\varphi^{EUF} \text{ is satisfiable iff } FC^E \wedge \mathit{flat}^E \text{ is satisfiable.}$$

FC^E and flat^E are obtained from φ^{EUF} by Ackermann's reduction. (Hint: FC^E is the same for φ^{EUF} and $\neg\varphi^{EUF}$.)

Solution:

We make precise here on which formula FC^E and $flat^E$ depend. Both are computed according to AR. First observe that

$$\varphi^{EUF} \text{ is valid} \quad \text{iff} \quad FC^E(\varphi^{EUF}) \rightarrow flat^E(\varphi^{EUF}) \text{ is valid.}$$

Taking the above hint into account, we get

$$\begin{array}{lll} \neg\varphi^{EUF} \text{ is valid} & \text{iff} & FC^E(\neg\varphi^{EUF}) \rightarrow flat^E(\neg\varphi^{EUF}) \text{ is valid} \\ & \text{iff} & FC^E(\varphi^{EUF}) \rightarrow flat^E(\neg\varphi^{EUF}) \text{ is valid.} \end{array}$$

Since φ^{EUF} and $flat^E(\varphi^{EUF})$ have the same propositional structure (only some arguments of literals change during the computation of $flat^E(\varphi^{EUF})$ from φ^{EUF}), the equality $flat^E(\neg\varphi^{EUF}) = \neg flat^E(\varphi^{EUF})$ holds. Consequently,

$$\neg\varphi^{EUF} \text{ is valid} \quad \text{iff} \quad FC^E(\varphi^{EUF}) \rightarrow \neg flat^E(\varphi^{EUF}) \text{ is valid.} \qquad (*)$$

For space reasons, we omit the superscript EUF in the following. Then we have:

			Explanation
φ is sat	iff	$\neg\varphi$ is *not* valid	Ψ is sat iff $\neg\Psi$ is not valid
	iff	$FC^E(\varphi) \rightarrow \neg flat^E(\varphi)$ is *not* valid	from $(*)$ above
	iff	$\neg\neg(FC^E(\varphi) \rightarrow \neg flat^E(\varphi))$ is *not* valid	$\neg\neg\Psi \equiv \Psi$
	iff	$\neg(FC^E(\varphi) \rightarrow \neg flat^E(\varphi))$ is sat	Ψ is sat iff $\neg\Psi$ is not valid
	iff	$FC^E(\varphi) \wedge flat^E(\varphi)$ is sat	basic propositional manipulations

b)

Clarify logical status

Clarify the logical status of each of the following formulas:

i. φ_1^{EUF}: $f(x) \doteq f(y) \wedge x \not\doteq y$

ii. φ_2^{EUF}: $x \doteq y \wedge f(x) \not\doteq f(y)$

If the formula is E-valid or E-unsatisfiable, then give a proof based on E-interpretations and semantics. If the formula is E-satisfiable but not E-valid, then present two E-interpretations, one satisfying the formula and one falsifying it. Argue formally why the formula is true respectively false under the considered E-interpretation.

Intuition, e.g. f = %2 function:

- i): is E-SAT but not E-VALID, x = 2, y = 4 -> output is the same (0)
- ii) is not E-SAT (also not E-VALID), x = 2, y = 2 -> output must be the same(0) but is not

i) 2 E-interpretations, we assume that f(i) = return i %2;

- t: x = 2, y = 4 (x != y) and f(x) = f(y) = 0
- f: x = 2, y = 3 (x != y) and f(x) != f(y) 0 vs. 1

ii) proof for E-unsatisfiability

Axiom:

$$\forall x_1, y_1, \ldots, x_n, y_n \left(\bigwedge_{i=1}^{n} x_i \doteq y_i \rightarrow f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n) \right)$$

Given: x = y and f(x) != f(y)

We replace x = y with the deduction of our axiom and get: f(x) = f(y) and f(x) != f(y) -> contradiction

Block 3

Show that $\{ F \}\, v := e \,\{ F[v/e] \}$ is not a sound axiom.

Short version

We know that {F[v/e]} v := e {F} is a sound axiom. (= wp)
To show that the above axiom is not a sound axiom it suffices to show that there exists a case where the assertion does not hold.

F: v = 1
p: v := 2 (e = 2)
F[v/e] = 2 = 1 (false) - we replace v and not the value of v!

(v=2) -> (2=1) … false

Long version

That means to get to the end state we replace the variable of our initial state with the expression in our assignment. Insert into the initial state.

Example: {x = 42} x:= x + 1 {F[x/e]}
end state: {x + 1 = 42} = {x = 41}

sanity check: x = 42 + 1 should compute 43 and not 41. Our postcondition is false w. r. t. the input state and program, a. s. the axiom is not sound.

From Hoare calculus we know:

$$\{ F[v/e] \}\ v := e\ \{ F \}\ \text{(as)}$$

The assignment axiom scheme is equivalent to saying that to find the precondition, first take the post-condition and replace all occurrences of the left-hand side of the assignment with the right-hand side of the assignment.

Auf gut Deutsch:" Man nimmt die Post-Condition her und setzt die Expression in die Variable ein." Man setzt die Gegenwart in die Zukunft ein um zur Vergangenheit zu gelangen. Immer in die Post-Condition einsetzen, nie in die Pre-Condition!!!!

$$\{ x+1 = 43 \}\quad y := x + 1\quad \{ y = 43 \}$$
$$\{ x + 1 \le N \}\quad x := x + 1\quad \{ x \le N \}$$

Be careful not to try to do this backwards by following this *incorrect* way of thinking: {*P*} *x*:=*E* {*P*[*E*/*x*]}; this rule leads to nonsensical examples like:
{ *x* = 5 } *x* := 3 { 3 = 5 }
Hier wurde die Variable x mit dem Wert 3 ersetzt und somit kommt Schwachsinn raus!

MORE EXAMPLES HERE: http://en.wikipedia.org/wiki/Hoare_logic

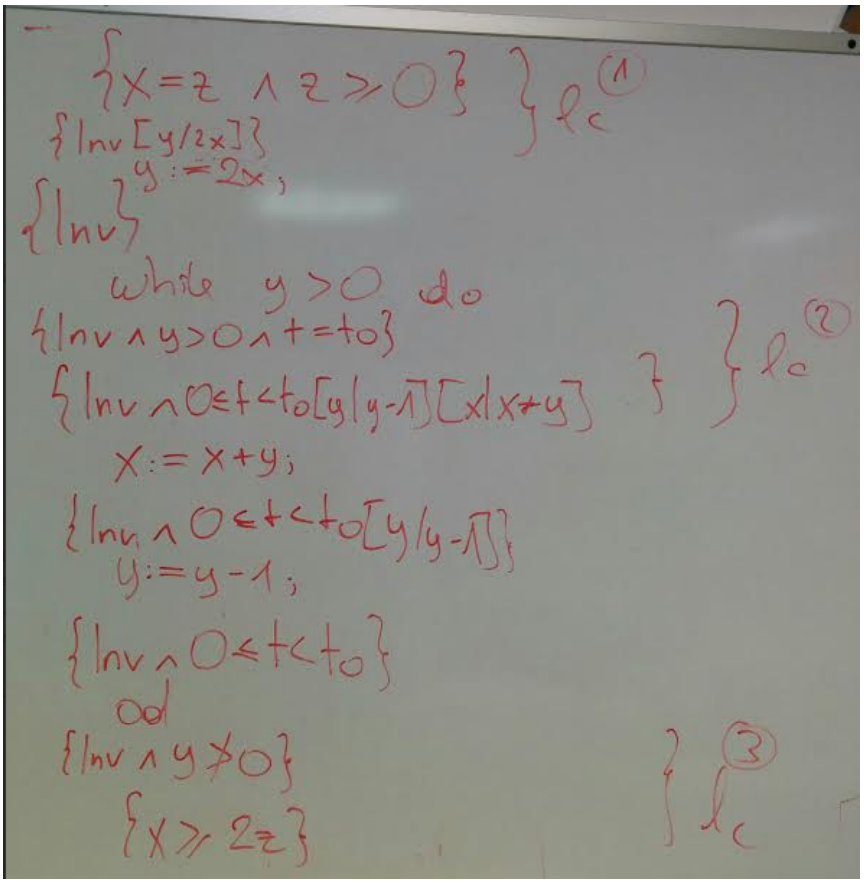
3.B)

(b) Prove that the following correctness assertion is true regarding total correctness. Use the invariant $2x + y^2 + y = 4z(z+1) \wedge y \geq 0$.

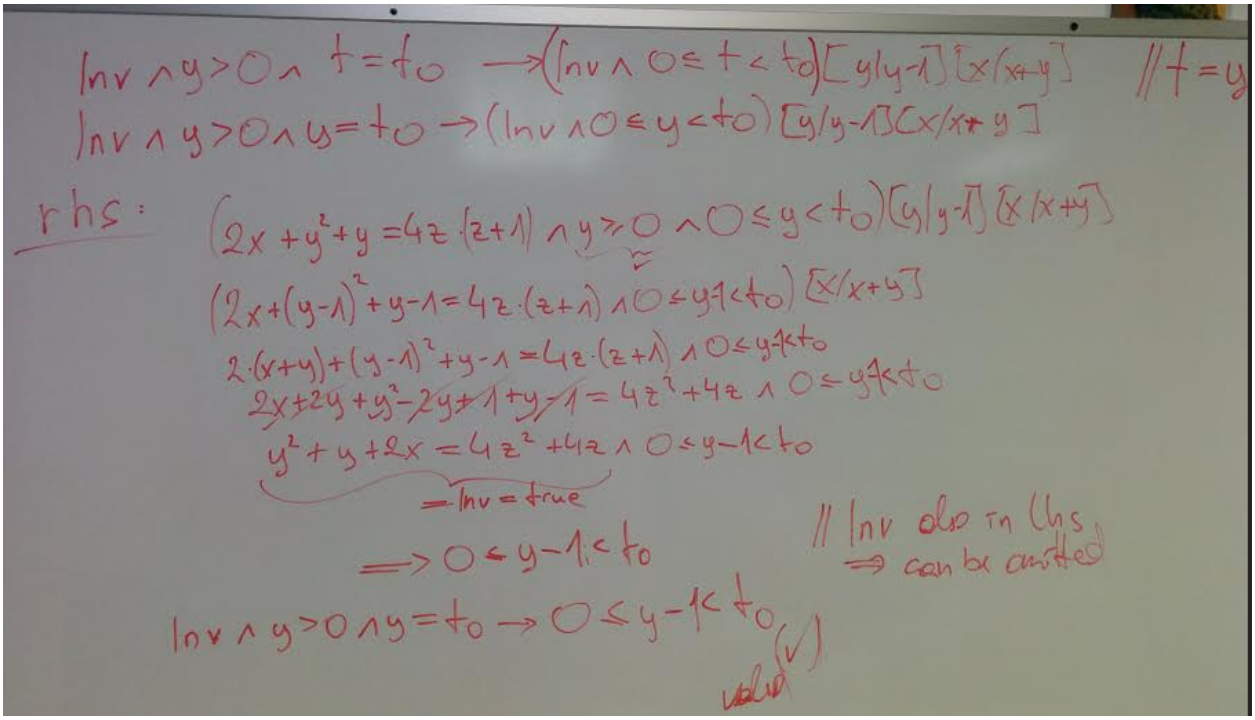
You may need one of the following annotation rules:

$\{\, Inv \,\}$while e do $\{\, Inv \wedge e \wedge t{=}t_0 \,\} \cdots \{\, Inv \wedge 0 \leq t < t_0 \,\}od\{\, Inv \wedge \neg e \,\}$
$\{\, Inv \,\}$while e do $\{\, Inv \wedge e \wedge t{=}t_0 \,\} \cdots \{\, Inv \wedge (e \Rightarrow 0 \leq t < t_0) \,\}od\{\, Inv \wedge \neg e \,\}$

$\{\, x = z \wedge z \geq 0 \,\}$
$y := 2x;$
while $y > 0$ do
    $x := x + y;$
    $y := y - 1;$
od
$\{\, x \geq 2z \,\}$



1. **z ≥ 0 -> z ≥ 2z (ok) (or same with x, because x = z)**
2. **Don't forget when evaluating expressions: first inner [...], then outer [...]**



1. **x = 2z (z + 1) -> x ≥ 2z (ok)**
    a. **2z (z + 1) ≥ 2z for x = 2z (z + 1)**
    b. **2z^2 + 2z ≥ 2z // -2z**
    c. **2z^2 ≥ 0 // always valid**

Our mistake - first inner [...], then outer [...], like here:

$$(x \geq y \wedge x \neq y \wedge x - y = z) \Rightarrow (x \geq y \wedge 0 \leq x - y < z)[y/y+2][x/x+1]$$
$$(x \geq y \wedge x \neq y \wedge x - y = z) \Rightarrow (x \geq (y+2) \wedge 0 \leq x - (y+2) < z)[x/x+1]$$

Now we have proven all 3 implications, that means we have proven that the given code is true regarding total correctness.

Block 4

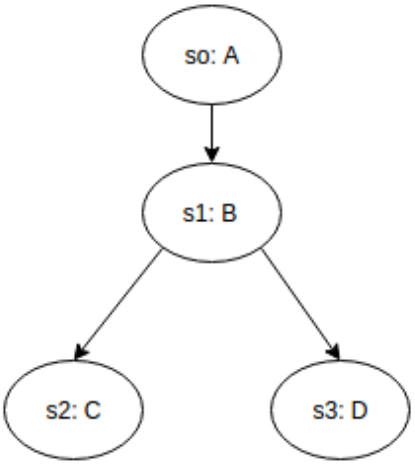
a)

Not sure if this works:

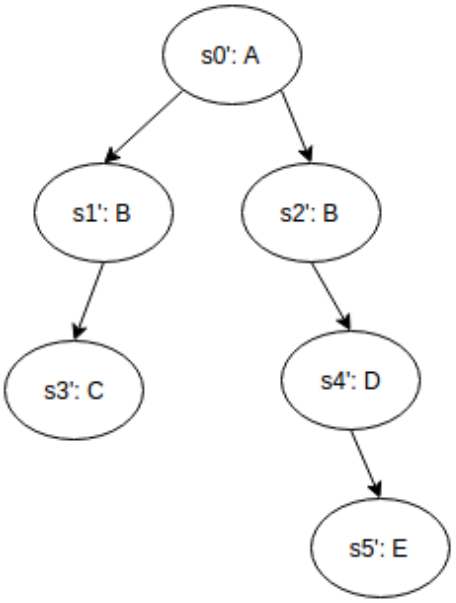
By counter example.

Consider following Kripke structures M1 and M2:

M1



M2

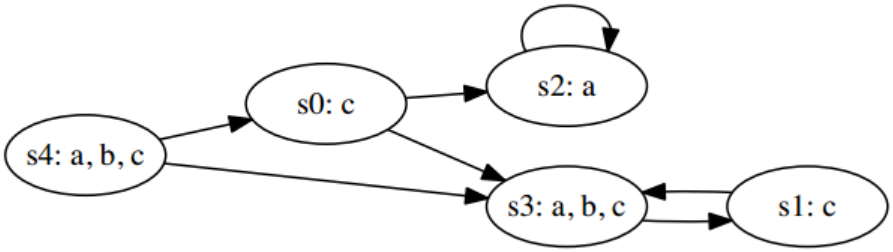


L(M1) = {A,B,C,D} (Not sure how to write traces)
L(M2) = {A,B,C,D,E}

So L(M1) $\subseteq$ L(M2) holds

But No simulation Relation H is possible since state s1 and the relating states s1', s2' have different successors.

b)



For each of the following formulae

i. determine if the formula is in CTL, LTL, and/or CTL*, and
ii. state on which states s_i the formula holds

$\mathbf{G}(a)$
$\mathbf{X}(b \wedge c)$
$\mathbf{AG}(c)$
$\mathbf{EF}(c)$

	LTL	CTL	CTL*	States
G(a)	x	-	x	s2
X(b and c)	x	-	x	s1
AG(c)	~~x~~	x	x	s1, s3
EF(c)	-	x	x	s0, s1, s3, s4

c)

```
int curWeight = 0, curValue = 0, chosen[N], j=0;
for(int i = 0; i < N; i++) {
       chosen[i]=(-1); //initialize chosen
       // if(nondet_bool() && curWeight + weights[i] < W )
       if(nondet_bool()) { //no need to check && curWeight + weights[i] < W as well
              Chosen[ j++] = i; //add the index to the array
              curWeight += weights[i];
              curValue += values[i];
       }
}
```

```
// assert(curWeight < W) but this is basically implicitly true so we need an extra
assertion?
// assert(curValue > V);
// just write one assertion
assert(curWeight > W || curValue < V, "ERROR: Counterexample: " +
chosen.arrayToString());
// instead of the above i find it simpler to negate the positive case:
assert(!(curWeight <= W && curValue >=V), "ERROR: Counterexample: " +
chosen.arrayToString());
```

Now my questions are: How to “report” the chosen values using CMBC?? that’s why i save them in an array currently, whether if they are chosen or not. but is that reporting?
I assumed that the items I are a set, so each item only exists once. (I think it states it’s a set)
Is this even valid CMBC? do i have to apply loop unwinding somehow? basically it randomly guesses possible sets and eventually every possible combination will be tried. the loop is bounded and i have an assertion so it should be possible to evaluate it using CMBC.

Exam 27.03.2015 (FMI.152.pdf)

http://www.logic.at/lvas/185291/angaben/fmi152.pdf

FIND-INPUT

INSTANCE: A pair (Π, I), where (i) I is a string, and (ii) Π is a program 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 the program Π on input I' returns I, i.e. $\Pi(I') = I$?

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

We recall properties of semi-decidability:

- for every positive instance of a problem the program returns true
- for every negative instance of a problem the program returns false or does not terminate

To show that FIND-INPUT is semi-decidable we can provide a procedure Π' that fulfills the above conditions. Π' simulates a run of (Π, I), an instance of FIND-INPUT. We argue that such an interpreter Π' is a semi-decision procedure for FIND-INPUT.

We distinguish the following (3) cases:

1. Π' returns true for every positive instance of FIND-INPUT. That is, if for any input I' returns I, then Π' returns true.
2. Π' return false for every negative instance of FIND-INPUT. That is, if for any input I' Π terminates and does not return I, then Π' returns false.
3. Π' does not terminate for a negative instance of FIND-INPUT. That is, if for any input I' Π does not terminate on I', then also our Π' does not terminate. Hence we have shown that our Π' is a valid semi-decision procedure.

We can omit case 3, because we know that Π will terminate on any input I', but to check all instances I' will take forever, thus Π' might not terminate for an arbitrary instance of FIND-INPUT.

Block 2

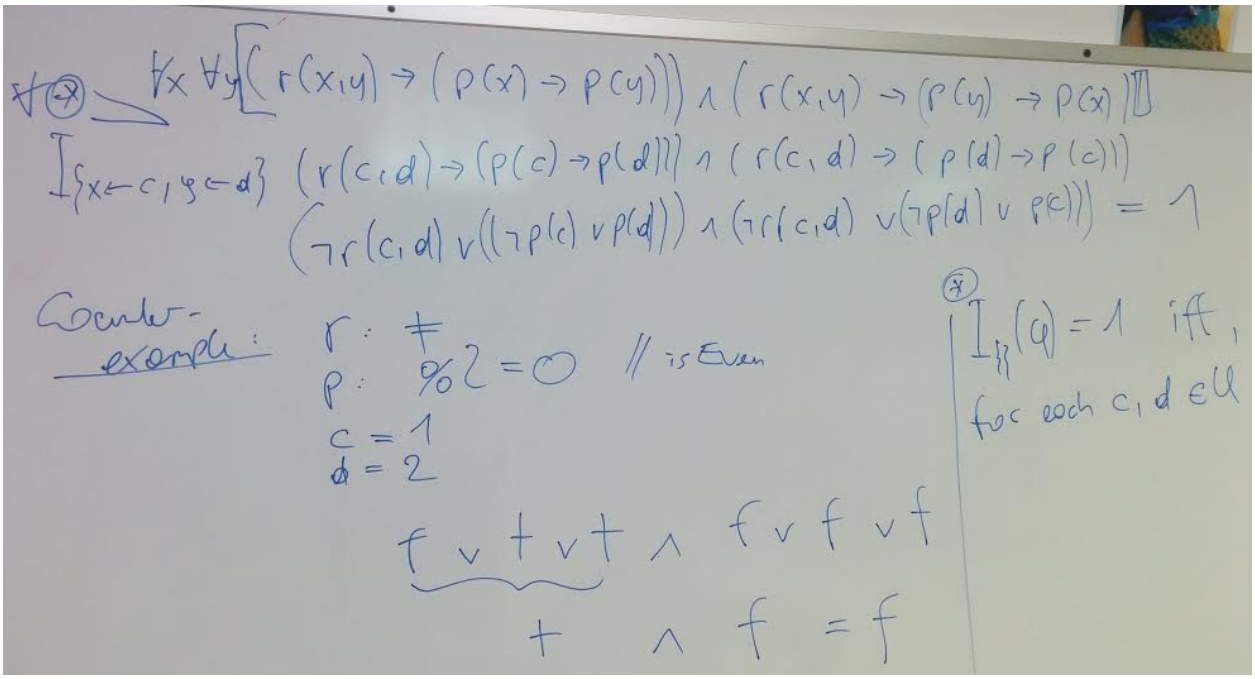
2a) First-order formula

(a) Let φ be the first-order formula

$$\forall x \forall y \left[\big(r(x,y) \to (p(x) \to p(y))\big) \wedge \big(r(x,y) \to (p(y) \to p(x))\big) \right] .$$

i. Is φ valid? If yes, present a proof. If no, give a counter-example and prove that it falsifies φ.

ii. Replace r in φ by $\doteq$ (equality) resulting in ψ. Is ψ E-valid? Argue formally! (Hint: Substitution axioms)



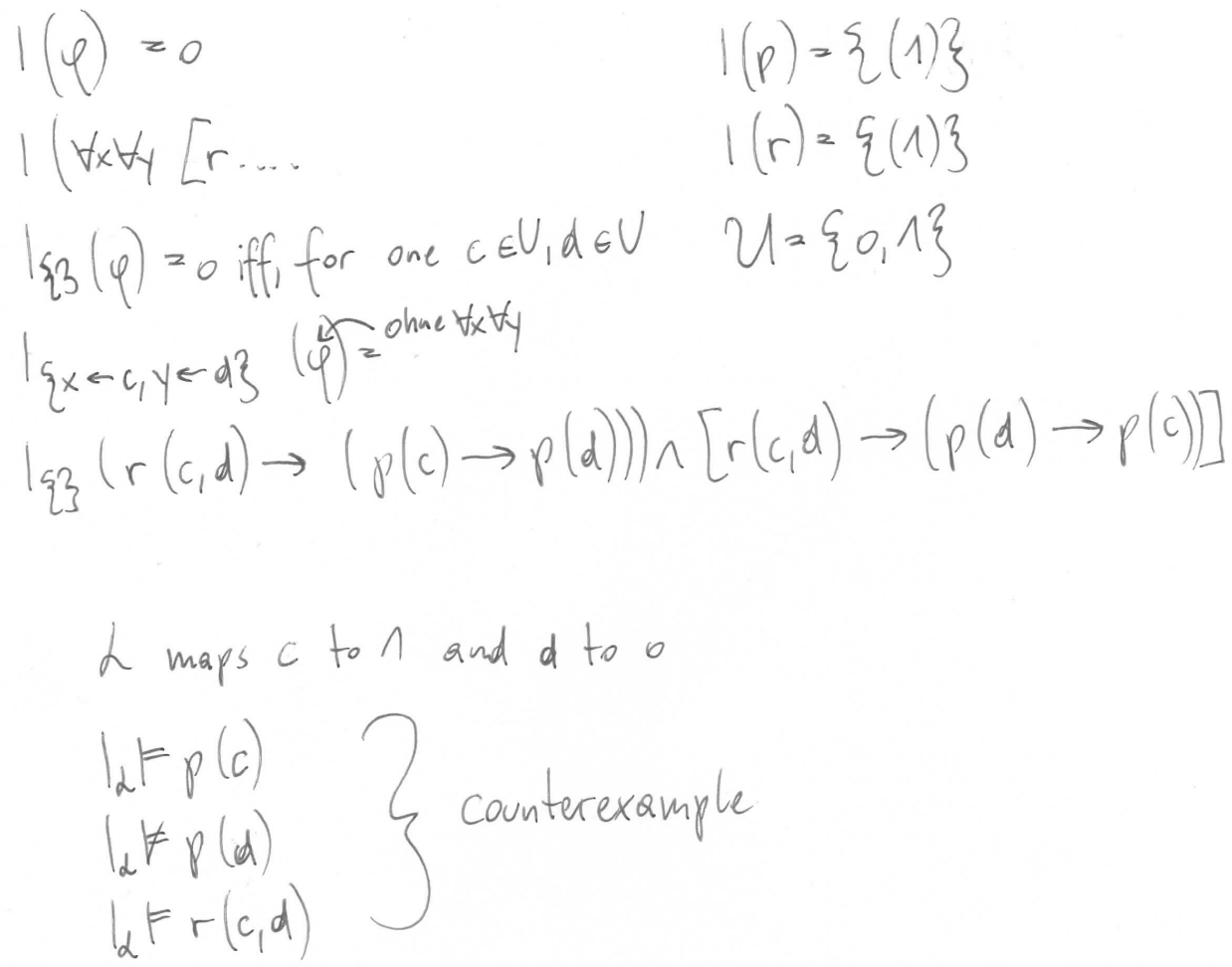
Do the same step as in 1a) to get rid of th

5. Substitution axioms for each predicate symbol p of arity n:

$$\forall x_1, y_1, \ldots, x_n, y_n \left(\bigwedge_{i=1}^{n} x_i \doteq y_i \to \big(p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n)\big) \right)$$

e quantifier. Then substitute lhs of above implication with the rhs in that formula. Then you have the equivalence in your formula that implies each side of the equivalence. Hence the rhs is always true, which makes your formula validate to true.

a) i. other solution (not sure if correct), status student:



b)

(b) Let φ be a propositional formula in negation normal form (NNF).

Prove: If φ contains only pure literals, then φ is satisfiable.

Hint 1: a literal is pure in a formula, if it occurs only positively or negatively in that formula, but not both.
Hint 2: negation normal form means that negations only occur directly in front of a propositional variable and nowhere else.
Hint 3: the number n of binary connectives occurring in φ may be arbitrary high, which proof principle is suitable for such cases?

Solution:
Since φ is in NNF, the only logical connectives are $\wedge, \vee$, and $\neg$ where the latter only occurs directly in front of a propositional variable. Thus the only binary connectives are $\wedge$ and $\vee$.

Let $\mathcal{BV}$ be the set of propositional variables that occur in φ. We define I as follows: if $b \in \mathcal{BV}$ occurs positively in φ, then $I(b) = 1$, else $I(b) = 0$. Since all literals in φ are pure, this definition of I does not contradict itself.

To show that φ is satisfiable, we use induction on the number n of binary connectives in φ, i.e., we show that every subformula of φ is true in I.

Base case $n = 0$: ψ is a literal, hence $I \models \psi$.

IH: $I \models \psi$ for every subformula ψ of φ with less-or-equal to n binary connectives.

Induction $n \mapsto n + 1$: let ψ contain $n + 1$ binary connectives, we distinct on the topmost binary connective of ψ:

- $\wedge$: then $\psi = \phi_1 \wedge \phi_2$ where ϕ_1 and ϕ_2 both contain less-or-equal than n binary connectives. By IH it thus holds that $I \models \phi_1$ and $I \models \phi_2$, hence $I \models \psi$.
- $\vee$: then $\psi = \phi_1 \vee \phi_2$ and again by IH it holds that $I \models \phi_1$, hence $I \models \psi$.
- nothing else is possible since φ is in NNF.

Therefore $I \models \psi$ for every subformula ψ of φ, specifically $I \models \varphi$ since φ is also a subformula of φ.

https://math.stackexchange.com/questions/1246956/satisfiability-proof-of-formulas-with-pure-literals

Block 3

(a) We have to show that the conclusion $\{\, Inv \,\}\,\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\,\{\, Inv \wedge \neg e \,\}$ is true whenever the premise $\{\, Inv \,\}\, p \,\{\, Inv \,\}$ is true, for all formulas Inv, e and programs p. This can be done by deriving the conclusion from the premise using rules that are already known to be admissible, like the regular rules of Hoare calculus.

$$\dfrac{\dfrac{(Inv \wedge e) \Rightarrow Inv \quad \{\, Inv \,\}\, p \,\{\, Inv \,\}}{\{\, Inv \wedge e \,\}\, p \,\{\, Inv \,\}}\ \mathrm{lc}}{\{\, Inv \,\}\,\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\,\{\, Inv \wedge \neg e \,\}}\ \mathrm{wh}$$

The formula $(Inv \wedge e) \Rightarrow Inv$ is a tautology. Therefore, by the correctness of the Hoare calculus, the conclusion $\{\, Inv \,\}\,\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\,\{\, Inv \wedge \neg e \,\}$ is true whenever the premise $\{\, Inv \,\}\, p \,\{\, Inv \,\}$ is true.

(b) To show the incompleteness of the modified Hoare calculus we give a counter-example, which consists of a concrete correctness assertion that is true with respect to partial correctness, but which cannot be derived in the modified calculus. Consider the assertion

$$\{\, x \geq 0 \,\}\,\mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\,\{\, x \geq 0 \wedge x \not> 0 \,\}\ .$$

This assertion is true as can e.g. be shown by deriving it in the regular calculus:

$$\dfrac{\dfrac{\overset{\text{valid}}{x \geq 0 \wedge x > 0 \Rightarrow x - 1 \geq 0} \quad \{\, x - 1 \geq 0 \,\}\, x := x - 1 \,\{\, x \geq 0 \,\}^{\text{as}}}{\{\, x \geq 0 \wedge x > 0 \,\}\, x := x - 1 \,\{\, x \geq 0 \,\}}\ \mathrm{lc}}{\{\, x \geq 0 \,\}\,\mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\,\{\, x \geq 0 \wedge x \not> 0 \,\}}\ \mathrm{wh}$$

But this assertion cannot be derived in the modified calculus as we will show by contradiction. Suppose it can be derived. Then the derivation must have the form

$$\dfrac{x \geq 0 \Rightarrow F \quad \dfrac{\dfrac{\text{some derivation of } \{F\}x := x-1\{F\}}{\{F\}\, x := x-1\, \{F\}}}{\{F\}\,\text{while } x > 0 \text{ do } x := x-1 \text{ od}\,\{F \wedge x \not> 0\}}\ \text{mw} \quad (F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)}{\{x \geq 0\}\,\text{while } x > 0 \text{ do } x := x-1 \text{ od}\,\{x \geq 0 \wedge x \not> 0\}}\ \text{lc}$$

for some suitable invariant F. Note that lc and mw are the only rules that can have our counter-example as conclusion. Moreover, mw has to be applied once since it is the only rule that can introduce a while statement. (In fact, the logical consequence rule can be applied several times, but the effect of several applications can always be achieved with just one application.)

Now, if such a derivation indeed exists, then the formulas $x \geq 0 \Rightarrow F$ and $(F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)$ are valid and the correctness assertion $\{F\}\, x := x-1\, \{F\}$ is true (since it can be derived). We show that this cannot happen simultaneously, hence the derivation does not exist. This means that the calculus is incomplete, since we have found a true correctness assertion that cannot be derived.

Consider a state σ with $\sigma(x) = 0$. Since $x \geq 0 \Rightarrow F$ is supposed to be valid and $x \geq 0$ is true for σ, F must also be true for σ. Since $\{F\}\, x := x-1\, \{F\}$ is true, we conclude that F is also true for σ', where $\sigma'(x) = -1$ (state after executing the assignment). But the implication $(F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)$ does not hold for σ': The premise is true since $\sigma'(x) \not> 0$, but the conclusion $x \geq 0 \wedge x \not> 0$ is not true, since σ' does not satisfy $x \geq 0$.

Block 4

added by student

a) H={(s0,t0),(s1,t3),(s2,t3),(s3,t2),(s4,t2),(s5,t2)}

b)

i) EG(a): CTL+CTL*: ~~s0,s1,s2,s3,s4~~ S0,S2,S3

ii) EX(b): CTL+CTL*: S2,S3,S4

iii) X(c): LTL+CTL*: S2,S3

iv) F(a): LTL+CTL*: s0,s1,s2,s3

c) $\widehat{M}$ is basically an existential abstraction of M, with $h = L$. From such reduced models, we know that $M \leq M^r$, using the simulation relation $H = \{s, h(s) \mid s \in S\}$, if (!) M and M^r are defined over the same abstract atomic propositions (which in this case, they are). By the theorem of Logic Preservation on reduced models, we know that, for every ACTL* formula $M^r \vDash \varphi \Rightarrow M \vDash \varphi$.

+done. But I suppose it's not that hard to show that H is indeed a simulation relation from M to $\widehat{M}$. WDYT?

Exam 30.01.2015 (FMI.151.pdf)

http://www.logic.at/lvas/185291/angaben/fmi151.pdf

Block 1

1.) Consider the following 2 problems:

3-COLORABILITY (3-COL)

INSTANCE: An undirected graph $G = (V, E)$.

QUESTION: Does G have a *3-coloring*? That is, does there exist a function μ from vertices in V to values in $\{1, 2, 3\}$ such that $\mu(v_1) \neq \mu(v_2)$ for any edge $[v_1, v_2] \in E$.

UNDIRECTED GRAPH HOMOMORPHISM (HOM)

INSTANCE: A pair (G_1, G_2), where $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are undirected graphs.

QUESTION: Does there exist a homomorphism from G_1 to G_2? That is, does there exist a function h from vertices in V_1 to vertices in V_2 such that: for any edge $[v_1, v_2] \in E_1$ we also have $[h(v_1), h(v_2)] \in E_2$?

We provide next a reduction from **3-COL** to **HOM**. Let $G = (V, E)$ be an arbitrary undirected graph (i.e. an arbitrary instance of **3-COL**). From G we construct a pair (G_1, G_2) of undirected graphs. We let $G_1 = G$ and let $G_2 = (V_2, E_2)$ be as follows:

- $V_2 = \{v_1, v_2, v_3\}$, and
- E_2 consists of exactly the 3 (undirected) edges $[v_1, v_2]$, $[v_2, v_3]$ and $[v_1, v_3]$.

Task: Prove the "$\Leftarrow$" direction in the proof of correctness of the reduction, i.e. prove the following statement: if (G_1, G_2) is a positive instance of **HOM**, then G is a positive instance of **3-COL**.

Note: For any property that you use in your proof, make it perfectly clear why this property holds (e.g., "by the problem reduction", "by the assumption X", "by the definition X", etc.)

Assume (G1, G2) is a positive instance of HOM, s.t. we have a function h from vertices in V1 to vertices in V2 s. t. for any edge [v1, v2] of E1 we also have [h(v1), h(v2)] of E2 -> G(V, E) is a positive instance of 3-COL.
We provide a proof by contradiction.

Let [v1, v2] of E of G(V, E) be an arbitrary edge. By construction of (G1, G2) we know that that edge does exist in G1 of (G1, G2) of HOM. We further know, by assumption that (G1, G2) is a pos. instance of HOM, that there exists an edge in E2 of G2 s. t. [h(v1), h(v2)]. We further assume that v1 and v2 from V (= [v1, v2] of E) of G = G1 have same coloring, s.t. my(v1) = my(v2) (e.g. 1 = 1). That would mean that h(v1) = h(v2) and that there exists an edge [h(v1), h(v2)] in G2. But by construction of E2 such an edge [h(v1), h(v2)] where h(v1) = h(v2) can never exist (self-loop).

Comment to solution above:
I think you should also explicitly define μ in the certificate proof. I would define it as follows: For $v \in V_1$, let $\mu(v) = c_i$ iff $h(v) = v_i$ (where $i \in \{1, 2, 3\}$ and c_i represents the i-th color). It remains to show that μ is indeed a valid 3-COL mapping, i.e. for all $(v, v') \in E_1$ $\mu(v) \neq \mu(v')$ has to hold. Assume this were not the case, i.e. $\mu(v) = \mu(v')$, then by the construction of E_2 *and the definition of HOM*. μ , $h(v) = h(v')$ must also hold. This can easily be disproved by the construction of
I think it's more formal this way.

Block 2

a)see slides theory definition

b)

by student, not complete

Basically we can try to prove by contradiction using axioms. We want to prove that ϕ = p(a,b) -> f(a,b) ≠ a ∧ f(a,b) ≠ b ∧ a ≠ b, so we try to contradict it

1. I not ⊧ ϕ (our contradiction assumption)
2. I ⊧ p(a,b) (comes from the implication of 1.)
3. I not ⊧ f(a,b) ≠ a ∧ f(a,b) ≠ b ∧ a ≠ b (implication from 1.)
4. I ⊧ p(a, f(a,b)) ∧ p(f(a,b),b) (the first axiom given, from 2.)
5. I ⊧ p(a, f(a,b)) (conjunction from 4.)
6. I ⊧ p(f(a,b),b) (conjunction from 4.)
7. I ⊧ a ≠ f(a,b) (the second given axiom, from 5.)
8. I ⊧ f(a,b) ≠ b (the second given axiom, from 6.)
9. I ⊧ a ≠ b (the second given axiom, from 2.)
10. I ⊧ f(a,b) ≠ a (symmetry, from 7.)
11. I ⊧ f(a,b) ≠ a ∧ f(a,b) ≠ b ∧ a ≠ b (?? not sure about this step, but basically when I entails 8., 9., and 10., i can simply conjugate it as all those 3 things are entailed and thus true anyway?? Basically this would lead to the contradiction, as before we said that this is not entailed. So our proof by contradiction is successful and we have shown the proof?)

Alternative solution: Direct proof

Solved by student

1. $I \vDash p(a,b)$ premise
2. $I \vDash a \neq b$ 1,p-irr.
3. $I \vDash p(a, f(a,b))$ 1,p-density (for any f)
4. $I \vDash p(f(a,b), b)$ 1,p-density
5. $I \vDash a \neq f(a,b)$ 3,p-irr.
6. $I \vDash f(a,b) \neq a$ 5,symmetry of ≠
7. $I \vDash f(a,b) \neq b$ 4,p-irr.
8. $I \vDash f(a,b) \neq a \wedge f(a,b) \neq b \wedge a \neq b$ 2,6,7 and semantics of ∧

c)

Substitute p(.,.) by H1(.,.)=x1.

$flat^E$:= h1 = x1 ∧ f3 = g2 → h2 = x2
FC^E := b=c → f1=f2 ∧
b=f2 → f1=f3 ∧
c=f2 → f2=f3 ∧
b=g1 → g1=g2 ∧
(a=a ∧ f1=c) → h1=h2

φ^E: $FC^E \rightarrow flat^E$

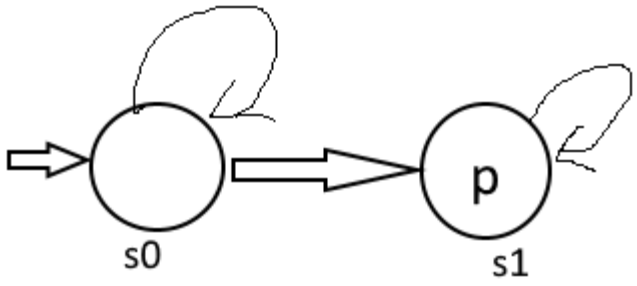
Block 3

Not done yet

Block 4

by student, not sure if understood correctly

a) By student please comment, what you think about the solution



AGEFp - Holds because from all paths, p is reachable

A(GFp->GFq) - the premise is false, therefore implication holds. It is not true that AGFp. Not for all paths, p will hold in the future i.e path s0,s0,s0,s0,s0

AG(p->AFq) - obviously does not hold

b)

```
int i;
//guessing coloring
for (i=0; i<N; i++) {
      coloring[i] = nondet_int() % 3; //(guess number of color-0,1,2)
}
int j;
int invalid_colors_found=false;
//checking if the colors for neighbors are different
for (i=0; i<N; i++) {
      for (j=0; j<N; j++) {
            if (graph[i][j] && coloring[i]==coloring[j]) {
                  invalid_colors_found=true;
            }
      }
}
assert(!invalid_colors_found);
```

c)

Proof by Contradiction:

We define H_1 as the simulation relation for $K_1 <= K_2$, H_2 for $K_2 <= K_3$ and H_3 for $K_1 <= K_3$.

Furthermore I use s1 ∈ S1, s2 ∈ S2 and s3 ∈ S3.

Assume K1 <= K3 Does not hold. Therefore one out of 3 cases is true:

- Case 1: ∃ (s1,s3) ∈ H3 where L1(s1) = L3(s3) does not hold. This means either ∃ (s1, s2) ∈ H1 with L1(s1) != L2(s2) or ∃ (s2, s3) ∈ H2 with L2(s1) != L3(s2). Since this contradicts the assumption that K1 <= K2 and K2 <= K3, this case is false.
- Case 2: ∃ (s1,t1) ∈ R1 and (s3,t3) ∈ R3 where (t1, t3) ∈ H3 does not hold. This means either ∃ (s1,t1) ∈ R1 and (s2,t2) ∈ R2 where (t1, t2) ∈ H2 does not hold or ∃ (s2,t2) ∈ R2 and (s3,t3) ∈ R3 where (t2, t3) ∈ H2 does not hold. This also contradicts our assumption and is therefore false.
- Case 3: ∃ s1 ∈ I1 and s3 ∈ I3 where (s1, s3) ∈ H3 does not hold. This means either ∃ s1 ∈ I1 and s2 ∈ I2 where (s1, s2) ∈ H1 does not hold or ∃ s2 ∈ I2 and s3 ∈ I3 where (s2, s3) ∈ H2 does not hold. This again contradicts our assumption and therefore also false.

Since all 3 cases are false and contradict the assumption that K1 <= K2 and K2 <= K3, the assumption “K1 <= K2 does not hold” is false and therefore
K1 <= K2 and K2 <= K3 implies K1 <= K3 holds.

Alternative solution to c)
$K_1 = (S_1, R_1, L_1)$, $K_2 = (S_2, R_2, L_2)$, $K_3 = (S_3, R_3, L_3)$

We know that $K_1 \leq K_2$ and that $K_2 \leq K_3$. Therefore, there exist simulation relations H_1 and H_2 for both ≤ relations. $H_1 \subseteq S_1 \times S_2$ and $H_2 \subseteq S_2 \times S_3$. To show: We can build a $H_3 \subseteq S_1 \times S_3$ that proofs $K_1 \leq K_3$.

In H_1 we have, $L_1(s_1)=L_2(s_2)$ and in H_2 $L_2(s_2)=L_3(s_3)$. Therefore, $L_1(s_1)=L_3(s_3)$. In other words, the H_3 makes only valid connections. That is, the labels of the states K_1 are connected to labels in K_3 where the same propositions hold.

For all (s_1,t_1) in R_1, there exist (s_2,t_2) in R_2 such that (t_1,t_2) in H_1
For all (s_2,t_2) in R_2, there exist (s_3,t_3) in R_3 such that (t_2,t_3) in H_2
Therefore, for all (s_1,t_1) in R_1, there exist (s_3,t_3) in R_3.

And the same tick, we can do with the initial states.

Exam 05.12.2014 (FMI.146.pdf)

http://www.logic.at/lvas/185291/angaben/fmi146.pdf

Block 1

Let $x = (\Pi_1, \Pi_2, I)$ be an arbitrary instance of BOTH-HALTS, then we construct $R(x) = (\Pi, I')$ as follows:

We set $I' = I$, and construct Π as a composition of Π_1 and Π_2 as follows: $\Pi(I)\{\ \Pi_1(I);\ \Pi_2(I)\ \}$

We claim that: Π_1 halts on I and Π_2 halts on $I \Leftrightarrow \Pi$ halts on I

- " ⇒ " Suppose Π_1 halts on I and Π_2 halts on I: by our construction of Π, Π_2 is executed after Π_1 halts on I, and Π halts afterΠ_2 halts on I⇒ Π halts on I
- " ⇐ " Suppose Π halts on I: by our construction of Π, Π can only halt on I if Π_2 halts on I, which can only be the case if first Π_1 halts on I ⇒ Π_1 halts on I and Π_2 halts on I

Block 2

(A) by student (not sure if its enough)

Not valid, because there is one interpretation which makes it false:

Take F(a) as a * -1, you'll see that -1*-1*-1*a = -1*a & -1*-1*a = a is true but -1*a /= a

What about this approach (omitted the ( ) because of lazyness, FFFa = F(F(F(a)))):

1. I |= FFFa = Fa & FFa = a -> Fa = a
2. I |= FFFa != Fa or FFa != a or Fa = a (-> from 1.)
3. I |/= FFFa = Fa & FFa = a & Fa != a (negate the entailment from 2.)
4. I |/= FFFa = Fa (& from 3)
5. I |/= FFa = a (& from 3)
6. I |/= Fa != a (& from 3)
7. I |= Fa = a (& from 3)
8. I |= FFa = Fa (function consistency on 7., if Fa = a, FFa must be Fa)
9. I |= FFa != a (negate 5)
10. I |= FFa != Fa (insert Fa = a from 7. Into 9.)
11. I |= False (as we have a contradiction between 8. And 10.)

And so we have proven that this contradicts. A Interpretation for this is stated above, but now we additionally proved it formally and not just by example.

(B) by student

First replace the predicates by a function including equality to a newly introduced term variable, e.g. p(a, (F(b)) becomes P(a, F(b)) = d1(d1 being the new term variable)

ϕEUF = P(a, F(b)) = d1 ∧ F(F(c)) = G(G(b)) ⇒ P(a, c) = d2

Then, enumerate our functions

P1(a, F1(b)) = d1 ∧ F3(F2(c)) = G2(G1(b)) ⇒ P2(a, c) = d2

To simplify it and compute flatE, we assume the term variables for the functions we introduce now are called the same, just in lower case (e.g. P1(a, F1(b)) becomes p1). So we can compute flatE as follows:

flatE(ϕEUF) = p1 = d1 ∧ f3 = g2 ⇒ p2 = d2

And at last we have to introduce the functional contraints FC again, you do this for each function symbol separately. You find the rules for the computation in tuwel, supplementary resources for block 2, ackermanns_bryants_reduction_B.pdf

FC[EF](ϕEUF) = (b = c ⇒ f1 = f2) ∧ (b = f2 ⇒ f1 = f3) ∧ (c = f2 ⇒ f2 = f3)
FC[EG](ϕEUF) = (b = g1 ⇒ g1 = g2)
FC[EP](ϕEUF) = (a = a ∧ f1 = c ⇒ p1 = p2)

The final solution of the reduction is ϕE = FC[EF](ϕEUF) ∧ FC[EG](ϕEUF) ∧ FC[EP](ϕEUF) ⇒ flatE(ϕEUF)

(C) by student
Both I and M provide semantics for ϕ, but an interpretation I is called “model” iff it satisfies ϕ. Interpretations may or may not satisfy ϕ

Block 3

a) added by student - please comment, not sure if it’s enough!

Show that the forward and backward reasoning axioms are equivalent

1. we assume that {G[v/e]}v=e{G} is correct and we have to show that we come up with a postcondition {G} if we apply the forward reasoning formula. After applying that formula, we come up with a postcondition like {Ev’(G[v/e][v/v’] & v=e[v/v’]}. After applying those replacements, we come up with a G’ where each v is replaced by an altered e, concretely where each v is replaced by v’ - like in the other conjunction. Therefore e’=[v/v’] and {Ev’(G[v/e’] & v=e’}. After applying the replacement we come up with a formula G’=G[v/e’] and therefore {Ev’(G’ & v=e’}. Because of the equivalence v=e’ we can apply another replacements, namely G’[e’/v] is equivalent to G[v/e’][e’/v] which is again equivalent to G.

2. we assume that {F}v=e{Ev’(F[v/v’] & v=e[v/v’]} is correct and show that we also come up with a state {G[v/e]} if the postcondition is declared as {G} and apply the

backward-reasoning rule. After assuming that backward reasoning is also correct, we come up with a precondition that looks like {Ev'(F[v/v'] & v=e[v/v'])[v/e]}. Again we can construct a F'=F[v/v'] and therefore no v will occur in F'. Applying [v/e] on F' won't change F'. So we come up with a formula {Ev'(F' & e=e[v/v']}. Since e=e[v/v'], v=v' must hold. We now can replace v' by v in F' (since they are equal) and come up with a F'[v'/v] which is equivalent to F[v/v'][v'/v] which is equivalent to F.

a) **Alternate solution see [above]**

b) missing, please add

b) added by student

```
F0: {0 <= a} from upwards logic equivalence
F1: {1 = 1^3 & 0 <= a} from upwards ass
b := 1;
F2: {b = (0+1)^3 & 0 <= 0^3 <= a} from upwards ass
c := 0;
F3: {b = (c+1)^3 & 0 <= c^3 <= a} from while
while b <= a do
      F4: {b = (c+1)^3 & 0 <= c^3 <= a & b <= a & a-b = t₀)} from while
      F5: {F6[d/3 * c + 6]} from upwards ass
      d := 3 * c + 6;
      F6: {F7[c/c + 1]} from upwards ass
      c := c + 1;
      F7: {F8[b / b + c * d + 1]} from upwards ass
      b := b + c * d + 1
      F8: {b = (c+1)^3 & 0 <= c^3 <= a & (b <= a -> 0 <= a-b < t₀)} from
while
od
F9: {Inv & b > a} from while
F10: {TRUE} // since we want to know for which states it terminates
```

F9 ⇒ F10 trivially holds

F5:= F8[c/c + 1][d/3 * c + 6][b / b + c * d + 1[c/c + 1][d/3 * c + 6]]

F5:= {b = $(c+2)^3$ & 0 <= $(c + 1)^3$ <= a & (b <= a -> 0 <= a-b < t_0)}[b / b + (c + 1) * (3 * c + 6) + 1]

F5:= {b = $(c+2)^3$ & 0 <= $(c + 1)^3$ <= a & (b <= a -> 0 <= a-b < t_0)}[b / b + $3*c^2$ + 3*c + 6 * c + 6 + 1]

F5:= {b = $(c+2)^3$ & 0 <= $(c + 1)^3$ <= a & (b <= a -> 0 <= a-b < t_0)}[b / b + $3*c^2$ + 9 * c + 7]

F5:= {b + $3*c^2$ + 9 * c + 7 = $(c+2)^3$ & 0 <= $(c + 1)^3$ <= a & (b + $3*c^2$ + 9 * c + 7 <= a -> 0 <= a - (b + $3*c^2$ + 9 * c + 7) < t_0)}

F4 ⇒ F5 iff all of the below hold

1. $F4 \Rightarrow b + 3*c^2 + 9 * c + 7 = (c+2)^3$
2. $F4 \Rightarrow 0 <= (c + 1)^3$
3. $F4 \Rightarrow (c + 1)^3 <= a$
4. $F4 \Rightarrow (b + 3*c^2 + 9 * c + 7 <= a \rightarrow 0 <= a - (b + 3*c^2 + 9 * c + 7) < t_0)$

1) holds because from F4 $b = (c+1)^3 \rightarrow c^3 + 3c^2 + 3c + 1 + 3*c^2 + 9 * c + 7 = (c+2)^3$ which trivially is true

2) holds because $0 <= c^3$ from F4

3) from F4: $b = (c+1)^3$ and $b <= a$ therefore $(c + 1)^3 <= a$ holds

4) $F4 \ \& \ b + 3*c^2 + 9 * c + 7 <= a \rightarrow (0 <= a - (b + 3*c^2 + 9 * c + 7) < t_0)$

from proof at 1 we know that $b + 3*c^2 + 9 * c + 7 = (c + 2)^3$

$F4 \ \& \ (c + 2)^3 <= a \Rightarrow (0 <= a - (c + 2)^3 < t_0)$

$F4 \ \& \ (c + 2)^3 <= a \Rightarrow (0 <= a - (c + 2)^3 \ \& \ a - (c + 2)^3 < t_0)$

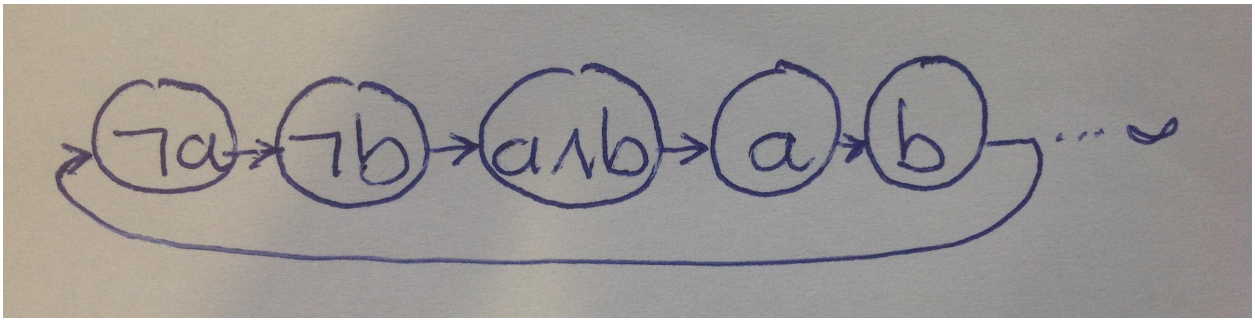
$F4 \ \& \ (c + 2)^3 <= a \Rightarrow (a - (c + 2)^3 < a - b)$

$F4 \ \& \ (c + 2)^3 <= a \Rightarrow ((c + 2)^3 > b)$ holds because of $b = (c+1)^3$ from F4

Block 4

added by student

a) H = {(s0,t0),(~~s1,t4~~),(s1,t5),(s2,t1),(s2,t4),(s3,t2),(s4,t2)}

b) 

So the same structure repeats infinitely… (as we only use LTL here)

Other Solution for b)

(a) ----> (b) -----> (a,b) --|
 ^ |
 |--------------------------|

(C)
Look at the slides, 03Abstraction.pdf , you will find an example there where the one simulates the other and vice versa, but they are not a bisimulation

The “intuitive” way to think about it if you look at the example in the pdf:
If you take e.g. b from M1, you can go either to d or c. Now on M2, if you are on the left b, then you can only go to c, but not to d. So this is not equal! In a bisimulation basically you have to think about it “two-way”.
But simulation is possible: As simulation is only viewed “one way”. So b from M1 can be simulated by the right b of M2.
And the other way around, both b from M2 can be simulated with b from M1.
But both things at the same time are not possible as stated before.

Exam 17.10.2014 (FMI.145.pdf)

http://www.logic.at/lvas/185291/angaben/fmi145.pdf

Block 1

The same exercise as in exam 151.
Variation of the exam from 30.01.2015

Block 2

a)
i) x = y & x != y
ii) x = y | x !=y

b)
Prove that the following formula ϕ is TE-cons-valid:
ϕ : ¬atom(x) ∧ car (x) = y ∧ cdr (x) = z → x = cons(y, z)
Hints: Recall the axiom of construction in TE-cons : **¬atom(x)** → cons(car (x), cdr (x)) = x

Proof by contradiction:
Suppose there exists an I s.t. I $\not\models$ ϕ

1. $I \models \phi$
2. $I \models \neg atom(x) \land car(x) = y \land cdr(x) = z$ // 1., semantics of $\rightarrow$
3. $I \models x = cons(y, z)$ // 1., semantics of $\rightarrow$
4. $I \models$ **$\neg atom(x)$** // 2., semantics of $\land$
5. $I \models car(x) = y$ // 2., semantics of $\land$
6. $I \models cdr(x) = z$ // 2., semantics of $\land$
7. $I \models cons(y, z) = x$ // 4., Axiom of construction and 5. + 6.
8. $I \models x = cons(y, z)$ // 7., symmetry law
9. $I \models \bot$ // 3.,8., contradiction

From the proof we know there does not exist an I s.t. $I \models \phi$.
Therefore $I \models \phi$ holds for all I. Thus ϕ is TE-cons-valid.

c)

Block 3

Block 4

added by student

a) H = { (s0,t0),(s1,t4),(s1,t6),(s2,t1),(s2,t3),(s3,t4),(s3,t6)}

b)

i) Fc: t3,t1

ii) G(b v c): none, because you can always reach a state which is labelled only a

iii) G(Fb): all; isn't this similar to Fc? but yeah

iv) G(b->(Xa->Xb)): ~~all?~~

By Student: All is not correct, because we have 3 cases (we have to analyse these cases for each possible path => AG(b->(Xa->Xb))):

1) We are in a state that is not **b**, so the implication holds
2) We are in a state that is **b** and the next state is **a** then if the next state is also **b** (so {**a,b**}) the implications hold
3) We are in a state that is **b** and the next state is not **a** then trivially both implications hold.

This results in the following states: ~~**All except T6**~~ because there exists a path where the next state is a, but on this single path the same next state is not labelled b.

I just realized I forgot something there! Because the implication has to hold globally and because in T6 it does not hold, it fails to hold as well for all

states from which T6 can be reached. New Solution would be therefore **none**, as T6 can be reached from all states.

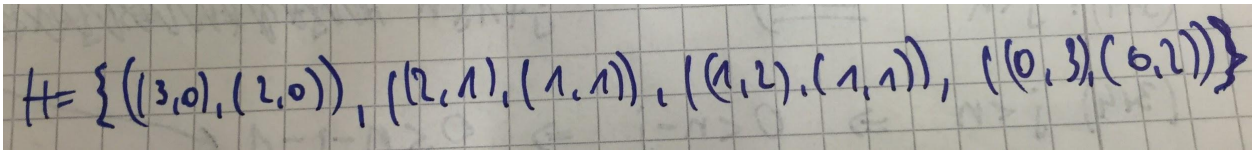
v) aU(bUc): t5, t3,t1

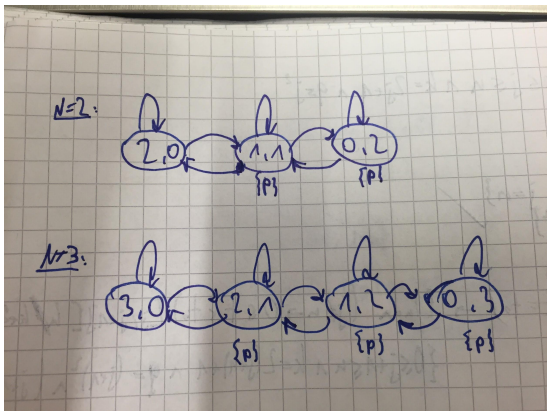
c) I'd choose (n,c) arbitrarily from S3 and show that for (n,c) and every possible following state a corresponding ((n,c),(n',c')) exists in H. Since (n,c) was chosen arbitrarily, this must hold for every state (and by design therefore also for every initial state) and hence M3 ≤ M2:

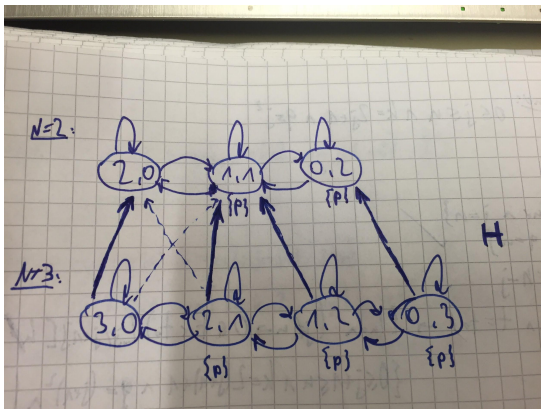
i) (n,c): ((n,c),(min(n,1),min(c,1))) ∈ H
ii) (n+1,c-1): ((n+1,c-1),(min(n,1)+1,min(c,1)-1)) ∈ H
iii) (n-1,c+1): ((n-1,c+1),(min(n,1)-1,min(c,1)+1)) ∈ H
iv) (n+0,c-0): ((n+0,c-0),(min(n,1)+0,min(c,1)-0)) ∈ H

c) Other solution (by student)

I just drew both Kripke structures, and then argued that the three properties of a simulation H (labels, transitions, starting states) are fulfilled. Here are my drawings.







Exam 04.07.2014 (FMI.144.pdf)

http://www.logic.at/lvas/185291/angaben/fmi144.pdf

Block 1

1.) Consider the following problem:

> **GRAPH-FORMULA (GF)**
>
> INSTANCE: A pair (G, φ), where G is an undirected graph and φ is a propositional formula.
>
> QUESTION: Is it the case that G is 3-colorable or φ is satisfiable?

By providing a reduction from an NP-complete problem, prove that **GF** is an NP-hard problem. Argue formally that your reduction is correct. **(15 points)**

We have chosen the problem: k-COLORABILITY for any k ≥ 3 and we fix the k to 3.

Therefore our problem we provide the reduction from is the 3-COL problem.

Let G' (V', E') be an arbitrary instance of 3-COL and my a valid 3-colorability function.
We construct an instance of GF as follow:

- G = G'
- ϕ = false

->
Assume G' (V', E') is a positive instance of 3-COL. By definition our graph G' is 3-colored. By construction of G of(G, ϕ) of GF from G' we know that G = G'. Therefore G is also 3-colored. Hence by definition of the problem of GF (G, ϕ) is a positive instance of GF.

<-
Assume GF (G, ϕ) is a positive instance of GF. By definition G is either 3-colorable or ϕ is satisfiable. We further assume that (G, ϕ) is a positive instance of GF because G is 3-colorable. By construction of G we know that G' is also 3-colorable. Hence G' (V', E') is a positive instance of 3-COL.

<- *Alternative:*
Assume GF (G, ϕ) is a positive instance of GF. By definition G is either 3-colorable or ϕ is satisfiable. Since by the reduction ϕ is set to false, ϕ can never be SAT. Therefore it must be the case that G is 3-COL and so also G' is 3-COL. Hence G' (V', E') is a positive instance of 3-COL.

Block 2

a) Solution by student

I constructed the graph, and recognized that every variable (node) is part of a simple contradictory cycle. So no further simplification is possible.
I added a single edge (x_1, x_3) to make the graph chordal.

The cycles are: x1,x2,x3 and x1,x3,x5 and x3,x4,x5, hence there are 9 'terms' in B_t.

$e(\varphi^E)$: $[(e_{1,5} \wedge e_{3,5}) \vee (\neg e_{1,2} \wedge e_{2,3})] \wedge (e_{4,5} \rightarrow \neg e_{3,4})$
B_t:
$(e_{1,2} \wedge e_{1,3} \rightarrow e_{2,3}) \wedge (e_{1,2} \wedge e_{2,3} \rightarrow e_{1,3}) \wedge (e_{1,3} \wedge e_{2,3} \rightarrow e_{1,2}) \wedge$
$(e_{1,3} \wedge e_{1,5} \rightarrow e_{3,5}) \wedge (e_{1,5} \wedge e_{3,5} \rightarrow e_{1,3}) \wedge (e_{1,3} \wedge e_{3,5} \rightarrow e_{1,5}) \wedge$
$(e_{3,4} \wedge e_{3,5} \rightarrow e_{4,5}) \wedge (e_{3,4} \wedge e_{4,5} \rightarrow e_{3,5}) \wedge (e_{4,5} \wedge e_{3,5} \rightarrow e_{3,4})$

ψ^E: $e(\varphi^E) \wedge B_t$

b) Solution by student

Let I be any interpretation of F. Since F is valid, I is a model.
Since F and F' differ only by one clause, we only have to prove that this one clause becomes true under I.

I makes all clauses C_i in F true, therefore it makes C_1 and C_2 true. 2 cases:
1) I assigns true to A_i: A_i in C_2 is false. Since I makes C_2 true, there has to be at least one other variable in C_2, that is assigned the value true by I.
2) I assigns false to A_i: A_i in C_1 is false. Since I makes C_1 true, there has to be at least one other variable in C_1, that is assigned the value true by I.

The clause res(C_1, C_2, A_i) contains variables from both C_1 and C_2. In any case, res(C_1, C_2, A_i) contains at least one variable (either from C_2 or C_1) that is set to true.
Therefore, I $\vDash$ res(C_1, C_2, A_i).
And finally, I $\vDash$ F'.

Block 3

a

We have to show that the conclusion { F } if e then p else q fi { G } is true whenever the premise
{ F } p { G } { F $\wedge$ ¬e } q { G } is true, for all formulas e and programs p, q.

This can be done by deriving the conclusion from the premise using rules that are already known to be admissible, like the regular rules of Hoare calculus.

$$\dfrac{\dfrac{F \wedge e \Rightarrow F \quad \{F\}p\{G\}}{\{F \wedge e\}p\{G\}}\,(\text{ic}) \qquad \{F \wedge \neg e\}p\{G\}}{\{F\}\,\text{if } e \text{ then } p \text{ else } q \text{ fi}\,\{G\}}\,(\text{if})$$

The premise F ∧ e ⇒ F is a tautology, hence the total correctness of
{ F } p { G } and { F ∧ ¬e } q { G } implies the total correctness of { F } if e then p else q fi { G }.

3b) modified if-rule

(b) Show that the Hoare calculus for partial correctness is no longer complete, if we replace the regular if-rule by the modified one. **(10 points)**

To show the incompleteness of the modified Hoare calculus we give a counterexample, which consists of a concrete correctness assertion that is true with respect to partial correctness, but which cannot be derived in the modified calculus.
Consider the assertion

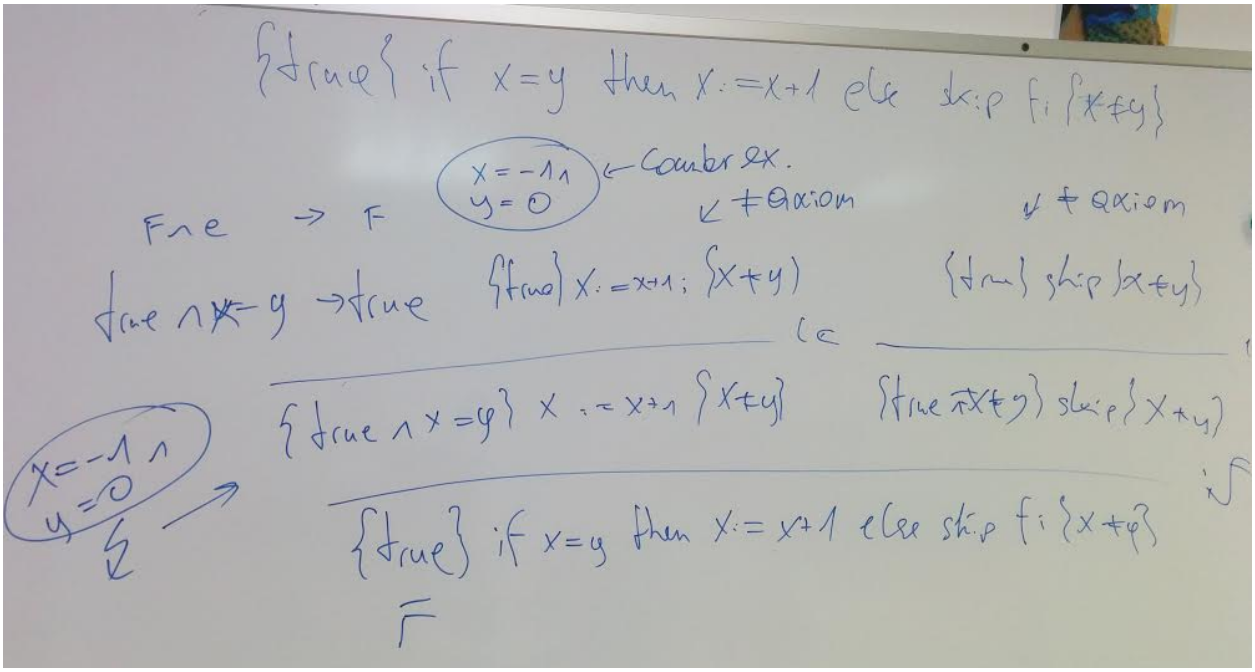
{ true } if x = y then x := x + 1 else skip fi { x != y } .

The assertion is partially and totally correct, since we have:

$\{1\colon \text{true}\}$
if $x = y$ then
  $\{7\colon \text{true} \wedge x = y\}$ if↓
  $\{6\colon x + 1 \neq y\}$ as↑
  $x := x + 1$
  $\{4\colon x \neq y\}$ fi↑
else
  $\{8\colon \text{true} \wedge x \neq y\}$ if↓
  $\{5\colon x \neq y\}$ sk↑
  skip
  $\{3\colon x \neq y\}$ fi↑
fi
$\{2\colon x \neq y\}$

and the two implications true ∧ x = y ⇒ x + 1 != y (7 ⇒ 6) and true ∧ x 6 = y ⇒ x != y (8 ⇒ 5) are valid.

In the modified calculus only two rules are applicable to the assertion above: if' and lc. Rule lc weakens the pre- and strengthens the postcondition. The precondition true cannot be weakened any further; Then if' has to be applied, resulting in the premises { true } x := x + 1 { G } cannot be derived by a correct calculus.



Summarising, the assertion above is correct, but cannot be derived in the modified Hoare calculus. Therefore the calculus is not complete.

Block 4

(a)

H = {(s0, t0), (s1, t1), (s2, t3), (s3, t3), (s4, t1), (s0, t2)}

(b)

(i) s0, s1, s2, s3
(ii) s0, s1, s2, s3 … alternative solution: in s0 always no prev state (verified with kripke)
(iii) s1, s2, s3, s4
(iv) s2, s4

Equivalent LTL formula for G(a -> Yb): G(Xa -> b) & !a
(G(Xa -> b) for reversing the formula. !a because Yb is always false in the first state, and thus, if the first state contains a, then the formula is automatically false).

Exam 09.05.2014 (FMI.143.pdf)

http://www.logic.at/lvas/185291/angaben/fmi143.pdf

Block 1

Let (Π', I') be a arbitrary instance of HALTING. We construct an instance of SOME-HALTS as follows:

- Π1 = Π2 = Π'
- I = I'

->

Assume (Π', I') is a positive instance of HALTING. For any input I', Π' halts on I'. By construction of SOME-HALTS we have that Π1 = Π2 = Π' and I = I'. Because Π' halts on any I' also Π1 and Π2 halt on I. Hence (Π1, Π2, I) is a positive instance of SOME-HALTS.

<-

Assume (Π1, Π2, I) is a positive instance of SOME-HALTS. That means Π1 halts on I or Π2 halts on I. By construction we know that Π' = Π1 = Π2 and I' = I. That means, because (Π1, Π2, I) is a positive instance and either Π1 or Π2 or both halt on I also Π' has to halt on I'. Hence (Π', I') is a positive instance of HALTING.

Block 2

2a)

(a) First define the concept of a $\mathcal{T}$-interpretation. Then use it to define the following:

i. the $\mathcal{T}$-satisfiability of a formula;

ii. the $\mathcal{T}$-validity of a formula.

Additionally define the completeness of a theory $\mathcal{T}$ and give an example for a complete theory and an incomplete theory. **(5 points)**

Definition

A theory $\mathcal{T} = (\Sigma, \mathcal{A})$ is

1. complete, if for every closed Σ-formula φ, $\mathcal{T} \models \varphi$ or $\mathcal{T} \models \neg\varphi$;
2. consistent, if there is at least one $\mathcal{T}$-interpretation;
3. decidable, if $\mathcal{T} \models \varphi$ is decidable for every Σ-formula φ.

Formulas φ_1 and φ_2 are equivalent in $\mathcal{T}$ or $\mathcal{T}$-equivalent if $\mathcal{T} \models \varphi_1 \leftrightarrow \varphi_2$, i.e., $I \models \varphi_1$ iff $I \models \varphi_2$ holds for all $\mathcal{T}$-interpretations I.

Example of a complete theory: Presburger arithmetic [link]

Example of an incomplete theory: group theory [link]

Definition

Given a theory $\mathcal{T} = (\Sigma, \mathcal{A})$.

1. A $\mathcal{T}$-interpretation I is an interpretation which satisfies $\mathcal{T}$'s axioms, i.e.,
$$I \models \varphi \qquad \text{for all } \varphi \in \mathcal{A}$$
2. A Σ-formula φ is valid in the theory $\mathcal{T}$, or $\mathcal{T}$-valid, if every $\mathcal{T}$-interpretation satisfies φ. Notation: $\mathcal{T} \models \varphi$
3. A Σ-formula φ is satisfiable in the theory $\mathcal{T}$, or $\mathcal{T}$-satisfiable, if some $\mathcal{T}$-interpretation satisfies φ.

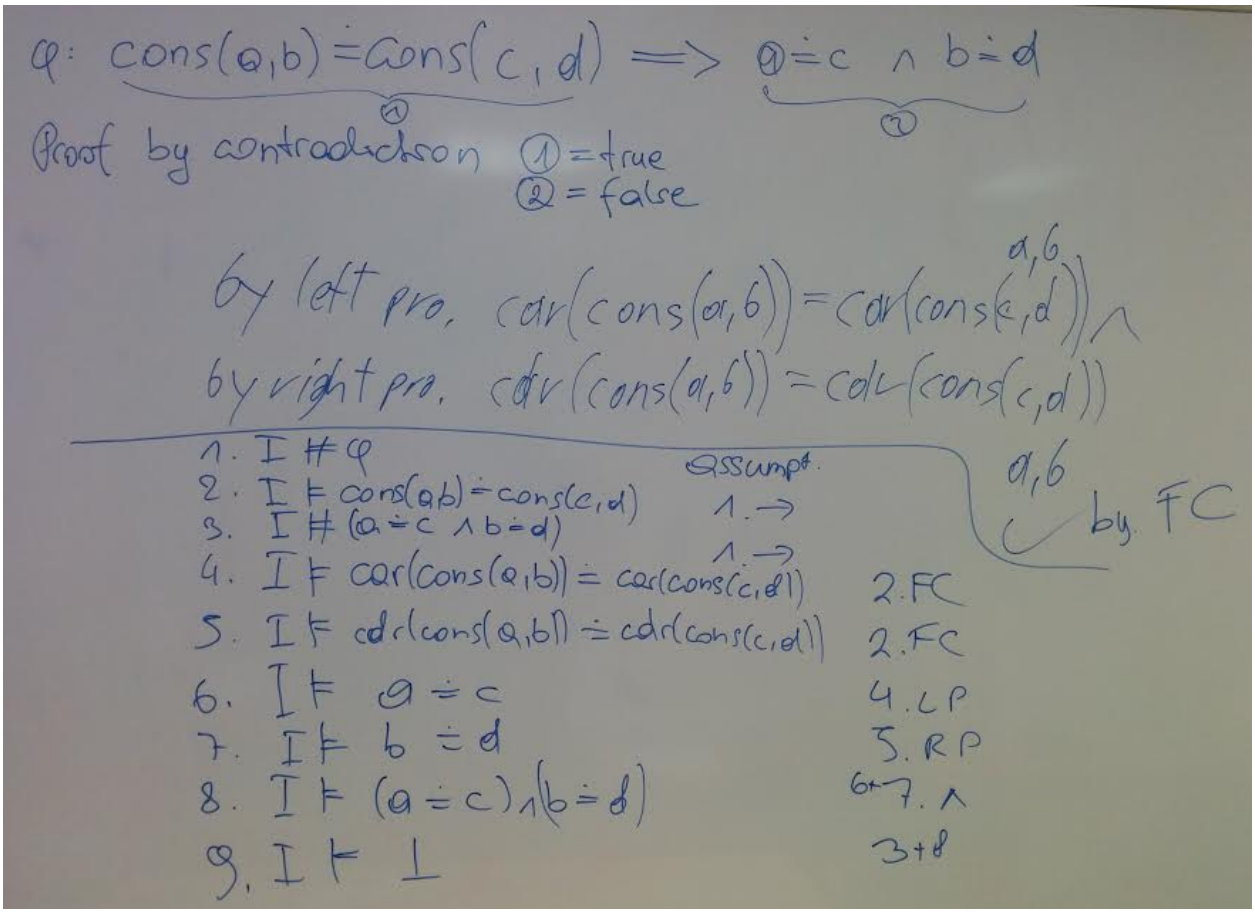
2b)

(b) Prove that the following formula φ is $\mathcal{T}^E_{cons}$-valid:

$$\varphi: \quad cons(a,b) \doteq cons(c,d) \rightarrow a \doteq c \wedge b \doteq d$$

Hints: Please be precise! Recall the axioms of left and right projection in $\mathcal{T}^E_{cons}$:

$$car(cons(x,y)) \doteq x \qquad \text{(left projection)}$$
$$cdr(cons(x,y)) \doteq y \qquad \text{(right projection)}$$



The assumption is false: ϕ is therefore T^E_{cons}-valid

Block 3

by r0f1 (comments welcome)

1(b) 3)

1: $\{k \geq 0\}$
$l := 1$
2: $\{k \geq 0 \wedge l = 1\}$ ↓as
$m := 0$
3: $\{k \geq 0 \wedge l = 1 \wedge m = 0\}$ ↓as

4: $\{Inv\}$ wh
while $l \leq k$ do
5: $\{Inv \wedge l \leq k \wedge t = t_0\}$ wh
11: $\{10[n/3m+6]\}$ as↑
$n = 3m + 6$
10: $\{9[m/m+1]\}$ as↑
$m = m+1$
9: $\{Inv \wedge (\ldots)[l/l+m\cdot n+1]\}$ as↑
$l := l + m \cdot n + 1$

6: $\{Inv \wedge (l \leq k \rightarrow 0 \leq t < t_0)\}$ wh

7: od $\{Inv \wedge l > k\}$ wh

8: $\{m^3 \leq k < (m+1)^3\}$

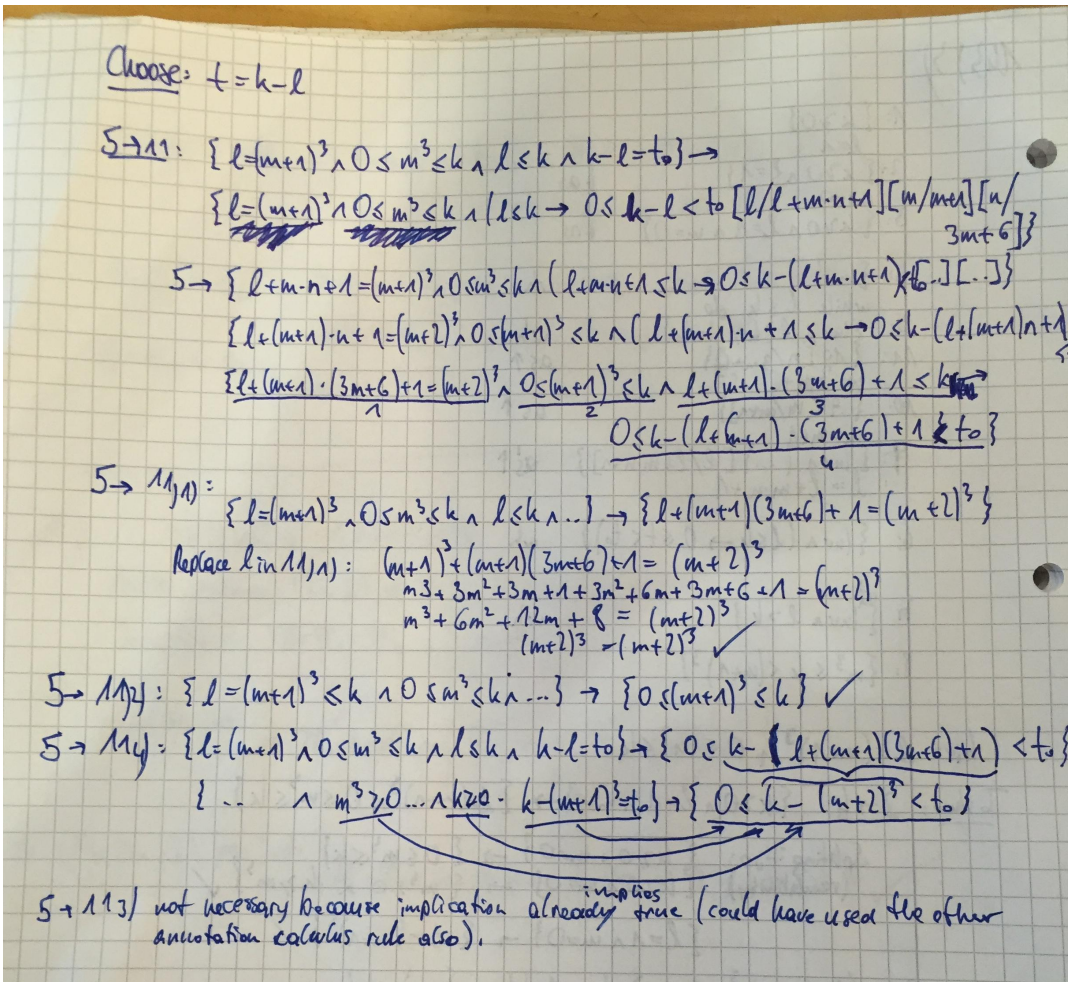
$Inv = l = (m+1)^3 \wedge 0 \leq m^3 \leq k$

To show: 3 → 4: $\{k \geq 0 \wedge l = 1 \wedge m = 0\} \rightarrow \{l = (m+1)^3 \wedge 0 \leq m^3 \leq k\}$

Splitting up: (readability)
$\{k \geq 0 \wedge m = 0\} \rightarrow \{0 \leq m^3 \leq k\}$
$\{k \geq 0 \wedge m = 0\} \rightarrow \{m^3 \geq 0 \wedge k \geq m^3\}$ ✓
$\{l = 1 \wedge m = 0\} \rightarrow \{l = (m+1)^3\}$ ✓

7 → 8a: $\{l = (m+1)^3 \wedge 0 \leq m^3 \leq k \wedge l > k\} \rightarrow \{m^3 \leq k\}$
7 → 8b:
implies ✓

$\{l = (m+1)^3 \wedge 0 \leq m^3 \leq k \wedge l > k\} \rightarrow \{k < (m+1)^3\}$
$\{k < l = (m+1)^3 \ldots\} \rightarrow \{k < (m+1)^3\}$ ✓



Block 4

a) Student: $K_2 \leq K_1$. $K_1 \leq K_2$ does not hold, because if you consider state s_2, you can decide between b and c. But there is no such a state in K_2

b)

c) s0–

Exam 28.03.2014 (FMI.142.pdf)

http://www.logic.at/lvas/185291/angaben/fmi142.pdf

Block 1

Solved by student

LATIN-SQUARE -> SAT is similar to the 3-COLORABILITY to SAT example from the lecture, only that we have *n* colors (symbols), and there are more constraints (not only adjacent nodes must have different symbols, but also all other cells in a **row** and **column**.)

Let $I_{LS} = (S = \{s_1,..., s_n\})$ be an arbitrary instance of LATIN-SQUARE, then we define $R(I_{LS})$ as follows:

We construct a formula φ_{LS} (an instance of SAT) as follows:

We use the propositional atoms $x_{ij}^{\ s}$, where $1 \leq i,j \leq n$and $s \in S$, to indicated that the cell at row i column j of L has the symbol s.

Then φ_{LS} is defined as $\varphi_{LS} = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$, where

(∩=∧, ∪=∨) because of Google's formula limitation

$\varphi_1 = \bigcap_{i=1}^{n} \bigcap_{j=1}^{n} \left(\bigcup_{s \in S} x_{ij}^{\ s} \right)$ (in other words: every cell has at least one symbol)

$\varphi_2 = \bigcap_{i=1}^{n} \bigcap_{j=1}^{n} \left(\bigcap_{s,s' \in S\ s \neq s'} \neg \left(x_{ij}^{\ s} \wedge x_{ij}^{\ s'} \right) \right)$ (in other words: every cell may have at most one symbol)

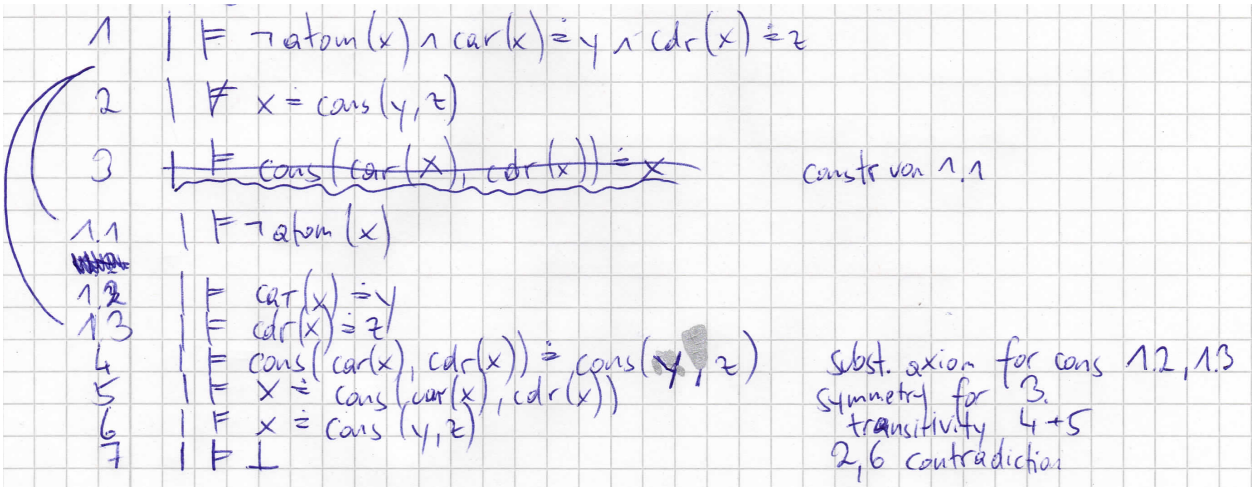
$\varphi_3 = \bigcap_{i=1}^{n} \bigcap_{j=1}^{n} \left(\bigcap_{s \in S} \bigcap_{k=1\ j \neq k}^{n} \neg (x_{ij}^{\ s} \wedge x_{ik}^{\ s}) \right)$(in other words, if a cell has a symbol, no other cells in that row may have the same symbol)

φ_4 is analogous to φ_3, only that i is permuted inside the parenthesis

I'm not 100% confident it's correct (or if it could be made simpler) but i think the intuition of breaking down each constraint into its own formula and then joining them is clear

Block 2

b) status student, pretty sure it's correct



Block 3

Block 4

a) Status: Student
   i.) G ¬ l5
   ii.) G(l2 → X l1)
   iii.) F l3 ∨ GF l1

Not really sure about b), c) and d): If the abstraction created in b) is able to simulate M, we need a transition to the error state in the abstraction. But then it is not possible to show that the abstraction satisfies the LTL formulae of a) (since the error location IS reachable). There has to be an error in my reasoning. Any ideas about this?

p = x≥0
q = y≥x
r = x≥100

b) 0:(all possible 8 predicate states) → 1: pq!r ← → 2: pq!r → 1: pqr → 3: pqr → 4: pqr

d) In my version the abstraction has no error state. So obviously it can't be reach as it doesn't exist anymore.
The theorem is: If M1 ≤ M2, M2 |= f -> M1 |= f, specifically, M2 (abstraction) |= G(!l5) -> M1 (original) |= G(!lhttp://www.logic.at/lvas/185291/angaben/fmi141.pdf5)

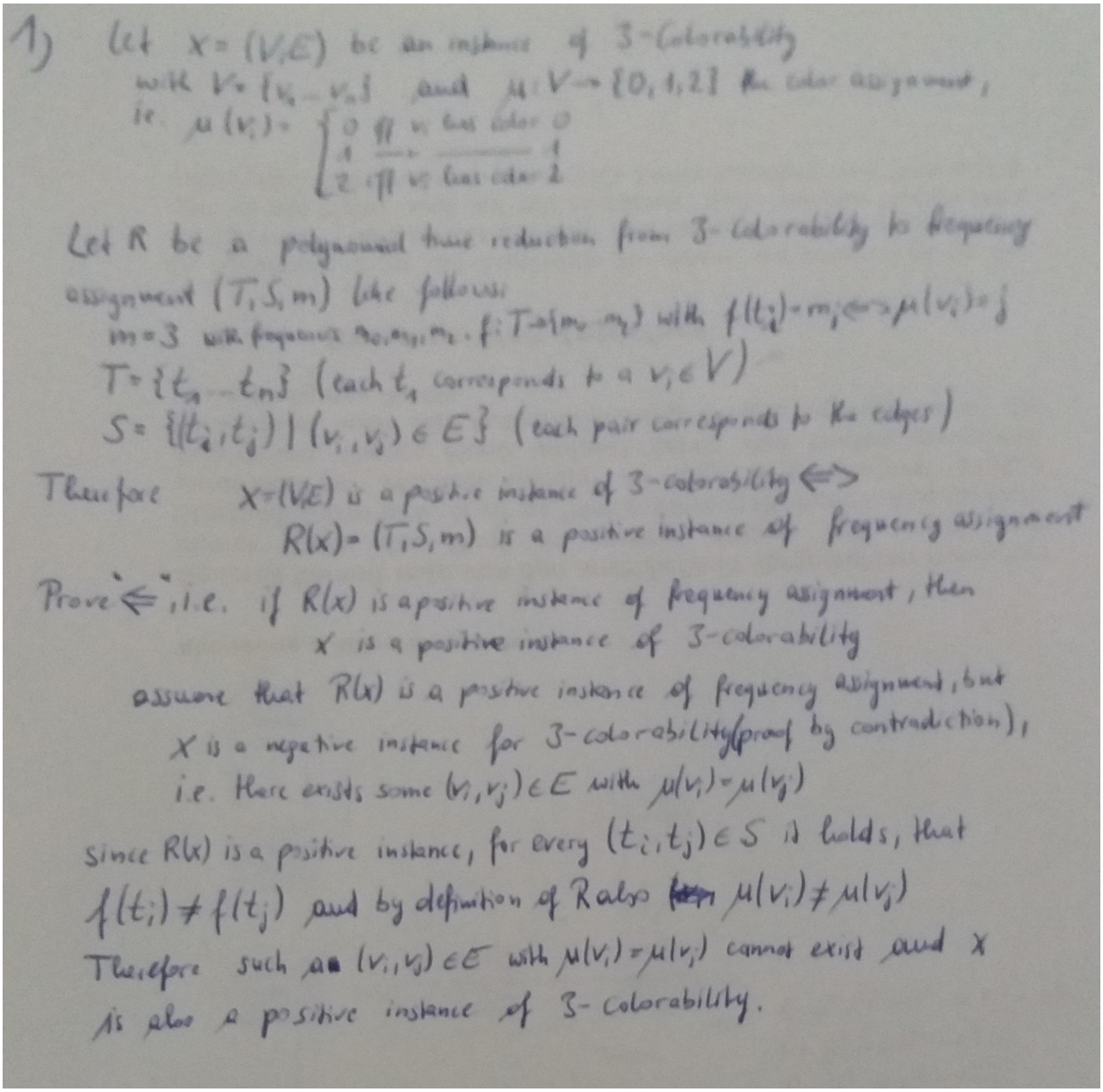
Status student:

c) $H = \{((0,x,y), s_i) \mid i \text{ in } \{0..7\}, ((1,x,y), s_8), ((2,x,y), s_9), ((1,x,y), s_{10}), ((3,x,y), s_{11}), ((4,x,y), s_{12}), ((5,x,y), s_{13})) \mid x,y \text{ in } Z\}$

Exam 31.01.2014 (FMI.141.pdf)

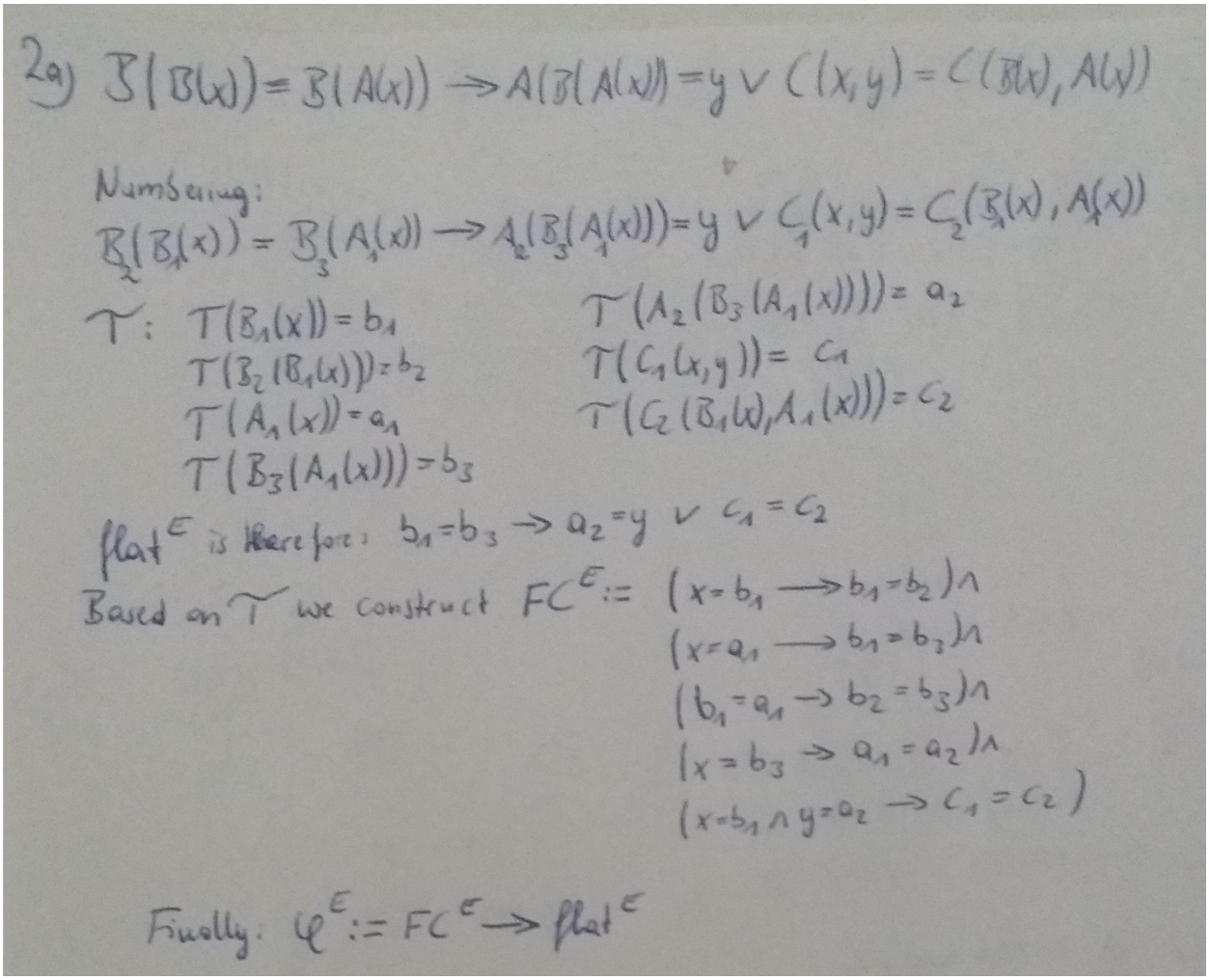
Block 1

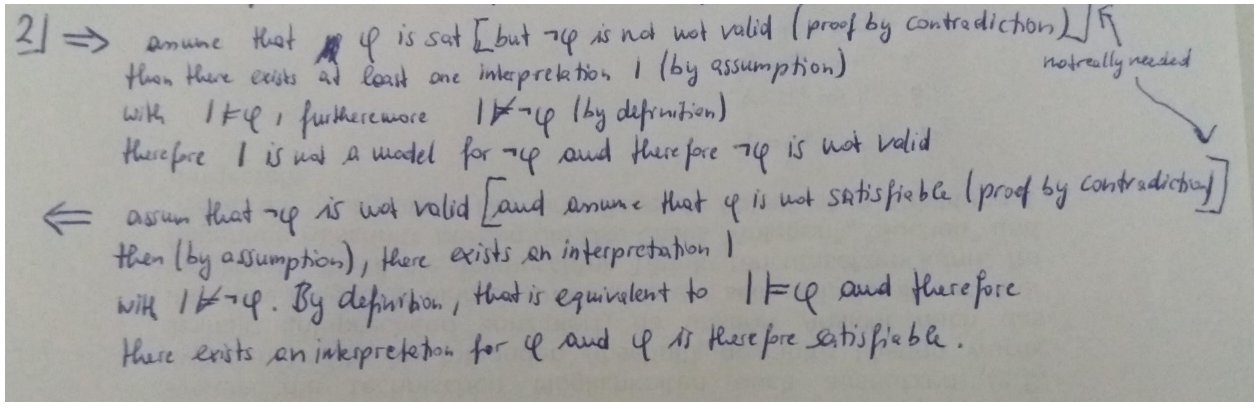
Status**: added by Student**



Block 2

Status**: added by Student**





Alternative solution to 2b)

⇒: Assume φ is satisfiable. This means, that there exists an interpretation I under which φ evaluates to *true*. That means, that under the same I, ¬φ evaluates to *false*. Therefore, there is an interpretation under which ¬φ is false, which is the definition of *¬φ is not valid*.

⇐: Assume ¬φ is not valid. Therefore, there exists an interpretation I that makes ¬φ *false*. Under I, φ evaluate to true. Therefore, φ is satisfiable.

Principle of structural induction

Proving properties about *app*

Let $\mathcal{P}(x)$ denote $app(x, nil) = x$.

Show that, for all lists ℓ, $\mathcal{P}(\ell)$ holds.

Proof: We proceed by structural induction on the definition of lists.

Basis case: $\ell = nil$. Then $\mathcal{P}(\ell)$ is $app(nil, nil) = nil$ which follows immediately from the first defining equality (1).

Induction hypothesis. Assume that $\mathcal{P}(\ell)$ holds for some list ℓ.

Principle of structural induction

Proving properties about *app* (cont'd)

Induction step. We have to establish $\mathcal{P}((c : \ell))$. The induction hypothesis gives us

$$app(\ell, nil) = \ell.$$

We concatenate c to the left on both sides resulting in

$$(c : app(\ell, nil)) = (c : \ell).$$

By the second defining equality (2), we get

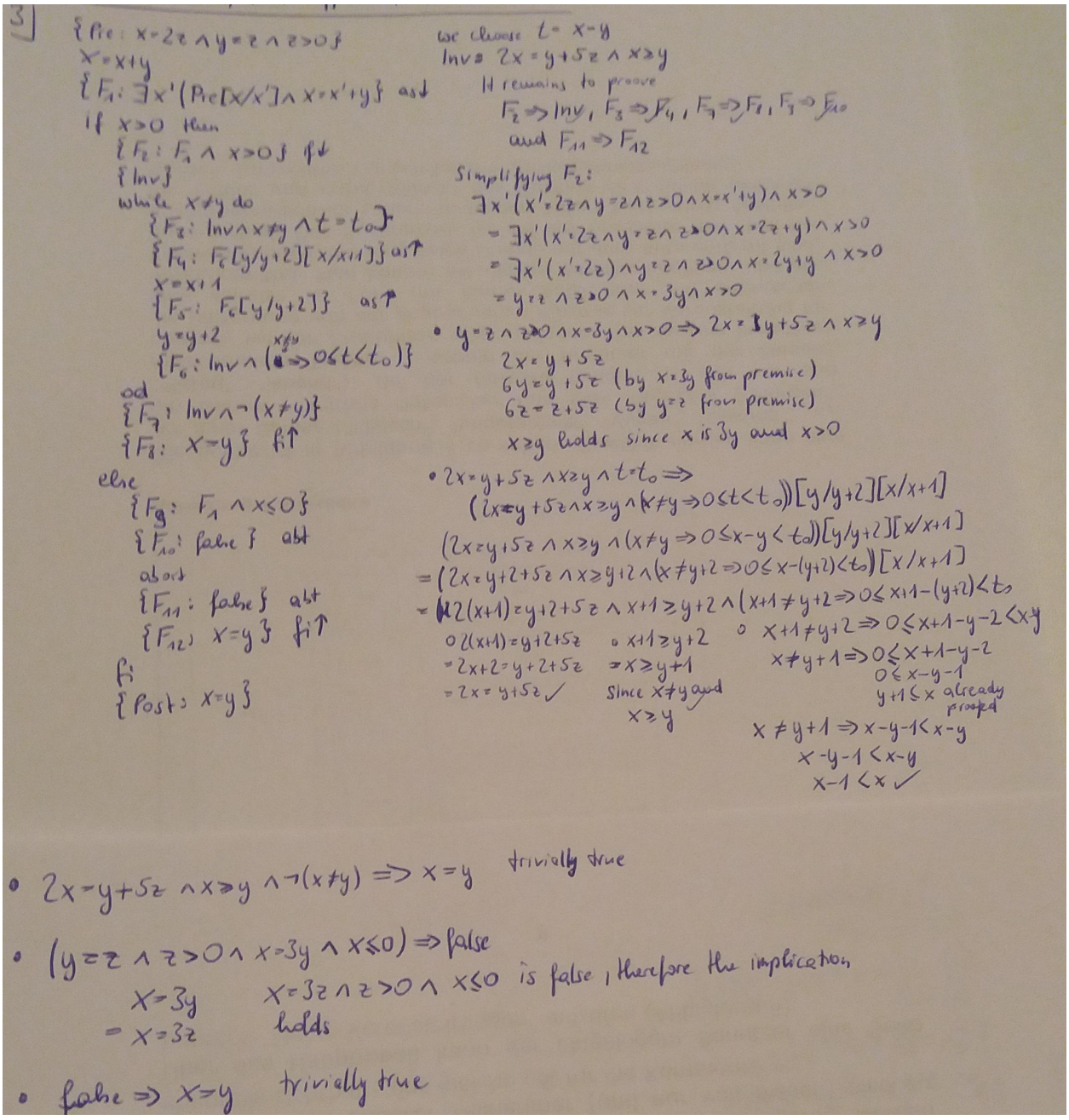
$$app((c : \ell), nil) = (c : \ell).$$

Hence, $\mathcal{P}((c : \ell))$ holds.
Consequently, $\mathcal{P}(\ell)$ is true for all lists. □

How can we use mathematical induction to prove the above result?

Block 3

Status: **added by Student**



Block 4

a) This proof is very similar to one of the proofs from the exercise, which I pasted below.

ROB

Exam 06.12.2013 (FMI.136.pdf)

https://www.logic.at/lvas/fminf/angaben/fmi136.pdf

Block 1

Solution by student:

To show: SOLVE-EQUATION is semi-decidable.

Let Z be the following set of numbers = {0,1,-1,2,-2,3,-3,...}. This is the set of integers $\mathbb{Z}$, which is countably-infinite, so we can enumerate all of its values. We provide a semi-decidability procedure:

1: for z in Z {
2: $x = \Pi_1(z)$;
3: $y = \Pi_2(z)$;
4: if (x == y) return true;
5: }

Let (Π_1,Π_2) be a positive instance of SOLVE-EQUATION. If we run the semi-decidability procedure we reach line 2, Π_1 will return (since this is guaranteed), then execution will reach line 3 and will also return and will finally it will reach line 4. Since (Π_1,Π_2) is a positive instance x and y will be the same and line 4 return true.

Let (Π_1,Π_2) be a negative instance of SOLVE-EQUATION. That is, it is impossible to find an integer, such that both programs return the same value. Our procedure will try number after number, and loop forever, never returning any value.

Summarizing, on positive instances we return true, on negative instances we loop forever. Therefore, this procedure is a semi-decidability procedure.

Block 2

2.) (a) Use Ackermann's reduction and translate

$$A(A(x)) \doteq A(B(x)) \rightarrow B(A(B(x))) \doteq y \vee C(x,y) \doteq C(A(x), B(x))$$

to a validity-equivalent E-formula φ^E. A, B, and C are function symbols, x and y are variables. **(4 points)**

(b) Show: φ is satisfiable iff $\neg\varphi$ is *not* valid. **(3 points)**

(c) Let φ^{uf} be an equality formula containing uninterpreted functions. Let $FC^E(\varphi^{uf})$ and $\mathit{flat}^E(\varphi^{uf})$ be obtained by Ackermann's reduction. Prove the following.

$$\varphi^{uf} \text{ is satisfiable} \quad \text{iff} \quad FC^E(\varphi^{uf}) \wedge \mathit{flat}^E(\varphi^{uf}) \text{ is satisfiable.}$$

Hints:
H1: φ^{uf} is valid iff $FC^E(\varphi^{uf}) \rightarrow \mathit{flat}^E(\varphi^{uf})$ is valid.
H2: $\mathit{flat}^E(\neg\varphi^{uf}) = \neg\mathit{flat}^E(\varphi^{uf})$.
H3: $FC^E(\varphi^{uf}) = FC^E(\neg\varphi^{uf})$.

(8 points)

Solution for 2.b:

Proposition 1.45 *Let ϕ be a formula of propositional logic. Then ϕ is satisfiable iff $\neg\phi$ is not valid.*

PROOF: First, assume that ϕ is satisfiable. By definition, there exists a valuation of ϕ in which ϕ evaluates to `T`; but that means that $\neg\phi$ evaluates to `F` for that same valuation. Thus, $\neg\phi$ cannot be valid.

Second, assume that $\neg\phi$ is not valid. Then there must be a valuation of $\neg\phi$ in which $\neg\phi$ evaluates to `F`. Thus, ϕ evaluates to `T` and is therefore satisfiable. (Note that the valuations of ϕ are exactly the valuations of $\neg\phi$.) □

Source: "Logic in Computerscience" by Huth

Block 3

Block 4