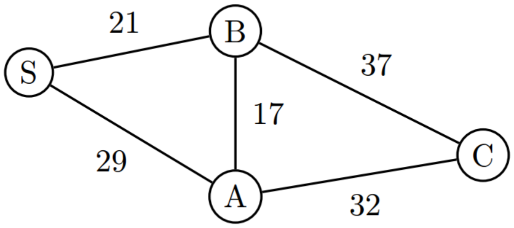
Practical Assignment Part 2
Automated Reasoning 2IMF25

Technische Universiteit Eindhoven
Jiahuan Zhang 0896785 (j.4.zhang@student.tue.nl)
Hector Joao Rivera Verduzco 0977393 (h.j.rivera.verduzco@student.tue.nl)

Problem 1

Three non-self-supporting villages A, B and C in the middle of nowhere consume one food package each per time unit. The required food packages are delivered by a truck, having a capacity of 300 food packages. The locations of the villages are given in the following picture, in which the numbers indicate the distance, more precisely, the number of time units the truck needs to travel from one village to another, including loading or delivering. The truck has to pick up its food packages at location S containing an unbounded supply. The villages only have a limited capacity to store food packages: for A and B this capacity is 120, for C it is 200. Initially, the truck is in S and is fully loaded, and in A, B and C there are 40, 30 and 145 food packages, respectively.



(a) Show that it is impossible to deliver food packages in such a way that each of the villages consumes one food package per time unit forever.

(b) Show that this is possible if the capacity of the truck is increased to 320 food packages. (Note that a finite graph contains an infinite path starting in a node v if and only if there is a path from v to a node w for which there is a non-empty path from w to itself.)

(c) Figure out whether it is possible if the capacity of the truck is set to 318.

Solution:

We generalize this problem for n number of villages and a truck with capacity T. We introduce $n \times (m+1)$ integer variables a_{ij} for $i = 1, ..., n$ and $j = 0, ..., m$, where m is the number of travels that the truck has performed, and a_{ij} represents the number of food packages in the village i after performing j number of travels. We also introduce $m+1$ integer variables p_j,

t_j and d_j for $j = 0, ..., m$, where p_j and t_j represent the position of the truck and the number of food packages in it respectively after j number of travels. Finally, d_j depicts the amount of food packages that are delivered to a village just after performing j number of travels.

First we define the boundaries of each variable. To do so, we define $max(i)$ as the maximum number of food packages that village i can store. Then we have the following formula that expresses that the number of food packages in each village should be in the range of zero and its store's capacity:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=0}^{m} 0 \leq a_{ij} \leq max(i).$$

Similarly, we construct the formula for the boundaries of the reminding variables.

$$\bigwedge_{j=0}^{m} 0 \leq p_j \leq n \ \wedge$$

$$\bigwedge_{j=0}^{m} 0 \leq t_j \leq T \ \wedge$$

$$\bigwedge_{j=0}^{m} 0 \leq d_j.$$

It is worth to mention that $p_j = 0$ means that the truck is in location S (Supplier), whereas $p_j = i$ for $i = 1, ..., n$ depicts that the truck is in village i. Hence, the boundaries of p_j must be in the range of zero and n.

The problem specifies the condition that initially the truck is fully loaded and in position S. Additionally, some villages already have an amount of food packages, lets define this initial number of packages as A_i for village i. Then we have the following formula:

$$p_0 = 0 \ \wedge$$

$$t_0 = T \ \wedge$$

$$\bigwedge_{i=0}^{n} a_{i0} = A_i.$$

Next, we express the condition that the number of delivered packages d in village i is limited by the capacity of this village. Also, when the truck is in position zero (Supplier) it should not deliver any package.

$$\bigwedge_{j=0}^{m} \bigwedge_{i=1}^{n} (p_j = i) \rightarrow (d_j \leq (max(i) - a_{ij})) \ \wedge$$

$$\bigwedge_{j=0}^{m} (p_j = 0) \rightarrow (d_j = 0).$$

Finally, when the truck is in a given position, the next position should be a neighboring village. Also, when moving to another position the amount of food in every village will decrement depending on the travel time. In order to express this conditions, we introduce $\mathbf{C}_i$ as the sets of neighbors of village i, and the mappings $f_i : \mathbf{C}_i \rightarrow \mathbf{N}$ to determine the time required to travel from i to one of its neighbor villages, eg. lets say that the set of neighbors of village 1 is $\mathbf{C}_1 = \{0, 2\}$ then villages 0 and 2 are connected to village 1, and $f_1(0)$ is the time required to go from village 1 to village 0.

Using all this elements, the formula that expresses this condition is the following:

$$\begin{aligned}\bigwedge_{j=0}^{m-1}\bigwedge_{i=0}^{n}(p_j = i) \rightarrow (\bigvee_{k\in\mathbf{C}_i} (p_{j+1} = k \ \ \wedge\ a_{ij+1} = a_{ij} + d_j - f_i(k) \ \wedge \\ \bigwedge_{1\le l\le n: i\neq l} a_{lj+1} = a_{lj} - f_i(k) \ \wedge \\ (k = 0) \rightarrow (t_{j+1} \geq t_j) \ \wedge \\ (k \neq 0) \rightarrow (t_{j+1} = t_j - d_{j+1}))).\end{aligned}$$

It is worth to mention that the last two expressions of this big formula express that when the selected neighbor is the Supplier ($k = 0$), then the next amount of packages in the truck t can only be bigger or equal to the previous one, because in this position the truck is being filled again. When $k \neq 0$ the packages in the truck will decrement since some packages will be delivered to the selected neighbor k.

The total formula now consists of the conjunction of all these ingredients. We can find a particular solution for this problem choosing $n = 3$, $T = 300$, $max(1) = max(2) = 120$, $max(3) = 200$, $A_1 = 40$, $A_2 = 30$, $A_3 = 145$, $\mathbf{C}_0 = \{1, 2\}$, $\mathbf{C}_1 = \{0, 2, 3\}$, $\mathbf{C}_2 = \{0, 1, 3\}$, $\mathbf{C}_3 = \{1, 2\}$ and the values of $f_i(k)$ as depicted in the picture of the villages.

The complete formula expressed in SMT syntax is as follow:

```
;Practical Assignment - Automated_ Reasoning 2IMF25
;Problem 1
(benchmark test.smt
:logic QF_UFLIA
:extrafuns
((a1_0 Int) (a2_0 Int) (a3_0 Int)
(a1_1 Int) (a2_1 Int) (a3_1 Int)
(a1_2 Int) (a2_2 Int) (a3_2 Int)
......
(p_0 Int) (t_0 Int) (d_0 Int)
(p_1 Int) (t_1 Int) (d_1 Int)
(p_2 Int) (t_2 Int) (d_2 Int)
......
)
:formula
(and
;the initial values for each village, the truck and position
(= p_0 0)
(= a1_0 40)
(= a2_0 30)
(= a3_0 145)
(= t_0 300)
;Bound of each variable
(>= a1_0 0) (<= a1_0 120) (>= a2_0 0) (<= a2_0 120) (>= a3_0 0) (<= a3_0 200)
(>= a1_1 0) (<= a1_1 120) (>= a2_1 0) (<= a2_1 120) (>= a3_1 0) (<= a3_1 200)
(>= a1_2 0) (<= a1_2 120) (>= a2_2 0) (<= a2_2 120) (>= a3_2 0) (<= a3_2 200)
......
(>= p_0 0) (<= p_0 3) (>= t_0 0) (<= t_0 300) (>= d_0 0)
(>= p_1 0) (<= p_1 3) (>= t_1 0) (<= t_1 300) (>= d_1 0)
(>= p_2 0) (<= p_2 3) (>= t_2 0) (<= t_2 300) (>= d_2 0)
......
;Step 1
(implies (= p_0 0) (and (= d_0 0)
(or (and (= p_1 1) (= a1_1 (- a1_0 29)) (= a2_1 (- a2_0 29)) (= a3_1 (- a3_0 29)) (= t_1 (- t_0 d_1)))
(and (= p_1 2) (= a1_1 (- a1_0 21)) (= a2_1 (- a2_0 21)) (= a3_1 (- a3_0 21)) (= t_1 (- t_0 d_1))))))

(implies (= p_0 1) (and (<= d_0 (- 120 a1_0))
(or (and (= p_1 0) (= a1_1 (- (+ a1_0 d_0) 29)) (= a2_1 (- a2_0 29)) (= a3_1 (- a3_0 29)) (>= t_1
t_0))
(and (= p_1 2) (= a1_1 (- (+ a1_0 d_0) 17)) (= a2_1 (- a2_0 17)) (= a3_1 (- a3_0 17)) (= t_1 (- t_0
d_1)))
(and (= p_1 3) (= a1_1 (- (+ a1_0 d_0) 32)) (= a2_1 (- a2_0 32)) (= a3_1 (- a3_0 32)) (= t_1 (- t_0
d_1))))))

(implies (= p_0 2) (and (<= d_0 (- 120 a2_0))
(or (and (= p_1 0) (= a1_1 (- a1_0 21)) (= a2_1 (- (+ a2_0 d_0) 21)) (= a3_1 (- a3_0 21)) (>= t_1
t_0))
(and (= p_1 1) (= a1_1 (- a1_0 17)) (= a2_1 (- (+ a2_0 d_0) 17)) (= a3_1 (- a3_0 17)) (= t_1 (- t_0
d_1)))
(and (= p_1 3) (= a1_1 (- a1_0 37)) (= a2_1 (- (+ a2_0 d_0) 37)) (= a3_1 (- a3_0 37)) (= t_1 (- t_0
d_1))))))

(implies (= p_0 3) (and (<= d_0 (- 200 a3_0))
(or (and (= p_1 1) (= a1_1 (- a1_0 32)) (= a2_1 (- a2_0 32)) (= a3_1 (- (+ a3_0 d_0) 32)) (= t_1 (-
t_0 d_1)))
(and (= p_1 2) (= a1_1 (- a1_0 37)) (= a2_1 (- a2_0 37)) (= a3_1 (- (+ a3_0 d_0) 37)) (= t_1 (- t_0
d_1))))))
......
))
```

Now we try to find a solution for each interrogant of the original problem:

(a) Show that it is impossible to deliver food packages in such a way that each of the villages consumes one food package per time unit forever.

Applying `yices-smt part2_1`a`.smt`, it yields SAT when choosing $m = 20$ but yields UNSAT when generating the code for $m = 21$, hence the truck can make at most 20 travels between villages before one consumes all its food. We conclude that is impossible to deliver food packages in such a way that each of the villages consumes one food package per time unit forever.

(b) Show that this is possible if the capacity of the truck is increased to 320 food packages.

In order to find a solution where the truck can deliver food forever, we first try to find a state that can be reached again in the future, so for this case the truck can perform the same route forever always passing for the same states. We find this adding conditions to the yices' code to compare two different states and yields SAT if they are equal. Performing some experiments we found out that after adding the following formula it yields satisfiable:

$$
\begin{gathered}
a_{1\_10} = a_{1\_3} \;\; \wedge \\
a_{2\_10} = a_{2\_3} \;\; \wedge \\
a_{3\_10} = a_{3\_3} \;\; \wedge \\
P_{\_10} = P_{\_3} \;\; \wedge \\
T_{\_10} = T_{\_3} \;\; \wedge \\
d_{\_10} = d_{\_3} \;\; \wedge
\end{gathered}
$$

Choosing $T = 320$ and applying `yices-smt -m part2_1b.smt` to the generated code, the tool yields the following result:

```
sat
(= t_0 320)
(= a1_0 40)
(= a2_0 30)
(= a3_0 145)
(= d_0 0)
(= p_0 0)
(= t_1 295)
(= a1_1 19)
(= a2_1 9)
(= a3_1 124)
(= d_1 25)
(= p_1 2)
(= t_2 177)
(= a1_2 2)
(= a2_2 17)
(= a3_2 107)
(= d_2 118)
(= p_2 1)
......
```

Hence we conclude that for the case when the truck has a capacity of 320 food packages, it is possible to deliver food in such a way that each village consumes food forever. The final result for this special case is depicted in the following table:

variables/state	0	1	2	3	4	5	6	7	8	9	10
Truck position (p)	S	B	A	B	S	A	C	B	S	A	B
Food's pkgs in truck (t)	320	295	177	58	320	253	67	0	296	177	58
Food's pkgs delivered (d)	0	25	118	119	0	67	186	67	0	119	119
Food's pkgs in village A ($a1$)	40	19	2	103	82	53	88	51	30	1	103
Food's pkgs in village B ($a2$)	30	9	17	0	98	69	37	0	46	17	0
Food's pkgs in village C ($a3$)	145	124	107	90	69	40	8	157	136	107	90

As can be observed, state 10 is exactly the same as state 3, therefor we have found a route that satisfies the requirement.

(c) Figure out whether it is possible if the capacity of the truck is set to 318.

Similarly, we prove that it is possible to deliver food forever for this case too. We generate the yices' code choosing $T = 318$ and adding the same extra condition as in b), after applying `yices-smt -m part2_1c.smt` it yields the following result:

```
sat
(= t_0 318)
(= a1_0 40)
(= a2_0 30)
(= a3_0 145)
(= d_0 0)
(= p_0 0)
(= t_1 293)
(= a1_1 19)
(= a2_1 9)
(= a3_1 124)
(= d_1 25)
(= p_1 2)
(= t_2 175)
(= a1_2 2)
(= a2_2 17)
(= a3_2 107)
(= d_2 118)
(= p_2 1)
......
```

The following table shows the solution for this problem:

variables/state	0	1	2	3	4	5	6	7	8	9	10
Truck position (p)	S	B	A	B	S	A	C	B	S	A	B
Food's pkgs in truck (t)	318	293	175	55	318	252	66	0	295	175	55
Food's pkgs delivered (d)	0	25	118	120	0	66	186	66	0	120	120
Food's pkgs in village A ($a1$)	40	19	2	103	82	53	87	50	29	0	103
Food's pkgs in village B ($a2$)	30	9	17	0	99	70	38	1	46	17	0
Food's pkgs in village C ($a3$)	145	124	107	90	69	40	8	157	136	107	90

Remark:

In order to make the code clear some variable renaming was necessary. For instance for representing village 1 after 10 steps, the notation $a1\_10$ was preferred over $a110$, the later can lead for multiple interpretations.

It is also worth to mention that we started solving this problem using NuSMV tool instead of yices, the advantage of NuSMV is that the code can be more easily expressed, however it was taking too long to solve each problem, so we decided to migrate to yices.

Generalization:

We solve this problem choosing $T = 300$, $T = 318$ and $T = 320$. But it would be interesting to know the result of other values of T. We found out that choosing $T = 316$ it is impossible to deliver food in such a way that all villages consume forever. For $T = 317$ it was tested using $m = 300$ (300 travels) and it was still satisfiable.

Problem 2

Three bottles can hold 144, 72 and 16 units (say, centiliters), respectively. Initially the first one contains 3 units of water, the others are empty. The following actions may be performed any number of times:

- One of the bottles is fully filled, at some water tap.
- One of the bottles is emptied.
- The content of one bottle is poured into another one. If it fits, then the full content is poured, otherwise the pouring stops when the other bottle is full.

a) Determine whether it is possible to arrive at a situation in which the first bottle contains 8 units and the second one contains 11 units. If so, give a scenario reaching this situation.

b) Do the same for the variant in which the second bottle is replaced by a bottle that can hold 80 units, and all the rest remains the same.

c) Do the same for the variant in which the third bottle is replaced by a bottle that can hold 28 units, and all the rest (including the capacity of 72 of the second bottle) remains the same.

Solution:

We generalize this problem for n number of bottles. First we introduce $n \times (m+1)$ integer variables of the type a_{ij} for $i = 1, ..., n$ and $j = 0, ..., m$ to represent the content of all bottles, where m is the total number of actions that are being performed. We also define $max(i)$ as the maximum amount of water that bottle i can hold, and A_i as the amount of water that bottle i initially contains.

First, we define the formula that expresses the boundaries of each variable. In other words, the minimum and maximum amount of water that a bottle can hold through all the steps, this is bounded by zero (empty bottle) and $max(i)$ (fully filled bottle):

$$\bigwedge_{j=0}^{m} \bigwedge_{i=1}^{n} 0 \leq a_{ij} \leq max(i).$$

The problem specifies that some bottles initially contain some amount of water. The following formula expresses this condition:

$$\bigwedge_{i=1}^{n} a_{i0} = A_i.$$

Next we express the steps or actions that can be performed. For the sake of simplicity we define that for all steps j it is only possible to perform one action at a time. Hence, we have the following formula:

$$\bigwedge_{j=0}^{m} Action1 \vee Action2 \vee Action3.$$

Now we construct the formulas that express each of the actions. For *action* 1 it is required that after one step, one of the bottles is fully filled and the other bottles remain with the same amount of water. The formula that expresses this is

$$\bigvee_{i=1}^{n}(a_{ij+1} = max(i) \wedge \bigwedge_{1 \leq k \leq n : k \neq i} a_{kj+1} = a_{kj}).$$

Similarly, the second action requires to empty one bottle after one step. We construct the formula for *action* 2 as follow:

$$\bigvee_{i=1}^{n}(a_{ij+1} = 0 \wedge \bigwedge_{1 \leq k \leq n : k \neq i} a_{kj+1} = a_{kj}).$$

Finally, *action* 3 specifies that the content of one bottle is poured into another one. If it fits, then the full content is poured, otherwise the pouring stops when the other bottle is full.

$$\bigvee_{i=1}^{n} \bigvee_{1 \leq k \leq n : k \neq i} ((a_{ij} \leq max(k) - a_{kj}) \rightarrow (a_{ij+1} = 0 \ \wedge \ a_{kj+1} = a_{kj} + a_{ij} \ \wedge$$

$$\bigwedge_{1 \leq l \leq n : l \neq i \wedge l \neq k} a_{lj+1} = a_{lj}) \wedge$$

$$\neg(a_{ij} \leq max(k) - a_{kj}) \rightarrow (a_{ij+1} = a_{ij} + a_{kj} - max(k) \wedge a_{kj+1} = max(k) \wedge$$

$$\bigwedge_{1 \leq l \leq n : l \neq i \wedge l \neq k} a_{lj+1} = a_{lj})).$$

At first sight, this big formula may seem hard to understand, but it is very straightforward. It basically states that you can select a bottle i to be poured into another bottle k ($k \neq i$), if it fits ($a_{ij} \leq max(k) - a_{kj}$) then the content of i will be added to the content of k in the next step ($a_{kj+1} = a_{kj} + a_{ij}$), whereas the content of bottle i will be empty. The other variables will remain unchange. If the content does not fit, then the bottle k will be filled to the top ($a_{kj+1} = max(k)$), while in bottle i will remain the amount of water that does not fit into k this is $a_{ij+1} = a_{ij} + a_{kj} - max(k)$.

The total formula now consists of the conjunction of all these ingredients.

$$\bigwedge_{j=0}^{m} \bigwedge_{i=1}^{n} 0 \leq a_{ij} \leq max(i) \wedge$$

$$\bigwedge_{i=1}^{n} a_{i0} = A_i \wedge$$

$$\bigwedge_{j=0}^{m} Action1 \vee Action2 \vee Action3.$$

The complete formula expressed in NuSMV syntax choosing $n = 3$, $max(1) = 144$, $max(2) = 72$, $max(3) = 16$, $A_1 = 3$ and $A_2 = A_3 = 0$ is as follow:

```
MODULE main
VAR
a1 :  0..144;
a2 :  0..72;
a3 :  0..16;
INIT
a1 = 3 & a2 = 0 & a3 = 0
TRANS
next(a1) = 144 & next(a2) = a2 & next(a3) = a3 |
next(a1) = a1 & next(a2) = 72 & next(a3) = a3 |
next(a1) = a1 & next(a2) = a2 & next(a3) = 16 |
next(a1) = 0 & next(a2) = a2 & next(a3) = a3 |
next(a1) = a1 & next(a2) = 0 & next(a3) = a3 |
next(a1) = a1 & next(a2) = a2 & next(a3) = 0 |

case (a1 <= (72 - a2)) :  next(a1) = 0 &
next(a2) = a2 + a1 &
next(a3) = a3;
TRUE : next(a1) = a1 - (72 - a2) &
next(a2) = 72 &
next(a3) = a3;

esac |

case (a1 <= (16 - a3)) :  next(a1) = 0 &
next(a2) = a2 &
next(a3) = a3 + a1;
TRUE : next(a1) = a1 - (16 - a3) &
next(a2) = a2 &
next(a3) = 16;

esac |

case (a2 <= (144 - a1)) :  next(a1) = a2 + a1 &
next(a2) = 0 &
next(a3) = a3;
TRUE : next(a1) = 144 &
next(a2) = a2 - (144 - a1) &
next(a3) = a3;

esac |
......

LTLSPEC G !(a1 = 8 & a2 = 11)
```

Now we try to find a solution for each interrogant of the original problem:

a) Determine whether it is possible to arrive at a situation in which the first bottle contains 8 units and the second one contains 11 units. If so, give a scenario reaching this situation.

In order to find a solution for this, we introduce the following LTL (Linear Temporal Logic) specification to the NuSMV code:

```
LTLSPEC G !(a1 = 8 & a2 = 11)
```

This states that globally the formula $a_1 = 8 \wedge a_2 = 2$ does not hold throughout all the possible reachable states. If it is possible to reach that condition then NuSMV will find a counterexample. After applying `NuSMV part2_2a.smv`, it yields the following result:

```
-- specification G !(a1 = 8 & a2 = 11) is false
-- as demonstrated by the following execution sequence
Trace Description:  LTL Counterexample
Trace Type:  Counterexample
-> State:  1.1 <-
a1 = 3
a2 = 0
a3 = 0
-> State:  1.2 <-
a2 = 72
-> State:  1.3 <-
a1 = 75
a2 = 0
......
-> State:  1.15 <-
a1 = 0
a2 = 11
-> State:  1.16 <-
a1 = 8
a3 = 0
-- Loop starts here
-> State:  1.17 <-
a1 = 19
a2 = 0
-> State:  1.18 <-
```

We can conclude from this result that it is possible to perform the actions to fill, to empty or to pour the content of the bottles in such a way that we reach the condition where the first bottle contains 8 units and the second one contains 11 units. This scenario is depicted in the following table:

variables/state	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Action		1	3	3	3	3	3	3	3	1	3	3	3	2	3	3
Content bottle 1 (a_1)	3	3	75	56	56	43	43	27	27	27	27	11	11	11	0	8
Content bottle 2 (a_2)	0	72	0	0	16	16	32	32	48	48	64	64	72	0	11	11
Content bottle 3 (a_3)	0	0	0	16	0	16	0	16	0	16	0	16	8	8	8	0

b) Do the same for the variant in which the second bottle is replaced by a bottle that can hold 80 units, and all the rest remains the same.

For this case, we generate the NuSMV code choosing $max(2) = 80$. Applying `NuSMV part2_2b.smv`, it yields the following result:

```
-- specification G !(a1 = 8 & a2 = 11) is true
```

It is not possible to reach an state where the condition $a1 = 8 \wedge a2 = 11$ holds, hence its impossible to arrive in a situation where the first bottle contains 8 units of water and the second one 11 units.

c) Do the same for the variant in which the third bottle is replaced by a bottle that can hold 28 units, and all the rest (including the capacity of 72 of the second bottle) remains the same.

Similarly to previous cases, we generate the NuSMV code choosing now $max(3) = 28$. We apply `NuSMV part2_2c.smv` yielding the following result:

```
-- specification G !(a1 = 8 & a2 = 11) is false
-- as demonstrated by the following execution sequence
Trace Description:  LTL Counterexample
Trace Type:  Counterexample
-> State:  1.1 <-
a1 = 3
a2 = 0
a3 = 0
-> State:  1.2 <-
a1 = 0
a2 = 3
-> State:  1.3 <-
a1 = 144
......
-> State:  1.25 <-
a2 = 11
a3 = 0
-> State:  1.26 <-
a1 = 36
a3 = 28
-> State:  1.27 <-
a3 = 0
-> State:  1.28 <-
a1 = 8
```

```
a3 = 28
-- Loop starts here
-> State:  1.29 <-
a1 = 19
a2 = 0
-> State:  1.30 <-
```

The specification is false and NuSMV gives a counterexample where the condition can be reached. We conclude that for this case it is possible to arrive to the situation specified by the problem. The following table depicts this situation in detail:

step	Action	a_1	a_2	a_3
0	Init	3	0	0
1	Pour $a1$ in $a2$	0	3	0
2	Fill $a1$	144	3	0
3	Pour $a1$ in $a2$	116	3	28
4	Pour $a3$ in $a2$	116	31	0
5	Pour $a1$ in $a3$	88	31	28
6	Pour $a3$ in $a2$	88	59	0
7	Pour $a1$ in $a3$	60	59	28
8	Pour $a3$ in $a2$	60	72	15
9	Pour $a2$ in $a1$	132	0	15
10	Pour $a3$ in $a2$	132	15	0
11	Pour $a1$ in $a3$	104	15	28
12	Pour $a3$ in $a2$	104	43	0
13	Pour $a1$ in $a3$	76	43	28
14	Pour $a3$ in $a2$	76	71	0
15	Pour $a1$ in $a3$	48	71	28
16	Pour $a3$ in $a2$	48	72	27
17	Pour $a2$ in $a1$	120	0	27
18	Pour $a3$ in $a2$	120	27	0
19	Pour $a1$ in $a3$	92	27	28
20	Pour $a3$ in $a2$	92	55	0
21	Pour $a1$ in $a3$	64	55	28
22	Pour $a3$ in $a2$	64	72	11
23	Empty $a2$	64	0	11
24	Pour $a3$ in $a2$	64	11	0
25	Pour $a1$ in $a3$	36	11	28
26	Empty $a3$	36	11	0
27	Pour $a1$ in $a3$	8	11	28

Remark:

In contrast to the previous problems, this time we use NuSMV to solve the three variations of this particular problem. Although this can also be expressed in SMT format and find a solution using yices, we decided to exploit the power of NuSMV in finding counterexamples, so the tool generates a solution automatically if the given specification does not hold. Also the code results to be more compact and understandable using this tool.

Generalization:

Since we generalize this problem for any number of bottles of any size, it would be interesting to find a size for bottle 1 in order to satisfy the subsection b) of this problem. We found out such case by reducing the capacity of bottle 1 just one unit, so choosing $max(1) = 143$ and $max(2) = 80$ the tool yields that the specification $G\, \neg(a1 = 8 \wedge a2 = 11)$ is false and it shows a solution where after performing 57 actions bottle 1 and 2 contain 8 and 11 units of water respectively.

Problem 3

The goal of this problem is to exploit the power of the recommended tools rather than elaborating the questions by hand

(a) In mathematics, a group is defined to be a set G with an element $I \in G$, a binary operator $*$ and a unary operator inv satisfying

$$x * (y * z) = (x * y) * z, x * I = x \text{ and } x * inv(x) = I,$$

for all $x, y, z \in G$. Determine whether in every group each of the four properties

$$I * x = x, inv(inv(x)) = x, inv(x) * x = I \text{ and } x * y = y * x$$

holds for all $x, y \in G$. If a property does not hold, determine the size of the smallest finite group for which it does not hold.

(b) A term rewrite system consists of the single rule

$$a(x, a(y, a(z, u))) \to a(y, a(z, a(x, u)))),$$

in which a is a binary symbol and x, y, z, u are variables. Moreover, there are constants b, c, d, e, f, g. Determine whether c and d may be swapped in $a(b, a(c, a(d, a(e, a(f, a(b, g))))))$ by rewriting, that is, $a(b, a(c, a(d, a(e, a(f, a(b, g))))))$ rewrites in a finite number of steps to $a(b, a(d, a(c, a(e, a(f, a(b, g))))))$.

Solution:

(a) In this problem, three assumptions are given. So we use `Prover9` to prove the four properties straightforwardly. The expressions in `Prover9` are as follows. Here we denote inv(x) as x' for the sake of simplicity.

```
formulas(assumptions).
% Group definition
x * I = x.
x * x’ = I.
x * (y * z) = (x * y) * z.
end_of_list.
formulas(goals).
I * x = x.
x’’ = x.
x’ * x = I.
x * y = y * x.
end_of_list.
```

After applying `prover9 -f part2_3a.in`, we found that the first 3 properties are proved, but the fourth one, which implies the property of commutativity, is failed. In order to determine the size of the smallest finite group for which it does not hold, we apply `mace4 part2_3a.in` to find the smallest noncommutative group by finding the counterexample to the statement that all groups are commutative. `Mace4` exits with 1 model. The model is as follows.

```
============================== DOMAIN SIZE 6 =========================
============================== MODEL =================================
interpretation( 6, [number=1, seconds=0], [
        function(I, [ 0 ]),
        function(c1, [ 1 ]),
        function(c2, [ 2 ]),
        function('(_), [ 0, 1, 2, 4, 3, 5 ]),
        function(*(_,_), [
               0, 1, 2, 3, 4, 5,
               1, 0, 3, 2, 5, 4,
               2, 4, 0, 5, 1, 3,
               3, 5, 1, 4, 0, 2,
               4, 2, 5, 0, 3, 1,
               5, 3, 4, 1, 2, 0 ])
]).
============================== end of model ==========================
```

`Mace4` generates the counterexample of a group of size 6, therefore, the size of the smallest finite group, for which $x * y = y * x$ does not hold, is 6.

(b) To determine the possibility of the rewriting in a finite number of steps, we use `Prover9` to prove the possibility.

This problem gives only a single rewriting rule, so the TRS R consists of the rule

$$a(x, a(y, a(z, u))) \to_R a(y, a(z, a(x, u)))),$$

The rewriting terms are the closed terms composed from the constants b, c, d, e, f, g. We want to show that the rewriting of

$$a(b, a(c, a(d, a(e, a(f, a(b, g)))))) \to a(b, a(d, a(c, a(e, a(f, a(b, g)))))).$$

can be done in zero or more rewrite steps.

Expressed in `Prover9`:

```
formulas(assumptions).
R(a(x,a(y,a(z,u))),a(y,a(z,a(x,u)))).
```

```
R(x,y) -> R(a(x,z),a(y,z)).
R(x,y) -> R(a(z,x),a(z,y)).
RR(x,x).
(RR(x,y) & R(y,z)) -> RR(x,z).
end_of_list.
formulas(goals).
RR(a(b,a(c,a(d,a(e,a(f,a(b,g)))))),a(b,a(d,a(c,a(e,a(f,a(b, g)))))))).
end_of_list.
```

In the codes, $a = a$ binary operator, $R =$ single rewrite step, and $RR =$ zero or more rewrite steps.

After running `prover9 -f part2_3b.in`, `Prover9` exits with 1 proof. Therefore, c and d can be swapped in $a(b, a(c, a(d, a(e, a(f, a(b, g))))))$ by rewriting in a finite number of steps to $a(b, a(d, a(c, a(e, a(f, a(b, g))))))$. The proof is in the following.

```
============================== PROOF =================================
% Proof 1 at 0.09 (+ 0.03) seconds.
% Length of proof is 22.
% Level of proof is 7.
% Maximum clause weight is 27.
% Given clauses 195.
2 R(x,y) -> R(a(z,x),a(z,y)) # label(non_clause).  [assumption].
3 RR(x,y) & R(y,z) -> RR(x,z) # label(non_clause).  [assumption].
4 RR(a(b,a(c,a(d,a(e,a(f,a(b,g)))))),a(b,a(d,a(c,a(e,a(f,a(b,g))))))) # label(non_clause)
# label(goal).
.........
800 -RR(a(b,a(c,a(d,a(e,a(f,a(b,g)))))),a(c,a(d,a(f,a(e,a(b,a(b,g))))))).  [ur(9,b,38,a,c,116,a)].
1604 RR(a(x,a(y,a(z,a(u,a(w,a(v5,v6)))))),a(y,a(z,a(w,a(u,a(v5,a(x,v6))))))).  [ur(9,a,319,a,b,170,a)].
1605 $F. [resolve(1604,a,800,a)].
============================== end of proof ==========================
```

Remark:

Since Problem $3a$ gives the rewriting rules and the goals explicitly, its expression in `Prover9` also goes straightforwardly. The searching for a finite model with the last property, which is failed in the proof of `Prover9`, is done by the tool `Mace4`. Problem $3b$ requires the definition of the term rewrite system (TRS) R to complete the proof. `Prover9` is an automated theorem prover for first-order and equational logic, and `Mace4` is an automated searcher for finite models and counterexamples. Both the tools are very powerful in automated reasoning. Thanks to their programmer, William McCune.

Problem 4

Give a precise description of a non-trivial problem of your own choice, and encode this and solve it by one of the given programs.

Self-defined problem:

In an undirected network the edges are colored red, blue and yellow, and the following is given:

- From A there is a red edge to either C, E or G.
- There are red edges BF, BI, and CH.
- From G there is a yellow edge to either D or F.
- There is a yellow edge EG.
- From D there is a blue edge to either A or B.
- There are blue edges CG, DI, and EH.

prove that a path from A to B exists in which no two consecutive edges are of the same color.

Solution:

To solve this problem, we need to specify the definitions for the edges and the paths.

Edge definition: A red edge from x to y is denoted as $red(x, y)$. A yellow edge from x to y is denoted as $yellow(x, y)$. A blue edge from x to y is denoted as $blue(x, y)$. Since it is an undirected network, $red(x, y) = red(y, x)$, $yellow(x, y) = yellow(y, x)$ and $blue(x, y) = blue(y, x)$.

Path definition: A path can be an edge or a sequential connected edges. A path starting with a red edge is called as *redpath*. Similarly, there are *bluepath* and *ypath*. Particularly, a *redpath* can be a single red edge or a red edge followed by a *bluepath* or a *ypath*. Likewise, a *bluepath* can be a single blue edge or a blue edge followed by a *redpath* or a *ypath*, and a *ypath* can be a single yellow edge or a yellow edge followed by a *redpath* or a *bluepath*. In this way we can make sure that the paths, which are *redpath*, *bluepath* or *ypath*, will not have two consecutive edges in the same color.

We generalize this problem for an undirected network with n nodes and m colors of the edges. We introduce

$$\mathbb{N} \text{ as the set of nodes, } \mathbb{N} = \{A, B, C, ...\},$$
$$\mathbb{E} \text{ as the set of colors, } \mathbb{E} = \{color_1, color_2, ...\},$$
$$\text{and } \mathbb{P} \text{ as the set of color paths, } \mathbb{P} = \{cpath_1, cpath_2, ...\}.$$

Subsequently, we have the mappings

$$color_x : \mathbb{N} \times \mathbb{N} \to \mathbb{B}, \text{ and } cpath_x : \mathbb{N} \times \mathbb{N} \to \mathbb{B}.$$

With reference to our definitions, both edges and paths have dependence of the colors. So we introduce $\mathbb{D}_x$ as the set of nodes pairs indicating the color dependence of the nodes in each pair. For instance, there is a red edge between node A and node B, then $(A, B) \in \mathbb{D}_{red}$.

Some nodes in the network have only one edge in a special color. Then we introduce $\mathbb{G}_x^I \subseteq \mathbb{N}$, where $I \in \mathbb{N}$, as the set of the nodes, only one of which has an edge with node I in Color $color_x$.
Now we formulate the conditions.

- Edge definition

  Each edge has a starting node and an ending node, and each edge has color dependency.

$$\bigwedge_{x=1}^{m} \forall_{d \in \mathbb{D}_x}[color_x(fst(d), snd(d))] \ \wedge$$

$$\bigwedge_{x=1}^{m} \forall_{n_1,n_2 \in \mathbb{N}}[color_x(n_1,n_2) : color_x(n_2,n_1)] \ \wedge$$

$$\bigwedge_{x=1}^{m} \exists^1_{g \in \mathbb{G}_x^I}[color_x(I,g)]$$

- Path definition

  Likewise, each path has a starting node and a ending node, and each path has color dependency.

$$\bigwedge_{x=1}^{m} \forall_{n_1,n_2 \in \mathbb{N}}[color_x(n_1,n_2) : cpath_x(n_1,n_2)] \ \wedge$$

$$\bigwedge_{x=1}^{m} \bigwedge_{y=1, y \neq x}^{m} \forall_{n_1,n_2,n_3 \in \mathbb{N}}[color_x(n_1,n_2) \wedge cpath_y(n_2,n_3) : cpath_x(n_1,n_3)]$$

- Goals

  The goal of the problem is to prove the existence of a path from two nodes, A to B, composed with the edges that none of them has the same color as its neighbour(s).

$$\bigvee_{x=0}^{m} cpath_x(A,B)$$

This problem gives

$$\mathbb{N} = \{A,B,C,D,E,F,G,H,I\},\ \mathbb{E} = \{red, yellow, blue\},$$
$$\mathbb{G}_{red}^A = \{C,E,G\}, \mathbb{G}_{yellow}^G = \{D,F\} \text{ and } \mathbb{G}_{blue}^D = \{A,B\},$$
$$\mathbb{D}_{red} = \{(B,F),(B,I),(C,H)\},\ \mathbb{D}_{yellow} = \{(E,G)\} \text{ and } \mathbb{D}_{blue} = \{(C,G),(D,I),(E,H)\}.$$

Accordingly, we derive $\mathbb{P} = \{redpath, ypath, bluepath\}$. We use `Prover9` to prove the existence of such a path. The codes are as follows.

```
formulas(assumptions).
% edge definition
% From A there is a red edge to either C, E or G.
(red(a,c) & -red(a,e) & -red(a,g)) |
(-red(a,c) & red(a,e) & -red(a,g)) |
(-red(a,c) & -red(a,e) & red(a,g)).
```

```
% There are red edges BF, BI, and CH.
red(b,f).  red(b,i).  red(c,h).
% From G there is a yellow edge to either D or F.
(yellow(g,d) & -yellow(g,f)) | (yellow(g,f) & -yellow(g,d)).
% There is a yellow edge EG.
yellow(e,h).
% From D there is a blue edge to either A or B.
(blue(d,a) & -blue(d,b)) | (blue(d,b) & -blue(d,a)).
% There are blue edges CG, GH and DI.
blue(c,g).  blue(d,i).  blue(g,h).
%This is an undirected network.
red(x,y) -> red(y,x).  yellow(x,y) -> yellow(y,x).  blue(x,y) -> blue(y,x).
% path definition
red(x,y) -> redpath(x,y).
yellow(x,y) -> ypath(x,y).
blue(x,y) -> bluepath(x,y).
red(x,y) & bluepath(y,z) -> redpath(x,z).
red(x,y) & ypath(y,z) -> redpath(x,z).
yellow(x,y) & red(y,z) -> ypath(x,z).
yellow(x,y) & bluepath(y,z) -> ypath(x,z).
blue(x,y) & redpath(y,z) -> bluepath(x,z).
blue(x,y) & ypath(y,z) -> bluepath(x,z).
end_of_list.
formulas(goals).
redpath(a,b) | bluepath(a,b) | ypath(a,b).
end_of_list.
```

After applying `prover9 -f part2_4.in`, `Prover9` exits with 1 proof. Hence, the existence that a path from A to B has no two consecutive edges in the same color is proved to be true.

Remark:

We proved that there exits a path from A to B has no two consecutive edges in the same color. If a real applicable solution is provided, the proof will be more reliable. However, `Prover9` is a theorem prover for giving proofs based on resolution. It generates only the proof steps(resolution steps), not a satisfiable solution. Another approach is to derive the solution from the resolution steps. Unfortunately, the proof consists of 406 steps. It is too complicated to process them.

`Prover9` exits with the following proof.

```
    ============================== PROOF =================================
    % Proof 1 at 0.05 (+ 0.03) seconds.
    % Length of proof is 58.
    % Level of proof is 12.
    % Maximum clause weight is 12.
    % Given clauses 148.
    1 red(a,c) & -red(a,e) & -red(a,g) | -red(a,c) & red(a,e) & -red(a,g) | -red(a,c) & -red(a,e)
& red(a,g) # label(non_clause).  [assumption].
    2 yellow(g,d) & -yellow(g,f) | yellow(g,f) & -yellow(g,d) # label(non_clause).  [assumption].
    .........
    .........
    271 red(a,g).  [back_unit_del(247),unit_del(a,267)].
    274 yellow(g,d).  [back_unit_del(99),unit_del(a,271)].
    283 $F. [ur(64,a,271,a,c,143,a),unit_del(a,274)].
    ============================== end of proof ==========================
```

Generalization:

The two introduced sets $\mathbb{N}$ and $\mathbb{E}$, are inter-related. The number of nodes in the network limits the number of colors in the network. For instance, there is a two-node network. This network has only two edges, then it can have two colors in maximum.

Once there exists a satisfiable path, there should be infinitely many paths satisfying the fact that the path has no two consecutive edges in the same color. Because one node may be reached several times. In the real-time world, the same actions done in different time units lead to different consequences.