Practical Assignment Part 1
Automated Reasoning 2IMF25

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

Problem 1

Six trucks have to deliver pallets of obscure building blocks to a magic factory. Every truck has a capacity of 7800 kg and can carry at most eight pallets. In total, the following has to be delivered:

- Four pallets of nuzzles, each of weight 700 kg.
- A number of pallets of prittles, each of weight 800 kg.
- Eight pallets of skipples, each of weight 1000 kg.
- Ten pallets of crottles, each of weight 1500 kg.
- Five pallets of dupples, each of weight 100 kg.

Prittles and crottles are an explosive combination: they are not allowed to be put in the same truck.
Skipples need to be cooled; only two of the six trucks have facility for cooling skipples.
Dupples are very valuable; to distribute the risk of loss no two pallets of dupples may be in the same truck.
Investigate what is the maximum number of pallets of prittles that can be delivered, and show how for that number all pallets may be divided over the six trucks.

Solution:

We generalize the problem to offer n trucks to deliver the pallets. n is a positive integer. Then we introduce some variables t_{ij} for $i = 1, ..., n$ and $j = 1, ..., 5$, which represents the number of pallets of obscure building blocks j on the i-th truck. t_{ij} is a natural number. Every truck has a capacity, which is denoted as C kg, and the maximum number of pallets each truck can carry is M.

The following table shows which index is related to which building material:

	nuzzles	prittles	skipples	crottles	dupples
j	1	2	3	4	5

The pallets of the same building blocks have the same weight, $weight(j)$. Each kind of building blocks has a predefined number of pallets, $\sharp pallets(j)$.

Now we consider the conditions for the delivery.

Clause 1: t_{ij} should be no less than 0.

This is expressed by the formula

$$\bigwedge_{i,j:1\leq i\leq n\wedge 1\leq j\leq 5} t_{ij} \geq 0.$$

Clause 2: Every truck has a maximum capacity of C kg.

$$\bigwedge_{i=1}^{n}(\sum_{j=1}^{5} t_{ij} \times weight(j)) \leq C.$$

Clause 3: Every truck can carry at most M pallets.

$$\bigwedge_{i=1}^{n}(\sum_{j=1}^{5} t_{ij}) \leq M.$$

Clause 4: The total numbers of the pallets of each obscure building blocks should be exactly the same as the given number by $\sharp pallets(j)$.

$$\bigwedge_{j=1}^{5}(\sum_{i=1}^{n} t_{ij}) = \sharp pallets(j).$$

Clause 5: Prittles and crottles are not allowed to be put in the same truck. So in every truck the number of prittles or the number of crottles should be zero.

$$\bigwedge_{i=1}^{n} t_{i2} = 0 \vee t_{i4} = 0.$$

Clause 6: Only two of the trucks can deliver skipples.

So let's assume the n-th and $n-2$-th trucks can deliver skipples. Then the amount of skipples in these two trucks should be equal to the total amount of this material.

$$t_{n3} + t_{(n-2)3} = \sharp pallets(3).$$

Clause 7: No two pallets of dupples may be in the same truck. This means that at most one pallet of dupple is allowed in each truck.

$$\bigwedge_{i=1}^{n} t_{i5} \leq 1.$$

The total formula now consists of the conjunction of all these ingredients, that is,

$$\bigwedge_{i,j:1\le i\le n,1\le j\le 5} t_{ij} \ge 0 \ \wedge$$

$$\bigwedge_{i=1}^{n}(\sum_{j=1}^{5} t_{ij} \times weight(j)) \le C \ \wedge$$

$$\bigwedge_{i=1}^{n}(\sum_{j=1}^{5} t_{ij}) \le M \ \wedge$$

$$\bigwedge_{j=1}^{5}(\sum_{i=1}^{n} t_{ij}) = \sharp pallets(j) \ \wedge$$

$$\bigwedge_{i=1}^{n} t_{i2} = 0 \vee t_{i4} = 0 \ \wedge$$

$$t_{n3} + t_{(n-2)3} = \sharp pallets(3) \ \wedge$$

$$\bigwedge_{i=1}^{n} t_{i5} \le 1$$

This formula is easily expressed in SMT syntax. For Problem 1, we define the number of trucks $n = 6$, the maximum number of pallets in each truck $M = 8$, the capacity of the trucks as $C = 7800$ kg, and the parameters $weight(j)$ and $\sharp pallets(j)$ as specified in the table below.

	nuzzles	prittles	skipples	crottles	dupples
j	1	2	3	4	5
$weight(j)$	700kg	800kg	1000kg	1500kg	100kg
$\sharp pallets(j)$	4	P	8	10	5

With the values above, we get the following Yices codes.

```
benchmark Part1_1.smt
:logic QF_ALIA
:extrafuns (
(t11 Int) (t12 Int) (t13 Int) (t14 Int) (t15 Int)
(t21 Int) (t22 Int) (t23 Int) (t24 Int) (t25 Int)
......
(t61 Int) (t62 Int) (t63 Int) (t64 Int) (t65 Int)
)
:formula
(and
(>= t11 0) (>= t12 0) (>= t13 0) (>= t14 0) (>= t15 0)
(>= t21 0) (>= t22 0) (>= t23 0) (>= t24 0) (>= t25 0)
......
(>= t61 0) (>= t62 0) (>= t63 0) (>= t64 0) (>= t65 0)
(<= (+ (* t11 700) (* t12 800) (* t13 1000) (* t14 1500) (* t15 100)) 7800)
(<= (+ (* t21 700) (* t22 800) (* t23 1000) (* t24 1500) (* t25 100)) 7800)
......
(<= (+ (* t61 700) (* t62 800) (* t63 1000) (* t64 1500) (* t65 100)) 7800)
(<= (+ t11 t12 t13 t14 t15) 8)
```

```
(<= (+ t21 t22 t23 t24 t25) 8)
......
(<= (+ t61 t62 t63 t64 t65) 8)
(= (+ t11 t21 t31 t41 t51 t61) 4)
(= (+ t12 t22 t32 t42 t52 t62) 18)
(= (+ t13 t23 t33 t43 t53 t63) 8)
(= (+ t14 t24 t34 t44 t54 t64) 10)
(= (+ t15 t25 t35 t45 t55 t65) 5)
(or (= t12 0) (= t14 0))
(or (= t22 0) (= t24 0))
......
(or (= t62 0) (= t64 0))
(= (+ t43 t63) 8)
(<= t15 1) (<= t25 1) (<= t35 1) (<= t45 1) (<= t55 1) (<= t65 1)
)
```

Applying `yices-smt -m Part1_1.smt` several times, we find when the number of pallets of prittles P is 19, it is UNSAT. When the number is 18, it is SAT. Therefore, we conclude that the maximal number of pallets of prittles is 18. The following result is yielded within a fraction of a second:

```
sat
(= t52 7)
(= t11 0)
(= t14 5)
(= t34 0)
(= t21 4)
(= t15 1)
(= t44 2)
(= t45 1)
(= t54 0)
(= t33 0)
(= t35 0)
(= t51 0)
(= t23 0)
(= t53 0)
(= t65 1)
(= t13 0)
(= t62 0)
(= t25 1)
(= t32 8)
(= t41 0)
(= t63 4)
(= t43 4)
(= t42 0)
(= t12 0)
(= t64 0)
(= t55 1)
(= t22 0)
(= t24 3)
(= t61 0)
(= t31 0)
```

The following table shows the solution for this specific problem.

	nuzzles	prittles	skipples	crottles	dupples
Truck 1	0	0	0	5	1
Truck 2	4	0	0	3	1
Truck 3	0	8	0	0	0
Truck 4	0	0	4	2	1
Truck 5	0	7	0	0	1
Truck 6	0	3	4	0	1

Remark

In this problem, we are required to find out the maximum number of the pallets of prittles. Such kind of maximum values searching is likely to become very time-consuming since there is no explicitly close upper bound to start the searching.

For this problem, we first find out the theoretically possible maximum number of the pallets of prittles through calculation. Because the capacity of each track is given, and it is also provided the total weights of the pallets of the other obscure building blocks, we can estimate the number.

$\frac{7800\times6-700\times4-1000\times8-1500\times10-100\times5}{800} = 25.625$

Because the number of pallets is a natural number, the maximum number should not be more than 25. Therefore, we can start the debugging from $P = 25$ downwards.

Generalization

We generalized all the invariants except the total number of kinds of the building blocks in this problem, because most of the requirements come from the features of different building blocks, which indicates that adding more different building blocks will generate more requirements, and as a consequent, the entire formula we summarized in the end of the solution is not applicable.

One more note is for the people who are interested in setting the number of trucks, n, larger than 10. They need to take care about the notations of the variables t_{ij}. For instance, the number of pallets of building blocks labeled 1 on the eleventh truck, $t_{11,1}$, is expressed as $t111$ in Yices codes. This expression can also represent the number of pallets of building blocks labeled 11 on the first truck, $t_{1,11}$. An extra symbol between the two numbers i and j is required to avoid this ambiguity.

Decreasing the number of trucks is more of interest. We did some testing to figure out how many pallets of prittles can satisfy these conditions with fewer trucks. When there are five trucks in all, ten pallets of prittles can be delivered with the satisfiability of the conditions. With fewer trucks, no satisfiability is reached. Since one truck can only deliver one pallet of drupples due to the condition, the number of trucks has to be larger than the number of pallets of dupples for delivery. It can be expressed as this

$$\sharp pallets(5) \leq m$$

Problem 2

Give a chip design containing three power components and eight regular components satisfying the following constraints:

- The width of the chip is 29 and the height is 22.
- All power components have width 4 and height 2.
- The sizes of the eight regular components are 9×7, 12×6, 10×7, 18×5, 20×4, 10×6, 8×6 and 10×8, respectively.
- All components may be turned 90, but may not overlap.
- In order to get power, all regular components should directly be connected to a power component, that is, an edge of the component should have at least one point in common with an edge of the power component.
- Due to limits on heat production the power components should be not too close: their centres should differ at least 17 in either the x direction or the y direction (or both).

Solution:

First of all, we generalize this problem for any width S_1 and height S_2 of the chip and for any number p and r of power and regular components respectively. Assume the chip plan is a positive plane whose left-bottom corner has coordinate $(0, 0)$, then the right-top corner has coordinate (S_1, S_2). Next, we introduce $2 \times p$ coordinates of the form (x_{ik}, y_{ik}) for $i = 1, 2, ..., p$ and $k = 1, 2$, to represent the corners of each power component. Hence, power component i has coordinates (x_{i1}, y_{i1}), (x_{i2}, y_{i1}), (x_{i1}, y_{i2}) and (x_{i2}, y_{i2}).

Similarly, we introduce $2 \times r$ coordinates of the form (n_{jk}, m_{jk}) for $j = 1, 2, ..., r$ and $k = 1, 2$, to represent the corners of each regular component.

Now, we give formulas that translates the constraints in terms of this variables.

Constraint 1: The chip is a positive plane, so all the coordinates should be positive. Additionally, the width of the chip is S_1 and the height is S_2.

Then we have

$$\bigwedge_{i=1}^{p} S_1 \geq x_{i2} \geq x_{i1} \geq 0 \wedge S_2 \geq y_{i2} \geq y_{i1} \geq 0 \ \wedge$$

$$\bigwedge_{i=1}^{r} S_1 \geq n_{i2} \geq n_{i1} \geq 0 \wedge S_2 \geq m_{i2} \geq m_{i1} \geq 0$$

Constraint 2: All coordinates of a given component have to respect its respective size, also a component may be turned 90. To formulate this, we define Z_1 and Z_2 as the height and width of all power components, and H_j and W_j as the height and width of regular component j.

Then we have the following formula for power components:

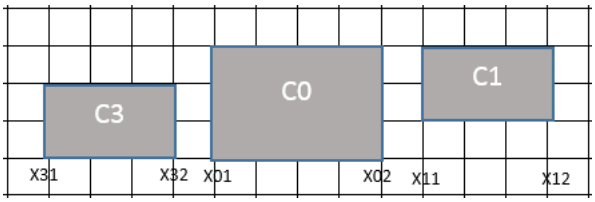
$$\bigwedge_{i=1}^{p}((x_{i2} - x_{i1} = Z_1 \ \wedge \ y_{i2} - y_{i1} = Z_2) \ \vee$$

$$(x_{i2} - x_{i1} = Z_2 \ \wedge \ y_{i2} - y_{i1} = Z_1))$$

Similarly, the formula for regular components is:

$$\bigwedge_{j=1}^{r}(n_{j2} - n_{j1} = W_j \ \wedge \ m_{j2} - m_{j1} = H_j) \ \vee (n_{j2} - n_{j1} = H_j \ \wedge \ m_{j2} - m_{j1} = W_j)$$

Constraint 3: All components may not overlap.



As the figure shows, only if a component's x_{i1} is larger than the other one's x_{k2}, then these two components are not overlapped. We also have the case in the y direction. Hence the formulas that express the two components may not overlap are the following.

Among the power components:

$$\bigwedge_{i,l:1\leq i<l\leq p} x_{i2} \leq x_{l1} \ \vee \ x_{l2} \leq x_{i1} \ \vee \ y_{i2} \leq y_{l1} \ \vee \ y_{l2} \leq y_{i1}$$

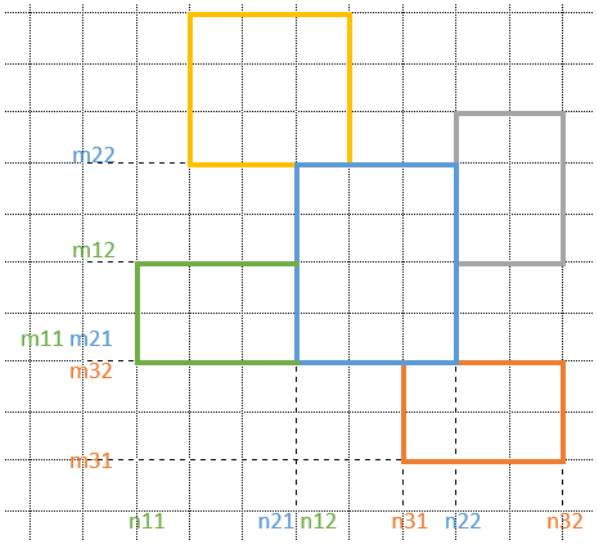
Among the regular components:

$$\bigwedge_{j,l:1\leq j<l\leq r} n_{j2} \leq n_{l1} \ \vee \ n_{l2} \leq n_{j1} \ \vee \ m_{j2} \leq m_{l1} \ \vee \ m_{l2} \leq m_{j1}$$

Between the power components and the regular components:

$$\bigwedge_{i,k:1\leq i\leq p,1\leq j\leq r} x_{i2} \leq n_{j1} \ \vee \ n_{j2} \leq x_{i1} \ \vee \ y_{i2} \leq m_{j1} \ \vee \ m_{j2} \leq y_{i1}$$

Constraint 4: An edge of the regular component should have at least one point in common with an edge of the power component.



As the figure shows, if two components are touched at least by one point, then at least one coordinate of a component is the same as the other one. Based on this observation, we can state that when the x coordinates of two components are the same, and also the y's coordinate of one is bound by the y's coordinates of the other component, we can conclude that they are touching. The same holds when they share a y's coordinate. We denote this condition as follows.

$$\bigwedge_{i,k:1\leq i\leq p,1\leq k\leq r} ((x_{i2} = n_{k1} \ \vee \ x_{i1} = n_{k2}) \ \wedge \ (y_{i2} \geq m_{k1} \wedge m_{k2} \geq y_{i1}) \vee$$

$$((y_{i2} = m_{k1} \ \vee \ y_{i1} = m_{k2}) \ \wedge \ (x_{i2} \geq n_{k1} \wedge n_{k2} \geq x_{i1})))$$

Constraint 5: The power components' centres should differ at least 17 in either the x direction or the y direction (or both).

The centers of the power components are $(x_{i1} + \frac{x_{i2}-x_{i1}}{2}, y_{i1} + \frac{y_{i2}-y_{i1}}{2})$.

For each pair of the power components, we have

$$\bigwedge_{i,k:1\leq i<k\leq p} [(x_{i1} + \frac{x_{i2} - x_{i1}}{2}) - (x_{k1} + \frac{x_{k2} - x_{k1}}{2}) \geq 17 \vee$$

$$(x_{k1} + \frac{x_{k2} - x_{k1}}{2}) - (x_{i1} + \frac{x_{i2} - x_{i1}}{2}) \geq 17 \vee$$

$$(y_{k1} + \frac{y_{k2} - y_{k1}}{2}) - (y_{i1} + \frac{y_{i2} - y_{i1}}{2}) \geq 17 \vee$$

$$(y_{i1} + \frac{y_{i2} - y_{i1}}{2}) - (y_{k1} + \frac{y_{k2} - y_{k1}}{2}) \geq 17]$$

The total formula now consists of the conjunction of all these clauses. To solve this particular problem, we made the Yices smt code considering the number of power components $p = 3$, the number of regular components $r = 8$, and replacing the parameters S_1, S_2, Z_1, Z_2, H_j and W_j with its own values of height and width as specified by the problem.

This is expressed in SMT syntax as follow:

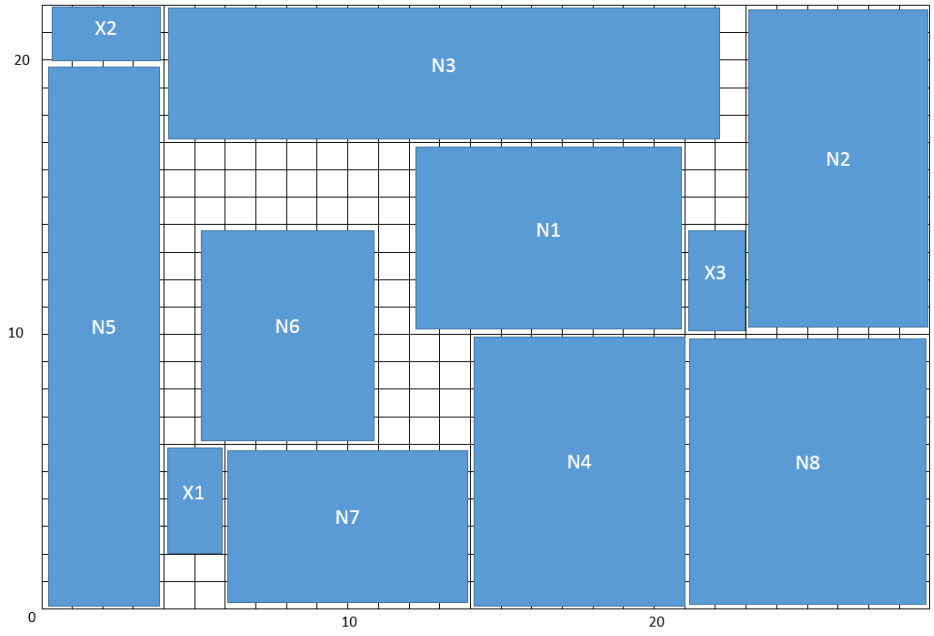
```
(benchmark Part1_2.smt
:logic QF_LIA
:extrafuns (
(x11 Int) (x12 Int) (y11 Int) (y12 Int)
(x21 Int) (x22 Int) (y21 Int) (y22 Int)
......
(n81 Int) (n82 Int) (m81 Int) (m82 Int)
)
:formula (and
(>= 29 x12) (>= x12 x11) (>= x11 0) (>= 22 y12) (>= y12 y11) (>= y11 0)
(>= 29 x22) (>= x22 x21) (>= x21 0) (>= 22 y22) (>= y22 y21) (>= y21 0)
......
(>= 29 n82) (>= n82 n81) (>= n81 0) (>= 22 m82) (>= m82 m81) (>= m81 0)
(or (and (>= 29 x12) (>= 22 y12) (= (- x12 x11) 4) (= (- y12 y11) 2))
(and (>= 22 x12) (>= 29 y12) (= (- y12 y11) 4) (= (- x12 x11) 2)))
......
(or (and (= (- x32 x31) 4) (= (- y32 y31) 2)) (and (= (- x32 x31) 2) (= (- y32 y31) 4)))
(or (and (= (- n12 n11) 9) (= (- m12 m11) 7)) (and (= (- n12 n11) 7) (= (- m12 m11) 9)))
......
(or (and (= (- n82 n81) 10) (= (- m82 m81) 8)) (and (= (- n82 n81) 8) (= (- m82 m81) 10)))
;; not overlap among power components
(or (<= x12 x21) (<= x22 x11) (<= y12 y21) (<= y22 y11))
(or (<= x12 x31) (<= x32 x11) (<= y12 y31) (<= y32 y11))
(or (<= x32 x21) (<= x22 x31) (<= y32 y21) (<= y22 y31))
;; not overlap among regular components
(or (<= n12 n21) (<= n22 n11) (<= m12 m21) (<= m22 m11))
(or (<= n12 n31) (<= n32 n11) (<= m12 m31) (<= m32 m11))
......
(or (<= n72 n81) (<= n82 n71) (<= m72 m81) (<= m82 m71))
;;not overlap between power components and regular ones
(or (<= x12 n11) (<= n12 x11) (<= y12 m11) (<= m12 y11))
(or (<= x12 n21) (<= n22 x11) (<= y12 m21) (<= m22 y11))
......
(or (<= x32 n81) (<= n82 x31) (<= y32 m81) (<= m82 y31))
;;Constraint 4
(or
(and (or (= x11 n12) (= x12 n11)) (not (or (< y12 m11) (> y11 m12))))
(and (or (= x21 n12) (= x22 n11)) (not (or (< y22 m11) (> y21 m12))))
......
(and (or (= y31 m12) (= y32 m11)) (not (or (< x32 n11) (> x31 n12))))
)
......
(or
(and (or (= x11 n82) (= x12 n81)) (not (or (< y12 m81) (> y11 m82))))
(and (or (= x21 n82) (= x22 n81)) (not (or (< y22 m81) (> y21 m82))))
......
(and (or (= y31 m82) (= y32 m81)) (not (or (< x32 n81) (> x31 n82))))
)
;;Constraint 5
(or (>= (- (+ x11 (/ (- x12 x11) 2)) (+ x21 (/ (- x22 x21) 2))) 17)
(>= (- (+ x21 (/ (- x22 x21) 2)) (+ x11 (/ (- x12 x11) 2))) 17)
(>= (- (+ y11 (/ (- y12 y11) 2)) (+ y21 (/ (- y22 y21) 2))) 17)
(>= (- (+ y21 (/ (- y22 y21) 2)) (+ y11 (/ (- y12 y11) 2))) 17))
......
(or (>= (- (+ x11 (/ (- x12 x11) 2)) (+ x31 (/ (- x32 x31) 2))) 17)
(>= (- (+ x31 (/ (- x32 x31) 2)) (+ x11 (/ (- x12 x11) 2))) 17)
```

```
(>= (- (+ y11 (/ (- y12 y11) 2)) (+ y31 (/ (- y32 y31) 2))) 17)
(>= (- (+ y31 (/ (- y32 y31) 2)) (+ y11 (/ (- y12 y11) 2))) 17))
))
```

Applying `yices-smt -m part1_2.smt`, we test out a satisfiable chip design plan.

```
sat
(= x11 4)
(= x12 6)
(= y11 2)
(= y12 6)
(= x21 0)
(= x22 4)
(= y21 20)
(= y22 22)
(= x31 21)
(= x32 23)
......
```

The following picture illustrates the solution of this problem given by the tool.



Remark:

Due to the complexity of the formulas, it is hard and time consuming to make the complete yices' code by hand. The best approach is to create a script that automatically generates the code, with this we ensure that the formulas are translated in the correct syntax removing the possibility of human mistakes.

Generalization:

We generalized our approach for any composition of a chip, some power components and regular components with known sizes to test if it is possible to obtain a fine chip design. Regardless the size of the chip and these components and the number of the components, this approach specifies two more constraints. One is the safe distance for the power components

due to heat production. The other is the common touching between power components and regular components. These two constraints are derived from the characteristics of the components to apply to the chip. Different components have different characteristics, which may require different clauses to formulate. This is the limitation of our generalization. The satisfiability for a good chip design highly depends on the size of the components.

Problem 3

Twelve jobs numbered from 1 to 12 have to be executed satisfying the following requirements:

- The running time of job i is i, for $i = 1, 2, ..., 12$.
- All jobs run without interrupt.
- Job 3 may only start if jobs 1 and 2 have been finished.
- Job 5 may only start if jobs 3 and 4 have been finished.
- Job 7 may only start if jobs 3, 4 and 6 have been finished.
- Job 9 may only start if jobs 5 and 8 have been finished.
- Job 11 may only start if Job 10 has been finished.
- Job 12 may only start if jobs 9 and 11 have been finished.
- Jobs 5,7 en 10 require a special equipment of which only one copy is available, so no two of these jobs may run at the same time.

Find a solution of this scheduling problem for which the total running time is minimal.

Solution:

We generalize this problem for n number of jobs. First we introduce two variables S_i and F_i for $i = 1, 2, 3, ..., n$.

- S_i is the starting time of Job i. $S_i \geq 0$. For example: If Job 1 starts at 0, then $S_1 = 0$. S_i can be any positive integer number if it is satisfied with all requirements.
- F_i is the finish time of Job i.

Secondly, we find the clauses which have to be satisfied by all requirements of this problem.

Clause 1: We defined that the starting time of all jobs have to be greater or equal to zero. This is expressed by the formula

$$\bigwedge_{i=1}^{n} (S_i \geq 0).$$

Clause 2: The running time of job i is i, for $i = 1, 2, ..., n$, and all jobs run without interrupt. So we have

$$\bigwedge_{i=1}^{n} (F_i = S_i + i).$$

Clause 3: The problem specifies that there is a dependency between some jobs, for instance job 3 may only start until jobs 1 and 2 have finished. In order to generalize this and to make a compact formula, we introduce the set of dependencies $\mathbf{D}_i$ as the set of $finish$ $times$ of the jobs that have to end before starting job i, for $i = 1, 2, ..., n$. So the set of

dependencies of job 3 may look like: $\mathbf{D}_3 = \{F_1, F_2\}$. Now we can formulate the clause of dependencies between jobs as follow:

$$\bigwedge_{i=1}^{n} \bigwedge_{d \in D_i} (S_i \geq d).$$

Clause 4: The last requirement states that there is an special equipment that can be only used by one job at a time. So the jobs that require this resource cannot work in parallel. To express this we consider the condition that if a job that requires the resource happens before another job that also needs the resource, then the later must only happen if the first job has finished.

In order to generalize this and formulate a compact expression, we define $\mathbf{P}$ as the set of all jobs that share the resource, and define the mappings $f : \mathbf{P} \to \mathbf{F}$ to map the jobs in $\mathbf{P}$ to its respective finish time, and $s : \mathbf{P} \to \mathbf{S}$ to map the jobs in $\mathbf{P}$ to its respective starting time, where $\mathbf{F}$ and $\mathbf{S}$ are the sets of all finish times and starting times.

Using all this elements the formula that expresses this clause is the following:

$$\bigwedge_{p,q:\, p,q \in \mathbf{P} \wedge p \neq q} (s(p) \leq s(q)) \Rightarrow (f(p) \leq s(q)).$$

Clause 5: The problem also require to find a solution of this scheduling problem for which the total running time is minimal. In order to find such scheduling, we introduce a variable T to represent the allowed running time, now we have to bound the finish time of all jobs to this variable. This can be expressed with the following formula:

$$\bigwedge_{i=1}^{n} (F_i \leq T).$$

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

$$\bigwedge_{i=1}^{n} (S_i \geq 0) \ \wedge$$

$$\bigwedge_{i=1}^{n} (F_i = S_i + i) \ \wedge$$

$$\bigwedge_{i=1}^{n} \bigwedge_{d \in D_i} (S_i \geq d) \ \wedge$$

$$\bigwedge_{p,q:\, p,q \in \mathbf{P} \wedge p \neq q} (s(p) \leq s(q)) \Rightarrow (f(p) \leq s(q)) \ \wedge$$

$$\bigwedge_{i=1}^{n} (F_i \leq T).$$

The complete formula expressed in SMT syntax choosing $n = 12$, $D_3 = \{F_1, F_2\}$, $D_5 = \{F_3, F_4\}$, $D_7 = \{F_3, F_4, F_6\}$, $D_9 = \{F_5, F_8\}$, $D_{11} = \{F_{10}\}$, $D_{12} = \{F_9, F_{11}\}$, $P = \{Job\ 5, Job\ 7, Job\ 10\}$ and $T = 36$ is as follow:

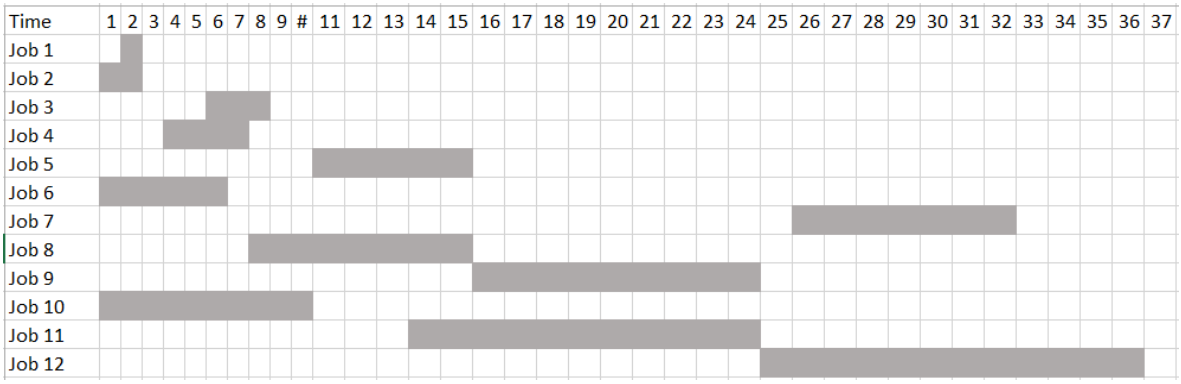
```
(benchmark part1_3
:logic QF_UFLIA
:extrafuns (
(S1 Int) (S2 Int) (S3 Int) (S4 Int) (S5 Int) (S6 Int)
(S7 Int) (S8 Int) (S9 Int) (S10 Int) (S11 Int) (S12 Int)
(F1 Int) (F2 Int) (F3 Int) (F4 Int) (F5 Int) (F6 Int)
(F7 Int) (F8 Int) (F9 Int) (F10 Int) (F11 Int) (F12 Int))
:formula (and
(>= S1 0) (>= S2 0) (>= S3 0) (>= S4 0) (>= S5 0) (>= S6 0)
(>= S7 0) (>= S8 0) (>= S9 0) (>= S10 0) (>= S11 0) (>= S12 0)
......
;All jobs run without interrupt
(= (+ S1 1) F1)
(= (+ S2 2) F2)
(= (+ S3 3) F3)
......
;Job dependencies
(>= S3 F1) (>= S3 F2)
(>= S5 F3) (>= S5 F4)
(>= S7 F3) (>= S7 F4) (>= S7 F6)
......
;Jobs 5,7 and 10 cannot execute in parallel
(implies (<= S5 S7) (<= F5 S7))
(implies (<= S5 S10) (<= F5 S10))
(implies (<= S7 S5) (<= F7 S5))
......
(<= F1 36)
(<= F2 36)
(<= F3 36)
......
```

Applying `yices-smt -m part1_3.smt` to test out a satisfiable scheduling, we got the following result:

```
sat
(= S1 1)
(= S2 0)
(= S3 5)
(= S4 3)
......
(= F5 15)
(= F6 6)
(= F7 32)
(= F8 15)
(= F9 24)
(= F10 10)
(= F11 24)
(= F12 36)
```

We know that this is the schedule where the total running time is minimal because we started increasing the value of T from 9 and stop until the SMT is SAT. Finally, we find when $T = 35$, it is UNSAT, but when $T = 36$, it is SAT. Therefore, we conclude that the minimal running time satisfying the requirements is 36.

The following picture illustrates such schedule:



Remark:

For this particular problem, the minimum running time was found testing the value of T for different values until we find the minimum satisfiable. Although doing this was relatively easy because the number of jobs is small, and therefore also the minimum running time, this method of finding the minimum value by hand is not very suitable for longer number of jobs and dependencies between them. For instance, if we have 1000 jobs with their respective dependencies, it would be very time consuming to find such value by hand. Since Yices 1.2 does not have a method for automatically find minimum values, a possible solution is to implement a script that performs the tests automatically.

Generalization:

We generalized this problem for any number of jobs and dependencies between them. Since we solved this choosing the values that states the problem, it would be interesting to know the results after changing some parameters. For instance, lets imagine that we have enough copies of the special equipment required by jobs 5, 7 and 10, so they can run in parallel. For this case, the minimum running time is 33, this is not a big improvement so maybe the money spent by buying the extra equipment does not pay off in performance.

Problem 4

Seven integer variables a_1, a_2, a_3, a_4, a_5, a_6, a_7 are given, for which the initial value of a_i is i for $i = 1, ..., 7$. The following steps are defined: choose i with $1 < i < 7$ and execute

$$a_i := a_{i-1} + a_{i+1},$$

that is, a_i gets the sum of the values of its neighbors and all other values remain unchanged. Show how it is possible that after a number of steps there is a number ≥ 50 that occurs at least twice in a_1, a_2, a_3, a_4, a_5, a_6, a_7.

Solution:

We generalize this problem by defining n integer variables $a_1, a_2...a_n$, with the same initialization pattern, where a_i is i for $i = 1, ..., n$. We also generalize the defined step, where i can be chosen within the range of $1 < i < n$. The part where the selected variable gets the sum of the value of its neighbors stays the same, whereas the restriction is generalized in the form that after m number of steps there is a number ≥ 50 that occurs at least twice in $a_1, a_2, ..., a_n$.

For doing so, we introduce $n \times (m+1)$ integer variables a_{ij} for $i = 1, ..., n$ and $j = 0, ..., m$, where a_{ij} represents the variable a_i after j number of steps. We also introduce m integer variables C_k for $k = 1, ..., m$, where C_k is the chosen index i to execute the procedure for step k. Finally, we introduce a variable P to represent the number ≥ 50 that we want to find after performing m steps.

The problem specifies to initialize the values of a_i. This is expressed with the introduced variables by the formula

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

Also we have to specified the boundaries of the chosen variable, so it has to be selected within the range of $1 < i < n$ in every step. This can be expressed by the formula

$$\bigwedge_{k=1}^{m} 1 < C_k < n.$$

Next we express the step that after selecting a variable, it gets the sum of the values of its neighbors and all other values remain unchanged. For clarity, we split this into two conditions. One to express that the remaining variables remain with the same value, and the other to execute the sum of the neighbors. The first one can easily be expressed with the introduced variables by specifying that if C_k is equal to l, then the values for a_{ik} with i different to l will be the same as the values of the previous step $a_{i(k-1)}$. This is expressed with the formula

$$\bigwedge_{k=1}^{m} \bigwedge_{l=2}^{n-1} (C_k = l) \rightarrow \Big(\bigwedge_{i:1 \leq i \leq n \wedge i \neq l} a_{ik} = a_{i(k-1)} \Big).$$

Similarly, the second condition can be expressed by specifying that if C_k is equal to l, then the value of a_{lk} should be equal to the sum of its neighbors in the previous step $a_{(l-1)(k-1)}$

and $a_{(l+1)(k-1)}$. This is expressed with the formula

$$\bigwedge_{k=1}^{m}\bigwedge_{l=2}^{n-1}(C_k = l) \rightarrow (a_{lk} = a_{(l-1)(k-1)} + a_{(l+1)(k-1)}).$$

It is worth to mention that these formulas are taking the conjunction considering l running from 2 to $n-1$. This is done this way because these are the only possible values that C_k can have, as specified in the second formula.

Finally, we consider the requirement that after m number of steps there is a number $P \geq 50$ that occurs at least twice. This is expressed by the formulas

$$\bigvee_{i,i':1\leq i<i'\leq m}(a_{im} = P \wedge a_{i'm} = P)$$

$$P \geq 50.$$

The total formula now consists of the conjunction of all these ingredients, that is,

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

$$\bigwedge_{k=1}^{m} 1 < C_k < n \ \wedge$$

$$\bigwedge_{k=1}^{m}\bigwedge_{l=2}^{n-1}(C_k = l) \rightarrow (\bigwedge_{i:1\leq i\leq n\wedge i\neq l} a_{ik} = a_{i(k-1)}) \ \wedge$$

$$\bigwedge_{k=1}^{m}\bigwedge_{l=2}^{n-1}(C_k = l) \rightarrow (a_{lk} = a_{(l-1)(k-1)} + a_{(l+1)(k-1)}) \ \wedge$$

$$\bigvee_{i,i':1\leq i<i'\leq m}(a_{im} = P \wedge a_{i'm} = P) \ \wedge$$

$$P \geq 50$$

This formula can be expressed in SMT syntax, for instance, for $n = 7$ and $m = 10$ one can generate

```
(benchmark part1_4.smt
:logic QF_UFLIA
:extrafuns
((a1_0 Int) (a2_0 Int) (a3_0 Int) (a4_0 Int) (a5_0 Int) (a6_0 Int) (a7_0 Int)
(a1_1 Int) (a2_1 Int) (a3_1 Int) (a4_1 Int) (a5_1 Int) (a6_1 Int) (a7_1 Int)
......
(a1_10 Int) (a2_10 Int) (a3_10 Int) (a4_10 Int) (a5_10 Int) (a6_10 Int) (a7_10 Int)
(C1 Int) (C2 Int) (C3 Int) (C4 Int) (C5 Int) (C6 Int) (C7 Int)
(C8 Int) (C9 Int) (C10 Int) (P Int)
)
:formula
(and
;The initial value of each variable is equal to its index
(= a1_0 1)
(= a2_0 2)
```

```
(= a3_0 3)
......
;Each choice has to be in the range of 1 to N
(< 1 C1) (< C1 7)
(< 1 C2) (< C2 7)
(< 1 C3) (< C3 7)
......
;If a choice is taken
(implies (= C1 2) (and (= a1_1 a1_0) (= a2_1 (+ a1_0 a3_0)) (= a3_1 a3_0) (= a4_1 a4_0)
(= a5_1 a5_0) (= a6_1 a6_0) (= a7_1 a7_0)))
(implies (= C1 3) (and (= a1_1 a1_0) (= a2_1 a2_0) (= a3_1 (+ a2_0 a4_0)) (= a4_1 a4_0)
(= a5_1 a5_0) (= a6_1 a6_0) (= a7_1 a7_0)))
(implies (= C1 4) (and (= a1_1 a1_0) (= a2_1 a2_0) (= a3_1 a3_0) (= a4_1 (+ a3_0 a5_0))
(= a5_1 a5_0) (= a6_1 a6_0) (= a7_1 a7_0)))
(implies (= C1 5) (and (= a1_1 a1_0) (= a2_1 a2_0) (= a3_1 a3_0) (= a4_1 a4_0)
(= a5_1 (+ a4_0 a6_0)) (= a6_1 a6_0) (= a7_1 a7_0)))
(implies (= C1 6) (and (= a1_1 a1_0) (= a2_1 a2_0) (= a3_1 a3_0) (= a4_1 a4_0)
(= a5_1 a5_0) (= a6_1 (+ a5_0 a7_0)) (= a7_1 a7_0)))
(implies (= C2 2) (and (= a1_2 a1_1) (= a2_2 (+ a1_1 a3_1)) (= a3_2 a3_1) (= a4_2 a4_1
(= a5_2 a5_1) (= a6_2 a6_1) (= a7_2 a7_1)))
(implies (= C2 3) (and (= a1_2 a1_1) (= a2_2 a2_1) (= a3_2 (+ a2_1 a4_1)) (= a4_2 a4_1)
(= a5_2 a5_1) (= a6_2 a6_1) (= a7_2 a7_1)))
(implies (= C2 4) (and (= a1_2 a1_1) (= a2_2 a2_1) (= a3_2 a3_1) (= a4_2 (+ a3_1 a5_1))
(= a5_2 a5_1) (= a6_2 a6_1) (= a7_2 a7_1)))
(implies (= C2 5) (and (= a1_2 a1_1) (= a2_2 a2_1) (= a3_2 a3_1) (= a4_2 a4_1)
(= a5_2 (+ a4_1 a6_1)) (= a6_2 a6_1) (= a7_2 a7_1)))
(implies (= C2 6) (and (= a1_2 a1_1) (= a2_2 a2_1) (= a3_2 a3_1) (= a4_2 a4_1)
(= a5_2 a5_1) (= a6_2 (+ a5_1 a7_1)) (= a7_2 a7_1)))
......
;After m number of steps there is a number P >= 50 that occurs at least twice
(or (and (= a1_10 P) (= a2_10 P))
(and (= a1_10 P) (= a3_10 P))
(and (= a1_10 P) (= a4_10 P))
(and (= a1_10 P) (= a5_10 P))
......
(>= P 50)
)
```

Applying `yices-smt -m part1_4.smt`, it yields the following results:

```
(= a1_0 1)
(= a2_0 2)
(= a3_0 3)
(= a4_0 4)
(= a5_0 5)
(= a6_0 6)
(= a7_0 7)
......
(= a1_10 1)
(= a2_10 4)
(= a3_10 57)
(= a4_10 53)
(= a5_10 50)
(= a6_10 57)
(= a7_10 7)
(= C1 4)
(= C2 2)
(= C3 6)
```

```
(= C4 5)
(= C5 6)
(= C6 4)
(= C7 5)
(= C8 4)
(= C9 3)
(= C10 6)
(= P 57)
```

The final result is shown in next table.

variables/step	0	1	2	3	4	5	6	7	8	9	10
C_k		4	2	6	5	6	4	5	4	3	6
a_1	1	1	1	1	1	1	1	1	1	1	1
a_2	2	2	4	4	4	4	4	4	4	4	4
a_3	3	3	3	3	3	3	3	3	3	57	**57**
a_4	4	8	8	8	8	8	23	23	53	53	53
a_5	5	5	5	5	20	20	20	50	50	50	50
a_6	6	6	6	12	12	27	27	27	27	27	**57**
a_7	7	7	7	7	7	7	7	7	7	7	7

As can be seen, after 10 steps, it is possible to find a number ≥ 50 that occurs at least twice. For this case this number is 57.

Remark:

Although the method presented here successfully solves the problem, it is not very practical for large number of steps. The reason is that for every extra step that we want to analyze, it is necessary to add $n + 1$ extra variables to the yices code, with its respective conditions, resulting in a significant increment of code. Additionally, since the steps have to be explicitly added, if for some reason such number of steps m does not exist (e.g. choosing n = 3), then no matter how many steps we added to the code, we will never get a satisfied condition.

Generalization:

We solved this problem choosing the number of variables $n = 7$, but it would be interesting to know the results for other values of n. For $n > 7$ it is easy to see that the formula is still satisfiable after 10 steps, since we can choose the same C_k variables as for $n = 7$ and the result will be the same. For $n = 3$ the resulting formula is unsatisfiable no matter how many number of steps we choose, this can be easily seen because we only can choose the variable a_2 to execute the steps for this specific case.

For $n = 4$ the formula again is always unsatisfiable. We can see this by observing the facts that we only can choose a_2 and a_3 to execute the steps, and that it will always be the case that $a_2 = 1 + a_3$ or $a_3 = 4 + a2$, so they will never be equal. For $n = 5$ and $n = 6$, and considering the number of steps $m = 10$, the formula is unsatisfiable.