Automated Reasoning
Practical Assignment – Part 1

Jelmer Firet (s1023433, jelmer.firet@ru.nl)
Bram Pulles (s1015194, bram.pulles@ru.nl)

October 10, 2022

1 Introduction

We have chosen to use Z3 for solving all of the assignments. In particular we use the pythonic Z3 library.

2 Pallets

In this task we need to distribute pallets of various goods over a limited number of trucks. To this end we have formulated the problem as a system of linear equations. We created an integer variable $v_{g,t}$ for every *good* and *truck* combination such that $v_{g,t} \in \mathit{good} \times \mathit{truck}$. This variable describes how many pallets of a certain good are in the specific truck. Using just these variables we can describe all of the constraints.

- We can only have a positive number of pallets in a truck.

$$\forall_{v_{g,t} \in \mathit{good} \times \mathit{truck}}\ v_{g,t} \geq 0$$

- Every truck has at most eight pallets.

$$\forall_{t \in \mathit{truck}} \left(\sum_{g \in \mathit{good}} v_{g,t} \right) \leq 8$$

- Every truck can carry at most 8000 kg. Let w be a function giving the weight of a given good $w : \mathit{good} \to \mathbb{N}$.

$$\forall_{t \in \mathit{truck}} \left(\sum_{g \in \mathit{good}} w(g) \right) \leq 8000$$

- For every good all the pallets are distributed. Except for prittles, as no number of pallets is specified, we need to maximize this. Let p be a function giving the number of pallets of a given good $p : good \to \mathbb{N}$.

$$\forall_{g \in good \setminus prittles} \left(\sum_{t \in truck} v_{g,t} \right) = p(g)$$

- Only three trucks can contain skipples. Let t_i be the $i-$th truck, starting at $i = 1$.

$$\forall_{t_i \in truck, i>3}\ v_{skipples,t} = 0$$

- No two pallets of nuzzles may be in the same truck.

$$\forall_{t \in truck}\ v_{nuzzles,t} \leq 1$$

- Prittles and crottles are not allowed to be put in the same truck.

$$\forall_{t \in truck}\ v_{prittles,t} = 0 \vee v_{crottles,t} = 0$$

All of the constraints described above can be easily converted to Z3. In order to let Z3 automatically maximize the number of pallets with prittles we set the maximisation function to maximize the formula shown below.

$$\sum_{t \in truck} v_{prittles,t}$$

Running our program *without* the last constraint (question 1) gives us the result shown below. Every row is a truck, every column is the good starting with that letter. As can be seen from the table, all the pallets can be distributed and there are a total of $8 + 5 + 5 + 4 = 22$ prittles pallets.

```
    n s c d p
0 : 0 0 0 0 8
1 : 0 8 0 0 0
2 : 1 0 2 0 5
3 : 1 0 2 0 5
4 : 0 0 2 6 0
5 : 0 0 0 4 4
6 : 1 0 2 5 0
7 : 1 0 2 5 0
```

Running our program *with* the last constraint (question 2) gives us the result shown below. Again, all the pallets can be distributed and there are a total of $4 + 8 + 8 = 20$ prittles pallets.

```
    n s c d p
0 : 0 4 0 0 4
1 : 1 2 2 1 0
2 : 0 2 2 4 0
3 : 0 0 0 0 8
4 : 1 0 2 5 0
5 : 1 0 2 5 0
6 : 1 0 2 5 0
7 : 0 0 0 0 8
```

Our program and formalisation are generalised under the number of trucks, the truck maximum weight and maximum number of pallets. It is also very easy to add more goods, change their weight or change the number of pallets.

3 Chip design

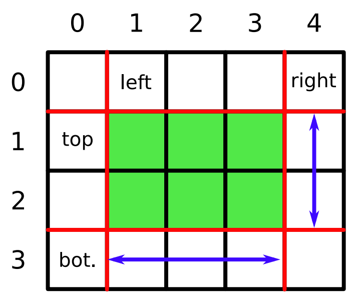
We consider the chip as a 30×30 grid. We number the rows from top to bottom, so the top row has index 0 and the bottom row has index 29. The columns are numbered the same way from left tot right.

3.1 variables

For each component i we introduce 4 integer variables: $\texttt{left}_i$, $\texttt{right}_i$, $\texttt{top}_i$ and $\texttt{bottom}_i$. These variables indicate where the edges of component i are.

$\texttt{left}_i$	The left-most column that contains component i
$\texttt{right}_i$	The left-most column to the right of component i
$\texttt{top}_i$	The top-most column that contains component i
$\texttt{bottom}_i$	The top-most column below component i

Note that $\texttt{left}_i$ is inclusive; the corresponding column contains component i.
However, $\texttt{right}_i$ is exclusive; the corresponding column does not contain component i.
Inclusive-exclusive indexing ensures that $\texttt{right}_i - \texttt{left}_i$ is the width of component i.
We also use this indexing for the rows, so $\texttt{bottom}_i - \texttt{top}_i$ is the height of component i.
Component s consists of the cells (r, c) with $\texttt{left}_i \leq c < \texttt{right}_i$ and $\texttt{top}_i \leq r < \texttt{bottom}_i$



$$\texttt{left}_i = 1 \qquad \texttt{right}_i = 4$$
$$\texttt{top}_i = 1 \qquad \texttt{bottom}_i = 3$$

$$\texttt{width}_i = \texttt{right}_i - \texttt{left}_i = 4 - 1 = 3$$
$$\texttt{height}_i = \texttt{bottom}_i - \texttt{top}_i = 3 - 1 = 2$$

3.2 Constraints

1. We want every component to be placed inside the 30×30 grid.
   For this we add four constrains for each component:

   $$0 \leq \texttt{left}_i \qquad \texttt{right}_i \leq 30 \qquad 0 \leq \texttt{top}_i \qquad \texttt{bottom}_i \leq 30$$

   For every cell (r, c) of component i we then have:

   $$0 \leq \texttt{left}_i \leq c < \texttt{right}_i \leq 30 \qquad 0 \leq \texttt{top}_i \leq r < \texttt{bottom}_i \leq 30$$

   So each cell of the component is within the grid.

2. We want each component to have the correct size.
   Suppose component i should have size (w, h) then we add the constraint:

   $$\begin{gathered}(\texttt{right}_i - \texttt{left}_i = w \ \wedge \ \texttt{bottom}_i - \texttt{top}_i = h) \vee \\ (\texttt{right}_i - \texttt{left}_i = h \ \wedge \ \texttt{bottom}_i - \texttt{top}_i = w)\end{gathered}$$

   The first line says that the component has the right size when it is not rotated. The second line says that the component has the right size if it is rotated 90°. Because we want the component to have the right size in either orientation, we combine these clauses with an $\vee$.

3. We do not want any of the components to overlap. For every pair of components $i < j$ we add the following constraint:

   $$\texttt{right}_i \leq \texttt{left}_j \ \vee \ \texttt{left}_i \geq \texttt{right}_j \ \vee \ \texttt{bottom}_i \leq \texttt{top}_j \ \vee \ \texttt{top}_i \geq \texttt{bottom}_j$$

   This constrains say that component j is entirely to the right, to the left, below or above component i. In each of these cases there is no overlap possible.

4. We want every regular component to share part of an edge with a power component.
   For every regular component i we add the following constraint:

   $$\bigvee_{p \in P} \left(\begin{array}{l} (\texttt{right}_i = \texttt{left}_p \ \wedge \ \texttt{top}_i < \texttt{bottom}_p \ \wedge \ \texttt{top}_p < \texttt{bottom}_i) \vee \\ (\texttt{left}_i = \texttt{right}_p \ \wedge \ \texttt{top}_i < \texttt{bottom}_p \ \wedge \ \texttt{top}_p < \texttt{bottom}_i) \vee \\ (\texttt{bottom}_i = \texttt{top}_p \ \wedge \ \texttt{left}_i < \texttt{right}_p \ \wedge \ \texttt{left}_p > \texttt{right}_i) \vee \\ (\texttt{top}_i = \texttt{bottom}_p \ \wedge \ \texttt{left}_i < \texttt{right}_p \ \wedge \ \texttt{left}_p > \texttt{right}_i) \end{array} \right)$$

   This statement starts with a big or over the power components. This ensures that there is a power component p that touches component i.
   Within the parenthesis we find four cases: component i touches the left, the right, the top or the bottom of power component p. To check if two components touch we need two parts.
   First, both components need to touch a vertical or horizontal line from different sides. This is ensured by the first condition of every case.

Second, they need to share a part of that line, which is ensured by the other conditions. We will only explain these other conditions for the first case.

In the first case the components touch a vertical line. For the components to share a part of this line, there must be a row r that contains both components.
This line must satisfy $\texttt{top}_i \leq r < \texttt{bottom}_i$ and $\texttt{top}_p \leq r < \texttt{bottom}_p$
We can combine these inequalities to form the following four conditions:

$$\texttt{top}_i < \texttt{bottom}_i \qquad \texttt{top}_i < \texttt{bottom}_p \qquad \texttt{top}_p < \texttt{bottom}_i \qquad \texttt{top}_p < \texttt{bottom}_p$$

The first and last condition are always satisfied (by part 2 and $0 < w, h$).
The other conditions are added to the first case to guarantee overlap.

5. The centers of the power components are far enough apart For every pair $p < q$ of power components we add the following constraint:

$$\begin{aligned}
&((\texttt{top}_p + \texttt{bottom}_p) - (\texttt{top}_q + \texttt{bottom}_q) \geq 2 \cdot D) \vee \\
&((\texttt{top}_q + \texttt{bottom}_q) - (\texttt{top}_p + \texttt{bottom}_p) \geq 2 \cdot D) \vee \\
&((\texttt{left}_p + \texttt{right}_p) - (\texttt{left}_q + \texttt{right}_q) \geq 2 \cdot D) \vee \\
&((\texttt{left}_q + \texttt{right}_q) - (\texttt{left}_p + \texttt{right}_p) \geq 2 \cdot D)
\end{aligned}$$

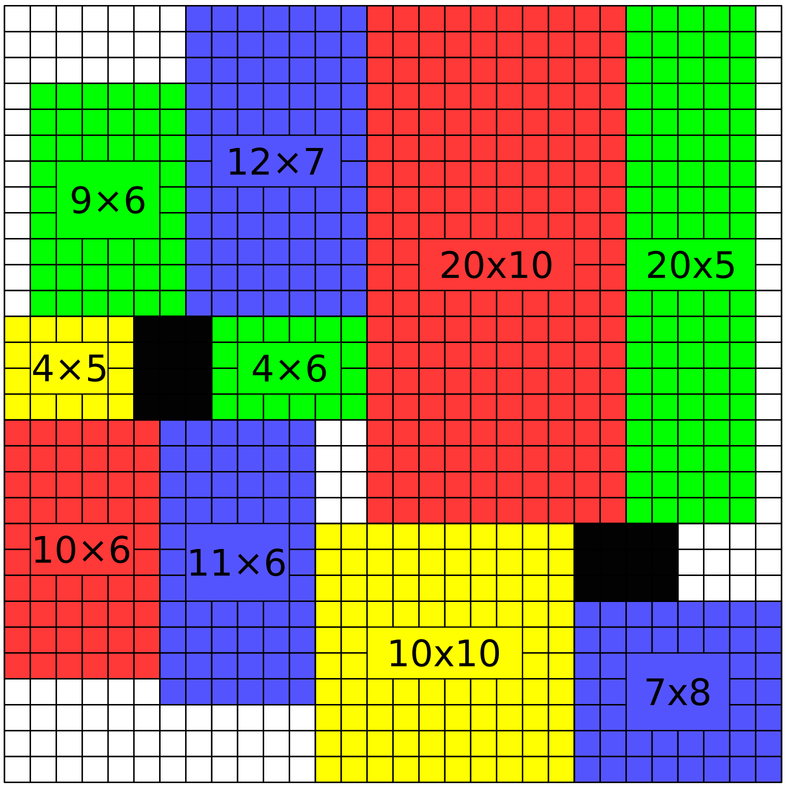
There are four cases: the center of component p is D units below, above, to the left or to the right of the center of component q. To ensure the vertical distance between the centers of two components we could use the following formula:

$$|(\texttt{top}_p + \texttt{bottom}_p)/2 - (\texttt{top}_q + \texttt{bottom}_q)/2| \geq D$$

However we would like a formula without division or absolute values. To get rid of the division we multiply everything by 2. We remove the absolute value by splitting the equation into two cases. This equation then becomes the first two cases of our constraint. The other two cases are found by considering the horizontal distance.

3.3 Result

The largest D for which we can find a result is 17.5. For $D = 17.5$ it finds a result in 1.865 seconds. For $D = 18$ it proves that it is unsatisfiable in 67.782 seconds.
A solution for $D = 17.5$ is illustrated below:



4 Dinner

5 Program safety

In order to test whether it is possible for the program to crash we use the SMT solver to check if the crashing statement can be reached. If the crash cannot be reached the formula will be unsatisfiable. If the crash can be reached the formula will be satisfiable and the variable assignment will provide us a value for the if-statement in every iteration.

We start by creating a variable $l = 10$ for the number of iterations the loop will make. We also create $l + 1$ integer variables a_i and b_i, where a_0 and b_0 are the initial values of a and b, the others describe the values of a and b at every iteration of the loop. Lastly, we create l boolean c_i variables to denote the value of the if-statement at iteration i.

1. We start with adding the constraints for the initial values of a and b.

$$a_0 = 1 \wedge b_0 = 1$$

2. If the if-statement is true we want to update the values of a and b accordingly. This is done using the following constraint.

$$\forall_{1\leq i\leq l+1}\,(c_{i-1} \rightarrow a_i = a_{i-1} + 2 \cdot b_{i-1} \wedge b_i = b_{i-1} + i)$$

3. If the if-statement is false we want to update the values of a and b accordingly. This is done using the following constraint.

$$\forall_{1 \leq i \leq l+1} (\neg c_{i-1} \rightarrow b_i = a_{i-1} + b_{i-1} \wedge a_i = a_{i-1} + i)$$

4. At last we want to check whether the program can crash, such that SAT $\leftrightarrow$ crash.

$$b_l = 700 + n$$

Running our program with all of these constraints yields the output shown below. The path shows the consecutive evaluations of the if-statement, with `f` as false and `t` as true, which leads to a crash.

```
n = 1 is sat -> crash
path ffttttftff
n = 2 is unsat
n = 3 is unsat
n = 4 is sat -> crash
path ttfftftttf
n = 5 is unsat
n = 6 is sat -> crash
path tftffffttff
n = 7 is unsat
n = 8 is sat -> crash
path ttftftftft
n = 9 is unsat
n = 10 is sat -> crash
path ttfttftttf
```