Formal Methods in Software Development, WS 2018. Lab 5

Formalize and solve, using Z3 the following problem[1]. Assume that we have three virtual machines (VMs) which require 100, 50 and 15 GB hard disk respectively. There are three servers with capabilities 100, 75 and 200 GB in that order. Find out a way to place VMs into servers in order to:

1. Minimize the number of servers used.
2. Minimize the operation cost (the servers have fixed daily costs 10, 5 and 20 USD respectively.)

Hints: Let us denote x_{ij} denote that VM i is put into server j and y_j denote that server j is in use. We need to express the followings.

1. A virtual machine is on exactly one server:

$$x_{i1} + x_{i2} + x_{i3} = 1, \quad \forall i \in \{1, ..., 3\}$$

2. An used server is implied by having a VM on it:

$$(x_{1j} = 1) \vee (x_{2j} = 1) \vee (x_{3j} = 1) \Rightarrow (y_j = 1), \quad \forall j \in \{1, ..., 3\}$$

3. Capability constraints

$$100x_{11} + 50x_{21} + 15x_{31} \leq 100y_1$$
$$100x_{12} + 50x_{22} + 15x_{32} \leq 75y_2$$
$$100x_{13} + 50x_{23} + 15x_{33} \leq 200y_3$$

Solution: There are various ways of encoding the variables of the problem:

(Variant 1**)** as integers

(Variant 3**)** as real

(Variant 2**)** as bool

(Variant 4**)** using `assert-soft` constraints

Variant 1. We declare each variable as integer, e.g.:

```
(declare-const x11 Int)
```

We also need to ensure that variables are 0/1, e.g.:

```
(assert (and (>= x11 0) (>= x12 0) (>= x13 0) (>= x21 0) ...
(assert (and (<= y1 1) (<= y2 1) (<= y3 1)))
```

Another variant for encoding the 0/1 integers is:

```
(assert (or (>= x11 0) (<= x11 1)
```

[1]From the tutorial on Z3 on objective functions

```
(assert (or (>= x12 0) (<= x12 1)
```

...

Constraint of type 2 can be encoded in 2 ways. For example:

```
(assert (and (>= y1 x11) (>= y1 x21) (>= y1 x31)))
```

...

Or as:

```
(assert (implies (= y1 1) (or (= x11 1) (= x21 1) (= x31 1) ))
```

...

Variant 2. We declare each variable as real. The constraints should be the same as for the integer encoding

Variant 3. We declare each variable as bool. The capability constraints require only integer/real variables, so we need to transform the bool variables into integer/real. This can be done by declaring a function as follows:

```
(define-fun bool_to_int ((b Bool)) Int (ite b 1 0) )
```

and cast the bool variables to int/real, e.g.

```
(assert (<= (+ (* 100 (bool_to_int x11)) (* 50 (bool_to_int x21)) (* 15 (bool_to_int x31))) (*
```

Variant 4. Another variant is to use `assert-soft` constraints. In an optimization problem, there can be constraints which must be fulfilled (hard constraints) or while others can be violated (soft constraints).

Z3 supports soft constraints. The `(assert-soft formula :weight numeral)` command asserts a weighted soft constraint. The weight must be a positive natural number, but is optional. If omitted, the weight is set to 1. For example: `(declare-const a1 Bool)`

```
(declare-const a2 Bool)
(declare-const a3 Bool)
(assert-soft a1 :weight 0.1)
(assert-soft a2 :weight 1.0)
(assert-soft a3 :weight 1)
(assert-soft (or (not a1) (not a2) (not a3)))
(check-sat)
(get-model)
```

It is also possible to declare multiple classes of soft assertions. To do this, use an optional tag to differentiate classes of soft assertions. For example: `(declare-const a Bool)`

```
(declare-const b Bool)
(declare-const c Bool)
(assert-soft a :weight 1 :id A)
(assert-soft b :weight 2 :id B)
(assert-soft c :weight 3 :id A)
(assert (= a c))
(assert (not (and a b)))
```

```
(check-sat)
(get-model)
(get-objectives)
```

For our problem, soft constraints can be used to encode the optimization goals:

```
(assert-soft (not y1) :id num_servers)
(assert-soft (not y2) :id num_servers)
(assert-soft (not y3) :id num_servers)

(assert-soft (not y1) :id costs :weight 10)
(assert-soft (not y2) :id costs :weight 5)
(assert-soft (not y3) :id costs :weight 20)
```

The `assert-soft` command represents MaxSMT (maximize the number of constraints which can be satisfied) which tries to maximize the weighted sum of boolean expressions belonged to the same id. Since we are doing minimization, negation is needed to take advantage of MaxSMT support.