

CR TP3 Programmation par contraintes

Ordonnancement de tâches sur deux machines

Julien LETOILE, Romain HUBERT

le 15/02/2021

Table des matières

Table des matières

I. [Réponses rédigées](#)

[Question 3.1](#)

[Question 3.2](#)

[Question 3.3](#)

[Question 3.4](#)

[Question 3.5](#)

[Question 3.6](#)

[Question 3.7](#)

[Question 3.8](#)

II. [Annexes](#)

[Code source](#)

I. Réponses rédigées

Question 3.1

```
1  taches(Taches):-  
2      Taches = [](tache(3, [], m1, _),  
3          tache(8, [], m1, _),  
4          tache(8, [4,5], m1, _),  
5          tache(6, [], m2, _),  
6          tache(3, [1], m2, _),  
7          tache(4, [1,7], m1, _),  
8          tache(8, [3,5], m1, _),  
9          tache(6, [4], m2, _),  
10         tache(6, [6,7], m2, _),  
11         tache(6, [9,12], m2, _),  
12         tache(3, [1], m2, _),  
13         tache(6, [7,8], m2, _)).
```

Test:

```
taches(T).
```

T =

Yes (0.00s cpu)

Question 3.2

```

1  /* Ecrit les tâches */
2
3  ecritTaches(Taches):-
4      (
5          foreachelem(T, Taches)
6          do
7              writeln(T)
8      ),
9      writeln(-----).

```

Test:

taches(T), ecritTaches(T).

tache(3, [], m1, _247)
 tache(8, [], m1, _252)
 tache(8, [4, 5], m1, _257)
 tache(6, [], m2, _266)
 tache(3, [1], m2, _271)
 tache(4, [1, 7], m1, _278)
 tache(8, [3, 5], m1, _287)
 tache(6, [4], m2, _296)
 tache(6, [6, 7], m2, _303)
 tache(6, [9, 12], m2, _312)
 tache(3, [1], m2, _321)
 tache(6, [7, 8], m2, _328)

T =

Yes (0.00s cpu)

Question 3.3

```
1  ?- local domain(machine(m1, m2)).
2
3  /*
4  On impose que le début est un entier positif, et que le début +
   durée est < fin.
5  On fixe aussi le domaine des machines pour plus tard.
6  */
7  domaine(Taches, Fin):-
8      (
9          foreach(lem(tache(Duree, _, Machine, Debut), Taches),
10             param(Fin)
11         do
12             Debut #>= 0,
13             Debut + Duree #=< Fin,
14             Machine &:: machine
15         ).
```

Test:

taches(T), domaine(T, F), ecritTaches(T).

```
tache(3, [], m1, _305{0 .. 1.0Inf})
tache(8, [], m1, _439{0 .. 1.0Inf})
tache(8, [4, 5], m1, _559{0 .. 1.0Inf})
tache(6, [], m2, _679{0 .. 1.0Inf})
tache(3, [1], m2, _799{0 .. 1.0Inf})
tache(4, [1, 7], m1, _919{0 .. 1.0Inf})
tache(8, [3, 5], m1, _1039{0 .. 1.0Inf})
tache(6, [4], m2, _1159{0 .. 1.0Inf})
tache(6, [6, 7], m2, _1279{0 .. 1.0Inf})
tache(6, [9, 12], m2, _1399{0 .. 1.0Inf})
tache(3, [1], m2, _1519{0 .. 1.0Inf})
tache(6, [7, 8], m2, _1640{0 .. 1.0Inf})
```

T =

Question 3.4

```
1  /* Récupérer les variables des taches et de Fin */
2  getVarlist(Taches, Fin, Liste):-
3      term_variables(Taches, L1),
4      term_variables(Fin, L2),
5      append(L2, L1, Liste).
6  /* Mettre L2 avant L1 => fixer Fin avant de générer le reste =>
   assure que Fin soit minimale */
```

Question 3.5

```
1  /* Le solve général */
2  solve(Taches, Fin):-
3      taches(Taches),
4      domaine(Taches, Fin),
5      precedences(Taches),
6      conflit(Taches),
7      getVarlist(Taches, Fin, Liste),
8      labeling(Liste),
9      ecritTaches(Taches).
```

Sans précedence et conflit, on a la réponse suivante:

solve(T, F).

T =

Sans precedence et conflit, on obtient le résultat ci-dessus. Or, on observe que plusieurs tâches utilisant la même machine commencent au même moment. Or cela ne respecte pas le sujet, donc, il va falloir mettre plus de contraintes (conflit et précédence).

Question 3.6

```
1  /*
2  Pour chaque élément, on fixe le début de la tâche après la fin
   des tâches précédentes.
3  */
4
5  precedences(Taches):-
6      (
7          foreachelem(tache(_, Noms, _, Debut), Taches),
8          param(Taches)
9      do
10         (
11             foreach(Tache, Noms),
12             param(Debut),
13             param(Taches)
14         do
15             tache(Duree, _, _, Deb) is Taches[Tache],
16             Debut #>= Duree + Deb
17         )
18     ).
```

Question 3.7

```
1  /*
2  On met des contraintes entre les tâches, telles que le début de
   l'une soit après la fin de l'autre.
3  Il faut cependant faire attention de ne pas mettre des
   contraintes sur soi même, sinon il n'y a pas de solutions.
4  */
```

```

5  conflit(Taches):-
6      (
7          for(I, 1, 12),
8          param(Taches)
9      do
10         (
11             for(J, I+1, 12),
12             param(I),
13             param(Taches)
14         do
15             tache(Duree1, _, Machine1, Debut1) is Taches[I],
16             tache(Duree2, _, Machine2, Debut2) is Taches[J],
17             (Machine1 &= Machine2) => (Duree1 + Debut1 #=<
Debut2 or Debut2 + Duree2 #=< Debut1)
18         )
19     ).

```

Ainsi grâce à ces contraintes, on a:

```
solve(T, F).
```

T =

F = 43

Yes (0.03s cpu, solution 1, maybe more) ?

Question 3.8

Notre solution est la meilleure car l'algorithme teste tous les agencements possibles des tâches en fonction de la valeur de Fin croissante.

Ainsi il augmente la valeur de Fin tant que les contraintes ne sont pas satisfaites, jusqu'à nous proposer la valeur minimale pour laquelle ça marche.

II. Annexes

Code source

```
1  :-lib(ic).
2  :-lib(ic_symbolic).
3
4  ?- local domain(machine(m1, m2)).
5
6  /* Q 3.5 */
7
8  solve(Taches, Fin):-
9      taches(Taches),
10     domaine(Taches, Fin),
11     precedences(Taches),
12     conflit(Taches),
13     getVarlist(Taches, Fin, Liste),
14     labeling(Liste),
15     ecritTaches(Taches).
16
17 /* Q3.1 */
18
19 taches(Taches):-
20     Taches = [](tache(3, [], m1, _),
21                tache(8, [], m1, _),
22                tache(8, [4,5], m1, _),
23                tache(6, [], m2, _),
24                tache(3, [1], m2, _),
25                tache(4, [1,7], m1, _),
26                tache(8, [3,5], m1, _),
27                tache(6, [4], m2, _),
28                tache(6, [6,7], m2, _),
29                tache(6, [9,12], m2, _),
30                tache(3, [1], m2, _),
31                tache(6, [7,8], m2, _)).
32
33 /*
34
```

```

35  T = [](tache(3, [], m1, _185), tache(8, [], m1, _190), tache(8,
    [4, 5], m1, _195), tache(6, [], m2, _204), tache(3, [1], m2,
    _209), tache(4, [1, 7], m1, _216), tache(8
36  , [3, 5], m1, _225), tache(6, [4], m2, _234), tache(6, [6, 7],
    m2, _241), tache(6, [9, 12], m2, _250), tache(3, [1], m2, _259),
    tache(6, [7, 8], m2, _266))
37  Yes (0.00s cpu)
38
39  */
40
41  /* Q 3.2 */
42
43  ecritTaches(Taches):-
44      (
45          foreachelem(T, Taches)
46          do
47              writeln(T)
48      ),
49      writeln(-----).
50
51  /*
52
53  tache(3, [], m1, _247)
54  tache(8, [], m1, _252)
55  tache(8, [4, 5], m1, _257)
56  tache(6, [], m2, _266)
57  tache(3, [1], m2, _271)
58  tache(4, [1, 7], m1, _278)
59  tache(8, [3, 5], m1, _287)
60  tache(6, [4], m2, _296)
61  tache(6, [6, 7], m2, _303)
62  tache(6, [9, 12], m2, _312)
63  tache(3, [1], m2, _321)
64  tache(6, [7, 8], m2, _328)
65  -----
66

```

```

67  T = [](tache(3, [], m1, _247), tache(8, [], m1, _252), tache(8,
    [4, 5], m1, _257), tache(6, [], m2, _266), tache(3, [1], m2,
    _271), tache(4, [1, 7], m1, _278), tache(8, [3, 5], m1, _287),
    tache(6, [4], m2, _296), tache(6, [6, 7], m2, _303), tache(6,
    [9, 12], m2, _312), tache(3, [1], m2, _321), tache(6, [7, 8],
    m2, _328))
68  Yes (0.00s cpu)
69
70  */
71
72  /* Q3.3 */
73
74  domaine(Taches, Fin):-
75      (
76          foreachelem(tache(Duree, _, Machine, Debut), Taches),
77          param(Fin)
78      do
79          Debut #>= 0,
80          Debut + Duree #=< Fin,
81          Machine &:: machine
82      ).
83
84  /*
85  [eclipse 18]: resoudre(T).
86  tache(3, [], m1, _305{0 .. 1.0Inf})
87  tache(8, [], m1, _439{0 .. 1.0Inf})
88  tache(8, [4, 5], m1, _559{0 .. 1.0Inf})
89  tache(6, [], m2, _679{0 .. 1.0Inf})
90  tache(3, [1], m2, _799{0 .. 1.0Inf})
91  tache(4, [1, 7], m1, _919{0 .. 1.0Inf})
92  tache(8, [3, 5], m1, _1039{0 .. 1.0Inf})
93  tache(6, [4], m2, _1159{0 .. 1.0Inf})
94  tache(6, [6, 7], m2, _1279{0 .. 1.0Inf})
95  tache(6, [9, 12], m2, _1399{0 .. 1.0Inf})
96  tache(3, [1], m2, _1519{0 .. 1.0Inf})
97  tache(6, [7, 8], m2, _1640{0 .. 1.0Inf})
98  -----
99

```

```

00  T = [(tache(3, [], m1, _305{0 .. 1.0Inf}), tache(8, [], m1,
    _439{0 .. 1.0Inf}), tache(8, [4, 5], m1, _559{0 .. 1.0Inf}),
    tache(6, [], m2, _679{0 .. 1.0Inf}), tache(3, [1], m2, _799{0 ..
    1.0Inf}), tache(4, [1, 7], m1, _919{0 .. 1.0Inf}), tache(8, [3,
    5], m1, _1039{0 .. 1.0Inf}), tache(6, [4], m2, _1159{0 ..
    1.0Inf}), tache(6, [6, 7], m2, _1279{0 .. 1.0Inf}), tache(6, [9,
    12], m2, _1399{0 .. 1.0Inf}), tache(3, [1], m2, _1519{0 ..
    1.0Inf}), tache(6, [7, 8], m2, _1640{0 .. 1.0Inf}))

01
02  -----
03
04      _305{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -3
05      _439{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -8
06      _559{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -8
07      _679{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -6
08      _799{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -3
09      _919{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -4
10      _1039{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -8
11      _1159{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -6
12      _1279{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -6
13      _1519{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -3
14      _1640{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -6
15  Yes (0.00s cpu)
16  */
17
18
19  /* Q3.4 */
20
21  getVarlist(Taches, Fin, Liste):-
22      term_variables(Taches, L1),
23      term_variables(Fin, L2),
24      append(L2, L1, Liste). /* Mettre L2 avant L1 => fixer Fin
    avant de générer le reste => assure que Fin soit minimale */
25
26
27  /* Q 3.5 */
28
29  /*
30  solve(Taches, Fin).
31  tache(3, [], m1, _312{0 .. 1.0Inf})
32  tache(8, [], m1, _446{0 .. 1.0Inf})

```

```

33  tache(8, [4, 5], m1, _566{0 .. 1.0Inf})
34  tache(6, [], m2, _686{0 .. 1.0Inf})
35  tache(3, [1], m2, _806{0 .. 1.0Inf})
36  tache(4, [1, 7], m1, _926{0 .. 1.0Inf})
37  tache(8, [3, 5], m1, _1046{0 .. 1.0Inf})
38  tache(6, [4], m2, _1166{0 .. 1.0Inf})
39  tache(6, [6, 7], m2, _1286{0 .. 1.0Inf})
40  tache(6, [9, 12], m2, _1406{0 .. 1.0Inf})
41  tache(3, [1], m2, _1526{0 .. 1.0Inf})
42  tache(6, [7, 8], m2, _1647{0 .. 1.0Inf})
43  -----
44
45  Taches = [(tache(3, [], m1, 0), tache(8, [], m1, 0), tache(8,
    [4, 5], m1, 0), tache(6, [], m2, 0), tache(3, [1], m2, 0),
    tache(4, [1, 7], m1, 0), tache(8, [3, 5], m1,
46    0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0), tache(6, [9,
    12], m2, 0), tache(3, [1], m2, 2), tache(6, [7, 8], m2, 1))
47  ...
48  */
49
50  /* Q3.6 */
51
52  precedences(Taches):-
53      (
54          foreach(elem(tache(_, Noms, _, Debut), Taches),
55              param(Taches)
56              do
57                  (
58                      foreach(Tache, Noms),
59                      param(Debut),
60                      param(Taches)
61                      do
62                          tache(Duree, _, _, Deb) is Taches[Tache],
63                          Debut #>= Duree + Deb
64                  )
65              ).
66
67  /* Q3.7 */
68
69  conflit(Taches):-
70      (

```

```

71         for(I, 1, 12),
72             param(Taches)
73     do
74         (
75             for(J, I+1, 12),
76                 param(I),
77                 param(Taches)
78             do
79                 tache(Duree1, _, Machine1, Debut1) is Taches[I],
80                 tache(Duree2, _, Machine2, Debut2) is Taches[J],
81                 (Machine1 &= Machine2) => (Duree1 + Debut1 #=<
Debut2 or Debut2 + Duree2 #=< Debut1)
82             )
83         ).
84
85     /*
86     solve(T, F).
87     T = [(tache(3, [], m1, 0), tache(8, [], m1, 29), tache(8, [4,
5], m1, 9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4,
[1, 7], m1, 25), tache(8, [3, 5], m1, 17), tache(6, [4], m2, 9),
tache(6, [6, 7], m2, 31), tache(6, [9, 12], m2, 37), tache(3,
[1], m2, 15), tache(6, [7, 8], m2, 25))
88     F = 43
89     Yes (0.03s cpu, solution 1, maybe more) ?
90     */
91
92     /* Q3.8 */
93
94     /* Notre solution est la meilleure car l'algorithme teste tous
les agencements possibles des taches en fonction de la valeur de
Fin croissante.
95     Ainsi il augmente la valeur de Fin tant que les contraintes ne
sont pas satisfaites, jusqu'à nous proposer la valeur minimale
pour laquelle ça marche.
96
97     */

```

