CR TP3 Programmation par contraintes

Ordonnancement de tâches sur deux machines

Julien LETOILE, Romain HUBERT

le 15/02/2021

Table des matières

I. Réponses rédigées

Question 3.1

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

Test:

> taches(T).
>
> T =
> Yes (0.00s cpu)

Question 3.2

```
/* Ecrit les tâches */

ecritTaches(Taches):-
    (
        foreachelem(T, Taches)
    do
        writeln(T)
    ),
    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

```
?- local domain(machine(m1, m2)).

/*
On impose que le début est un entier positif, et que le début + durée est < fin.
On fixe aussi le domaine des machines pour plus tard.
*/
domaine(Taches, Fin):-
    (
        foreachelem(tache(Duree, _, Machine, Debut), Taches),
        param(Fin)
    do
        Debut #>= 0,
        Debut + Duree #=< Fin,
        Machine &:: machine
    ).
```

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

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

Question 3.5

```
/* Le solve général */
solve(Taches, Fin):-
    taches(Taches),
    domaine(Taches, Fin),
    precedences(Taches),
    conflit(Taches),
    getVarlist(Taches, Fin, Liste),
    labeling(Liste),
    ecritTaches(Taches).
```

Sans précédence 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

```
/*
Pour chaque élement, on fixe le début de la tâche après la fin
des tâches précédentes.
*/

precedences(Taches):-
    (
        foreachelem(tache(_, Noms, _, Debut), Taches),
        param(Taches)
    do
        (
            foreach(Tache, Noms),
            param(Debut),
            param(Taches)
        do
            tache(Duree, _, _, Deb) is Taches[Tache],
            Debut #>= Duree + Deb
        )
    ).
```

Question 3.7

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

```
conflit(Taches):-
    (
        for(I, 1, 12),
        param(Taches)
    do
        (
            for(J, I+1, 12),
            param(I),
            param(Taches)
        do
            tache(Duree1, _, Machine1, Debut1) is Taches[I],
            tache(Duree2, _, Machine2, Debut2) is Taches[J],
            (Machine1 &= Machine2) => (Duree1 + Debut1 #=<
Debut2 or Debut2 + Duree2 #=< Debut1)
        )
    ).
```

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 taches 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

```
:-lib(ic).
:-lib(ic_symbolic).

?- local domain(machine(m1, m2)).

/* Q 3.5 */

solve(Taches, Fin):-
    taches(Taches),
    domaine(Taches, Fin),
    precedences(Taches),
    conflit(Taches),
    getVarlist(Taches, Fin, Liste),
    labeling(Liste),
    ecritTaches(Taches).

/* Q3.1 */

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

/*

```

```
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
```

```
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))
Yes (0.00s cpu)

*/

/* Q3.3 */

domaine(Taches, Fin):-
    (
        foreachelem(tache(Duree, _, Machine, Debut), Taches),
        param(Fin)
    do
        Debut #>= 0,
        Debut + Duree #=< Fin,
        Machine &:: machine
    ).

/*
[eclipse 18]: resoudre(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 = [](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}))

----------------------

        _305{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -3
        _439{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -8
        _559{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -8
        _679{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -6
        _799{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -3
        _919{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -4
        _1039{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -8
        _1159{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -6
        _1279{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -6
        _1519{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -3
        _1640{0 .. 1.0Inf} - _357{8 .. 1.0Inf} #=< -6
Yes (0.00s cpu)
*/


/* Q3.4 */

getVarlist(Taches, Fin, Liste):-
    term_variables(Taches, L1),
    term_variables(Fin, L2),
    append(L2, L1, Liste). /* Mettre L2 avant L2 => fixer Fin
avant de générer le reste => assure que Fin soit minimale */


/* Q 3.5 */

/*
solve(Taches, Fin).
tache(3, [], m1, _312{0 .. 1.0Inf})
tache(8, [], m1, _446{0 .. 1.0Inf})
```

```
tache(8, [4, 5], m1, _566{0 .. 1.0Inf})
tache(6, [], m2, _686{0 .. 1.0Inf})
tache(3, [1], m2, _806{0 .. 1.0Inf})
tache(4, [1, 7], m1, _926{0 .. 1.0Inf})
tache(8, [3, 5], m1, _1046{0 .. 1.0Inf})
tache(6, [4], m2, _1166{0 .. 1.0Inf})
tache(6, [6, 7], m2, _1286{0 .. 1.0Inf})
tache(6, [9, 12], m2, _1406{0 .. 1.0Inf})
tache(3, [1], m2, _1526{0 .. 1.0Inf})
tache(6, [7, 8], m2, _1647{0 .. 1.0Inf})
----------------------------------------

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,
 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))
...
*/

/* Q3.6 */

precedences(Taches):-
    (
        foreachelem(tache(_, Noms, _, Debut), Taches),
        param(Taches)
    do
        (
            foreach(Tache, Noms),
            param(Debut),
            param(Taches)
        do
            tache(Duree, _, _, Deb) is Taches[Tache],
            Debut #>= Duree + Deb
        )
    ).

/* Q3.7 */

conflit(Taches):-
    (
```

```
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  */
```