Rapport Travaux Pratiques : Programmation par Contraintes - TP 3 : **Contraintes Logiques**

Nicolas Desfeux
Aurélien Texier

15 mars 2011

Dans ce T.P., nous allons utiliser la programmation par contraintes pour résoudre un problème d'ordonnancement de tâches à effectuer sur deux machines.
Dans un premier temps, nous définirons les prédicats qui fixent les domaines dans lesquels nous travaillerons, puis nous ajouterons les contraintes liées au fait que les tâches doivent être effectuées seulement après que certaines autres soient faites. Enfin, nous finirons par une dernière contraintes qui vise à empêcher que deux tâches se fassent simultanément sur la même machine.

Question 3.1 Nous définissons ici un prédicat *taches(?Taches)* qui unifie Taches au tableau des tâches.

Listing 1 – "taches"

```
1  taches(Taches) :-        Taches =        [](tache(3,[],m1,_),
2                                            tache(8,[],m1,_),
3                                            tache(8,[4,5],m1,_),
4                                            tache(6,[],m2,_),
5                                            tache(3,[1],m2,_),
6                                            tache(4,[1,7],m1,_),
7                                            tache(8,[3,5],m1,_),
8                                            tache(6,[4],m2,_),
9                                            tache(6,[6,7],m2,_),
10                                           tache(6,[9,12],m2,_),
11                                           tache(3,[1],m2,_),
12                                           tache(6,[7,8],m2,_)).
13
14 /*Test
15
16 [eclipse 3]: taches(T).
17
18 T = [tache(3, [], m1, _169), tache(8, [], m1, _176), tache(8, [4, 5], m1,
       _183), tache(6, [], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
       m1, _210), tache(8, [3, 5], m1, _221), tache(6, [4], m2, _232), tache(6,
```

```
    [6, 7], m2, _241), tache(6, [9, 12], m2, _252), tache(3, [1], m2, _263),
    tache(6, [7, 8], m2, _272)]
Yes (0.00s cpu)

*/
```

Question 3.2 Nous définissons ici un prédicat *affiche(+Taches)* qui affiche chaque élément, à savoir chaque tâche constituant le problème. Nous définiront ce prédicat à l'aide d'un itérateur.

Listing 2 – "affiche"

```
affiche(Taches) :-      dim(Taches,[Dim]),
                        (for(Indice,1,Dim),param(Taches)
                                                do
                                                        Elem is Taches[Indice
                                                            ],
                                                        writeln(Elem)
                                                ).


/*
[eclipse 4]: taches(T),affiche(T).
tache(3, [], m1, _235)
tache(8, [], m1, _242)
tache(8, [4, 5], m1, _249)
tache(6, [], m2, _260)
tache(3, [1], m2, _267)
tache(4, [1, 7], m1, _276)
tache(8, [3, 5], m1, _287)
tache(6, [4], m2, _298)
tache(6, [6, 7], m2, _307)
tache(6, [9, 12], m2, _318)
tache(3, [1], m2, _329)
tache(6, [7, 8], m2, _338)

T = [tache(3, [], m1, _235), tache(8, [], m1, _242), tache(8, [4, 5], m1,
    _249), tache(6, [], m2, _260), tache(3, [1], m2, _267), tache(4, [1, 7],
    m1, _276), tache(8, [3, 5], m1, _287), tache(6, [4], m2, _298), tache(6,
    [6, 7], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
    tache(6, [7, 8], m2, _338)]
Yes (0.00s cpu)

*/
```

Question 3.3 Nous définissons ici un prédicat *domaines(+Taches, ?Fin)* qui contraint chaque tâche à commencer après l'instant 0 et à finir avant Fin, variable qui correspond à l'instant où toutes les tâches sont terminées.

Listing 3 – "domaines"

```
domaines(Taches,Fin) :- dim(Taches,[Dim]),
                        (for(Indice,1,Dim),param(Taches,Fin)
```

```
3                                                   do
4                                                           tache(Duree,_Nom,
                                                                _Machine,Debut)
                                                                is Taches[Indice
                                                                ],
5                                                           Debut + Duree #=< Fin
                                                                ,
6                                                           Debut #>= 0
7                                                   ).
8
9   /*
10  [eclipse 5]: taches(T),domaines(T,10).
11
12  T = [tache(3, [], m1, _421{0 .. 7}), tache(8, [], m1, _644{0 .. 2}), tache(8,
        [4, 5], m1, _867{0 .. 2}), tache(6, [], m2, _1090{0 .. 4}), tache(3,
        [1], m2, _1313{0 .. 7}), tache(4, [1, 7], m1, _1536{0 .. 6}), tache(8,
        [3, 5], m1, _1759{0 .. 2}), tache(6, [4], m2, _1982{0 .. 4}), tache(6,
        [6, 7], m2, _2205{0 .. 4}), tache(6, [9, 12], m2, _2428{0 .. 4}), tache
        (3, [1], m2, _2651{0 .. 7}), tache(6, [7, 8], m2, _2874{0 .. 4})]
13
14  There are 12 delayed goals. Do you want to see them? (y\n)
15  Yes (0.00s cpu)
16  */
```

Question 3.4 Voici le prédicat *getVarList(+Taches, ?Fin, ?List)* qui permet de récupérer la liste des variables du problème.

Listing 4 – "getVarList"

```
1   getVarList(Taches,Fin,ListFin):- dim(Taches,[Dim]),
2                                    (for(Indice,1,Dim),fromto([],In,Out,List),
                                         param(Taches)
3                                    do
4                                            Xi is Taches[Indice],
5                                            Xi = tache(_,_,_,Debut),
6                                            Out=[Debut|In]
7                                    ),
8                                    ListFin=[Fin|List].
9
10
11  /*
12  [eclipse 6]: taches(T), getVarList(T, Fin, L).
13
14  T = [](tache(3, [], m1, _238), tache(8, [], m1, _243), tache(8, [4, 5], m1,
        _248), tache(6, [], m2, _257), tache(3, [1], m2, _262), tache(4, [1, 7],
        m1, _269), tache(8, [3, 5], m1, _278), tache(6, [4], m2, _287), tache(6,
        [6, 7], m2, _294), tache(6, [9, 12], m2, _303), tache(3, [1], m2, _312),
        tache(6, [7, 8], m2, _319))
15  Fin = Fin
16  L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
        _238]
17  Yes (0.00s cpu)
18
```

```
19
20 */
```

Question 3.5 On définit le prédicat *solve(?Taches, ?Fin)* qui permet, en utilisant les trois prédicats précédents, de trouver un ordonnancement qui respecte les contraintes de domaines définies.
Le test effectué atteste bien la conformité du prédicat car solve rend bien comme première solution toutes les tâches avec des débuts à 0, puis il incrémente chacune comme solutions suivantes. Il unifie Fin à 8 puisque c'est la durée de la tâche la plus longue, ce qui est logique aussi.

Listing 5 – "solve1"

```
1  solve1(Taches,Fin) :-    taches(Taches),
2                                                    domaines(Taches,Fin),
3                                                    getVarList(Taches,Fin,Liste),
4                                                    labeling(Liste),
5                                                    affiche(Taches).
6  /*
7  [eclipse 4]: solve1(Taches,Fin).
8  lists.eco  loaded in 0.00 seconds
9  tache(3, [], m1, 0)
10 tache(8, [], m1, 0)
11 tache(8, [4, 5], m1, 0)
12 tache(6, [], m2, 0)
13 tache(3, [1], m2, 0)
14 tache(4, [1, 7], m1, 0)
15 tache(8, [3, 5], m1, 0)
16 tache(6, [4], m2, 0)
17 tache(6, [6, 7], m2, 0)
18 tache(6, [9, 12], m2, 0)
19 tache(3, [1], m2, 0)
20 tache(6, [7, 8], m2, 0)
21
22 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, 0), tache(6, [7, 8], m2, 0))
23 Fin = 8
24 Yes (0.00s cpu, solution 1, maybe more) ? ;
25 tache(3, [], m1, 1)
26 tache(8, [], m1, 0)
27 tache(8, [4, 5], m1, 0)
28 tache(6, [], m2, 0)
29 tache(3, [1], m2, 0)
30 tache(4, [1, 7], m1, 0)
31 tache(8, [3, 5], m1, 0)
32 tache(6, [4], m2, 0)
33 tache(6, [6, 7], m2, 0)
34 tache(6, [9, 12], m2, 0)
35 tache(3, [1], m2, 0)
36 tache(6, [7, 8], m2, 0)
37
```

```
38 Taches = [](tache(3, [], m1, 1), 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, 0), tache(6, [7, 8], m2, 0))
39 Fin = 8
40 Yes (0.01s cpu, solution 2, maybe more) ? ;
41 tache(3, [], m1, 2)
42 tache(8, [], m1, 0)
43 tache(8, [4, 5], m1, 0)
44 tache(6, [], m2, 0)
45 tache(3, [1], m2, 0)
46 tache(4, [1, 7], m1, 0)
47 tache(8, [3, 5], m1, 0)
48 tache(6, [4], m2, 0)
49 tache(6, [6, 7], m2, 0)
50 tache(6, [9, 12], m2, 0)
51 tache(3, [1], m2, 0)
52 tache(6, [7, 8], m2, 0)
53
54 Taches = [](tache(3, [], m1, 2), 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, 0), tache(6, [7, 8], m2, 0))
55 Fin = 8
56 Yes (0.01s cpu, solution 3, maybe more) ?
57 */
```

Question 3.6 On définit ici un prédicat *precedences(+Taches)* qui contraint chaque tâche à démarrer après la fin de ses tâches préliminaires.
On modifie alors solve pour prendre en compte ces contraintes.
Les résultats de solve sont bien conformes car les deux premières tâches commencent à 0 puisqu'elles n'ont besoin d'aucune autre tâche effectuée au préalable pour s'effectuer. La tâche 5 qui a besoin de la tâche 1 effectuée commence bien à 3, qui est la durée de la tâche 1. Tout est donc correct.
On peut donc en conclure que sans la contrainte des machines différentes qui va suivre, la solution optimale prendrait 38 unités de temps pour se faire (car le solveur unifie Fin à 38).

Listing 6 – "precedences"

```
1  precedences(Taches) :-  dim(Taches,[Dim]),
2                          (for(Indice,1,Dim),param(Taches)
3                          do
4                                  Elem is Taches[Indice],
5                                  Elem = tache(_D,Noms,_M,Debut),
6                                  (foreach(I,Noms),param(Debut,Taches)
7                                  do
8                                          tache(Duree2,_N,_M,Debut2) is Taches[
                                               I],
9                                          Debut #>= Debut2+Duree2
10                                 )
11                         ).
```

```
12
13 solve(Taches,Fin) :-      taches(Taches),
14                                                  domaines(Taches,Fin),
15                                                  precedences(Taches),
16                                                  getVarList(Taches,Fin,Liste),
17                                                  labeling(Liste),
18                                                  affiche(Taches).
19
20 /*
21 [eclipse 44]: solve(T, Fin).
22 tache(3, [], m1, 0)
23 tache(8, [], m1, 0)
24 tache(8, [4, 5], m1, 6)
25 tache(6, [], m2, 0)
26 tache(3, [1], m2, 3)
27 tache(4, [1, 7], m1, 22)
28 tache(8, [3, 5], m1, 14)
29 tache(6, [4], m2, 6)
30 tache(6, [6, 7], m2, 26)
31 tache(6, [9, 12], m2, 32)
32 tache(3, [1], m2, 3)
33 tache(6, [7, 8], m2, 22)
34
35 T = [](tache(3, [], m1, 0), tache(8, [], m1, 0), tache(8, [4, 5], m1, 6),
       tache(6, [], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], m1, 22),
       tache(8, [3, 5], m1, 14), tache(6, [4], m2, 6), tache(6, [6, 7], m2, 26),
        tache(6, [9, 12], m2, 32), tache(3, [1], m2, 3), tache(6, [7, 8], m2,
       22))
36 Fin = 38
37
38 */
```

Question 3.7 Enfin, on définit le prédicat *conflits(+Taches)* qui impose que, sur une machine, deux tâches ne se déroulent pas en même temps.
On modifie solve de la même manière qu'à la question précédente pour obtenir une solution du problème prenant en compte cette dernière contrainte.
Enfin, avec cette dernière contrainte, on obtient une solution qui dure un peu plus longtemps, 43 unités de temps.

Listing 7 – "conflits"

```
1 conflits(Taches) :- dim(Taches,[Dim]),
2                     (for(Indice,1,Dim),param(Taches,Dim)
3                     do
4                         Elem is Taches[Indice],
5                         I2 is Indice+1,
6                         /*Il faut assurer que les indices soient
                              differents, sinon on va se retrouver a
                              comparer deux fois la meme taches*/
7                         (for(I,I2,Dim),param(Taches,Elem)
8                         do
9                             Elem2 is Taches[I],
```

```
10                                                  machinesDifferentes(Elem,Elem2)
11                                          )
12                                  ).
13
14  machinesDifferentes(tache(_,_,M1,_),tache(_,_,M2,_)):- \=(M1,M2),!.
15  machinesDifferentes(tache(Duree,_,M,Debut),tache(Duree2,_,M,Debut2)) :- ((
        Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).
16
17
18  solve2(Taches,Fin) :-    taches(Taches),
19                                                      domaines(Taches,Fin),
20                                                      precedences(Taches),
21                                                      conflits(Taches),
22                                                      getVarList(Taches,Fin,Liste),
23                                                      labeling(Liste),
24                                                      affiche(Taches).
25  /*
26  [eclipse 10]: solve2(Taches,Fin).
27  tache(3, [], m1, 0)
28  tache(8, [], m1, 29)
29  tache(8, [4, 5], m1, 9)
30  tache(6, [], m2, 0)
31  tache(3, [1], m2, 6)
32  tache(4, [1, 7], m1, 25)
33  tache(8, [3, 5], m1, 17)
34  tache(6, [4], m2, 12)
35  tache(6, [6, 7], m2, 31)
36  tache(6, [9, 12], m2, 37)
37  tache(3, [1], m2, 9)
38  tache(6, [7, 8], m2, 25)
39
40  Taches = [](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, 12), tache(6, [6, 7], m2, 31)
        , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
        25))
41  Fin = 43
42  Yes (0.01s cpu, solution 1, maybe more) ? ;
43  tache(3, [], m1, 1)
44  tache(8, [], m1, 29)
45  tache(8, [4, 5], m1, 9)
46  tache(6, [], m2, 0)
47  tache(3, [1], m2, 6)
48  tache(4, [1, 7], m1, 25)
49  tache(8, [3, 5], m1, 17)
50  tache(6, [4], m2, 12)
51  tache(6, [6, 7], m2, 31)
52  tache(6, [9, 12], m2, 37)
53  tache(3, [1], m2, 9)
54  tache(6, [7, 8], m2, 25)
55
56  Taches = [](tache(3, [], m1, 1), 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, 12), tache(6, [6, 7], m2, 31)
    , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
    25))
Fin = 43
Yes (0.01s cpu, solution 2, maybe more) ? ;
tache(3, [], m1, 2)
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, 12)
tache(6, [6, 7], m2, 31)
tache(6, [9, 12], m2, 37)
tache(3, [1], m2, 9)
tache(6, [7, 8], m2, 25)

Taches = [](tache(3, [], m1, 2), 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, 12), tache(6, [6, 7], m2, 31)
    , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
    25))
Fin = 43
Yes (0.01s cpu, solution 3, maybe more) ?
*/

/*
[eclipse 11]: solve2(Taches,42).

No (0.00s cpu)
*/
```

Question 3.8 Oui, la solution est la meilleure ! Prolog résoud les contraintes en incrémentant le début des tâches, jusqu'à obtenir le respect des contraintes. Comme il incrémente, la première trouvée est forcément la solution optimale au problème, à savoir l'ordonnancement des tâches au plus tôt.
Pour vérifier tout de même, nous avons, dans nos tests de la question précédente, effectué une requête de *solve2* avec Fin à 42 (au lieu de la solution donnée qui est 43), et le solveur nous répond No, ce qui justifie bien qu'il n'y a pas de solution plus courte pour ce problème d'ordonnancement.

1 Code Complet, avec l'ensemble des tests

Listing 8 – "TP3"

```
1  :-lib(ic).
2  :-lib(ic_symbolic).
3
4  taches(Taches) :-       Taches =        [](tache(3,[],m1,_),
5                                          tache(8,[],m1,_),
6                                          tache(8,[4,5],m1,_),
7                                          tache(6,[],m2,_),
8                                          tache(3,[1],m2,_),
9                                          tache(4,[1,7],m1,_),
10                                         tache(8,[3,5],m1,_),
11                                         tache(6,[4],m2,_),
12                                         tache(6,[6,7],m2,_),
13                                         tache(6,[9,12],m2,_),
14                                         tache(3,[1],m2,_),
15                                         tache(6,[7,8],m2,_)).
16
17 /*Test
18
19 [eclipse 3]: taches(T).
20
21 T = [tache(3, [], m1, _169), tache(8, [], m1, _176), tache(8, [4, 5], m1,
       _183), tache(6, [], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
       m1, _210), tache(8, [3, 5], m1, _221), tache(6, [4], m2, _232), tache(6,
       [6, 7], m2, _241), tache(6, [9, 12], m2, _252), tache(3, [1], m2, _263),
       tache(6, [7, 8], m2, _272)]
22 Yes (0.00s cpu)
23
24 */
25
26 affiche(Taches) :-      dim(Taches,[Dim]),
27                         (for(Indice,1,Dim),param(Taches)
28                                         do
29                                                 Elem is Taches[Indice
                                                       ],
30                                                 writeln(Elem)
31                                         ).
32
33
34 /*
35 [eclipse 4]: taches(T),affiche(T).
36 tache(3, [], m1, _235)
37 tache(8, [], m1, _242)
38 tache(8, [4, 5], m1, _249)
39 tache(6, [], m2, _260)
40 tache(3, [1], m2, _267)
41 tache(4, [1, 7], m1, _276)
42 tache(8, [3, 5], m1, _287)
43 tache(6, [4], m2, _298)
44 tache(6, [6, 7], m2, _307)
```

```
45 tache(6, [9, 12], m2, _318)
46 tache(3, [1], m2, _329)
47 tache(6, [7, 8], m2, _338)
48
49 T = [tache(3, [], m1, _235), tache(8, [], m1, _242), tache(8, [4, 5], m1,
       _249), tache(6, [], m2, _260), tache(3, [1], m2, _267), tache(4, [1, 7],
       m1, _276), tache(8, [3, 5], m1, _287), tache(6, [4], m2, _298), tache(6,
       [6, 7], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
       tache(6, [7, 8], m2, _338)]
50 Yes (0.00s cpu)
51
52 */
53
54
55 domaines(Taches,Fin) :- dim(Taches,[Dim]),
56                         (for(Indice,1,Dim),param(Taches,Fin)
57                                                 do
58                                                         tache(Duree,_Nom,
                                                               _Machine,Debut)
                                                               is Taches[Indice
                                                               ],
59                                                         Debut + Duree #=< Fin
                                                               ,
60                                                         Debut #>= 0
61                                                 ).
62
63 /*
64 [eclipse 5]: taches(T),domaines(T,10).
65
66 T = [tache(3, [], m1, _421{0 .. 7}), tache(8, [], m1, _644{0 .. 2}), tache(8,
        [4, 5], m1, _867{0 .. 2}), tache(6, [], m2, _1090{0 .. 4}), tache(3,
       [1], m2, _1313{0 .. 7}), tache(4, [1, 7], m1, _1536{0 .. 6}), tache(8,
       [3, 5], m1, _1759{0 .. 2}), tache(6, [4], m2, _1982{0 .. 4}), tache(6,
       [6, 7], m2, _2205{0 .. 4}), tache(6, [9, 12], m2, _2428{0 .. 4}), tache
       (3, [1], m2, _2651{0 .. 7}), tache(6, [7, 8], m2, _2874{0 .. 4})]
67
68 There are 12 delayed goals. Do you want to see them? (y\n)
69 Yes (0.00s cpu)
70 */
71
72
73 getVarList(Taches,Fin,ListFin):- dim(Taches,[Dim]),
74                                  (for(Indice,1,Dim),fromto([],In,Out,List),
                                       param(Taches)
75                                  do
76                                          Xi is Taches[Indice],
77                                          Xi = tache(_,_,_,Debut),
78                                          Out=[Debut|In]
79                                  ),
80                                  ListFin=[Fin|List].
81
82
83 /*
```

```
84  [eclipse 6]: taches(T), getVarList(T, Fin, L).
85
86  T = [](tache(3, [], m1, _238), tache(8, [], m1, _243), tache(8, [4, 5], m1,
        _248), tache(6, [], m2, _257), tache(3, [1], m2, _262), tache(4, [1, 7],
        m1, _269), tache(8, [3, 5], m1, _278), tache(6, [4], m2, _287), tache(6,
        [6, 7], m2, _294), tache(6, [9, 12], m2, _303), tache(3, [1], m2, _312),
        tache(6, [7, 8], m2, _319))
87  Fin = Fin
88  L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
        _238]
89  Yes (0.00s cpu)
90  */
91
92  solve(Taches,Fin) :-    taches(Taches),
93                                          domaines(Taches,Fin),
94                                          precedences(Taches),
95                                          getVarList(Taches,Fin,Liste),
96                                          labeling(Liste),
97                                          affiche(Taches).
98
99  precedences(Taches) :-  dim(Taches,[Dim]),
100                         (for(Indice,1,Dim),param(Taches)
101                         do
102                                 Elem is Taches[Indice],
103                                 Elem = tache(_D,Noms,_M,Debut),
104                                 (foreach(I,Noms),param(Debut,Taches)
105                                 do
106                                         tache(Duree2,_N,_M,Debut2) is Taches[
                                                I],
107                                         Debut #>= Debut2+Duree2
108                                 )
109                         ).
110
111 /*
112 [eclipse 44]: taches(T), solve(T, Fin).
113 tache(3, [], m1, 0)
114 tache(8, [], m1, 0)
115 tache(8, [4, 5], m1, 6)
116 tache(6, [], m2, 0)
117 tache(3, [1], m2, 3)
118 tache(4, [1, 7], m1, 22)
119 tache(8, [3, 5], m1, 14)
120 tache(6, [4], m2, 6)
121 tache(6, [6, 7], m2, 26)
122 tache(6, [9, 12], m2, 32)
123 tache(3, [1], m2, 3)
124 tache(6, [7, 8], m2, 22)
125
126 T = [](tache(3, [], m1, 0), tache(8, [], m1, 0), tache(8, [4, 5], m1, 6),
        tache(6, [], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], m1, 22),
        tache(8, [3, 5], m1, 14), tache(6, [4], m2, 6), tache(6, [6, 7], m2, 26),
         tache(6, [9, 12], m2, 32), tache(3, [1], m2, 3), tache(6, [7, 8], m2,
        22))
```

```
127 Fin = 38
128 */
129
130 conflits(Taches) :- dim(Taches,[Dim]),
131                     (for(Indice,1,Dim),param(Taches,Dim)
132                     do
133                             Elem is Taches[Indice],
134                             I2 is Indice+1,
135                             /*Il faut assurer que les indices soient
                                    differents, sinon on va se retrouver a
                                    comparer deux fois la meme taches*/
136                             (for(I,I2,Dim),param(Taches,Elem)
137                             do
138                                     Elem2 is Taches[I],
139                                     machinesDifferentes(Elem,Elem2)
140                             )
141                     ).
142
143 machinesDifferentes(tache(_,_,M1,_),tache(_,_,M2,_)):- \=(M1,M2),!.
144 machinesDifferentes(tache(Duree,_,M,Debut),tache(Duree2,_,M,Debut2)) :- ((
        Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).
145
146 solve2(Taches,Fin) :-   taches(Taches),
147                                                 domaines(Taches,Fin),
148                                                 precedences(Taches),
149                                                 conflits(Taches),
150                                                 getVarList(Taches,Fin,Liste),
151                                                 labeling(Liste),
152                                                 affiche(Taches).
153 /*
154 [eclipse 78]: taches(T), solve2(T, Fin).
155 tache(3, [], m1, 0)
156 tache(8, [], m1, 29)
157 tache(8, [4, 5], m1, 9)
158 tache(6, [], m2, 0)
159 tache(3, [1], m2, 6)
160 tache(4, [1, 7], m1, 25)
161 tache(8, [3, 5], m1, 17)
162 tache(6, [4], m2, 12)
163 tache(6, [6, 7], m2, 31)
164 tache(6, [9, 12], m2, 37)
165 tache(3, [1], m2, 9)
166 tache(6, [7, 8], m2, 25)
167
168 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, 12), tache(6, [6, 7], m2, 31)
        , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
        25))
169 Fin = 43
170 Yes (0.00s cpu, solution 1, maybe more) ?
171 */
172
```

```
173 /* Question 3.8
174
175 Oui, la solution est la meilleure !
176 Prolog resoud les contraintes en incrementant le debut des taches, jusqu'a
        obtenir le respect des contraintes.
177
178 */
```