

Excellent travail

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 contrainte 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,
```

OK

```

        [6, 7], m2, _241), tache(6, [9, 12], m2, _252), tache(3, [1], m2, _263),
        tache(6, [7, 8], m2, _272)]
19 Yes (0.00s cpu)
20
21 */

```

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"

```

1 affiche(Taches) :-      dim(Taches,[Dim]),
2                          (for(Indice,1,Dim),param(Taches)
3                              do
4                                  Elem is Taches[Indice
5                                      ],
6                                      writeln(Elem)
7                                  ).
8
9 /*                                OK
10 [eclipse 4]: taches(T),affiche(T).
11 tache(3, [], m1, _235)
12 tache(8, [], m1, _242)
13 tache(8, [4, 5], m1, _249)
14 tache(6, [], m2, _260)
15 tache(3, [1], m2, _267)
16 tache(4, [1, 7], m1, _276)
17 tache(8, [3, 5], m1, _287)
18 tache(6, [4], m2, _298)
19 tache(6, [6, 7], m2, _307)
20 tache(6, [9, 12], m2, _318)
21 tache(3, [1], m2, _329)
22 tache(6, [7, 8], m2, _338)
23
24 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)]
25 Yes (0.00s cpu)
26
27 */

```

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"

```

1 domaines(Taches, Fin) :- dim(Taches,[Dim]),
2                          (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                                     Oui
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                                     Bien
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.

Très bien

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),
23           , tache(6, [], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
24           tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
25           tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
26
27 Fin = 8
28 Yes (0.00s cpu, solution 1, maybe more) ? ;
29 tache(3, [], m1, 1)
30 tache(8, [], m1, 0)
31 tache(8, [4, 5], m1, 0)
32 tache(6, [], m2, 0)
33 tache(3, [1], m2, 0)
34 tache(4, [1, 7], m1, 0)
35 tache(8, [3, 5], m1, 0)
36 tache(6, [4], m2, 0)
37 tache(6, [6, 7], m2, 0)
38 tache(6, [9, 12], m2, 0)
39 tache(3, [1], m2, 0)
40 tache(6, [7, 8], m2, 0)
41
```

```

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[
9                                  I],
10                                 Debut #>= Debut2+Duree2
11                                 ).

```

Bien

```

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 *conflicts(+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 – "conflicts"

```

1 conflicts(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
                              différents, 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),

```

TB

```

    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))
57 Fin = 43
58 Yes (0.01s cpu, solution 2, maybe more) ? ;
59 tache(3, [], m1, 2)
60 tache(8, [], m1, 29)
61 tache(8, [4, 5], m1, 9)
62 tache(6, [], m2, 0)
63 tache(3, [1], m2, 6)
64 tache(4, [1, 7], m1, 25)
65 tache(8, [3, 5], m1, 17)
66 tache(6, [4], m2, 12)
67 tache(6, [6, 7], m2, 31)
68 tache(6, [9, 12], m2, 37)
69 tache(3, [1], m2, 9)
70 tache(6, [7, 8], m2, 25)
71
72 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))
73 Fin = 43
74 Yes (0.01s cpu, solution 3, maybe more) ?
75 */
76
77 /*
78 [eclipse 11]: solve2(Taches,42).
79
80 No (0.00s cpu)
81 */

```

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 candidate pour la solution optimale au problème, à savoir l'ordonnancement des tâches au plus tôt. Seulement, rien ne nous assure que la première est la meilleure, car cela dépend de l'ordre de la déclaration des tâches au début, dans le prédicat *taches(?Taches)*.

Pour vérifier donc, 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. Donc nous avons eu dans notre cas la solution optimale.

Oui !

A priori, il n'y a aucune raison pour que la première solution soit la meilleure. Pour garantir que la première solution est la meilleure, il faut commencer l'énumération par la variable *Fin*. Ainsi, vous chercherez des solutions en instanciant *Fin* par ordre croissant -> La première solution sera donc la meilleure.

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,
22      _183), tache(6, [], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
23      m1, _210), tache(8, [3, 5], m1, _221), tache(6, [4], m2, _232), tache(6,
24      [6, 7], m2, _241), tache(6, [9, 12], m2, _252), tache(3, [1], m2, _263),
25      tache(6, [7, 8], m2, _272)]
26 Yes (0.00s cpu)
27
28 */
29
30 affiche(Taches) :-      dim(Taches,[Dim]),
31                        (for(Indice,1,Dim),param(Taches)
32                        do
33                        Elem is Taches[Indice
34                        ],
35                        writeln(Elem)
36                        ).
37
38 /*
39 [eclipse 4]: taches(T), affiche(T).
40 tache(3, [], m1, _235)
41 tache(8, [], m1, _242)
42 tache(8, [4, 5], m1, _249)
43 tache(6, [], m2, _260)
44 tache(3, [1], m2, _267)
45 tache(4, [1, 7], m1, _276)
46 tache(8, [3, 5], m1, _287)
47 tache(6, [4], m2, _298)
48 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 (0.00 s 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,
59                                       _Machine, Debut)
60                                       is Taches[Indice
61                                       ],
62                                       Debut + Duree #=< Fin
63                                       ,
64                                       Debut #>= 0
65                                       ).
66  /*
67  [eclipse 5]: taches(T), domaines(T, 10).
68
69  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})]
70
71  There are 12 delayed goals. Do you want to see them? (y\n)
72  Yes (0.00 s cpu)
73  */
74
75  getVarList(Taches, Fin, ListFin) :- dim(Taches, [Dim]),
76                                       (for(Indice, 1, Dim), fromto([], In, Out, List),
77                                           param(Taches)
78                                           do
79                                               Xi is Taches[Indice],
80                                               Xi = tache(_, _, _, Debut),
81                                               Out=[Debut|In]
82                                           ),
83                                       ListFin=[Fin|List].
84  /*

```

```

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
94                             domaines(Taches, Fin),
95                             precedences(Taches),
96                             getVarList(Taches, Fin, Liste),
97                             labeling(Liste),
98                             affiche(Taches).
99
100 precedences(Taches) :-    dim(Taches, [Dim]),
101                            (for(Indice, 1, Dim), param(Taches)
102                            do
103                                Elem is Taches[Indice],
104                                Elem = tache(_D, Noms, _M, Debut),
105                                (foreach(I, Noms), param(Debut, Taches)
106                                do
107                                    tache(Duree2, _N, _M, Debut2) is Taches[
108                                        I],
109                                    Debut #>= Debut2+Duree2
110                                )
111                            ).
112
113 /*
114 [eclipse 44]: taches(T), solve(T, Fin).
115 tache(3, [], m1, 0)
116 tache(8, [], m1, 0)
117 tache(8, [4, 5], m1, 6)
118 tache(6, [], m2, 0)
119 tache(3, [1], m2, 3)
120 tache(4, [1, 7], m1, 22)
121 tache(8, [3, 5], m1, 14)
122 tache(6, [4], m2, 6)
123 tache(6, [6, 7], m2, 26)
124 tache(6, [9, 12], m2, 32)
125 tache(3, [1], m2, 3)
126 tache(6, [7, 8], m2, 22)
127
128 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
136                            différents, sinon on va se retrouver a
137                            comparer deux fois la meme taches*/
138                         (for(I,I2,Dim),param(Taches,Elem)
139                         do
140                             Elem2 is Taches[I],
141                             machinesDifferentes(Elem,Elem2)
142                         )
143                     ).
144 machinesDifferentes(tache(_,_,M1,_),tache(_,_,M2,_)):- \=(M1,M2),!.
145 machinesDifferentes(tache(Duree,_,M,Debut),tache(Duree2,_,M,Debut2)) :- ((
146     Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).
147
148 solve2(Taches,Fin) :-      taches(Taches),
149                             domaines(Taches,Fin),
150                             precedences(Taches),
151                             conflits(Taches),
152                             getVarList(Taches,Fin,Liste),
153                             labeling(Liste),
154                             affiche(Taches).
155 /*
156 [eclipse 10]: solve2(Taches,Fin).
157 tache(3, [], m1, 0)
158 tache(8, [], m1, 29)
159 tache(8, [4, 5], m1, 9)
160 tache(6, [], m2, 0)
161 tache(3, [1], m2, 6)
162 tache(4, [1, 7], m1, 25)
163 tache(8, [3, 5], m1, 17)
164 tache(6, [4], m2, 12)
165 tache(6, [6, 7], m2, 31)
166 tache(6, [9, 12], m2, 37)
167 tache(3, [1], m2, 9)
168 tache(6, [7, 8], m2, 25)
169
170 Taches = [(tache(3, [], m1, 0), tache(8, [], m1, 29), tache(8, [4, 5], m1,
171     9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
172     tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31),
173     tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
174     25))
175 Fin = 43
176 Yes (0.01s cpu, solution 1, maybe more) ? ;
177 tache(3, [], m1, 1)
178 tache(8, [], m1, 29)

```

```

173  tache(8, [4, 5], m1, 9)
174  tache(6, [], m2, 0)
175  tache(3, [1], m2, 6)
176  tache(4, [1, 7], m1, 25)
177  tache(8, [3, 5], m1, 17)
178  tache(6, [4], m2, 12)
179  tache(6, [6, 7], m2, 31)
180  tache(6, [9, 12], m2, 37)
181  tache(3, [1], m2, 9)
182  tache(6, [7, 8], m2, 25)
183
184  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))
185  Fin = 43
186  Yes (0.01s cpu, solution 2, maybe more) ? ;
187  tache(3, [], m1, 2)
188  tache(8, [], m1, 29)
189  tache(8, [4, 5], m1, 9)
190  tache(6, [], m2, 0)
191  tache(3, [1], m2, 6)
192  tache(4, [1, 7], m1, 25)
193  tache(8, [3, 5], m1, 17)
194  tache(6, [4], m2, 12)
195  tache(6, [6, 7], m2, 31)
196  tache(6, [9, 12], m2, 37)
197  tache(3, [1], m2, 9)
198  tache(6, [7, 8], m2, 25)
199
200  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))
201  Fin = 43
202  Yes (0.01s cpu, solution 3, maybe more) ?
203  */
204
205  /*
206  [eclipse 11]: solve2(Taches,42).
207
208  No (0.00s cpu)
209  */

```