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

Nicolas Desfeux
Aurélien Texier

14 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.

Listing 5 – "solve1"

```
1 solve1(Taches,Fin) :-  taches(Taches),
2                                                domaines(Taches,Fin),
3                                                getVarList(Taches,Fin,Liste),
4                                                labeling(Liste),
5                                                affiche(Taches).
```

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.

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 /*
14 [eclipse 44]: taches(T), solve(T, Fin).
15 tache(3, [], m1, 0)
16 tache(8, [], m1, 0)
17 tache(8, [4, 5], m1, 6)
18 tache(6, [], m2, 0)
19 tache(3, [1], m2, 3)
20 tache(4, [1, 7], m1, 22)
21 tache(8, [3, 5], m1, 14)
22 tache(6, [4], m2, 6)
23 tache(6, [6, 7], m2, 26)
24 tache(6, [9, 12], m2, 32)
25 tache(3, [1], m2, 3)
26 tache(6, [7, 8], m2, 22)
27
28 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))
Fin = 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.

Listing 7 – "conflits"

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

machinesDifferentes(tache(_,_,M1,_),tache(_,_,M2,_)):- \=(M1,M2),!.
machinesDifferentes(tache(Duree,_,M,Debut),tache(Duree2,_,M,Debut2)) :- ((
    Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).


solve2(Taches,Fin) :-   taches(Taches),
                                            domaines(Taches,Fin),
                                            precedences(Taches),
                                            conflits(Taches),
                                            getVarList(Taches,Fin,Liste),
                                            labeling(Liste),
                                            affiche(Taches).
/*
[eclipse 78]: taches(T), solve2(T, Fin).
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)
```

```
37 tache(3, [1], m2, 9)
38 tache(6, [7, 8], m2, 25)
39
40 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))
41 Fin = 43
42 Yes (0.00s cpu, solution 1, maybe more) ?
43 */
```

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.

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  */
93
94
95  solve(Taches,Fin) :-    taches(Taches),
96                                          domaines(Taches,Fin),
97                                          precedences(Taches),
98                                          getVarList(Taches,Fin,Liste),
99                                          labeling(Liste),
100                                         affiche(Taches).
101
102 precedences(Taches) :-  dim(Taches,[Dim]),
103                         (for(Indice,1,Dim),param(Taches)
104                         do
105                                 Elem is Taches[Indice],
106                                 Elem = tache(_D,Noms,_M,Debut),
107                                 (foreach(I,Noms),param(Debut,Taches)
108                                 do
109                                         tache(Duree2,_N,_M,Debut2) is Taches[
                                                I],
110                                         Debut #>= Debut2+Duree2
111                                 )
112                         ).
113
114 /*
115 [eclipse 44]: taches(T), solve(T, Fin).
116 tache(3, [], m1, 0)
117 tache(8, [], m1, 0)
118 tache(8, [4, 5], m1, 6)
119 tache(6, [], m2, 0)
120 tache(3, [1], m2, 3)
121 tache(4, [1, 7], m1, 22)
122 tache(8, [3, 5], m1, 14)
123 tache(6, [4], m2, 6)
124 tache(6, [6, 7], m2, 26)
125 tache(6, [9, 12], m2, 32)
126 tache(3, [1], m2, 3)
127 tache(6, [7, 8], m2, 22)
128
129 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))
Fin = 38

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

machinesDifferentes(tache(_,_,M1,_),tache(_,_,M2,_)):- \=(M1,M2),!.
machinesDifferentes(tache(Duree,_,M,Debut),tache(Duree2,_,M,Debut2)) :- ((
    Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).


solve2(Taches,Fin) :-    taches(Taches),
                                                    domaines(Taches,Fin),
                                                    precedences(Taches),
                                                    conflits(Taches),
                                                    getVarList(Taches,Fin,Liste),
                                                    labeling(Liste),
                                                    affiche(Taches).
/*
[eclipse 78]: taches(T), solve2(T, Fin).
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)

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))
```

```
173 Fin = 43
174 Yes (0.00s cpu, solution 1, maybe more) ?
175 */
176
177
178 /* Question 3.8
179
180 Oui, la solution est la meilleure !
181 Prolog resoud les contraintes en incrementant le debut des taches, jusqu'a
        obtenir le respect des contraintes.
182
183 */
```