Programmation logique par contraintes

Partie II

Plan du cours

- Programmation logique et Prolog (PL)
  - SWI-Prolog, Sicstus
- **Programmation logique par contraintes (PLC)**
  - **Sicstus**
- Problèmes de satisfaction de contraintes (CSP/PC)
  - Choco

Qu'est-ce qu'une contrainte?

- Une *contrainte* est une formule logique, construite sur un langage fixé d'avance.
- Une contrainte dénote un ensemble de solutions (les solutions de la formule) pour une interprétation logique fixée d'avance.
- Exemple : La contrainte X + Y = 1 dénote les deux solutions {X = 0, Y = 1} et {X = 1, Y = 0} si le domaine d'interprétation est N.
- C'est une généralisation des problèmes d'unification :
  - Problème d'unification ⇒ contrainte (formule)
  - Unificateur ⇒ Solution
- Utilité: Utiliser les techniques de la programmation logiques pour des domaines autres que les termes symboliques.

Pourquoi faire?

- Problèmes combinatoires (jeux, . . .)
- Problèmes de planification (par exemple un emploi de temps)
- Problèmes d'ordonnancement (par exemple trouver une affectation de tâches à exécuter sur des machines)
- Problèmes de placement (par exemple placer des objets dans un espace limité)
- Tous ces problèmes avec éventuellement une fonction objective f à optimiser (utiliser un espace minimal, un temps minimal, un nombre minimal de machines, . . .)

Comment ça marche ?

- Le programmeur utilise des contraintes (formules logiques) pour modéliser son problème.

- L'interprète Prolog peut faire appel à des solveurs de contraintes pour savoir si une contrainte a une solution ou pas.

- Il y a des solveur de contraintes pour des domaines différents (« Systèmes de contraintes ») : arithmétique, domaine finis, . . .

- Difficulté : en général ces solveurs ne sont pas complets !

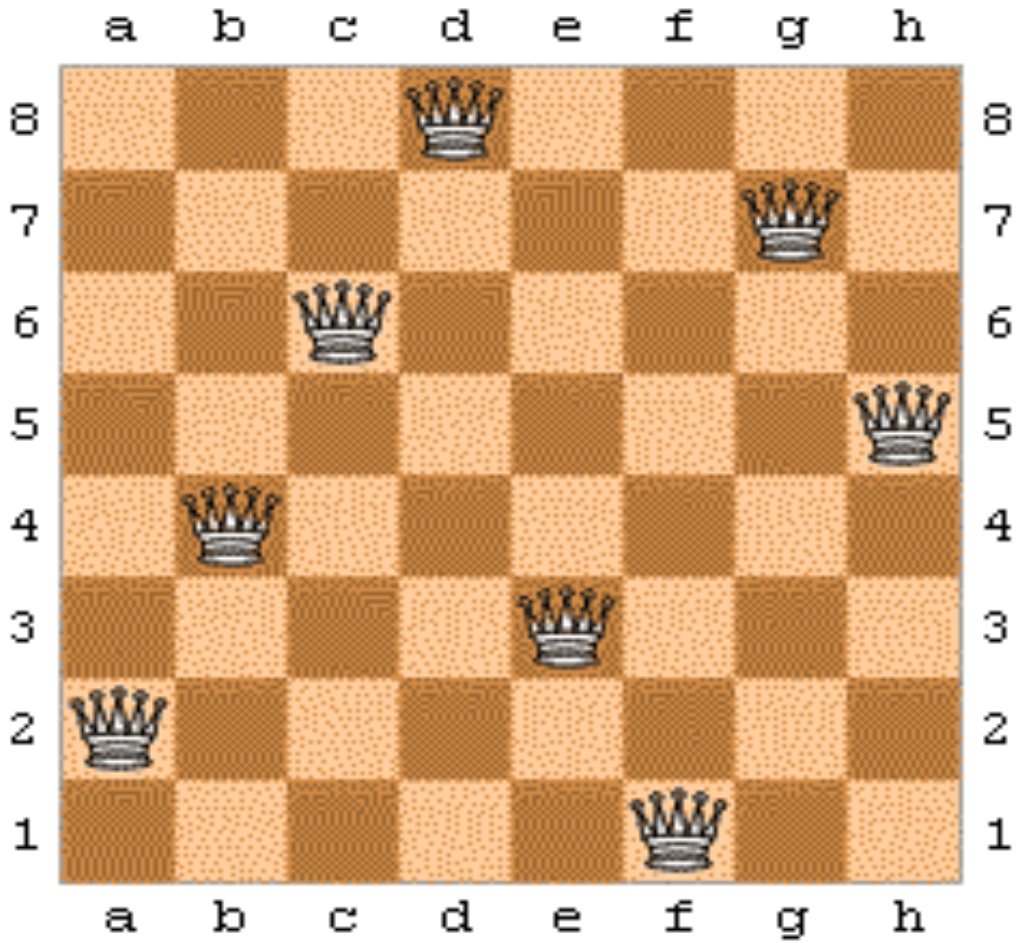
Le rôle du programmeur?

- Choisir le bon système de contraintes
- Choisir la bonne modélisation du problème par contraintes
- Programmer l'entrée/sortie
- Programmer la génération de contraintes
- Comprendre les conséquences de l'incomplétude du solveurs de contraintes, programmer une stratégie de recherche.

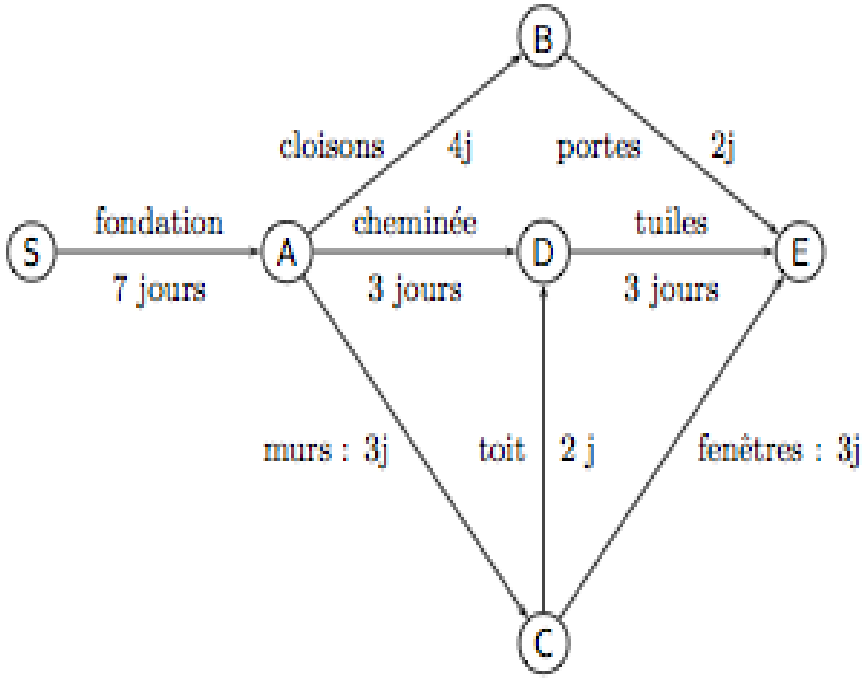
Exemple : Sudoko

	4				2		1	9
			3	5	1		8	6
3	1			9	4	7		
	9	4						7
2						8	9	
		9	5	2			4	1
4	2		1	6	9			
1	6		8				7	

Exemple : n reines



Exemple : peut-on construire la maison en 14 jours



Exemple : Ordonnancement

- Exécuter des tâches sur plusieurs machines.
-
- Un ensemble de tâches est donné
  - avec des précédences (des tâches doivent être terminées avant d'autres)
  - et des ressources partagées (des tâches ont besoin de la même machine)

- Déterminer pour toute tâche la machine et le temps de démarrage,
  - en satisfaisant les contraintes
  - en minimisant le temps global

Plan de cette partie

- Généralités, Contraintes Arithmétiques sur R
- Contraintes : Syntaxe et sémantique
- Exemple : contraintes linéaires sur R
- Contraintes non-linéaires, solveurs incomplets.

Contraintes : Syntaxe

- Donnée un langage de la logique du premier ordre :
- F: symboles de fonctions (et constantes);
- P: symboles de prédicat.
- Contrainte simple : Prédicat appliqué à des termes.
- Contrainte : conjonction de contraintes simples
  - $C = c1 \wedge c2 \wedge \ldots \wedge ck$
  - Exemple $X \geq 42 \wedge X = Y + 2$
- Contraintes spéciales :
  - true : conjonction vide, toujours vraie
  - false : toujours fausse
- Une contrainte est une formule de la logique du premier ordre. (Normalement sans négation, disjonction, quantificateurs)

Système de Contraintes

- Domaine de contraintes : D. Par exemple : l'ensemble des entiers, l'ensemble des nombres réels, . . .
- Un système de contraintes est donné par F,P,D et une interprétation des symboles en F et P.
- Par exemple: Système des contraintes numériques linéaires : F = {+,−,0,1,...},P = {=,≤,<,≥, >, ≠},D = N
- Interprétations : comme d'habitude.
- Autre systèmes de contraintes : nombres réels, contraintes de Herbrand, contraintes de domaine fini, contraintes d'ordre, . . .

Contraintes : Sémantiques

- Affectation : fonction partielle des variables vers le domaine de contraintes.
- Une affectation θ viole une contrainte simple, si elle la rend fausse, et viole une contrainte si elle viole au moins une de ses contraintes simples.
- Une affectation θ est consistante pour une contrainte si elle ne la viole pas.
- Solution : une affectation totale et consistante
  - $X \geq 42 \wedge X = Y + 2$ a une solution
  - $\theta = \{X \leftarrow 43, Y \leftarrow 41\}$
- C'est exactement la sémantique de la logique du premier ordre.

Contraintes : Satisfiabilité, équivalence

- Une contrainte est satisfaisable, si elle a une solution.
- L'ordre des contraintes simples peut être important, certains algorithmes dépendent de l'ordre.
- Pour $C=c1 \wedge c2 \wedge \ldots \wedge ck$ on définit
  - ensemble(C) = {c1,c2,...,ck}.
- Une contrainte c1 implique une contrainte c2 si toute solution de c1 est aussi solution de c2.
  - Anglais : c1 entails c2.
- Deux contraintes sont équivalentes si elles ont le même ensemble de solutions (équivalence logique).

Problèmes de satisfaction de contraintes

Constraint Satisfaction Problems – CSP

Données :

- Les variables du probl`eme avec leur domaines
- Une contrainte C

Questions :

- C est satisfaisable ?
- Donnez une solution, si C en a une.

- Un solveur de contraintes répond à la première question. Mais souvent aussi à la deuxième.

Satisfaction de contraintes

- Comment résoudre le problème de satisfaction de contrainte ?
- Approche naïve :
  - essayer toutes les affectations ne marchera pas pour les réelles, entiers, etc.
  - pour les domaines finis, on va essayer d'être plus intelligent.

On risque de rencontrer des limites :

- Non-décidabilité
- Complexité (problèmes NP-complets, ou pire)

Exemple : équations linéaires sur R

- Langage :
  - Constantes : R (on peut écrire toutes les constantes)
  - Fonctions : + (binaire), – (unaire et binaire), $*$ (binaire)
  - Prédicats : = (binaire)
- Pour l'instant, restriction à des termes arithmétiques linéaires : pas de produits entre variables.
- Domaine : R
- Interprétation : comme d'habitude.

Exemple : équations linéaires sur R

Exemples de contraintes arithmétiques linéaires :

- $X = Y+Z \wedge Y = 1+Z$
- $2*Y = 17*(X+42)-3*X$

Ne sont pas de contraintes arithmétiques linéaires :

- $X = 5*Y*Z$
- $Y = X * (42 + Z)$
- $2*X+Y*Y = 3*Z+Y*Y$

Exemple : Résolution de contraintes arithmétiques linéaires

- Forme résolue : $x1 = t_1 \wedge \ldots \wedge x_n = t_n$ où
  - $x_i \neq x_j$ si $i \neq j$
  - $x_i \notin V(t_j)$ pour tous i, j.
- Toute forme résolue est satisfaisable en R.
- $x_1, \ldots, x_n$: variables déterminées
- On a même le droit de choisir les valeurs des variables non déterminées.
- En général : définition des formes résolues fait partie du solveurs de contraintes.

Formes résolues

- Exemple d'une forme résolue :

  $x_1 = 2*y+5*z$

  $x_2 = 3-y-z$

  $x_3 = 42*y-17*z$

- Une solution est:

  $y \rightarrow 1,\ z \rightarrow 1,\ x_1 \rightarrow 7,\ x_2 \rightarrow 1,\ x_3 \rightarrow 25$

- On peut même, pour n'importe quel choix de valeurs pour y et z, trouver des valeurs de x_1, x_2, x_3 satisfaisant les équations.

Formes résolues

- N'est pas une forme résolue :

  $x1 = 2 * y + 5 * z$

  $17 = 42$

- N'est pas une forme résolue :

  $x1 = 2 * x2 + 5 * z$

  $x2 = 3 - y - x3$

  $x3 = 42 * y - 17 * x1$

Résoudre des contraintes arithmétiques linéaires

- L'algorithme est donné par des règles de transformation.
- On applique les règles tant que possible, dans n'importe quel ordre.
- Si on ne peut plus appliquer une règle on s'arrête, et on renvoie la contrainte obtenue.
- Equation normalisée :Soit une équation entre deux constantes, soit une équation de la forme $x = t$ où $x \notin V(t)$.
- Exemple d'une équation normalisée : $x = 17 + 3 * y + 5 * z$.
- On peut transformer toute équation linéaire en une équation normalisée qui lui est équivalente.

Résoudre des contraintes arithmétiques linéaires

- Règle 1 : Choisir une équation non normalisée, et la normaliser.

- Règle2: S'il y a une équation c1 =c2, où c1 et c2 sont des constantes différentes, alors remplacer toute la contrainte par $\perp$.

- Règle 3: S'il y une équation c = c, où c constante, la supprimer.

- Règle 4: S'il y a une équation normalisée x = t (avec $x \notin V(t)$) et x apparaît dans d'autres équations alors remplacer dans toutes les autres équations x par t.

exemple

$x+1 = y+2$

$y+3 = z+4-2x$

$z+2 = 2x+u$

- On résout la première équation pour x, et remplace x par y + 1 :

  $x=y+1$

  $3y+3 = z+2$

  $z+2 = 2y+2+u$

- On résout la deuxième équation pour z, on remplace z par 3y + 1, on résout la dernière équation pour u, et on obtient une forme résolue :

  $x=y+1$

  $z = 3y+1$

  $u = -3y-1$

Correction du solveur

- Toute règle est une transformation d'équivalence (les deux contraintes sont équivalentes)

- L'application des règles termine toujours : trouver un ordre de terminaison.

- Si aucune règle n'est applicable alors on a soit $\perp$, soit une forme résolue.

Contraintes réelles en Sicstus

- Avec la bibliothèque clpr :

  F = {+, −, ∗, /, sin, cos, tan,...}

  P= {=,<,>,=<,>=, =/=,...}

  D=R (les nombres réels)

- Ne pas oublier : use_module(library(clpr))
- Ecrire des contraintes entre accolades { et }
- Ecrire une virgule pour la conjonction logique.

L’exemple en Sicstus Prolog

```
?- use_module(library(clpr)).

?- {X+1=Y+2, Y+3=Z+4-2*X, Z+2=2*X+U}.

     {Y= -1.0+X},
     {Z= -2.0+3.0*X},
     {U=X} ? ;
     no
```

Ne sont pas des contraintes numériques

en Prolog:

- **e1=:=e2** car il faut que e1 et e2 soient close.
- **e1=e2** car Prolog traite les deux expressions de façon syntaxique (unification).
- **X is e** car il faut que e soit close, et X une variable.

Les contraintes expriment des relations.

Intégration des contraintes dans la programmation logique

- Atome : Prédicat, ou contrainte
- Configuration : liste d'atomes, plus une contrainte résolue

  Notée : $B \mid c$
- Tant que la liste B n'est pas vide : configuration $a, B' \mid c$
  - si le premier atome est un prédicat : le remplacer par le corps d'une clause de sa définition (comme Prolog)
  - si le premier atome est une contrainte : appliquer le solveur de contraintes à la contrainte $a \wedge c$.
    - Si résultat $\bot$: échec.
    - Si résultat est une forme résolue c' : passer à la configuration $B' \mid c'$.
- Donne lieu à un arbre de recherche comme Prolog.

Exemple : contraindre la somme d'une liste

```
:- use_module(library(clpr)).
listsum([],X) :- {X=0}.
listsum([H|R],X) :-
        {X = H + XR},
        listsum(R,XR).
```

Exemple : contraindre la somme d'une liste

```
?- listsum([2,3,4],X).
    X = 9.0 ? ;
    no
?- listsum([2,X,4],9).
    X = 3.0 ? ;
    no
?- listsum([2,X,Y],9).
    {Y=7.0-X} ? ;
    no
?- listsum(L,9).
    L = [9.0] ? ;
    L = [_A,_B],
    {_B=9.0-_A} ? ;
    L = [_A,_B,_C],
    {_C=9.0-_A-_B} ? ;
    …
```

Exécution du programme listsum

```
listsum([],X) :- {X=0}.
listsum([H|R],X) :-
                {X = H + XR},
                listsum(R,XR).
```

listsum([2, Y], 5)	| T
{5 = 2+XR},listsum([Y],XR)	| T
listsum([Y],XR)	| XR=3
{XR=Y+XR′ },listsum([],XR′)	| XR=3
listsum([],XR′)	| Y=3 -XR’
{XR′ =0}	| Y=3 -XR’
	|Y=3

Contraintes non-linéaires

- Maintenant on permet des équations arithmétiques quelconques, pas nécessairement linéaires.
- Par exemple $X + (Y * Z) = 3 * Z * Z * Z + 2 * Y * Y$
- Il est toujours théoriquement possible d'écrire un solveur de contraintes (résultat de Tarski, 1951). Mais cet algorithme a une complexité catastrophique.
- C'est possible car il s'agit des nombres réels.

Solveurs incomplets

- En général, un solveur peut être incomplet.
- Un solveur peut donner trois réponses possibles :
  - « non » (ou $\perp$)
  - « oui » (ou une forme résolue)
  - « je ne sais pas » (ou une formule seulement partiellement résolue)
- Dans le cas d'un solveur sur R : les équations qui contiennent des produits entre variables ne peuvent pas être traitées (sauf si l'équation devient linaire à cause de l'instantiation de variables).

Intégration de solveurs incomplets en Prolog

- Quand une contrainte ne peut pas être traitée par le solveur elle reste en suspens, et Prolog continue sur l'atome suivant.

- Quand la contrainte résolue est modifiée, les contraintes en suspens sont examinées de nouveau.

- Implémentation plus efficace : maintenir une liste de contraintes en suspens par variable, réexaminer seulement les contraintes en suspens qui contiennent une variable pour laquelle la contrainte résolue a des nouvelles informations.

Exemple : somme des carrés

```
listsqsum([],X) :- {X=0}.
listsqsum([H|R],X) :-
        {X = H*H + XR},
        listsqsum(R,XR).

?- listsqsum([2,3,4],X).
X = 29.0 ?;
no
?- listsqsum([2,X,4],29).
{9.0-X^2.0=0.0};
no
```

Intérêts Composés

Le programme CLP(R) suivant exprime la formule des intérêts composés avec le prédicat **int(P,T,I,B,M)** où :

P : capital
T : durée en mois
I : intérêt mensuel
B : balance
M : le montant mensuel

```
:- use_module(library(clpr)).

int(P,T,I,B,M):- {T > 0, T =< 1, B + M = P * (1 + I)}.
int(P,T,I,B,M):-
        {T > 1, P1 = P * (1 + I) - M, T1 = T - 1},
        int(P1, T1, I, B, M).
```

Intérêts Composés

?- int(120000, 120, 0.01, 0, M).
M = 1721.6513808310488 ? ;
No

?- int(P, 120, 0.01, 0, 1721.6513808310488).
P = 119999.99999999994 ? ;
no

?- int(P,120,0.01,0,M).
{M=0.014347094840258747*P} ? ;
No

Calcul de la température d'une surface discrétisée

On veut modéliser la température d'une feuille de métal. Pour cela, on découpe la feuille en une matrice de dimension m x n de points. Si la feuille est dans un état stable, chaque point de la matrice a la même température que la moyenne de ses quatre voisins.
Etant données les températures des points limites, les valeurs des autres points sont déterminées.

```
:-use_module(library(clpr)).

laplace([H1,H2,H3|T]):-
    laplace_vec(H1,H2,H3),
    laplace([H2,H3|T]).
laplace([_,_]).

laplace_vec([TL,T,TR|T1],[ML,M,MR|T2],[BL,B,BR|T3]):-
    {B + T + ML + MR - 4 * M = 0},
    laplace_vec([T,TR|T1],[M,MR|T2],[B,BR|T3]).
laplace_vec([_,_],[_,_],[_,_]).
```

Calcul de la température d'une surface discrétisée

```
test(X):-  X =  [
    [0,0,0,0,0,0,0,0,0,0,0],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,_,_,_,_,_,_,_,_,_,100],
    [100,100,100,100,100,100,100,100,100,100,100]
    ],
   laplace(X).

test2(L):-L=[
    [B11, B12, B13, B14],
    [B21, M22, M23, B24],
    [B31, M32, M33, B34],
    [B41, B42, B43, B44]
    ],
    laplace(L).
```

Calcul de la température d'une surface discrétisée

?- test(X).

```
X=[[0,0,0,0,0,0,0,0,0,0,0],
[100,51.11,32.52,24.56,21.11,20.12,21.11,24.56,32.52,51.11,100],
[100,71.91,54.41,44.63,39.74,38.26,39.74,44.63,54.41,71.91,100],
[100,82.12,68.59,59.80,54.97,53.44,54.97,59.80,68.59,82.12,100],
[100,87.97,78.03,71.00,66.90,65.56,66.90,71.00,78.03,87.97,100],
[100,91.71,84.58,79.28,76.07,75.00,76.07,79.28,84.58,91.71,100],
[100,94.30,89.29,85.47,83.10,82.30,83.10,85.47,89.29,94.30,100],
[100,96.20,92.82,90.20,88.56,88.00,88.56,90.20,92.82,96.20,100],
[100,97.67,95.59,93.96,92.93,92.58,92.93,93.96,95.59,97.67,100],
[100,98.89,97.90,97.12,96.63,96.46,96.63,97.12,97.90,98.89,100],
[100,100,100,100,100,100,100,100,100,100,100]] ?
```

? - test2(X).

```
B12 =  -B21 - 4*B31 + 16*M32 - 8*M33 + B34 - 4*B42 + B43,
B13 =  -B24 + B31 - 8*M32 + 16*M33 - 4*B34 + B42 - 4*B43,
M22 =  -B31 + 4*M32 - M33 - B42,
M23 =  -M32 + 4*M33 - B34 - B43 ?
```