

Programmation (Logique) par **Contraintes**

Peter Habermehl



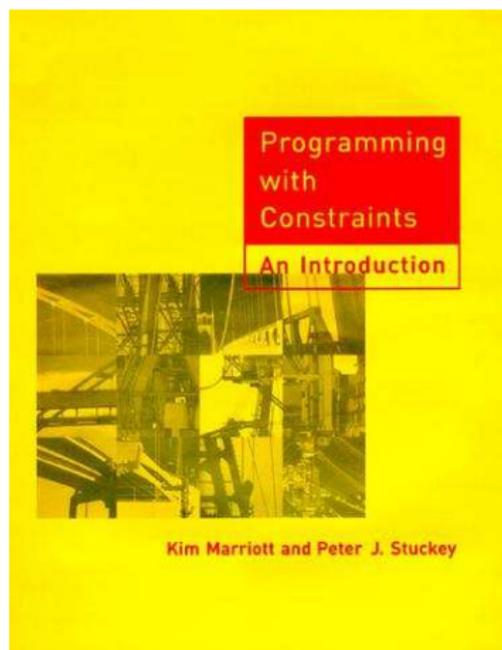
Université Paris Cité

UFR Informatique

Laboratoire IRIF

`Peter.Habermehl@irif.fr`

Livre principalement utilisé de cette partie



The MIT Press, 1998,
people.eng.unimelb.edu.au/pstuckey/book/book.html

Qu'est-ce que c'est les contraintes ?

- 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 $\{X \leftarrow 0, Y \leftarrow 1\}$ et $\{X \leftarrow 1, Y \leftarrow 0\}$ si le domaine d'interprétation est \mathbb{N} .
- C'est une généralisation des problèmes d'unification :
 - ▶ Problème d'unification \Rightarrow contrainte (formule)
 - ▶ Unificateur \Rightarrow Solution
- Utilité (entre autres) : Utiliser les techniques de la programmation logique pour des domaines autres que les termes symboliques.

Pour quelle problèmes ça sert ?

- Problèmes combinatoires (jeux, ...)
- Problèmes de planification (par exemple un emploi de temps)
- Problèmes d'ordonnancement (par exemple trouver une affectation de tâches à exécuter à des machines)
- Problèmes de placement (par exemple placer des objets dans un volume limité)
- Tous ces problèmes avec en plus l'optimisation (utiliser un espace minimal, un temps minimal, un nombre minimal de machines, ...)

Comment est-ce que 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 solveurs de contraintes pour des domaines différents : (“systèmes de contraintes”) : arithmétique, domaine fini, ...
- 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 (en Prolog)
- Comprendre les conséquences de l'incomplétude du solveur de contraintes, programmer une stratégie de recherche.

Tout ça sera objet de cette partie du cours . . .

Exemple : Sudoku

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

Exemple : le problème des n Reines

| | | | | | | | |
|---|---|---|---|---|---|--|---|
| | | |  | | | | |
| | | | | | |  | |
| | |  | | | | | |
| | | | | | | |  |
| |  | | | | | | |
| | | | |  | | | |
|  | | | | | | | |
| | | | | |  | | |

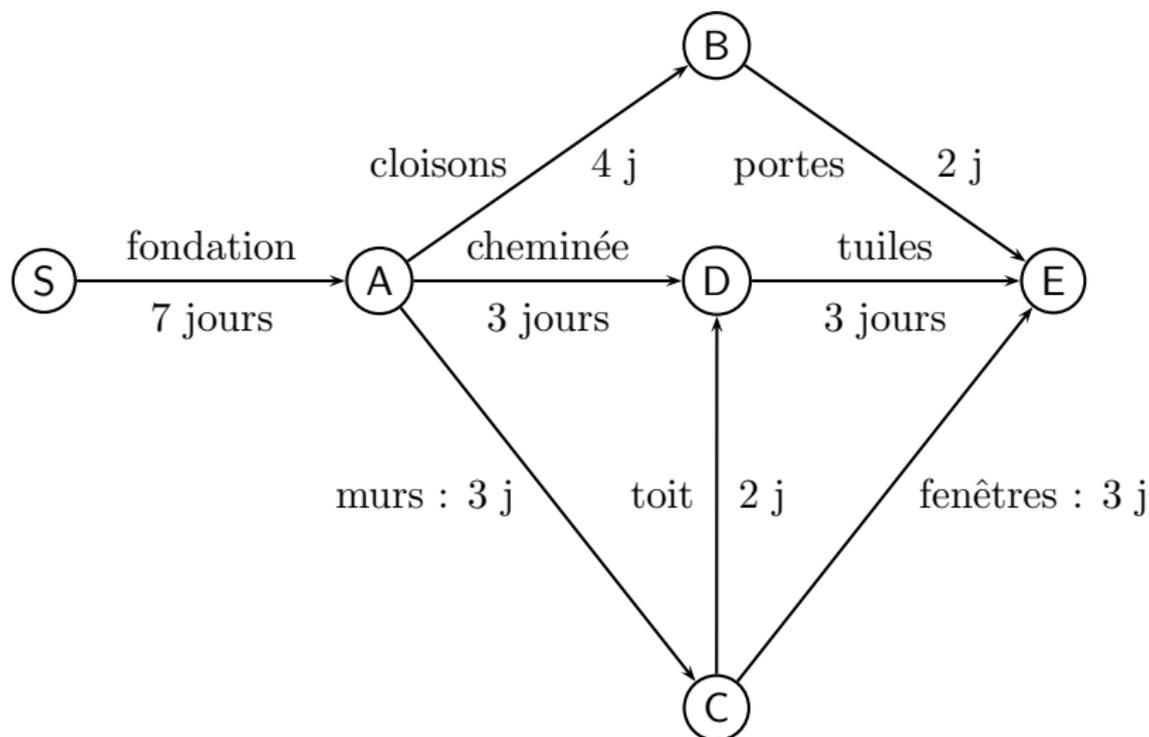
Exemple : coupures de morceaux

- Soient données largeur et hauteur d'une planche (de bois).
- Soient données les dimensions des morceaux qu'on souhaite obtenir.
- Peut-on obtenir les morceaux à partir de la planche donnée ?
(c.-à-d., est ce qu'on peut placer tous les morceaux sur la surface de départ, sans qu'ils se recoupent ?)
- Est-ce aussi possible si on ne peut couper un morceaux de planche que sur toute sa largeur ou toute son hauteur ?

Exemple : Livraison de colis

- Soient données les dimensions d'un certain nombre de colis
- Soit données les dimensions d'une ou plusieurs camionnettes
- Est-ce qu'il est possible de ranger tous les colis dans les camionnettes ?

Exemple : Peut-on construire la maison en 14 jours ?



Exemple : Ordonnancement de tâches

- Exécuter des tâches sur plusieurs machines.
- Un ensemble de tâches est donné
 - ▶ avec des **contraintes** de préséances (certaines tâches doivent être terminées avant des autres)
 - ▶ et des **contraintes** de ressources partagées (certaines tâches ont besoin de la même machine)
- Déterminer pour toute tâche la machine et le temps de démarrage tels que
 - ▶ les **contraintes** sont satisfaites
 - ▶ le temps global est minimisé
- Il y a plein de variantes

Plan du cours (préliminaire)

- Introduction
- Généralités, contraintes arithmétiques sur \mathbb{R} , algorithme de Fourier-Motzkin
- Contraintes sur un domaine fini
- Notions de consistance
- Optimisation, algorithme Simplex
- Modélisation

Plan de cette partie

Généralités, Contraintes Arithmétiques sur \mathbb{R}

- Contraintes : Syntaxe et sémantique
- Exemple : contraintes linéaires sur \mathbb{R}
- Programmation logique avec contraintes
- Contraintes non-linéaires, solveurs incomplets.

Contraintes : Syntaxe

- Soit donné 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 (obtenus à partir de F et d'un ensemble de variables).
- *Contrainte* : conjonction de contraintes simples
 $C = c_1 \wedge c_2 \wedge \dots \wedge c_k$
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 \mathbb{N} , l'ensemble des nombres rationnels \mathbb{Q} (réels \mathbb{R}), ...
- 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 :

$$F = \{+, -, 0, 1, \dots\}, \quad P = \{=, \leq, <, \geq, >, \neq\}, \quad D = \mathbb{N} \text{ ou } \mathbb{Q} \text{ ou } \mathbb{R}$$

Interprétations : comme d'habitude.

- Autre systèmes de contraintes : contraintes de Herbrand (à la Prolog, sur les termes), contraintes de domaine fini, contraintes d'ordre, ...

Contraintes : Sémantique

- *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
 - ▶ p.e. $X \geq 42 \wedge X = Y + 2$ a une solution $\theta = \{X \leftarrow 43, Y \leftarrow 41\}$
- C'est exactement la sémantique de la logique du premier ordre.

Contraintes : Satisfaisabilité, 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 $C = c_1 \wedge c_2 \wedge \dots \wedge c_k$ on définit $ensemble(C) = \{c_1, c_2, \dots, c_k\}$.
- Une contrainte c_1 *implique* une contrainte c_2 si toute solution de c_1 est aussi solution de c_2 .
Anglais : c_1 *entails* c_2 .
- Deux contraintes sont *équivalentes* si elles ont le même ensemble de solutions (équivalence logique).

Problèmes de satisfaction de contraintes

Anglais *Constraint Satisfaction Problems – CSP*

Données :

- Les variables du problème avec leur domaines
- Une contrainte C

Questions :

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

Un **solutionneur 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 domaines infinis (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 \mathbb{R}

- Langage :
 - ▶ Constantes : \mathbb{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 : \mathbb{R}
- Interprétation : comme d'habitude.

Exemple : Équations linéaires sur \mathbb{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 : $x_1 = t_1 \wedge \dots \wedge x_n = t_n$ où
 - ▶ $x_i \neq x_j$ si $i \neq j$
 - ▶ $x_i \notin \mathcal{V}(t_j)$ pour tous i, j ($\mathcal{V}(t_j)$ est l'ensemble des variables de t_j)
- Toute forme résolue est satisfaisable en \mathbb{R} .
- x_1, \dots, x_n : variables *déterminées*
- On a même le droit de choisir les valeurs des variables non déterminées.
- En général : la définition des **formes résolues** fait partie du solutionneur 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:

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

On peut même, pour n'importe quel choix de valeurs pour y et z , trouver des valeurs de x_1, x_2, x_3 telles que les équations sont satisfaites.

Formes résolues

N'est *pas* une forme résolue :

$$x_1 = 2 * y + 5 * z$$

$$17 = 42$$

N'est *pas* une forme résolue :

$$x_1 = 2 * x_2 + 5 * z$$

$$x_2 = 3 - y - x_3$$

$$x_3 = 42 * y - 17 * x_1$$

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.
- *Équation normalisée* : Soit une équation entre deux constantes, soit une équation de la forme $x = t$ où $x \notin \mathcal{V}(t)$.
- 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.
- Règle 2 : S'il y a une équation $c_1 = c_2$, où c_1 et c_2 sont des constantes différentes, alors remplacer toute la contrainte par \perp .
- Règle 3: S'il y a une équation $c = c$, où c constante, la supprimer.
- Règle 4: S'il y a une équation normalisée $x = t$ (avec $x \notin \mathcal{V}(t)$) et x paraît dans des 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 , remplace z par $3y + 1$, résout la dernière équation pour u , et obtient une forme résolue :

$$x = y + 1$$

$$z = 3y + 1$$

$$u = -3y - 1$$

Correction du solutionneur

- Toute règle est une transformation d'équivalence (les deux contraintes sont équivalentes)
- L'application de règles termine toujours : trouver un ordre de terminaison.
- Si aucune règle est applicable alors on a soit \perp , soit une forme résolue.

Contraintes réelles en Eclipse CLP

Eclipse CLP permet d'utiliser des contraintes réelles avec la bibliothèque `clpr` (qu'il faut installer) :

$F = \{+, -, *, /, \sin, \cos, \tan, \dots\}$

$P = \{=, <, >, = <, > =, = / =, \dots\}$

$D =$ les nombres réelles

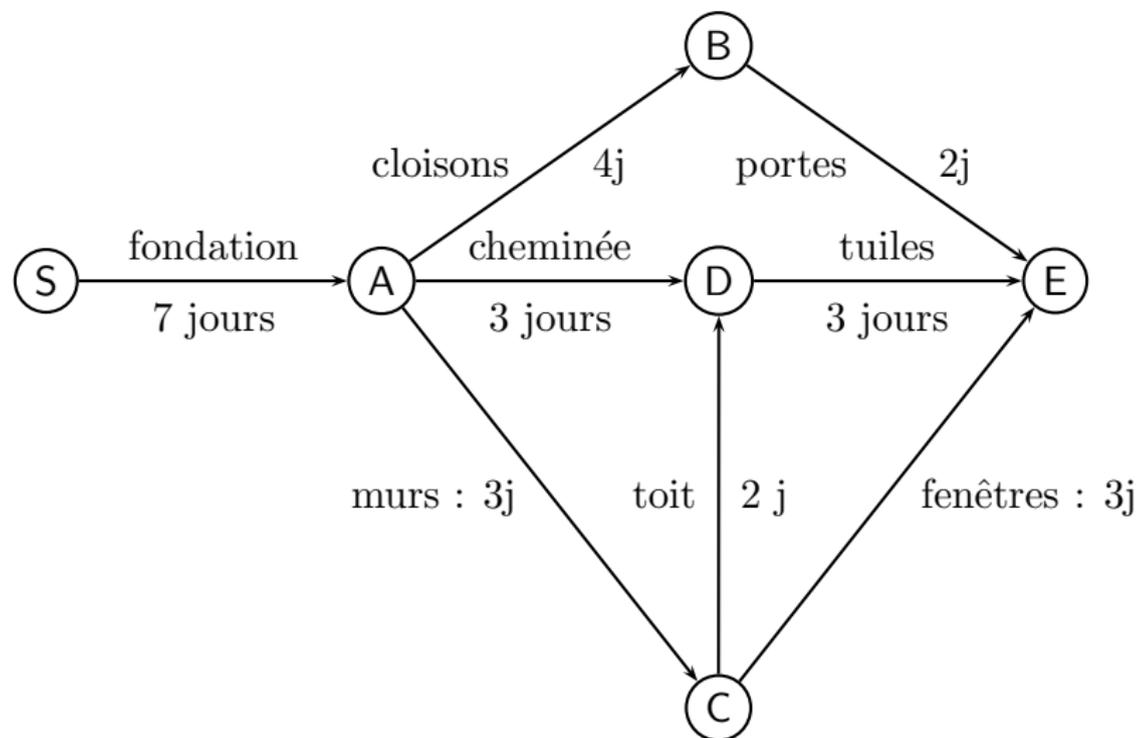
- **Ne pas oublier** : `use_module(library(clpr)).` ou `lib(clpr).`
- Écrire des contraintes entre accolades `{ et }`
- Écrire une virgule pour la conjonction logique.
- En réalité les réelles ne sont pas des réelles, mais des **floats** !
- On peut également utiliser les rationnelles avec `lib(clpq).`

L'exemple en Eclipse CLP

```
      : use_module(library(clpr)).  
      ....  
Yes  
      : {X+1=Y+2,Y+3=Z+4-2*X,Z+2=2*X+U}.  
      ...  
% Linear constraints:  
{  
    Y = -1.0 + X,  
    Z = -2.0 + 3.0 * X  
}Yes
```

L'ordre des variables peut être différent.

Construction de la maison en ≤ 14 jours



Modélisation

- Extension du système de contraintes : $<$, \leq
- On introduit une variable T par tâche T qui dénote le début de cette tâche.
- La fin de la tâche T de durée d_T est $T + d_T$.
- Si tâche R dépend de la tâche S : $S + d_S \leq R$.
- Toutes les tâches ≥ 0 .
- Toutes les fins de tâches ≤ 14 .

La contrainte pour la construction de la maison :

$$\begin{array}{lll} \text{Fond} \geq 0 & & \wedge \text{Fonds} + 7 \leq 14 \\ \wedge \text{Cloisons} \geq 0 & \wedge \text{Cloisons} \geq \text{Fond} + 7 & \wedge \text{Cloisons} + 4 \leq 14 \\ \wedge \text{Cheminée} \geq 0 & \wedge \text{Cheminée} \geq \text{Fond} + 7 & \wedge \text{Cheminée} + 3 \leq 14 \\ \wedge \text{Murs} \geq 0 & \wedge \text{Murs} \geq \text{Fond} + 7 & \wedge \text{Murs} + 3 \leq 14 \\ \wedge \text{Toit} \geq 0 & \wedge \text{Toit} \geq \text{Murs} + 3 & \wedge \text{Toit} + 2 \leq 14 \\ \wedge \text{Portes} \geq 0 & \wedge \text{Portes} \geq \text{Cloisons} + 4 & \wedge \text{Portes} + 2 \leq 14 \\ \wedge \text{Fenêtres} \geq 0 & \wedge \text{Fenêtres} \geq \text{Murs} + 3 & \wedge \text{Fenêtres} + 3 \leq 14 \\ \wedge \text{Tuiles} \geq 0 & \wedge \text{Tuiles} \geq \text{Cheminée} + 3 & \wedge \text{Tuiles} + 3 \leq 14 \\ & \wedge \text{Tuiles} \geq \text{Toit} + 2 & \end{array}$$

Il n'y a pas de solution.

Ne sont pas de contraintes numériques

en Prolog:

- $e_1 ::= e_2$ car il faut que e_1, e_2 soient closes.
- $e_1 = e_2$ car Prolog traite les deux expressions de façon *syntaxique* (unification).
- $X \text{ is } e$ 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
- *But* : liste d'atomes, plus une contrainte résolue
Notée : $I \mid c$
- Tant que la liste I n'est pas vide : but $a, I \mid c$
 - ▶ si le premier atome est un prédicat : le remplacer par le corps d'une clause de sa définition (comme Prolog) après unification
 - ▶ si le premier atome est une contrainte : appliquer le solveur de contraintes à la contrainte $a \wedge c$.
 - ★ Si résultat \perp : échec.
 - ★ Si résultat est une forme résolue c' : passer au but $I \mid c'$.
- Donne lieu à un arbre de dérivation comme Prolog.

Exemple : la somme des éléments d'une liste

En Prolog pure (sans accumulateur):

```
listsumold([],0).  
listsumold([X|L],Somme) :- listsumold(L,S), Somme is S+X.
```

Avec contraintes:

```
:- 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  
: listsum([2,X,4],9).  
X = 3.0  
: listsum([2,X,Y],9).  
{X=7.0-Y}  
: listsum(L,9).  
L = [9.0]  
et  
L = [_253, _570]  
{_570 = 9.0 - _253}  
et  
L = [_253, _570, _928]  
{_928 = 9.0 - _570 - _253}  
etc.
```

Exécution du programme `listsum`

`listsum([],X) :- {X=0}.`

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

| | |
|---|--------------------------------------|
| <code>listsum([2, Y], 5)</code> | \top |
| <code>{5 = 2 + XR}, listsum([Y], XR)</code> | \top |
| <code>listsum([Y], XR)</code> | $XR = 3$ |
| <code>{XR = Y + XR'}, listsum([], XR')</code> | $XR = 3$ |
| <code>listsum([], XR')</code> | $Y = 3 - XR' \wedge XR = 3$ |
| <code>{XR' = 0}</code> | $Y = 3 - XR' \wedge XR = 3$ |
| | $Y = 3 \wedge XR = 3 \wedge XR' = 0$ |

La réponse est la projection de la contrainte résolue aux variables de la requête: $Y = 3$.

Contraintes non-linéaires

- 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*.
- Le problème est *non décidable* quand on change le domaine en \mathbb{N} (résultat de Yuri Matijasevic, 1970).

Solutionneurs incomplets

- En général, un solutionneur peut être *incomplet*.
- Un solutionneur 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 du solutionneur pour \mathbb{R} : les équations qui contiennent des produits entre variables ne peuvent pas être traitées (sauf si l'équation devient linéaire à cause de l'instantiation de variables avec des valeurs).

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 à 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  
    : listsqsum([2,X,4],29).  
X = X  
Delayed goals:  
    {9.0 - X ^ 2.0 = 0}
```

Exemple : Approximation de la racine carré

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

root(0,0).
root(Input,Result) :- newton(Input,Result,1).

newton(Input,Result,Current) :-
    { Current*Current = Input, Result=Current}.
newton(Input,Result,Current) :-
    { Next = Current/2 + Input/(2*Current) },
    newton(Input,Result,Next).
```

Malgré la multiplication de variables, ça fonctionne.

Intégration de solveurs incomplets en Prolog

- Parfois, des équations non-linéaires peuvent devenir linéaires par simplification.

Exemple : $X + Y * Y = 3 + Z + Y * Y$

- Souvent (comme en Eclipse): Nombres réels avec une précision limitée.

Conséquence : Erreurs d'arrondi, des termes peuvent

« disparaître » quand un coefficient devient 0. Eclipse considère l'égalité par rapport à un petit ϵ ($\{0.0000000001 = 0\}$. donne vrai).

Cela peut aussi rendre un terme linéaire.

- Il n'est pas toujours évident de savoir, si le solveur va arriver à résoudre une contrainte ou pas.

Méthode de Fourier-Motzkin

- Solutionneur pour les inéquations linéaires sur les rationnels/réels
- du à Joseph Fourier et Theodore Motzkin



Problème

- Donnée: une contrainte d'inéquations (non-strictes) sur les rationnels ou réels (conjonction d'inéquations)
- Question: Est-ce que la contrainte est satisfaisable ?
- Exemple: Construction de la maison
- Autre exemple:

$$3y - 2 \leq -2x$$

$$-y \geq -x - 1$$

$$x \leq 5$$

$$y \geq 1$$

Exemple

- Idée: Transformer la contrainte en une contrainte **équi-satisfaisable** avec une variable en moins
- On choisit x et réécrit les contraintes de sorte que x apparaisse seule et on regroupe les \leq et les \geq pour x :

$$x \leq -\frac{3}{2}y + 1$$

$$x \leq 5$$

$$x \geq y - 1$$

$$y \geq 1$$

- Donc on doit avoir $y - 1 \leq \min(-\frac{3}{2}y + 1, 5)$
- On élimine x en combinant tous les termes plus petits que x avec ceux plus grands.
- On garde les inéquations sans x .

Exemple

$$x \leq -\frac{3}{2}y + 1$$

$$x \leq 5$$

$$x \geq y - 1$$

$$y \geq 1$$

devient

$$y - 1 \leq -\frac{3}{2}y + 1$$

$$y - 1 \leq 5$$

$$y \geq 1$$

Exemple

$$y - 1 \leq -\frac{3}{2}y + 1$$

$$y - 1 \leq 5$$

$$y \geq 1$$

- On continue avec y :

$$y \leq \frac{4}{5}$$

$$y \leq 6$$

$$y \geq 1$$

Exemple

$$y \leq \frac{4}{5}$$

$$y \leq 6$$

$$y \geq 1$$

- On élimine y :

$$1 \leq \frac{4}{5}$$

$$1 \leq 6$$

- C'est faux. Donc, la contrainte d'origine n'est pas satisfaisable.

En général

- Deux contraintes c_1 et c_2 sont dites **équi-satisfaisables** si: si c_1 est satisfaisable, alors aussi c_2 et vice-versa.
ATTENTION: **équi-satisfaisables** n'est pas **équivalentes**
- Soit c une contrainte linéaire d'inéquations non-strictes et x une variable de c
- On réécrit c dans la forme $c_1 \wedge \dots \wedge c_k \wedge c'_1 \wedge \dots \wedge c'_{k'} \wedge c''_1 \wedge \dots \wedge c''_{k''}$ (k, k' et k'' peuvent être 0) de sorte que
 - ▶ les c_i soient de la forme $x \leq t$ (avec $x \notin \mathcal{V}(t)$)
 - ▶ les c'_i soient de la forme $x \geq t'$ (avec $x \notin \mathcal{V}(t')$)
 - ▶ les c''_i ne contiennent pas x
- On obtient une contrainte c' à partir de c en prenant toutes les inéquations c''_i et **toutes** les inéquations $t' \leq t$ ou t est un terme des c_i et t' un terme des c'_i .
 - ▶ S'il n'y a pas de c_i ou de c'_i , on n'obtient que les c''_i
- c et c' sont équi-satisfaisables et c' contient une variable de moins que c

En général

- Il est facile de décider si une contrainte d'inéquations sans variable est satisfaisable.
- Étant donné une contrainte d'inéquations c on la transforme en contrainte equi-satisfaisable avec une variable de moins.
- On itère ce processus jusqu'à ce qu'on obtienne une contrainte c_v sans variables
- La contrainte c d'origine est satisfaisable si et seulement si c_v est satisfaisable.

Remarques

- Est-ce que cette méthode est efficace ?
- Comment traiter des contraintes avec équations et inéquations ?
- Comment traiter des inéquations strictes ($<$) ?
- Comment récupérer une solution, si la contrainte est satisfaisable ?
- Comment faire sur les entiers ?
 - ▶ Est-ce que $3x \leq 2 \wedge 3x \geq 1$ est satisfaisable sur les rationnelles ? sur les entiers ?