

Outils et langages de Programmation par Contraintes

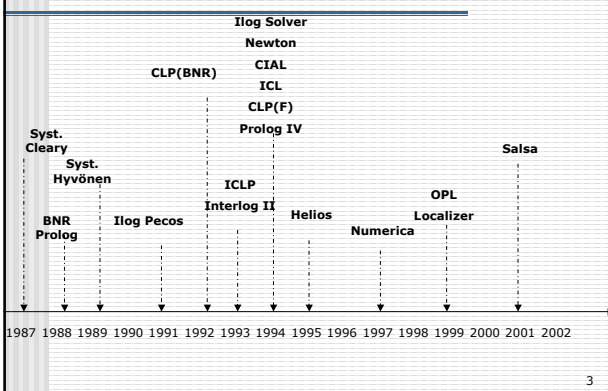
1

Programmation Par Contraintes

- L'approche Programmation Logique
 - Prolog IV
 - Chip
- L'approche Bibliothèque de PPC
 - Solver
- L'avenir de la PPC
 - l'exemple de la suite optimisation de ILOG (OPL studio)

2

Historique



3

Prolog IV

- Prolog IV est un Prolog (1972) avec contraintes, conservant de ses ancêtres
 - du Prolog originel, l'unification et le backtracking


```
>> f(g(1),h(X)) = f(Y,h(prolog_iv)).
X = prolog_iv,
Y = g(1).
```
 - de Prolog II, la capacité de gérer des diséquations, les possibilités de gestion du contrôle


```
>> freeze(X,dif(Y,1)), X = f(Y), X = f(1).
false.
```
 - de Prolog III, toutes les contraintes (Chats, Volatiles, Pattes, Têtes,...)


```
>> P = 4*C+2*V, T = C+V, P = 10, V = 1 .
P = 10, V = 1, C = 2, T = 3.
```

4

Prolog IV

- hérite de Prolog III des contraintes sur les arbres, les listes, les booléens, ainsi que de contraintes numériques linéaires (équations, diséquations, inégalités)
- Les techniques de résolution utilisées s'appuient sur des méthodes exactes (programmation linéaire sur les rationnels)
- Prolog IV intègre des problèmes sur les entiers (domaines finis) et sur les réels
 - solveur basé sur l'arithmétique des intervalles
 - permet de mélanger des contraintes sur les entiers, les réels et les booléens
- + de 100 contraintes prédéfinies

5

Prolog IV

- Exemple 1 : calculer la somme des éléments d'une liste


```
somme([],0).
somme([X|L], X.+S) :- somme(L,S).
>> somme([1,2,3,4,5], S).
S = 10.
>> somme([cc(1,2), cc(1,3), cc(2,4)], S).
S ~ cc(4,9).
>> X ~ cc(1,3), Y ~ cc(2,4), S ~ cc(2,4),
somme([X,Y],S).
S ~ cc(3,4), Y ~ cc(2,3), X ~ cc(1,2).
```

6

Prolog IV

■ Exemple 2 : indexation d'une liste

X est une valeur dans l'intervalle [1,5], L est une liste de valeurs

N est un index (une position entre 1 et 5)

X doit être dans L \Rightarrow que vaut X ? que vaut N ?

>> `cc(X,1,5)`, `L = [2,3,6,7,10]`, `X = L:N`.

`L = [2,3,6,7,10]`,

`N ~ cc(1,2)`,

`X ~ cc(2,3)`.

7

Prolog IV

■ Applications

- système expert de diagnostic de pannes (moteur Mercedes)
- aide au diagnostic sur les droits des jeunes issus de l'immigration (INJEP)
- aide à la décision pour l'acquisition des gestes en chirurgie cardiaque (INSERM)
- simulations de systèmes socio-économiques (Banque La Hénin)

■ Conclusion (<http://prologianet.univ-mrs.fr>)

- pionnier de l'intégration de contraintes en Programmation Logique
- bénéficie de travaux de recherche importants (constante évolution)
- mais un peu complexe à maîtriser

8

CHIP

■ CHIP V5 (depuis 1996)

■ architecture ouverte

- noyau écrit en C, versions C, C++ et Prolog
- moteur de contraintes indépendant du langage de programmation
- interface graphique, interface BD, ...

■ solveurs

- domaines finis, continus
- optimisation : recuit simulé, méthode tabou

■ contraintes utilisateurs

■ possibilités de contrôle évoluées

■ contraintes globales : all-diff, cycle, cumulatives

- techniques de relaxation

9

CHIP

■ domaines finis

- méthodes syntaxiques (réécriture des inégalités et diséquations) : forward-checking, lookahead
- contraintes numériques : $<$, $>$, $=$
- contraintes symboliques : \neq , `element(X,L,Y)` "Y est le Xième élément de L", `alldifferent`, `atmost`, `atleast...`
- méthodes sémantiques ou contraintes globales (pour les domaines finis) : `cumulative` (ordonnancement de tâches), `diffn` (placement), `cycle` (graphes), `among`, `precedence`, `sequence`

■ domaines continus

- élimination gaussienne
- simplexe

10

CHIP

■ Applications

- roulement du personnel de service TGV (SNCF)
- évacuation de déchets nucléaires (EDF)
- optimisation des mixtures de brut et de mélange (FINA)

■ Conclusion (<http://www.cosytec.com>)

- produit commercial très complet et efficace
- a bénéficié de programmes de recherche européens conséquents (depuis l'ECRC, 1984) avec une vocation industrielle importante

11

ILOG Solver

■ ILOG Solver (1997)

- suite de ILOG PECOS (1991), bibliothèque LISP
- bibliothèque extensible C++ (outil de PPC par objets)

■ variables

- sans restriction de type (objets)

■ domaines

- finis
- intervalles
- ensembles

12

ILOG Solver

- **Contraintes**
 - prédéfinies, globales
 - contraintes sur des objets, contraintes de classe
 - primitives de création de nouvelles contraintes personnalisées
- **Solveurs**
 - réduction des domaines finis pour les contraintes logiques et ensembliste
 - simplexe pour les contraintes linéaires
 - spécifiques aux contraintes de distribution et d'affectation
 - spécifiques aux contraintes d'ordonnement

13

ILOG Solver

- **Applications**
 - gestion des terminaux d'aéroport (British Airways)
 - commerce électronique (CAMIF)
 - optimisation de découpe de pellicules-photos (Agfa, Kodak)
- **Conclusion (<http://www.ilog.fr>)**
 - bibliothèque
 - → facilite l'intégration dans une application C++
 - intégration de contraintes sur des classes d'objets, des ensembles, des flottants, tous domaines finis

14

Exemple : SEND + MORE = MONEY

- Un même exemple traité par PROLOG IV, CHIP et ILOG Solver (d'après A. Fron)

```
  S  E  N  D
+  M  O  R  E
-----
= M  O  N  E  Y
```

- Il s'agit d'affecter aux 8 lettres S, E, N, D, M, O, R, Y des chiffres tous différents de façon à ce que l'addition SEND + MORE = MONEY soit correcte

15

Exemple : Prolog IV

```
solution(I, J, K) :- chiffresTousDifférents(<M, S, O, E, N, R, D, Y>), tousEntiers(<M, S, O, E, N, R, D, Y>)
{ S ≠ 0, M ≠ 0, I + J = K, I = 1000*S + 100*E + 10*N + D,
  J = 1000*M + 100*O + 10*R + E, K = 10000*M + 1000*O + 100*N + 10*E + Y}.

tousEntiers(<>).
tousEntiers(<E>.R) :- enum(E), tousEntiers(R).
chiffresTousDifférents(<>).
chiffresTousDifférents(<X>.S) :- horsDe(X, S), chiffresTousDifférents(S) {0 ≤ X, X ≤ 9}.
horsDe(X, <>).
horsDe(X, <Y>.S) :- horsDe(X, S) {X ≠ Y}.
```

16

Exemple : CHIP

```
tous_différents([]).
tous_différents([X | Y]) :- hors_de(X, Y), tous_différents(Y).
hors_de(X, []).
hors_de(X, [F | T]) :- X ≠ F, hors_de(X, T).
labeling([]).
labeling([Premier | Reste]) :- indomain(Premier), labeling(Reste).
domain sendmory(0..9).
sendmory([S, E, N, D, M, O, R, Y]) :- tous_différents([S, E, N, D, M, O, R, Y]), S ≠ 0, M ≠ 0,
1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E = 10000*M + 1000*O + 100*N + 10*E + Y,
labeling([S, E, N, D, M, O, R, Y]).
```

17

Exemple : ILOG Solver

```
int main(){
  IlcManager m(IlcEdit);
  IlcIntVar
    S(m, 0, 9), E(m, 0, 9), N(m, 0, 9), D(m, 0, 9),
    M(m, 0, 9), O(m, 0, 9), R(m, 0, 9), Y(m, 0,
9);
  m.add( S != 0 );      m.add( M != 0 );
  IlcIntVar send = 1000*S + 100*E + 10*N + D;
  IlcIntVar more = 1000*M + 100*O + 10*R + E;
  IlcIntVar money = 10000*M + 1000*O
    + 100*N + 10*E + Y;
  m.add( send + more == money );
  IlcIntArray letters(m, 8, S, E, N, D, M, O, R, Y);
  m.add(IlcAllDiff(letters));
  m.add(IlcGenerate(letters));
  m.nextSolution();
  m.end();
}
```

18

Conclusion

19

Conclusion

- La PPC est une technique d'utilisation d'un outil informatique souple et déclaratif de résolution de problèmes, souvent suffisante
- La PPC a déjà fait ses preuves dans des projets industriels de taille significative
- Par sa déclarativité, elle permet une plus grande souplesse et une plus grande facilité de mise en œuvre que les algorithmes traditionnels
- Elle propose des algorithmes et des techniques de résolution génériques dont le programmeur n'a pas à se soucier

20

Conclusion

- Ses limites : la taille du problème ?
⇒ techniques de RO plus aptes à traiter de "gros" problèmes ⇒ la plupart des outils actuels de PPC intègrent des techniques empruntées à la RO (Simplexe, Recuit Simulé, ...)
- Souvent utilisée pour juger de la faisabilité d'un problème avant de recourir à la RO (recherche de techniques de résolution plus spécifiques au problème)

21

Conclusion

- Dans certains cas, on peut indiquer un choix d'algorithmes plus pertinents que d'autres
 - savoir si le problème a une solution ⇒ résolution de contraintes
 - obtenir une solution satisfaisant même partiellement les contraintes ⇒ recuit simulé
 - proposer une solution même inexacte ⇒ mesurer la difficulté du problème, évaluer le gain d'une solution partielle
 - problèmes convexes ou non convexes ⇒ propagation de contraintes + énumération intelligente
 - problèmes discrets linéaires ⇒ simplexe (forme de la matrice), propagation et satisfaction de contraintes (générique, petite taille), recuit simulé (nombre important de variables, lever des contraintes), algorithmes gloutons (choix de valeur pertinent)

22

Conclusion

- La PPC est un outil perfectible, les améliorations des performances passent par :
 - algorithmes de consistance ou de résolution
 - intégration d'algorithmes de RO, de techniques mathématiques
 - processeurs, mémoires, parallélisation
 - primitives (contraintes) de haut-niveau pour un problème spécifique (affectation, ordonnancement)
 - dynamicité (flexibilité)
 - satisfaction partielle
 - langages de modélisation évolués

23

Conclusion

- Savoir-faire nécessaire dans la modélisation (facteur de 1 à 1000 en temps de résolution entre 2 modélisations du problème des 8 reines)
- car, si le nombre de combinaisons possibles est trop élevé, aucun ordinateur (actuel) ne saura le résoudre en un temps raisonnable
- Les Langages de Modélisation évolués (AMPL, Localizer...) peuvent aider

24

Bibliographie

- 2 ouvrages (entre autres ...)
 - Annick Fron, *Programmation par contraintes*, Editions Addison-Wesley France, 1994
 - Pour se familiariser avec la programmation par contraintes (termes, produits), nombreux exemples d'applications, des types de problèmes que peut résoudre la PPC
 - Edward Tsang, *Foundations of Constraint Satisfaction*, Collection Computation in Cognitive Science, Academic Press, 1993
 - Recueil des principaux résultats théoriques (algorithmes, techniques, heuristiques) de la théorie de la satisfaction de contraintes

25

Quelques URL

- Le site Web centralisé des contraintes :
 - <http://www.cs.unh.edu/ccp/archive/>
- Le newsgroup des contraintes :
comp.constraints
- A propos des méthodes de recherche locale de solutions :
 - <http://www.informatik.hu-berlin.de/~weinert/localsearch.html>
 - <http://solon.cma.univie.ac.at/~neum/glopt.html>

26