Résolution pratique de problèmes par ASP

Pascal Nicolas
pascal.nicolas@univ-angers.fr
LERIA – Université d'Angers

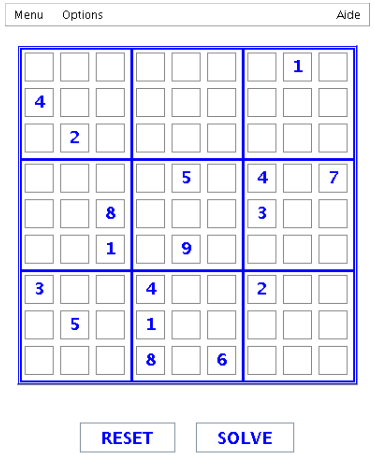
Journées Nationales de l'IA Fondamentale
Grenoble
1, 2 juillet 2007

ASP : quelques préliminaires

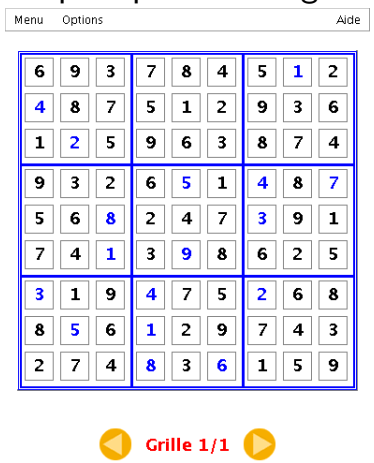
- 1988 : définition formelle par Michael Gelfond et Vladimir Lifschitz [GL88]
- 1996 : premiers solveurs efficaces
- 2000 : intégration au sein d'un système de planification pour la navette spatiale de la NASA [NBG+01]
- 2002. . .2005 : WASP, groupe de travail européen (FET)

Résoudre un sudoku

La grille initiale et le problème
sont codés à l'aide du programme ci-contre.



L'appel d'un solveur
`lparse fichierSudoku.nlp | smodels`
produit
Answer : 1
Stable Model : sol(3,1,1) sol(9,1,2) . . .sol(9,9,9)
ce qui représente la grille résolue




```
% taille du sudoku
const size=3.
% les numéros des carrés sont organisés comme suit
% 1 2 3
% 4 5 6
% 7 8 9
% coordonnées des cases et valeurs des nombres
row(1..size * size).      col(1..size * size).      n(1..size * size).
% la case I,J est dans le carré 1, ou 2, ... ou 9
in(K,I,J) :- row(I), col(J), K=(I-1)-((I-1) mod size) + ((J-1) - (J-1) mod siz
% on place ou on ne place pas K dans la case I, J
sol(I,J,K) :- row(I), col(J), n(K), not nsol(I,J,K).
nsol(I,J,K) :- row(I), col(J), n(K), not sol(I,J,K).
% au moins 1 nombre par case
okcase(I,J) :- row(I), col(J), n(K), sol(I,J,K).
:- row(I), col(J), not okcase(I,J).
% au plus 1 nombre par case
:- row(I), col(J), n(K1), n(K2), K1 < K2, sol(I,J,K1), sol(I,J,K2).
% au plus 1 fois chaque nombre sur chaque ligne
:- row(I), col(J1), col(J2), n(K),
sol(I,J1,K), sol(I,J2,K), J1<J2.
% au plus 1 fois chaque nombre sur chaque colonne
:- row(I1), row(I2), col(J), n(K),
sol(I1,J,K), sol(I2,J,K), I1<I2.
% jamais 2 fois le même nombre à l'intérieur d'un même carré
% sur des lignes différentes
:- in(C,I1,J1), in(C,I2,J2), I1<I2, n(K), sol(I1,J1,K), sol(I2,J2,K).
% sur des colonnes différentes
:- in(C,I1,J1), in(C,I2,J2), J1<J2, n(K), sol(I1,J1,K), sol(I2,J2,K).
%les chiffres déjà placés
sol(1,8,1).   sol(2,1,4).   sol(3,2,2).   sol(4,5,5).   sol(4,7,4).   sol(4,9,7).
sol(5,3,8).   sol(5,7,3).   sol(6,3,1).   sol(6,5,9).   sol(7,1,3).   sol(7,4,4).
sol(7,7,2).   sol(8,2,5).   sol(8,4,1).   sol(9,4,8).   sol(9,6,6).
```

ASP : what is it ?

Answer Set Programming

= programmation par ensemble (de) réponses

= programmation logique non monotone

= paradigme de programmation déclarative

- syntaxe proche de Prolog
- sémantique définie en terme d'ensembles de réponses correspondant à différentes vues possibles (des modèles) du « monde » décrit par le programme.
- adapté
  - à la représentation des connaissances en IA (raisonnement non monotone),
  - à la spécification et à la résolution de problèmes combinatoires

Héritage multiple : logique des défauts, bases de données déductives, programmation par contraintes, SAT, ...

ASP : le noyau

Un programme logique normal est un ensemble de règles de la forme

$$c \leftarrow a_1, \ \ldots, \ a_n, \ not \ b_1, \ \ldots, \ not \ b_m.$$

où $n \geq 0, m \geq 0$ et $c, a_1, \ldots, \ a_n, \ b_1, \ldots \ b_m$ sont des atomes propositionnels.

« *Si tous les a_i appartiennent à un ensemble réponse et si aucun des b_j n'y appartient, alors c doit appartenir à cet ensemble réponse.* »

$tête(r) = c$	la tête
$corps^+(r) = \{a_1, \ldots, \ a_n\}$	le corps positif
$corps^-(r) = \{b_1, \ldots, \ b_m\}$	le corps négatif

Sémantique des modèles stables

P un programme logique normal. X un ensemble d'atomes.
Le réduit (dit de Gelfond et Lifschitz) de P par X est le programme

$$P^X = \{tête(r) \leftarrow corps^+(r). \mid corps^-(r) \cap X = \emptyset\}$$

- P^X est un programme défini, donc il possède un unique modèle de Herbrand minimal noté $Cn(P)$.
- Un modèle stable [GL88] de P est un ensemble d'atomes S tel que $S = Cn(P^S)$.
- Un programme peut avoir aucun, un ou plusieurs modèles stables.
- Déterminer si un programme possède ou non un modèle stable est NP-complet

Quelques exemples simples

- $P_1 = \left\{ \begin{array}{l} a \leftarrow b. \\ b \leftarrow a. \end{array} \right\}$ possède 1 seul modèle stable $\emptyset$, alors qu'il a deux modèles classiques $\{\neg a, \neg b\}$ $(\emptyset)$ et $\{a, b\}$.
- $P_2^{\{a,b\}} = \left\{ \begin{array}{l} a \leftarrow . \\ b \leftarrow a, not\ d. \\ c \leftarrow a, not\ b. \end{array} \right\}$ $\quad Cn(P_2^{\{a,b\}}) = \{a, b\}$

  P_2 possède un seul modèle stable $\{a, b\}$
- $P_3 = \left\{ \begin{array}{l} a \leftarrow not\ b. \\ b \leftarrow not\ a. \end{array} \right\}$ possède deux modèles stables $\{a\}$ et $\{b\}$
- $P_4 = \left\{ \begin{array}{l} a \leftarrow . \\ b \leftarrow a, not\ d. \\ d \leftarrow b. \end{array} \right\}$ n'en possède aucun, il est inconsistant.

Un modèle est construit par un enchaînement maximal de règles génératrices, ie : applicables (à partir de $\emptyset$) et non contradictoires.

ASP : variations autour d'un même thème

- Intégration de la négation forte pour distinguer
  $traverser\_rail \leftarrow not\ train.$ et $traverser\_rail \leftarrow \neg train.$
  On parle de programme logique étendu et d'answer set [GL91] au lieu de modèles stables, mais une réduction au cas sans négation est possible.
- Intégration de la disjonction en tête de règle [GL91]
  $$c_1;\ \ldots;\ c_p \leftarrow a_1,\ \ldots,\ a_n,\ not\ b_1,\ \ldots,\ not\ b_m.$$
  sémantique identique sauf que l'on accepte uniquement les modèles minimaux.
  $P = \{\ a; b \leftarrow .\ \}$ possède 2 answer sets : $\{a\}$ et $\{b\}$
  Le problème de décision de l'existence d'un modèle pour un programme disjonctif devient Σ_2^p-complet, à cause de la minimalité.
- Préférences : disjonctions ordonnées dans la tête des règles, relation d'ordre partiel sur les règles ou sur les atomes, . . .[DSTW04]

ASP : variations autour d'un même thème...

- Possibilistic Answer Set Programming [NGSL06] : degré de certitude attaché à chaque règle, définition de modèles stables possibilistes

Exemple

$$P = \left\{ \begin{array}{ll} (mary., 1) & (john., 0.6) \\ (stormy\_meeting \leftarrow mary, john., 0.8) & \\ (bob \leftarrow not\ peter., 0.9) & (peter \leftarrow not\ bob., 0.6) \\ (stormy\_meeting \leftarrow mary, bob., 1) & \end{array} \right\}$$

$$S_1 = \{(mary, 1), (john, 0.6), (bob, 0.9), (stormy\_meeting, 0.9)\}$$

$$S_2 = \{(mary, 1), (john, 0.6), (peter, 0.6), (stormy\_meeting, 0.6)\}$$

- Fuzzy Answer Set Programming [NCV06] : approche multivaluée.

Domaines d'applications

- Web sémantique
  projet à Vienne `http://www.kr.tuwien.ac.at/staff/roman/asp_sw`
  workshop `http://www.bd.cesma.usb.ve/alpsws07,...`
- D'un point de vue formel l'ASP est un sous-cas de la logique des défauts. Donc le développement de systèmes opérationnels de raisonnement par défaut est aujourd'hui possible.
  Ex : thèse de F. Nouia (LIPN, D. Kayser) « Extraction et Utilisation des Normes pour un Raisonnement Causal dans un Corpus Textuel »
  - Modélisation en logique des défauts.
  - Implémentation en ASP : traduction des formules logiques et des règles de défauts en ASP. Certaines restrictions peuvent apparaître en théorie, elles sont surmontées en pratique.
  - Au final un système opérationnel contenant environ 200 règles, utilisant le solveur Smodels et capable de faire émerger les connaisances implicites (violations de normes) dans des récits d'accidents.
- Causalité, ...
- Planification, langage d'actions, ...

Des programmes avec variables

On peut écrire des programmes au 1erordre.
Au moment de la résolution les solveurs travaillent avec toutes les règles propositionnelles qu'il est possible (mais pas inutile) de produire en remplaçant chaque variable par toutes les constantes du domaine.

$$P = \left\{ \begin{array}{l} p(1).,\ p(2).,\ q(1).,\ q(2).,\ r(2). \\ s(X,Y) \leftarrow p(X),\ q(Y), not\ r(Y). \end{array} \right\}$$

est équivalent à

$$P = \left\{ \begin{array}{l} p(1).,\ p(2).,\ q(1).,\ q(2).,\ r(2). \\ s(1,1) \leftarrow p(1),\ q(1), not\ r(1). \\ s(1,2) \leftarrow p(1),\ q(2), not\ r(2). \\ s(2,1) \leftarrow p(2),\ q(1), not\ r(1). \\ s(2,2) \leftarrow p(2),\ q(2), not\ r(2). \end{array} \right\}$$

qui admet un modèle stable $\{p(1), p(2), q(1), q(2), r(2), s(1,1), s(2,1)\}$.

Une méthode pour résoudre certains CSP

- On modélise le problème par un programme.
- Chaque solution au problème est un ensemble réponse (un modèle) du programme.
- La méthode : écrire un programme comportant 3 types de règles
  - Règles de description/énumération des données.
  - Règles de «guess» pour décrire l'espace de recherche, pour générer tous les ensembles de réponses possibles, toutes les solutions potentielles du problème.
  - Règles de «check» pour décrire les contraintes et supprimer les ensembles qui ne peuvent pas être des solutions.

La 3-coloration d'un graphe non orienté $G = (V, E)$

DONNÉES

// les n sommets du graphe et ses arêtes

$v(1). \ldots . v(n) \ldots e(i, j). \ldots .$

GUESS

// un sommet est rouge s'il n'est pas vert, ni bleu

$rouge(X) \leftarrow v(X),\ not\ vert(X),\ not\ bleu(X).$

// un sommet est vert s'il n'est pas rouge, ni bleu

$vert(X) \leftarrow v(X),\ not\ rouge(X),\ not\ bleu(X).$

// un sommet est bleu s'il n'est pas rouge, ni vert

$bleu(X) \leftarrow v(X),\ not\ rouge(X),\ not\ vert(X).$

CHECK

// deux voisins ne doivent pas être tous les deux de la même couleur

$\leftarrow e(X, Y),\ rouge(X),\ rouge(Y).$

$\leftarrow e(X, Y),\ vert(X),\ vert(Y).$

$\leftarrow e(X, Y),\ bleu(X),\ bleu(Y).$

Des extensions pour une meilleure modélisation

NB : dépendants des solveurs.

- *Expressions arithmétiques :* $X + Y < Z, \ldots$
- *Littéraux conditionnels :* $\{p(X) : q(X)\}$ *représente l'énumération* $\{\ldots, p(a_i), \ldots\}$ *satisfaisant également* $q(a_i)$
- *Contraintes de cardinalité :* $K_{min}\{\ldots, l_i, \ldots\}K_{max}$ *représente les ensembles contenant entre* K_{min} *et* K_{max} *littéraux parmi les* l_i*.*
- *Extensions avec des littéraux pondérés :* $P_{min}\{\ldots, l_i = p_i \ldots\}P_{max}$ *la somme des poids* p_i *des littéraux* l_i *appartenant à l'ensemble doit être comprise entre* P_{min} *et* P_{max}*.*
- *Fonctions d'aggrégat :* $min, max, sum, \ldots$ *pour calculer une valeur numérique à partir d'un ensemble, ex :*
  $0 <= \#countX, Y : a(X, Z, k), b(1, Z, Y) <= 3.$

Ex : une 3-coloration plus précise.

```
%     2
%     |
%5—1—3
%     |
%     4
% le graphe
v(1..5).
e(1,Y) :- v(Y), Y>1.
% les colorations possibles
rouge(X) :- v(X), not vert(X), not bleu(X).
vert(X) :- v(X), not rouge(X), not bleu(X).
bleu(X) :- v(X), not rouge(X), not vert(X).
% les contraintes de voisinage
:- e(X,Y), rouge(X), rouge(Y).
:- e(X,Y), vert(X), vert(Y).
:- e(X,Y), bleu(X), bleu(Y).
% 1 seul sommet rouge dans la solution
okrouge :- 1 { rouge(X) :v(X) } 1.
:- not okrouge.
% au moins 3 sommets verts dans la solution
okvert :- 3 { vert(X) :v(X) }.
:- not okvert.
```

Ex : planification dans le monde des blocs

```
% les données du problème
const grippers=2.
const lasttime=3.
time(0..lasttime).
block(1..6).
location(B) :- block(B).
location(table).
% état initial
on(1,2,0).
on(2,table,0).
on(3,4,0).
on(4,table,0).
on(5,6,0).
on(6,table,0)
% état final
:- not on(3,2,lasttime).
:- not on(2,1,lasttime).
:- not on(1,table,lasttime).
:- not on(6,5,lasttime).
:- not on(5,4,lasttime).
:- not on(4,table,lasttime).
```

```
% tous les mouvements possibles à tout instant
{ move(B,L,T) : block(B) : location(L) } grippers :- time(T), T<lasttime.
% les effets du déplacement d'un bloc
on(B,L,T+1) :- move(B,L,T), block(B), location(L), time(T), T<lasttime.
% loi d'inertie
on(B,L,T+1) :- on(B,L,T), not neg_on(B,L,T+1),
   location(L), block(B), time(T), T<lasttime.
% unicité de localisation
neg_on(B,L1,T) :- on(B,L,T), L !=L1, block(B), location(L), location(L1), time(T).
% neg_on est le contraire de on
 :- on(B,L,T), neg_on(B, L, T), block(B), location(L), time(T).
% 2 blocs ne peuvent pas être simultanément sur le même bloc
 :- 2 { on(B1,B,T) : block(B1) }, block(B), time(T).
% un bloc ne peut pas être déplacé si un autre bloc est posé dessus
 :- move(B,L,T), on(B1,B,T), block(B), block(B1), location(L), time(T), T<lasttime.
% un bloc ne peut pas être déplacé sur un bloc qui est déplacé simultanément
 :- move(B,B1,T), move(B1,L,T), block(B), block(B1), location(L), time(T), T<lasttime.
```

```
lparse planning.nlp | smodels
produit
move(1,table,0) move(3,table,0) move(2,1,1) move(5,4,1) move(3,2,2) move(6,5,2)
```

Domaines d'applications

- Planification, langage d'actions, ...
- Problèmes de théorie des graphes.
- Comparaison sur CSPLib avec solveurs commerciaux [CMMP06] : Ramsey problem, social golfer, Golomb rulers, car sequencing.
- Configuration de produits (PC), dépendance de paquets dans une distribution Linux.
- Bioinformatique.
- Résolution de SAT ou QBF.
- ...

Les principaux solveurs pour ASP

- Les historiques
  - Smodels et son front-end Lparse, Helsinki, code ouvert, `http://www.tcs.hut.fi/Software/smodels`
  - Dlv, Vienne, code fermé (tendance commerciale), `http://www.dbai.tuwien.ac.at/proj/dlv`

  Points de choix sur les atomes $a \in ensemble \vee a \notin ensemble$
- L'école de Potsdam
  - Nomore (en Prolog) suivi de Nomore ++ (en C++)
    `http://www.cs.uni-potsdam.de/nomore/`
    Points de choix sur les règles r est une règle génératrice ou non, combinés avec points de choix sur atomes.
  - récent : Clasp `http://www.cs.uni-potsdam.de/clasp`, inspiré de solveurs SAT, dirigé par les conflits, techniques inspirées de SAT mais adaptées à ASP.

Les principaux solveurs pour ASP ...

- Ceux basés sur un solveur SAT
  - Assat, Hong Kong, `http://assat.cs.ust.hk`
  - Cmodels, Austin, `http://www.cs.utexas.edu/users/tag/cmodels.html`

  On utilise un solveur SAT pour calculer un modèle (classique) de la complétion de Clark du programme. Si on a un modèle, on en vérifie la minimalité. Si c'est positif, alors c'est un answer set. Sinon, on calcule des formules représentant des boucles dans le programme que l'on ajoute pour créer un nouveau problème SAT à résoudre. Le processus converge vers un answer set ou vers la preuve de l'inconsistance du programme.

  \+ un seul pas suffit pour les programmes tight

  \- nombre exponentiel de boucles possibles.

Évaluation de solveurs pour ASP

- Plateforme de tests de solveurs `http://asparagus.cs.uni-potsdam.de`
- Première compétition à LPNMR'07

  `http://asparagus.cs.uni-potsdam.de/contest`

Conclusion

- Des ressources
  - La revue Theory and Practice of Logic Programming
    `http://www.cambridge.org/journals/journal_catalogue.asp?mnemonic=tlp`
  - La conférence Logic Programming and Nonmonotonic Reasoning
    `http://lpnmr2007.googlepages.com`
  - Le workshop Answer Set Programming `http://www.cs.ttu.edu/asp07`
  - Le groupe WASP `http://wasp.unime.it` (action terminée).
- Lorsque vous aurez un problème à résoudre, pensez à l'approche ASP, ça marche !

Bibliographie

[CMMP06] M. Cadoli, T. Mancini, D. Micaletto, and F. Patrizi. Evaluating asp and commercial solvers on the csplib. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *ECAI 2006, 17th European Conference on Artificial Intelligence,*, pages 68–72, 2006.

[DSTW04] J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 20(2) :308–334, 2004.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *International Conference on Logic Programming*, pages 1070–1080. The MIT Press, 1988.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4) :363–385, 1991.

[NBG+01] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In *PADL '01 : Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages*, volume 1990, pages 169–183, London, UK, 2001. Springer-Verlag.

[NCV06] D. Van Nieuwenborgh, M. De Cock, and D. Vermeir. Fuzzy answer set programming. In *Logics in Artificial Intelligence, 10th European Conference JELIA*, volume 4160 of *LNCS*, pages 359–372. Springer-Verlag, 2006.

[NGSL06] P. Nicolas, L. Garcia, I. Stéphan, and C. Lefèvre. Possibilistic uncertainty handling for answer set programming. *Annals of Mathematics and Artificial Intelligence*, 47(1-2) :139–181, 2006.