Séminaire bibliographique

La programmation logique l'Answer Set Programming (ASP)

Élaboré par :
Emna BEN ABDALLAH

Encadrée par:
M. Olivier ROUX
M. Maxime FOLSCHETTE

Plan

- Introduction
- AnsProlog
  - Syntaxe et alphabet
- L'ASP
  - L'ASP dans AnsProlog
  - Modélisation
  - Exemple des n-Queens
  - Les extensions
- Calcul de l'Answer set
  - Génération de l'Answer set
  - Elimination de l'Answer set
- Conclusion

Introduction

développement de la logique :

son origine : motivations philosophiques

son objet : l'étude des lois de pensée

son rôle : l'étude des *énoncés* et des *jugements*

l'étude des *raisonnements*

→ les moyens d'assembler des énoncés pour en produire des nouveaux.

AnsProLog : Answer Sets Programming in Logic

AnsProLog :

- Langage de programmation logique utilisant la sémantique de l'Answer set
- Très bien adapté à la représentation des connaissances, le raisonnement et la résolution de problèmes déclaratifs.

Syntaxe et alphabet

Un programme logique d'AnsProlog est une collection des règles de la forme:

$$\textbf{\textit{Head}} \leftarrow \textbf{\textit{Body}}$$

$$L_0 \leftarrow L_1,\ldots, L_m, \textbf{\textit{not}}\ L_{m+1}, \ldots, \textbf{\textit{not}}\ L_n$$

L_i des littéraux, *head* et *body* ensembles des littéraux

Head : *conclusion*
Body : *prémisse*

Syntaxe et alphabet

- ***Terme***: t (variable, constante, $f(t_1,\ldots,t_n)$)
- ***Prédicat***: p
- ***Atome*** : $p(t_1,\ldots, t_n)$ = A
- ***Littéral*** : A ¬A

Règles

Head* ← *Body

$A \leftarrow B^+, \textbf{not } B^-$

$L_o \leftarrow L_1,\ldots, L_m, \textbf{not } L_{m+1}, \ldots, \textbf{not } L_n$

$body^+(r) = pos(r) = B^+ = \{ L_1,\ldots L_m \}$

$body^-(r) = neg(r) = B^- = \{ L_{m+1},\ldots L_n \}$

si $L_{k+1},\ldots, L_m$ *est* ***vrai***
et $L_{m+1},\ldots, L_n$ est **faux**

$\Big\}$ L_o *est* ***vrai***

Règles

Une réalité ou **un fait** : *body* est vide

$L_0 \leftarrow$ ou L_0

Une contrainte: une règle telle que k=0 et $L_0=\perp$ Ce symbole faux au *head*, est souvent éliminé

$\leftarrow L_1, \ldots, L_m, \textbf{\textit{not}}\ L_{m+1}, \ldots, \textbf{\textit{not}}\ L_n$

Programmes d'AnsProlog

L_0 or ... or $L_k \leftarrow L_{k+1}, ..., L_m,$ ***not*** $L_{m+1}, ...,$ ***not*** L_n

- AnsProlog$^{\textbf{-not}}$: k=0 et n=m
- AnsProlog$^{\neg}$: k=0 et $\neg$ A
  extended logic programs
- AnsProlog$^{\textbf{or}}$: k > 0, L_i du ***head*** sont séparés par "or" (ou ";").
- AnsProlog$^{\perp}$: contraintes
- AnsProlog$^{\perp}$, AnsProlog$^{\text{-not},\perp}$, AnsProlog$^{\neg,\perp}$, AnsProlog$^{\text{or},\perp}$
- AnsProlog$^{\neg,\text{ or},\perp}$: **AnsProlog*** : ASP

Answer Set Programming (ASP)

L'ASP

- ASP = AnsProLog*
  - Apparition fin 1990
  - Nouvelle vie à la programmation logique
  - Spécifie '*quels*' sont les résultats attendus et non pas '*comment*' les atteindre
  - Outil efficace de représentation de connaissances
  - Composé d'un ensemble de règles, de faits et de contraintes
  - L'Answer set est l'ensemble des faits cohérents qui peuvent être dérivés à l'aide des règles (calculé par le solveur)

La sémantique de l'ASP

Règles des programmes ASP:

L_o or ... or $L_k \leftarrow L_{k+1}, \ldots L_m$, ***not*** $L_{m+1}, \ldots,$ ***not*** L_n

si $L_{k+1}, \ldots, L_m$ *est* ***vrai***
et $L_{m+1}, \ldots, L_n$ est **faux**

} au moins un des littéraux :
$L_o, \ldots, L_k$ *est* ***vrai***

Modélisation

Les étapes de modélisation

- Enumérer avec des faits,
- Expliquer avec des règles,
- Générer toutes les possibilités avec des cardinalités,
- Filtrer avec des contraintes

Modélisation

Exemple n-Queens

- Un échiquier de dimension n × n,
- Deux reines n'apparaissent jamais sur la même ligne, colonne, ou diagonale.

Modélisation

- **Enumération avec des faits:**
  - un atome *q (i, j):* une reine est à la position (i, j), $i^{éme}$ ligne et $j^{éme}$ colonne.
  - *q'(i, j):* pas de reine à la position (i, j)
  - *d*(X): X est une dimension de l'échiquier

Modélisation

- **Expliquer avec des règles**

$$q(X, Y) \leftarrow d(X), d(Y), \textbf{not}\ q'(X, Y)$$

$$q'(X, Y) \leftarrow d(X), d(Y), \textbf{not}\ q(X, Y)$$

Modélisation

- **Générer toutes les possibilités avec des cardinalités**

Un échiquier tel que n=2 → 16 answer sets

{q'(1,1), q'(1,2), q'(2,1), q'(2,2), d(1), d(2)}

{q (1,1), q'(1,2), q'(2,1), q'(2,2), d(1), d(2)}

q	

{q(1,1), q'(1,2), q'(2,1), q(2,2), d(1), d(2)}

q	
	q

{q(1,1), q(1,2), q(2,1), q(2,2), d(1), d(2)}

q	q
q	q

Modélisation

- **Filtrer avec des contraintes**

Jamais deux reines sur la même ligne, colonne ou diagonale

$\leftarrow q(X, Y), q(X', Y), X' \neq X, d(X), d(Y), d(X')$

$\leftarrow q(X, Y), q(X, Y'), Y' \neq Y, d(X), d(Y), d(Y')$

$\leftarrow q(X, Y), q(X', Y'), |X - X'| = |Y - Y'| \; X' \neq X, Y' \neq Y, \; d(X), d(Y), d(X'), d(Y')$

Modélisation

- **Filtrer avec des contraintes**

Que toutes les n reines soient sur l'échiquier

- *hasq(X)* est vrai quand il y a une reine dans la X^{ème} ligne :

$$hasq(X) \leftarrow d(X), d(Y), q(X, Y)$$

Une ligne qui n'a pas une reine doit invalider la solution:

$$\leftarrow d(X), \textbf{not}\ hasq(X)$$

Les extensions du Langage

- **Programme logique disjonctif**

la disjonction des atomes : q_i avec $0 \leq i \leq k$

$$q_0 \; ; \ldots ; \; q_k.$$

p ; q signifie que « *p est vrai* ***XOU*** *q est vrai* »

Exemple n-Queens:

$$q(X, Y) \; ; \; q'(X, Y) \leftarrow \; d(X), d(Y)$$

l'existence (q) et la non-existence (q') dans la même position (X, Y) est absurde

Les extensions du Langage

- **Disjonction ordonnée**

p × q : si p possible alors p, mais si p est impossible alors au moins q.

Exemple:

viande × végétarien ←

jus × eau ← viande

eau × jus ← ***not*** viande

L'answer set: { viande, jus }

Les extensions du Langage

- **Programme logique imbriqué**

Les corps et les têtes des règles peuvent contenir des expressions booléennes arbitraires

Exemple n-Queens:

$$q(X, Y) \ ; \ \textbf{not}\ q(X, Y) \leftarrow \ d(X), d(Y)$$

Les extensions du Langage

- **Les contraintes de cardinalité**

Étendre les littéraux de la forme:

$$l\ \{q_1, \ldots, q_m\}\ u$$

$m \geq 1$ et l et u les limites inférieure et supérieure

Pour que X satisfait cette contrainte:

$$l \leq |\ \{q_1, \ldots, q_m\} \cap X\ | \leq u$$

Exemple n-queens:

$1\ \{\ q(X, Y) : d(X)\ \}\ 1 \leftarrow d(Y)$

$1\ \{\ q(X, Y) : d(Y)\ \}\ 1 \leftarrow d(X)$

Les extensions du Langage

- Autres extensions
  - la somme (sum),
  - le compteur (*count*),
  - le minimum (*min*)…

Calcul des answer sets

- **La fermeture**

Soit Π programme ASP
X un ensemble de littéraux
si pour chaque règle r dans Π,
$head(r) \subseteq X$
si $body^{+}(r) \subseteq X$ *et* $body^{-}(r) \cap X = \varnothing$

} Alors X est fermé sous Π

$C_n(\pi)$: le plus petit ensemble d'atomes qui est fermé sous π
l'answer set de π

Calcul des answer sets

- **La réduction**

π un programme d'ASP

X un ensemble d'atomes de HB_π

$\boldsymbol{\pi^X} = \{ \text{head}(r) \leftarrow \text{body}^+(r) \text{ tel que } r \in \pi \text{ avec } \text{body}^-(r) \cap X = \emptyset \}$

Si $\boldsymbol{C_n(\pi^X)=X}$: X est un answer set de π^X

$\Rightarrow$ X est aussi un answer set de π.

Calcul des answer sets

Application:

Soit le programme:

$p \leftarrow p$ (r1)

$q \leftarrow \textbf{\textit{not}}\ p$ (r2)

head(r1)={p} $body^{+}$(r1)) = {p} , $body^{-}$(r1) = ø , et

head(r2)={q} $body^{+}$(r2) = ø , $body^{-}$(r2) = {p}.

Calcul des answer sets

$\boldsymbol{\pi^X} = \{ \text{head}(r) \leftarrow \text{body}^+(r) \text{ tel que } r \in \pi \text{ avec } \text{body}^-(r) \cap X = \emptyset \}$

- Si on prend X= Ø,

$\text{body}^-(r1) \cap X = \emptyset$
à ok alors $\{ \text{head}(r1) \leftarrow \text{body}^+(r1) \} \in \boldsymbol{\pi^X}$

$\text{body}^-(r2) \cap X = \emptyset$
à ok alors $\{ \text{head}(r2) \leftarrow \text{body}^+(r2) \} \in \boldsymbol{\pi^X}$

$p \leftarrow p$ (r1)

$q \leftarrow \textbf{\textit{not}}\ p$ (r2)

$\pi^X = \{p \leftarrow p, q \leftarrow\}$
$C_n(\pi^X) = \{q\}$
$C_n(\pi^X) \nmid X$
→ X= ø, n'est pas un answer set de π

Calcul des answer sets

$\boldsymbol{\pi}^{\mathbf{X}} = \{ \text{head}(r) \leftarrow \text{body}^{+}(r) \text{ tel que } r \in \pi \text{ avec } \text{body}^{-}(r) \cap X = \emptyset \}$

- Si $X = \{q\}$

$\text{body}^{-}(r1) \cap X = \emptyset$
→ ok alors $\{ \text{head}(r1) \leftarrow \text{body}^{+}(r1) \} \in \boldsymbol{\pi}^{\mathbf{X}}$
$\text{body}^{-}(r2) \cap X = \emptyset$
→ ok alors $\{ \text{head}(r2) \leftarrow \text{body}^{+}(r2) \} \in \boldsymbol{\pi}^{\mathbf{X}}$

$p \leftarrow p$ (r1)

$q \leftarrow \textbf{\textit{not}}\ p$ (r2)

$\pi^{X} = \{p \leftarrow p\ ,\ q\}$
$C_n(\pi^{X}) = \{q\}$
$C_n(\pi^{X}) = X$
→ $X = \{q\}$ est un answer set de π

Calcul des answer sets

- **Les answer sets des programmes disjonctifs**

A; $\neg A$

Décomposition du body$^+$ alors,

n règles de cette forme → 2^n answer sets possibles

Exemple

p ; $\neg p$

q ; $\neg q$

2 règles disjonctives: 4 answer sets :

$\{p, q\}$, $\{p, \neg q\}$, $\{\neg p, q\}$ *et* $\{\neg p, \neg q\}$.

Calcul des answer sets

- **Elimination des answer sets indésirables:**

Solution: contraintes (*head* vide)

$\leftarrow L_1, \ldots, L_m,$ ***not*** $L_{m+1}, \ldots,$ ***not*** L_n

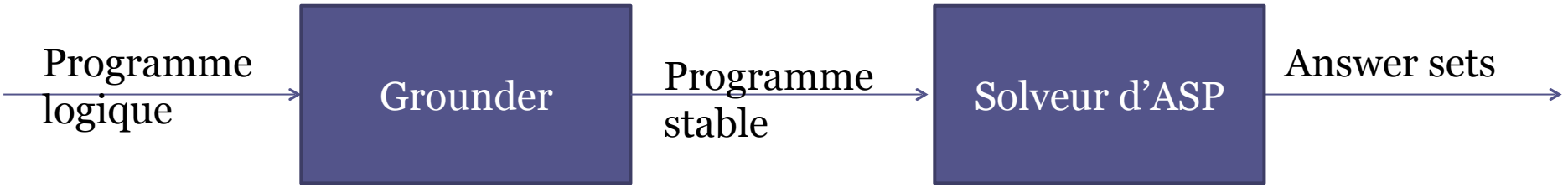
Soit X est un answer set de π

π' programme contraint de π

Si $L_1, \ldots, L_m \in X$ et $L_{m+1}, \ldots, L_n \notin X$

Alors X est un answer set de π'

Les solveurs de l'Answer Set



Programme logique: ensemble de règles (constantes, variables, atomes)

Grounder : supprime les variables
calcule un ordre
capable de traiter les règles disjonctives:
Logiciels : GRINGO, DLV, LPARSE

Solveurs calculer l'ensemble des anwser sets pour les programmes finis sans ***not*** dans les heads des règles (sans règles disjonctives)
Logiciels: SMODELS, DLV, CMODELS, CLASP

Conclusion

L'***Answer set programming*** (ASP) : AnsProlog* particulièrement adapté à la résolution de problèmes combinatoires complexes de recherche:

- les applications de planification de la production,
- la configuration du produit,
- le diagnostic et
- les problèmes de la théorie des graphes.