

Raisonnement non monotone

Module IAMSI

*Intelligence Artificielle et Manipulation
Symbolique de l'Information*

Cours 4



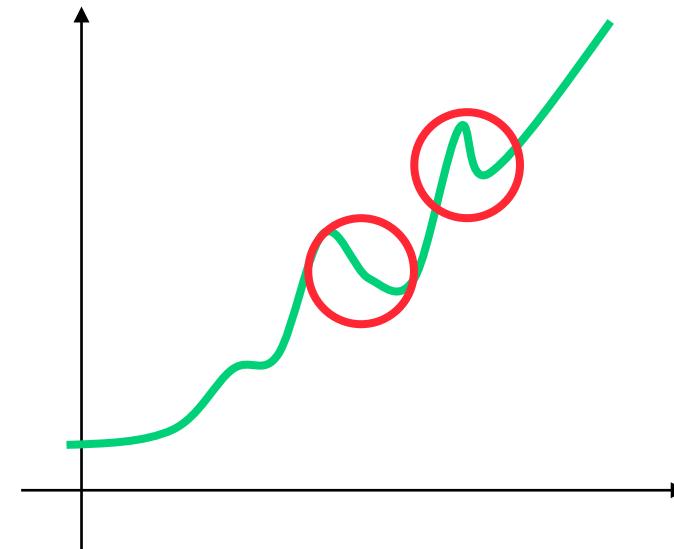
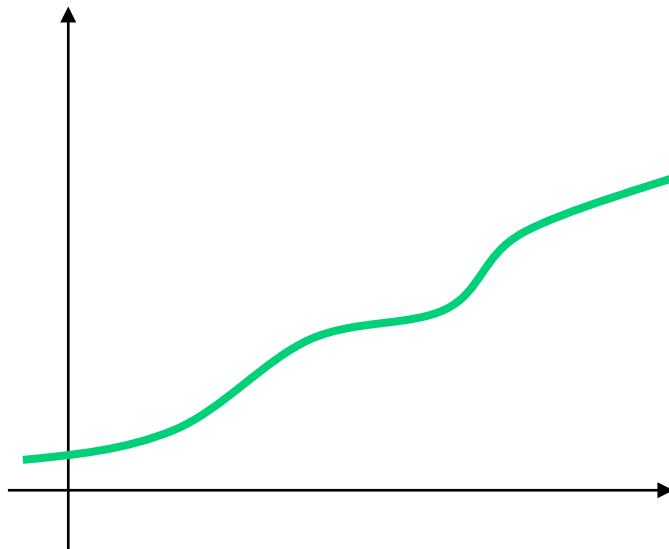
Raisonnement non monotone

Non monotonie et raisonnement
Logique des défauts
Hypothèse du monde clos
Circonscription
« Answer Set Programming »



Fonction non monotone

- Un fonction f est monotonessi
$$\forall(x, y) [x>y \Rightarrow f(x) > f(y)]$$



Systèmes formels: démonstration

Système formel:

- Axiomes
- Règles d'inférences

Preuve: une *preuve* d'un théorème A est une séquence finie de formules F_0, F_1, \dots, F_n telles que

- $F_n = A$
- $\forall i \in [0, n]$ F_i est soit un *axiome*, soit obtenu par application d'une *règle d'inférence* sur un ensemble de formules F_j telles que $j < i$

Intrinsèquement monotone: si on accroît le nombre d'axiomes, on accroît aussi le nombre de théorèmes.



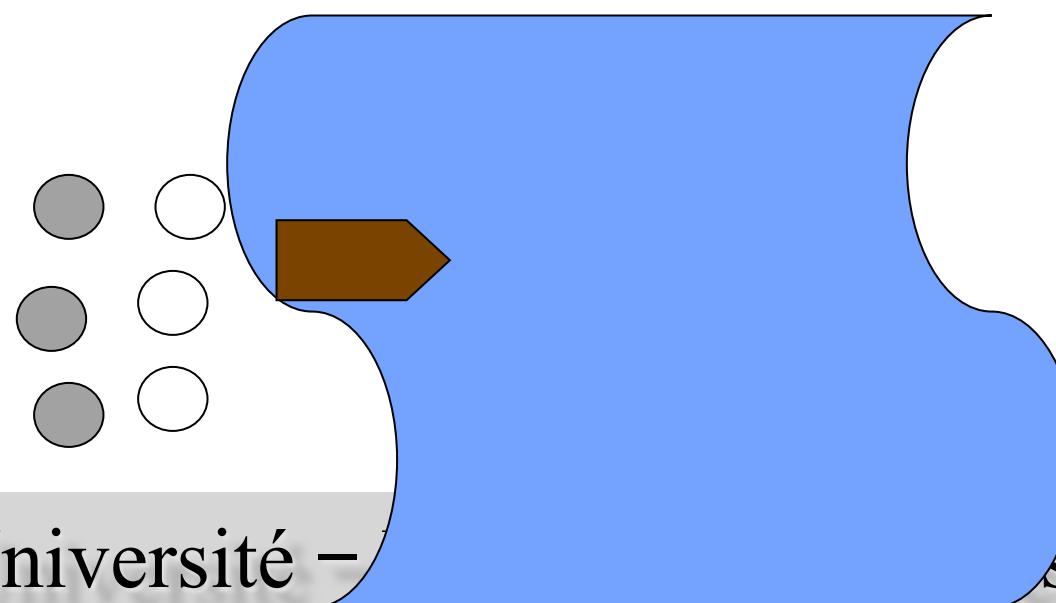
« Le robot et le bébé »

John McCarthy



Les missionnaires et les cannibales

Trois missionnaires et trois cannibales se trouvent sur les rives d'un fleuve en pleine Amazonie. La barque ne supporte pas plus de deux passagers, et le nombre de missionnaires doit partout et toujours être supérieur ou au moins égal à celui des cannibales pour éviter les drames... Comment faire pour que tous traversent, sans perte d'aucune sorte ?



A
S
A



CNRS – Sorbonne Université –

Besoin de non monotonie: questions techniques

- **Bases de données**
 - valeurs nulles
- **Clauses de Horn**
 - un seul littéral positif dans les clauses.
 - « $P :- N_1, N_2, \dots N_q$ »
 - Négation par échec
 - pas de négation en conclusion
- **CLIPS: négation**
 - dans les prémisses – négation par échec
 - dans les conclusions, « retract »: maintien de la cohérence



Retrait des faits

L
I
P

A
C
A
S
A

~~BF = {A, C}~~

R1: A => B

R2: B et C => D

R3: C et D => E

~~BF = {A, B, C}~~

~~BF = {A, B, C, D}~~

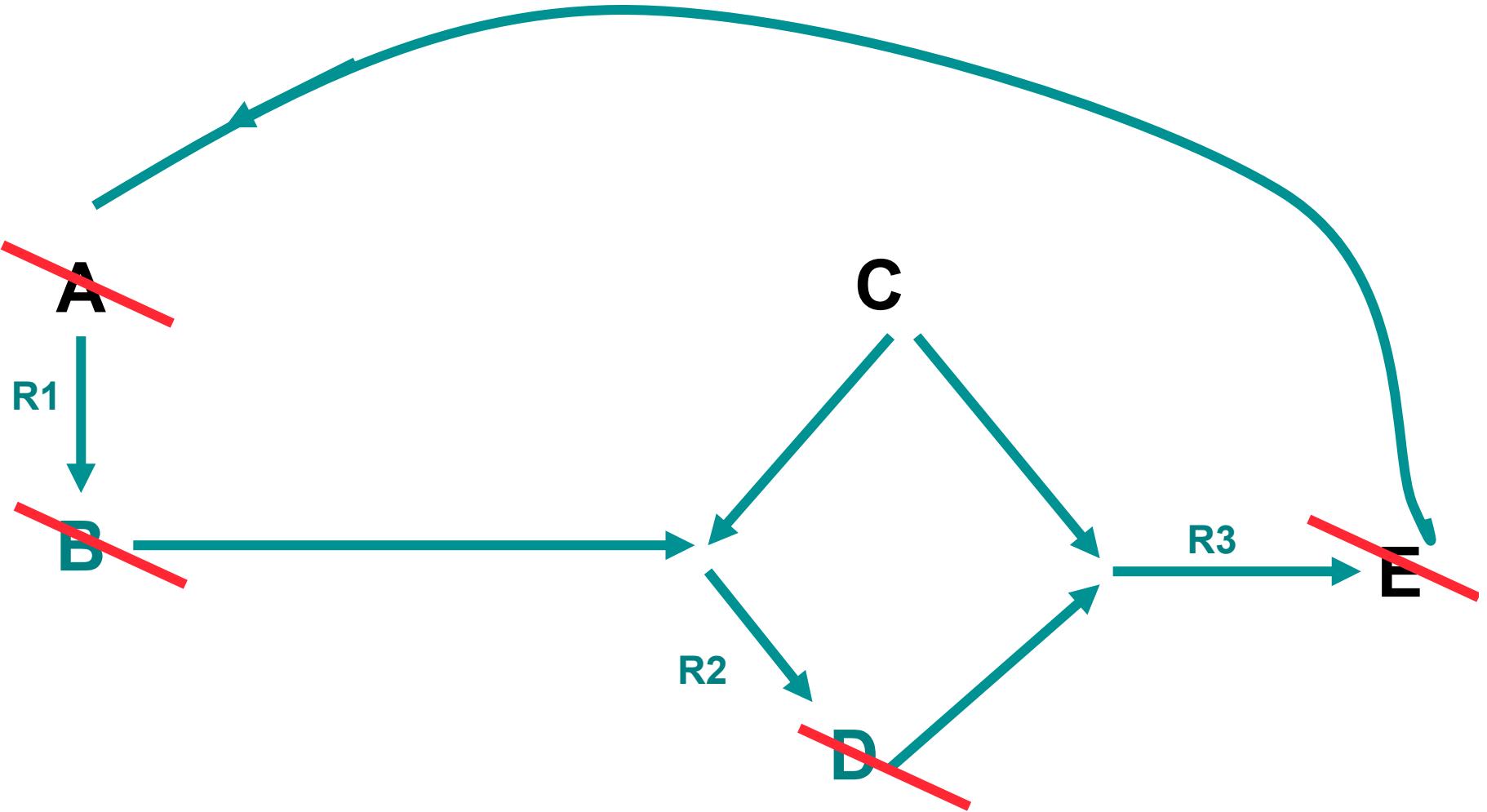
~~BF = {A, B, C, D, E}~~

R4: E => $\neg A$



L
I
P
6
A
C
A
S
A

Réseau de justifications Système de maintien de la vérité (TMS)



Incôherences – exemple

$\forall x \text{oiseau}(x) \Rightarrow \text{vol}(x)$

$\forall x \text{autruche}(x) \Rightarrow \neg \text{vol}(x)$

$\forall x \text{ pingouin}(x) \Rightarrow \neg \text{vol}(x)$

$\forall x \text{autruche}(x) \Rightarrow \text{oiseau}(x)$

$\forall x \text{ pingouin}(x) \Rightarrow \text{oiseau}(x)$

$\text{oiseau}(\text{Samuel})$

$\text{pingouin}(\text{Philémon})$

$\text{autruche}(\text{Jeanne})$



vol(Samuel)
 $\neg \text{vol}(\text{Philémon})$
 $\neg \text{vol}(\text{Jeanne})$



CWA – “Closed World Assumption”

Négation par échec

Principe: si on ne peut pas prouver P alors on ajoute
 $\neg P$ — ou $\text{not } P$ (*négation PROLOG*) —

- Ce qui n'est pas explicitement vrai est faux:

Si $R(a_1, \dots, a_i) \in W$ Alors $R(a_1, \dots, a_i)$ est vrai

Sinon $R(a_1, \dots, a_i)$ est faux

- Opposée à “Open World Assumption”: l'absence de connaissance n'implique pas la fausseté

Ecrivains	Livre
D. Lecourt	Encyclopédie des sciences
E. Klein	Encyclopédie des sciences
M. Serres	Trésor des sciences
E. Klein	Paysage des sciences



Incohérences – exemple

$\forall x \text{ oiseau}(x) \text{ et not } \neg \text{vol} \Rightarrow \text{vol}(x)$

$\forall x \text{ autruche}(x) \Rightarrow \neg \text{vol}(x)$

$\forall x \text{ pingouin}(x) \Rightarrow \neg \text{vol}(x)$

$\forall x \text{ autruche}(x) \Rightarrow \text{oiseau}(x)$

$\forall x \text{ pingouin}(x) \Rightarrow \text{oiseau}(x)$

oiseau(Samuel)

$\text{pingouin(Philémon)}$

autruche(Jeanne)



vol(Samuel)
 $\neg \text{vol(Philémon)}$
 $\neg \text{vol(Jeanne)}$





Incohérences – Résolution

$\text{non_vol}(x) \vee \text{vol}(x)$

$\neg\text{autruche}(x) \vee \text{non_vol}(x)$

$\neg\text{pingouin}(x) \vee \text{non_vol}(x)$

$\neg\text{autruche}(x) \vee \text{oiseau}(x)$

$\neg\text{pingouin}(x) \vee \text{oiseau}(x)$

$\text{oiseau}(\text{Samuel})$

$\text{pingouin}(\text{Philémon})$

$\text{autruche}(\text{Jeanne})$

Résolution

$\text{oiseau}(\text{Philémon})$

$\text{oiseau}(\text{Jeanne})$

$\text{non_vol}(\text{Philémon})$

$\text{non_vol}(\text{Jeanne})$



Supposition du monde fermé

- Si f est un littéral positif qui n'est pas impliqué par K (autrement dit tel que $\neg [K \models f]$) alors $K \vdash \neg f$

non_vol(x) \vee vol(x)

\neg autruche(x) \vee non_vol(x)

\neg pingouin(x) \vee non_vol(x)

\neg autruche(x) \vee oiseau(x)

\neg pingouin(x) \vee oiseau(x)

oiseau(Samuel)

pingouin(Philémon)

autruche(Jeanne)

Réso

oiseau(Phi

oiseau(Jea

non_vol(Ph

non_vol(Je

\neg vol(Philémon)

\neg vol(Jeanne)

\neg pingouin(Samuel)

\neg autruche(Samuel)

\neg pingouin(Jeanne)

\neg autruche(Philémon)

2 modèles possibles :

\neg non_vol(Samuel)

vol(Samuel)

ou

\neg vol(Samuel)

non_vol(Samuel)



Formalisation de la CWA

- **Ajouter la négation de ce qui n'est pas impliqué**
{ français (Anatole) \vee roumain (Gabriel) }
- **Qu'en va-t-il de** français (Anatole) **et de** roumain (Gabriel) ?
- **Il ne sont pas impliqués... En effet, la négation de l'un ou de l'autre de ces littéraux ne conduit pas à dériver la clause vide par application de la résolution**
- **En appliquent la CWA, on doit donc ajouter:**
 \neg français (Anatole) **et** \neg roumain (Gabriel).
- **Mais, le résultat à savoir:** { français (Anatole) \vee roumain (Gabriel), \neg français (Anatole), \neg roumain (Gabriel) } **est incohérent...**



Formalisation de la CWA (suite)

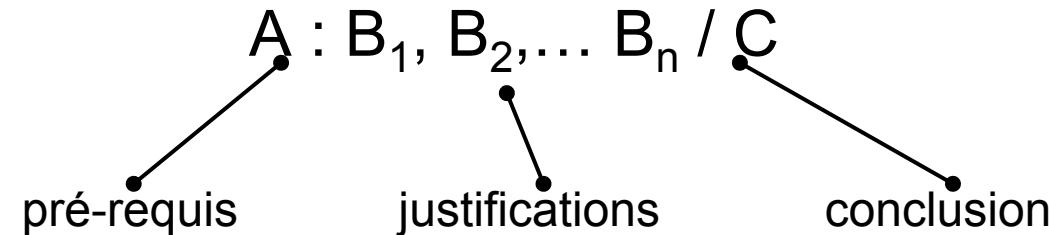
- **Propriété:** la CWA n'introduit pas d'incohérences sur la base de connaissance K si l'intersection de tous les modèles de Herbrand K est aussi un modèle de K
- **En logique des propositions, cette condition est équivalente à avoir un modèle minimal unique.**
- **Autres formalisations:** $K \wedge \{ \neg f \mid f \in F \}$
- **Différentes définitions de F ont été proposées:**
 - CWA: f est un littéral positif qui n'est pas impliqué par K (autrement dit tel que $\neg [K \models f]$)
 - GCWA (generalized): f est un littéral positif tel que, pour chaque clause positive c si $K \models c$ est faux alors $K \models c \vee f$ est faux aussi
 - ECCWA (extended GCWA): comme GCWA, mais si f est une conjonction de littéraux positifs



Raisonnements erronés et logique des défauts

La logique des défauts

- Logique non-monotone [Reiter, 80]
- Règle de défaut :



- Peu d'américains sont socialistes :
$$\frac{\text{américain}(x) : \neg\text{socialiste}(x)}{\neg\text{socialiste}(x)}$$



un défaut est une expression du type A : B₁, B₂,... B_n / C

- Interprétation:

Si A est reconnu être vrai

Si B₁, B₂, ... et B_n sont cohérents avec les autres assertions

Alors conclure C

- Exemples:

criminal(X) \wedge foreigner(X) : expel(X) / expel(X)

politicalRefugee(X) \Rightarrow \neg expel(X)

politician(X) \wedge introducedAbroad(X) : \neg diplomat(X)
/ traitor(X)

oiseau(X) : vole(X) / vole(X)

pingouin(X) / oiseau(X)

pingouin(X) / \neg vole(X)

- **Sémantique:**

Une théorie de défauts est une paire (Δ, W) où

- Δ est un ensemble de défauts inclus dans l' ensemble Γ
- W est un ensemble de propositions de L

- **Extension d'une théorie de défauts (Δ, W) :**

- Points fixes de la fonction $\Gamma_{(\Delta, W)}$
- Etant donné un ensemble E de formules propositionnelles, $\Gamma_{(\Delta, W)}(E)$ est définie comme suit: c'est le plus petit ensemble de formules telles que
 - ❖ $\Gamma_{(\Delta, W)}(E)$ contient E
 - ❖ Pour tout défaut δ de Δ du type:
si $p(d) \in \Gamma_{(\Delta, W)}(E)$ et
si $\neg j(d) \cap E = \emptyset$ (*autrement dit si E n'est pas contradictoire avec j(d)*)
alors $c(d) \in \Gamma_{(\Delta, W)}(E)$
 - ❖ $\Gamma_{(\Delta, W)}(E)$ est déductivement clos

Incohérences – défauts

oiseau(x) : vol(x) / vol(x)

autruche(x) / \neg vol(x)

pingouin(x) / \neg vol(x)

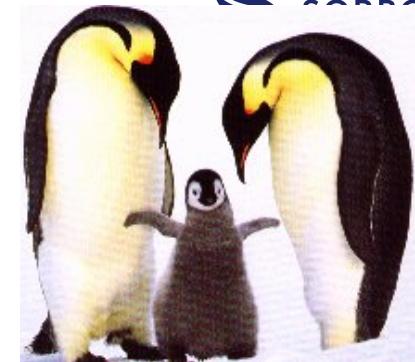
autruche(x) / oiseau(x)

pingouin(x) / oiseau(x)

oiseau(Samuel)

pingouin(Philémon)

autruche(Jeanne)



oiseau(Philémon)

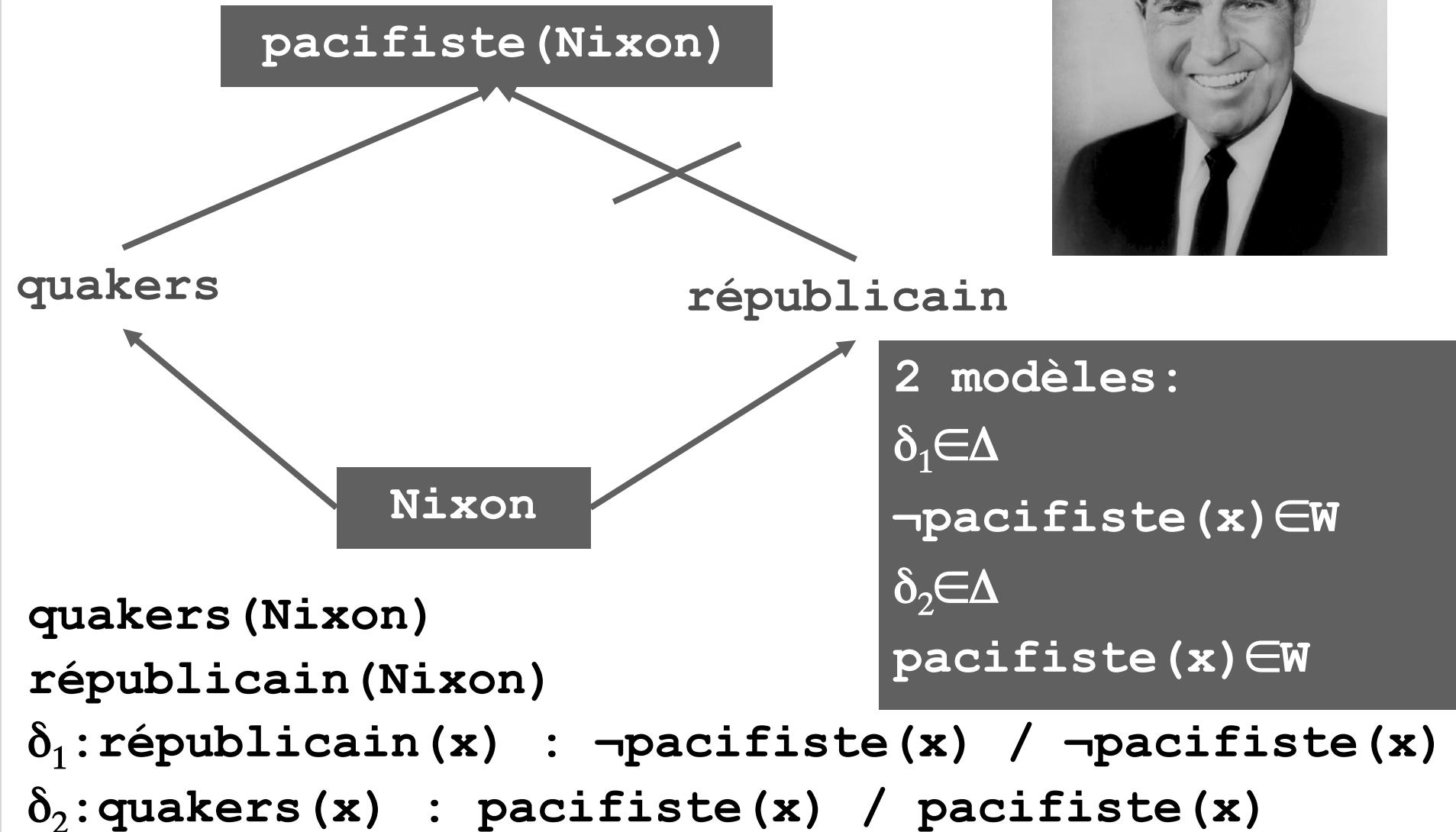
oiseau(Jeanne)

vol(Samuel)

\neg vol(Philémon)

\neg vol(Jeanne)

Nixon est-il pacifiste?



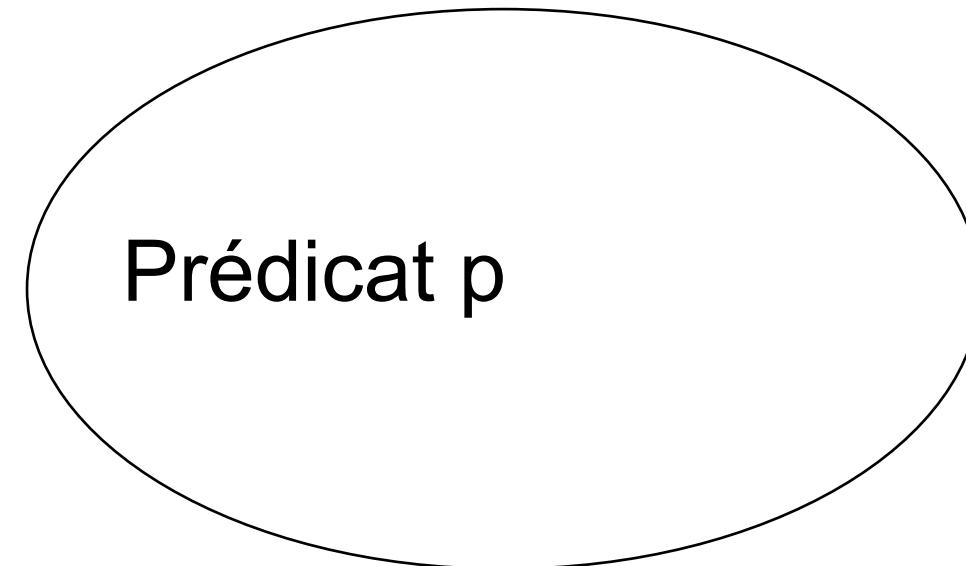
Circonscription – John McCarthy 1980

- Idée: construire les modèles les plus « petits »
- $\text{Circ}(T) = \{M \mid M \models T \text{ et } \neg \exists N \text{ tel que } N \models T \text{ et } N \subseteq M\}$
- *Exemple:* la formule $T = a \wedge (b \vee c)$ admet trois modèles:
 1. a, b et c sont vraies
 2. a et b sont vraies, c est fausse
 3. a et c sont vraies, b est fausse.
- *Le premier modèle n'est pas minimal*
- *Les modèles 2 et 3 sont incomparables*



Circonscription – John McCarthy 1980

- **Idée:** *circonscrire* l'extension d'un ou de plusieurs prédictats dans une théorie logique du premier ordre



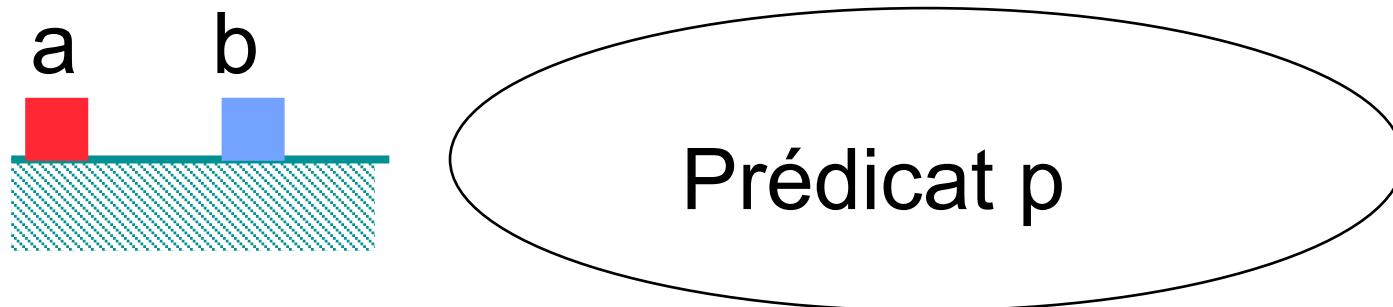
- **Méthode:** utilisation d'axiomes du second ordre



Circonscription – John McCarthy 1980

- $\text{Circ}(A; p) = A(p) \wedge \neg \exists P (A(P) \wedge P < p)$

où $P < p$ signifie que l' extension de P est un sous-ensemble de l' extension de p



Exemple:

théorie T: $\text{sur_table}(a) \wedge \text{sur_table}(b)$

$\text{Circ}(T; \text{sur_table})$ est équivalent à
 $\forall X. \text{sur_table}(X) \Leftrightarrow (X = a) \vee (X = b)$



ASP – Answer Set Programming

- **Answer Sets** (ensembles réponses):
« modèles stables »
- **ASP**: Programmation déclarative fondée sur la sémantique des « modèles stables » de la programmation logique
- **But des « modèles stables »**: définir une sémantique claire pour les programmes logiques avec une négation par échec
- **Origine**: Gelfond & Lifschitz 1988



ASP – Answer Set Programming

Un programme Π est un ensemble d'expression ρ

$$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$$

Où

- les L_i sont des littéraux (*atomes ou négation d'atomes*)
- Le « not » est une négation par échec

Signification intuitive: pour toute interprétation de Herbrand qui rend vraie $\{L_{k+1}, L_{k+2}, \dots, L_m\}$ sans satisfaire $\{L_{m+1}, \dots, L_n\}$ on peut dériver $\{L_0, L_1, \dots, L_k\}$



ASP - exemple

```
vole(X) ← oiseau(X), not non_vole(X)  
non_vole(X) ← autruche(X)
```



```
oiseau(X) ← autruche(X)  
oiseau(anatole) ←  
autruche(jeanne) ←
```

**S = {vole(anatole), oiseau(anatole), oiseau(jeanne),
non_vole(jeanne), autruche(jeanne)}**
est un “answer set” en français « ensemble réponse »



ASP – exemple 2

p ← a

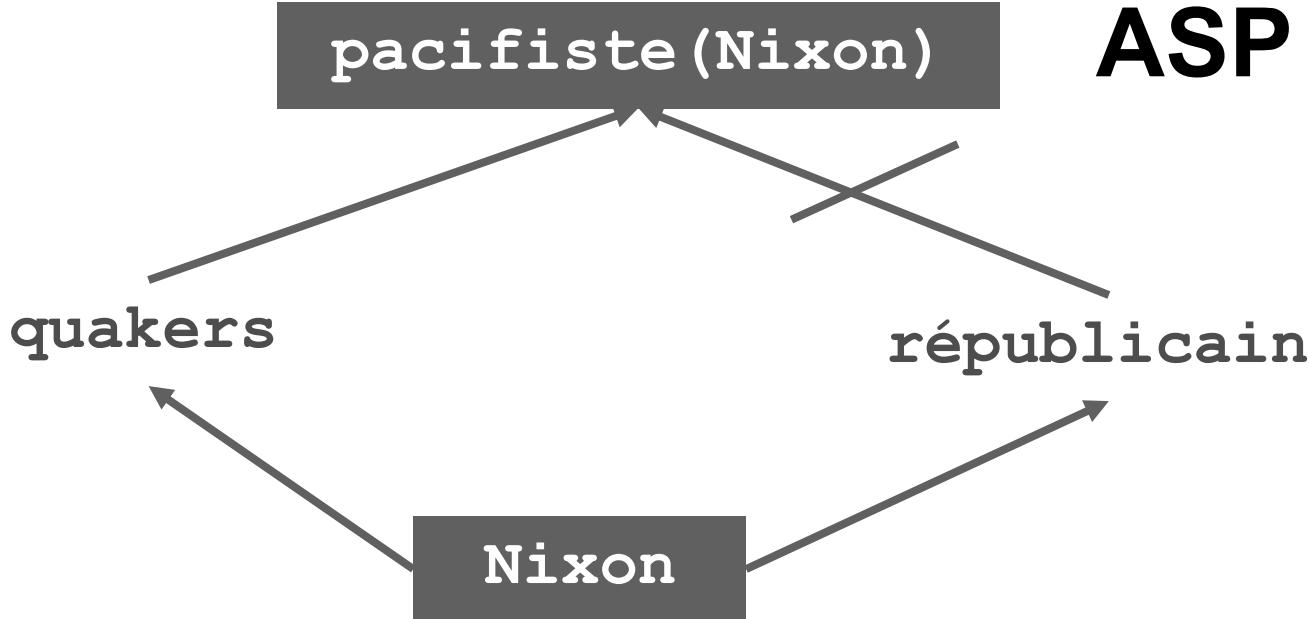
a ← not b

b ← not a

$S_1 = \{a, p\}$ est un “answer set”

$S_2 = \{b\}$ est un “answer set”





ASP – exemple 3



$\neg \text{pacifiste}(X) \leftarrow \text{republicain}(X), \text{not pacifiste}(X).$
 $\text{pacifiste}(X) \leftarrow \text{quakers}(X), \text{not } \neg \text{pacifiste}(X).$
 $\text{republicain}(\text{nixon}).$
 $\text{quakers}(\text{nixon}).$

« Ensembles réponses »

$$\begin{aligned} S_1 &= \{\text{republicain}(\text{nixon}), \text{quakers}(\text{nixon}), \text{pacifiste}(\text{nixon})\} \\ S_2 &= \{\text{republicain}(\text{nixon}), \text{quakers}(\text{nixon}), \neg \text{pacifiste}(\text{nixon})\} \end{aligned}$$



AnsProlog: implémentations

$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$

AnsProlog^{-not}: sans négation par échec

AnsProlog[⊥]: avec \perp dans la tête

AnsProlog^{¬, ⊥}: avec \neg et \perp dans la tête

AnsProlog^{or, ⊥}: avec or et \perp dans la tête

AnsProlog^{or, ¬, ⊥}: correspond à AnsProlog*

...



Sous-classes usuelles de programmes

$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$

- **Programmes logiques généraux – programmes logiques normaux – AnsProlog :**
Ensembles Π de règles ρ dont les L_i sont des atomes et telles que $k = 0$.
- **Programmes définis ou programmes logiques de Horn – AnsProlog^{-not}:**
Ensembles Π de règles ρ dont les L_i sont des atomes avec $k = 0$ et $m = n$
- **Programmes logiques étendus – AnsProlog⁺:**
Ensembles Π de règles ρ telles que $k = 0$.
- **Programmes logiques disjonctifs normaux – AnsProlog^{or}:**
Ensembles Π de règles ρ dont les L_i sont des atomes et $m = n$



Exemple de programme logique normal – AnsProlog

$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$

- Programmes logiques généraux – programmes logiques normaux – AnsProlog :

Ensembles Π de règles ρ dont les L_i sont des atomes telles que $k = 0$.

```
vole(X)  ← oiseau(X) ,  not ab(X) .  
ab(X)   ← pingouin(X) .  
oiseau(X) ← pingouin(X) .  
oiseau(albert) .  
pingouin(alphonse) .
```



Exemple de programme logique défini AnsProlog^{-not}

$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$

- Programmes définis ou programmes logiques de Horn – AnsProlog^{-not}:

Ensembles Π de règles ρ dont les L_i sont des atomes avec $k = 0$ et $m = n$

```
anc(X, Y) ← parent(X, Y).
```

```
anc(X, Y) ← parent(X, Z), anc(Z, Y).
```

```
parent(idoménée, deucalion).
```

```
parent(deucalion, minos).
```

```
parent(minos, europe).
```



Exemple de programme logique étendu – AnsProlog⁷

$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$

- Programmes logiques étendus – AnsProlog⁷:

Ensembles Π de règles ρ telles que $k = 0$

```
vole(X) ← oiseau(X) , not ¬vole(X) .
```

```
¬vole(X) ← pingouin(X) .
```

```
oiseau(albert) .
```

```
oiseau(alphonse) .
```

```
pingouin(alphonse) .
```



Exemple de programme logique disjonctif normal – AnsProlog^{or}

$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$

- Programmes logiques disjonctifs normaux – AnsProlog^{or}:

Ensembles Π de règles ρ dont les L_i sont des atomes et $m = n$

`oiseau(X) or reptile(X) ← pond_oeufs(X).`

`pond_oeufs(georges).`



AnsProlog* - AnsProlog or, \neg , \perp

$$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$$

Syntaxe: PROLOG d' Edimbourg

`L0|L1|... |Lk :- Lk+1, ..., Lm, not Lm+1, ..., not Ln.`

Les L_k sont des littéraux, c'est-à-dire des atomes A_k ou des négations d'atomes $\neg A_k$

`p :- a.`

`p :- a.`

`a :- not b.`

`a :- not b.`

`b :- not a.`

`b :- not a.`



AnsProlog* - AnsProlog or, \neg , \perp

```

oiseau(X) :- autruche(X) .
oiseau(X) :- canari(X) .
oiseau(X) :- pingouin(X) .
vole(X) :- oiseau(X) , not non_vole(X) .
non_vole(X) :- autruche(X) .

```

```

non_vole(X) :- pingouin(X) .

```

```

autruche(jeanne) .
canari(romain) .
oiseau(alphonse) .
pingouin(albert) .

```



AnsProlog* - AnsProlog or, \neg , \perp

Exemple traité avec CWA – suite

```
oiseau(X) :- autruche(X) .
```

```
oiseau(X) :- pingouin(X) .
```

```
-vole(X) :- autruche(X) .
```

```
-vole(X) :- pingouin(X) .
```

```
vole(X) :- oiseau(X) , not -vole(X) .
```

```
autruche(jeanne) .
```

```
oiseau(samuel) .
```

```
pingouin(philemon) .
```



AnsProlog*: programmation exemple 2

p ← a
a ← not b
b ← not a

p:-a.
a:-not b.
b:-not a.



ASP – exemple 3 programmation AnsProlog*

```
-pacifiste(X) :- republicain(X), not pacifiste(X).  
pacifiste(X) :- quakers(X), not -pacifiste(X).  
republicain(nixon).  
quakers(nixon).
```



```
pacifiste(X) :- quakers(X), not -pacifiste(X) .  
-pacifiste(X) :- republicain(X), not pacifiste(X) .  
quakers(nixon) .  
republicain(nixon) .
```



Applications de AnsProlog*

- **Bases de données déductives**
- **Contraintes**
- **Planification**
- **Modélisation du raisonnement**
- **Représentation des connaissances**
- **Preuve de programme ou de spécifications**
- ...



Domaine de Herbrand infini

- **Exemple 4**

p(a) .

p(b) .

p(c) .

p(f(X)) :- p(X) .

- **Exemple 4 bis**

base(a ; b ; c) .

p(B) :- base(B) .

**p(f(B)) :- base(B),
p(B) .**



ASP – Sémantique: univers de Herbrand $HU_{\mathcal{L}}$

L
I
P
6
A
C
.A
S
A

Un programme Π est un ensemble d'expression ρ

$$\rho : L_0 \text{ or } L_1 \text{ or } \dots L_k \leftarrow L_{k+1}, L_{k+2}, \dots L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$$

Soit \mathcal{L} un langage contenant des

- Variables
- Fonctions
- Prédicats
- Connectives $\{\neg, \text{or}, \leftarrow, \text{not}, ',', '\cdot'\}$
- Ponctuations $\{, '(', ')', '\cdot'\}$
- Le symbole \perp

L'univers de Herbrand sur \mathcal{L} dénoté $HU_{\mathcal{L}}$ est l'ensemble des termes totalement instanciés formés avec les fonctions de \mathcal{L}

La base de Herbrand sur \mathcal{L} dénoté $HB_{\mathcal{L}}$ est l'ensemble des atomes de \mathcal{L} totalement instanciés par des termes de $HU_{\mathcal{L}}$



Sémantique des programmes

AnsProlog*

- $\text{ground}(\Pi)$ rassemble l' ensemble des instances de Herbrand des règles ρ de Π
- Une *interprétation de Herbrand* d'un programme Π est un sous-ensemble I de la base de Herbrand de Π notée HB_{Π}
- Un *ensemble de réponse* (« answer set ») d'un programme et un sous-ensemble minimal S de HB_{Π} qui est clos sur les instances de base de $\text{ground}(\Pi)$



Exemples

- Exemple a

a:-not b.

Un modèle stable

- Exemple b

a:-not b.

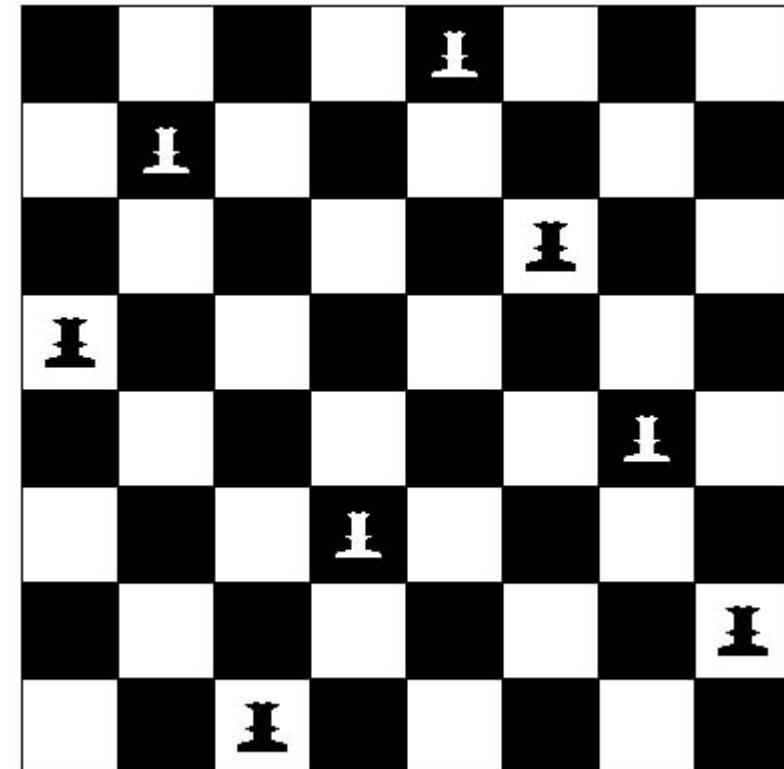
b:-not a.

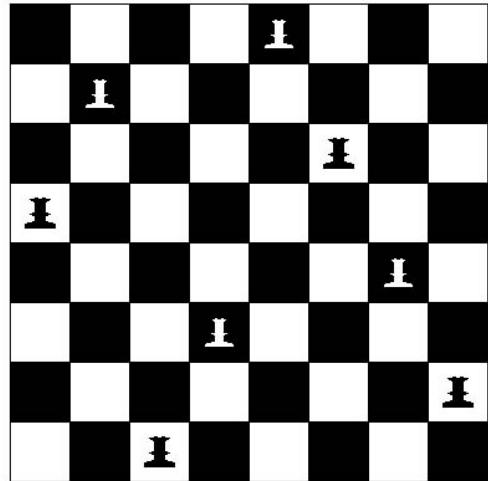
Deux modèles stables



Les huit reines

- Placer 8 reines sur un échiquier en sorte que l'on ait jamais deux reines en prise l'une sur l'autre





```

reine(1..8).
ligne(1..8).
colonne(1..8).

% limite le nombre de possibilités: le nom
% de la reine = nom colonne
:- reine(R), ligne(X), colonne(Y),
   a(R, X, Y), neq(R, Y).

% placement de toutes les reines
placee(R) :- reine(R), ligne(X),
            colonne(Y), a(R, X, Y).

:- reine(R), not placee(R).

% Il doit y avoir au moins une
a(R, X, Y) :- reine(R), ligne(X),
             colonne(Y), not not_a(R, X, Y).

not_a(R, X, Y) :- reine(R), ligne(X),
                 colonne(Y), not a(R, X, Y).

```

Les huit reines: programmation avec AnsProlog*

```

:- reine(I), ligne(X), colonne(Y),
   ligne(U), colonne(V), a(I, X, Y),
   a(I, U, V), neq(Y, V).

:- reine(I), ligne(X), colonne(Y),
   ligne(U), colonne(V), a(I, X, Y),
   a(I, U, V), neq(X, U).

:- reine(R), ligne(X), colonne(Y),
   a(R, X, Y), reine(S), a(S, X, Y),
   neq(R, S).

:- reine(R), ligne(X), colonne(Y),
   colonne(V), a(R, X, Y), reine(S),
   a(S, X, V), neq(Y, V).

:- reine(R), ligne(X), colonne(Y),
   a(R, X, Y), ligne(U), reine(S),
   a(S, U, Y), neq(X, U).

:- ligne(L), colonne(C), ligne(U),
   colonne(V), reine(R), reine(S),
   a(R, L, C), a(S, U, V), neq(R, S),
   eq(L-C, U-V).

:- ligne(L), colonne(C), ligne(U),
   colonne(V), reine(R), reine(S),
   a(R, L, C), a(S, U, V), neq(R, S),
   eq(L+C, U+V).

```



Notion de domaine

- **On peut « typer » les variables**
- **Exemple:**

```
#domain reine(R).  
#domain reine(RR).
```

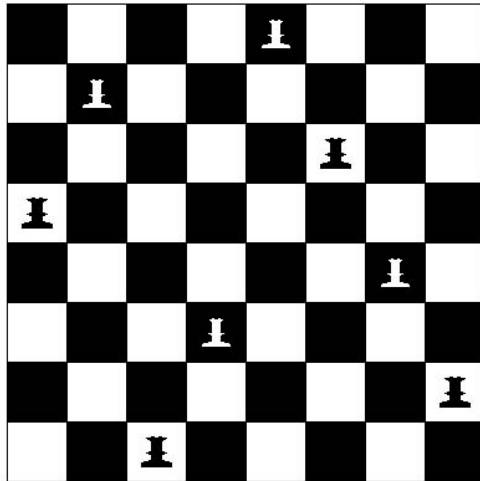
```
reine(1..8).
```

```
: - a(R, X, Y), a(RR, X, Y).
```

est équivalent à:

```
: - reine(R), a(R, X, Y),  
  reine(RR), a(RR, X, Y).
```





Les huit reines avec AnsProlog*

```

#domain reine(R) .                                % Le nom de la reine correspond à celui de
#domain reine(RR) .                             la colonne
#domain ligne(L) .                               :- a(R, L, C), neq(R,C).

#domain ligne(LL) .                            placee(R) :- a(R, L, R).
#domain colonne(C) .                           :- reine(R), not placee(R).

#domain colonne(CC) .                         a(R, L, R) :- not not_a(R, L, R).
#domain colonne(CC) .                         not_a(R, L, R) :- not a(R, L, R).

reine(1..8) .                                 :- a(R, L, R), a(R, LL, R), neq(L, LL).
ligne(1..8) .                                 :- a(R, L, R), a(RR, L, RR), neq(R, RR).
colonne(1..8) .                                :- a(R, L, R), a(RR, LL, RR), neq(R, RR),
                                         eq(L-R, LL-RR).
                                         :- a(R, L, R), a(RR, LL, RR), neq(R, RR),
                                         eq(L+R, LL+RR).

```



AnsProlog*: le menteur

```
#domain proposition(Prop).
#domain personne(P).
#domain personne(PP).

faux(Prop) :- not vrai(Prop).
vrai(Prop) :- not faux(Prop).

vrai(Prop) :- dit(P, Prop), not
menteur(P).
faux(Prop) :- dit(P, Prop),
menteur(P).

:- vrai(Prop), faux(Prop).

personne(jean;alphonse;simon;the
obald).
proposition(la_terre_est_bleue_c
omme_une_orange;
un_couteau_n_a_ni_manche_ni_la
me; menteur(P)).
```

```
#domain proposition(Prop).
dit(jean,
la_terre_est_bleue_comme_une_o
range).
dit(alphonse,
un_couteau_n_a_ni_manche_ni_la
me).
%dit(simon, menteur(simon)).

menteur(PP) :- dit(P,
menteur(PP)).
menteur(PP) :-_
vrai(menteur(PP)).
:- menteur(PP),
faux(menteur(PP)).

#hide proposition(Prop).
#hide personne(P).
#hide personne(PP).

#hide dit(P, Prop).
#hide personne(P).
#hide faux(Prop).
```



L
I
P
6

A
C
A
S
A

SUDOKU

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

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

