Systèmes à base de règles

en logique du 1^{er} ordre

Un petit aperçu des objets de base
(HAI710I)

Rappels de logique du premier ordre

- Cette logique décrit des **objets** et les **relations** entre ces objets
- Les objets sont appelés termes : variables ou constantes
  (pas de fonctions ici)
- Les relations sont appelées des prédicats
  Tout prédicat a une arité (nombre d'arguments, qui est fixe)
- Atome **$p(e_1 \ldots e_k)$**
  où **p** est un prédicat (ou relation)
  les $\mathbf{e_i}$ sont des termes
- Les variables sont quantifiées universellement ou existentiellement
- On ne raisonne que sur des formules fermées : toute variable est dans la portée d'un quantificateur

Règles (conjonctives) positives et faits

Règle : $\forall x_1 \ldots \forall x_n \, (H \rightarrow C)$ où :

- H est une conjonction d'atomes et C est un atome
- $x_1 \ldots x_n$ sont les variables de H
- **toutes** les variables de **C** apparaissent dans **H**

$\forall x \forall y$ ((Pays(x) $\wedge$ FaitPartie(x, UE) $\wedge$ PermisValable (y,x)) $\rightarrow$ PermisValable (y, F))

Notation simplifiée (on omet les quantificateurs) :

Pays(x) $\wedge$ FaitPartie(x, UE) $\wedge$ PermisValable (y,x) $\rightarrow$ PermisValable (y, F)

Un fait correspond à une règle à hypothèse vide :
c'est donc un atome instancié (sans variables)

Pays(Danemark), FaitPartie(Danemark, UE), ...

Exemple (permis de conduire) K = (BF,BR)

F1 : Ville(Copenhague)
F2 : Pays(Danemark)
F3 : FaitPartie(Copenhague, Danemark)
F4 : FaitPartie(Danemark,UE)
F5 : LieuObtentionPermis(Ingrid, Copenhague)
F6 : Pays(F)
F7 : FaitPartie(F, UE)

R1 : Ville(x1) $\wedge$ Pays(y1) $\wedge$ FaitPartie(x1,y1) $\wedge$LieuObtentionPermis(z1,x1) $\rightarrow$ PermisValable(z1,y1)

"Si z1 obtient un permis (de conduire) dans une ville qui fait partie d'un certain pays, alors z1 a un permis valable dans ce pays"

R2 : Pays(x2) $\wedge$ FaitPartie(x2, UE) $\wedge$ PermisValable (y2,x2) $\rightarrow$ PermisValable (y2,F)

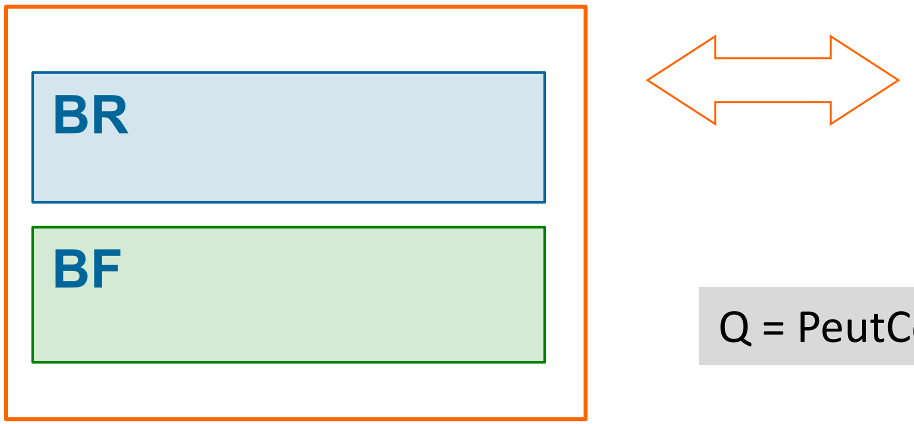
"Les permis valables dans un pays de l'UE sont valables en France"

R3 : PermisValable(x3,y3) $\rightarrow$ PeutConduire(x3,y3)

"Si on a un permis valable pour un certain lieu, on peut conduire dans ce lieu »

Requête Q = PeutConduire(Ingrid,F) ?

Interrogation d'une base de connaissances



Requête Q

Q = PeutConduire(Ingrid,F) ?

Q = PeutConduire (Ingrid,x) $\wedge$ FaitPartie(x, UE) ?

Requête conjonctive : conjonction d'atomes (vue comme un ensemble)

- Si instanciée (sans variables) : réponse oui / non
- Sinon, on veut toutes les valeurs possibles pour les variables dans la base de connaissances

Idée :
1) Calculer la base de faits saturée BF*
2) Interroger BF*

Saturation par retour à la logique des propositions

Idée : se ramener à des règles d'ordre 0 en instanciant les variables de toutes les façons possibles

Ex: R3 : PermisValable(x3,y3) → PeutConduire(x3,y3)
instanciée par les 6 constantes apparaissant dans *K*

=> 36 règles :

PermisValable(Cop,Cop) → PeutConduire (Cop,Cop)

...

PermisValable(Ingrid,France) → PeutConduire (Ingrid,France)

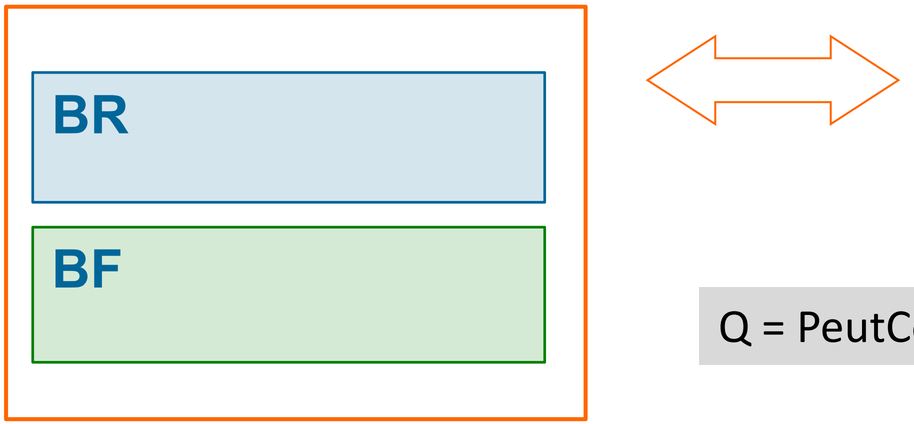
Chaque atome instancié est ensuite vu comme un **symbole propositionnel** :

Ville_Copenhague
PermisValable_Ingrid_France

On peut donc appliquer le chaînage avant propositionnel pour calculer la base de faits saturée

Pas de miracle : la base de règles propositionnalisée est exponentiellement plus grande que la base d'origine

Interrogation d'une base de connaissances



Requête Q

Q = PeutConduire(Ingrid,F) ?

Q = PeutConduire (Ingrid,x) $\wedge$ FaitPartie(x, UE) ?

Requête conjonctive : conjonction d'atomes (vue comme un ensemble)

- Si instanciée (sans variables) : réponse oui / non
- Sinon, on veut toutes les valeurs possibles pour les variables dans la base de connaissances

Idée :

1) Calculer la base de faits saturée BF*
2) Interroger BF*

Interrogation d'une base de faits

Oublions les règles pour l'instant

BF

p(a,b)
p(b,a)
p(a,c)
q(b,b)
q(a,c)
q(c,b)

Q = { p(x,y), p(y,z), q(z,x) }

Réponses à Q dans BF ?

$$x \mapsto b \quad y \mapsto a \quad z \mapsto c$$

$$x \mapsto b \quad y \mapsto a \quad z \mapsto b$$

Un **homomorphisme *h*** de Q dans BF est une application des variables de Q dans les constantes de BF telle que :

$$h(Q) \subseteq BF$$

où h(Q) est obtenu à partir de Q en substituant chaque variable x par h(x)

Répondre à une requête conjonctive (Q) dans une BF

- Si Q est sans variable : la réponse à Q est oui si $Q \subseteq BF$
  (autrement dit, il existe un homomorphisme « vide » de Q dans BF)

  Traduction logique : $BF \vDash Q$

- De façon générale :

  tout homomorphisme h de Q dans BF définit une réponse à Q

  Traduction logique : $BF \vDash h(Q)$
  où h est un homomorphisme de Q dans BF

  Remarque : comme Q n'est pas une formule fermée, $BF \vDash Q$ n'aurait pas de sens ; on considère donc les formules fermées obtenues en appliquant les homomorphismes

Exemple : Base de connaissances (pistes cyclables)

BF

Direct(A,B)

Direct(B,C)

Direct(C,D)

Direct(D,B)

Requêtes

Direct(A,C) ?   Direct(x,B) Λ Direct(B,y) ?

Comment demander s'il y a un chemin de A à C ?

BR

Direct(x,y) → Chemin(x,y)

Direct(x,y) Λ Chemin(y,z) → Chemin(x,z)

Requête Q = Chemin(A,C) ?

Pour répondre aux requêtes, on va considérer la base de faits saturée BF*

Chaînage avant (logique d'ordre 1)

BF

Direct(A,B)
Direct(B,C)
Direct(C,D)
Direct(D,B)

BR

Direct(x,y) → Chemin(x,y)
Direct(x,y) ∧ Chemin(y,z) → Chemin(x,z)

Une règle R : H → C est applicable à BF s'il existe un homomorphisme h de H dans BF

- Cette application est utile si h(C) ∉ BF
- Appliquer R à BF consiste à ajouter h(C) dans BF
- BF est saturée (par rapport à BR) si aucune application d'une règle de BR à BF n'est utile

ALGORITHME DE CHAÎNAGE AVANT (ORDRE 1)

Algorithme ForwardChaining (K) *// Données : K = (BF, BR)*
Début *// Résultat : BF saturée par BR*

Fin ← faux

Tant que non fin

nouvFaits ← ∅ *// ensemble des nouveaux faits obtenus à cette étape*

Pour toute règle R : H → C ∈ BR

Pour tout (nouvel) homomorphisme S de H dans BF

Si S(C) ∉ (BF UnouvFaits)
Ajouter S(C) à nouvFaits

Une règle peut s'appliquer plusieurs fois

Si nouvFaits = ∅
Fin ← vrai
Sinon Ajouter les éléments de nouvFaits à BF

Fin

BF* peut être exponentielle en la taille de BF
(l'exposant est l'arité maximale des prédicats)
La complexité de FC(K) n'est plus polynomiale

Adéquation et complétude du chaînage avant

- Le chaînage avant est **adéquat** pour les règles positives :

  Pour tout atome instancié A, si $A \in BF^*$ alors $BF, BR \models A$

- Le chaînage avant est **complet** pour les règles positives :

  Pour tout atome instancié A, si $BF, BR \models A$ alors $A \in BF^*$

Soit Q une requête conjonctive
et s une substitution des variables de Q par des constantes

$BF, BR \models s(Q)$ ssi $s(Q) \subseteq BF^*$

autrement dit : *ssi* s est un homomorphisme de Q dans BF^*

Exemple (pistes cyclables)

BF

Direct(A,B)
Direct(B,C)
Direct(C,D)
Direct(D,B)

BR

Direct(x,y) → Chemin(x,y)
Direct(x,y) ∧ Chemin(y,z) → Chemin(x,z)

Q = Chemin(A,x) ∧ Chemin(x,D)

« trouver tous les x qui sont sur un chemin de A à D »

On cherche les homomorphismes de Q dans BF*

Réponses : $x \mapsto B$ $x \mapsto C$ $x \mapsto D$

Synthèse (Règles conjonctives positives en ordre 1)

- Il faut instancier les règles pour pouvoir les appliquer sur une base de faits
- Ceci peut se faire :
  - « a priori » en instanciant les variables des règles par toutes les constantes de la base de connaissances
  - « à la volée » par des tests d'homomorphisme
- Le chaînage avant est plus complexe mais il reste adéquat et complet (par rapport à la conséquence logique)
- Les réponses à une requête conjonctive q sur une base de faits F sont obtenues en recherchant les homomorphismes de q dans F