

# Moteurs d'inférence

Boi Faltings

Laboratoire d'Intelligence Artificielle  
[boi.faltings@epfl.ch](mailto:boi.faltings@epfl.ch)  
<http://moodle.epfl.ch/>

# Inférence

Un *moteur d'inférence* est un algorithme qui trouve des conséquences logiques  $q$  d'un ensemble de prémisses  $\{P\}$ :

$$\{P\} \vdash q$$

Par exemple:

Premisses  $\{P\}$ :

- ① lapin  $\Rightarrow$  animal
- ② animal  $\Rightarrow$  bouge

Résultat  $q$ :

lapin  $\Rightarrow$  bouge

Algorithme central à un système à base de connaissances!

# Propriétés

L'algorithme est:

- *fondé* si  $q$  est toujours une conséquence de  $\{P\}$
- *complet* s'il trouve tous les  $q$

# Propriétés

L'algorithme est:

- *fondé* si  $q$  est toujours une conséquence de  $\{P\}$
- *complet* s'il trouve tous les  $q$
- aucun algorithme n'est fondé et complet!

# Propriétés

L'algorithme est:

- *fondé* si  $q$  est toujours une conséquence de  $\{P\}$
- *complet* s'il trouve tous les  $q$
- aucun algorithme n'est fondé et complet!
- Nous allons voir la résolution, qui est *complète pour la réfutation*: est garanti de trouver toutes les contradictions qui découlent de  $\{P\}$ .

# Propriétés

L'algorithme est:

- *fondé* si  $q$  est toujours une conséquence de  $\{P\}$
- *complet* s'il trouve tous les  $q$
- aucun algorithme n'est fondé et complet!
- Nous allons voir la résolution, qui est *complète pour la réfutation*: est garanti de trouver toutes les contradictions qui découlent de  $\{P\}$ .
- La résolution permet de trouver des preuves par contradiction.

# Etapes de l'inférence

- ① transformation des expressions en forme normale.
- ② application itérative de la résolution.
- ③ extraction du résultat.

# Forme Normale

Premier pas: standardiser le format des données.

Pour un moteur d'inférence, il convient d'utiliser la forme normale *conjonctive*:

$$\{\mathcal{P}\} = a_1 \wedge a_2 \wedge \dots$$

On appelle les  $a_i$  des *clauses*; ils sont soit:

- une proposition simple, ou
- une disjonction de propositions simples ( $a_i = b_1 \vee b_2 \vee \dots$ ).

Exemple:  $a_1 \wedge (a_2 \vee (a_3 \wedge a_4)) \Rightarrow a_1 \wedge (a_2 \vee a_3) \wedge (a_2 \vee a_4)$

Base de connaissances = ensemble de clauses

{1. $a_1$ , 2.( $a_2 \vee a_3$ ), 3.( $a_2 \vee a_4$ ), 4., ....}

# Représentation informatique

- Proposition simple: liste (prédicat, arg1, ...)  
Exemple: (oncle, Jacques, Charles)
- Proposition composée:  
liste imbriquée (<connecteur>, exp1, ...)  
Exemple:  
(AND, (jeune, Jacques), (OR, exp1, exp2))

## Exemple: Forme Normale

- ➊  $v \wedge (d \vee (y \wedge b))$   
 $\Rightarrow v \wedge (d \vee y) \wedge (d \vee b)$   
 $\Rightarrow (\text{AND}, v, (\text{OR}, d, y), (\text{OR}, d, b))$
- ➋  $m \vee (d \vee (y \wedge b))$   
 $\Rightarrow (m \vee d \vee y) \wedge (m \vee d \vee b)$   
 $\Rightarrow (\text{AND}, (\text{OR}, m, d, y), (\text{OR}, m, d, b))$
- ➌  $(e \wedge y) \vee (\neg e \wedge (d \vee y))$

## Exemple: Forme Normale

- ①  $v \wedge (d \vee (y \wedge b))$   
 $\Rightarrow v \wedge (d \vee y) \wedge (d \vee b)$   
 $\Rightarrow (\text{AND}, v, (\text{OR}, d, y), (\text{OR}, d, b))$
- ②  $m \vee (d \vee (y \wedge b))$   
 $\Rightarrow (m \vee d \vee y) \wedge (m \vee d \vee b)$   
 $\Rightarrow (\text{AND}, (\text{OR}, m, d, y), (\text{OR}, m, d, b))$
- ③  $(e \wedge y) \vee (\neg e \wedge (d \vee y))$   
 $\Rightarrow (e \vee \neg e) \wedge (e \vee d \vee y) \wedge (y \vee \neg e) \wedge (y \vee d \vee y)$

## Exemple: Forme Normale

①  $v \wedge (d \vee (y \wedge b))$

$$\Rightarrow v \wedge (d \vee y) \wedge (d \vee b)$$

$$\Rightarrow (\text{AND}, v, (\text{OR}, d, y), (\text{OR}, d, b))$$

②  $m \vee (d \vee (y \wedge b))$

$$\Rightarrow (m \vee d \vee y) \wedge (m \vee d \vee b)$$

$$\Rightarrow (\text{AND}, (\text{OR}, m, d, y), (\text{OR}, m, d, b))$$

③  $(e \wedge y) \vee (\neg e \wedge (d \vee y))$

$$\Rightarrow (\neg e \vee \neg e) \wedge (e \vee d \vee y) \wedge (y \vee \neg e) \wedge (y \vee d \vee \neg y)$$

## Exemple: Forme Normale

- ➊  $v \wedge (d \vee (y \wedge b))$   
 $\Rightarrow v \wedge (d \vee y) \wedge (d \vee b)$   
 $\Rightarrow (\text{AND}, v, (\text{OR}, d, y), (\text{OR}, d, b))$
- ➋  $m \vee (d \vee (y \wedge b))$   
 $\Rightarrow (m \vee d \vee y) \wedge (m \vee d \vee b)$   
 $\Rightarrow (\text{AND}, (\text{OR}, m, d, y), (\text{OR}, m, d, b))$
- ➌  $(e \wedge y) \vee (\neg e \wedge (d \vee y))$   
 $\Rightarrow (\neg e \vee \neg e) \wedge (e \vee d \vee y) \wedge (y \vee \neg e) \wedge (y \vee d \vee \neg y)$   
 $\Rightarrow (\text{AND}, (\text{OR}, e, d, y), (\text{OR}, y, (\text{NOT}, e)), (\text{OR}, y, d))$

# Inférence par résolution

- ① Base de données  $\mathcal{BD} \leftarrow \{\mathcal{P}\}$
- ② sélectionner deux clauses  $\in \mathcal{BD}$ :

$$p_1 : a_1 \vee \dots \vee a_n \vee \underline{X}$$

$$p_2 : b_1 \vee \dots \vee b_m \vee \neg \underline{X}$$

- ③ règle de résolution  $\Rightarrow Resolvent$

$$q : a_1 \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_m$$

- ④ si  $q \notin \mathcal{BD}$ , ajouter  $q$  à  $\mathcal{BD}$  ;
- ⑤ si  $q$  correspond à une solution, imprimer  $q$  (et arrêter) ;
- ⑥ répéter depuis 2.

# Exemple

Premisses  $\{\mathcal{P}\}$ :

1.  $\neg \textit{lapin} \vee \underline{\textit{animal}}$  ( $= \textit{lapin} \Rightarrow \textit{animal}$ )
2.  $\underline{\neg \textit{animal}} \vee \textit{bouge}$  ( $= \textit{animal} \Rightarrow \textit{bouge}$ )

Résolution:

$$\neg \textit{lapin} \vee \textit{bouge} \quad (= \textit{lapin} \Rightarrow \textit{bouge})$$

# Procédure complète

- L'inférence par résolution est:
  - *fondée*
  - *complète pour la réfutation*
- $\Rightarrow$  procédure complète pour preuves par contradiction:  
$$(\{P\} \cup \neg q) \vdash \perp$$
prouve que  $q$  est une conséquence de  $\{P\}$ .
- $q$  doit être fournie à l'entrée
- La procédure peut ne pas s'arrêter si  $q$  n'est pas une conséquence de  $\{P\}$ .

## Exemple de preuve

Prémisses  $\{P\}$ :

1.  $\neg \text{lapin} \vee \text{animal}$  ( $= \text{lapin} \Rightarrow \text{animal}$ )
2.  $\neg \text{animal} \vee \text{bouge}$  ( $= \text{animal} \Rightarrow \text{bouge}$ )

A prouver:  $q = \neg \text{lapin} \vee \text{bouge}$  ( $= \text{lapin} \Rightarrow \text{bouge}$ )

Transformation en négation  $\Rightarrow \text{lapin} \wedge \neg \text{bouge}$ :

3.  $\text{lapin}$
4.  $\neg \text{bouge}$

Résolutions binaires (1.+2., 4.+5.):

5.  $\neg \text{lapin} \vee \text{bouge}$
6.  $\neg \text{lapin}$

Résolution entre 3. et 6.  $\Rightarrow \perp$  Contradiction

## Un autre exemple...

Prémisses  $\{P\}$ :

1.  $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$
2.  $\neg C \Rightarrow A$

## Un autre exemple...

Prémisses  $\{P\}$ : Sous forme de clauses:

1. $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$  :
  - 1.'  $(A \wedge \neg B) \vee (\neg C \vee D)$
  - 1.''  $A \vee \neg C \vee D$
  - 1.''':  $\neg B \vee \neg C \vee D$
2. $\neg C \Rightarrow A$  :
  2.  $C \vee A$

## Un autre exemple...

Prémisses  $\{P\}$ : Sous forme de clauses:

1. $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$  :
  - 1.'  $(A \wedge \neg B) \vee (\neg C \vee D)$
  - 1.''  $A \vee \neg C \vee D$
  - 1.''':  $\neg B \vee \neg C \vee D$
2. $\neg C \Rightarrow A$  : 2.  $C \vee A$

A prouver:  $q = \neg A \Rightarrow D$  donc  $A \vee D$ :

3.  $\neg A$
4.  $\neg D$

## Un autre exemple...

Prémisses  $\{P\}$ : Sous forme de clauses:

1. $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$  :
  - 1.'  $(A \wedge \neg B) \vee (\neg C \vee D)$
  - 1.''  $A \vee \neg C \vee D$
  - 1.''':  $\neg B \vee \neg C \vee D$
2. $\neg C \Rightarrow A$  : 2.  $C \vee A$

A prouver:  $q = \neg A \Rightarrow D$  donc  $A \vee D$ :

3.  $\neg A$
4.  $\neg D$

Résolutions binaires:

## Un autre exemple...

Prémisses  $\{P\}$ : Sous forme de clauses:

1. $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$  :
  - 1.'  $(A \wedge \neg B) \vee (\neg C \vee D)$
  - 1.''  $A \vee \neg C \vee D$
  - 1.''':  $\neg B \vee \neg C \vee D$
2. $\neg C \Rightarrow A$  : 2.  $C \vee A$

A prouver:  $q = \neg A \Rightarrow D$  donc  $A \vee D$ :

3.  $\neg A$
4.  $\neg D$

Résolutions binaires:

$$1.'' + 3.: 5. \neg C \vee D$$

## Un autre exemple...

Prémisses  $\{P\}$ : Sous forme de clauses:

1. $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$  :
  - 1.'  $(A \wedge \neg B) \vee (\neg C \vee D)$
  - 1.''  $A \vee \neg C \vee D$
  - 1.''':  $\neg B \vee \neg C \vee D$
2. $\neg C \Rightarrow A$  : 2.  $C \vee A$

A prouver:  $q = \neg A \Rightarrow D$  donc  $A \vee D$ :

3.  $\neg A$
4.  $\neg D$

Résolutions binaires:

- 1.'' + 3.: 5.  $\neg C \vee D$
4. + 5.: 6.  $\neg C$

## Un autre exemple...

Prémisses  $\{P\}$ : Sous forme de clauses:

1. $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$  :
  - 1.'  $(A \wedge \neg B) \vee (\neg C \vee D)$
  - 1.''  $A \vee \neg C \vee D$
  - 1.''':  $\neg B \vee \neg C \vee D$
2. $\neg C \Rightarrow A$  : 2.  $C \vee A$

A prouver:  $q = \neg A \Rightarrow D$  donc  $A \vee D$ :

3.  $\neg A$
4.  $\neg D$

Résolutions binaires:

- 1.'' + 3.: 5.  $\neg C \vee D$
4. + 5.: 6.  $\neg C$
2. + 3.: 7.  $C$

## Un autre exemple...

Prémisses  $\{P\}$ : Sous forme de clauses:

1. $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$  :
  - 1.'  $(A \wedge \neg B) \vee (\neg C \vee D)$
  - 1.''  $A \vee \neg C \vee D$
  - 1.''':  $\neg B \vee \neg C \vee D$
2. $\neg C \Rightarrow A$  : 2.  $C \vee A$

A prouver:  $q = \neg A \Rightarrow D$  donc  $A \vee D$ :

3.  $\neg A$
4.  $\neg D$

Résolutions binaires:

- 1.'' + 3.: 5.  $\neg C \vee D$
4. + 5.: 6.  $\neg C$
2. + 3.: 7.  $C$
6. + 7.: Contradiction

# Demandons à ChatGPT (version 3.5, 2024)

*Sur la base des prémisses:*

- ①  $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$
  - ②  $\neg C \Rightarrow A$
- peut-on conclure que:*
- ③  $\neg A \Rightarrow D$

ChatGPT:

Pour résoudre cette question en utilisant les prémisses données, nous pouvons procéder par déduction logique. Commençons par utiliser la deuxième prémissse,  $\neg C \Rightarrow A$ , et appliquons la contraposée. **La contraposée de cette prémissse est  $A \Rightarrow C$ .**

# Demandons à ChatGPT

ChatGPT:

Maintenant, puisque nous avons  $A \Rightarrow C$  et  $(A \Rightarrow B) \Rightarrow (C \Rightarrow D)$ , nous pouvons **conclure**  $A \Rightarrow B$ , **car** si A implique C, et si C implique D, alors **A implique D**.

....

Maintenant, nous avons  $A \Rightarrow B$ , et si A est vrai, alors B doit être vrai. **Puisque B est vrai, et que A implique B, nous avons que A est vrai.**

Finalement, si A est vrai, alors C est vrai (en utilisant la contraposée de la deuxième prémissse).

Si C est vrai, alors D doit être vrai (en utilisant la première prémissse).

**Donc, nous pouvons conclure que  $\neg A \Rightarrow D$  est vrai.**

# Demandons à ChatGPT

Sur la base des prémisses:

*Si, quand c'est le matin il y a cours, alors quand c'est l'après-midi il y a des exercices.*

*Si ce n'est pas l'après-midi, alors c'est le matin.*

peut-on conclure que:

*Si ce n'est pas le matin, alors il y a des exercices.*

ChatGPT (3.5, 2024):

La première prémissse dit: "Si, quand c'est le matin il y a cours, alors quand c'est l'après-midi il y a des exercices." Cela signifie que s'il y a cours le matin, alors il y aura des exercices l'après-midi.

La deuxième prémissse dit: "Si ce n'est pas l'après-midi, alors c'est le matin." Cela signifie que si ce n'est pas l'après-midi, alors c'est le matin.

En combinant ces deux prémisses, si ce n'est pas l'après-midi (c'est-à-dire que c'est le matin), alors il y a cours, ce qui selon la première prémissse signifie qu'il y aura des exercices l'après-midi.

Donc, si ce n'est pas le matin (c'est-à-dire que c'est l'après-midi), alors il y a des exercices, car la première prémissse nous dit que s'il y a cours le matin, il y aura des exercices l'après-midi.

En conclusion, en utilisant ces deux prémisses, on peut déduire que si ce n'est pas le matin, alors il y a des exercices.

# Demandons à ChatGPT

Sur la base des prémisses:

*Si, quand c'est le matin il y a cours, alors quand c'est l'après-midi il y a des exercices.*

*Si ce n'est pas l'après-midi, alors c'est le matin.*

peut-on conclure que:

*Si ce n'est pas le matin, alors il y a des exercices.*

ChatGPT (4o, 2025):

*Traduction des prémisses en logique formelle:*

*Si, quand c'est le matin il y a cours, alors quand c'est l'après-midi il y a des exercices.*

*M : "C'est le matin"    A : "C'est l'après-midi"*

*C : "Il y a cours"       E : "Il y a des exercices"*

**Cette prémissse peut s'écrire sous forme de deux implications:**

$M \Rightarrow C$

$A \Rightarrow E$

*Deuxième prémissse : Si ce n'est pas l'après-midi, alors c'est le matin.*

*Cette prémissse s'écrit :  $\neg A \Rightarrow M$*

*La contraposée de cette implication est :  $\neg M \Rightarrow A$*

**... D'après la première prémissse :  $A \Rightarrow E$  ...**

*En combinant ces deux implications ...nous obtenons, par transitivité :*

$\neg M \Rightarrow E$

# Demandons à ChatGPT

Est-ce que ce raisonnement est correct; Ce n'est pas le matin, alors "Si c'est le matin, alors il y a cours" est toujours vrai. Donc la conclusion de la première prémissse, "Si c'est l'après-midi, alors il y a exercices" est vrai. Par la contraposée de la deuxième prémissse, c'est l'après-midi. Donc il y a exercices.

ChatGPT (3.5, 2024):

Oui, votre raisonnement est correct. Permettez-moi de le reprendre étape par étape pour clarifier :

Première prémissse : "Si c'est le matin, alors il y a cours."

Seconde prémissse : "Si ce n'est pas l'après-midi, alors c'est le matin."

En partant du fait que ce n'est pas le matin, nous pouvons conclure que : "Si c'est le matin, alors il y a cours" est toujours vrai (car la condition n'est pas remplie). Par conséquent, selon la première prémissse, si ce n'est pas le matin, alors il y a exercices. En utilisant la contraposée de la deuxième prémissse, si ce n'est pas le matin, alors c'est l'après-midi. Donc, puisque c'est l'après-midi, selon la première prémissse, il y a exercices. Donc, votre conclusion est correcte : il y a des exercices.

## Moteurs d'inférence pratiques

- La résolution fonctionne pour n'importe quelle proposition logique, mais essayer toutes les combinaisons de clauses est très coûteux.
- On peut souvent limiter les connaissances à:
  - un ensemble de faits  $\mathcal{F}$  qui sont des propositions simples
  - un ensemble de règles  $\mathcal{R}$  de la forme:

$$cond_1 \wedge cond_2 \wedge \dots \Rightarrow consequence$$

- et les solutions à des propositions simples (sans négation).
- $\Rightarrow$  l'inférence par modus ponens.
  - Temps linéaire dans le nombre de règles et propositions simples.

# Clauses de Horn

Clauses de HORN: au plus 1 proposition sans negation:

$$cond_1 \wedge cond_2 \wedge \dots \Rightarrow consequence$$

$$\neg(cond_1 \wedge cond_2 \wedge \dots) \vee consequence$$

$$\neg cond_1 \vee \neg cond_2 \vee \dots \vee consequence$$

Conjonction de conditions + 1 seule conclusion (Règles)  
Peut avoir 0 conditions (Faits)

# Modus Ponens = Résolution sur clauses de Horn

Résolution de clauses de Horn  $\Rightarrow$  clause de Horn:

$$\frac{\begin{array}{c} \neg\dots\vee\neg\dots\vee \quad \neg X \quad \vee Y \\ \neg\dots\vee\neg\dots\vee \quad X \end{array}}{\neg\dots\vee\dots\vee\neg\dots\vee\neg\dots\vee \qquad \qquad Y}$$

Seule contradiction possible entre clauses de Horn:

$X$  et  $\neg X$

$\Rightarrow$  procédure simplifiée

# Résolution sur clauses de Horn = *Modus Ponens*

A

$A \Rightarrow B$

$\vdash B$

Le modus ponens est une application de la résolution:

$$\frac{A \quad \neg A \vee B}{B}$$

## Inférence par Modus Ponens

L'application en chaîne de plusieurs inférences s'appelle *chaînage*:

$$F_1 \xrightarrow{R_1} F_2 \xrightarrow{R_2} \dots \xrightarrow{R_n} \text{Solution}$$

Deux manières d'appliquer:

- chaînage "avant": à partir de tous les faits connus, produire toutes les conséquences jusqu'à ce qu'une solution se trouve parmi les inférences.
- chaînage "arrière": à partir d'une description de la solution recherchée, produire les étapes intermédiaires hypothétiques qui permettront de déduire la solution.

# Chaînage sur clauses de Horn

Procédure complète:  $\{P\} \cup \{\neg q\} \Rightarrow \perp$

$\Rightarrow$  2 procédures complètes sur clauses de Horn:

- Chaînage avant:

Appliquer le modus ponens pour produire  $q$  et ainsi avoir la contradiction.

- Chaînage arrière:

Résolution de  $\neg q$  avec règle ou fait, ensuite application récursive sur les conditions de la règle.

# Traduction en clauses de Horn

Disjonction des conditions:

$$A \vee B \Rightarrow C \simeq$$

$$A \Rightarrow C, B \Rightarrow C$$

Conjonction des conclusions:

$$A \Rightarrow B \wedge C \simeq$$

$$A \Rightarrow B, A \Rightarrow C$$

Disjonction des conclusions:

$$A \Rightarrow B \vee C$$

# Traduction en clauses de Horn

Disjonction des conditions:

$$A \vee B \Rightarrow C \simeq$$

$$A \Rightarrow C, B \Rightarrow C$$

Conjonction des conclusions:

$$A \Rightarrow B \wedge C \simeq$$

$$A \Rightarrow B, A \Rightarrow C$$

Disjonction des conclusions:

$$A \Rightarrow B \vee C$$

pas de traduction possible!

# Spécification du but

Le but sera une conjonction de propositions positives:

$$q = a_1 \wedge a_2 \wedge \dots \wedge a_n$$

et donc  $\neg q$  ne contient aucune proposition positive, et donc est une clause de Horn.

Les propositions de  $q$  peuvent contenir des variables:

oncle(?x, Jacques)

(*intérêt = valeur de la variable ?x*)

Permet l'utilisation de preuves pour faire du calcul.

# Exemples de chaînage

Faits et règles de départ:

- R1:  $\text{vin} \wedge <-2\text{-litres} \Rightarrow \text{petite-quantité}$
- R2.  $\text{cognac} \wedge <-1\text{-litre} \Rightarrow \text{petite-quantité}$
- R3.  $<-100\text{-Frs} \Rightarrow \text{petite-quantité}$
- R4.  $\text{petite-quantité} \wedge \text{adulte} \Rightarrow \text{hors-taxe}$
- F1.  $\text{vin}$
- F2.  $<-2\text{-litres}$
- F3.  $<-100\text{-Frs}$
- F4.  $\text{adulte}$

But:

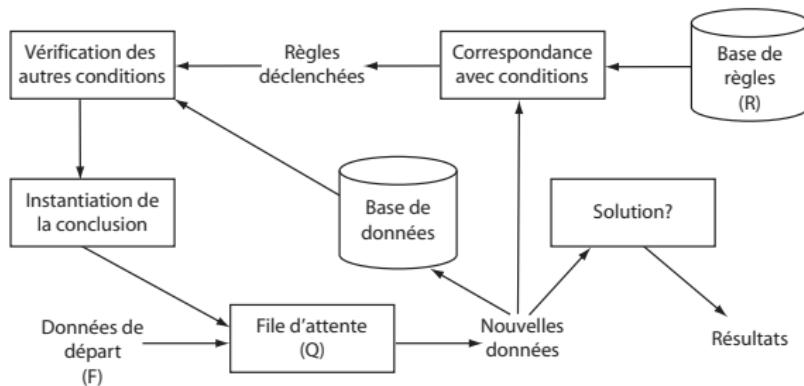
$$q = \text{hors-taxe}$$

# Solution par chaînage avant

- ➊ R1, F1, F2 → F5: petite-quantite
- ➋ R3, F3 → F5: petite-quantite
- ➌ R4, F5, F4 → F6: hors-taxe ⇒ **but**

Pour améliorer l'efficacité:  
limiter les inférences inutiles (e.g., 2.)

# Architecture: chaînage avant



- Base de règles; connaissances
- Base de données: prémisses et inférences
- File d'attente: important pour éviter que des boucles empêchent d'atteindre la solution.

## Algorithme: chaînage avant

Procédure chaînage-simple( $F, R$ )

$Q \leftarrow \{\mathcal{F}\}$

**while**  $Q$  n'est pas vide **do**

$q \leftarrow \text{first}(Q); Q \leftarrow \text{rest}(Q)$

**if**  $q \notin$  base de données **then**

        ajouter  $q$  à la base de données

**if**  $q$  est un noeud but **then** imprimer  $q$

**for**  $r \in \{\mathcal{R}\}$  **do**

**if**  $q \in \text{conditions}(r)$

**and**  $\text{conditions}(r) \subseteq$  base de données **then**

            ajouter  $\text{conclusion}(r)$  à la queue  $Q$

# Propriétés du chaînage avant

- On a déclenché une règle uniquement si on ajoute  $q$  à la base de données
- Le nombre de faits  $n$  est fini (toutes les conclusions des règles)
- $\Rightarrow$  pas plus que  $n$  itérations.
- $\Rightarrow$  l'algorithme s'arrête toujours.
- $\Rightarrow$  temps de calcul linéaire.

# La quantification

Pour exprimer des connaissances générales, les arguments des prédictats peuvent être des *variables*.

Toute variable doit être "définie" par un quantificateur:

- quantification universelle:  $(\forall x)$  = pour toutes les substitutions de la variable  $x$ , l'expression qui suit est vraie.
- quantification existentielle:  $(\exists x)$  = il existe une substitution pour la variable  $x$  qui rend l'expression qui suit vraie.

# Types de quantification

0ème ordre: aucune quantification

vole(Titi)

1er ordre: quantification sur les symboles

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

2ème ordre: quantification sur les symboles et les prédictats

$(\forall p) \text{ p(Charles)} \Rightarrow \text{p(Pierre)}$

Des algorithmes d'inférence existent uniquement pour les 0ème et 1er ordres  $\Rightarrow$  2ème ordre non utilisé en IA.

# Exemples (1)

① “Tous mes livres sont intéressants.”

$$(\forall x) [(\text{livre}(x) \wedge \text{poss\`ede}(\text{moi}, x)) \Rightarrow \text{intéressant}(x)]$$

② “Je n'ai aucun livre mince.”

$$\neg (\exists x) [\text{livre}(x) \wedge \text{poss\`ede}(\text{moi}, x) \wedge \text{mince}(x)]$$

## Exemples (2)

① "Tout le monde parle une langue."

- a.  $(\forall x) \text{ personne}(x) \Rightarrow (\exists y) (\text{langue}(y) \wedge \text{parle}(x,y))$
- b.  $(\exists y) \text{ langue}(y) \wedge (\forall x) (\text{personne}(x) \Rightarrow \text{parle}(x,y))$

② "Il y a exactement une langue que tout le monde parle."

- $$(\exists y) \text{ langue}(y) \wedge [(\forall x) (\text{personne}(x) \Rightarrow \text{parle}(x,y))] \wedge (\forall z) [\text{langue}(z) \wedge (\forall x) (\text{personne}(x) \Rightarrow \text{parle}(x,z))] \Rightarrow (z=y)$$

# Equivalence d'expressions quantifiées

$$(\forall x)P(x) \Leftrightarrow (\forall y)P(y)$$

$$(\exists x)P(x) \Leftrightarrow (\exists y)P(y)$$

$$\neg(\exists x)P(x) \Leftrightarrow (\forall x)[\neg P(x)]$$

$$\neg(\forall x)P(x) \Leftrightarrow (\exists x)[\neg P(x)]$$

$$(\forall x)[P(x) \wedge Q(x)] \Leftrightarrow (\forall x)P(x) \wedge (\forall y)Q(y)$$

$$(\exists x)[P(x) \vee Q(x)] \Leftrightarrow (\exists x)P(x) \vee (\exists y)Q(y)$$

## Les fonctions de Skolem

- Les quantificateurs gênent les procédures d'inférence.
- Si  $(\exists x)p(x)$ :  $p(A)$  doit être vrai pour un certain  $A$
- Considérons le schéma suivant:  
 $(\forall x) [(\exists y) p(x,y)]$   
= "pour tout  $x$ , il existe un  $y$  tel que  $p(x,y)$ "
- On peut remplacer  $y$  par la fonction de Skolem  
 $f(x)$  retournant toujours un  $y$  qui remplit la condition  $p(x,y)$ :  
 $(\forall x) p(x,f(x))$
- Arguments = toutes les variables avec quantification universelles dont la portée inclut la variable remplacée.

# La forme normale

- Remplacer variables avec quantifications existentielles par des fonctions de Skolem.  
⇒ élimination des quantificateurs existentiels
- Pour les variables qui restent:  
quantification universelle sous-entendue
- Notation pour distinguer variables/constantes:
  - minuscules/majuscules: FRERE(x,CHARLES)
  - point d'interrogation: frere(?x,Charles)

## Exemple (forme normale)

$(\forall x) \text{ personne}(x) \Rightarrow (\exists y) (\text{langue}(y) \wedge \text{parle}(x,y))$

- ① Eliminer les quantificateurs existentiels:

$(\forall x) \text{ personne}(x) \Rightarrow$   
 $(\text{langue}(l(x)) \wedge \text{parle}(x,l(x)))$

- ② Laisser tomber les quantificateurs universels:

$\text{personne}(\text{?}x) \Rightarrow \text{langue}(l(\text{?}x)) \wedge \text{parle}(\text{?}x,l(\text{?}x))$

Identifier les variables par un "?"

# Résolution avec variables

Unification:

$U(X, Y) = \text{substitution de variables pour rendre } U(X) = U(Y).$

Pour adapter la résolution à des expressions quantifiées, il faut les deux règles d'inférence:

① résolution binaire:

$$\begin{aligned} & (L_1 \vee A), (\neg L_2 \vee B), \text{Unify}(L_1, L_2) = U \\ & \Rightarrow (U(A) \vee U(B)) \end{aligned}$$

② factorisation (nouveau):

$$\begin{aligned} & (L_1 \vee L_2 \vee A), \text{Unify}(L_1, L_2) = U \\ & \Rightarrow (U(L_2) \vee U(A)) \end{aligned}$$

## Résolution avec variables (exemple)

Prémisses  $\{\mathcal{P}\}$ :

$$\begin{aligned}(\forall x)a(x) \vee b(x) : & \quad 1. \quad a(?x) \vee b(?x) \\(\forall x)a(x) \Rightarrow c(x) : & \quad 2. \quad \neg a(?y) \vee c(?y) \\(\forall x)c(x) \Rightarrow d(x) : & \quad 3. \quad \neg c(?z) \vee d(?z) \\(\forall x)b(x) \Rightarrow d(x) : & \quad 4. \quad \neg b(?w) \vee d(?w)\end{aligned}$$

A prouver:  $d(?x)$ : 5.  $\neg d(?x)$

$$\begin{aligned}1. + 2. \Rightarrow 6. \quad & b(?x) \vee c(?x) \\3. + 6. \Rightarrow 7. \quad & d(?z) \vee b(?z) \\4. + 7. \Rightarrow 8. \quad & d(?w) \vee d(?w)\end{aligned}$$

Factorisation de 8.: 9.  $d(?w)$

Résolution binaire entre 5. + 9.  $\Rightarrow$  Contradiction

# Filtrage et unification

Problème du filtrage (pattern match):  
étant donné:

- l'expression pattern avec variables
- l'expression datum sans variables

trouver toutes les combinaisons de substitutions des variables de pattern pour rendre datum = pattern.

```
filtrer((regarde, Paul, Pierre), (regarde, ?x, ?y))  
(?x : Paul, ?y : Pierre)
```

Dans l'unification, les deux expressions peuvent contenir des variables:

```
unifier((regarde, ?x, Pierre), (regarde, Paul, ?y))  
(?x : Paul, ?y : Pierre)
```

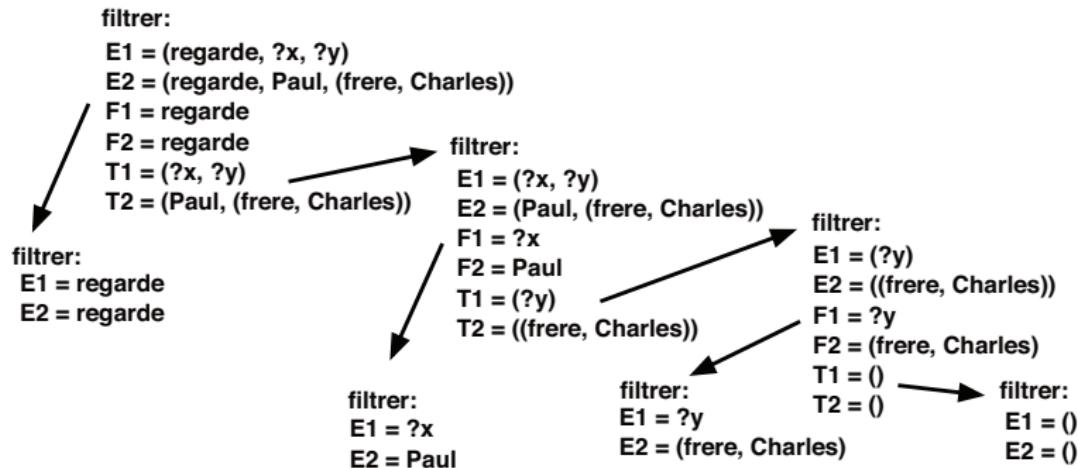
Algorithme central de tout système à base de connaissances.

# Algorithme de filtrage

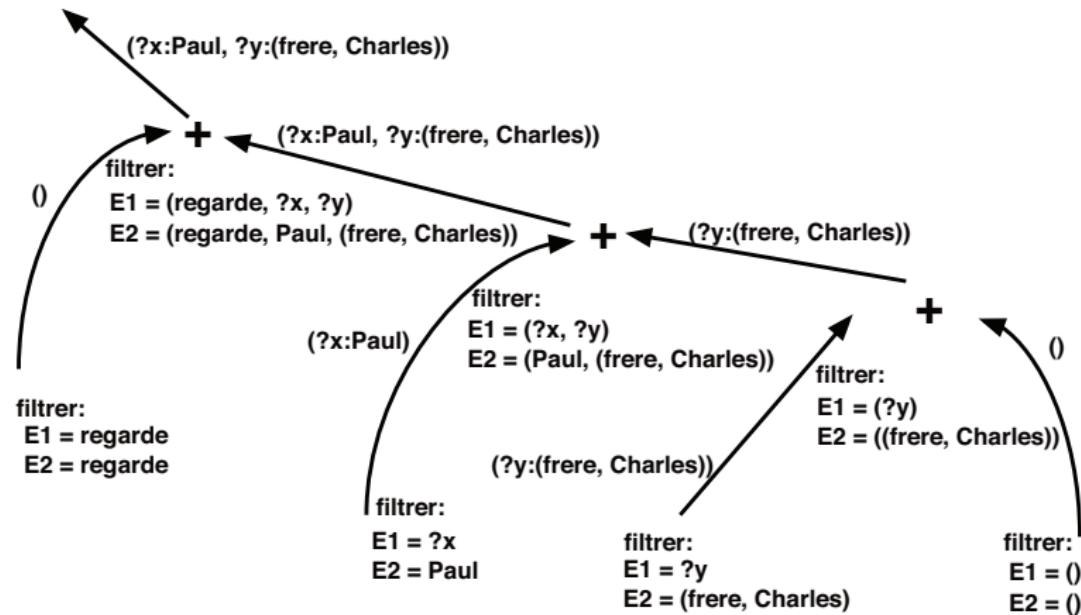
```
Function Filtrer(pattern,datum)
if pattern est un symbol then
    if datum et pattern sont identiques then return {}
    if pattern est une variable then return {pattern/datum}
    return ECHEC
if datum est un symbol then return ECHEC
F1 ← premier élément de pattern, T1 ← reste de pattern
F2 ← premier élément de datum, T2 ← reste de datum
Z1 ← FILTRER(F1,F2)
if Z1 = ECHEC then return ECHEC
G1 ← remplacer les variables de T1 par les unifications Z1
G2 ← T2
Z2 ← FILTRER(G1,G2)
if Z2 = ECHEC then return ECHEC
return { Z1 ∪ Z2 }
```

# Exemple de filtrage

Appels récursifs:



# Retour des appels



# Algorithme d'unification

Function Unifier( $E_1, E_2$ )

**if**  $E_1$  ou  $E_2$  est un symbole **then**

  Interchanger les arguments de  $E_1$  et de  $E_2$  (si nécessaire) de sorte que  $E_1$  soit un symbole

**if**  $E_1$  et  $E_2$  sont identiques **then return** {}

**if**  $E_1$  est une variable **then**

**if**  $E_1$  apparaît dans  $E_2$  **then return** ECHEC

**return** { $E_1 : E_2$ }

**if**  $E_2$  est une variable **then return** { $E_2 : E_1$ }

**return** ECHEC

$F_1 \leftarrow$  premier élément de  $E_1$ ,  $T_1 \leftarrow$  reste de  $E_1$

$F_2 \leftarrow$  premier élément de  $E_2$ ,  $T_2 \leftarrow$  reste de  $E_2$

$Z_1 \leftarrow$  UNIFIER( $F_1, F_2$ )

**if**  $Z_1 =$  ECHEC **then return** ECHEC

$G_1 \leftarrow$  remplacer les variables de  $T_1$  par les substitutions  $Z_1$

$G_2 \leftarrow$  remplacer les variables de  $T_2$  par les substitutions  $Z_1$

$Z_2 \leftarrow$  UNIFIER( $G_1, G_2$ )

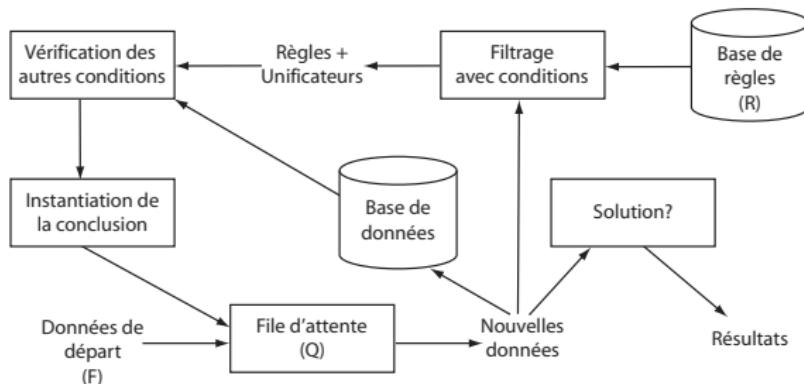
**if**  $Z_2 =$  ECHEC **then return** ECHEC

**return** {  $Z_1 \cup Z_2$  }

# Intégration avec un système de règles

- Possibilité de plusieurs unificateurs:  
*différents éléments de la base de données peuvent s'unifier avec la même règle*
- Procédure RETE pour optimiser l'unification de plusieurs faits et règles.
- Noms de variables uniques

# Architecture: chaînage avant avec variables



Différence par rapport au moteur simple: le filtrage/unification des différentes conditions doit être consistant!

## Exemple d'un chaînage avec variables

Faits et règles de départ:

- $R1: \text{père}(\text{?x}, \text{?y}) \wedge \text{frère}(\text{?y}, \text{?z}) \Rightarrow \text{père}(\text{?x}, \text{?z})$
- $R2: \text{père}(\text{?x}, \text{?y}) \wedge \text{frère}(\text{?x}, \text{?z}) \Rightarrow \text{oncle}(\text{?z}, \text{?y})$
- $F1: \text{père}(\text{Jacques}, \text{Charles}),$
- $F2: \text{frère}(\text{Charles}, \text{Francois}),$
- $F3: \text{frère}(\text{Jacques}, \text{Pierre})$

But:

$\text{oncle}(\text{?x}, \text{Francois})$

## Solution par chaînage avant

$R1: \text{père}(\text{?x}, \text{?y}) \wedge \text{frère}(\text{?y}, \text{?z}) \Rightarrow \text{père}(\text{?x}, \text{?z})$   
 $R2: \text{père}(\text{?x}, \text{?y}) \wedge \text{frère}(\text{?x}, \text{?z}) \Rightarrow \text{oncle}(\text{?z}, \text{?y})$   
 $F1: \text{père}(\text{Jacques}, \text{Charles}),$   
 $F2: \text{frère}(\text{Charles}, \text{Francois}),$   
 $F3: \text{frère}(\text{Jacques}, \text{Pierre})$

## Solution par chaînage avant

$R1: \text{père}(\text{x}, \text{y}) \wedge \text{frère}(\text{y}, \text{z}) \Rightarrow \text{père}(\text{x}, \text{z})$   
 $R2: \text{père}(\text{x}, \text{y}) \wedge \text{frère}(\text{x}, \text{z}) \Rightarrow \text{oncle}(\text{z}, \text{y})$   
 $F1: \text{père}(\text{Jacques}, \text{Charles}),$   
 $F2: \text{frère}(\text{Charles}, \text{Francois}),$   
 $F3: \text{frère}(\text{Jacques}, \text{Pierre})$

- ①  $R1, F1, F2 \rightarrow F4: \text{père}(\text{Jacques}, \text{Francois}))$

## Solution par chaînage avant

$R1: \text{père}(\text{x}, \text{y}) \wedge \text{frère}(\text{y}, \text{z}) \Rightarrow \text{père}(\text{x}, \text{z})$   
 $R2: \text{père}(\text{x}, \text{y}) \wedge \text{frère}(\text{x}, \text{z}) \Rightarrow \text{oncle}(\text{z}, \text{y})$   
 $F1: \text{père}(\text{Jacques}, \text{Charles}),$   
 $F2: \text{frère}(\text{Charles}, \text{Francois}),$   
 $F3: \text{frère}(\text{Jacques}, \text{Pierre})$

- ①  $R1, F1, F2 \rightarrow F4: \text{père}(\text{Jacques}, \text{Francois}))$
- ②  $R2, F1, F3 \rightarrow F5: \text{oncle}(\text{Pierre}, \text{Charles})$

## Solution par chaînage avant

$R1: \text{père}(\text{x}, \text{y}) \wedge \text{frère}(\text{y}, \text{z}) \Rightarrow \text{père}(\text{x}, \text{z})$   
 $R2: \text{père}(\text{x}, \text{y}) \wedge \text{frère}(\text{x}, \text{z}) \Rightarrow \text{oncle}(\text{z}, \text{y})$   
 $F1: \text{père}(\text{Jacques}, \text{Charles}),$   
 $F2: \text{frère}(\text{Charles}, \text{Francois}),$   
 $F3: \text{frère}(\text{Jacques}, \text{Pierre})$

- ➊  $R1, F1, F2 \rightarrow F4: \text{père}(\text{Jacques}, \text{Francois})$
- ➋  $R2, F1, F3 \rightarrow F5: \text{oncle}(\text{Pierre}, \text{Charles})$
- ➌  $R2, F4, F3 \rightarrow F6: \text{oncle}(\text{Pierre}, \text{Francois}) \Rightarrow \text{but}$

## Application: Business rules/Thames Water

- Thames Water fournit l'eau à la région de Londres
- 250'000 demandes de changement de raccordement/année
- Difficile à automatiser à cause de la complexité des règlements

# Asset Data Services Decision Making Engine

Système de règles (Business Rules):

- remplace une équipe de 30 personnes
- diminué le temps de réponse de 50%
- économise 3 millions de livres par an en coûts
- évite 1.4 millions de livres d'erreurs par an

Application typique de business rules

# Resumé

- Inférence par résolution
- Inférence par chaînage
- Inférence avec variables
- Filtrage et Unification