Système DUCE Muggleton

- **Généralisation de ID3**

On de donne

- **Ensemble d'apprentissage:**
  - $E_i \Rightarrow cl(E_i)$
  - $cl(E_i) \in \{C_1, C_2, \ldots, C_k\}$ ce sont les classes
  - E_i ce sont les exemples d'apprentissage

On cherche à

- **Construire les définitions des classes $\{C_1, C_2, \ldots, C_k\}$**
  - $D_{i,j} \Rightarrow C_j \quad \forall j \in [1, k]$
- **Minimiser l'information (équivalent entropie)**

Exemple

Chaussures	Cheveux	Sourire	Boucles	Classe
Rouge	Blonds	Oui	Non	G
Rouge	Blonds	Oui	Oui	G
Marron	Blonds	Oui	Non	T
Noir	Châtains	Oui	Non	T
Noir	Foncés	Non	Non	T
Noir	Foncés	Non	Oui	G
Marron	Foncés	Non	Oui	G

Exemple – représentation DUCE

Chaussures	Cheveux	Sourire	Boucles	Classe
Rouge	Blonds	Oui	Non	G
Rouge	Blonds	Oui	Oui	G
Marron	Blonds	Oui	Non	T
Noir	Châtains	Oui	Non	T
Noir	Foncés	Non	Non	T
Noir	Foncés	Non	Oui	G
Marron	Foncés	Non	Oui	G

Rouge & Blonds & S=Oui & B=Non ⇒ G

Rouge & Blonds & S=Oui & B=Oui ⇒ G

Marron & Blonds & S=Oui & B=Non ⇒ T

Noir & Châtains & S=Oui & B=Non ⇒ T

Noir & Foncés & S=Non & B=Non ⇒ T

Noir & Fondés & S=Non & B=Oui ⇒ G

Marrons & Foncés & S=Non & B=Oui ⇒ G

DUCE: opérateurs V

<u>Absorbtion</u>: a & b ⇒ h1 et a ⇒ h2 donnent h2 & b ⇒ h1 et a ⇒ h2

triangle & rouge ⇒ plus et triangle ⇒ polygone donnent

polygone & rouge ⇒ plus et triangle ⇒ polygone

<u>Troncature</u>: a & b ⇒ h1 et a & d ⇒ h1 donnent a ⇒ h1

triangle & rouge ⇒ plus et triangle & vert ⇒ plus donnent

triangle ⇒ plus

<u>Identification</u>: a & b ⇒ h1 et b & h2 ⇒ h1 donnent

b & h2 ⇒ h1 et a ⇒ h2

triangle & vert ⇒ plus et polygone & vert ⇒ plus donnent

polygone & vert ⇒ plus et triangle ⇒ polygone.

DUCE: opérateurs W

Dichotomisation: a & b ⇒ h1, a & c ⇒ ¬h1 et a & d ⇒ ¬h1
donnent a & h2 ⇒ h1, a & ¬h2 ⇒ ¬h1, b ⇒ h2, c ⇒ ¬h2 et d ⇒ ¬h2

rouge & losange ⇒ plus, rouge & triangle ⇒ ¬plus et
rouge & cercle ⇒ ¬plus donnent
rouge & quadrilatère ⇒ plus, rouge & ¬quadrilatère ⇒ ¬plus,
losange ⇒ quadrilatère, triangle ⇒ ¬quadrilatère et
cercle ⇒ ¬quadrilatère.

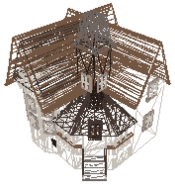
DUCE: opérateurs W (suite)

<u>Inter-construction</u>: a & b ⇒ h1 et a & c ⇒ h1 donnent h2 & b ⇒ h1, h2 & c ⇒ h1 et a ⇒ h2.

rouge & carré ⇒ plus et vert & carré ⇒ plus <u>donnent</u> quadrilatère & rouge ⇒ plus, quadrilatère & vert ⇒ plus et carré ⇒ quadrilatère.

<u>Intra-construction</u>: a & b ⇒ h1 et a & c ⇒ h1 donnent h2 & a ⇒ h1, b ⇒ h2 et c ⇒ h2.

rouge & carré ⇒ plus, rouge & triangle ⇒ plus <u>donnent</u> polygone & rouge ⇒ plus, carré ⇒ polygone et triangle ⇒ polygone.



Exemple

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

Exemple

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

- **Troncature: a & b → h1 et a & d → h1 donnent a → h1**

Marron & B=Non ⇒ T (règle 3) Noir & B=Oui ⇒ G (règle 6)

Noir & B=Non ⇒ T (règles 4, 5) Marron & B=Oui ⇒ G (règle 7)

Exemple

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

Marron & B=Non ⇒ T
Noir & B=Non ⇒ T
Noir & B=Oui ⇒ G
Marron & B=Oui ⇒ G

Exemple

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

Marron & B=Non ⇒ T
Noir & B=Non ⇒ T
Noir & B=Oui ⇒ G
Marron & B=Oui ⇒ G

- **Intra-construction:**

a & b → h1 et a & c → h1 donnent

h2 & a → h1, b → h2 et c → h2.

Marron ⇒ Terne

Noir ⇒ Terne

Terne & B=Non ⇒ T

Exemple *(suite)*

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

Marron & B=Non ⇒ T
Noir & B=Non ⇒ T
Marron ⇒ terne
Noir ⇒ terne
Terne & B=Non ⇒ T

Noir & B=Oui ⇒ G
Marron & B=Oui ⇒ G

Exemple *(suite)*

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

Marron & B=Non ⇒ T
Noir & B=Non ⇒ T
Marron ⇒ terne
Noir ⇒ terne
Terne & B=Non ⇒ T

Noir & B=Oui ⇒ G
Marron & B=Oui ⇒ G

Troncature:
a & b → h1 et a & d → h1 donnent **a → h1**

Rouge & Blonds & S=Oui ⇒ G

Exemple *(suite)*

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

Marron & B=Non ⇒ T
Noir & B=Non ⇒ T
Marron ⇒ terne
Noir ⇒ terne
Terne & B=Non ⇒ T

Noir & B=Oui ⇒ G
Marron & B=Oui ⇒ G
Rouge & Blonds & S=Oui ⇒ G

Exemple *(suite)*

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

Marron & B=Non ⇒ T
Noir & B=Non ⇒ T
Marron ⇒ terne
Noir ⇒ terne
Terne & B=Non ⇒ T

Noir & B=Oui ⇒ G
Marron & B=Oui ⇒ G
Rouge & Blonds & S=Oui ⇒ G

Absorbtion:
a & b → h1 et a → h2 donnent **h2 & b → h1 et a → h2**
Terne & B=Oui ⇒ G

Exemple *(suite)*

Rouge & Blonds & S=Oui & B=Non ⇒ G
Rouge & Blonds & S=Oui & B=Oui ⇒ G
Marron & Blonds & S=Oui & B=Non ⇒ T
Noir & Châtains & S=Oui & B=Non ⇒ T
Noir & Foncés & S=Non & B=Non ⇒ T
Noir & Fondés & S=Non & B=Oui ⇒ G
Marrons & Foncés & S=Non & B=Oui ⇒ G

Marron & B=Non ⇒ T
Noir & B=Non ⇒ T
Marron ⇒ terne
Noir ⇒ terne
Terne & B=Non ⇒ T

Noir & B=Oui ⇒ G
Marron & B=Oui ⇒ G
Rouge & Blonds & S=Oui ⇒ G
Terne & B=Oui ⇒ G

Marron ⇒ terne
Noir ⇒ terne
Terne & B=Non ⇒ T

Rouge & Blonds & S=Oui ⇒ G
Terne & B=Oui ⇒ G

Résolution en logique des propositions

- **soit A et B deux clauses,**
- **soit L un atome tel que $L \in A$ et $\neg L \in B$,**

on peut alors construire le résolvant de A et B parL, que l'on note:

$Rés(A, B; L) = (A - \{L\}) \cup (B - \{\neg L\})$

Notation: $A, B \vdash_{Rès} Rés(A, B; L)$

<u>Exemples</u>:

Rés(homme, $\neg$homme$\vee$mortel) = mortel *(c'est l'équivalent du modus ponens)*

Rés($\neg$mortel, $\neg$homme$\vee$mortel) = $\neg$homme *(c'est l'équivalent du modus tolens)*

<u>Théorème de résolution:</u>

S insatisfiable si et seulement si $A, B \vdash_{Rès} \square$(clause vide)

DUCE: opérateurs V

<u>Absorbtion</u>: a & b ⇒ h1 et a ⇒ h2 donnent h2 & b ⇒ h1 et a ⇒ h2

triangle & rouge ⇒ plus et triangle ⇒ polygone donnent

polygone & rouge ⇒ plus et triangle ⇒ polygone

<u>Troncature</u>: a & b ⇒ h1 et a & d ⇒ h1 donnent a ⇒ h1

triangle & rouge ⇒ plus et triangle & vert ⇒ plus donnent

triangle ⇒ plus

<u>Identification</u>: a & b ⇒ h1 et b & h2 ⇒ h1 donnent

b & h2 ⇒ h1 et a ⇒ h2

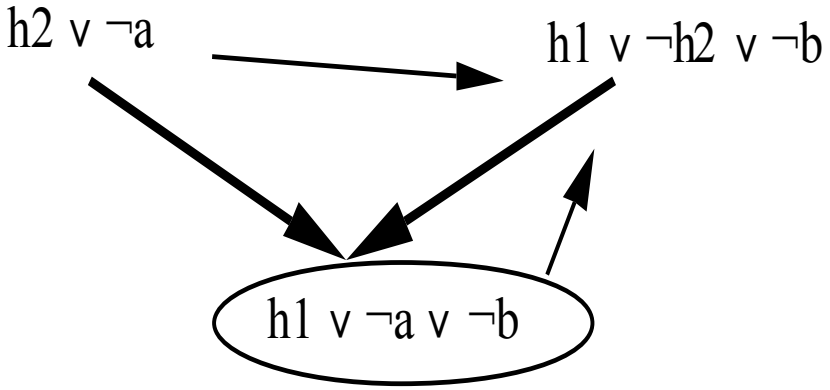
triangle & vert ⇒ plus et polygone & vert ⇒ plus donnent

polygone & vert ⇒ plus et triangle ⇒ polygone.

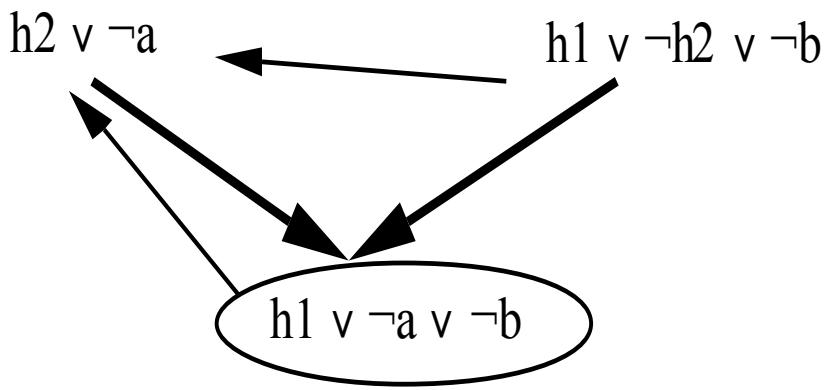
L I P 6 C N R S

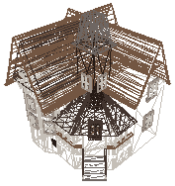
DUCE
Opérateurs V

Absorption



Identification





L I P 6 C N R S

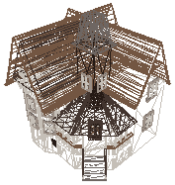
DUCE: opérateurs W

<u>Inter-construction</u>: a & b ⇒ h1 et a & c ⇒ h1 donnent h2 & b ⇒ h1, h2 & c ⇒ h1 et a ⇒ h2.

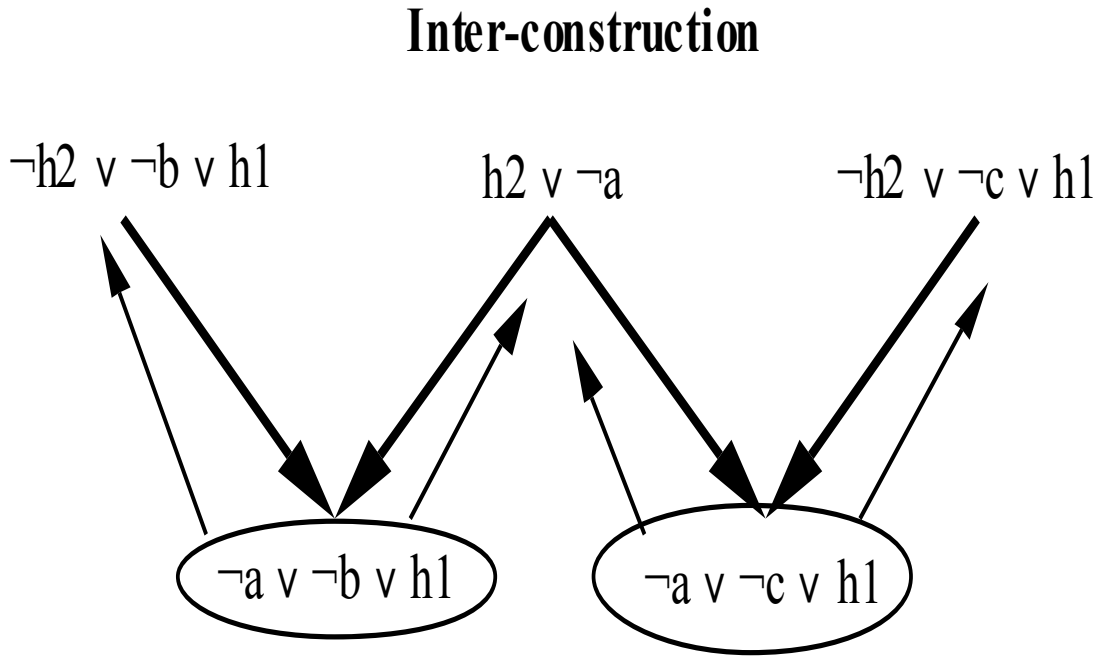
rouge & carré ⇒ plus et vert & carré ⇒ plus <u>donnent</u> quadrilatère & rouge ⇒ plus, quadrilatère & vert ⇒ plus et carré ⇒ quadrilatère.

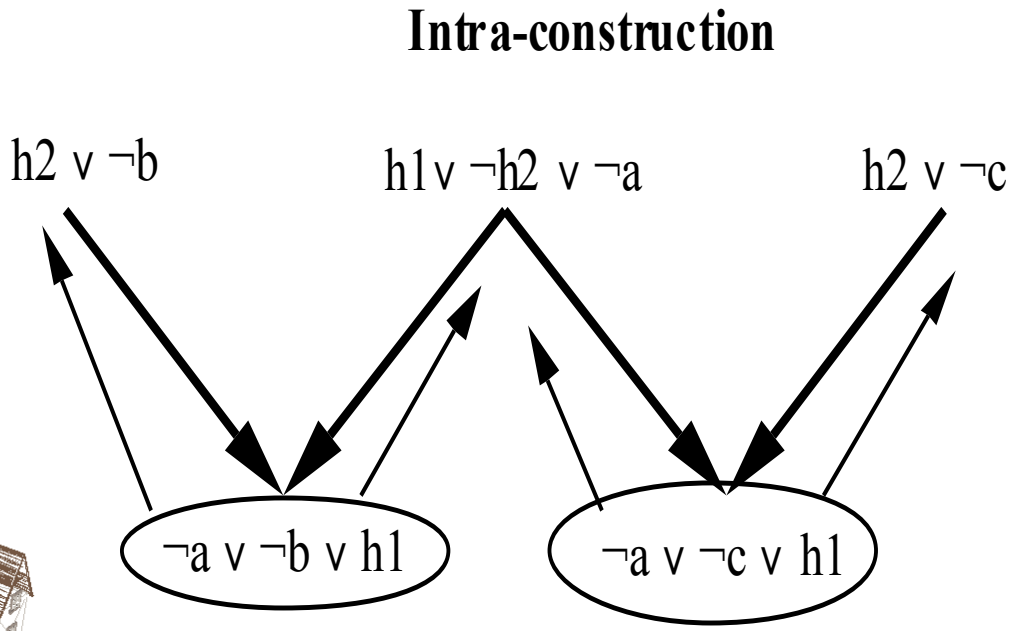
<u>Intra-construction</u>: a & b ⇒ h1 et a & c ⇒ h1 donnent h2 & a ⇒ h1, b ⇒ h2 et c ⇒ h2.

rouge & carré ⇒ plus, rouge & triangle ⇒ plus <u>donnent</u> polygone & rouge ⇒ plus, carré ⇒ polygone et triangle ⇒ polygone.

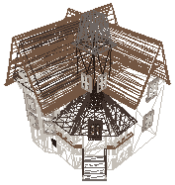


L I P 6 C N R S

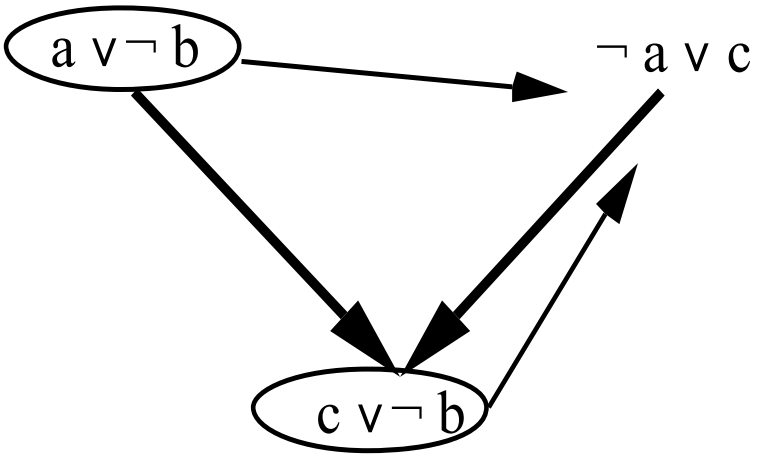




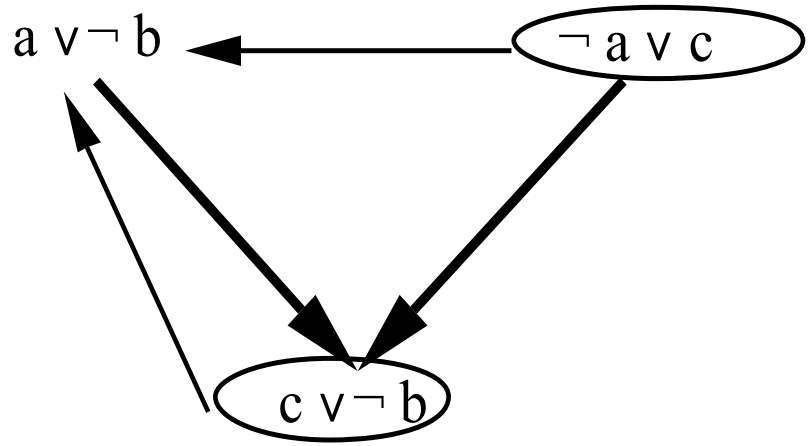
Duce opérateurs W



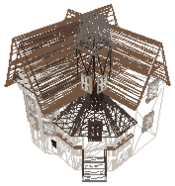
Inversion de la résolution



Inversion de la résolution : 1



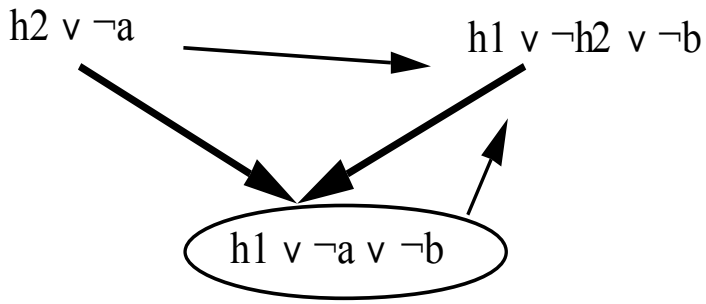
Inversion de la résolution : 2



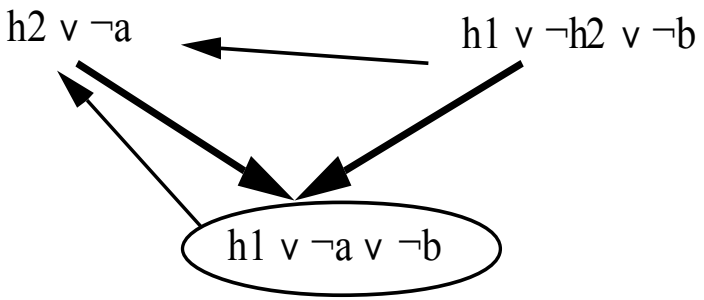
L I P 6 C N R S

DUCE

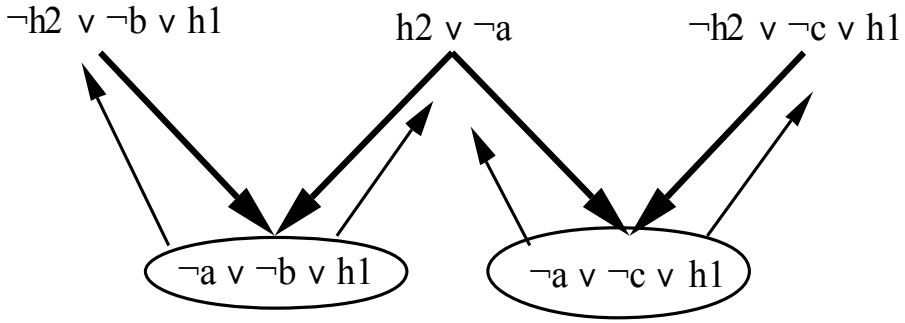
Absorption



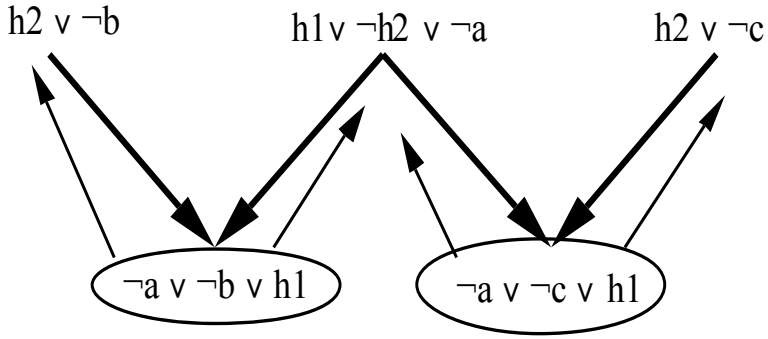
Identification

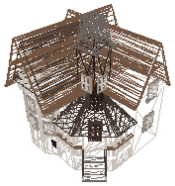


Inter-construction

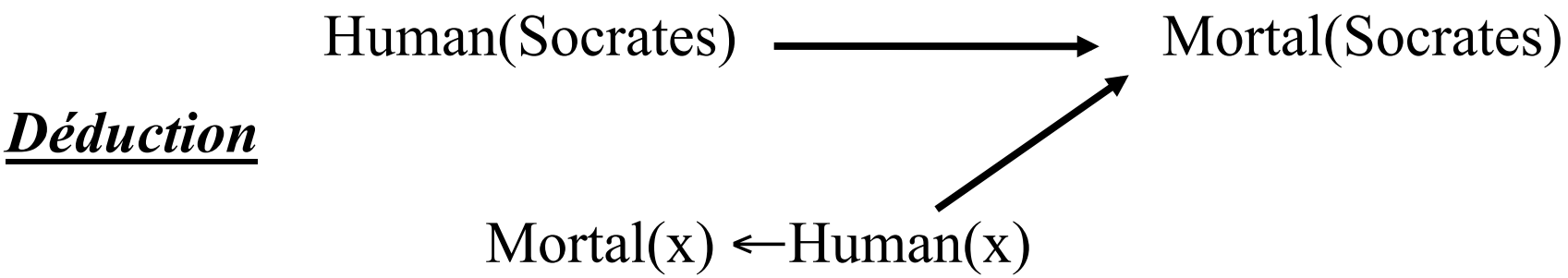


Intra-construction

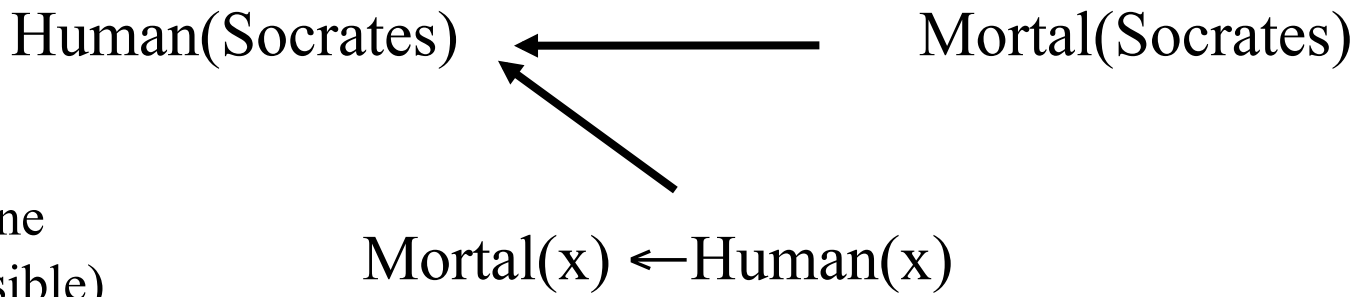




Déduction, Abduction *rappels*



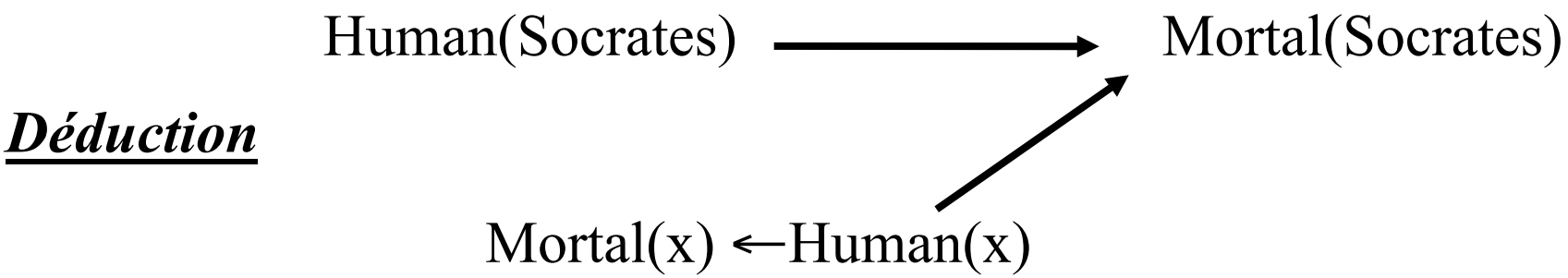
Déduction



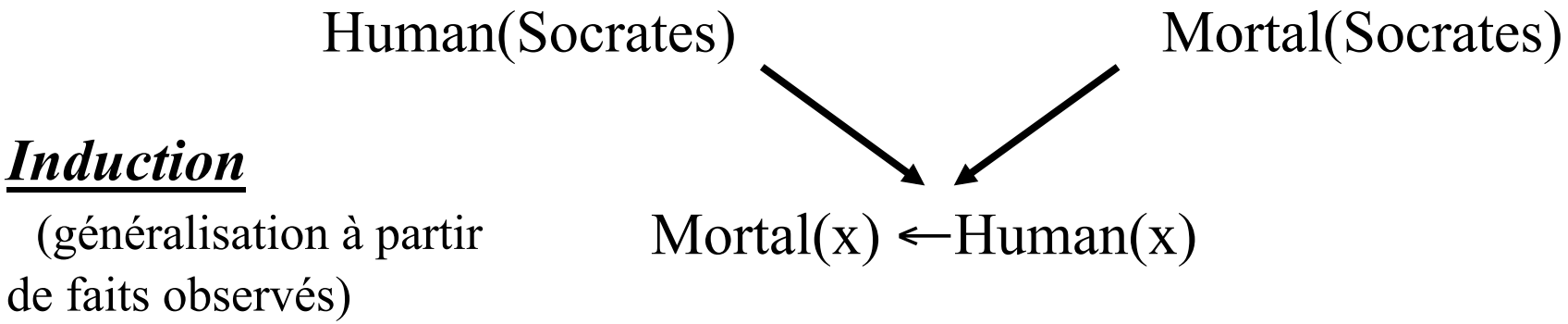
Abduction

(recherche d'une explication plausible)

Déduction, Induction *rappels*



Déduction



Induction

(généralisation à partir de faits observés)

Résolution générale

Définition:

- **soit A et B deux clauses dont les variables sont disjointes (au besoin, renommer les variables de l'une d'entre elles),**
- **soit L_1 un littéral de A et L_2 un littéral de B tels que soit L_1 et $\neg L_2$ s'unifient, soit $\neg L_1$ et L_2 s'unifient avec la substitution σ.**

On dira que C est le résolvant de A et B par L_1 et L_2**, ce que l'on notera:**

$C = Rés(A, B; L_1, L_2) = (A\sigma - \{L_1\sigma\}) \cup (B\sigma - \{L_2\sigma\})$

Exemples:

Rés(**R(x, y)** , **¬R(u, v)** ; **R(x, y), ¬R(u, v)) =∅**

Rés(**P(x)** ∨ **P(y), ¬P(z)** ∨ **¬P(r); P(x), ¬P(z)) = P(y)** ∨ **¬P(r)**

Rés(**homme(Socrate)**, **¬homme(z)** ∨ **mortel(z);**
homme(Socrate), ¬homme(z) = mortel(Socrate)

Résolution : exemple

C_1: vénéneux(X) ∨ ¬champignon(X) ∨ ¬non-comestible(X)

C_2: toxique(U) ∨ ¬vénéneux(U)

C_3: ¬toxique(a)

Rés(**C_1, C_2 ; vénéneux(X) , ¬vénéneux(U)) =**

Rés(vénéneux(X) ∨ **¬champignon(X) ∨ ¬non-comestible(X),**
toxique(U) ∨ ¬vénéneux(U)
; vénéneux(X) , ¬vénéneux(U)) =

¬champignon(X) ∨ ¬non-comestible(X) ∨ toxique(X)

C_1 **signifie** champignon(X) ∧ non-comestible(X) ⇒ vénéneux(X)

C_2 **signifie** vénéneux(U) ⇒ toxique(U)

La résolvante de C_1 et C_2 signifie
champignon(X) ∧ non-comestible(X) ⇒ toxique(X)

Résolution : suite de l'exemple

C_1 : vénéneux(X) ∨ ¬champignon(X) ∨ ¬non-comestible(X)

C_2 : toxique(U) ∨ ¬vénéneux(U)

C_3 : ¬toxique(a)

Rés(**C_2, C_3 ; toxique(U) , ¬toxique(a)**) =

Rés(toxique(U) ∨ **¬vénéneux(U)**, ¬toxique(a) ,
; toxique(U) , ¬toxique(a)) = **¬vénéneux(a),**

C_2 **signifie** vénéneux(U) ⇒ toxique(U), **ce qui est équivalent à** ¬toxique(U) ⇒ ¬vénéneux(U),

C_3 **signifie** ¬toxique(a) **(autrement dit a n'est pas toxique)**

La résolvante de C_2 et C_3 signifie ¬vénéneux(a) **autrement dit que a n'est pas vénéneux.**

Facteur

Définition:

- soit A une clause

- soit L1 et L2 deux littéraux unifiables de A dont le pgu est σ

Alors A σ -{L1 σ} est un *facteur* de A.

Exemple: **soit A = P(x, y) $\vee$ P(x, x) $\vee$ P(y, y),**
L1 = P(x, x), L2 = P(y, y),
P(z, z) est un *facteur* de A.

R-Déduction et R-réfutation

Définition:

- **soit S un ensemble de clauses,**
- **soit A une clause,**

on a une *R-déduction* **de S vers A, noté S $|\!\!-_{Rès}$ A,**
si et seulement si ∃ B1, B2, ... Bn tels que:

- Bn = A

- ∀ i ∈ [1, n] 1- Bi ∈ S ou

2- ∃ j < i, ∃ k < i, ∃ L1 ∈ Bj, ∃ L2 ∈ Bk,
Bi = Rès(Bj, Bk; L1, L2) ou

3- ∃ j < i tel que Bi et un facteur de Bj**.**

Définition**: une** *R-réfutation* **d'un ensemble de clauses S est une R-déduction de S vers la clause vide.**

R-réfutation : Exemple

***Exemple*: S = {C1, C2, C3, C4, C5, C6}**

C1: vénéneux(X) ∨ ¬champignon(X) ∨ ¬non-comestible(X)

C2: toxique(X) ∨ ¬vénéneux(X)

C3: champignon(X) ∨ ¬amanite-phalloïde(X)

C4: non-comestible(X) ∨ ¬amanite-phalloïde(X)

C5: amanite-phalloïde(a)

C6: ¬toxique(a)

C7:	**champignon(a)**	**[Rès. C5, C3]**
C8:	**vénéneux(a) ∨ ¬non-comestible(a)**	**[Rès. C1, C7]**
C9:	**toxique(a) ∨ ¬non-comestible(a)**	**[Rès. C2, C8]**
C10:	**toxique(a) ∨ ¬amanite-phalloïde(a)**	**[Rès. C4, C9]**
C11:	**toxique(a)**	**[Rès. C5, C10]**
C12:		**[Rès. C11, C6]**

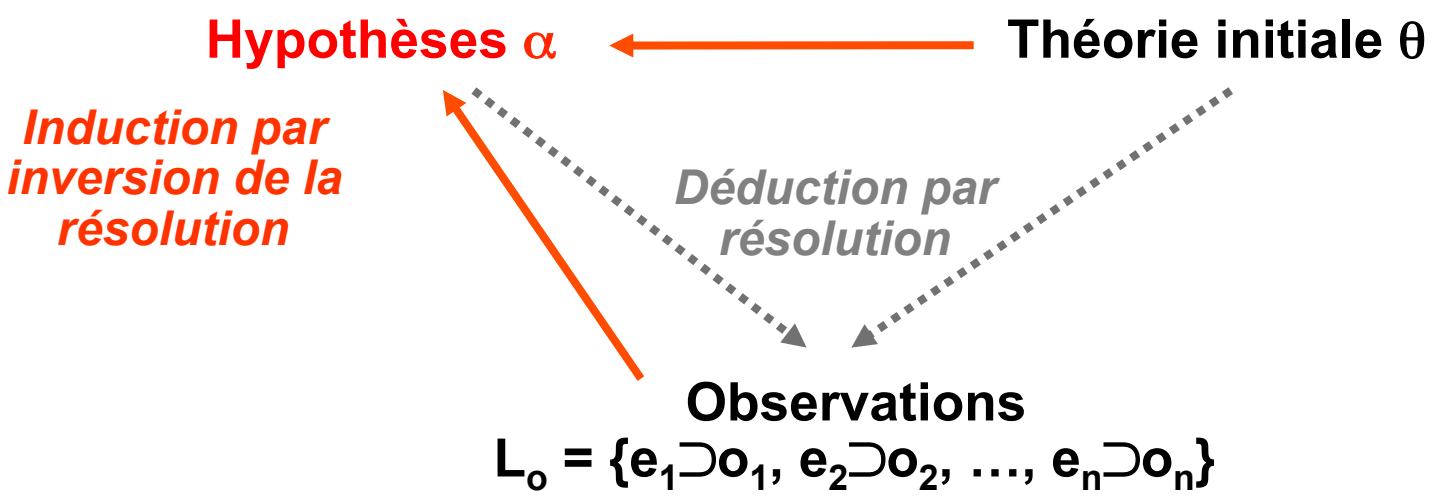
Théorème de résolution

Théorème de résolution: **soit S un ensemble de clauses,**

***S est insatisfiable si et seulement si il existe une R-réfutation de S*, autrement dit on peut dériver la clause vide par application réitérée de la règle ce résolution, ce qui s'ecrit S $\vdash_{Rès}$.**

Induction = Déduction^{-1}

Inductive Logic Programming **(Muggleton 1992)**



- Le problème:
  - **Etant donné**
    - » **un ensemble d'observation L_o et**
    - » **une théorie θ**
  - **Construire une hypothèse α telle que**
    - ≈ $\forall e_i \supset o_i \in L_o \; \alpha \wedge \theta \wedge e_i \vdash_{Res} o_i$
- L'induction vue comme une inversion de la résolution
  - **Il s'agit de construire une hypothèse α telle que**

    $L_o \wedge \theta \vdash_{Res^{-1}} \alpha$

Induction = Déduction^{-1}

Inductive Logic Programming **(Muggleton 1992)**

- Le problème:
  - **Etant donné**
    - **un ensemble d'observation $L_o = \{e_1 \supset o_1, \ldots, e_n \supset o_n\}$ et**
    - **une théorie θ**
  - **Construire une hypothèse α telle que**
    - **$\forall\ e_i \supset o_i \in L_o\ \ \alpha \wedge \theta \wedge e_i \vdash_{Res} o_i$**

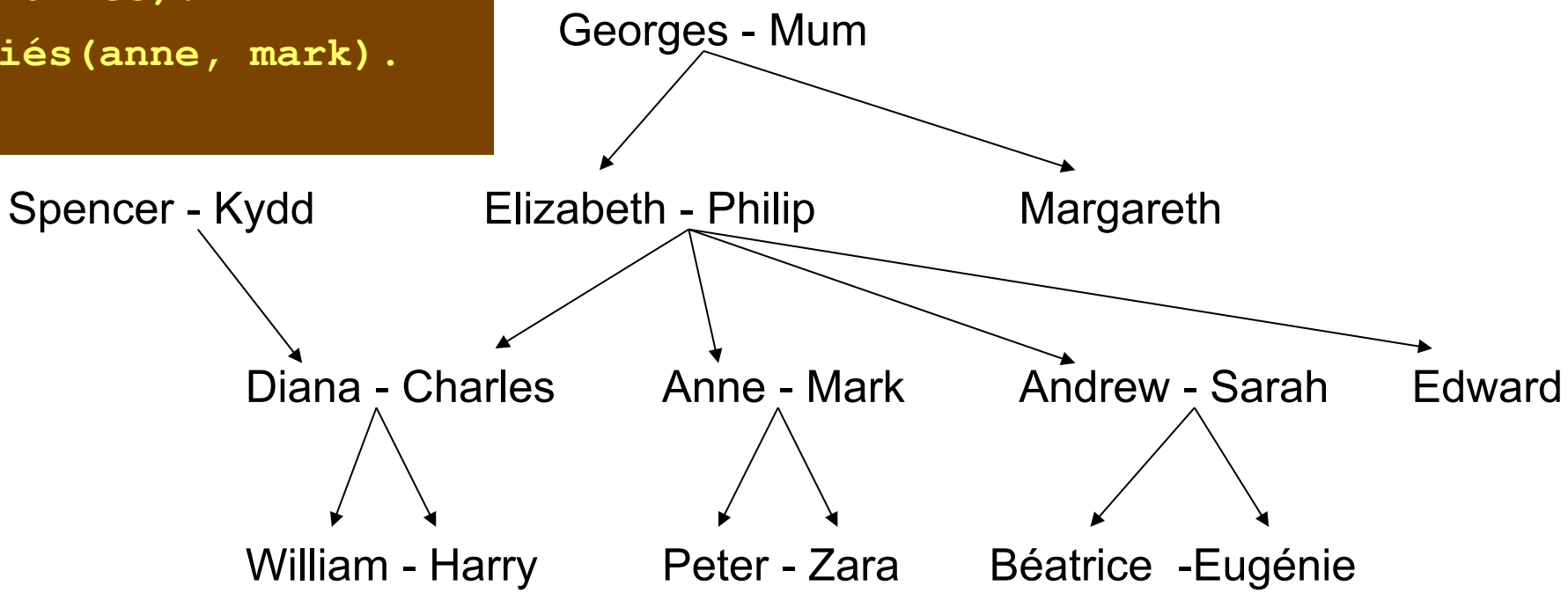
- L'induction vue comme une inversion de la résolution
  - **Il s'agit de construire une hypothèse α telle que**

    $L_o \wedge \theta \vdash_{Res^{-1}} \alpha$

Exemple1 - relations familiales

```
père(philip, charles).
père(philip, anne).
…
mère(mum, margareth).
mère(anne, peter).
…
mariés(diana,
  charles).
mariés(anne, mark).
…
```

```
homme(philip).
homme(andrew).
…

femme(béatrice).
femme(margareth).
…
```



Apprentissage « grand-parent(X, Y) »

- On part des observations (exemples positifs et négatifs) relatives à ce concept :

Ex. : `grand-parent(mum, charles).` } E^+

`grand-parent(elizabeth, béatrice).`

`...`

`¬grand-parent(mum, harry).` } E^-

`¬grand-parent(spencer, peter).`

`...`

- On veut construire des clauses qui permettent de dériver automatiquement le concept « `grand-parent(X, Y)` » à partir des exemples – Hypothèse α:

H_1 : grand-parent(X,Y) ≡ [∃Z / mère(X,Z) ∧ mère(Z,Y)]
∨ [∃Z / mère(X,Z) ∧ père(Z,Y)]
∨ [∃Z / père(X,Z) ∧ mère(Z,Y)]
∨ [∃Z / père(X,Z) ∧ père(Z,Y)]

Apprentissage « grand-parent(X, Y) » avec connaissance implicite

- On part des observations (exemples positifs et négatifs) relatives à ce concept :

Ex. : grand-parent(mum, charles). } E^+

grand-parent(elizabeth, béatrice). } E^+

…

¬grand-parent(mum, harry). } E^-

¬grand-parent(spencer, peter). } E^-

…

- On aimerait obtenir (hypothèse α):

H_1 : grand-parent(X,Y) ≡ [∃Z / (père(X,Z) ∧ père(Z,Y)) ∨
(père(X,Z) ∧ mère(Z,Y)) ∨
(mère(X,Z) ∧ père(Z,Y)) ∨
(mère(X,Z) ∧ mère(Z,Y))]

Apprentissage « grand-parent(X, Y) » avec connaissance implicite

- On part des observations (exemples positifs et négatifs) relatives à ce concept :

Ex. : `grand-parent(mum, charles).` } E^+

`grand-parent(elizabeth, béatrice).`

...

`¬grand-parent(mum, harry).` } E^-

`¬grand-parent(spencer, peter).`

...

- Supposons maintenant que l'on dispose d'une connaissance supplémentaire (théorie Θ):

`parent(X,Y) ≡ (mère(X,Y) ∨ père(X,Y))`

On aimerait obtenir (hypothèse α):

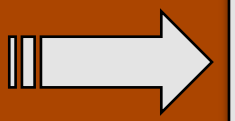
H_2 : `grand-parent(X,Y) ≡ [∃Z / parent(X,Z) ∧ parent(Z,Y)]`

Quelques exemples

- **Apprentissage de la fonction “member”**

member(a, [a,b,c]).
member(b,[a,b,c]).
member(3,[5,4,3,2,1]).
:- member(b, [1,2,3]).
:- member(3, [a,b,c]).

Exemples



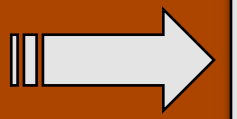
member(X, [X|Y]).
member(X, [Y|Z]) :- member(X,Z).

Hypothèse α

Quelques exemples *(suite)*

- **Utilisation de théorie du domaine**
- **Apprentissage du tri rapide (quicksort)**

```
qsort([b,c,a], [a,b,c]).
qsort([], []) .
qsort([5,3],[3,5]).
:- qsort([5,3],[5,3]).
:- qsort([1,3] [3]).

split(L, A, B) :- ...
append(A,B,C) :- ...
```

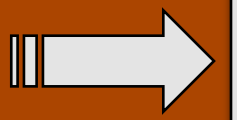


```
qsort([], []).
qsort([X], [X]).
qsort(X,Y) :-
      split(X, A, B),
      qsort(A, AS),
      qsort(B, BS),
      append(AS, BS, Y).
```

Quelques exemples *(fin)*

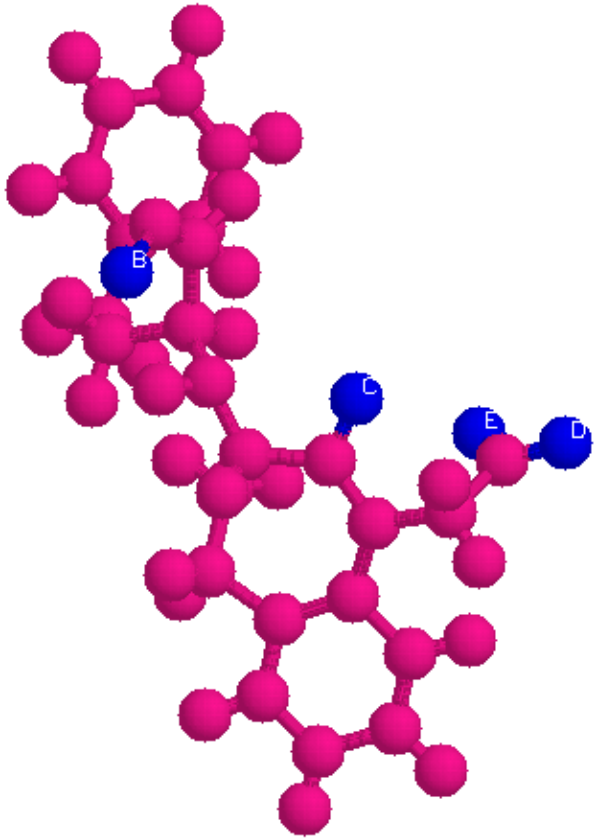
- **Possibilité d'apprentissage de contraintes**

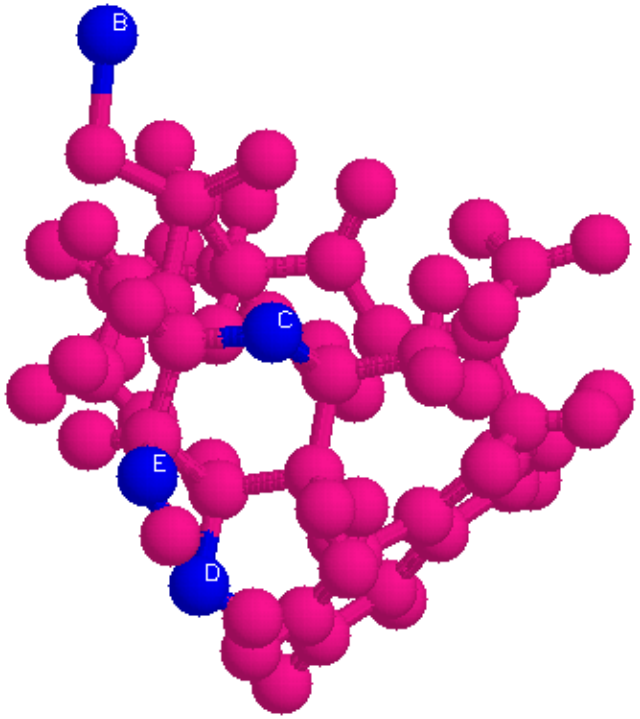
```
parent(jack,mary).
parent(mary,bob).
father(jack,mary).
mother(mary,bob).
male(jack).
male(bob).
female(mary).
```



```
:- male(X), female(X).
male(X) :- father(X,Y).
father(X,Y); mother(X,Y) :- parent(X,Y).
…
```

- **Autre exemple**
  - **molecules:** **(Muggleton et al. 1996)**





Related Work: Robot Scientist

(King et al. - Nature)

Molécules

Description de molécules:

```
atm(m1,a1,o,2,3.430400,-3.116000,0.048900).
atm(m1,a2,c,2,6.033400,-1.776000,0.679500).
atm(m1,a3,o,2,7.026500,-2.042500,0.023200).
...
bond(m1,a2,a3,2).
bond(m1,a5,a6,1).
bond(m1,a2,a4,1).
bond(m1,a6,a7,du).
...
```

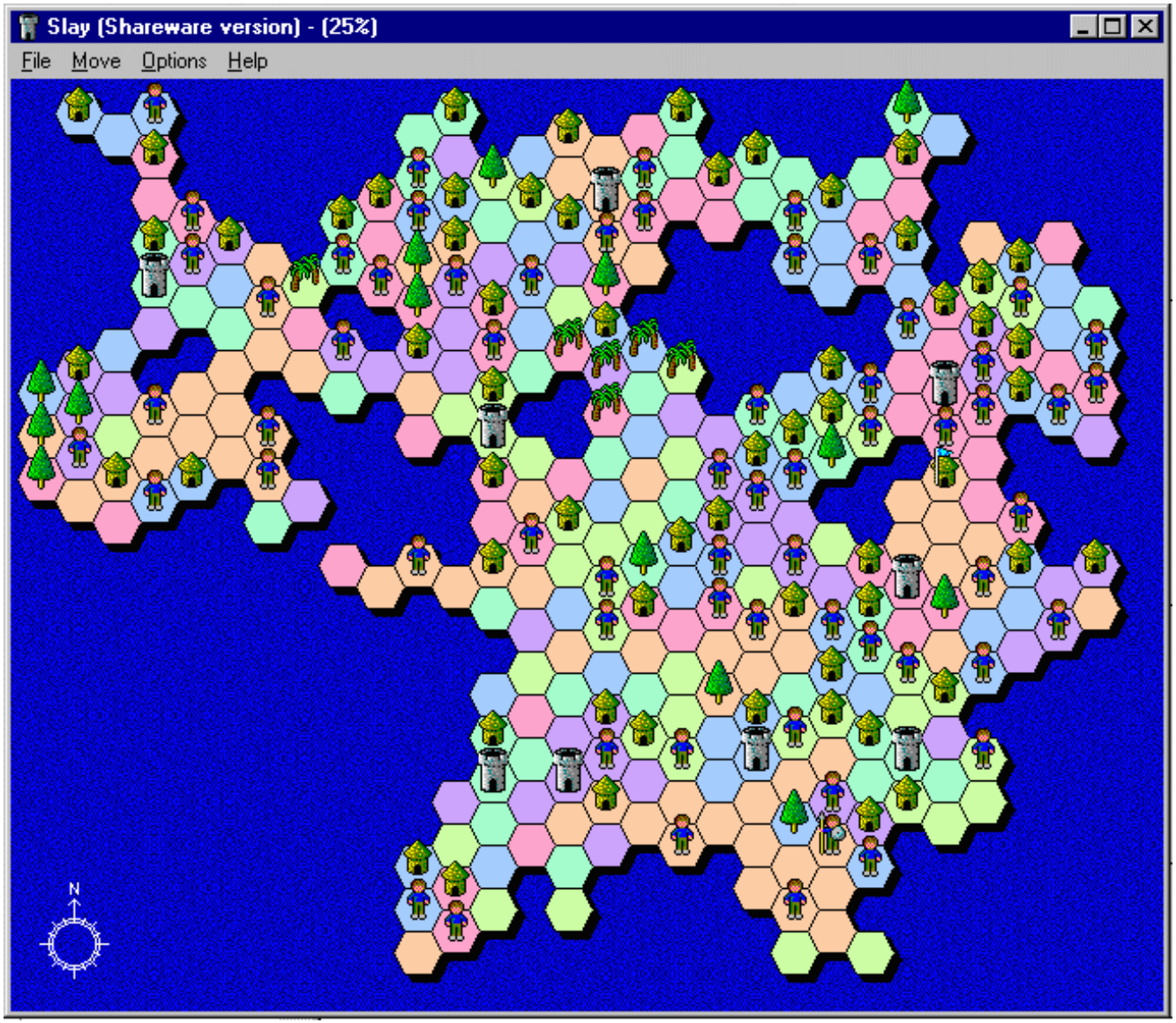
Connaissances implicites

```
...
hacc(M,A):- atm(M,A,o,2,_,_,_).
hacc(M,A):- atm(M,A,o,3,_,_,_).
hacc(M,A):- atm(M,A,s,2,_,_,_).
hacc(M,A):- atm(M,A,n,ar,_,_,_).

zincsite(M,A):-
        atm(M,A,du,_,_,_,_).

hdonor(M,A) :-
        atm(M,A,h,_,_,_,_),
        not(carbon_bond(M,A)), !.
...
```

-> Hypothese:

```
active(A) :- zincsite(A,B), hacc(A,C), hacc(A,D), hacc(A,E),
        dist(A,C,B,4.891,0.750), dist(A,C,D,3.753,0.750), dist(A,
        C,E,3.114,0.750), dist(A,D,B,8.475,0.750), dist(A,D,E,
        2.133,0.750), dist(A,E,B,7.899,0.750).
```

Apprentissage de stratégies pour les jeux



Résolution (rappels)

Définition:

- soit A et B deux clauses dont les variables sont disjointes (au besoin, renommer les variables de l'une d'entre elles),
- soit L1 un littéral de A et L2 un littéral de B,
- soit σ le pgu (plus grand unifieur) de L1 et ¬L2

On dira que **C est le résolvant de A et B par L1 et L2**, (ce que l'on notera Rès(A, B; L1, L2)) ssi

$C = Rès(A, B; L1, L2) = (A\sigma - \{L1\sigma\}) \cup (B\sigma - \{L2\sigma\})$

Exemple: **Rès(R(x, y), ¬R(u, v); R(x, y), ¬R(u, v)) = Ø**

Rès(P(x) ∨ P(y), ¬P(z) ∨ ¬P(r); P(x), ¬P(z)) = P(y) ∨ ¬P(r)

Déduction automatique: règle de résolution

- Règle de résolution ***(Robinson 1965)*** :
  Soit C_1 et C_2 deux clauses.
  Etant donné L_1 et L_2 deux littéraux de C_1 et C_2.
  Si L_1 et $\neg L_2$ ont un unifieur plus général σ alors la clause C, est un résolvant binaire de C_1 et C_2 par L_1 et L_2:

$$C = Res(C_1 , C_2; L_1, L_2) = (C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$$

- Complétude du principe de résolution
  Un ensemble S de clauses est insatisfiable si et seulement si il y a une déduction de la clause vide à partir de S par application de la règle de résolution.
  En d'autres termes, S est insatisfiable ssi $S \vdash_{Res}$

L I P 6 C N R S

Inversion de la résolution (suite)

Hypothèse 1: C_1 et C_2 n'ont aucun littéral en commun

$C = (C_1\sigma_1 - \{L_1\sigma_1\}) \cup (C_2\sigma_2 - \{\neg L_1\sigma_1\})$

Cela donne

$(C_2\sigma_2 - \{\neg L_1\sigma_1\}) = C - (C_1\sigma_1 - \{L_1\sigma_1\})$

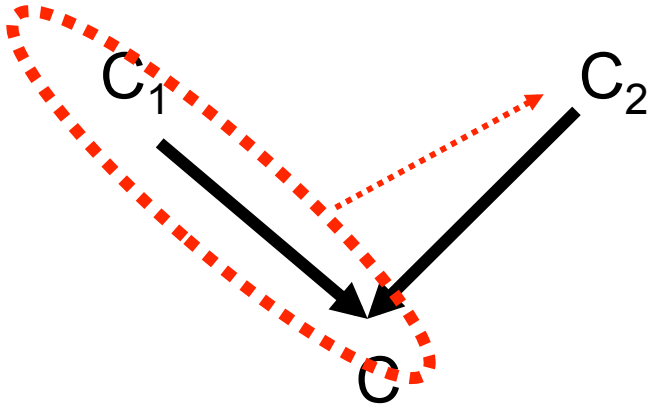
d'où $C_2\sigma_2 = (C - (C_1\sigma_1 - \{L_1\sigma_1\})) \cup \{\neg L_1\sigma_1\}$

Enfin, pour calculer on fait appel à une inversion de substitution:

$C_2 = (C - (C_1\sigma_1 - \{L_1\sigma_1\}))\sigma_2^{-1} \cup \{\neg L_1\sigma_1\}\sigma_2^{-1}$

Inversion: opérateurs V

L I P 6 C N R S



$$C_2 = (C - (C_1\sigma_1 - \{L_1\sigma_1\}))\sigma_2^{-1} \cup \{\neg L_1\sigma_1\}\sigma_2^{-1}$$

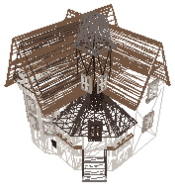
Hypothèse 2 *(seulement pour les opérateurs V):*

La clause C_1 est supposée unitaire,

$C_1\,\sigma_1 - \{L_1\,\sigma_1\}$ est donc égal à la clause vide, d'où

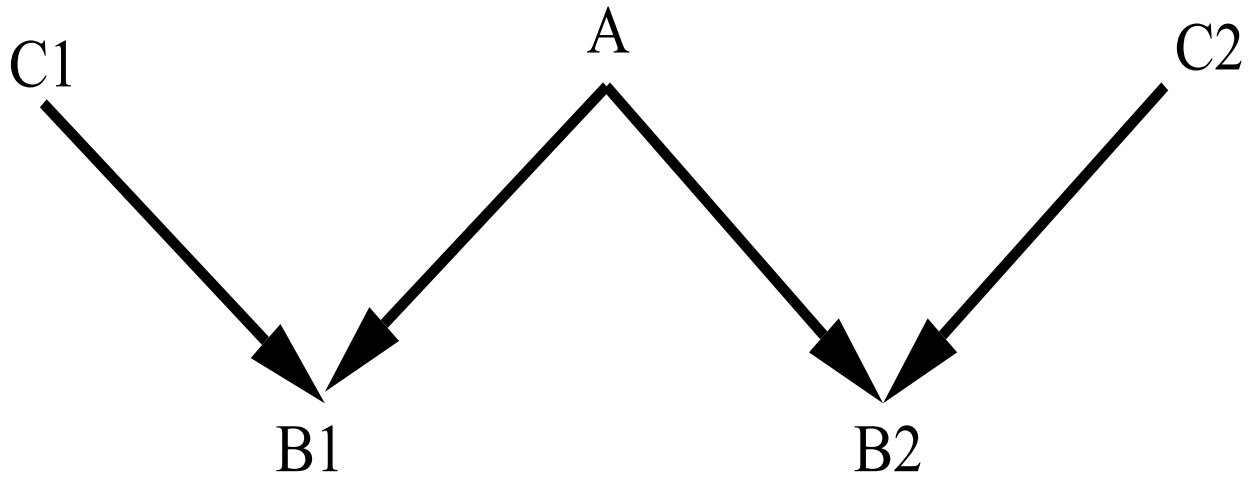
$C_1\,\sigma_1 = L_1\,\sigma_1$ et

$C_2 = C\,\sigma_2^{-1} \cup \{\neg C_1\,\sigma_1\}\,\sigma_2^{-1}$



L I P 6 C N R S

Opérateurs W

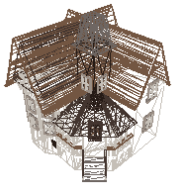


$B_i = (A - \{L\})\ \theta_{A,i} \cup (C_i - \{L_i\})\ \theta_{C,i}$

Si les clauses C_i sont des clauses unitaires:

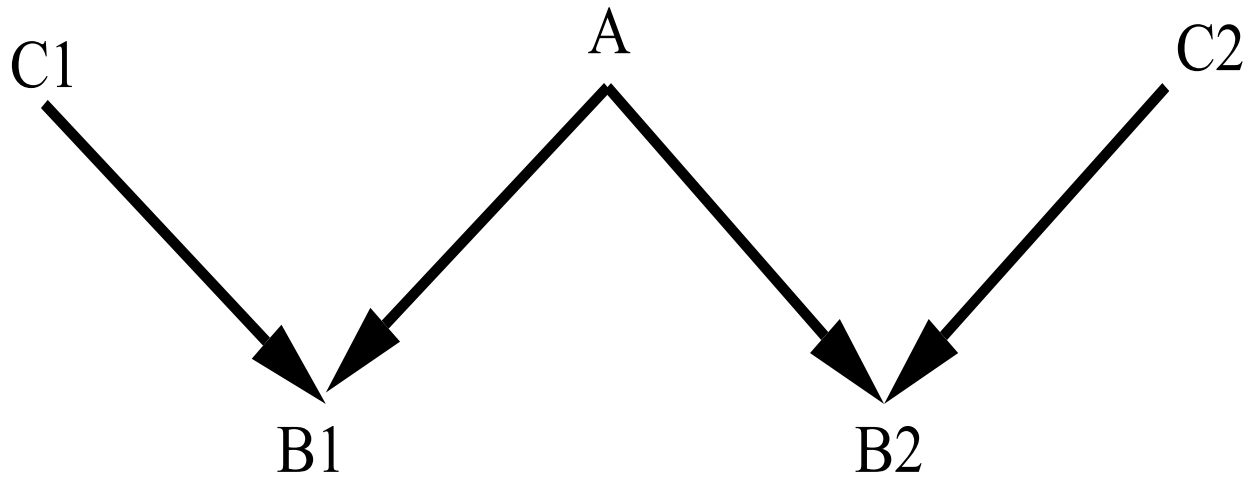
$B_i = (A - \{L\})\ \theta_{A,i}$

$A = B_i\ (\theta_{A,i})^{-1} \cup \{L\}.$



L I P 6 C N R S

Opérateurs W



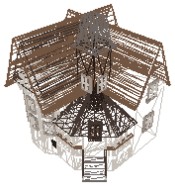
$B_1 = (A - \{\neg L_1\})\ \theta_{A,1} \cup (C_1 - \{L_1\})\ \theta_{C,1}$

$B_2 = (A - \{\neg L_2\})\ \theta_{A,2} \cup (C_2 - \{L_2\})\ \theta_{C,2}$

Si les clauses C_1 et C_2 sont des clauses unitaires:

$B_1 = (A - \{L\})\ \theta_{A,1}$ et $B_2 = (A - \{L\})\ \theta_{A,2}$

$A = B_1\ (\theta_{A,1})^{-1} \cup \{L\} = B_2\ (\theta_{A,2})^{-1} \cup \{L\}.$



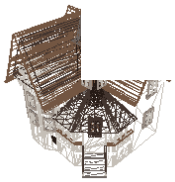
L I P 6 C N R S

Rappel
Induction = Déduction^{-1}

Inductive Logic Programming **(Muggleton 1992)**

- Le problème:
  - **Etant donné**
    - **un ensemble d'observation $L_o = \{e_1 \supset o_1, \ldots, e_n \supset o_n\}$ et**
    - **une théorie θ**
  - **Construire une hypothèse α**
    - $\forall\, e_i \supset o_i \in L_o \;\; \alpha \wedge \theta \wedge e_i \vdash_{Res} o_i$
- L'induction vue comme une inversion de la résolution
  - **Il s'agit de construire une hypothèse α telle que**

$L_o \wedge \theta \vdash_{Res^{-1}} \alpha$



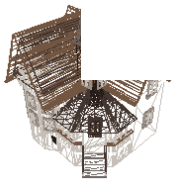
LIP6 CNRS

Rappel
Induction = Déduction^{-1}

Inductive Logic Programming **(Muggleton 1992)**

- Le problème:
  - **Etant donné**
    - **un ensemble d'observation $L_o = \{e_1 \supset o_1, \ldots, e_n \supset o_n\}$ et**
    - **une théorie θ qui n'explique pas les exemples de L_o**
  - **Construire une hypothèse α qui explique les exemples**
    - $\forall e_i \supset o_i \in L_o \; \alpha \wedge \theta \wedge e_i \vdash_{Res} o_i$
- L'induction vue comme une inversion de la résolution
  - **Il s'agit de construire une hypothèse α telle que**

    $L_o \wedge \theta \vdash_{Res^{-1}} \alpha$



Inversion de la résolution: un exemple

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

Théorie θ

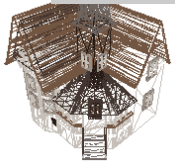
explique l'exemple:

(E1): grandpère(armand, jean-luc) ←
père(armand, edmond) ∧
enfant(jean-luc, edmond) ∧
sexe(edmond, masculin).

mais pas:

(E2) : grandpère(tom,liz) ← père(tom,helen) ∧
sexe(helen,féminin) ∧ enfant(liz, helen)

Ensemble d'observations L_o



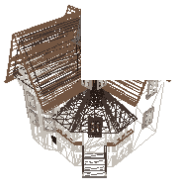
L I P 6 C N R S

Rappel
Induction = Déduction^{-1}

Inductive Logic Programming **(Muggleton 1992)**

- Le problème:
  - **Etant donné**
    - **un ensemble d'observation $L_o = \{e_1 \supset o_1, \ldots, e_n \supset o_n\}$ et**
    - **une théorie θ qui n'explique pas tous les exemples**
  - **Construire une hypothèse α qui explique les exemples**
    - $\forall\ e_i \supset o_i \in L_o\ \ \alpha \wedge \theta \wedge e_i \vdash_{Res} o_i$
- L'induction vue comme une inversion de la résolution
  - **Il s'agit de construire une hypothèse α telle que**

    $L_o \wedge \theta \vdash_{Res^{-1}} \alpha$



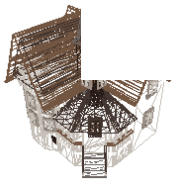
LIP6 CNRS

Rappel
Induction = Déduction^{-1}

Inductive Logic Programming **(Muggleton 1992)**

- Le problème:
  - **Etant donné**
    - **un ensemble d'observation $L_o = \{e_1 \supset o_1, \ldots, e_n \supset o_n\}$ et**
    - **une théorie θ** *(qui n'explique pas les exemples)*
  - **Construire une hypothèse α telle que $\alpha \wedge \theta$ explique les exemples**
    - $\forall e_i \supset o_i \in L_o \; \alpha \wedge \theta \wedge e_i \vdash_{Res} o_i$
- L'induction vue comme une inversion de la résolution
  - **Il s'agit de construire une hypothèse α telle que**

    $L_o \wedge \theta \vdash_{Res^{-1}} \alpha$ **cad telle que $\alpha \wedge \theta$ explique les exemples**

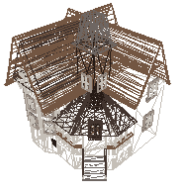


Explication d'un exemple

- **Définition**: On dit qu'une théorie T qu'elle *"explique" une clause* $L_1 \vee L_2 \vee ... \vee L_k$:- $L_{k+1} \wedge ... \wedge L_m$ si et seulement si: $T \cup \{L_{k+1} \wedge ... \wedge L_m\} \vdash \{L_1 \vee L_2 \vee ... \vee L_k\}$

- **Définition**: un exemple d'un concept C (ou d'un prédicat P) est une clause du type C :- $L_{k+1} \wedge ... \wedge L_m$ (ou P(x) :- $L_{k+1} \wedge ... \wedge L_m$)

- **Définition**: Une théorie T explique l'exemple (ou l'observation) C :- $L_{k+1} \wedge ... \wedge L_m$ si et seulement si $T \cup \{L_{k+1} \wedge ... \wedge L_m\} \vdash C$



L I P 6 C N R S

Un exemple

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

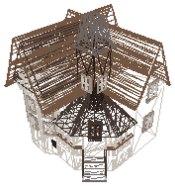
(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

La théorie *{T1, T2, T3}* explique l'exemple *E1*:

(E1): grandpère(armand, jean-luc) ←
père(armand, edmond) ∧
enfant(jean-luc, edmond) ∧
sexe(edmond, masculin).

mais pas l'exemple *E2*:

(E2) : grandpère(tom,liz) ← père(tom,helen) ∧
sexe(helen,féminin) ∧ enfant(liz, helen)



L I P 6 C N R S

Un exemple - suite

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

La théorie *{T1, T2, T3}* explique l'exemple *E1*:

(E1): grandpère(armand, jean-luc) ←

père(armand, edmond) ∧

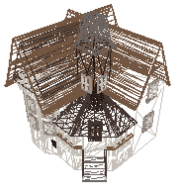
enfant(jean-luc, edmond) ∧

sexe(edmond, masculin).

Car *{T1, T2, T3}* ∪

{père(armand, edmond) ∧ enfant(jean-luc, edmond) ∧

sexe(edmond, masculin)}* ⊢ *grandpère(armand, jean-luc)



Explication de E1

(E1): grandpère(armand, jean-luc) ←
père(armand, edmond) ∧
enfant(jean-luc, edmond) ∧
sexe(edmond, masculin).

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

Théorie θ

Partant de la théorie θ et des trois clauses:

père(armand, edmond)
enfant(jean-luc, edmond)
sexe(edmond, masculin).

Obervation E1

On dérive (par application de la résolution)

grandpère(armand, jean-luc)

Explication de E1

(E1): grandpère(armand, jean-luc) ←
père(armand, edmond) ∧
enfant(jean-luc, edmond) ∧
sexe(edmond, masculin).

Théorie

(T1) : grandpère(X,Z) :- père(X,Y), père(Y,Z).

(T2) : mère(X,Y) :- sexe(X, feminin), enfant(Y,X).

(T3) : père(X,Y) :- sexe(X, masculin) , enfant(Y,X).

Obervation E1

père(armand, edmond)
enfant(jean-luc, edmond)
sexe(edmond, masculin).

But

:- grandpère(armand, jean-luc).

1. *:- père(armand, Y), père(Y, jean-luc)*
2. *:- père(edmond, jean-luc).*
3. *:- sexe(edmond, masculin), enfant(jean-luc, edmond)*
4. *:- enfant(jean-luc, edmond).*
5. ∅

L I P 6 C N R S

Un exemple – suite(suite)

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

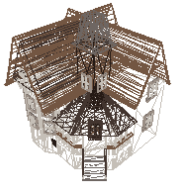
(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

La théorie *{T1, T2, T3}* n'explique pas l'exemple *E2*:

(E2) : grandpère(tom,liz) ← père(tom,helen) ∧ sexe(helen,féminin) ∧ enfant(liz, helen)

Car nous n'avons pas

***{T1, T2, T3}* ∪ {père(tom,helen) ∧ sexe(helen,féminin) ∧ enfant(liz, helen)} |- grandpère(tom,liz)**



Non explication de E2

(E2) : grandpère(tom,liz) ←
père(tom,helen) ∧
sexe(helen,féminin)∧
enfant(helen,liz)

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

Théorie θ

Partant de la théorie θ et des trois clauses:

père(tom,helen)

enfant(liz, helen)

sexe(helen,féminin)

Obervation E2

On ne peut pas dériver (par application de la résolution)

grandpère(tom, liz)

Non explication de E2

(E2) : grandpère(tom,liz) ←
père(tom,helen) ∧
sexe(helen,féminin)∧
enfant(helen,liz)

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z). **Théorie θ**

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

père(tom,helen)

enfant(liz, helen)

Obervation E2

sexe(helen,féminin)

:- grandpère(tom, liz).

1. **:- père(tom, Y), père(Y, liz).**
2. **:- père(helen, liz).**
3. **:- sexe(helen, masculin), enfant(liz, helen).**
4. **:- sexe(helen, masculin)…**

Echec de la démonstration…

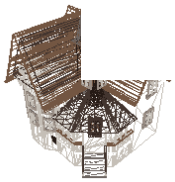
L I P 6 C N R S

Rappel
Induction = Déduction^{-1}

Inductive Logic Programming **(Muggleton 1992)**

- Le problème:
  - **Etant donné**
    - **un ensemble d'observation $L_o = \{e_1 \supset o_1, \ldots, e_n \supset o_n\}$ et**
    - **une théorie θ** *(qui n'explique pas les exemples)*
  - **Construire une hypothèse α telle que $\alpha \wedge \theta$ explique les exemples**
    - $\forall e_i \supset o_i \in L_o \; \alpha \wedge \theta \wedge e_i \vdash_{Res} o_i$
- L'induction vue comme une inversion de la résolution
  - **Il s'agit de construire une hypothèse α telle que**

    $L_o \wedge \theta \vdash_{Res^{-1}} \alpha$ **cad telle que $\alpha \wedge \theta$ explique les exemples**



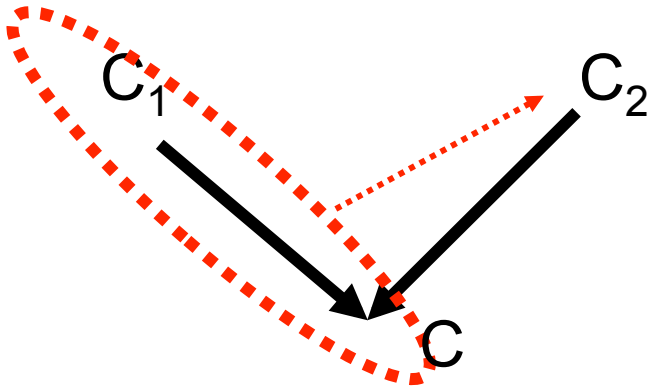
Exemple *suite*

L I P 6 C N R S

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).



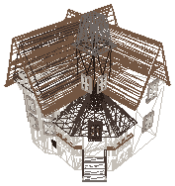
C correspond à l'exemple E2

(E2) : grandpère(tom,liz) ∨ ~père(tom,helen) ∨

~sexe(helen,féminin) ∨ ~enfant(liz, helen)

C1 à T2

(T2) : mère(X,Y) ∨ ~sexe(X, feminin) ∨ ~enfant(Y,X).



Exemple *suite*

L

I

P

6

C

N

R

S

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

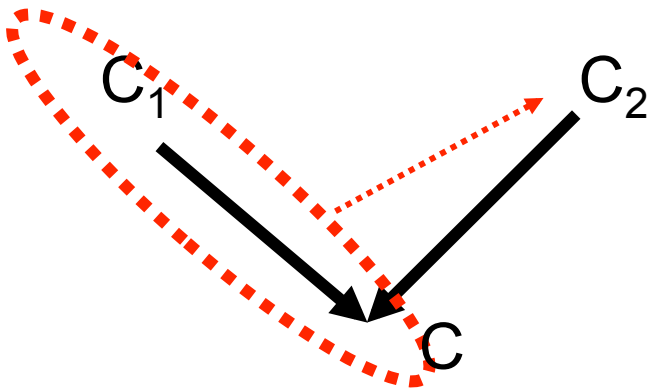
(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

But: "expliquer"

(E2) : grandpère(tom,liz) ∨ ~père(tom,helen) ∨ ~sexe(helen,féminin) ∨ ~enfant(liz, helen)

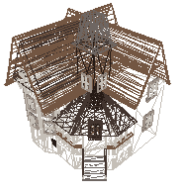
Avec T2 et C2



(T2) : mère(X,Y) ∨ ~sexe(X, feminin) ∨ ~enfant(Y,X).

C2 : grandpère(tom,liz) ∨ ~père(tom,helen) ∨ ~mère(helen,liz)

(E2) : grandpère(tom,liz) ∨ ~père(tom,helen) ∨ ~sexe(helen,féminin) ∨ ~enfant(liz, helen)



L I P 6 C N R S

Inversion de la résolution (rappel)

Hypothèse 1: C_1 et C_2 n'ont aucun littéral en commun

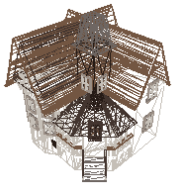
$C = (C_1\sigma_1 - \{L_1\sigma_1\}) \cup (C_2\sigma_2 - \{\neg L_1\sigma_1\})$ donne

$(C_2\sigma_2 - \{\neg L_1\sigma_1\}) = C - (C_1\sigma_1 - \{L_1\sigma_1\})$ d'où

$C_2\sigma_2 = (C - (C_1\sigma_1 - \{L_1\sigma_1\})) \cup \{\neg L_1\sigma_1\}$

Pour calculer on fait appel à une inversion de substitution:

$C_2 = (C - (C_1\sigma_1 - \{L_1\sigma_1\}))\sigma_2^{-1} \cup \{\neg L_1\sigma_1\}\sigma_2^{-1}$

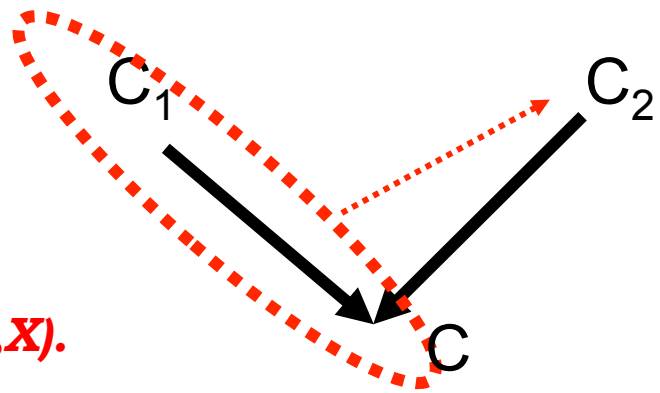


Exemple *suite*

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).
(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).
(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

C correspond à l'exemple E2

(E2) : grandpère(tom,liz) v ~père(tom,helen) v
~sexe(helen,féminin) v ~enfant(liz, helen)



C1 à T2

(T2) : mère(X,Y) v ~sexe(X, feminin) v ~enfant(Y,X).

En posant L_1 = mere(X, Y) et σ_1 = {X/helen, Y/liz}

Et en plongeant dans $C_2 = (C - (C_1\sigma_1 - \{L_1\sigma_1\}))\sigma_2^{-1} \cup \{\neg L_1\sigma_1\}\sigma_2^{-1}$

Cela donne:

$C_2 = (E2 - (T2\sigma_1 - \{mere(X, Y)\sigma_1\}))\sigma_2^{-1} \cup \{\neg mere(X,Y)\sigma_1\}\sigma_2^{-1}$

Et, on obtient:

$C_2 = (grandpère(tom,liz) \vee \neg père(tom, helen))\,\sigma_2^{-1} \cup \{\neg mère(helen, liz)\}\,\sigma_2^{-1}$

Exemple (suite)

$C_2 =$ *(grandpère(tom,liz) v ~père(tom,helen))* $\sigma_2^{-1} \cup$
{¬ mère(helen, liz)} σ_2^{-1}

On cherche la substitution σ_2 la plus générale.

Puisqu'il y a trois constantes *tom,liz* et *helen* cela donne: {U/*tom*, V/*liz*, W/*helen*} d'où on tire

$C_2 =$ *(grandpère(U,V) v ~père(U,W) v ~mère(W,V) }*

ce qui donne une nouvelle clause induite:

(T4) : grandpère(X,Y) ← père(X,V) ∧ mère(V,Y).

L I P 6 C N R S

Inversion (exemple suite)

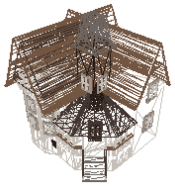
Nouvelle théorie

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧
enfant(Y,X).

(T4) : grandpère(X,Y) ← père(X,V) ∧ mère(V,Y).



L I P 6 C N R S

Inversion (exemple suite)
Invention de descripteurs

Avec la nouvelle théorie

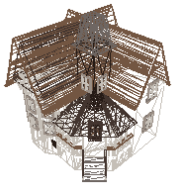
(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧
enfant(Y,X).

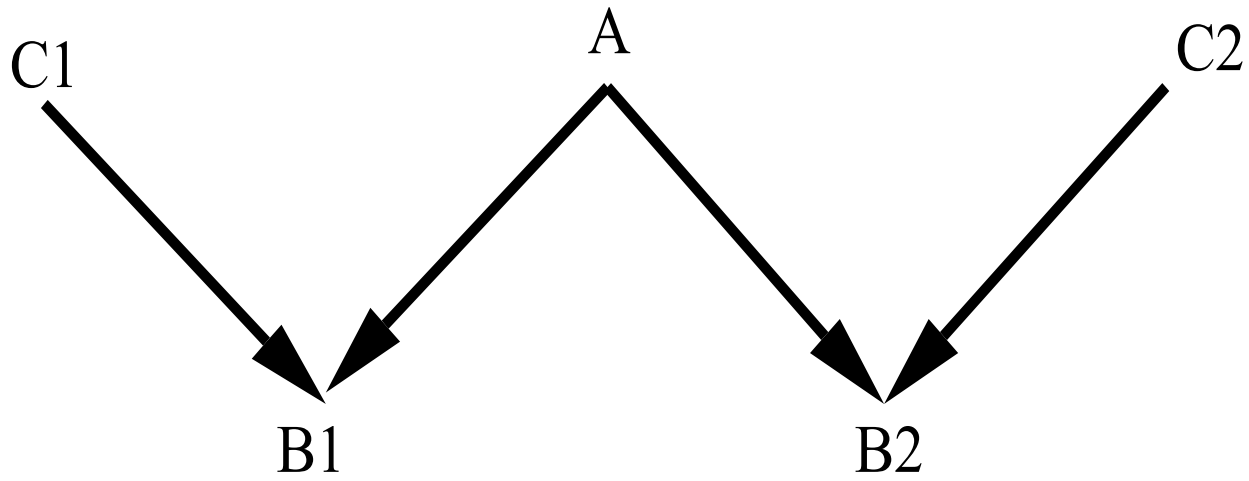
(T4) : grandpère(X,Y) ← père(X,V) ∧ mère(V,Y).

On applique les opérateurs W



L I P 6 C N R S

Opérateurs W



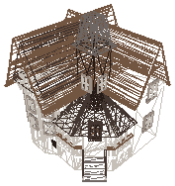
$B_1 = (A - \{\neg L_1\})\ \theta_{A,1} \cup (C_1 - \{L_1\})\ \theta_{C,1}$

$B_2 = (A - \{\neg L_2\})\ \theta_{A,2} \cup (C_2 - \{L_2\})\ \theta_{C,2}$

Si les clauses C_1 et C_2 sont des clauses unitaires:

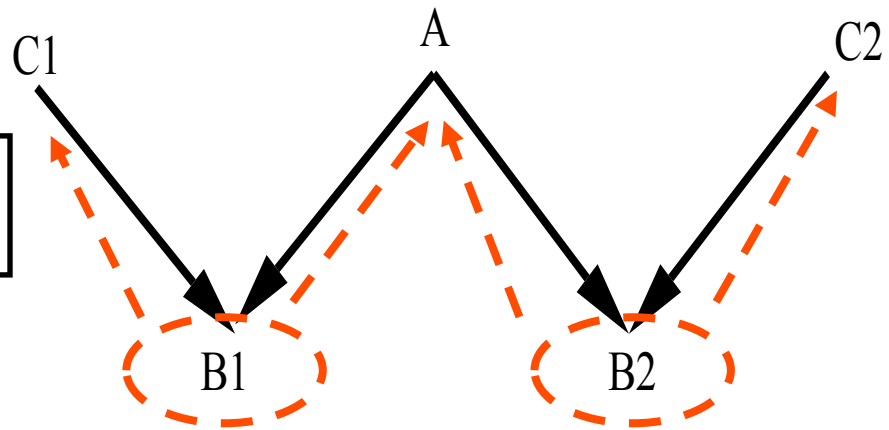
$B_1 = (A - \{L\})\ \theta_{A,1}$ et $B_2 = (A - \{L\})\ \theta_{A,2}$

$A = B_1\ (\theta_{A,1})^{-1} \cup \{L\} = = B_2\ (\theta_{A,2})^{-1} \cup \{L\}.$



LIP6 CNRS

Exemple (opérateur W)



Avec la nouvelle théorie

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

(T4) : grandpère(X,Y) ← père(X,V) ∧ mère(V,Y).

$B_i = (A - \{L\})\ \theta_{A,i} \cup (C_i - \{L_i\})\ \theta_{C,i}$.

(B1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(B2) : grandpère(X,Z) ← père(X,Y) ∧ mère(Y,Z).

L1= père(_,_)

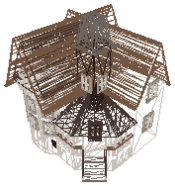
L2=mère(_,_)

L = nouv(_,_)

(A) : grandpère(X,Z) ← père(X,Y) ∧ nouv(Y,Z).

(C1) : nouv(X,Y) ← père(X,Y).

(C2) : nouv(X,Y) ← mère(X,Y).



L I P 6 C N R S

Inversion (exemple suite)
Invention de descripteurs

Avec la nouvelle théorie

(T1) : grandpère(X,Z) ← père(X,Y) ∧ père(Y,Z).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).

(T4) : grandpère(X,Y) ← père(X,V) ∧ mère(V,Y).

On obtient en appliquant les opérateurs W

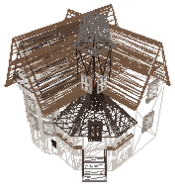
(T'1) : grandpère(X,Z) ← père(X,Y) ∧ nouv(Y,Z).

(T'2) : nouv(X,Y) ← père(X,Y).

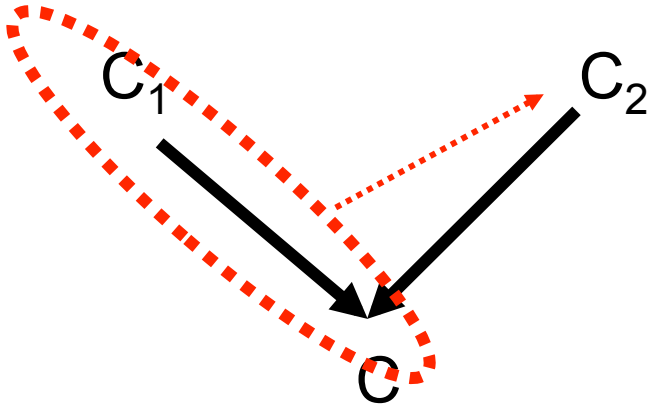
(T'3) : nouv(X,Y) ← mère(X,Y).

(T2) : mère(X,Y) ← sexe(X, feminin) ∧ enfant(Y,X).

(T3) : père(X,Y) ← sexe(X, masculin) ∧ enfant(Y,X).



Inversion: opérateurs V



$$C_2 = (C - (C_1\sigma_1 - \{L_1\sigma_1\}))\sigma_2^{-1} \cup \{\neg L_1\sigma_1\}\sigma_2^{-1}$$

Hypothèse 2 *(seulement pour les opérateurs V):*

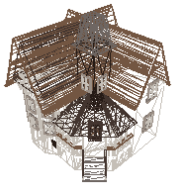
La clause C_1 est supposée unitaire,

$C_1\,\sigma_1 - \{L_1\,\sigma_1\}$ est donc égal à la clause vide,

d'où

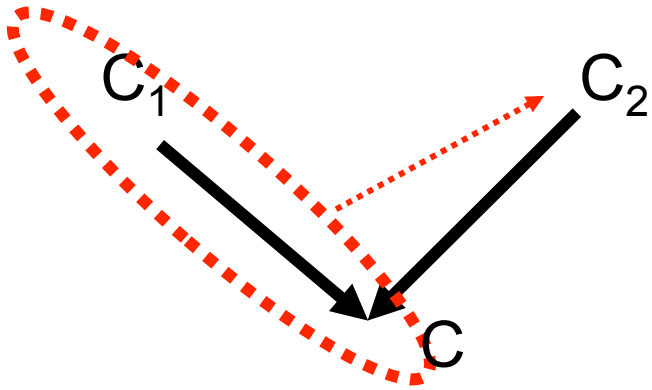
$C_1\,\sigma_1 = L_1\,\sigma_1$ et

$C_2 = C\,\sigma_2^{-1} \cup \{\neg C_1\,\sigma_1\}\,\sigma_2^{-1}$



L I P 6 C N R S

Exemple 2



Exemples:

C_1: Y < s(Y)

C: s(s(s(0))) < s(s(s(s(s(0)))))

On induit:

$C_2 = (C - (C_1\sigma_1 - \{L_1\sigma_1\}))\ \sigma_2^{-1} \cup \{\neg L_1\sigma_1\}\ \sigma_2^{-1}$

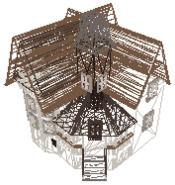
Comme nous sommes dans l'hypothèse 2 où C_1 est unitaire,

$C_1\sigma_1 - \{L_1\sigma_1\} = \varnothing$ d'où

$C_2 = C\ \sigma_2^{-1} \cup \{\neg C_1\sigma_1\}\ \sigma_2^{-1}$

Ce qui donne:

$C_2 = s(s(s(0))) < s(s(s(s(s(0)))))\ \sigma_2^{-1} \cup \{\neg Y < s(Y)\ \sigma_1\}\ \sigma_2^{-1}$



L I P 6 C N R S

Exemple 2 (suite)

Etant donné cette clause:

C_2 = s(s(s(0))) < s(s(s(s(s(0))))) σ_2^{-1} ∪ {¬ Y < s(Y) σ_1} σ_2^{-1}

Comment construire les substitutions σ_1 et σ_2^{-1} ?

1- Construction de σ_2^{-1} :

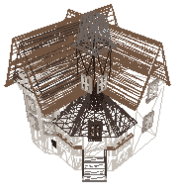
Généralisation maximale de s(s(s(0))) < s(s(s(s(s(0)))))

On remplace s(s(s(0))) par X: X < s(s(X))

Puis s(s(X)) par Z: X < Z

D'où σ_2 = {X/s(s(s(0))), Z/s(s(X))} et

C_2 = X < Z ∪ {¬ Y < s(Y) σ_1} σ_2^{-1}



L I P 6 C N R S

Exemple 2 (suite)

$C_2 = X < Z \cup \{\neg Y < s(Y)\ \sigma_1\}\ \sigma_2^{-1}$

$\sigma_2 = \{X/s(s(s(0))), Z/s(s(X))\}$

2- Construction de σ_1 :

On veut éliminer Y et généraliser en faisant apparaître des termes de σ_2^{-1}, c'est-à-dire s(s(s(0))) et s(s(X))

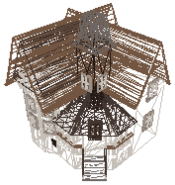
Pour ça, on peut remplacer Y par s(X): s(X) < s(s(X)) car Z est substitué par s(s(X)), ce qui donne: s(X) < Z

$\sigma_1 = \{Y/s(X), Z/s(s(X))\}$

$C_2 = X < Z \cup \{\neg s(X) < Z\}$

D'où la clause:

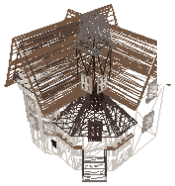
X < Z :- s(X) < Z



L I P 6 C N R S

Construction de programme - FOIL

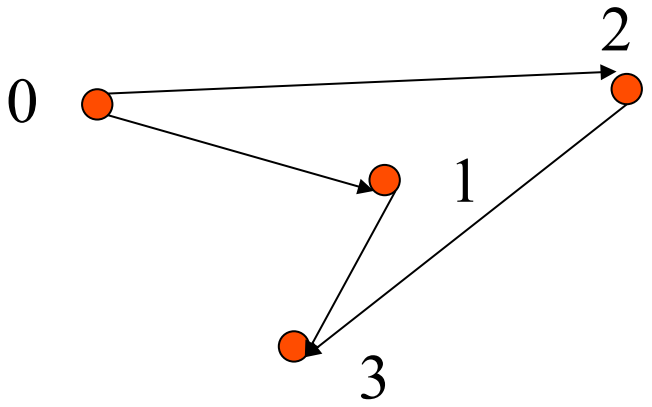
- **Inversion de la résolution: recherche ascendante**
- **FOIL: Recherche descendante – comme ID3 –** *Quinlan et Cameron-Jones 1993*
- **Entrée:**
  - **Ensemble E d'exemples et de contre-exemples du concept à apprendre**
  - **Connaissance implicite**
- **Exploration:**
  - **On part d'une hypothèse vide qui correspond au programme que l'on veut apprendre**
  - **On spécialise progressivement les clauses de façon à couvrir tous les exemples et à exclure les contre-exemples**
- **Comment limiter la recherche?**
  - **Limiter le langage – c'est-à-dire la forme de clauses – biais de langage**
  - **Limiter les hypothèses explorées – biais de recherche**



Exemple

- **Entrée du programme:**
- **Connaissance implicite**

```
arc(0,1).   arc(0,2).
arc(1,3).   arc(2,3).
```

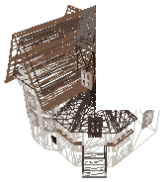


- **Exemples:**

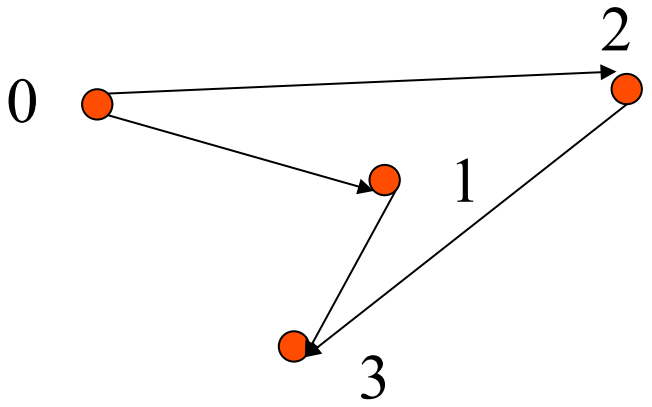
(+):
```
connecte(0,1). connecte(1,3). connecte(0,2).
connecte(2,3). connecte(0,3).
```

(-):
```
connecte(0,0).
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```

- **But: construire le prédicat connecte**
- **Biais de langage: il ne comprend que *arc/2* dans son corps**



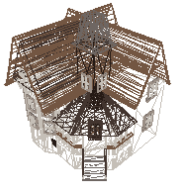
Exemple - suite



- **But: construire le prédicat** `connecte(X,Y)`
- **Biais de langage: connecte ne comprend que arc/2 dans son corps**
- **Programmes du type:**

```
connecte(X, Y).
connecte(X, Y) :- arc(U, V).
connecte(X, Y) :- arc(U, V), arc(W,Z).
connecte(X, Y) :- arc(U, V), arc(W,Z),
                  arc(R,T).
connecte(X, Y) :- arc(U, V), arc(W,Z),
                  arc(R,T),…
```

- **Construction:**
  - **Partir du programme le plus général:** `connecte(X, Y).`
  - **Le spécialiser par spécialisation et adjonction de clauses de façon à couvrir tous les exemples et à exclure tous les contre-exemples**
  - **spécialisation: exclusion de contre-exemples**
  - **adjonction de clauses: augmentation de la couverture**



Spécialisation de clauses

- **Techniques usuelles**
  - **Ajout littéral:**

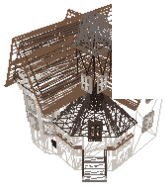
```
member(a, U) :- cons(a, T, U) devient
member(a, U) :- cons(b, nil, T),
                cons(a, T, U)
```

  - **Unifier deux occurrences de deux variables différentes**

```
member(X, U) :- cons(Y, nil, T),
                cons(X, T, U) devient
member(X, U) :- cons(X, nil, T),
                cons(X, T, U)
```

  - **Remplacer toutes les occurrences d'une même variable par une constante**

```
member(X, U) :- cons(X, T, U) devient
member(a, U) :- cons(a, T, U)
```



Autre Exemple: relations familiales

Connaissances a priori

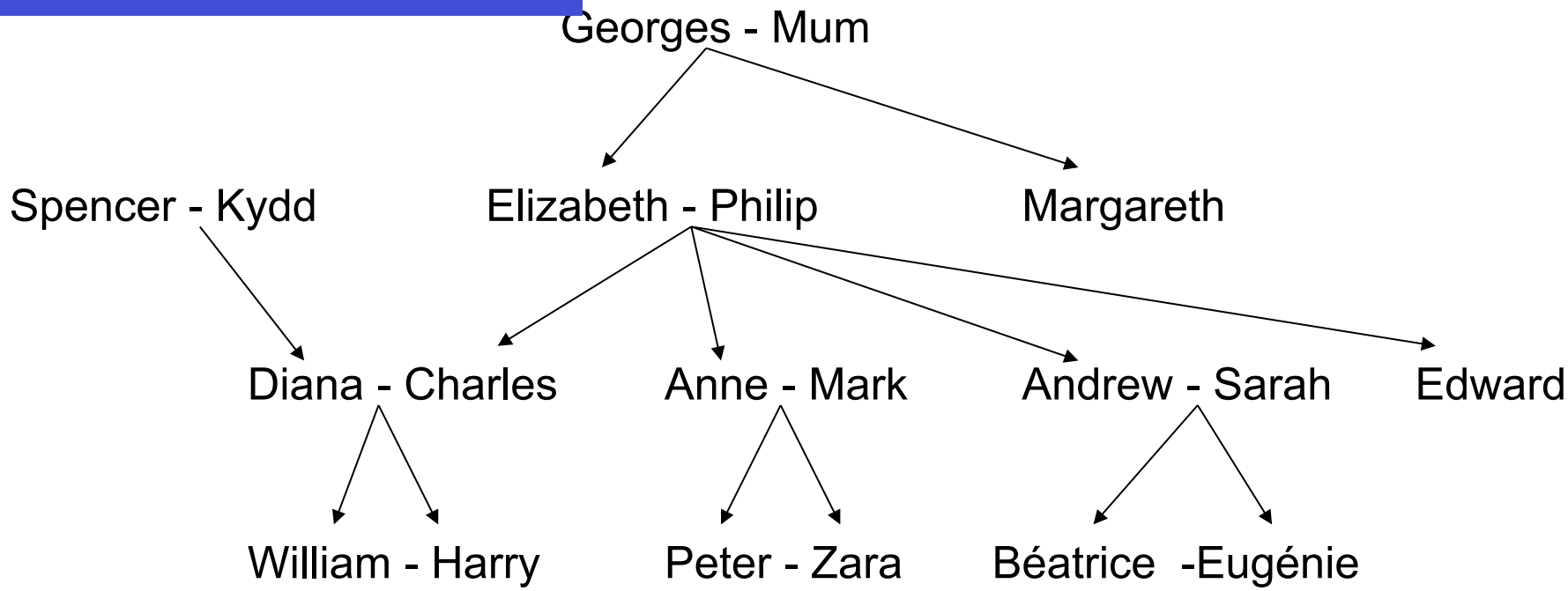
```
BK3 =
{père(philip,charles).
père(philip,anne). …
mère(mum,margareth).
mère(anne,peter). …

 parent(george,philip).
parent(mark,zara). … }
parent(X, Y) :- mere(X, Y).
parent(X, Y) :- pere(X, Y).
```

Exemples positifs E+ :

```
E+ = {grand-père(george,anne).
grand-père(philip,peter). etc. }
```

Exemples négatifs E- :

```
E- = {¬grand-père(george,elizabeth).
¬grand-père(harry,zara).
¬grand-père(george,george).
¬grand-père(margareth,charles).
¬grand-père(mum,eugénie). etc. }
```



L I P 6 C N R S

Fonctionnement de FOIL

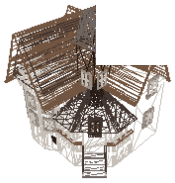
- FOIL commence avec une seule clause : la plus générale pour le prédicat grand-père.

  H_1 : { `grand-père(X,Y) ←` }

- Cette clause classe tous les exemples positifs correctement et tous les exemples négatifs (ce qui est incorrect).
- En effet, elle est équivalente à affirmer que `grand-père(X,Y)` est vrai quelles que soient les valeurs de X et Y !
- Cette clause est trop générale; c'est pourquoi il est nécessaire de la spécialiser
  - Pour cela, ou peut ajouter des conditions (prémisses) dans le corps de la clause.
  - FOIL parcourt tous les prédicats disponibles et les ajoute, un par un, au corps de la clause pour obtenir un ensemble de clauses possibles :

H_2 =

```
grand-père(X,Y) ← père(X,Y).
grand-père(X,Y) ← parent(X,Z).
grand-père(X,Y) ← père(X,Z).
etc.
```



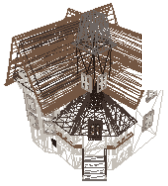
L I P 6 C N R S

Fonctionnement de FOIL: choix des spécialisations

```
                grand-père(X,Y) ← père(X,Y).
H2 =            grand-père(X,Y) ← parent(X,Z).
                grand-père(X,Y) ← père(X,Z).
                etc.
```

- **La première clause `grand-père(X,Y) ← père(X,Y).` ne couvre aucun exemple positif**
- **Les deux autres clauses couvrent tous les exemples positifs.**

  Par ex., grand-père(philip,margareth) est vrai s'il existe **un Z (charles par exemple) tel que père(philip,Z) est vrai. Il n'y a pas de relations entre les variables Y et Z, en tout cas pour l'instant.**
- **Par contre, elles couvrent (chacune) une partie des exemples négatifs.**
- **La 3e clause écarte plus d'exemples négatifs : elle est donc choisie pour la suite.**



```
H3 : { grand-père(X,Y) ← père(X,Z). }
```

Fonctionnement de FOIL: itération du processus

Connaissances a priori

```
BK3 = {père(philip,charles).
père(philip,anne). …
mère(mum,margareth).
mère(anne,peter). …
 parent(george,philip).
parent(mark,zara). … }
parent(X, Y) :- mere(X, Y).
parent(X, Y) :- pere(X, Y).
```

Exemples positifs E+ :

```
E+ = {grand-père(george,anne).
grand-père(philip,peter). etc. }
```

Exemples négatifs E- :

```
E- = {¬grand-père(george,elizabeth).
¬grand-père(harry,zara).
¬grand-père(george,george).
¬grand-père(margareth,charles). ¬grand-
père(mum,eugénie). etc. }
```

H_3 : { grand-père(X,Y) ← père(X,Z). }

Principe: on ne conserver que les clauses qui classent correctement les E^+ et rejettent les E^-.

Si des exemples positifs ne sont pas couverts, on ajoute une autre clause

- Ici, il convient d'ajouter un littéral pour rejeter les exemples où Z n'est pas un parent de Y :

H_4 : { grand-père(X,Y) ← père(X,Z), parent(Z,Y). }

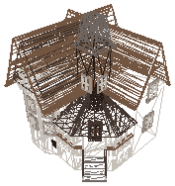
- Ce nouvel ensemble de clauses H_4 classe correctement toutes les instances. L'apprentissage est donc terminé.-

L I P 6 C N R S

Recherche engendrer et tester

- E, ensemble d'exemples positif E+ et négatif E-
- Connaissance implicite: BK

1. E_courant ← E
2. H ← ∅ ; *correspond à la queue de la clause*
3. Répéter
4. c ← concept permis
5. c_meilleur ← specialisation(c, E_courant)
6. Si c_meilleur ≠ echec Alors
7. H ← H ∪ c_meilleur
8. E_courant ← E_courant – couv(c_meilleur)
9. Fin si
10. Jusqu'à E_courant+ = ∅ ou c_meilleur = echec
11. Retourner H

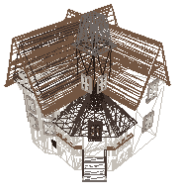


L I P 6 C N R S

Algorithme de spécialisation FOIL

- **Entrées:**
  - **clause courante c,**
  - **ensemble d'exemples E_courant**

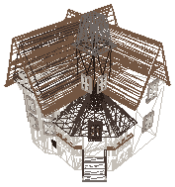
1. **Répéter**
2. **LCand** ← *{littéraux candidats à la spécialisation de c}*
3. **Si LCant** ≠ ∅ **Alors**
4. L_i ← *meilleur littéral de* **LCand** *à ajouter à c*
5. **C** ← **C** ∪ L_i
6. **E_courant** ← **{e∈E_courant | ∃θ, e = tete(c)θ et corps(c)θ ⊆ BK ∪ E_courant}**
7. **Fin si**
8. **Jusqu'à E_courant- =** ∅ **ou LCand =** ∅
9. **Retourner H**



Critères spécialisation FOIL

- Littéraux candidats à l'ajout:
  - symbole de prédicat appartient au langage des hypothèses
  - arguments de ce littéral contiennent au moins une variable apparaissant dans c – déterminisme
- Critère de gain d'information pondéré: FOIL compte les preuves pour chaque exemple plutôt que les exemples
  - c: clause à spécialiser, L littéral candidat
  - p_0: nombre de « substitutions positives » de c
  - n_0: nombre de « substitutions négatives » de c
  - p_1: nombre de « substitutions positives » de c ∪ L
  - n_1: nombre de « substitutions négatives » de c ∪ L
  - t: nombre de « substitutions positives » de c qui sont également des « substitutions positives » de c ∪ L

$$Gain_{FOIL}(c, L) = t \cdot \left[\log_2 \left(\frac{p_1}{p_1 + n_1} \right) - \log_2 \left(\frac{p_0}{p_0 + n_0} \right) \right]$$

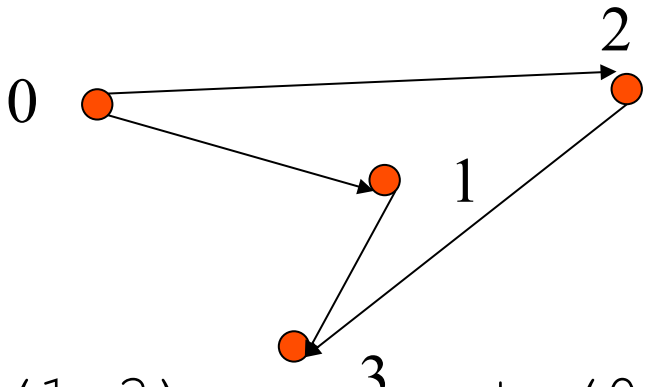


Exemple

LIP6 CNRS

- **Connaissance implicite**

```
arc(0,1).   arc(0,2).
arc(1,3).   arc(2,3).
```



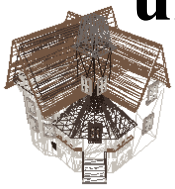
- **Exemples:**

(+):
```
connecte(0,1). connecte(1,3). connecte(0,2).
connecte(2,3). connecte(0,3).
```

(-):
```
connecte(0,0).
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```

But: construire le prédicat `connecte/2`

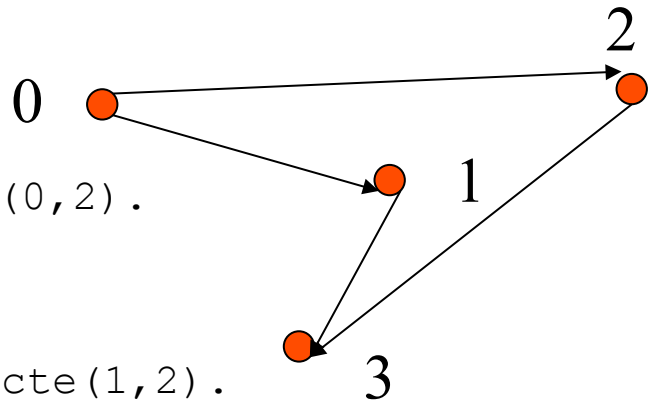
Biais de langage: la définition du prédicat contient uniquement le prédicat *`arc/2`* dans le corps de la clause



Exemple

- **Connaissance implicite**

```
arc(0,1).      arc(0,2).
arc(1,3).      arc(2,3).
```

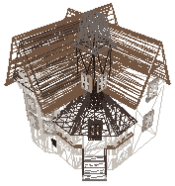


- **Exemples:**

(+):
```
connecte(0,1). connecte(1,3). connecte(0,2).
connecte(2,3). connecte(0,3).
```

(-):
```
connecte(0,0).
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```

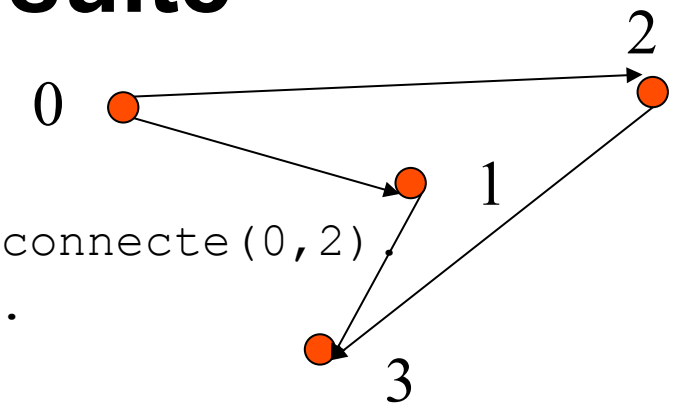
C	Li	Nbre+	Nbre-
connecte(X,Y)	arc(X, X)	0	0
	arc(Y, Y)	0	0
	arc(X, Y)	4	0
	arc(Y, X)	0	4
	arc(X, U)	8	9
	arc(Y, U)	2	13
	arc(U, X)	2	13
	arc(U, Y)	8	9



Exemple-suite

- **Connaissance implicite**

```
arc(0,1).    arc(0,2).
arc(1,3).    arc(2,3).
```



- **Exemples:**

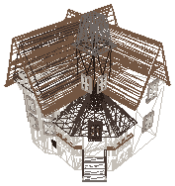
(+):
```
connecte(0,1).   connecte(1,3).   connecte(0,2).
connecte(2,3).   connecte(0,3).
```

(-):
```
connecte(0,0).
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```

Première clause apprise: `connecte(X, Y) :- arc(X, Y).`
On enlève les exemples positifs couverts par la clause, ce qui donne:

(+):
```
connecte(0,3).
```

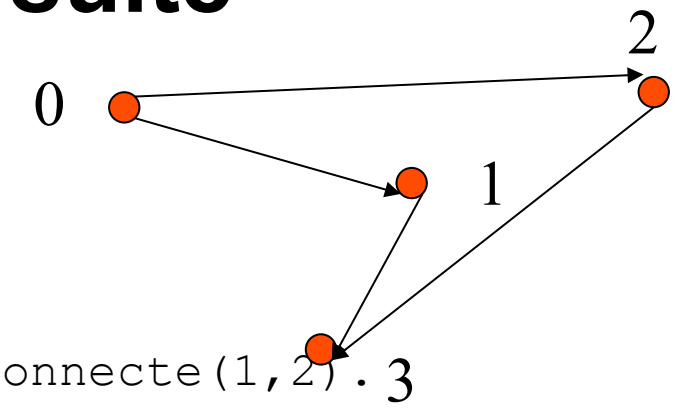
(-):
```
connecte(0,0).
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```



Exemple-suite

- **Connaissance implicite**

```
arc(0,1).    arc(0,2).
arc(1,3).    arc(2,3).
```



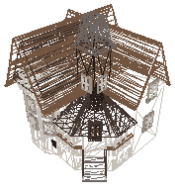
- **Exemples:**

(+): `connecte(0,3).`

(-): `connecte(0,0).`

```
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```

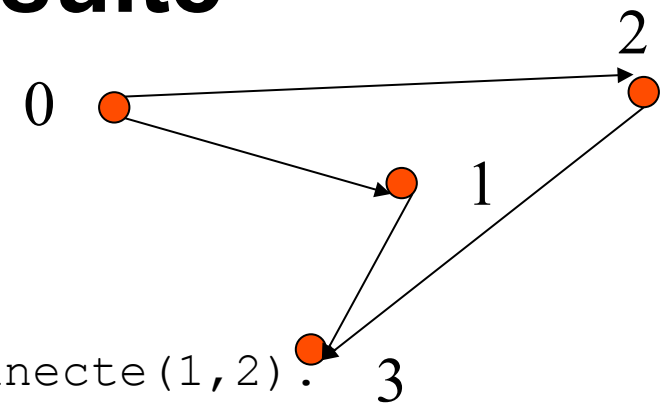
- **Une première clause a été apprise:** `connecte(X, Y):-arc(X, Y).`
- **Elle exclue tous les contre exemples, mais ne couvre pas tous les exemples**
- **Il faut rajouter une autre clause couvre l'exemple positif connecte(0, 3)**



Exemple-suite

- **Connaissance implicite**

```
arc(0,1).    arc(0,2).
arc(1,3).    arc(2,3).
```



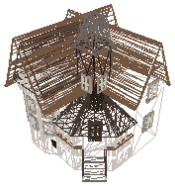
- **Exemples:**

(+): `connecte(0,3).`

(-): `connecte(0,0).`

```
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```

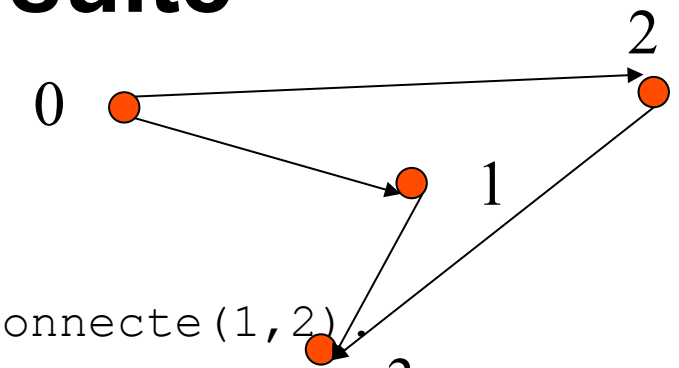
C connecte(X,Y)	Li	Nbre+	Nbre-
	arc(X, X)	0	0
	arc(Y, Y)	0	0
	arc(X, Y)	0	0
	arc(Y, X)	0	4
	arc(X, U)	2	9
	arc(Y, U)	0	13
	arc(U, X)	0	13
	arc(U, Y)	2	9



Exemple-suite

- **Connaissance implicite**

```
arc(0,1).    arc(0,2).
arc(1,3).    arc(2,3).
```



- **Exemples:**

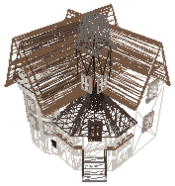
(+): `connecte(0,3).`

(-): `connecte(0,0).`

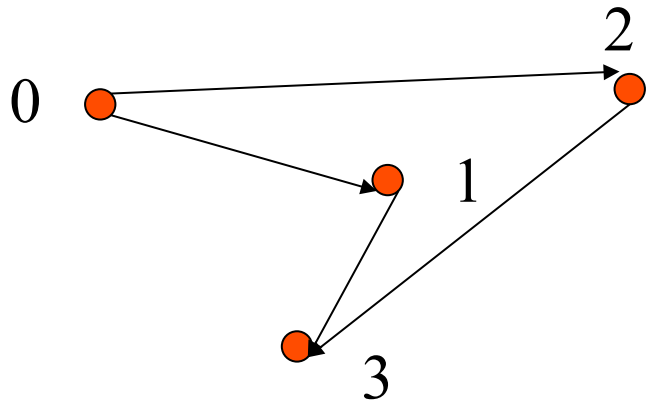
```
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```

- **Choix du littéral L_i:** `arc(X, U)`
- **La clause** `connecte(X, Y):-arc(X, U).` **couvre des contre exemples**
- **Elle doit être spécialisée**

C	Li	Nbre+	Nbre-
connecte(X,Y):-arc(X, U)	arc(X, X)	0	0
	arc(Y, Y)	0	0
	arc(X, Y)	0	0
	arc(Y, X)	0	0
	arc(X, U)	2	9
	arc(Y, U)	0	9
	arc(U, X)	0	0
	arc(U, Y)	2	0



Exemple-fin



- **Connaissance implicite**

```
arc(0,1).      arc(0,2).
arc(1,3).      arc(2,3).
```

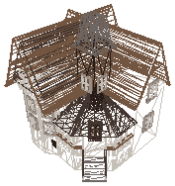
- **Exemples:**

(+): `connecte(0,3).`

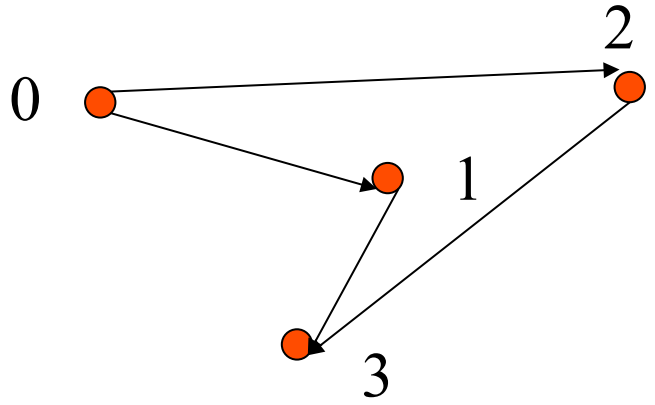
(-): `connecte(0,0).`

```
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3).
```

- **Choix du littéral L_i:** `arc(X, U)`
- **La clause** `connecte(X, Y):-arc(X, U)` **couvre des contre exemples**
- **Elle doit être spécialisée**
- **Choix du littéral de spécialisation:** `arc(U, Y)`
- **La clause** `connecte(X, Y):-arc(X, U), arc(U,Y)` **ne couvre plus de contre-exemples**



Exemple - récapitulation



- **Connaissance implicite**

```
arc(0,1).      arc(0,2).
arc(1,3).      arc(2,3).
```

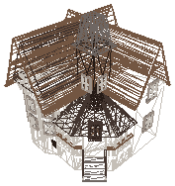
- **Exemples:**

(+): `connecte(0,1). connecte(1,3). connecte(0,2).`
`connecte(2,3). connecte(0,3).`

(-): `connecte(0,0).`
`connecte(1,0). connecte(1,1).connecte(1,2).`
`connecte(2,0). connecte(2,1).connecte(2,2).`
`connecte(3,0). connecte(3,1).connecte(3,2).`
`connecte(3,3).`

Programme appris:

```
connecte(X, Y) :- arc(X, Y).
connecte(X, Y) :- arc(X, U), arc(U, Y).
```

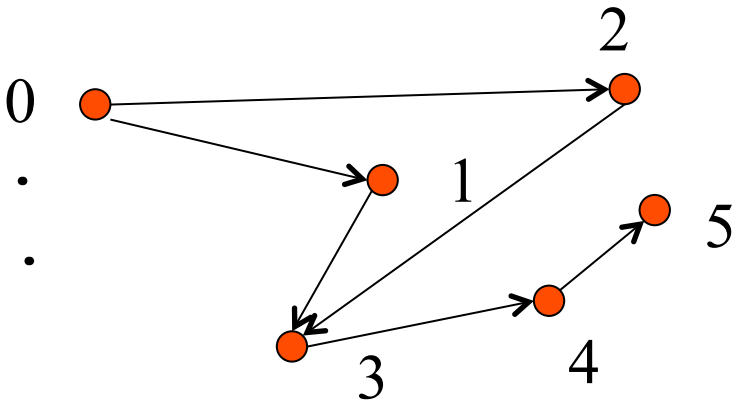


Exercise n°2

LIP6 CNRS

- **input:**
- **Background knowledge**

```
arc(0,1). arc(0,2). arc(1,3).
arc(2,3). arc(3,4). arc(4,5).
```



- **Examples:**

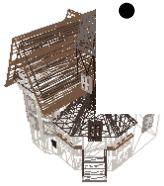
(+):

```
connecte(0,1). connecte(1,3). connecte(0,2).
connecte(2,3). connecte(0,3). connecte(1,4).
connecte(1,5). connecte(0,5). connecte(2,5).
```

(-):

```
connecte(0,0).
connecte(1,0). connecte(1,1).connecte(1,2).
connecte(2,0). connecte(2,1).connecte(2,2).
connecte(3,0). connecte(3,1).connecte(3,2).
connecte(3,3). connecte(5,2).connecte(5,1).
```

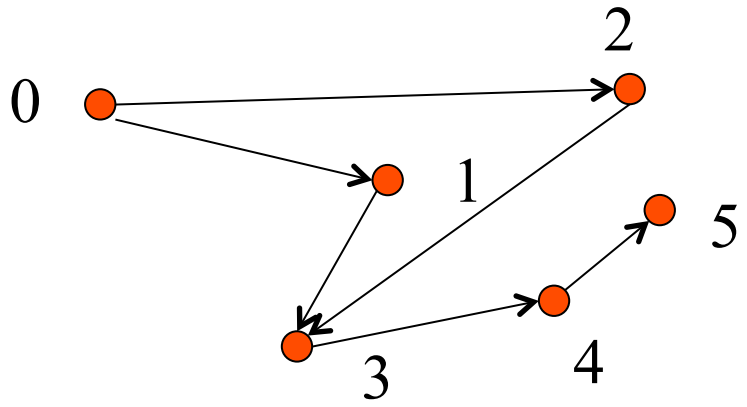
- **Goal: build the predicate `connecte/2`**
- **Language bias: the predicate contains only the *`arc/2`* and *`connecte/2`* predicates in its body**



New example

- **input:**
- **Background knowledge**

```
arc(0,1). arc(0,2). arc(1,3).
arc(2,3). arc(3,4). arc(4,5).
```

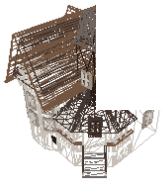


- **Examples:**

(+): `connecte(0,1). connecte(1,3). connecte(0,2). connecte(2,3). connecte(0,3). connecte(1,4). connecte(1,5). connecte(0,5). connecte(2,5).`

(-): `connecte(0,0). connecte(1,0). connecte(1,1). connecte(1,2). connecte(2,0). connecte(2,1). connecte(2,2). connecte(3,0). connecte(3,1). connecte(3,2). connecte(3,3). connecte(5,2). connecte(5,1).`

C	Li	Nbre+	Nbre-
connecte(X,Y)	arc(X, X)	0	0
	arc(Y, Y)	0	0
	arc(X, Y)	4	0
	arc(Y, X)	0	4
	arc(X, U)	x	ε
	arc(Y, U)	x	ε
	arc(U, X)	x	ε
	arc(U, Y)	x	ε

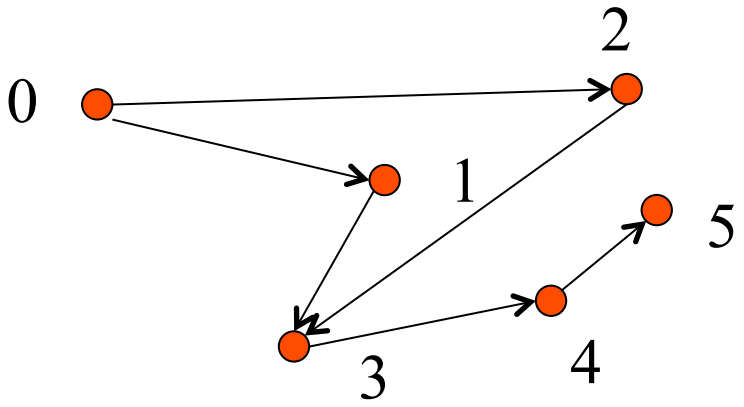


New example

LIP6 CNRS

- **input:**
- **Background knowledge**

```
arc(0,1). arc(0,2). arc(1,3).
arc(2,3). arc(3,4). arc(4,5).
```

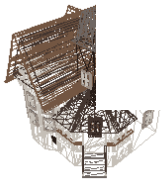


- **Examples:**

(+): `connecte(0,1). connecte(1,3). connecte(0,2). connecte(2,3). connecte(0,3). connecte(1,4). connecte(1,5). connecte(0,5). connecte(2,5).`

(-): `connecte(0,0). connecte(1,0). connecte(1,1). connecte(1,2). connecte(2,0). connecte(2,1). connecte(2,2). connecte(3,0). connecte(3,1). connecte(3,2). connecte(3,3). connecte(5,2). connecte(5,1).`

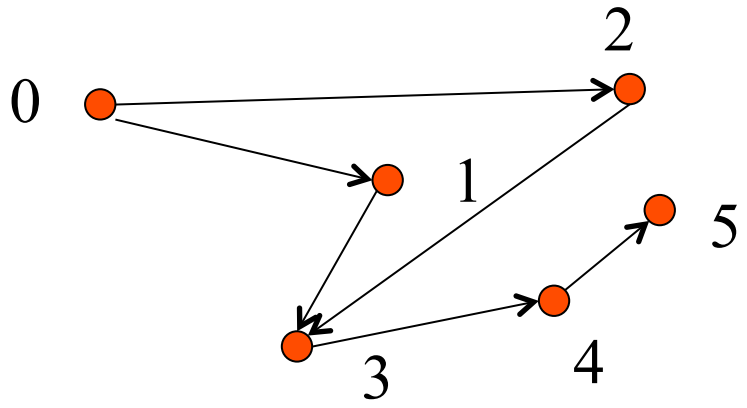
C	Li	Nbre+	Nbre-
connecte(X,Y)	connecte(X, X)	0	0
	connecte(Y, Y)	0	0
	connecte(X, Y)	0	0
	connecte(Y, X)	0	ε
	connecte(X, U)	0	ε
	connecte(Y, U)	0	ε
	connecte(U, X)	0	ε
	connecte(U, Y)	0	ε



New example

- **input:**
- **Background knowledge**

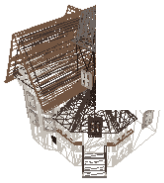
```
arc(0,1). arc(0,2). arc(1,3).
arc(2,3). arc(3,4). arc(4,5).
```



- **First learned clause:** `connecte(X, Y) :- arc(X, Y).`
- **We remove the positive examples that are already covered, which gives:**

(+): ~~`connecte(0,1). connecte(1,3). connecte(0,2). connecte(2,3).`~~ `connecte(0,3). connecte(1,4). connecte(1,5). connecte(0,5). connecte(2,5).`

(-): `connecte(0,0). connecte(1,0). connecte(1,1). connecte(1,2). connecte(2,0). connecte(2,1). connecte(2,2). connecte(3,0). connecte(3,1). connecte(3,2). connecte(3,3). connecte(5,2). connecte(5,1).`

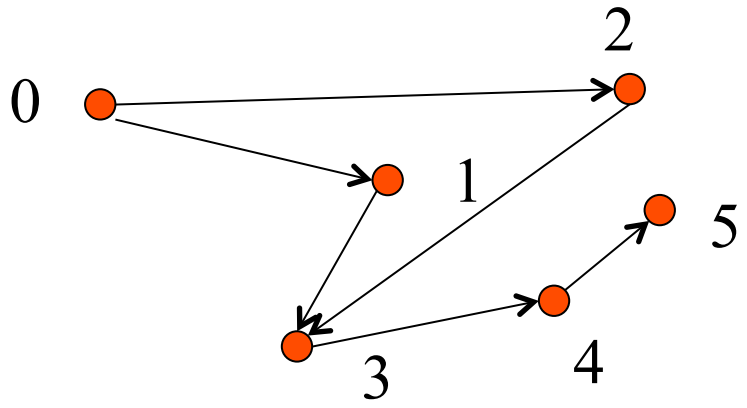


New example

LIP6 CNRS

- **input:**
- **Background knowledge**

```
arc(0,1). arc(0,2). arc(1,3).
arc(2,3). arc(3,4). arc(4,5).
```

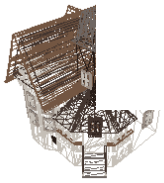


- **First learned clause:** `connecte(X, Y) :- arc(X, Y).`
- **We remove the positive examples that are already covered, which gives:**

(+): ~~`connecte(0,1). connecte(1,3). connecte(0,2). connecte(2,3).`~~ `connecte(0,3). connecte(1,4). connecte(1,5). connecte(0,5). connecte(2,5).`

(-): `connecte(0,0). connecte(1,0). connecte(1,1). connecte(1,2). connecte(2,0). connecte(2,1). connecte(2,2). connecte(3,0). connecte(3,1). connecte(3,2). connecte(3,3). connecte(5,2). connecte(5,1).`

Second learned clause: `connecte(X, Y) :- arc(X,Z), connecte(Z,Y).`



L I C N R S Autre exemple § appartient(N,L)

Exemples du concept

```
appartient(1,[1]).      appartient(2,[1,2]).    appartient(2,[1,2,3]).
appartient(2,[2]).      appartient(2,[2,3]).    appartient(3,[1,2,3]).
appartient(3,[3]).      appartient(3,[2,3]).
appartient(1,[1,2]).    appartient(1,
                        [1,2,3]).
```

Contre-exemples

```
¬appartient(1,[]).      ¬appartient(3,[]).      ¬appartient(1,[2,3]).
¬appartient(2,[]).      ¬appartient(1,[2]).     etc.
```

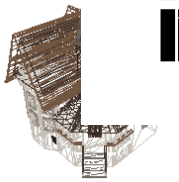
E^+

L'**univers** est limité

- aux entiers naturels 1, 2 et 3, et
- aux listes ordonnées formées à partir de ces entiers : [], [1], [2], [3], [1,2], [2,3] et [1,2,3].

Connaissance* a priori complémentaire :

- le prédicat **composant(L,H,T)** qui permet de déterminer à partir d'une liste L l'élément situé en tête H et la suite de la liste T.



Exemple § appartient(N,L)

- L'ensemble de clauses initial est :

H_1 = { appartient(X,Y) ← … }

- H_1 couvre tous les exemples, qu'ils soient positifs ou négatifs.
- L'extension de cette première clause comporte des exemples négatifs : il faut donc choisir un littéral à ajouter au corps de cette clause.
- On utilise le prédicat composant(U, V, W), puis on le spécialise (voir plus haut) en établissant un lien avec les variables X et Y

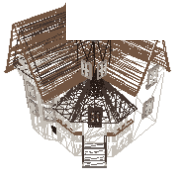
H_2 = { appartient(X,Y) ← composant(Y,X,Z)… }

- Cette nouvelle clause appartient(X,Y) ← composant(Y,X,Z). possède 3 variables et est vérifiée 6 affectations (interprétations):

```
X=1, Y=[1], Z=[].     X=3, Y=[3], Z=[].     X=2, Y=[2,3], Z=[3].
X=2, Y=[2], Z=[].     X=1, Y=[1,2], Z=[2].  X=1, Y=[1,2,3], Z=[2,3].
```

- Cette clause couvre bien des exemples positifs – et aucun exemple négatif – sur ces interprétations, mais il en manque

On rajoute donc une clause…



L I P 6

Exemple § appartient(N,L) (5)

- **La première clause** appartient(X,Y) ← composant(Y,X,Z). **ne couvrant pas** d'exemple négatif, **elle est cohérente; on peut construire une nouvelle clause :**

  H_3 = { appartient(X,Y) ← composant(Y,X,Z).
  appartient(X,Y) ← … }

- **On retire de E les exemples déjà couverts par la première clause :**

Exemples du concept

```
appartient(1,[1]).      appartient(2,[1,2]).    appartient(2,[1,2,3]).
appartient(2,[2]).      appartient(2,[2,3]).    appartient(3,[1,2,3]).
appartient(3,[3]).      appartient(3,[2,3]).
appartient(1,[1,2]).    appartient(1,
                        [1,2,3]).
```

Contre-exemples

```
¬appartient(1,[]).      ¬appartient(3,[]).      ¬appartient(1,[2,3]).
¬appartient(2,[]).      ¬appartient(1,[2]).     etc.
```

S

- **On décide d'ajouter le littéral** composant(Y,Z,W) **à la nouvelle clause :**

  H_4 = { appartient(X,Y) ← composant(Y,X,Z).
  appartient(X,Y) ← composant(Y,Z,W)… }

Exemple § appartient(N,L)

Exemples:

```
appartient(2,[1,2]).
appartient(3,[2,3]).
appartient(2,[1,2,3]).
appartient(3,[1,2,3]).
```

Contre-exemples:

```
¬appartient(1,[]).
¬appartient(2,[]).
¬appartient(3,[]).
etc.
```

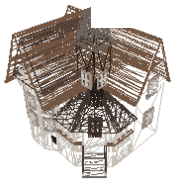
- **On décide d'ajouter le littéral composant(Y,Z,W) à la nouvelle clause :**

H_4 = { appartient(X,Y) ← composant(Y,X,Z).

appartient(X,Y) ← composant(Y,Z,W)… }

- **Avec (cette fois) 4 variables, les interprétations vérifiant cette clause partielle sont les suivantes :**

```
X=2,Y=[1,2],Z=1,W=[2].
X=3,Y=[2,3],Z=2,W=[3].
X=2,Y=[1,2,3],Z=1,W=[2,3].
X=3,Y=[1,2,3],Z=1,W=[2,3].
X=1,Y=[2],Z=2,W=[].
X=1,Y=[3],Z=3,W=[].
X=1,Y=[2,3],Z=2,W=[3].
X=2,Y=[1],Z=1,W=[].
X=2,Y=[3],Z=3,W=[].
X=3,Y=[1],Z=1,W=[].
X=3,Y=[2],Z=2,W=[].
X=3,Y=[1,2],Z=1,W=[2].
```



Exemple § appartient(N,L) (7)

- **Cette clause couvre encore des exemples négatifs, c'est pourquoi on choisi d'y ajouter un nouveau littéral :**

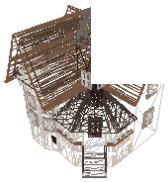
  H_5 = { appartient(X,Y) ← composant(Y,X,Z).

  appartient(X,Y) ← composant(Y,Z,W), appartient(X,W)… }

- **Les seules interprétations où cette clause est vérifiée sont les suivantes :**

```
X=2,Y=[1,2],Z=1,W=[2].
X=3,Y=[2,3],Z=2,W=[3].
X=2,Y=[1,2,3],Z=1,W=[2,3].
X=3,Y=[1,2,3],Z=1,W=[2,3].
```

- **Elles ne sont associées** qu'à **des exemples positifs : on peut donc fermer cette clause.**



Exemple § appartient(N,L)

Exemples du concept

```
appartient(1,[1]).      appartient(2,[1,2]).    appartient(2,[1,2,3]).
appartient(2,[2]).      appartient(2,[2,3]).    appartient(3,[1,2,3]).
appartient(3,[3]).      appartient(3,[2,3]).
appartient(1,[1,2]).    appartient(1,
                        [1,2,3]).
```

Contre-exemples

```
¬appartient(1,[]).      ¬appartient(3,[]).      ¬appartient(1,[2,3]).
¬appartient(2,[]).      ¬appartient(1,[2]).     etc.
```

Tous les exemples positifs sont à présent couverts par l'ensemble de clauses suivant :

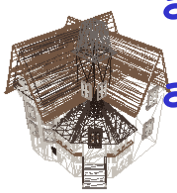
H_6 = { `appartient(X,Y) ← composant(Y,X,Z).`

`appartient(X,Y) ← composant(Y,Z,W), appartient(X,W).` }

H_6 est consistant : il couvre E^+ et rejette E^-.

En prolog, ce programme est équivalent à:

```
appartient(X, [X|Z]).

appartient(X, [Z|W]) :- appartient(X, W).
```



Limitation du nombre d'appariements

- **"Le principe du rasoir d' Occam" *(entropie d'information etc.)***
- **Contraintes syntaxiques (Inductive Logic Programming)**
  - **Determinate clause *Muggleton 1992***
  - **Relational cliché *Silverstein & Pazzani 1991***
  - **Stochastic bias *Sebag & Rouveirol 1997, Pompe & Al. 1996, Giordana & Al. 1997***
  - **List of modes *Muggleton & Srinivasan 1994***
  - **…**
- **Contraintes sémantiques**
  - **Notion de** moriologie*: **restriction des appariements aux parties qui appartiennent au même** morion*, **c'est-à-dire à la même partie fonctionnelle. *Zucker & Ganascia 96, 98***

Étape clef: subsomption

Définition: D **subsume** C si et seulement si $\exists\ \sigma$ tel que **$(D\sigma \subseteq C)$**, autrement dit tel que **$(D\sigma \Rightarrow C)$** pour toute instance**.** On le note $D \leq C$.
Notation: A et B étant deux clauses, $(A \subseteq B)$ signifie que les littéraux de la clause A se retrouvent tous dans la clause B

Rappel de l'algorithme:

1- Si D est vide fin.

2- Trouver

- un littéral L appartenant à D et

- une substitution θ

tels que $L\theta \in C$

Si cela s'avère impossible, alors, échec.

3- Remplacer D par $(D\theta - \{L\theta\})$ et retour en 2.

Coût de la subsomption

Rappel de l'algorithme:

1- Si D est vide fin.

2- Trouver

- un littéral L appartenant à D et

- une substitution θ

tels que $L\theta \in C$

Si cela s'avère impossible, alors, échec.

3- Remplacer D par ($D\theta$ - {$L\theta$ }) et retour en 2.

Trouver un littéral

Rappel de l'algorithme:

1- Si D est vide fin.

2- Trouver

- un littéral L appartenant à D et

- une substitution θ

tels que Lθ ∈ C

Si cela s'avère impossible, alors, échec.

3- Remplacer D par (Dθ - {Lθ }) et retour en 2.

Simplification 1: prendre le 1er littéral

Trouver une subsitution

Rappel de l'algorithme:

1- Si D est vide fin.

2- Trouver

- un littéral L appartenant à D et

- une substitution θ

tels que Lθ ∈ C

Si cela s'avère impossible, alors, échec.

3- Remplacer D par (Dθ - {Lθ }) et retour en 2.

```
p(X, Y, Z):-
q(X, U),
r(Y, V),
h(U, E),
m(E, R).
```

```
p(X, Y, Z):-
h(U, E),
m(E, R),
q(X, U),
r(Y, V),
```

.

Simplification 1: prendre le 1er littéral

Simplification 2: limiter le nombre d'instanciations → déterminisme

Chemin des variables

une seule variable nouvelle par littéral

Traduction dans le formalisme PLI

- **Réduction des appariements à un seul : notion de clause déterminée**

Définitions : **Un littéral B_i est** déterminé **dans une clause définie C: H$\leftarrow$ B_1, B_2, ..., B_k . Si pour toutes les substitutions σ qui unifient H avec quelque $o \in O$ tel que BK|—$(B_1 \wedge B_2 \wedge ... \wedge B_{i-1})\sigma$, il y a au plus une substitution θ telle que BK|— $B_i\sigma\theta$.**

Une clause est déterminée **si tous ses littéraux sont déterminés.**

- **Comment relâcher cette contrainte ?**
  - **FOCL *relational cliché* Silverstein & Pazzani 91**
  - **Stochastic bias (STILL Sebag & Rouveirol 97, SFOIL Pompe & Al. 96, REGAL Giordana & Al. 97)**
  - **PROGOL, lists of modes Muggleton & Srinivasan 94**

Bibliographie

- G. Plotkin. *Automatic Methods of Inductive Inference.* Ph.D. thesis, Edinburgh University (1971).

- J.R. Quinlan. Learning logical definitions from relations. In: *Machine Learning*, vol. 1 pp. 81-106 (1990).

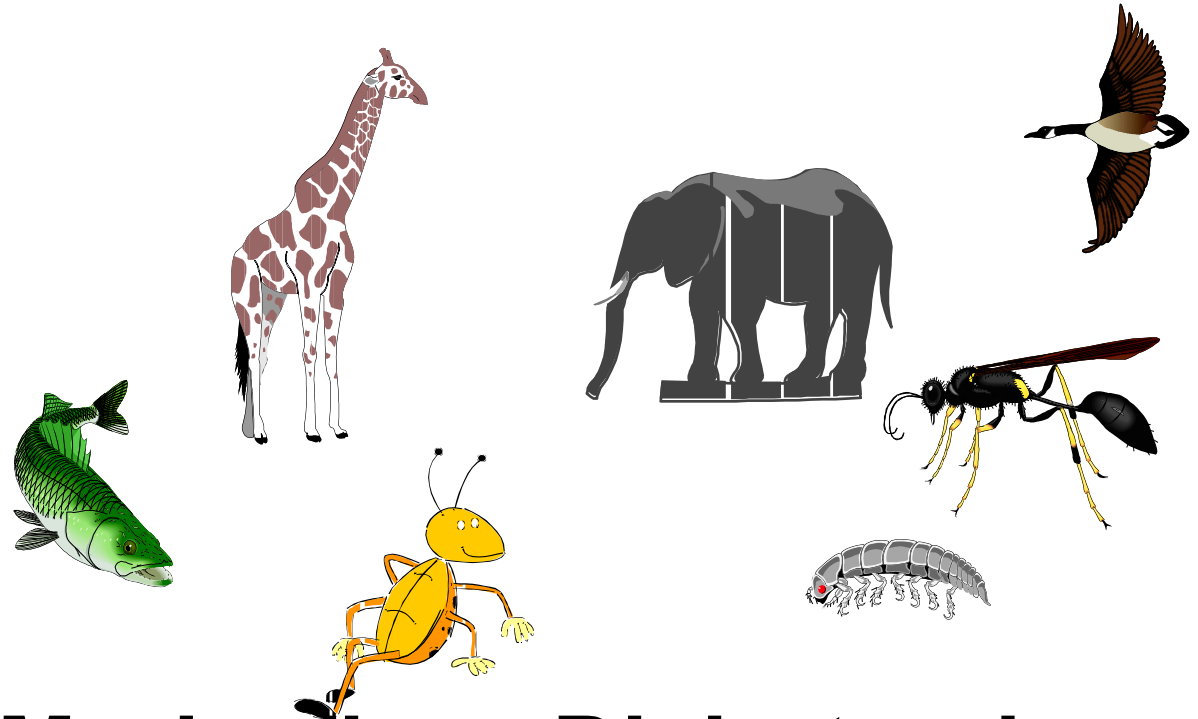
- S.H. Muggleton. Inductive Logic Programming. In: *New Generation Computing*, 8(4): pp. 295-318 (1991).

- J.R. Quinlan & R.M. Cameron-Jones. Induction of Logic Programs : FOIL and Related Systems. In: *Journal of New Generation Computing*, vol. 13, pp. 287-312 (1995).

FOIL pour linux : http://www.rulequest.com/Personal/

L I P 6 C N R S

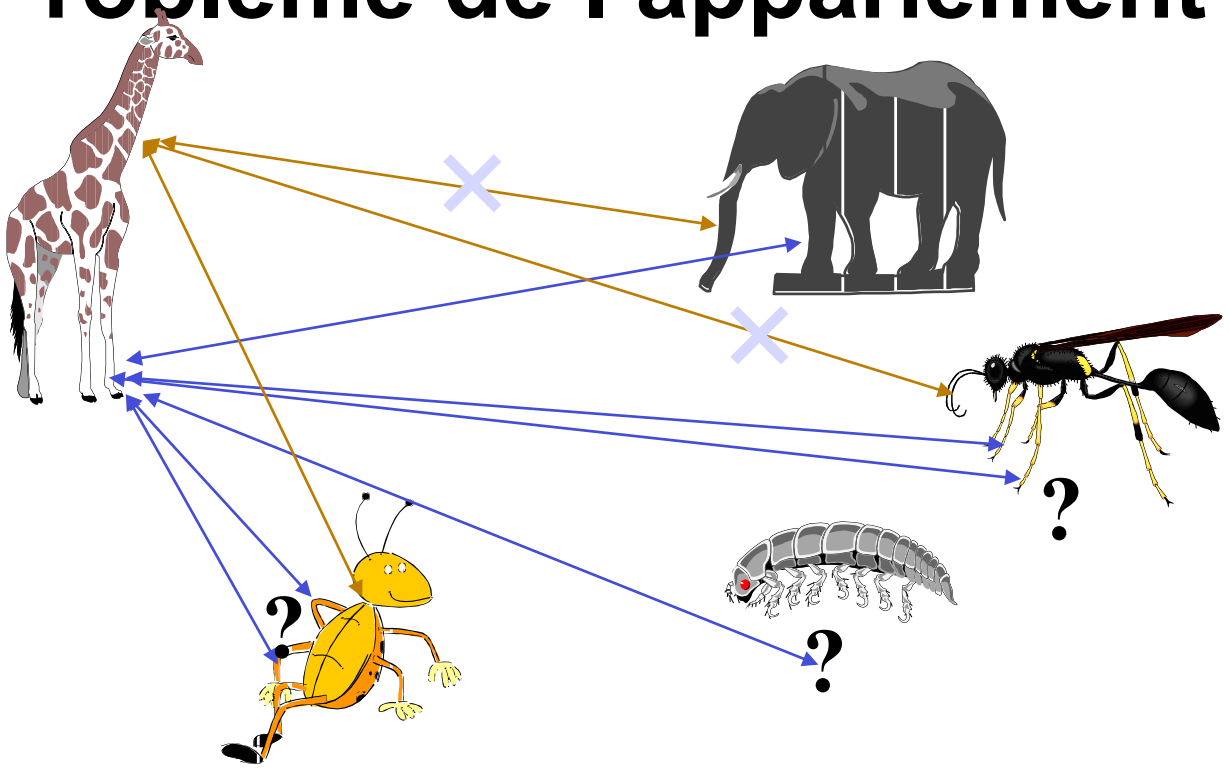
Source : Aristote
Zoologie
“*Parties des animaux, chap. 1*”



“Morion” vs. Dichotomie

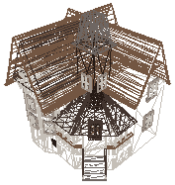
L I P 6 C N R S

Problème de l'appariement

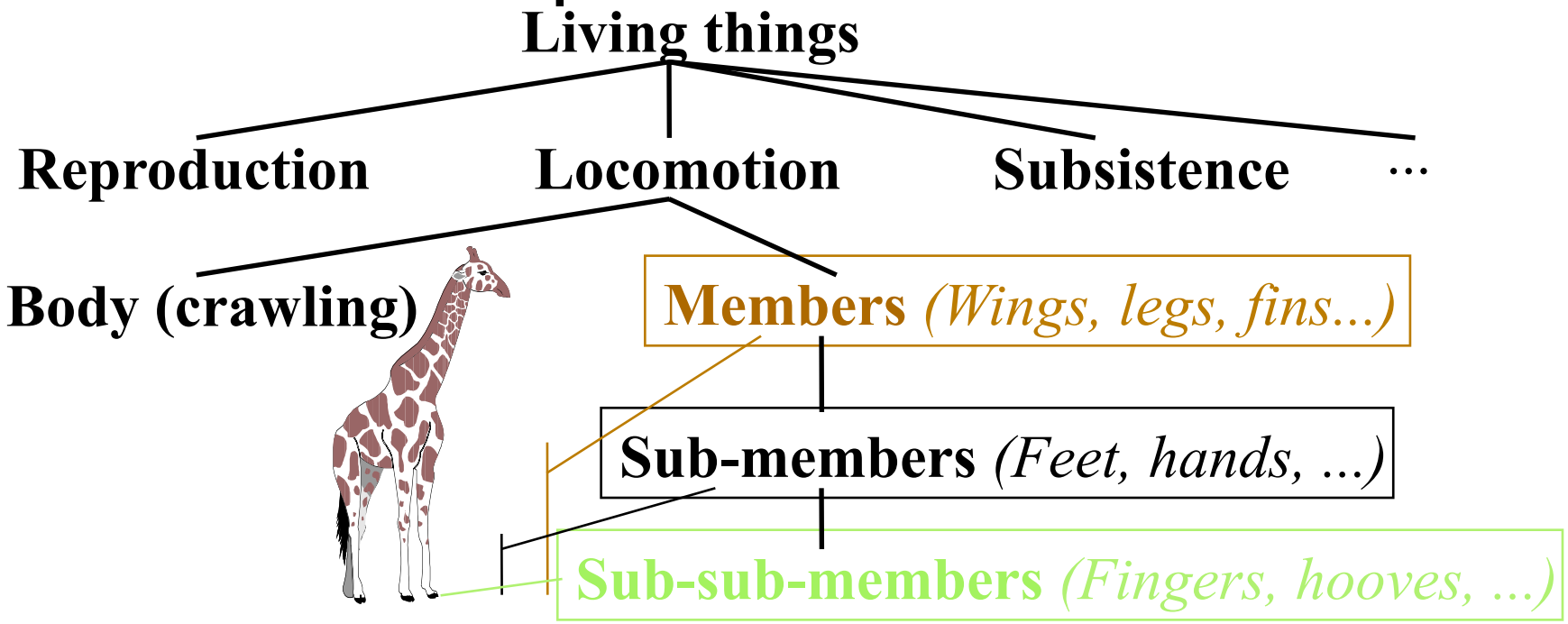


Comment limiter le nombre d'appariements ?
Un vieux problème, même en IA...

Interference Matching, 1978, Hayes-Roth & McDermott, THOT, 1979, Vere, AGAPE, 1983, Kodratoff & Ganascia, REMO, 1996, Zucker & Ganascia, ...



“Morion” = partie fonctionnelle

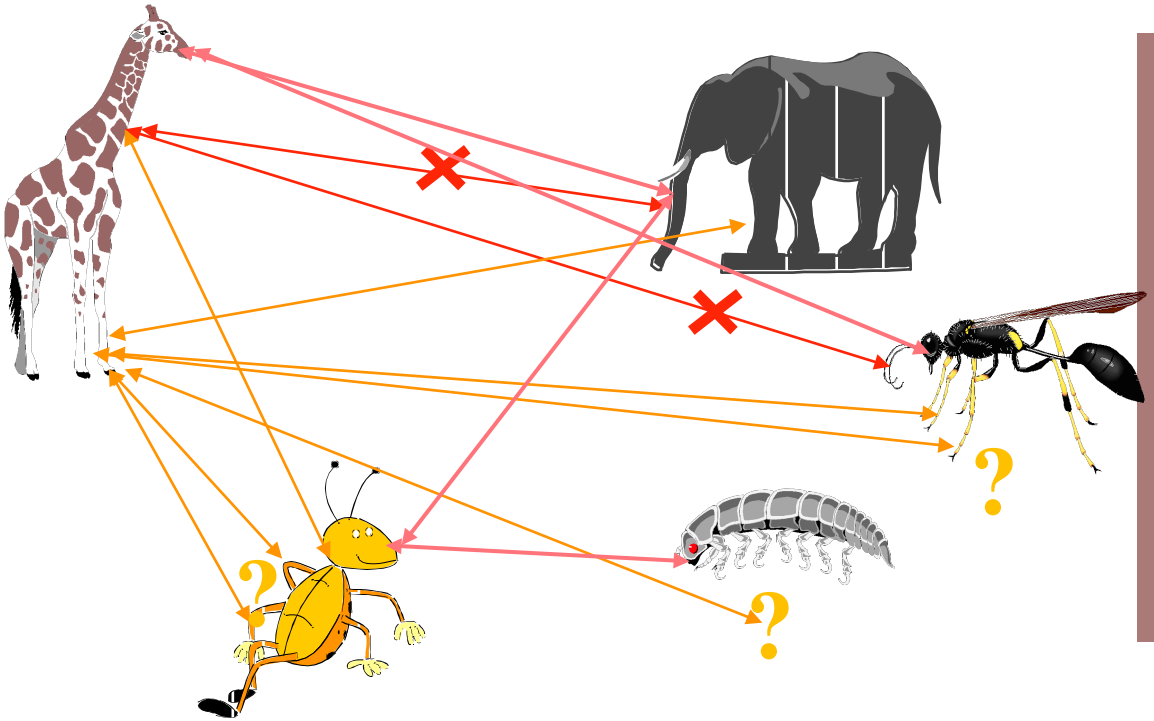


Restriction des appariements aux appariements entre objets appartenant au même motion, c’est-à-dire à la même partie ou sous-partie fonctionnelle.

Application aux caractères chinois : induction de règles phonétique à partir de 3285 caractères structurés.

情

Restriction des appariements



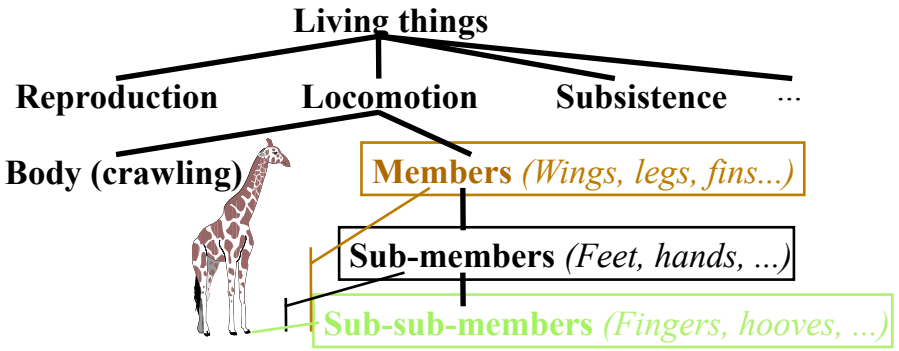
Application à la dérivation de règles phonétiques à partir de la structure de milliers de caractères [redacted]

Locomotion
Respiration
Appariements interdits

) **Parties fonctionnelles**

Quatre étapes

Zucker & Ganascia 96, Zucker 96, ...



0- Construire une « moriologie »

1- Choisir un « morion » (par exemple, couple de membres)

2- Reformuler les exemples de l'ensemble d'apprentissage : chaque appariement avec le morion choisi fournit un exemple d'apprentissage

(par exemple, 4 exemples de "couples de membres")

3- Apprentissage par un système de classification (CHARADE) .

4- Traduction dans le formalisme initial

Reformulation des exemples

- **Etape 1: choisir un morion (ex. couple de jambes)**

S: Animal(X) :- member(X, X1), member(X, X2),
sub-member(X1, X3).

- **Etape 2: ajout des propriétés eu égard à la connaissance implicite**

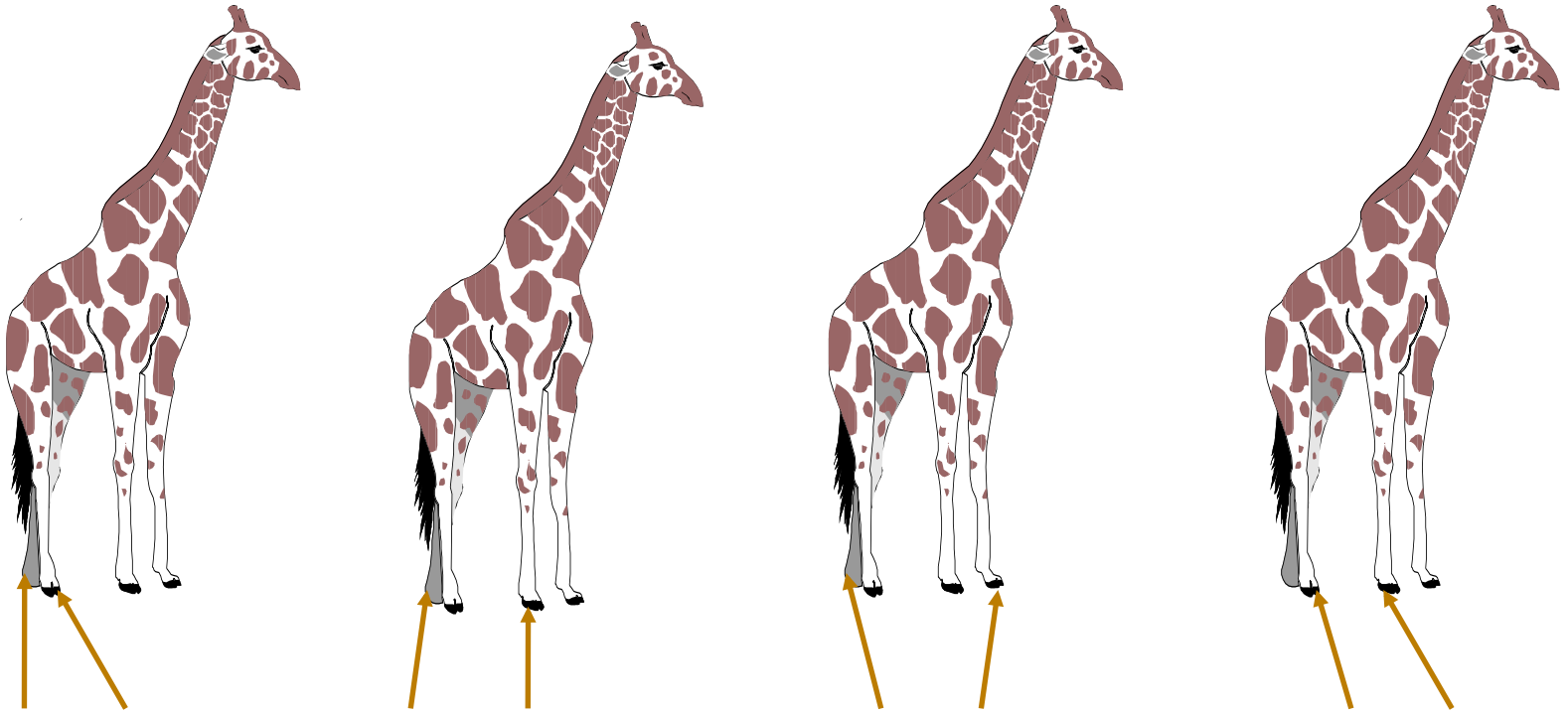
Cs: Animal(X) :- member(X, X1), type(X1,T1), size(X1,L1),
member(X, X2), type(X2, T2), size(X2,L2),
sub-member(X1, X3), type(X3, T3).

- **Etape 3: reformulation en logique propositionnelle**

Attributes for X1, LI, X2, L2, X3

Reformulation des observations

- **Etape 4: pour toutes les observations o et pour toutes les substitutions θ_i qui θ-subsument Cs, construire une instance d'apprentissage...**



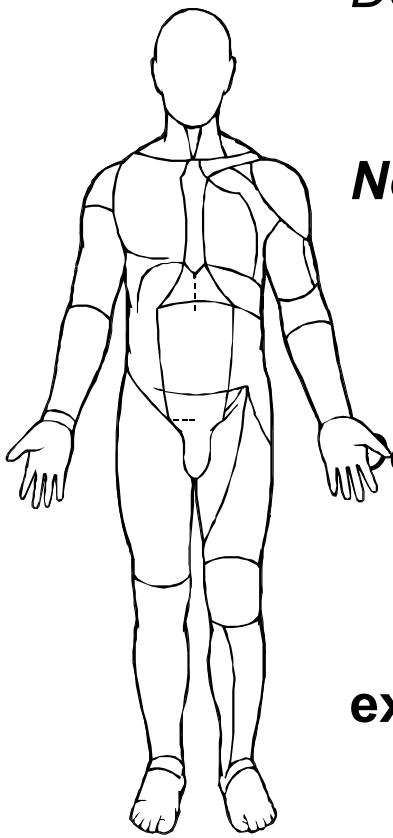
θ_1: X_1, X_2,
T_1= leg,...

θ_2: X_1, X_2,
T_1= leg,...

θ_3= X_1, X_2,
T_1= leg,...

θ_4= X_1, X_2,
T_1= leg,...

Clauses structurellement indérminées



Restriction aux clauses structurelles

Définition : **Un littéral P(X, Y) est un** littéral tructurel **ssi P(X, Y) signifie que X est dans une relation *composant/objet* (c-a-d moriologique) avec Y. Un littéral est dit** fonctionnel **sinon.**

***Note* toutes les relations *composant/objet* possèdent au moins trois propriétés axiomatisable ; elles sont anti-réflexives, antisymétriques et transitives.**

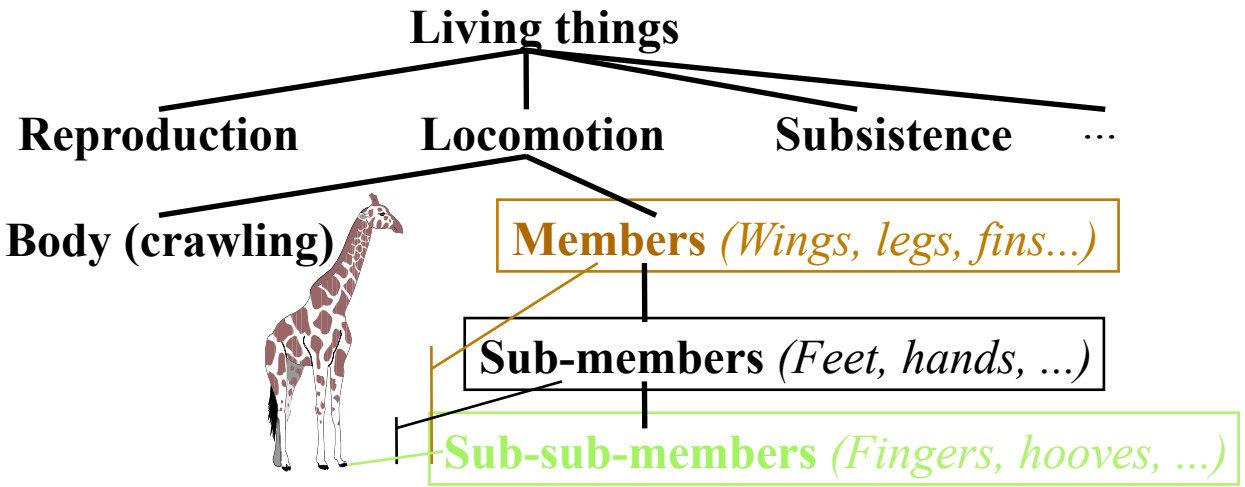
Définition : **Un ensemble de littéraux est** structurellement compatible **ssi leur fermeture transitive par la relation composant/objet ne contient pas de littéraux réflexif ou symétrique.**

ex. {member(X, X1), sub-member(X1, X2), member(X, X3)} est compatible tandis que {member(X, X1), sub-member(X1, X2), member(X2, X)} ne l'est pas

Clauses structurelles

Définition : **Une clause $H \leftarrow B_1, B_2, ..., B_k$. Est dite être une** clause structurelle **ssi $\{B_1, B_2, ..., B_k\}$ est un ensemble de littéraux structurés totalement compatibles**

Ceci signifie que le chemin des variables de la clause va en descendant dans la hiérarchie composant/objet, c-a-d dans la morionomie



S1 Animal(X) :- member(X, X1).

S2 Animal(X) :- member(X, X1), member(X, X2), sub-member(X1, X3).