Systèmes Experts

Richard Moot
Richard.Moot@labri.fr

Définition

- "un logiciel intelligent qui utilise des connaissances et des inférences logiques pour résoudre des problèmes qui sont suffisamment difficiles pour nécessiter une expertise humaine important pour trouver une solution" (Feigenbaum 1982)
- un système expert est un logiciel qui sait donner des recommandations — pour une domaine et une application bien défini — au même niveau d'un expert humain de ce domaine.

Systèmes Experts

Architecture

Architecture de systèmes experts

Base de connaissances, en formules logiques $(p \wedge q) \rightarrow r$
Système d'inférences, qui fait des conclusions grâce aux faits fournis par l'utilisateur et grâce aux connaissances dans la base.

Architecture des systèmes experts

- La base de données des connaissances ainsi que le système d'inférence sont fondés sur la logique et l'inférence logique,
- Alors, on a besoin de commencer par une petite introduction en logique et déduction automatique,
- On finira par voir comment ces principes donnent un langage de programmation: PROLOG, PROgrammation en LOGique.

Logique des Propositions Classique

- On coupe le monde en petit morceaux, des phrases simples comme "il fait beau" ou "le soleil gravite autour de la terre" qui sont soit vrai (indiqué par la valeur 1) soit faux (indiqué par la valeur 0).
- Comme abstraction, on remplace ces phrases par des lettres p,q,r,... qu'on appelle des propositions atomiques ou des formulas atomiques.

Logique des Propositions Classique — Syntaxe

1. Une proposition atomique est une proposition,
2. Si X est une proposition (pas nécessairement atomique) alors ¬X — "non X" ou "la négation de X" — est une proposition,

Logique des Propositions Classique — Syntaxe

3. Si X et Y sont des propositions alors,
   - 3.a. $(X \wedge Y)$ "X et Y"
   - 3.b. $(X \vee Y)$ "X ou Y"
   - 3.c. $(X \rightarrow Y)$ "si X alors Y"
   - 3.d. $(X \leftrightarrow Y)$ "X si et seulement si Y"

sont aussi des propositions

Logique des Propositions Classique — Syntaxe

4. Rien d'autre n'est une proposition.

Interprétation des Propositions Complexes

- L'interprétation d'un formule complexe dépend que de l'interprétation de ses sous-formules directs.
- Alors, étant donné une interprétation pour les sous-formules atomiques d'un formule X, on peut trouver l'interprétation X de façon très simple.

¬X	X
0	1
1	0

¬: négation logique

X∧Y	X	Y
0	0	0
0	0	1
0	1	0
1	1	1

∧: "et" logique

X∨Y	X	Y
0	0	0
1	0	1
1	1	0
1	1	1

∨: "ou" logique

X∨Y	X	Y
0	0	0
1	0	1
1	1	0
1	1	1

Remarque
le "ou" logique est inclusif, aux menus "dessert ou fromage" et "vin ou café" ne veulent pas dire "aux moins un des deux", mais plutôt "exactement un"

X→Y	X	Y
1	0	0
1	0	1
0	1	0
1	1	1

→: implication logique

X→Y	X	Y
1	0	0
1	0	1
0	1	0
1	1	1

L'implication logique est peut-être le connecteur le plus difficile à comprendre. Aucun causalité entre X et Y ne peut être déduit! X→Y est faux que quand X est vrai et Y est faux et vrai dans tous les autres cas.

X↔Y	X	Y
1	0	0
0	0	1
0	1	0
1	1	1

↔: "équivalence" logique

Digression

$v(\neg A) = 1 - v(A)$

$v(A \wedge B) = v(A) \cdot v(B)$

$v(A \vee B) = v(A) + v(B)$

X∧Y	X	Y
0	0	0
0	0	1
0	1	0
1	1	1

X∨Y	X	Y
0	0	0
1	0	1
1	1	0
1	1	1

Digression

$v(\neg A) = 1 - v(A)$

$v(A \wedge B) = v(A) \cdot v(B)$

$v(A \vee B) = \min(1, v(A) + v(B))$

X∧Y	X	Y
0	0	0
0	0	1
0	1	0
1	1	1

X∨Y	X	Y
0	0	0
1	0	1
1	1	0
1	1	1

Digression

$v(\neg A) = 1 - v(A)$

$v(A \wedge B) = \min(v(A), v(B))$

$v(A \vee B) = \max(v(A), v(B))$

X∧Y	X	Y
0	0	0
0	0	1
0	1	0
1	1	1

X∨Y	X	Y
0	0	0
1	0	1
1	1	0
1	1	1

Digression

$v(\neg A) = 1 - v(A)$

$v(A \wedge B) = \min(v(A), v(B))$

$v(A \vee B) = \max(v(A), v(B))$

Ceci semble d'être un moyen d'avoir des valeur de vérité entre 0 et 1.

Par exemple pour permettre à un expert de donner un dégrée de confiance à une proposition.

Digression

$v(\neg A) = 1 - v(A)$

$v(A \wedge B) = \min(v(A), v(B))$

$v(A \vee B) = \max(v(A), v(B))$

Par contre, cette solution n'est pas sans problèmes:

Supposons que notre expert croit, avec une confidence de 0.5, que p est vrai.

Alors, par la négation, la confidence en ¬p est aussi 0.5

Digression

$v(\neg A) = 1 - v(A)$

$v(A \wedge B) = \min(v(A), v(B))$

$v(A \vee B) = \max(v(A), v(B))$

Alors $v(p) = 0.5$ et $v(\neg p) = 0.5$

Maintenant, on est obligé de conclure que

$v(p \wedge \neg p) = 0.5$

$v(p \vee \neg p) = 0.5$

Pourquoi ça ne nous plaît pas trop?

Digression

- Quand on ajoute des probabilités, on préfère donner une probabilité 1 aux tautologies et une probabilité 0 aux contradictions.
- On verra dans quelques cours comment on peut ajouter des dégrées de confiance aux règles d'un tel façon que les logiciens et les statisticiens/probabilistes seront content.

Vérifier tout les valeurs de vérité

- Des fois on s'intéresse à savoir pour quels valeurs de vérité des sous-formules atomiques un formule complexe est vrai ou faux,
- S'il se trouve qu'un formule X n'est jamais vrai, on parle d'une contradiction
- Sinon on le formule X est valide,
- De plus, s'il se trouve X est toujours vrai, on parle d'un tautologie

Vérifier tout les valeurs de vérité

- Par contre, décider si un formule est valide, un tautologie ou une contradiction est difficile.
- Car, pour **a** formules atomiques, il y a 2^a combinaisons à vérifier.

Algorithme: comment construire un table de vérité pour une formule

1. On fait un table avec:
   - 1 ligne pour chaque combinaison de valeurs de vérité pour les formules atomique
   - 1 colonne pour chaque sous-formule de la formule.
2. On remplit les lignes et les colonnes correspondant aux formules atomiques en énumérant toutes les possibilités (il y en a 2^a)

Algorithme: comment construire un table de vérité pour un formule

3. Maintenant pour chaque formule X dont on a énuméré tout les possibilité de valeurs de ses sous-formules direct, on remplit la colonne qui correspond a X grâce à la définition du table de vérité pour ce connecteur.
4. Continuer jusqu'au moment où on a rempli tous les champs du table.

Algorithme: comment construire un table de vérité pour un formule

- A la fin, on vérifie la colonne qui correspond à la formule X: si on trouve que des 1, c'est un tautologie, si on trouve que des 0, c'est une contradiction, sinon la formule est consistent (mais pas un tautologie).
- De plus, pour des formules consistent on peut voir dans le tableaux pour quelles combinaisons de valeurs de vérité la formule est vrai et pour quelles elle est fausse.

Algorithme: comment construire un table de vérité pour un formule

- Si on s'intéresse juste à savoir si X est un tautologie, on peut travailler par ligne et s'arrêter avec échec dès qu'on trouve un 0 pour X: cette ligne est un contre-exemple.
- Pareil pour les contradictions, où on peut s'arrêter dès qu'on trouve un 1 pour X, ce qui démontre que la formule est valide. Ceci est un fameux problème NP complet.

tautologie

- L'exemple le plus simple possible: un proposition atomique, p, et une implication.
- p peut être soit faux (0) soit vrai (1).

p→p	p
	0
	1

tautologie

- Il nous reste à remplir la colonne p→p.
- On vérifie la table de vérité pour X→Y. Pour p→p on a juste deux cas.
  - X = 1 et Y = 1, qui donne 1
  - X = 0 et Y = 0, qui donne 1

p→p	p
	0
1	1

X→Y	X	Y
1	0	0
1	0	1
0	1	0
1	1	1

tautologie

- Il nous reste à remplir la colonne p→p.
- On vérifie la table de vérité pour X→Y. Pour p→p on a juste deux cas.
  - X = 1 et Y = 1, qui donne 1
  - X = 0 et Y = 0, qui donne 1

p→p	p
1	0
1	1

X→Y	X	Y
1	0	0
1	0	1
0	1	0
1	1	1

tautologie

- Il nous reste à remplir la colonne p→p.
- On vérifie la table de vérité pour X→Y. Pour p→p on a juste deux cas.
  - X = 1 et Y = 1, qui donne 1
  - X = 0 et Y = 0, qui donne 1
- Alors on est terminé et p→p est un tautologie

p→p	p
1	0
1	1

X→Y	X	Y
1	0	0
1	0	1
0	1	0
1	1	1

tautologie

- Encore une exemple, cette fois avec négation et disjonction: p∨¬p
- On commence comme avant, énumérant les possibilités pour p et en faisant un colonne pour chaque sous-formule: p, ¬p et p∨¬p

p∨¬p	¬p	p
		0
		1

tautologie

- La formule dont on sait les possibilités de ses sous-formules est ¬p (car on vient de traiter p)
- Grâce au table de vérité pour la négation on remplit la colonne ¬p

¬X	X
0	1
1	0

p∨¬p	¬p	p
	1	0
	0	1

tautologie

- Maintenant on sait les valeur de vérité pour les deux sous-formules de p∨¬p: p et ¬p
- En utilisant la table on trouve 1∨0 = 1 et 0∨1 = 1
- Alors p∨¬p est un tautologie

X∨Y	X	Y
0	0	0
1	0	1
1	1	0
1	1	1

p∨¬p	¬p	p
1	1	0
1	0	1

Désavantages des tableaux de vérité

- L'algorithme pour des tableaux de vérité est très simple et direct, mais n'est pas très efficace à se rendre compte quand on n'a pas besoin d'énumérer toutes les valeurs de vérité.
- Par exemple, pour une formule (p∧q)→r, si r est vrai, la formule est vrai pour tout les valeurs de vérité de p et q. (Pourquoi?)

Désavantages des tableaux de vérité

- On verra deux de ces algorithmes: l'algorithme de Quine et la résolution.
- Des variations sur ces deux algorithmes sont très fréquemment utilisés pour la recherche automatique des démonstrations.
- Par exemple, Prolog utilise une version particulière de la résolution.

Algorithme de Quine

- les constants T,⊥
- substitution des formules
- équivalences
- l'algorithme

Algorithme de Quine

Les constants T et ⊥

- T, "vrai", est un constante qui a toujours la valeur de vérité 1
- ⊥, "faux", est un constante qui a toujours la valeur de vérité 0

Substitution

Lemme: Substitution

Soit
X un formule,
Y un des sous-formules de X,
Z un formule tel que Y↔Z est un tautologie,

alors quand on remplace toutes les occurrences de Y dans X par Z, dénoté par X[Y := Z], X est équivalent à X[Y := Z], c'est à dire

$$X \leftrightarrow X[Y := Z]$$

Substitution

- Le lemme de substitution dit qu'on peut toujours remplacer une sous-formule par une autre qui a toujours la même valeur de vérité.
- Le sens de formules, dans notre logique classique très simple, est donné par leur valeur de vérité. Alors on ne peut pas distinguer deux formules qui ont toujours la même valeur de vérité.

Algorithme de Quine Equivalences

Définitions de → et ↔

- $(X \rightarrow Y) \equiv (\neg X \vee Y)$
- $(X \leftrightarrow Y) \equiv ((X \rightarrow Y) \wedge (Y \rightarrow X))$

Algorithme de Quine
Equivalences

Définitions de $\rightarrow$ et $\leftrightarrow$

- $(X \rightarrow Y) \equiv (\neg X \vee Y)$
- $(X \leftrightarrow Y) \equiv ((X \rightarrow Y) \wedge (Y \rightarrow X)) \equiv ((\neg X \vee Y) \wedge (\neg Y \vee X))$

Algorithme de Quine
Equivalences

Commutativité de $\wedge$ et $\vee$

- $(X \wedge Y) \equiv (Y \wedge X)$
- $(X \vee Y) \equiv (Y \vee X)$

Algorithme de Quine
Equivalences

Associativité de $\wedge$ et $\vee$

- $((X \wedge Y) \wedge Z) \equiv (X \wedge (Y \wedge Z))$
- $((X \vee Y) \vee Z) \equiv (X \vee (Y \vee Z))$

Algorithme de Quine
Equivalences

Distributivité de $\wedge$ et $\vee$

- $((X \wedge Y) \vee Z) \equiv ((X \vee Z) \wedge (Y \vee Z))$
- $((X \vee Y) \wedge Z) \equiv ((X \wedge Z) \vee (Y \wedge Z))$

Algorithme de Quine Equivalences

Règles de De Morgan

- $\neg T \equiv \bot$
- $\neg \bot \equiv T$
- $\neg\neg X \equiv X$
- $\neg(X \wedge Y) \equiv (\neg X \vee \neg Y)$
- $\neg(X \vee Y) \equiv (\neg X \wedge \neg Y)$

Algorithme de Quine Equivalences

Simplifications de Quine

- $(T \rightarrow X) \equiv X$
- $(X \rightarrow T) \equiv T$
- $(\bot \rightarrow X) \equiv T$
- $(X \rightarrow \bot) \equiv \neg X$

Algorithme de Quine Equivalences

Simplifications de Quine

- $(T \vee X) \equiv T$
- $(X \vee T) \equiv T$
- $(\bot \vee X) \equiv X$
- $(X \vee \bot) \equiv X$

Algorithme de Quine Equivalences

Simplifications de Quine

- $(T \wedge X) \equiv X$
- $(X \wedge T) \equiv X$
- $(\bot \wedge X) \equiv \bot$
- $(X \wedge \bot) \equiv \bot$

Algorithme de Quine

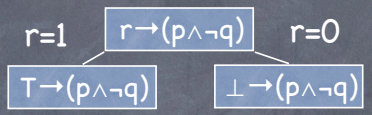
L'algorithme

1. Initialiser par l'arbre simple qui contient que la formule a vérifier comme feuille (et racine).
2. Applique les simplifications, ajoutant une feuille pour chaque simplification. Continue jusqu'au moment qu'aucune simplification ne peut être fait.
3. Trouve un feuille qui contient un formule atomique, disons X avec sous-formule atomique p. Ajout deux feuilles X[p := T] et X[p :=⊥]. Continue avec 2.

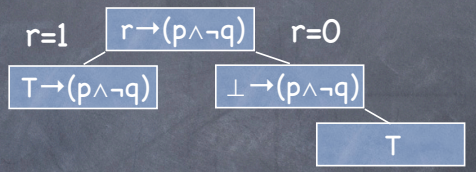
Exemple de l'algorithme de Quine

$r \to (p \wedge \neg q)$

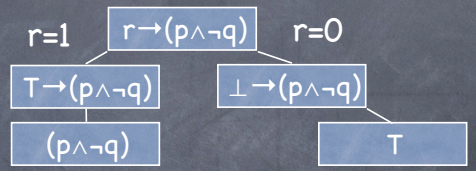
Exemple de l'algorithme de Quine



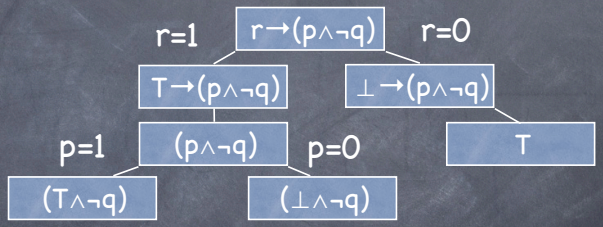
Exemple de l'algorithme de Quine



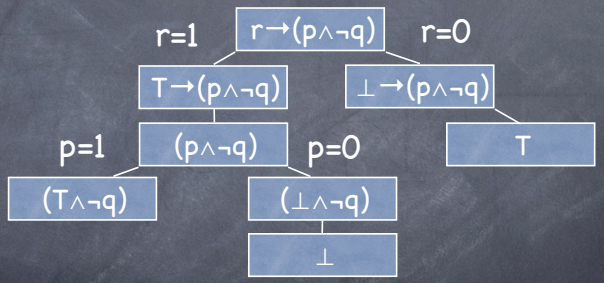
Exemple de l'algorithme de Quine



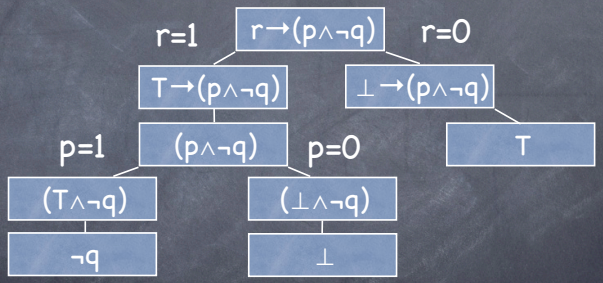
Exemple de l'algorithme de Quine



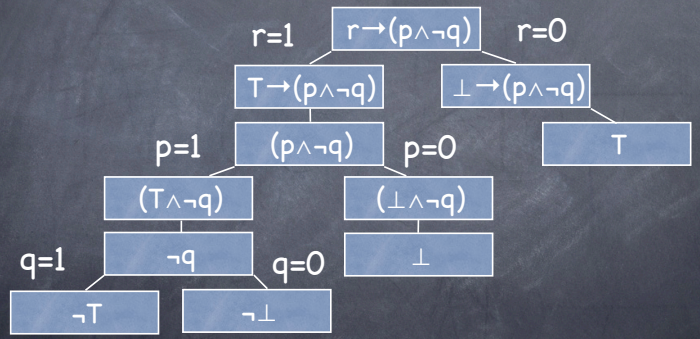
Exemple de l'algorithme de Quine



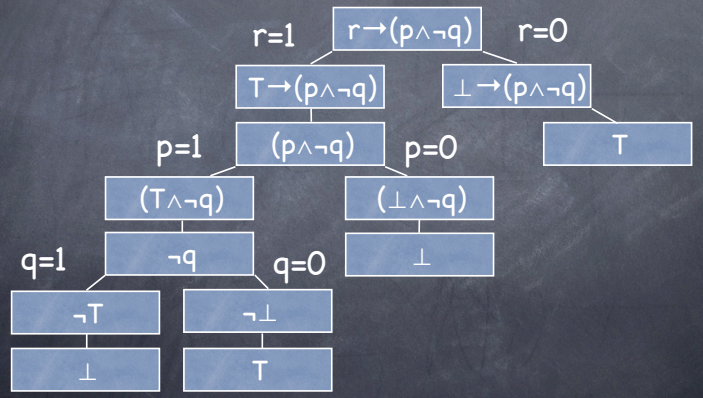
Exemple de l'algorithme de Quine



Exemple de l'algorithme de Quine



Exemple de l'algorithme de Quine



Un exemple plus complexe

H1: S'il est venu seul,
il a pris le bus ou le train.

H2: S'il a pris le bus ou son automobile,
alors il est arrivé en retard et
a manqué la réunion.

H3: Il n'est pas arrivé en retard.

C: Donc, s'il est venu seul, il a pris le train

((H1∧H2)∧H3)→C

Un exemple plus complexe

H1: S'il est venu seul,
il a pris le bus ou le train.

H1 = s→(b∨t)

((H1∧H2)∧H3)→C

Un exemple plus complexe

H2: S'il a pris le bus ou son automobile, alors il est arrivé en retard et a manqué la réunion.

$H2 = (b \vee a) \rightarrow (r \wedge m)$

$((H1 \wedge H2) \wedge H3) \rightarrow C$

Un exemple plus complexe

$H3 = \neg r$

H3: Il n'est pas arrivé en retard.

$((H1 \wedge H2) \wedge H3) \rightarrow C$

Un exemple plus complexe

$C = s \rightarrow t$

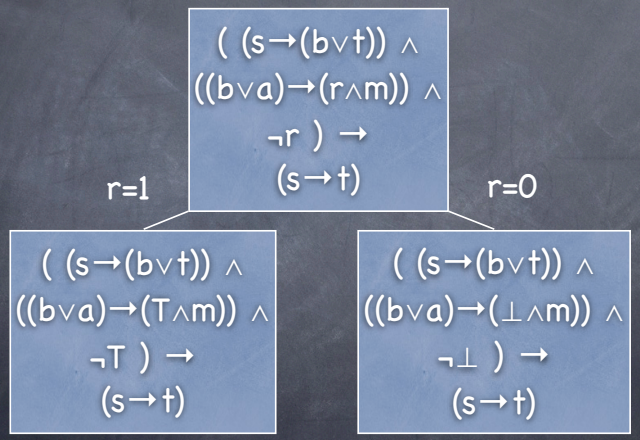
C: Donc, s'il est venu seul, il a pris le train

$((H1 \wedge H2) \wedge H3) \rightarrow C$

Un exemple plus complexe

$$((s \rightarrow (b \vee t)) \wedge ((b \vee a) \rightarrow (r \wedge m)) \wedge \neg r) \rightarrow (s \rightarrow t)$$

Un exemple plus complexe



Un exemple plus complexe

((s→(b∨t)) ∧
((b∨a)→(T∧m)) ∧
¬T) →
(s→t)

Un exemple plus complexe

((s→(b∨t)) ∧
((b∨a)→(T∧m)) ∧
¬T) →
(s→t)

Un exemple plus complexe

((s→(b∨t)) ∧
((b∨a)→(T∧m)) ∧
⊥) →
(s→t)

Un exemple plus complexe

$$(\underline{(s \to (b \vee t)) \wedge \bot}) \to (s \to t)$$

Un exemple plus complexe

$$\underline{\bot \to} \; (s \to t)$$

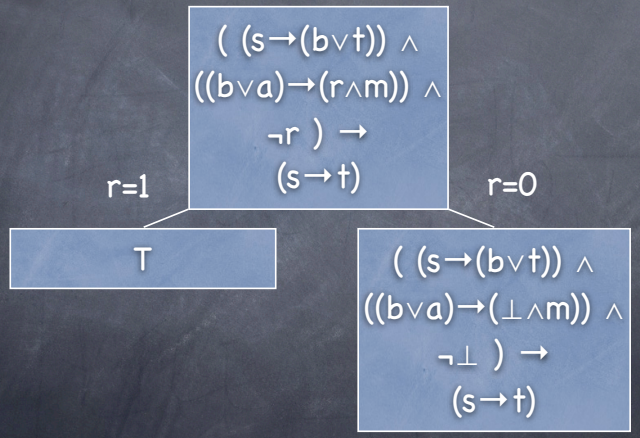
Un exemple plus complexe

$$\underline{\bot \to} \; \underline{(s \to t)}$$

Un exemple plus complexe

$$\underline{\top}$$

Un exemple plus complexe



Un exemple plus complexe

((s→(b∨t)) ∧
((b∨a)→(⊥∧m)) ∧
¬⊥) →
(s→t)

Un exemple plus complexe

((s→(b∨t)) ∧
((b∨a)→(⊥∧m)) ∧
T) →
(s→t)

Un exemple plus complexe

((s→(b∨t)) ∧
((b∨a)→(⊥∧m)))
→
(s→t)

Un exemple plus complexe

$$((s \rightarrow (b \vee t)) \wedge ((b \vee a) \rightarrow \bot)) \rightarrow (s \rightarrow t)$$

Un exemple plus complexe

$$((s \rightarrow (b \vee t)) \wedge ((b \vee a) \rightarrow \bot)) \rightarrow (s \rightarrow t)$$

Un exemple plus complexe

$$((s \rightarrow (b \vee t)) \wedge \neg(b \vee a)) \rightarrow (s \rightarrow t)$$

Un exemple plus complexe

$$((s \rightarrow (b \vee t)) \wedge (\neg b \wedge \neg a)) \rightarrow (s \rightarrow t)$$

Un exemple plus complexe

$$((s \rightarrow (b \vee t)) \wedge (\neg b \wedge \neg a)) \rightarrow (s \rightarrow t)$$

Un exemple plus complexe

$$((s \rightarrow (b \vee t)) \wedge (\neg b \wedge \neg a)) \rightarrow (s \rightarrow t)$$

b=1 b=0

$$((s \rightarrow (T \vee t)) \wedge (\neg T \wedge \neg a)) \rightarrow (s \rightarrow t)$$

$$((s \rightarrow (\bot \vee t)) \wedge (\neg \bot \wedge \neg a)) \rightarrow (s \rightarrow t)$$

Un exemple plus complexe

$$((s \rightarrow (b \vee t)) \wedge (\neg b \wedge \neg a)) \rightarrow (s \rightarrow t)$$

b=1 b=0

$$((s \rightarrow (T \vee t)) \wedge (\underline{\neg T} \wedge \neg a)) \rightarrow (s \rightarrow t)$$

$$((s \rightarrow (\bot \vee t)) \wedge (\neg \bot \wedge \neg a)) \rightarrow (s \rightarrow t)$$

Un exemple plus complexe

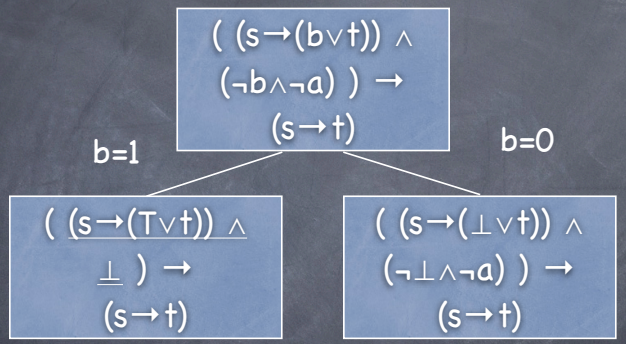
$$((s \rightarrow (b \vee t)) \wedge (\neg b \wedge \neg a)) \rightarrow (s \rightarrow t)$$

b=1 b=0

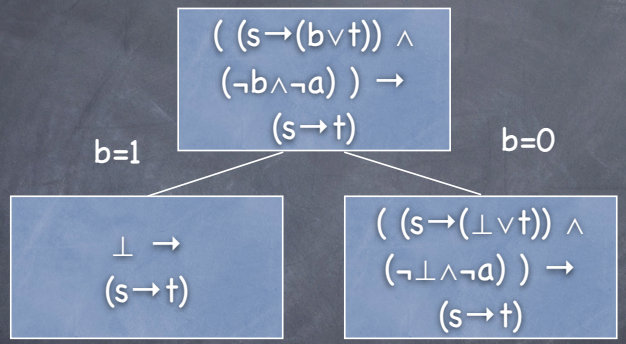
$$((s \rightarrow (T \vee t)) \wedge (\underline{\bot \wedge \neg a})) \rightarrow (s \rightarrow t)$$

$$((s \rightarrow (\bot \vee t)) \wedge (\neg \bot \wedge \neg a)) \rightarrow (s \rightarrow t)$$

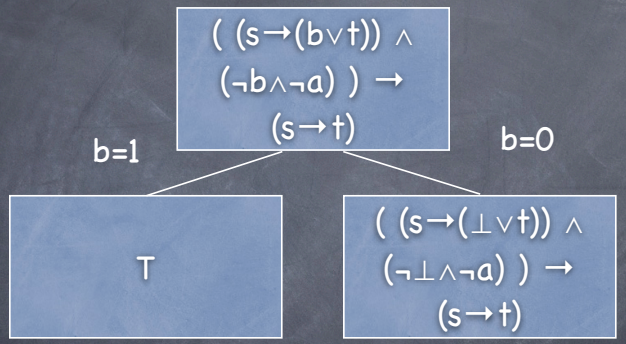
Un exemple plus complexe



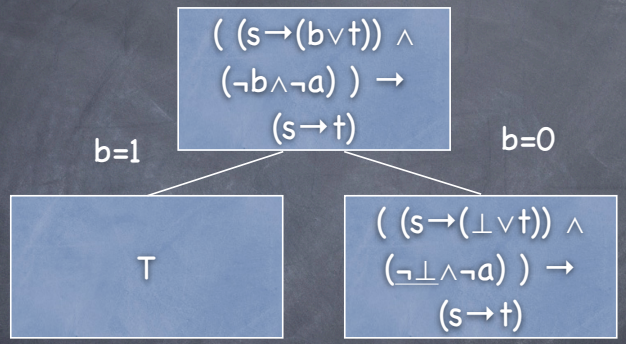
Un exemple plus complexe



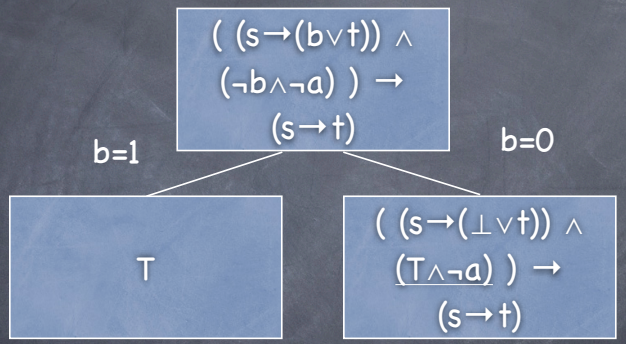
Un exemple plus complexe



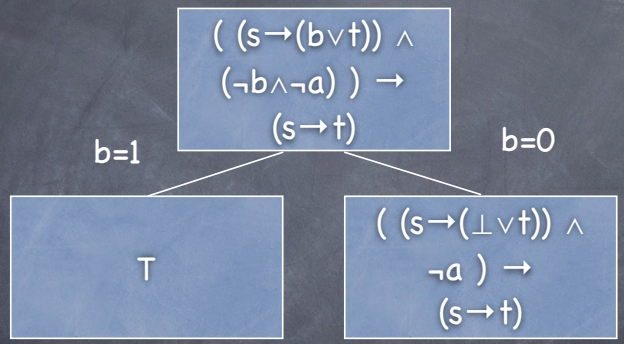
Un exemple plus complexe



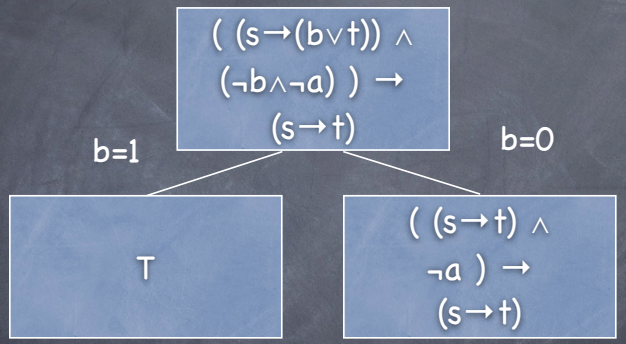
Un exemple plus complexe



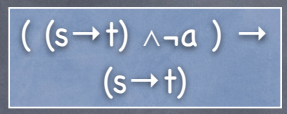
Un exemple plus complexe



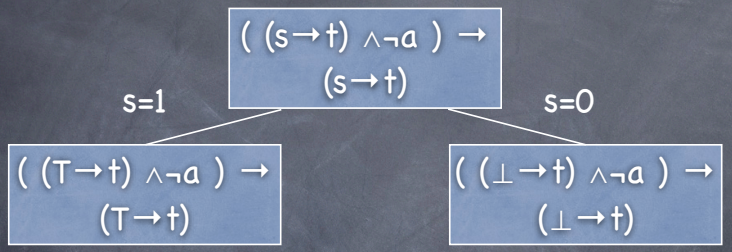
Un exemple plus complexe



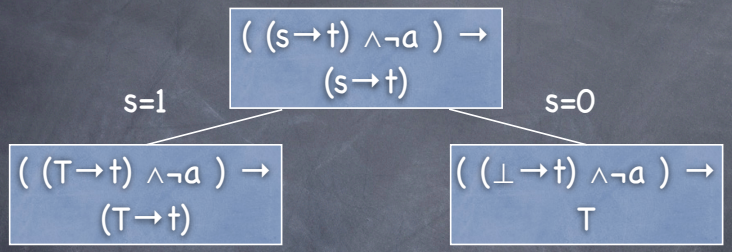
Un exemple plus complexe



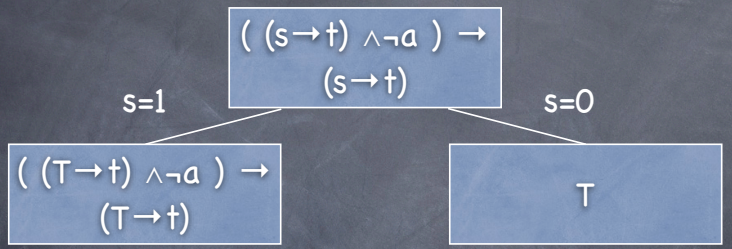
Un exemple plus complexe



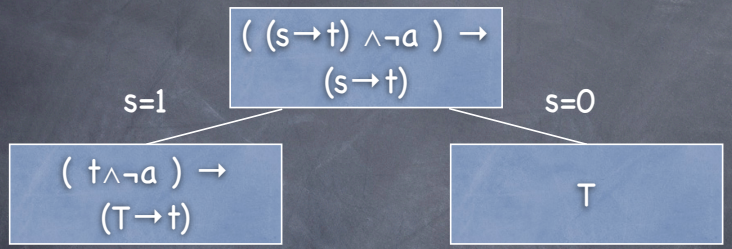
Un exemple plus complexe



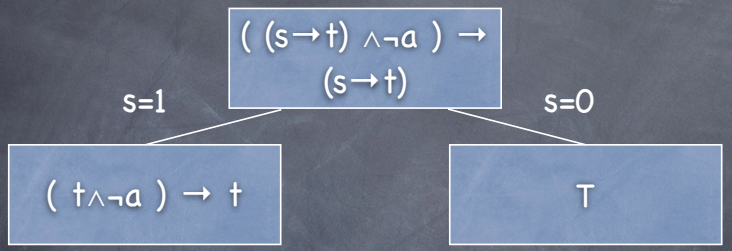
Un exemple plus complexe



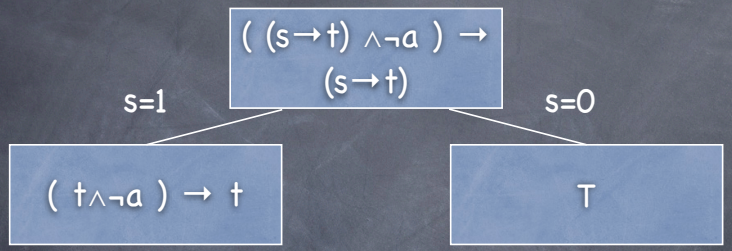
Un exemple plus complexe



Un exemple plus complexe



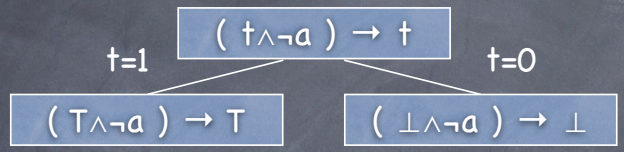
Un exemple plus complexe



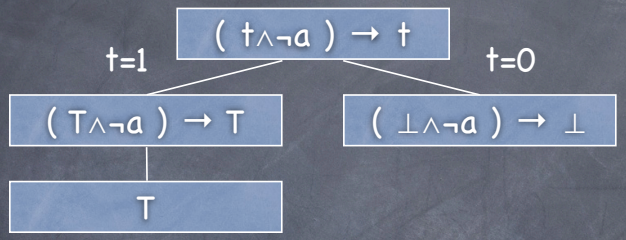
Un exemple plus complexe

(t∧¬a) → t

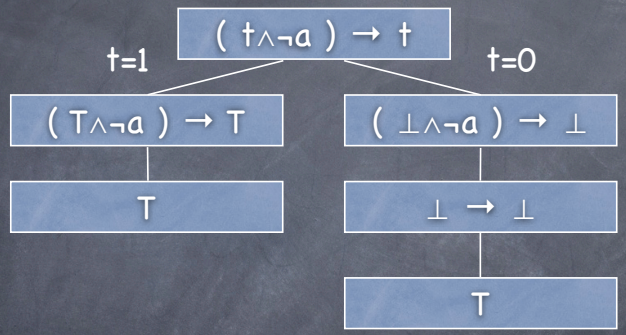
Un exemple plus complexe



Un exemple plus complexe



Un exemple plus complexe



Remarques sur l'algorithme de Quine

- Tout les branches se terminent par T, alors on a un tautologie (pour une contradiction on aurait eu que des ⊥ et pour une formule cohérent une combinaison de T et ⊥).
- L'algorithme termine toujours.
- L'algorithme est correct.

Avantages de l'algorithme de Quine

- Comme on a pu voir dans l'exemple, dans presque tous les cas, on a pu simplifier un des deux choix pour un valeur de vérité en "vrai".
- On est loin d'avoir énuméré les 64 possibilités dans cet exemple.
- Mais, en partie c'est parce qu'on a bien choisi l'atome à éliminer.

Résolution

- On veut démontrer que X est valide.
- On transforme la formule ¬X en une formule Y qui est équivalent et en forme normale conjonctive.
- On démontre une contradiction à partir de Y.
- Alors X est valide.

Résolution

- $\neg X \leftrightarrow Y$
- $Y \rightarrow \bot$
- $\neg X \rightarrow \bot$
- X

Forme Normale Conjonctive/Disjonctive

Conjonction $\wedge$

Forme Normale Conjonctive/Disjonctive

Conjonction $\wedge$

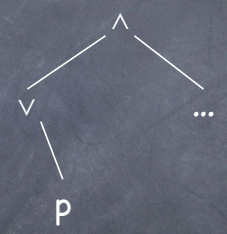
de disjonctions $\vee$ ...

Forme Normale Conjonctive/Disjonctive

Conjonction

de disjonctions

des atomes



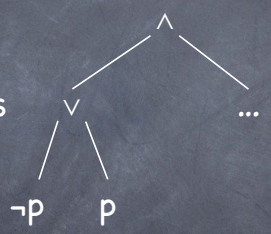
Forme Normale Conjonctive/Disjonctive

Conjonction

de disjonctions

des atomes

et leur négations



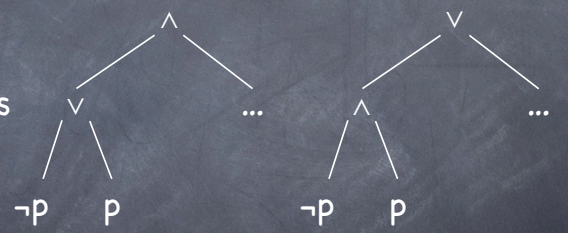
Forme Normale Conjonctive/Disjonctive

Conjonction

de disjonctions

des atomes

et leur négations



Transformer en forme normal conjonctive

- Utiliser les définitions de → et ↔ pour assurer que la formule contient que ¬ , ∧ et ∨.
- Utiliser les lois De Morgan pour bouger les négations vers les formules atomiques.
- Utiliser les lois de distribution pour bouger les conjonctions vers l'extérieur.

Forme Normale Conjonctive

$\neg(((s \rightarrow (b \vee t)) \wedge ((b \vee a) \rightarrow (r \wedge m)) \wedge \neg r) \rightarrow (s \rightarrow t))$

$\neg(\neg((s \rightarrow (b \vee t)) \wedge ((b \vee a) \rightarrow (r \wedge m)) \wedge \neg r)) \vee (s \rightarrow t))$

$\neg\neg((s \rightarrow (b \vee t)) \wedge ((b \vee a) \rightarrow (r \wedge m)) \wedge \neg r)) \wedge \neg(s \rightarrow t)$

$((s \rightarrow (b \vee t)) \wedge ((b \vee a) \rightarrow (r \wedge m)) \wedge \neg r)) \wedge \neg(s \rightarrow t)$

Forme Normale Conjonctive

$((s \rightarrow (b \vee t)) \wedge ((b \vee a) \rightarrow (r \wedge m)) \wedge \neg r)) \wedge \neg(s \rightarrow t)$

$(s \rightarrow (b \vee t)) \wedge$
$((b \vee a) \rightarrow (r \wedge m)) \wedge \neg r)) \wedge$
$\neg(s \rightarrow t)$

$(\neg s \vee (b \vee t)) \wedge$
$((b \vee a) \rightarrow (r \wedge m)) \wedge \neg r)) \wedge$
$\neg(s \rightarrow t)$

Forme Normale Conjonctive

$(\neg s \vee (b \vee t)) \wedge$
$((b \vee a) \rightarrow (r \wedge m)) \wedge$
$\neg r \wedge$
$\neg(s \rightarrow t)$

$\neg s \vee b \vee t \wedge$
$(\neg(b \vee a) \vee (r \wedge m)) \wedge$
$\neg r \wedge$
$\neg(\neg s \vee t)$

Forme Normale Conjonctive

$\neg s \vee b \vee t \wedge$
$(\neg(b \vee a) \vee (r \wedge m)) \wedge$
$\neg r \wedge$
$\neg(\neg s \vee t)$

$\neg s \vee b \vee t \wedge$
$(\neg(b \vee a) \vee (r \wedge m)) \wedge$
$\neg r \wedge$
$\neg\neg s \wedge$
$\neg t$

Forme Normale Conjonctive

¬s∨b∨t ∧
(¬(b∨a)∨(r∧m)) ∧
¬r ∧
¬(¬s∨t)

¬s∨b∨t ∧
(¬(b∨a)∨(r∧m)) ∧
¬r ∧
s ∧
¬t

Forme Normale Conjonctive

¬s∨b∨t ∧
(¬(b∨a)∨(r∧m)) ∧
¬r ∧
s ∧
¬t

Forme Normale Conjonctive

¬s∨b∨t ∧
((¬b∧¬a)∨(r∧m)) ∧
¬r ∧
s ∧
¬t

Forme Normale Conjonctive

¬s∨b∨t ∧
((¬b∧¬a)∨(r∧m)) ∧
¬r ∧
s ∧
¬t

$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$

Forme Normale Conjonctive

$\neg s \vee b \vee t \wedge$
$((\neg b \wedge \neg a) \vee (r \wedge m)) \wedge$
$\neg r \wedge$
$s \wedge$
$\neg t$

$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$

$A = \neg b$ $B = \neg a$ $C = r \wedge m$

Forme Normale Conjonctive

$\neg s \vee b \vee t \wedge$
$\neg b \vee (r \wedge m) \wedge \neg a \vee (r \wedge m) \wedge$
$\neg r \wedge$
$s \wedge$
$\neg t$

$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$

$A = \neg b$ $B = \neg a$ $C = r \wedge m$

Forme Normale Conjonctive

$\neg s \vee b \vee t \wedge$
$\neg b \vee (r \wedge m) \wedge \neg a \vee (r \wedge m) \wedge$
$\neg r \wedge$
$s \wedge$
$\neg t$

$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

$A = \neg b$ $B = r$ $C = m$

Forme Normale Conjonctive

$\neg s \vee b \vee t \wedge$
$\neg b \vee r \wedge \neg b \vee m \wedge \neg a \vee (r \wedge m) \wedge$
$\neg r \wedge$
$s \wedge$
$\neg t$

$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

$A = \neg b$ $B = r$ $C = m$

Forme Normale Conjonctive

¬s∨b∨t ∧
¬b∨r ∧ ¬b∨m ∧ ¬a∨(r∧m) ∧
¬r ∧
s ∧
¬t

$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

A = ¬a B = r C = m

Forme Normale Conjonctive

¬s∨b∨t ∧
¬b∨r ∧ ¬b∨m ∧ ¬a∨r ∧ ¬a∨m ∧
¬r ∧
s ∧
¬t

$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

A = ¬a B = r C = m

Forme Normale Conjonctive

¬s∨b∨t ∧
¬b∨r ∧
¬b∨m ∧
¬a∨r ∧
¬a∨m ∧
¬r ∧
s ∧
¬t

Remarques sur le Forme Normale Conjonctive

- Le forme normale conjonctive nous donne un formule du forme $C_1 \wedge \ldots \wedge C_n$,
- On appelle chacun des C_i un clause,
- Chaque clause est une disjonction de formules atomiques et leur négations (une disjonction de littéraux)

Remarques sur le Forme Normale Conjonctive

- L'ordre des clauses dans un formule en forme normale conjonctive ainsi que l'ordre des littéraux dans des clauses n'est pas important (pourquoi?)

Résolution

- La résolution est un méthodes standards pour la démonstration automatique.
- La résolution travaille avec des formules en forme normale conjonctive.
- On cherche a démontrer qu'une formule X est un tautologie.
- Pour faire ça on démontre que ¬X est inconsistante (ou contradictoire)

Résolution

$p \vee X_1 \vee \ldots \vee X_n$

$\ldots$

$\neg p \vee Y_1 \vee \ldots \vee Y_n$

$p \vee X_1 \vee \ldots \vee X_n$

$\ldots$

$\neg p \vee Y_1 \vee \ldots \vee Y_m$

$X_1 \vee \ldots \vee X_n \vee Y_1 \vee \ldots \vee Y_m$

$(p \vee X) \wedge (\neg p \vee Y) \equiv (p \vee X) \wedge (\neg p \vee Y) \wedge (X \vee Y)$

Résolution

$\neg s \vee b \vee t$

$\neg b \vee r$

$\neg b \vee m$

$\neg a \vee r$

$\neg a \vee m$

$\neg r$

s

$\neg t$

Résolution

¬s∨b∨t
¬b∨r
¬b∨m
¬a∨r
¬a∨m
¬r
s
¬t

b∨t

Résolution

¬s∨b∨t
¬b∨r
¬b∨m
¬a∨r
¬a∨m
¬r
s
¬t

b∨t
b

Résolution

¬s∨b∨t
¬b∨r
¬b∨m
¬a∨r
¬a∨m
¬r
s
¬t

b∨t
b
r

Résolution

¬s∨b∨t ∧
¬b∨r ∧
¬b∨m ∧
¬a∨r ∧
¬a∨m ∧
¬r ∧
s ∧
¬t

b∨t
b
r
⊥

Clauses de Horn

- Un clause de Horn est une disjonction qui contient exactement un littéral positif.
- C'est-à-dire: un clause de Horn a une unique formule atomique sans négation, les autres formules ont toutes une négation.

Résolution
Clauses de Horn

¬s∨b∨t
¬b∨r
¬b∨m
¬a∨r
¬a∨m
¬r
s

¬t

Résolution
Clauses de Horn

¬s∨b∨t
¬b
¬b∨m
¬a
¬a∨m
¬r
s

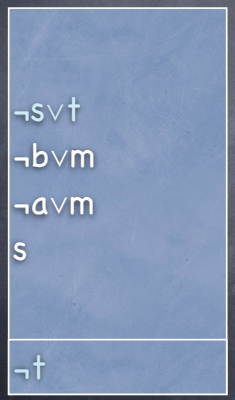
¬t

Résolution
Clauses de Horn

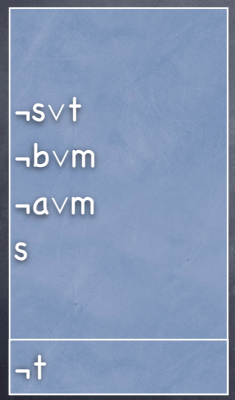
¬s∨b∨t
¬b
¬b∨m
¬a
¬a∨m
s

¬t

Résolution
Clauses de Horn



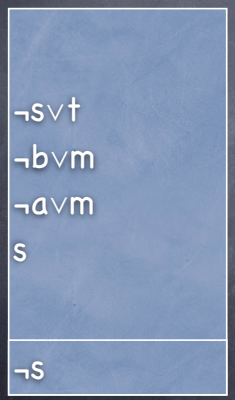
Résolution
Clauses de Horn



Ceci sont tous de clauses de Horn, sauf le ¬t, qu'on appelle le query (ou la question)

On veut démontrer que t est vrai, alors comme avant: on assume que t est faux est démontre que ça donne une contradiction

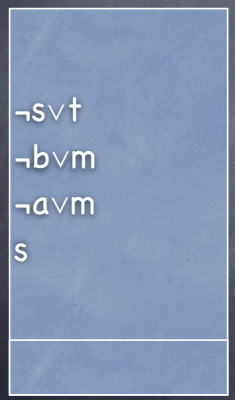
Résolution
Clauses de Horn



Ceci sont tous de clauses de Horn, sauf le ¬t, qu'on appelle le query (ou la question)

On veut démontrer que t est vrai, alors comme avant: on assume que t est faux est démontre que ça donne une contradiction

Résolution
Clauses de Horn



Ceci sont tous de clauses de Horn, sauf le ¬t, qu'on appelle le query (ou la question)

On veut démontrer que t est vrai, alors comme avant: on assume que t est faux est démontre que ça donne une contradiction

Résolution
Clauses de Horn

$\neg s \vee t$
$\neg b \vee m$
$\neg a \vee m$
s

Les clauses de Horn s'écrivent, naturellement en forme d'implication.

Résolution
Clauses de Horn

$s \rightarrow t$
$b \rightarrow m$
$a \rightarrow m$
s

Les clauses de Horn s'écrivent, naturellement en forme d'implication.

Résolution
Clauses de Horn

$t \leftarrow s$
$m \leftarrow b$
$m \leftarrow a$
s

Les clauses de Horn s'écrivent, naturellement en forme d'implication.

Prolog utilise des clause de Horn pour encoder sa base de connaissances et la résolution pour en déduire de réponses aux questions.

Dans le contexte de Prolog on inverse les implications, ceci n'est qu'une différence de notation

Logique de Prédicats

- La logique propositionnel est un peu trop simple pour modéliser les connaissance.
- On peut parle que des phrases simples.
- On aimerait bien faire plus que ça.

Logique de Prédicats

- On veut parler des individus et leur donner un nom.
- On veut parler de propriétés de ces individus et des relations entre eux.

Exemples

- "Chaque homme aime une femme".