INF4230 – Intelligence Artificielle

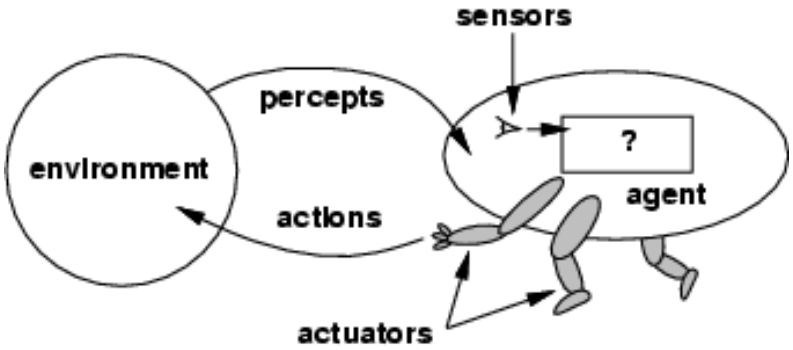
Agents logiques, logique propositionnelle et logique du premier ordre

Hiver 2017

Sommaire

- Agents à base de connaissances
- Monde du Wumpus
- Logique propositionnelle
- Logique du premier ordre
- Inférence logique
- Résolution
- Chaînage avant
- Chaînage arrière

Rappel : Agents et environnements



- Un **agent** peut être vu comme une **fonction** qui associe un historique de données sensorielles (*percept history*) à une action :

$$[f: \mathcal{P}^{\star} \rightarrow \mathcal{A}]$$

- En pratique le processus est un implémenté par un programme sur une architecture matérielle particulière .

- Agent = Architecture + Programme

Un agent logique simple

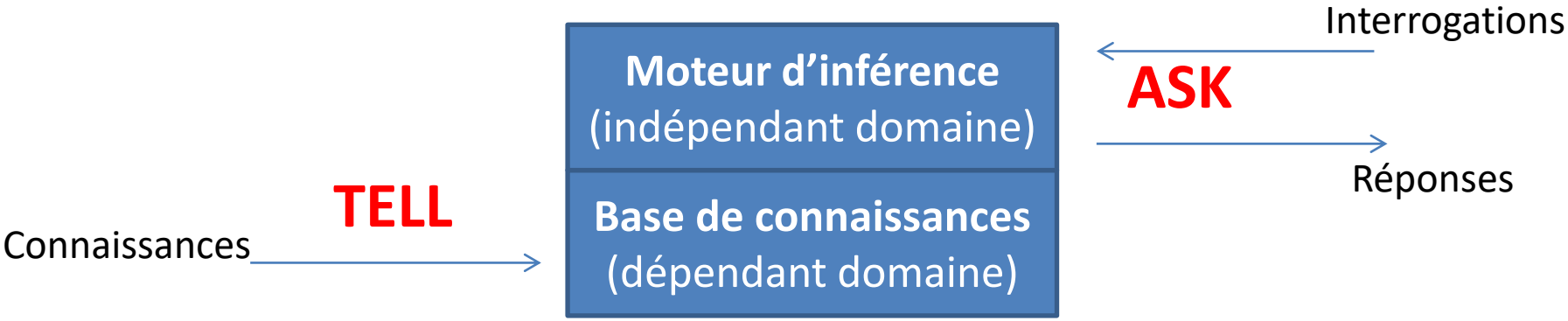
```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

L'agent doit :

- Représenter les états et les actions dans une base de connaissances (KB).
- Intégrer de nouvelles données sensorielles dans la KB.
- Déduire des informations cachées à propos du monde.
- Déduire des actions appropriées (décider) à exécuter.

Base de connaissances

- **Base de connaissances** : ensemble d'**énoncés** (*statements*) spécifiés dans un **langage formel**.
- Ces énoncés sont des **assertions** à propos d'un **domaine spécifique** (monde, *world*).
- **Approche déclarative** : on **dit (TELL)** au système ce qu'il doit connaître.
- On peut **interroger** (**ASK**) la base de connaissances, à l'aide d'un **moteur d'inférence**, pour obtenir une réponse déduite de ce qui a été dit (**TELL**) précédemment au système.
- Le moteur d'inférence est indépendant du domaine (monde, *world*).



Connaissances dans la KB

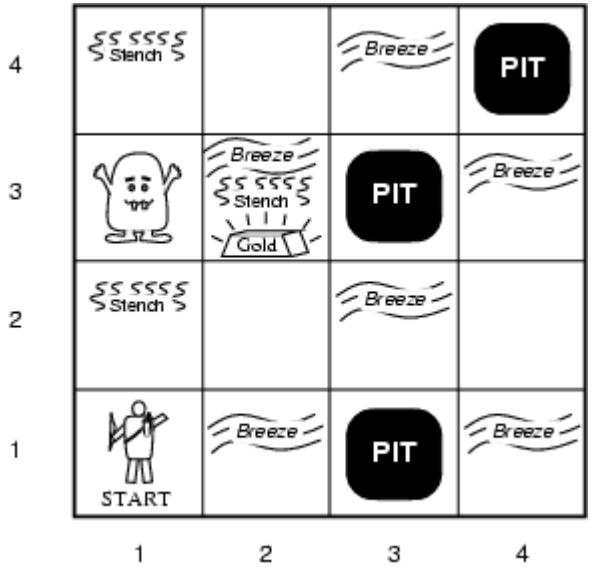
- Toutes les formes de savoir et de savoir-faire
  - Représentation (modélisation) abstraite de la réalité.
  - Objets et concepts.
  - Relations entre objets ou concepts.
  - Procédures.
  - Etc.

Représentation des connaissances

- Les connaissances doivent être données sous **forme symbolique (langage formel)** afin d'être exploitables par un système de raisonnement automatique (algorithmes).
- Plusieurs formalismes / langages :
  - **logique propositionnelle**;
  - **logique du premier ordre**;
  - réseau, règles de production, schémas (*frames*), etc.

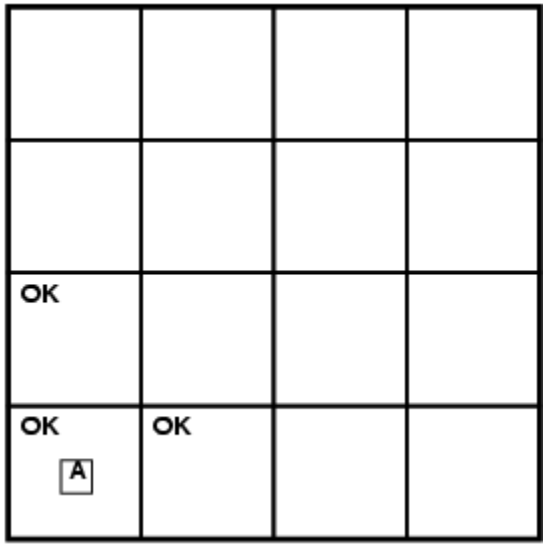
Exemple : Le Monde des *Wumpus (rappel)*

- **Mesure de performance**
  - or +1000, mort -1000
  - -1 par un pas, -10 pour une flèche
- **Environnement**
  - Senteur dans les chambres adjacent au wumpus.
  - Brise dans les chambres adjacentes à une fosse
  - Scintillement si l'or est dans la chambre
  - Le wumpus meurt si on lui tire une flèche de face
  - On a une seule flèche
  - On peut ramasser l'or dans la même chambre
  - On peut lâcher l'or dans une chambre

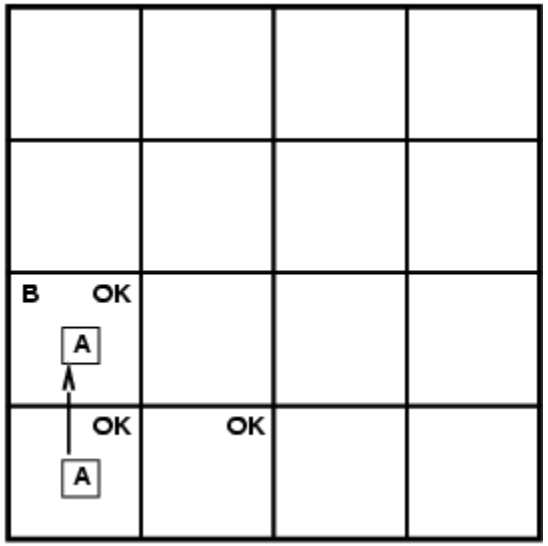


- **Senseurs**: *stench* (senteur), *breeze* (brise), *glitter* (scintillement), *bumper* (choc), *scream* (cri).
- **Actuateurs**: *Left turn*, *Right turn*, *Forward*, *Grab*, *Release*, *Shoot*

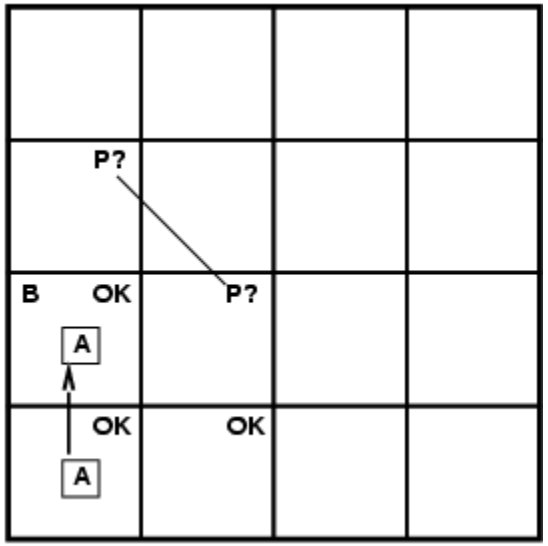
Exploration d'un monde de Wumpus



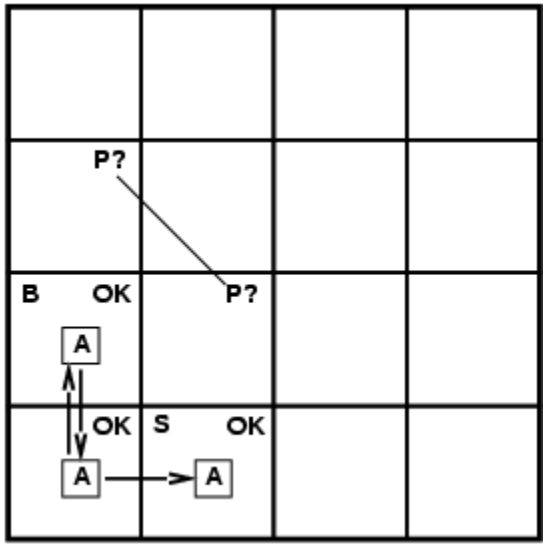
Exploration d’un monde de Wumpus



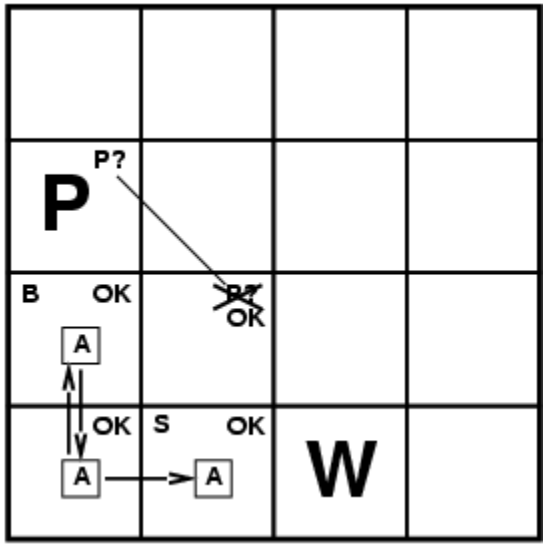
Exploration d’un monde de Wumpus



Exploration d'un monde de Wumpus

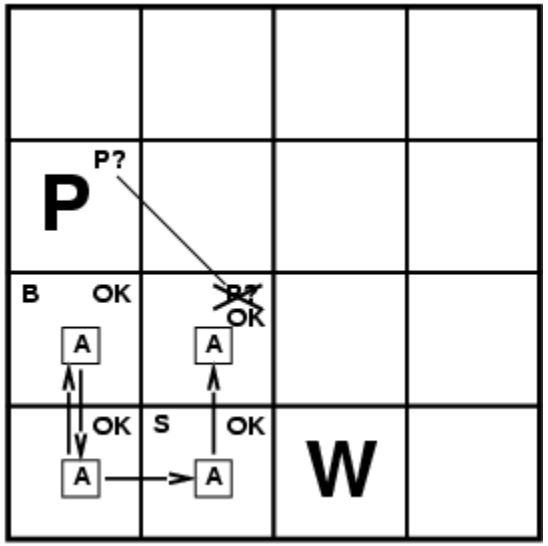


Exploration d'un monde de Wumpus

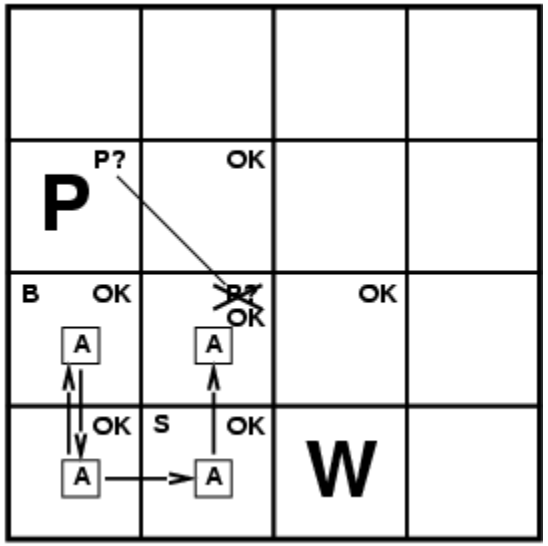


On déduit **logiquement** la présence d'un puits en [1,3], la présence du Wumpus en [3,1], et l'absence de Wumpus et de puits en [2,2]

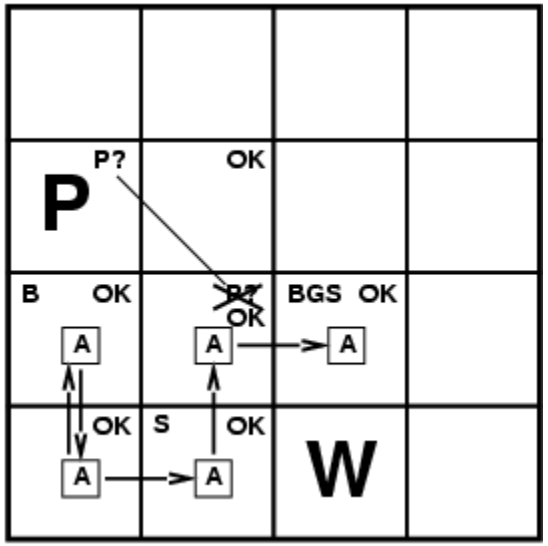
Exploration d’un monde de Wumpus



Exploration d’un monde de Wumpus



Exploration d’un monde de Wumpus



Conséquence logique (*Entailment*)

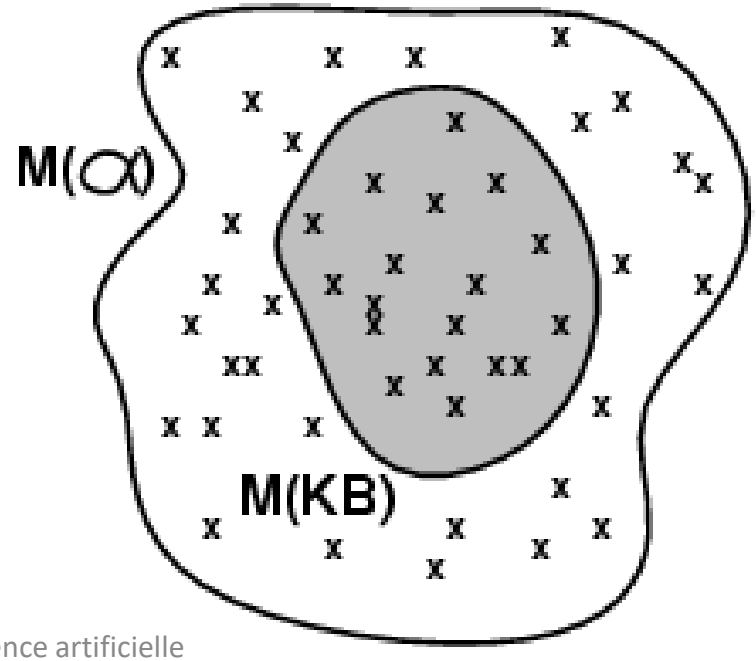
- Une **conséquence logique** signifie qu'un fait (α) est soutenu par un ensemble d'autres faits (KB).
- On écrit :

  $KB \models \alpha$

  pour dire que **la base de connaissances KB supporte l'énoncé α** ou **l'énoncé α est une conséquence logique de la base de connaissances KB.**
- Cela signifie que α doit être vrai dans toutes les interprétations possible du monde où *KB* est vrai.
  - Exemple : (A) et ($A \Rightarrow B$) ont pour conséquence logique B.

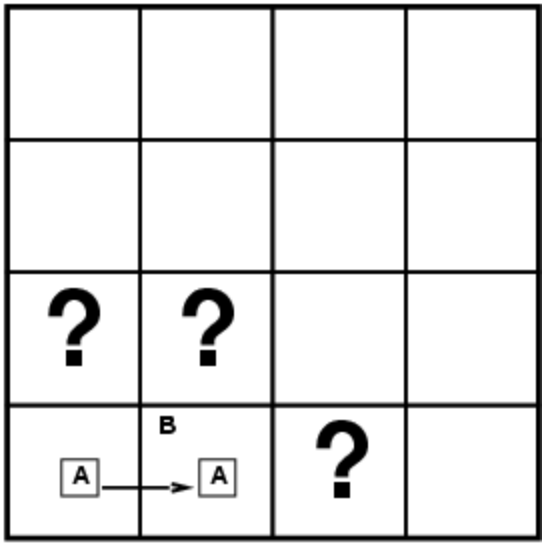
Notion de Modèles

- Pour un ensemble d'énoncés, il peut y avoir plusieurs interprétations possibles (de ce qui n'est pas connu).
- En logique, on utilise le terme **modèle** (=**monde possible)**.
- On dit que *m* est un modèle d'un ensemble d'énoncés α si α est vrai dans *m.*
- *M(α)* est l'ensemble de tous les modèles de α.
- KB $\models$ α ssi $M(KB) \subseteq M(\alpha)$.



Conséquence logique dans le monde du Wumpus

Base de connaissances *KB* =
Situation après avoir rien détecter en [1,1], action *MoveRight*, et perçu une brise dans [2,1]

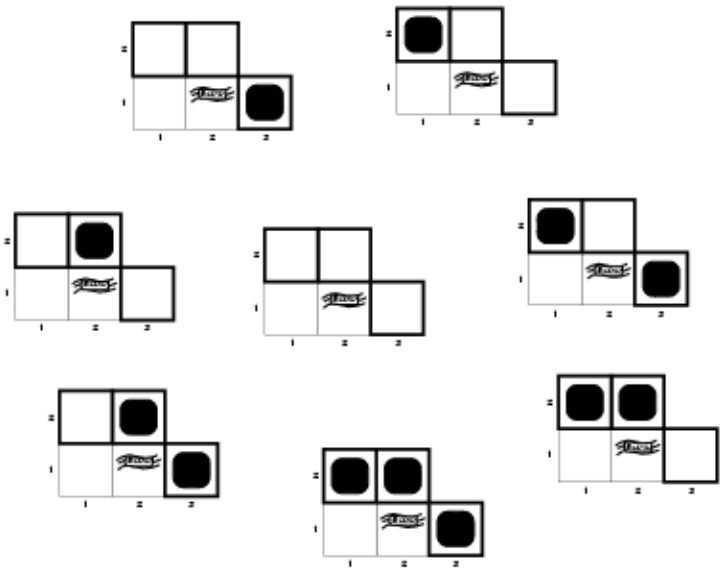


Considérons plusieurs modèles pour *KB* uniquement à propos des puits aux 3 endroits marqués de «?».

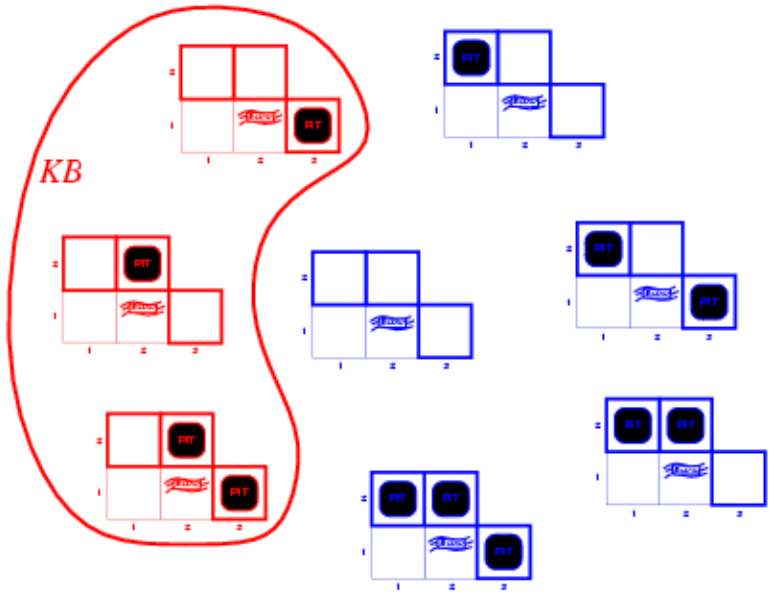
3 valeurs booléennes

$\Rightarrow 2^3$=8 modèles possibles

Modèles du Wumpus



Modèles du Wumpus

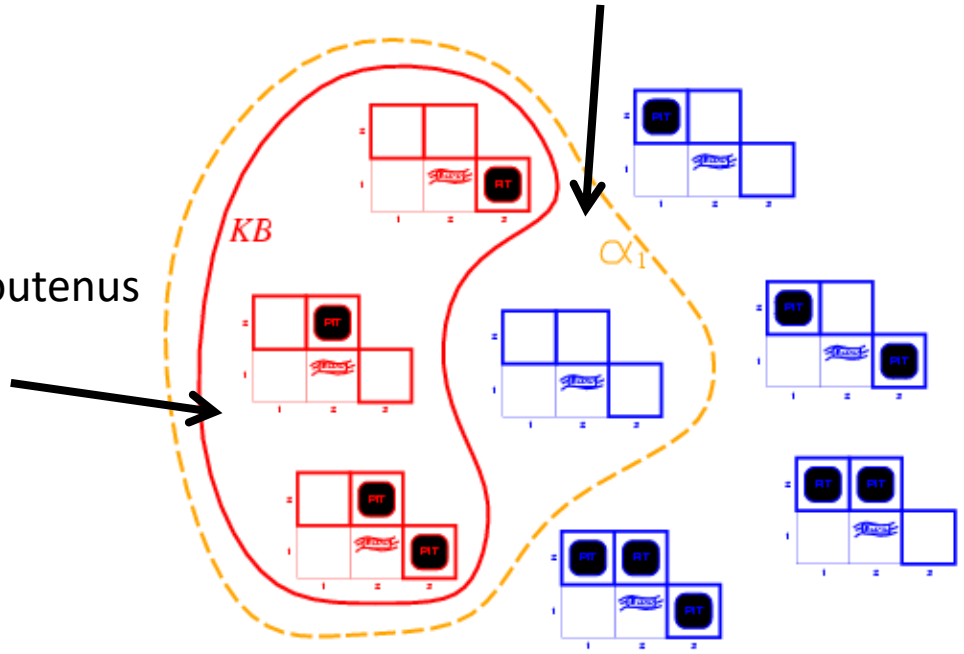


- *KB* = règles du monde du wumpus + observations

Modèles du Wumpus

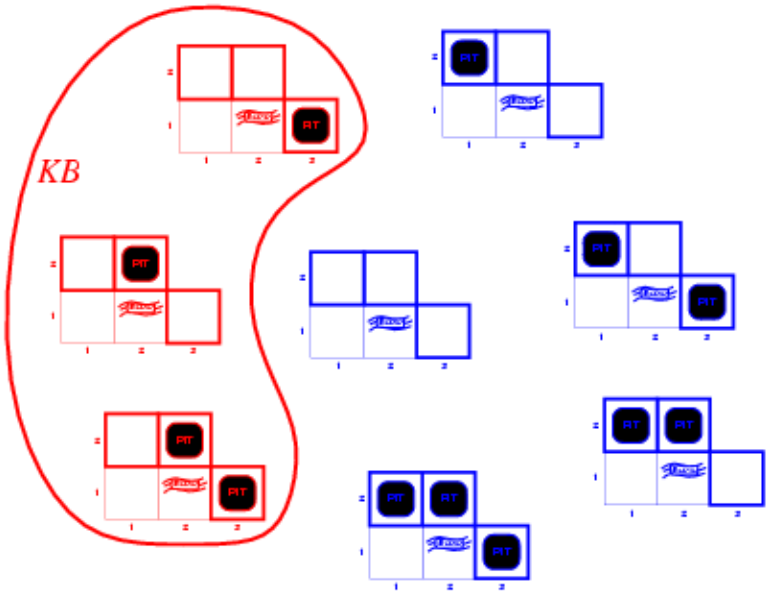
$M(\alpha_1)$ = Modèles(interprétation) où la case [1,2] est sécuritaire.

M(*KB*)=Modèles soutenus par les la base de connaissances



- KB = règles du monde du wumpus + observations.
- Soit l'énoncé α_1 = « la case [1,2] est sécuritaire ».
- KB $\models \alpha_1$, soit (KB supporte α_1), est prouvé par **vérification de modèle** (*model checking*).

Modèles du Wumpus

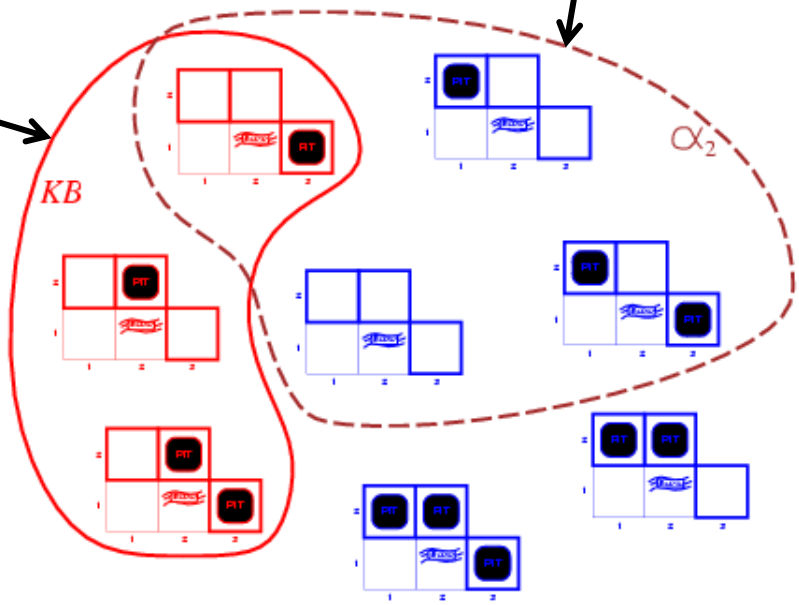


- KB = règles du monde du wumpus + observations

Modèles du Wumpus

M(α_1) = Modèles(interprétation) où la case [2,2] est sécuritaire.

M(*KB*)=Modèles soutenus par les la base de conn.



- KB = règles du monde du wumpus + observations
- Soit l'énoncé α_2 = «[2,2] est sécuritaire »
- *Donc,* KB $\not\models \alpha_2$

Langages formels pour la logique

- En **logique**, les connaissances (informations) sont représentées dans à l'aide d'un formalisme / **langage formel**.
- La **syntaxe** définit la structures des énoncés du langage.
- La **sémantique** définit la signification des énoncés du langage.
  - Ex.: définir la vérité d'un énoncé.
- Exemple : le langage de l'arithmétique.
  - **x+2 ≥ y** est un énoncé;
  - **x2+y > {}** n'est pas un énoncé (syntaxe incorrecte).
  - **x+2 ≥ y** est vrai ssi le nombre x+2 est pas plus petit que le nombre y
  - **x+2 ≥ y** est vrai dans un monde où x = 7, y = 1
  - **x+2 ≥ y** est faux dans un monde où x = 0, y = 6

Langages formelles pour la logique

- Logique propositionnelle
  - Langage le plus simple.
  - Utilisé pour présenter les concepts.
- Logique du premier ordre
  - Peut être vue comme une extension de la logique propositionnelle.
  - La plupart des systèmes logiques utilisent la logique du premier ordre ou un dérivé de.

Syntaxe de la logique propositionnelle

- Si S est **proposition**, $\neg S$ est une **proposition** (négation)
- Si S_1 et S_2 **propositions**, $S_1 \wedge S_2$ est une **formule** (conjonction)
- Si S_1 et S_2 sont **formule**, $S_1 \vee S_2$ est une **formule**(disjonction)
- Si S_1 et S_2 sont **formule**, $S_1 \Rightarrow S_2$ est une **formule**(implication)
- Si S_1 et S_2 sont **formule**, $S_1 \Leftrightarrow S_2$ est une **formule**(double impl.)

Sémantique de la logique propositionnelle

- Chaque modèle définie une valeur de vérité (vrai/faux, *true/false*) pour chaque symbole de proposition

E.g.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	false	true	false

- Avec ces trois symboles, 8 modèles (mondes possibles) peuvent être énumérés.
- Règles pour évaluer la valeur de vérité selon un modèle *m*:

$\neg S$	est vrai ssi	S est faux	
$S_1 \wedge S_2$	est vrai ssi	S_1 est vrai et	S_2 est vrai
$S_1 \vee S_2$	est vrai ssi	S_1 est vrai ou	S_2 est vrai
$S_1 \Rightarrow S_2$	est vrai ssi	S_1 est vrai ou	S_2 est vrai
	est false ssi	S_1 est vrai et	S_2 est faux
$S_1 \Leftrightarrow S_2$	est vrai ssi	$S_1 \Rightarrow S_2$ est vrai et	$S_2 \Rightarrow S_1$ est vrai

- Une évaluation récursive permet d'évaluer n'importe quelle formule. Exemple :

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \mathit{true} \wedge (\mathit{true} \vee \mathit{false}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$$

Table de vérités

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Équivalances

- Deux formules sont logiquement équivalentes ssi elle sont vrai dans les mêmes modèles: α ≡ ß ssi α ⊨ β et β ⊨ α

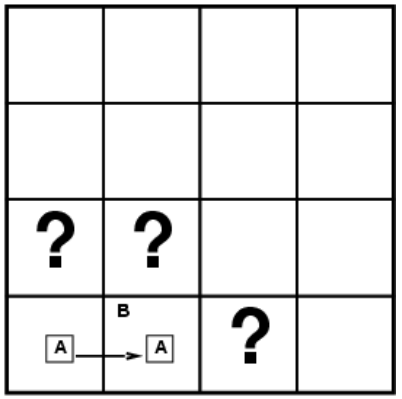
$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Formules dans le monde du Wumpus

- Soit la proposition $P_{i,j}$: il y a un puits dans [i, j].
- Soit la proposition $B_{i,j}$: une brise est perçu dans [i, j].

$\neg P_{1,1}$
$\neg B_{1,1}$
$B_{2,1}$

Énoncés permettant de représenter la situation suivante à droite



- «Un puits cause une brise dans les cases voisines» se traduit formellement en les formules :

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
…

Connaissances connues à l'avance sur le monde.

KB pour le monde du Wumpus

KB = {$\neg P_{1,1}$,

$\neg B_{1,1}$,

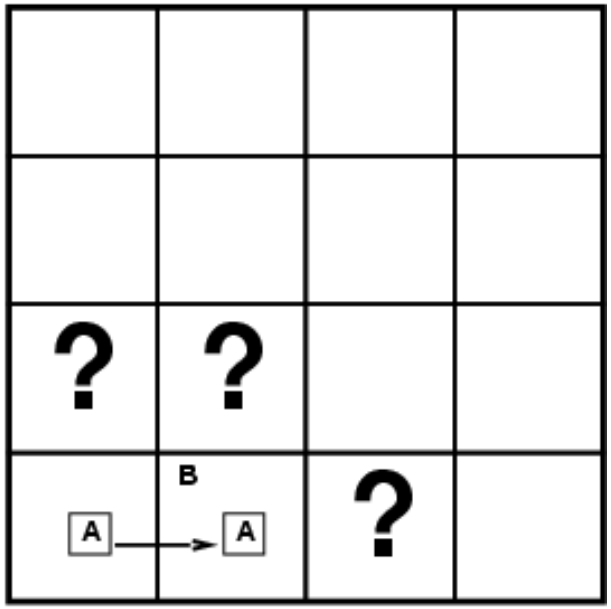
$B_{2,1}$,

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$,

$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$,

…

}



Procédure d'inférence

- Le but est de déduire des faits (conclusions) à partir d'un ensembles de faits déjà connus.
- Cela s'appelle de **l'inférence**.
- Par exemple, à partir de KB, on peut inférer (conclure) $\neg P_{1,2}$.
- Une procédure d'inférence (algorithme d'inférence) permet cela.

Table de vérités pour inférence

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
$false$	$false$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$false$	$false$	$false$	$false$	$false$	$true$	$false$	$true$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$false$	$true$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$true$	$false$	$false$	$false$	$false$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$false$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$true$	$false$	$false$	$false$	$true$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$true$	$true$	$true$	$true$	$true$	$true$	$true$	$false$	$false$

- α_1 = « la case [1,2] est sécuritaire (sans puits) ».
- Si α_1 est toujours vraie quand KB est vrai, alors on peut conclure que $KB \models \alpha_1$.

Inférence par énumération

- Une énumération en profondeur de tous les modèles est correct et complet.
- Pour *n* symboles, complexité temporelle=O(2^n), complexité spatiale= O(*n*)

function TT-ENTAILS?(KB, α) **returns** *true* or *false*

symbols ← a list of the proposition symbols in KB and α
return TT-CHECK-ALL(KB, α, *symbols*, [])

---

function TT-CHECK-ALL(KB, α, *symbols*, *model*) **returns** *true* or *false*
if EMPTY?(*symbols*) **then**
if PL-TRUE?(KB, *model*) **then return** PL-TRUE?(α, *model*)
else return *true*
else do
$P \leftarrow$ FIRST(*symbols*); *rest* ← REST(*symbols*)
return TT-CHECK-ALL(KB, α, *rest*, EXTEND(P, *true*, *model*)) **and**
TT-CHECK-ALL(KB, α, *rest*, EXTEND(P, *false*, *model*))

Validité et satisfiabilité

Une formule est **valide** si elle est vrai dans tous les modèles.

Ex.: *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

La validité est liée à l'inférence par déduction .

$KB \models \alpha$ si est seulement si $(KB \Rightarrow \alpha)$ est valide

Une formule est **satisfiable** si elle est vraie dans certains modèles.

e.g., $A \vee B$, C

Une formule est **insatisfiable** si elle est vraie pour aucun modèle.

e.g., $A \wedge \neg A$

La satisfiabilité est lié à l'inférence :

$KB \models \alpha$ si et seulement si $((KB \wedge \neg \alpha)$ est **insatisfiable**).

Méthode de résolution

1. Spécifier les connaissances dans un langage formel (ex.: la logique propositionnelle).
2. Convertir les énoncés sous forme conjonctive normale.
3. Appliquer la résolution.

Forme conjonctive normale

- Conjonction : et logique ($\wedge$)
- Disjonction : ou logique ($\vee$)
- Une base de connaissance en **forme conjective normale** (*Conjunctive Normal Form*, *CNF*) est **une conjonction de disjonctions de propositions**.
- Chaque disjonction est une **clause**.
- Exemples :

  $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

  $(l_i \vee \ldots \vee l_k) \wedge (m_1 \vee \ldots \vee m_n)$

Conversion en forme conjonctive normale (CNF)

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Éliminer l'équivalence $\Leftrightarrow$: remplacer $\alpha \Leftrightarrow \beta$ par $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

2. Éliminer l'implication $\Rightarrow$: remplacer $\alpha \Rightarrow \beta$ par $\neg\alpha \vee \beta$.

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

3. Déplacer les négations $\neg$ à l'intérieur devant les symboles propositionnels avec la loi de Morgan:

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. Appliquer la loi de distribution ($\wedge$ over $\vee$) :

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Exercice : conversions pour le monde de Wumpus

$$KB = \{\neg P_{1,1}, \neg B_{1,1}, B_{2,1}, B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}),$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}),$$
$$B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$$
$$\}$$

Règle de résolution (pour CNF):

- Soit deux clauses en CNF : $(\ell_i \vee \ldots \vee \ell_k) \wedge (m_1 \vee \ldots \vee m_n)$
- Si ℓ_i est un complément de m_j alors on peut écrire:

$$(\ell_i \vee \ldots \ell_{i-1} \vee \cancel{\ell_i} \vee \ell_{i+1} \ldots \vee \ell_k) \wedge (m_1 \vee \ldots m_{j-1} \vee \cancel{m_j} \vee m_{j+1} \vee \ldots \vee m_n)$$

$$\ell_i \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n$$

- La ligne signifie qu'à partir de ce qui est en haut (deux clauses), on peut déduire ce qui est en bas (une nouvelle clause).

Exemple de résolution

$$\frac{(P_{1,3} \vee P_{2,2}) \wedge \qquad \neg P_{2,2}}{P_{1,3}}$$

Exercice : conversions pour le monde de Wumpus

- Prouver :

  $\neg P_{1,2}$

- Pouvez-vous prouver :

  $\neg P_{2,2}$ et $P_{3,1}$???

LOGIQUE DU PREMIER ORDRE

Logique du premier ordre

- La logique du premier ordre peut être vue comme une extension de la logique propositionnelle.
- On peut avoir des **variables** et des **fonctions**.
- Aussi appelé le **calcul des prédicats**.

Exemples de raisonnement déductif

Prouvez que Marcus hait César à partir de:

1. Marcus est une personne.
2. Marcus est un pompéien.
3. Tous les pompéiens sont des romains.
4. César est un dirigeant.
5. Tout le monde est loyal à quelqu'un.
6. Tous les romains sont loyaux à César ou le haïssent.
7. Les seuls dirigeants qu'une personne essaie d'assassiner sont ceux auxquels elle n'est pas loyal
8. Marcus a essayer d'assassiner César.

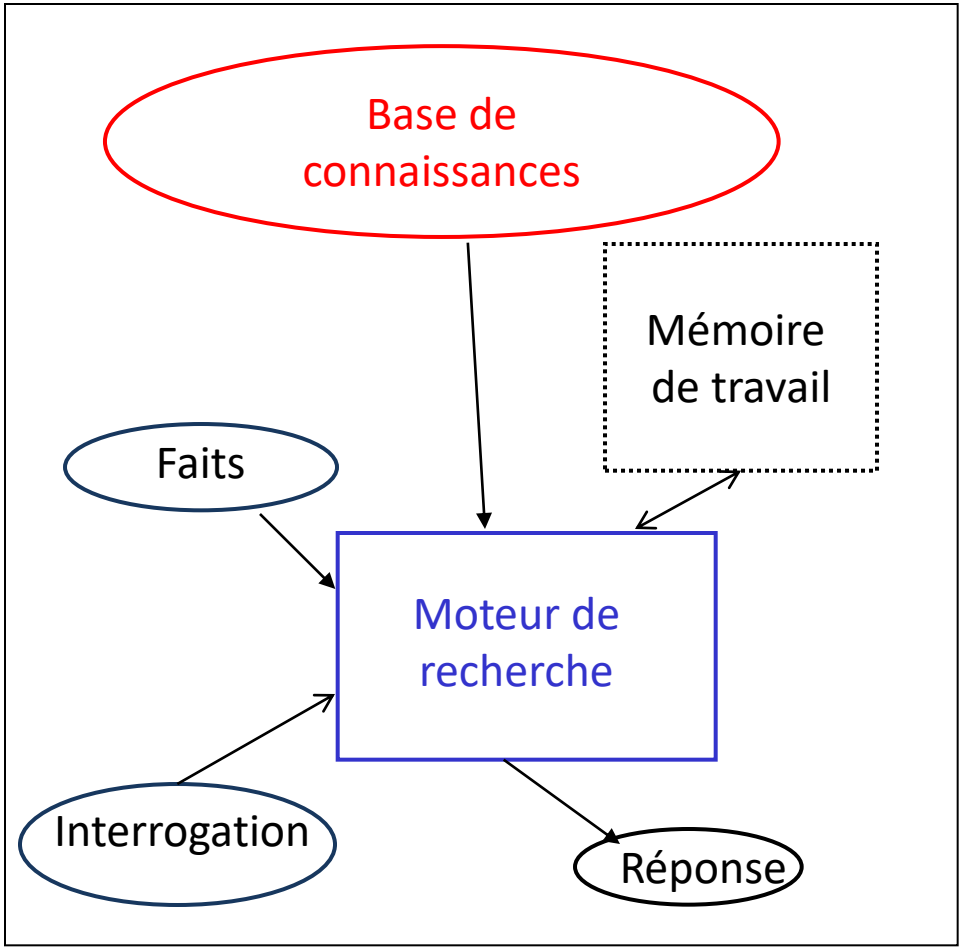
Déduire la maladie du patient et le traitement approprié, à partir de:

1. Symptômes d'un patient
2. Règles de causalité entre les symptômes et les pathologies.
3. Règles de causalité sur les traitements.

Diagnostiquer le problème d'un véhicule à partir de:

1. Symptômes d'un véhicule
2. Règles de causalité pour la mécanique auto.

Exemple: Schéma d'un système expert à base de règles de production



La base de connaissances est spécifiée par des règles logiques.

Les faits sont des propositions logiques.

La mémoire de travail peut être vue comme un état.

Le moteur de recherche est une exploration de l'espace d'états.

On a besoin d'un algorithme d'unification *(pattern matching)*

Exemple:
Java Expert System Shell (JESS)

Plan de présentation sur la logique du premier ordre

- En premier lieu nous allons voir la ***syntaxe***, c'est-à-dire, la forme des expressions qu'on peut écrire dans le calcul des prédicats.

- Ensuite nous considérons la ***sémantique***, c'est-à-dire, comment on détermine si une expression est logiquement correcte.

- Enfin nous verrons une règle d'inférence appelée ***résolution***, c'est-à-dire une méthode pour déterminer si une expression est une conséquence logique d'un ensemble d'expressions logiques données.

Syntaxe des formules

- Une expression du calcul des prédicats est appelée une ***formule***.
- Les formules sont des combinaisons de ***prédicats*** à l'aide de ***connecteurs logiques*** et des ***quantificateurs***.
- Les ***prédicats*** décrivent des faits ou des relations entre les objets selon l'application.
- Les objets sont décrits par des ***termes***, c'est-à-dire des **constantes**, des **variables** ou des **fonctions**.
- Les prédicats, les connecteurs logiques, les quantificateurs et les termes sont décrits par des ***symboles***.

Symboles

- **Constantes**. Par exemple: *41, Dupont, Robot1, Janvier, Soleil, LivreIA*, etc.
- **Fonctions**. Par exemple: *temperature(x), position(x), auteur(l), etc.*
- **Prédicats**. Par exemple: *mortel(x), plusGrand(x,y), intéressant(l), etc.*
  - le nombre d'arguments d'une fonction ou d'un prédicats est appelé « arité ».
  - On pourrait comprendre les prédicats comme des cas particuliers de fonctions qui retournent des valeurs binaires (*vrai* ou *faux*). Ici ils jouent un rôle fondamental de sorte qu'on doit les traiter séparément des fonctions.
- **Variables**. Par exemple: *x, y, z*
- **Connecteurs**: $\neg$ (*non*), $\wedge$ (*et*), $\vee$ (*ou*), $\rightarrow$ (*implique*)
- **Quantificateurs**: $\forall$ (*pourTout*), $\exists$ (*ilExiste*)

Termes

- Les constantes et les variables sont des termes.
- Les applications des fonctions aux termes sont des termes.
  - En d'autres mots, si $t_1, \ldots, t_n$ sont des termes et f une fonction à n arguments, alors $f(t_1, \ldots, t_n)$ est aussi un terme.
  - Par exemple: *pere(John), pere(x), pere(pere(x))*

Syntaxe des formules

- Un prédicat est une formule.
  - Plus précisément, si $t_1, \ldots, t_n$ sont des termes et p est un prédicat à n arguments, alors $p(t_1, \ldots, t_n)$ est une formule.
  - C'est-à-dire la formule la plus simple qui soit (cas de base).
- La négation ou la conjonction d'une formule est aussi une formule.
  - Plus précisément, si α et β sont des formules, alors $\neg \alpha$ ainsi que $\alpha \wedge \beta$ sont des formules.
- La quantification universelle d'une formule est une formule.
  - Plus précisément, si α est une formule et x est une variable, alors $\forall x(\alpha)$ est une formule.

Notations

- Priorités et parenthèses
  - Le connecteur $\neg$ a priorité sur les connecteurs $\wedge$ et $\vee$
  - On utilise les parenthèses de la même façon que dans les expressions arithmétiques pour éviter les ambiguïtés.

- Abréviations (macros ou équivalences)
  - $\alpha \vee \beta$ est équivalent à $\neg (\neg\alpha \wedge \neg\beta)$
  - $\alpha \rightarrow \beta$ est équivalent à $\neg\alpha \vee \beta$
  - $\exists v\ (\alpha)$ est équivalent à $\neg(\forall v\ (\neg\alpha))$

Notations

- Les notations spécifiques des symboles sont selon les conventions ou les goûts.

- Dans un programme, on utilisera par exemple les symboles not (ou !), and (ou &&), or (ou ||), implies (ou ->), forall (ou V), exists (ou E), plutôt que les symboles grecs $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\forall$ et $\exists$.

- Souvent on écrit aussi $\exists$ v.α et $\forall$ v.α au lieu de $\exists$ v(α) et $\forall$ v(α)

Commentaire

- Les termes dénotent des objets alors que les prédicats dénotent des faits ou des relations qui sont vraies ou fausses entre ces objets.
- Souvent on appelle « Domaine » ou « Univers du discours » l'ensemble des objets qu'on peut décrire avec les termes qu'on s'est choisis.
- Par exemple, supposons que l'univers du discours est formé des objets (y compris les personnes) dans la classe.
  - On veut par exemple envoyer des robots intelligents dans la classe.
  - Ces derniers sont équipés d'algorithmes d'analyse d'images sophistiqués pour reconnaître les objets (y compris les personnes).

Commentaire

- Ainsi on aurait une constante pour chaque objet (et personne) dans la classe
  - *Robot1*, *Robot2*, *Projecteur*, *Tableau*, *Eric, Froduald*, *etc*.
- On aurait aussi des fonctions pour décrire de nouveaux objets en fonction d'autres objets
  - *age(Eric), taille(Froduald), position(Robot), etc.*
- Nous aurons aussi des prédicats pour décrire des relations entre ces objets
  - *EricAime(coursIA), Aime(Eric, coursIA), sentiment(Eric, coursIA, bon), sentiment(Eric, cours(IA), bon)*

Commentaire

- On utilise des connecteurs logiques pour décrire des expressions arbitrairement complexes:
  - *Jean aime bien le cours d'IA mais n'aime pas les TPs du cours*
    - $Aime(Jean, coursIA) \wedge \neg Aime(Jean, TP(coursIA))$
  - *Si quelqu'un aime un cours, il aime aussi les TPs*
    - $\forall x \ \forall y \ (Aime(x, y) \rightarrow Aime(x, TP(y)))$
  - *Si quelqu'un est inscrit à l'Université, il aime au moins un cours*
    - $\forall x \ (Inscrit(x) \rightarrow \exists y \ Aime(x, y))$
- *Le calcul des prédicats est un langage. Les expressions (phrases) précises dépendront de l'application.*

Commentaire

- En théorie, on pourrait remplacer la quantification par une conjonction très longue, du moins pour des cas finis.
- Par exemple, considérons l'expression « toutes les personnes sont mortelles »
- Une façon d'exprimer la même chose est de considérer toutes les personnes individuellement:
  - $Mortel(Personne_1) \wedge \ldots Mortel(Personne_n)$
- Avec les quantificateurs, l'expression devient plus simple :
  - $\forall x\ (Personne(x) \rightarrow Mortel(x))$
- *Dans certaines applications, la quantification est juste une notation pour une conjonction finie (quantification bornée).*

Exercice : conversions de phrases en formules

1. Marcus est une personne.
2. Marcus est un pompéien.
3. Tous les pompéiens sont des romains.
4. César est un dirigeant.
5. Tout le monde est loyal à quelqu'un.
6. Tous les romains sont loyaux à César ou le haïssent.
7. Les seuls dirigeants qu'une personne essaie d'assassiner sont ceux auxquels elle n'est pas loyal
8. Marcus a essayer d'assassiner César.

Les convertir en formules de logique du premier ordre

1. $personne(Marcus)$
2. $pompeien(Marcus)$
3. $\forall x(pompeien(x) \rightarrow romain(x))$
4. $dirigeant(Cesar)$
5. $\forall x \exists y\ loyal(x,y)$
6. $\forall x(romain(x) \rightarrow loyal(x,Cesar) \vee hait(x,Cesar))$
7. $\forall x \forall y((personne(x) \wedge dirigeant(y) \wedge assassiner(x,y)) \rightarrow \neg loyal(x,y))$
8. $assassiner(Marcus,Cesar)$

Sémantique

- On interprète une formule en lui assignant un sens, c'est-à-dire une valeur « vrai » ou « faux ».
- Le sens d'une formule dépend de la signification de prédicats, des connecteurs ainsi que des quantificateurs qui la constitue :
  - La signification des prédicats dépend du contexte (de l'état), c.-à-d., de la définition du « prédicat » et des termes qui sont ses arguments.
  - La signification des connecteurs (*et, ou, implique*) et des quantificateurs (*pour tout, il existe*) est fixée une fois pour toute; elle est indépendante du contexte ou de l'application.
- Puisque la signification des connecteurs et des quantificateurs est fixée, une *interprétation* (c-à-d., l'assignation d'un sens) d'une formule revient à une interprétation des prédicats.

Interprétation des fonctions et des prédicats

- *Chaque fonction doit être bien définie*
  - On peut utiliser des fonctions du langage de programmation sous-jacent.
  - Ou toute autre fonction définie par l'usager.
- *Les prédicats sont interprétés (définies) en fonction de l'application*
  - Intuitivement, on peut comprendre chaque prédicat *p* à *n* arguments comme une table à *n* attributs dans une base de données relationnelle:
    - $p(t_1, \ldots, t_n)$ est vrai si $p(t_1, \ldots, t_n)$ est dans la base de données.
  - En pratique, on interprète chaque prédicat simplement via une fonction *evalPredicate(p)* qui va prendre le prédicat *p* comme argument et retourne « vrai » ou « faux » selon que le prédicat est vrai ou faux.

Interprétation des formules

- Pour évaluer une formule arbitraire, nous utilisons la fonction *evalFormula* en considérant les prédicats comme le cas de base.
- Étant donné une formule arbitraire *f*, *evalFormula(f)* fonctionne comme suit:
  - si *f* est un prédicat : retourne *evalPredicate(f);*
  - si f est de la forme $\neg f_1$: si *evalFormula*(f_1) retourne « vrai » alors retourne « faux » sinon retourne « vrai »;
  - si *f* est de la forme $f_1 \wedge f_2$: si *evalFormula*(f_1) retourne « vrai » et *evalFormula(f2)* retourne « vrai », alors retourne « vrai »; sinon retourne « faux »;
  - si *f* est de la forme $\forall x\,(f_1)$: si *evalFormula*(f_2) retourne « vrai » pour chaque constante *c* et chaque formule f_2 obtenue de f_1 en remplaçant chaque *occurrence libre de x* par *c*, alors retourne « vrai »; sinon retourne « faux »;
- Les cas $\vee$, $\rightarrow$ et $\exists$ sont traités par des macros.

Validité, satisfiabilité, consistance et conséquence logique

- Si une formule est vraie dans toutes les interprétations (c'est-à-dire, quelque soit la valeur de *evalPredicate()*, on dit qu'elle ***valide***. Exemple: $P(a) \vee \neg P(a)$
- Si une formule est fausse dans toutes les interprétations possibles (c'est-à-dire, quelque soit la valeur de *evalPredicate()*, on dit qu'elle est ***inconsistante***. Exemple: $P(a) \wedge \neg P(a)$
- Si une formule est vraie dans quelques interprétations, on dit qu'elle est ***satisfiable*** (par les interprétations qui la rendent vraie).
  - Exemple: $\exists x\ (Inscrit(x) \wedge Aime(x, coursIA))$
- Une formule f_2 est une ***conséquence logique*** d'une formule f_1 si toute interprétation satisfaisant f_1 satisfait aussi f_2

Processus d'inférence

- Les *processus d'inférence* sont des processus qui permettent de déduire des formules qui sont des conséquences logiques d'autres formules.
- Ils sont considérés comme des processus d' « intelligence ».
- Un bon processus d'inférence doit être ***correcte*** (« *sound* »), c-à-d., toute formule déduite d'un ensemble de formules doit être une conséquence logique de ces formules.
- Un processus d'inférence est ***complet*** si il est capable de déduire toute formule qui est une conséquence logique d'autres formules.

Règles d'inférence: *Modus ponens*, instanciation universelle

- Modus ponens.
  - A partir d'un ensemble $\{f_1,\ f_1 \rightarrow f_2\}$ on déduit f_2.

- Instantiation universelle.
  - A partir de $\forall x\ (f_1)$ on déduit f_2 obtenu de f_1 en remplaçant toutes les *occurrences libres* de *x* par un terme n'ayant pas de variable en commun avec f_1.

Substitutions

- Un **littéral** est un prédicat ou la négation d'un prédicat.
  - Exemples : $p_1(x, y)$, $\neg p_1(x, y)$
- Une **clause** est un ensemble de littéraux.
  - Exemples : $p_1(x, y) \vee p_2(x, y, z) \vee p_1(x, z)$
- Une **substitution** θ est un ensemble (possiblement vide) de pairs noté $\theta=\{x_1=t_1,\ldots,x_n=t_n\}$ où les x_i sont des variables, les t_i sont des termes, et les x_i sont distincts.

Substitutions

- L'application d'une substitution $\theta=\{x_1=t_1, \ldots x_n=t_n\}$ à un littéral α donne un littéral $\alpha\theta$ obtenu de θ en remplaçant simultanément toute occurrence de x_i par t_i dans α, pour chaque paire $x_i=t_i$.

- $\alpha\theta$ est appelé instance de α pour θ.

- Exemple :

$$\alpha = p(x, y, f(a)) \text{ et } \theta=(x=b, y=x)$$
$$\alpha\theta = p(b, x, f(a))$$

- Si S est la clause $\{\alpha_1, \ldots, \alpha_n\}$
  alors $S\theta=\{\alpha_1\theta, \ldots, \alpha_n\theta\}$

Composition de substitutions

- Soit $\theta=\{x_1=s_1, \ldots x_m=s_m\}$ et $\sigma=\{y_1=t_1, \ldots y_n=t_n\}$.
- La composition de et est la substitution obtenue comme suit :
  1. construire l'ensemble;

     $\{x_1=s_1\sigma, \ldots x_m=s_m\sigma, y_1=t_1, \ldots y_n=t_n\}$

     En appliquant à tous les termes s_i.
  2. supprimer les identités, c-à-d. les paires pour lesquelles est devenu x_i;
  3. supprimer toutes les paires $y_i=t_i$ telles que $y_i \in \{x_1, \ldots, x_m\}$
- Exemple :

  $\theta=\{x=f(y), y=z)$, $\sigma=\{x=a, y=b, z=y\}$

  $\theta\sigma=\{x=f(b), z=y\}$

Propriétés des substitutions

1. La substitution identité, noté ε, est l'ensemble vide {}.
2. $\theta\varepsilon = \theta$, pour toute substitution θ.
3. $(\alpha\sigma)\theta = \alpha(\sigma\theta)$, pour toute clause α et substitutions σ et θ.
4. $(\gamma\sigma)\theta = \gamma\ (\sigma\theta)$, pour toutes substitutions γ, σ et θ.

Unification

- Soit s = (α_1, α_2) une paire de littéraux.
- Une substitution θ est appelé un **unificateur** de S si $\alpha_1\theta = \alpha_2\theta$.
- Un unificateur θ de S est appelé **unificateur le plus général (upg)** si pour tout unificateur σ de S, il existe une substitution γ, telle que $\sigma=\theta\gamma$.
- Exemple :
  - $p(f(x), a)$ et $p(y, f(w))$ ne sont pas unifiable.
  - $p(f(x), z)$ et $p(y, a)$ a pour upg $\sigma=(y=f(x), z=a)$.
- On appelle ensemble de désaccords entre deux littéraux la paire des premiers termes pour lesquels les deux littéraux diffèrent.

Algorithme d'unification

Algorithme Unify(S)

1. k=0; $\sigma_0=\varepsilon$
2. Si σ_k est un unificateur pour S,
   alors return σ_k; // σ_k est le upg de S.
   sinon calculer Dk l'ensemble de désaccord de $S\sigma_k$
3. Si il existe une paire v=t tel que v est une variable dans Dk et n'apparait pas dans t, et tel que {v=t} est un unificateur pour Dk,
   Alors : $\sigma_{k+1}=\sigma_k$ {v=t}, k=k+1; Goto 2.
   Sinon return Échec. // S n'est pas unifiable

Conversion d'une formule sous forme clausale normale

- **Étape 1 : Éliminer les implications.**
  Utiliser l'équivalence :
  $$\alpha \rightarrow \beta \quad \equiv \quad \neg\, \alpha \vee \beta$$
  Pour enlever toutes les implications dans la formule.

- **Étape 2 : Réduire la portée de ¬**
  Utiliser les lois de Morgan :
  1. $\neg(f_1 \vee f_2) \equiv (\neg\, f_1 \wedge \neg\, f_2)$
  2. $\neg(f_1 \wedge f_2) \equiv (\neg\, f_1 \vee \neg\, f_2)$
  3. $\neg(\neg f) = f$

  de sorte que ¬ soit toujours suivi d'un prédicat.

Conversion d'une formule sous forme clausale normale

- **Étape 3 : Standardiser les variables (localement)**

  C'est-à-dire, renommer les variables de telle sorte qu'aucune paire de quantificateur ne porte sur la même variable.
- **Étape 4 : Éliminer les quantificateurs existentiels**

  C'est-à-dire, chaque quantificateur existentiel est éliminé, en ayant remplacé sa variable par une fonction skolem dont les arguments sont les variables quantifiées par des quantificateurs universels contenant le quantificateur qu'on veut éliminer dans leur porté.

Conversion d'une formule sous forme clausale normale

- **Étape 5 : Mettre en forme prenex**

  C'est-à-dire, mettre tous les quantificateurs universels en tête. La partie sans quantificateur est appelée une matrice.
- **Étape 6 : Mettre la matrice en forme normale conjonctive**

  C'est-à-dire sous forme de disjonction de conjonctions de littéraux, en utilisant les équivalences de distributivité :

  $f_1 \vee (f_2 \wedge f_3) \equiv \quad (f_1 \vee f_2) \wedge (f_1 \vee f_3)$
- **Étape 7 : Éliminer les quantificateurs universels.**

Conversion d'une formule sous forme clausale normale

- **Étape 8 : Éliminer les conjonctions**
  - On obtient ainsi un ensemble de clauses.

- **Étape 9 : Standardiser les variables (globalement)**
  - Renommer les variables de telle sorte que deux clauses différentes n'aient pas les mêmes variables.

Preuve par résolution

- Pour prouver que f_1 implique f_2 :
  - Transformer f_1 en un ensemble de clauses;
  - Ajouter les clauses pour non f_2;
  - Appliquer répétitivement la règle de résolution afin d'obtenir la clause vide, notée {}.
    - Rappel : une clause est une conjonction. Une clause vide équivaut à *false*.

Règle de résolution pour les prédicats

- Soit deux clauses L et M, tel que :
  - $M = \{M_1, \ldots, M_m\}$
  - $L = \{L_1, \ldots, L_n\}$
- Trouver un littéral L_k et un littéral M_l tel qu'il existe un unificateur le plus général θ, et $L_k\theta = \neg M_l\theta$.
- La clause résolvante de L et M est :

$$(L\theta - \{L_k\theta\}) + (M\theta - \{M_k\theta\})$$

Répondre à des questions

- La résolution permet de répondre à une question dont la réponse est oui ou non, à savoir si f_1 est une conséquence logique de f_2.
- On peut aussi exploiter les traces du processus de preuve pour trouver des valeurs (instanciations) qui permettent de déduire que f_2 est une conséquence logique de f_1 :
  - On ajoute $Rep(x_1, \ldots, x_n)$ à chaque clause résultante de f_2, avec xi les variables apparaissant dans la clause.
  - On applique la résolution
  - On arrête lorsqu'on n'a une clause composées uniquement de littéraux *Rep*(…)

Factorisation des clauses

- Si un sous-ensemble de littéraux dans une clause ont un *unificateur le plus général (upg) :* remplacer cette clause par son *facteur*.
- Le facteur d'une clause est la clause obtenue, en appliquant l'upg et en supprimant les littéraux redondant.
- Par exemple, la clause $p(x) \vee p(f(y))$ a comme facteur p(f(y)) (obtenu par l'*upg* {x = f(y) }.
- Les deux clauses $p(x) \vee p(f(y))$ *et* $\neg p(w) \vee \neg p(f(z))$ sont contradictoires, pourtant la preuve par résolution n'aboutit pas à une contradiction.
- Par contre en remplaçant $p(x) \vee p(f(y))$ par son facteur, ça marche.
- Il faut systématique ajouter la factorisation aux étapes de la résolution annoncées précédemment pour qu'elle soit complète.

Exemple 1

- Tous les chiens sont des animaux

  $\forall x\ (dog\ (x) \rightarrow animal\ (x))$

- Tous les animaux vont mourir

  $\forall y\ (animal\ (y) \rightarrow die\ (y))$

- Fido est un chien

  $dog\ (Fido)$

- Prouvez que Fido va mourir

  die(Fido)

Exemple 1

Format normal

1. $\forall x(dog(x) \rightarrow animal(x))$
2. $\forall y(animal\ (y) \rightarrow die(y))$
3. *dog (Fido)*

Niez la conclusion que Fido va mourir

4. $\neg$ die(Fido)

Format clausale

1. $\neg$ *dog (x)* $\vee$ *animal (x)*
2. $\neg$ *animal (y)* $\vee$ *die (y)*
3. *dog (Fido)*
4. $\neg$ die(Fido)

---

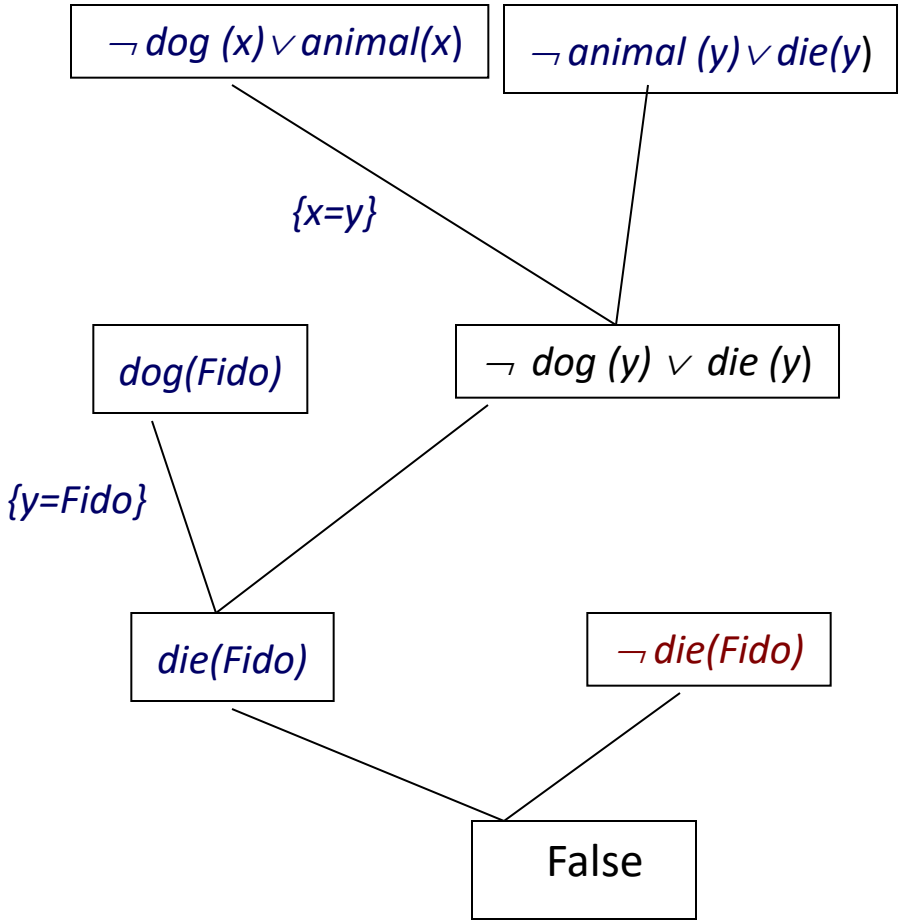
5. $\neg$ *dog (y)* $\vee$ *die (y)*

   1, 2, {x=y}
6. *die(fido)*

   3, 5 {y=fido)
7. *False.*

   4, 6

Exemple 1



1. ¬ dog (x) ∨ animal (x)
2. ¬ animal (y) ∨ die (y)
3. dog (Fido)
4. ¬ die(Fido)

---

5. ¬ dog (y) ∨ die (y) : 1, 2, {x=y}
6. die(fido) : 3, 5 {y=fido)
7. False : 4, 6

Exemple 2

1. Marcus est une personne.
2. Marcus est un pompéien.
3. Tous les pompéiens sont des romains.
4. César est un dirigeant.
5. Tout le monde est loyal à quelqu'un.
6. Tous les romains sont loyaux à César ou le haïssent.
7. Les seuls dirigeants qu'une personne essaie d'assassiner sont ceux auxquels elle n'est pas loyal
8. Marcus a essayer d'assassiner César.

Prouvez que Marcus hait César

1. $personne(Marcus)$
2. $pompeien(Marcus)$
3. $\forall x(pompeien(x) \rightarrow romain(x))$
4. $dirigeant(Cesar)$
5. $\forall x \exists y\ loyal(x,y)$
6. $\forall x(romain(x) \rightarrow loyal(x,Cesar) \vee hait(x,Cesar))$
7. $\forall x \forall y((personne(x) \wedge dirigeant(y) \wedge assassiner(x,y)) \rightarrow \neg loyal(x,y))$
8. $assassiner(Marcus,Cesar)$

Prouvez : hait(Marcus,Cesar)

Étape 1 : éliminer l'implication

1. personne(Marcus)
2. pompeien(Marcus)
3. $\forall x(pompeien(x) \rightarrow romain(x))$
4. dirigeant(Cesar)
5. $\forall x \exists y$ loyal(x,y)
6. $\forall x(romain(x) \rightarrow loyal(x,Cesar) \vee hait(x,Cesar))$
7. $\forall x \ \forall y((personne(x) \wedge dirigeant(y) \wedge assassiner(x,y)) \rightarrow \neg loyal(x,y))$
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. $\forall x(\neg pompeien(x) \vee romain(x))$
4. $dirigeant(Cesar)$
5. $\forall x \exists y$ loyal(x,y)
6. $\forall x(\neg romain(x) \vee loyal(x,Cesar) \vee hait(x,Cesar))$
7. $\forall x \ \forall y(\neg (personne(x) \wedge dirigeant(y) \wedge assassiner(x,y)) \vee \neg loyal(x,y))$
8. assassiner(Marcus,Cesar)

Étape 2 : réduire la porte de ¬

1. personne(Marcus)
2. pompeien(Marcus)
3. ∀ x(¬ pompeien(x) ∨ romain(x))
4. dirigeant(Cesar)
5. ∀x∃y loyal(x,y)
6. ∀ x(¬ romain(x)∨ loyal(x,Cesar)∨ hait(x,Cesar))
7. *∀x ∀y(¬ (personne(x) ∧ dirigeant(y) ∧ assassiner(x,y)) ∨ ¬ loyal(x,y))*
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. ∀ x(¬ pompeien(x) ∨ romain(x))
4. dirigeant(Cesar)
5. ∀x∃y loyal(x,y)
6. ∀ x(¬ romain(x)∨ loyal(x,Cesar)∨ hait(x,Cesar))
7. *∀x ∀y(¬ personne(x) ∨ ¬ dirigeant(y) ∨ ¬ assassiner(x,y) ∨ ¬ loyal(x,y))*
8. assassiner(Marcus,Cesar)

Étape 3 : standardiser les variables

1. personne(Marcus)
2. pompeien(Marcus)
3. $\forall x(\neg pompeien(x) \vee romain(x))$
4. dirigeant(Cesar)
5. $\forall x \exists y\ loyal(x,y)$
6. $\forall x(\neg romain(x) \vee loyal(x,Cesar) \vee hait(x,Cesar))$
7. $\forall x \forall y(\neg personne(x) \vee \neg dirigeant(y) \vee \neg assassiner(x,y) \vee \neg loyal(x,y))$
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. $\forall x1(\neg pompeien(x1) \vee romain(x1))$
4. dirigeant(Cesar)
5. $\forall x2 \exists x3\ loyal(x2,x3)$
6. $\forall x4(\neg romain(x4) \vee loyal(x4,Cesar) \vee hait(x4,Cesar))$
7. $\forall x5 \forall x6(\neg personne(x5) \vee \neg dirigeant(x6) \vee \neg assassiner(x5,x6) \vee \neg loyal(x5,x6))$
8. assassiner(Marcus,Cesar)

Étape 4 : éliminer les quantificateurs existentiels

1. personne(Marcus)
2. pompeien(Marcus)
3. ∀x1(¬ pompeien(x1) ∨ romain(x1))
4. dirigeant(Cesar)
5. *∀x2∃x3 loyal(x2,x3)*
6. ∀ x4(¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar))
7. ∀x5∀x6(¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6))
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. ∀x1(¬ pompeien(x1) ∨ romain(x1))
4. dirigeant(Cesar)
5. *∀x2 loyal(x2, f1(x2))*
6. ∀ x4(¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar))
7. ∀x5∀x6(¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6))
8. assassiner(Marcus,Cesar)

Étape 5 : mettre les formules en forme prenex

1. personne(Marcus)
2. pompeien(Marcus)
3. ∀x1(¬ pompeien(x1) ∨ romain(x1))
4. dirigeant(Cesar)
5. ∀x2 loyal(x2, f1(x2))
6. ∀ x4(¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar))
7. ∀ x5∀ x6(¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6))
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. ∀x1(¬ pompeien(x1) ∨ romain(x1))
4. dirigeant(Cesar)
5. ∀x2 loyal(x2, f1(x2))
6. ∀ x4(¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar))
7. ∀x5∀x6(¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6))
8. assassiner(Marcus,Cesar)

Aucun changement dans ce cas-ci

Prenex = mettre tous les quantificateurs universels en tête.

Étape 6 : mettre la matrice sous forme normale conjonctive

1. personne(Marcus)
2. pompeien(Marcus)
3. $\forall$x1($\neg$ pompeien(x1) $\vee$ romain(x1))
4. dirigeant(Cesar)
5. $\forall$x2 loyal(x2, f1(x2))
6. $\forall$ x4($\neg$ romain(x4)$\vee$ loyal(x4,Cesar)$\vee$ hait(x4,Cesar))
7. $\forall$ x5$\forall$ x6($\neg$ personne(x5) $\vee$ $\neg$ dirigeant(x6) $\vee$ $\neg$ assassiner(x5,x6) $\vee$ $\neg$ loyal(x5,x6))
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. $\forall$x1($\neg$ pompeien(x1) $\vee$ romain(x1))
4. dirigeant(Cesar)
5. $\forall$x2 loyal(x2, f1(x2))
6. $\forall$ x4($\neg$ romain(x4)$\vee$ loyal(x4,Cesar)$\vee$ hait(x4,Cesar))
7. $\forall$ x5$\forall$x6($\neg$ personne(x5) $\vee$ $\neg$ dirigeant(x6) $\vee$ $\neg$ assassiner(x5,x6) $\vee$ $\neg$ loyal(x5,x6))
8. assassiner(Marcus,Cesar)

Aucun changement dans ce cas-ci

Étape 7 : éliminer les quantificateurs universels

1. personne(Marcus)
2. pompeien(Marcus)
3. $\forall$x1($\neg$ pompeien(x1) $\vee$ romain(x1))
4. dirigeant(Cesar)
5. $\forall$x2 loyal(x2, f1(x2))
6. $\forall$ x4($\neg$ romain(x4)$\vee$ loyal(x4,Cesar)$\vee$ hait(x4,Cesar))
7. $\forall$ x5$\forall$x6($\neg$ personne(x5) $\vee$ $\neg$ dirigeant(x6) $\vee$ $\neg$ assassiner(x5,x6) $\vee$ $\neg$ loyal(x5,x6))
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. $\neg$ pompeien(x1) $\vee$ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. $\neg$ romain(x4)$\vee$ loyal(x4,Cesar)$\vee$ hait(x4,Cesar)
7. $\neg$ personne(x5) $\vee$ $\neg$ dirigeant(x6) $\vee$ $\neg$ assassiner(x5,x6) $\vee$ $\neg$ loyal(x5,x6)
8. assassiner(Marcus,Cesar)

Étape 8 : éliminer les conjonctions

1. personne(Marcus)
2. pompeien(Marcus)
3. ¬ pompeien(x1) ∨ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. ¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar)
7. ¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6)
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. ¬ pompeien(x1) ∨ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. ¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar)
7. ¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6)
8. assassiner(Marcus,Cesar)

Aucun changement dans ce cas-ci

Étape 9 : standardiser les variables

1. personne(Marcus)
2. pompeien(Marcus)
3. ¬ pompeien(x1) ∨ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. ¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar)
7. ¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6)
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. ¬ pompeien(x1) ∨ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. ¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar)
7. ¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6)
8. assassiner(Marcus,Cesar)

Aucun changement dans ce cas-ci

Étape 10 : Ajouter les clauses de la *négation* l'expression à prouver

1. personne(Marcus)
2. pompeien(Marcus)
3. ¬ pompeien(x1) ∨ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. ¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar)
7. ¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6)
8. assassiner(Marcus,Cesar)

1. personne(Marcus)
2. pompeien(Marcus)
3. ¬ pompeien(x1) ∨ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. ¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar)
7. ¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6)
8. assassiner(Marcus,Cesar)
9. ***¬ hait(Marcus, Cesar)***

Étape 11 : Appliquer la résolution itérativement jusqu'à la clause vide

1. personne(Marcus)
2. pompeien(Marcus)
3. ¬ pompeien(x1) ∨ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. ¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar)
7. ¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6)
8. assassiner(Marcus,Cesar)
9. ¬ hait(Marcus, Cesar)
10. romain(Marcus)

    2, 3, {x1=Marcus}
11. loyal(Marcus,Cesar)∨hait(Marcus,Cesar)

    6, 10, {x4=Marcus}
12. loyal(Marcus,Cesar)

    9, 11
13. ¬ personne(Marcus) ∨ ¬ dirigeant(Cesar) ∨ ¬ assassiner(Marcus,Cesar)

    7, 12, {x5=Marcus, x6=Cesar}
14. ¬ personne(Marcus) ∨ ¬ dirigeant(Cesar)

    8,13
15. ¬ personne(Marcus)

    4,14
16. False

    1, 15 (clause vide)

Exemple 3. Répondre à la question : qui hait César?

1. personne(Marcus)
2. pompeien(Marcus)
3. ¬ pompeien(x1) ∨ romain(x1)
4. dirigeant(Cesar)
5. loyal(x2, f1(x2))
6. ¬ romain(x4)∨ loyal(x4,Cesar)∨ hait(x4,Cesar)
7. ¬ personne(x5) ∨ ¬ dirigeant(x6) ∨ ¬ assassiner(x5,x6) ∨ ¬ loyal(x5,x6)
8. assassiner(Marcus,Cesar)
9. ***¬ hait(x7,Cesar) ∨ Rep(x7)***

La 9ème clause est obtenue comme suit:

- la clause à prouver est : *∃x hait(x,Cesar)*
- *sa négation est : ∀x ¬ hait(x,Cesar)*
- *ce qui donne après standaradisation des variables : ¬ hait(x7,Cesar)*
- *on ajoute : Rep(x7)*

10. romain(Marcus)

2, 3, {x1=Marcus}

11. loyal(Marcus,Cesar)∨hait(Marcus,Cesar)

6, 10, {x4=Marcus}

12. loyal(Marcus,Cesar) *∨ Rep(Marcus)*

9, 11 {x7=Marcus}

13. ¬ personne(Marcus) ∨ ¬ dirigeant(Cesar) ∨ ¬ assassiner(Marcus,Cesar) *∨ Rep(Marcus)*

7, 12, {x5=Marcus, x6=Cesar}

14. ¬ personne(Marcus) ∨ ¬ dirigeant(Cesar) *∨ Rep(Marcus)*

8,13

15. ¬ personne(Marcus) *∨ Rep(Marcus)*

4,14

16. *Rep(Marcus)*

1,15

Réponse: Marcus

Fin !