IFT 615 – Intelligence artificielle

Logique du premier ordre

Hugo Larochelle
Département d'informatique
Université de Sherbrooke
http ://www.dmi.usherb.ca/~larocheh/cours/ift615.html

Objectifs

- Comprendre ce qu'est la logique de premier ordre
  - connaître la syntaxe
  - savoir décrire des faits sous forme de logique du premier ordre

- Savoir faire du raisonnement déductif en logique du premier ordre
  - prouver qu'un « nouveau » fait est une conséquence logique d'une base initiale de faits, à l'aide de la preuve par résolution

Logique du premier ordre : un langage

- Avec la recherche heuristique, nous pouvons résoudre des problèmes qui se traduisent facilement en une recherche dans un graphe d'états

- Pour résoudre des problèmes plus complexes, qui demandent par exemple des connaissances d'un expert, nous avons besoin en plus d'un langage permettant :
  - de **représenter les connaissances** d'un expert facilement
  - de **faire des déductions logiques** avec ces connaissances

- La **logique du premier ordre** (appelé aussi le « **calcul des prédicats** ») est la base de plusieurs formalismes de représentation des connaissances et du **raisonnement déductif** utilisé entre autres par les systèmes experts

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 loyale.
  8. Marcus a essayé 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.

Exemples de raisonnement déductif

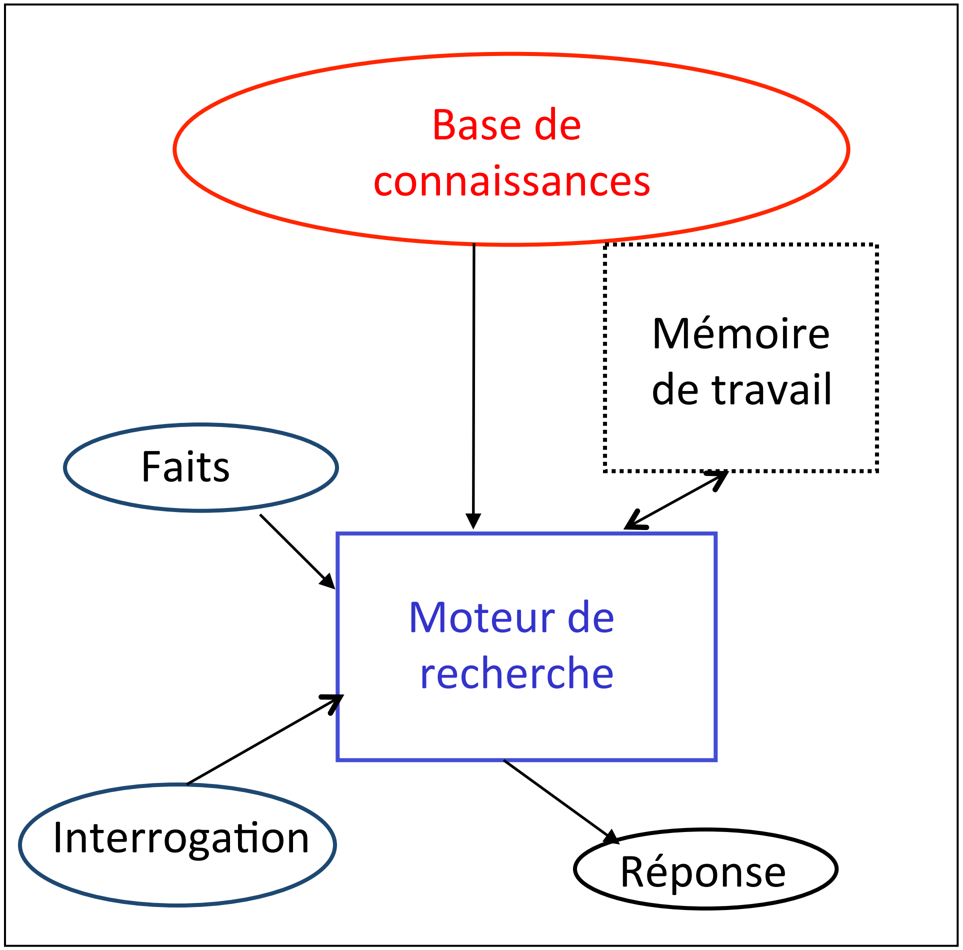
- Trouver un plan d'action permettant d'accomplir un but, à partir de :
  1. Actions possibles d'un agent (robot, personnage d'un jeu comme le *wumpus world*, etc.).
  2. But à accomplir.

- Beaucoup d'autres applications :
  - web sémantique
  - programmation logique

- Dans la plupart des applications, ce n'est pas la logique en tant que telle qui est utilisée

- Ce sont plutôt des langages et des algorithmes d'inférence inspirés plus ou moins de la logique du premier ordre

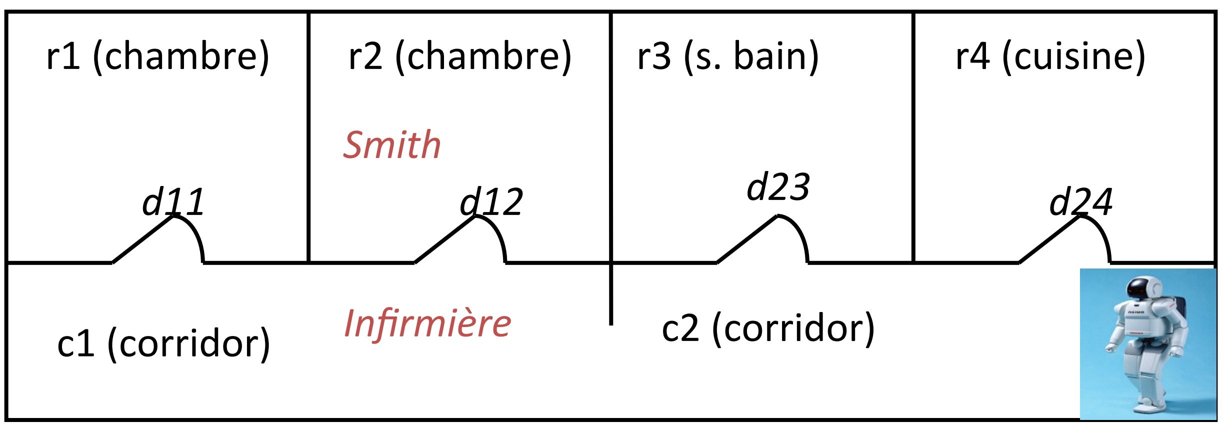
Exemple : schéma d'un système expert à base de règles de production



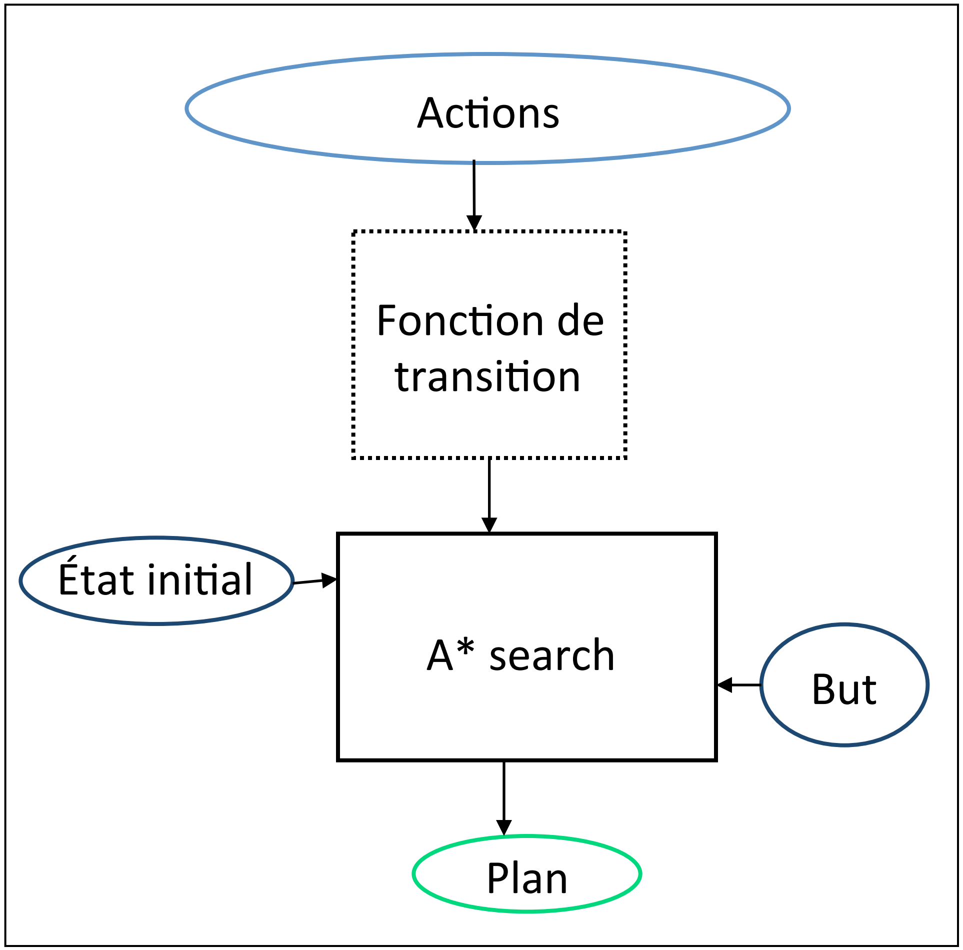
- La base de connaissances est spécifiée par des règles logiques
- Les faits sont des propositions logiques
- La mémoire de travail peut être vue comme un état
- Le moteur de recherche est une exploration de l'espace d'états
- Exemple :
  - Java Expert System Shell (JESS)

Exemple : planification en IA

- Rôle de la planification
  - on dit au robot quoi faire (le but)
    - exemple : transporter des objets d'un endroit à un autre
  - la liste des opérations à faire pour accomplir le but n'est pas codée d'avance
    - le robot utilise un planificateur pour déterminer les actions à prendre
  - une formulation sous forme de logique permet à un système unique de faire **plusieurs tâches différentes** (suffit de changer le but)



Architecture d'un planificateur basé sur A*



- Les actions, l'état et le but sont décrits dans un formalisme/langage inspiré de la logique
- Actions décrites selon Planning Domaine Definition Language (PDDL) :
  - identificateur
  - précondition / contraintes
  - effets
- Voir cours **IFT 702 - Planification en intelligence artificielle** pour plus sur ce sujet

Logique du premier ordre : un modèle mathématique du raisonnement déductif

- À l'origine, la logique du premier ordre est un modèle mathématique du raisonnement déductif

- Généralement, pour construire un modèle d'un objet réel, on détermine les caractéristiques principales de l'objet et on crée un modèle ou une maquette ayant ces caractéristiques

- Dans notre cas, on veut modéliser le raisonnement déductif, c'est-à-dire le processus mental permettant d'inférer des « expressions correctes » à partir des faits et d'autres expressions correctes

- La capacité de modéliser le raisonnement déductif, même de manière approximative, nous permettra par la suite de le programmer dans un système expert, par exemple

Caractéristiques principales du raisonnement déductif

- Une partie de la véracité d'une expression dépend uniquement des faits vrais (prémisses) dans une situation donnée
  - Toutes les personnes sont mortelles.
  - Le patient a une température de 41 degrés Celsius.
  - La voiture ne démarre pas.

- Une autre partie dépend uniquement de la syntaxe de l'expression
  - **Si** être une personne implique qu'on est mortel **et si** Dupont est une personne **alors** Dupont est mortel
  - **Si** $P(x)$ implique M(x) pour tout x **et si** $P(a)$ **alors** $M(a)$

- Dans le dernier cas, la valeur logique des expressions dépend uniquement de leur forme (syntaxe)
  - elle est totalement indépendante de leur contenu

Reste de la leçon

- En premier lieu nous allons voir la **syntaxe**, c'est-à-dire la forme des expressions qu'on peut écrire dans la logique du premier ordre

- Ensuite 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 en logique du premier ordre est appelée une **formule** (*sentence*)
- Les formules sont des combinaisons de **prédicats**, à l'aide de
  - **connecteurs logiques :** et, ou, etc.
  - **quantificateurs :** il existe, pour tout
- Les **prédicats** décrivent des faits (vrai ou faux), qui correspondent souvent à des relations entre des objets
- Les **objets** sont décrits par des **termes** :
  - **constantes :** *Caesar*, *Marcus*, etc.
  - **variables :** *x*, *y*, etc.
  - **fonctions :** *jambeGauche*(*Marcus*)
- Les **prédicats**, les **connecteurs logiques**, les **quantificateurs** et les **termes** sont décrits par des **symboles**

Symboles

- **Constantes** : *41*, *Dupont*, *Robot1*
- **Fonctions :** *temperature*(*x*), *position*(*x*)
- **Prédicats :** *mortel*(*x*), *plusGrand*(*x*,*y*), *partieTerminée*
  - le nombre d'arguments d'une fonction ou d'un prédicat est appelé **arité**
  - les **prédicats ne sont pas des 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 (ils sont à la base des formules)
- **Variables :** *x*, *y*, *z*
- **Connecteurs :** $\neg$ (non), $\wedge$ (et), $\vee$ (ou), $\rightarrow$ (implique)
- **Quantificateurs :** $\forall$ (pour tout), $\exists$ (il existe)

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, …, t_n sont des termes et f une fonction à n arguments, alors $f(t_1, …, t_n)$ est aussi un terme
  - par exemple : *pere*(*John*), *pere*(*x*), *pere*(*pere*(*x*))

- On pourrait éviter les fonctions en définissant une constante par argument possible de la fonction
  - *pereJohn* et *pereLouis* à la place de *pere*(*John*) et *pere*(*Louis*)
  - par contre, on perd la possibilité de raisonner de façon générale à l'aide des variables
    - » $\forall\, x\, \forall\, y\ sontFreres(x,y) \rightarrow egaux(pere(x),pere(y))$

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 la formule la plus simple qui soit (cas de base)
- La **négation**, la **conjonction**, la **disjonction** et l'**implication** de formules sont aussi des formules
  - plus précisément, si α et β sont des formules, alors $\neg\, \alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$ et $\alpha \rightarrow \beta$ sont des formules
- La **quantification universelle** et **la quantification existentielle** d'une formule est une formule
  - plus précisément, si α est une formule et x est une variable, alors $\forall\, x\, \alpha$ et $\exists\, x\, \alpha$ sont des formules

Notations

- **Priorités et parenthèses**
  - ordre des priorités : $\neg$, $\wedge$, $\vee$, $\rightarrow$
  - on utilise les parenthèses de la même façon que dans les expressions arithmétiques pour éviter les ambiguïtés
  - les quantifieurs s'appliquent à toute la formule à sa droite
    - ex. : . $\forall x\ \ p(x) \vee q(x) \rightarrow r(x)$ est équivalent à
      $\forall x\ \ (p(x) \vee q(x) \rightarrow r(x))$ et non
      $(\forall x\ \ p(x)) \vee q(x) \rightarrow r(x)$
- **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 $\neg \forall v \neg \alpha$

Remarques

- Les **termes dénotent des objets** alors que les **prédicats dénotent des relations** qui sont vraies ou fausses entre ces objets

- Les notations spécifiques des symboles sont selon les conventions ou les goûts
  - dans le livre, on préfère des symboles qui débutent par une majuscule

Exercice

- Faits :
  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 essayé d'assassiner César.

- Forme 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. $assassiner(Marcus,Cesar)$

Processus d'inférence

- On a vu comment écrire une base de connaissance de faits sous la forme de formules logique du premier ordre
- On va maintenant voir comment déduire si une nouvelle formule est une conséquence logique de cette base de connaissance

- Les **processus d'inférence** sont des processus qui permettent de déduire des formules qui sont des conséquences logiques d'autres formules
  - 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

Exemples de règles d'inférence

- **Modus ponens**
  - à partir de f_1 et $f_1 \rightarrow f_2$, on déduit f_2
    - si on a (*wumpusAhead* ∧ *wumpusAlive*) → *shoot* et on a (*wumpusAhead* ∧ *wumpusAlive*), alors *shoot* peut être inféré

- **Instantiation universelle**
  - à 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
    - par exemple : tous les chiens sont des mammifères, Fido est un chien, donc Fido est un mammifère

Preuve par résolution

- Procédure générale pour faire de l'inférence
  - modus ponens et l'instantiation universelle sont des cas particuliers

- Cette procédure est correcte et complète (sous certaine condition, à voir plus tard)

- On aura besoin des outils suivants :
  - la **substitution**
  - l'**unification**
  - la **transformation sous forme normale conjonctive**

Substitution

- On définit un **littéral** comme un prédicat ou la négation d'un prédicat
  - ex. : $p_1(x, y)$, $\neg\, p_1(x, y)$
- On définit une **clause** comme une disjonction de littéraux
  - ex. : $p_1(x, y) \vee p_2(x, y, z) \vee \neg\, p_1(x, z)$
- Une **substitution** est un ensemble (possiblement vide) de paires de la forme $x_i = t_i$ où x_i est une variable et t_i est un terme et les x_i sont **distincts**.

Substitution

- 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 α 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 : $\alpha = \mathrm{p}(x, y, f(a))$, $\theta = \{y = x, x = b\}$

    $\alpha\theta = \mathrm{p}(b, x, f(a))$

- Si C est la clause $\alpha_1 \vee \ldots \vee \alpha_n$, $C\theta$ est la clause $\alpha_1\theta \vee \ldots \vee \alpha_n\theta$

Composition de substitutions

- Quelle serait la substitution équivalent à l'application successive de deux substitution $\theta = \{ x_1 = s_1,\ldots, x_m = s_m \}$ et $\sigma = \{ y_1 = t_1,\ldots, y_n = t_n \}$
  - on note une telle **composition** $\theta\sigma$

- La composition $\theta\sigma$ de θ et σ est la substitution obtenue comme suit :
  1. construire l'ensemble
     $$\{ x_1 = s_1\sigma,\ldots, x_m = s_m\sigma,\ y_1 = t_1,\ldots,y_m = t_n \}$$
     en appliquant σ à tous les termes s_i
  2. supprimer toutes les paires $y_i = t_i$ telles que $y_i \in \{x_1, \ldots, x_m\}$
  3. supprimer les identités, c-à-d., les paires pour lesquelles $s_i\sigma$ est devenu x_i

Composition de substitutions : exemple

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

Composition de substitutions : exemple

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

1. $\{\, x = f(y),\ y = z,\ x = a,\ y = b,\ z = y \,\}$

Composition de substitutions : exemple

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

1. $\{ x = f(b), y = y, x = a, y = b, z = y \}$

Composition de substitutions : exemple

- Composition de θ = $\{ x = f(y), y = z \}$ et σ = $\{ x = a, y = b, z = y \}$

1. $\{ x = f(b), y = y, x = a, y = b, z = y \}$

2. $\{ \underline{x} = f(b), \underline{y} = y, \cancel{\underline{x} = a}, \cancel{\underline{y} = b}, z = y \}$

Composition de substitutions : exemple

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

1. $\{ x = f(b), y = y, x = a, y = b, z = y \}$
2. $\{ \underline{x} = f(b), \underline{y} = y, \cancel{\underline{x} = a}, \cancel{\underline{y} = b}, z = y \}$
3. $\{ x = f(b), \cancel{y = y}, \cancel{x = a}, \cancel{y = b}, z = y \}$

Composition de substitutions : exemple

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

1. $\{ x = f(b), y = y, x = a, y = b, z = y \}$
2. $\{ \underline{x} = f(b), \underline{y} = y, \cancel{\underline{x} = a}, \cancel{\underline{y} = b}, z = y \}$
3. $\{ x = f(b), \cancel{y = y}, \cancel{x = a}, \cancel{y = b}, z = y \}$

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

Propriétés des substitutions

- La substitution identitée, notée ε, est l'ensemble vide
- $\theta\varepsilon = \theta$, pour toute substitution θ
- $(\alpha\sigma)\theta = \alpha(\sigma\theta)$, pour toute clause α et substitutions θ et σ
- $(\theta\sigma)\gamma = \theta(\sigma\gamma)$, pour toutes substitutions θ, σ et γ

Unification

- Soit S = {α_1, α_2} une paire de 2 littéraux, on aimerait trouvé une substitution θ qui **unifie** α_1 et α_2, c.-à-d. telle que $\alpha_1\theta = \alpha_2\theta$
  - ex. : { $p(f(x),z)$, $p(y,a)$ } sont unifiés par θ = {$y = f(x), z = a$} (a est un constante)

    $p(f(x),z)\ \theta = \boldsymbol{p(f(x),a)}$ et $p(y,a)\ \theta = \boldsymbol{p(f(x),a)}$

  - ex. : { $p(f(x),a)$, $p(y,f(w))$ } ne sont pas unifiables, puisqu'on ne peut pas substituer la constante a par $f(w)$

Unificateur le plus général

- Un **unificateur** θ de S est appelé **unificateur le plus général** (**UPG**) de S si pour tout unificateur σ de S, il existe une substitution γ telle que $\sigma = \theta\gamma$
  - ex. : $\theta = \{y = f(x), z = a\}$ est un UPG pour $\{ p(f(x),z), p(y,a) \}$
    - » $p(f(x),z)\,\theta = p(y,a)\,\theta = p(f(x),a)$
  - ex. : $\sigma = \{y = f(a), x = a, z = a\}$ est unificateur mais pas UPG pour $\{ p(f(x),z), p(y,a) \}$
    - » $p(f(x),z)\,\sigma = p(y,a)\,\sigma = p(f(a),a)$
  - la substitution $\gamma = \{x = a\}$ permet d'obtenir $\sigma = \theta\,\gamma = \{y = f(a), x = a, z = a\}$
  - aucune substitution γ permet d'obtenir $\theta = \sigma\,\gamma$
- On appelle **ensemble de désaccord** entre deux littéraux la paire des premiers termes des deux littéraux qui diffèrent (à partir de la gauche)
  - $\{ p(f(x),z), p(y,a) \}$: l'ensemble de désaccord est $\{ f(x), y \}$
  - $\{ p(f(x),z), p(f(x),a) \}$: l'ensemble de désaccord est $\{ z, a \}$

Unificateur le plus général

Algorithme UNIFICATEUR(*S*)

1. k=1; σ_1 =ε
2. **Si** σ_k est unificateur pour *S*,
   alors exit; σ_k est le UPG de *S*
   Sinon calculer D_k l'ensemble de désaccord de $S\sigma_k$
3. **Si** il existe une paire (v, t) telle que v est une **variable** dans D_k et n'apparaît pas dans t, $\{v = t\}$ est un unificateur pour D_k,
   alors $\sigma_{k+1} = \sigma_k\{v = t\}$, k=k+1;
   retourner à 2.
   Sinon exit; *S* n'est pas unifiable.

Unificateur le plus général : exemple 1

- Trouver l'UPG de $p(x, f(x), y)$ et $p(y, z, u)$

- **Itération k=1**

  1. $\sigma_1 = \varepsilon = \{\}$ (σ$_k$ est la valeur courante de l'UPG que l'on construit)

  2. σ_1 unifie-t-elle $p(x, f(x), y)$ et $p(y, z, u)$
     » **non** : $p(x, f(x), y)\ \sigma_1 \rightarrow p(x, f(x), y) \neq p(y, z, u) \leftarrow p(y, z, u)\ \sigma_1$
     » alors cherche ensemble de désaccord $D_1 = \{x, y\}$

  3. Existe-t-il un substitution qui unifie les éléments de D_1
     » **oui** : $\{x = y\}$ (on aurait aussi pu choisir $\{y = x\}$ à la place)
     » alors met à jour UPG : $\sigma_2 = \sigma_1 \{x = y\} = \{x = y\}$

Unificateur le plus général : exemple 1

- Trouver l'UPG de $p(x, f(x), y)$ et $p(y, z, u)$

- **Itération k=2**

  1. $\sigma_2 = \{x = y\}$

  2. σ_2 unifie-t-elle $p(x, f(x), y)$ et $p(y, z, u)$
     » **non** : $p(x, f(x), y)\ \sigma_2 \rightarrow p(y, f(y), y) \neq p(y, z, u) \leftarrow p(y, z, u)\ \sigma_2$
     » alors cherche ensemble de désaccord $D_2 = \{f(y), z\}$

  3. Existe-t-il un substitution qui unifie les éléments de D_2
     » **oui** : $\{z = f(y)\}$
     » alors met à jour UPG : $\sigma_3 = \sigma_2 \{z = f(y)\} = \{x = y, z = f(y)\}$

Unificateur le plus général : exemple 1

- Trouver l'UPG de $p(x, f(x), y)$ et $p(y, z, u)$

- **Itération k=3**
  1. $\sigma_3 = \{x = y, z = f(y)\}$
  2. σ_3 unifie-t-elle $p(x, f(x), y)$ et $p(y, z, u)$
     » **non** : $p(x, f(x), y)\ \sigma_3 \rightarrow p(y, f(y), y) \neq p(y, f(y), u) \leftarrow p(y, z, u)\ \sigma_3$
     » alors cherche ensemble de désaccord $D_3 = \{y, u\}$
  3. Existe-t-il un substitution qui unifie les éléments de D_3
     » **oui** : $\{y = u\}$ (on aurait aussi pu choisir $\{u = y\}$ à la place)
     » alors met à jour UPG : $\sigma_4 = \sigma_3 \{y = u\} = \{x = u, z = f(u), y = u\}$

Unificateur le plus général : exemple 1

- Trouver l'UPG de $p(x, f(x), y)$ et $p(y, z, u)$

- **Itération k=4**
    1. $\sigma_4 = \{x = u, z = f(u), y = u\}$
    2. σ_4 unifie-t-elle $p(x, f(x), y)$ et $p(y, z, u)$
        » **oui** : $p(x, f(x), y)\ \sigma_4 \rightarrow p(u, f(u), u) = p(u, f(u), u) \leftarrow p(y, z, u)\ \sigma_4$
        » alors on retourne l'UPG σ_4

Unificateur le plus général : exemple 2

- Trouver l'UPG de $p(f(y), x)$ et $p(x, y)$

- **Itération k=1**

  1. $\sigma_1 = \varepsilon = \{\}$ (σ_k est la valeur courante de l'UPG que l'on construit)

  2. σ_1 unifie-t-elle $p(f(y), x)$ et $p(x, y)$
     » **non** : $p(f(y), x)\ \sigma_1 \rightarrow p(f(y), x) \neq p(x, y) \leftarrow p(x, y)\ \sigma_1$
     » alors cherche ensemble de désaccord $D_1 = \{f(y), x\}$

  3. Existe-t-il un substitution qui unifie les éléments de D_1
     » **oui** : $\{x = f(y)\}$
     » alors met à jour UPG : $\sigma_2 = \sigma_1 \{x = f(y)\} = \{x = f(y)\}$

Unificateur le plus général : exemple 2

- Trouver l'UPG de $p(f(y), x)$ et $p(x, y)$

- **Itération k=2**

1. $\sigma_2 = \{x = f(y)\}$

2. σ_2 unifie-t-elle $p(f(y), x)$ et $p(x, y)$
   » **non** : $p(f(y), x)\ \sigma_2 \rightarrow p(f(y), f(y)) \neq p(f(y), y) \leftarrow p(x, y)\ \sigma_1$
   » alors cherche ensemble de désaccord $D_2 = \{f(y), y\}$

3. Existe-t-il un substitution qui unifie les éléments de D_1
   » **non** : $y = f(y)$ n'est pas valide puisque y apparaît à gauche et à droite
   » alors retourne faux (**n'a pas d'UPG puisque n'est pas unifiable**)

Exercice en classe

- Dire si un UPG existe pour les littéraux suivants, et si oui l'identifier
  - $p(x, f(x))$ et $p(x, y)$
  - $p(x, z)$ et $p(z, f(x))$
  - $p(f(y), z)$ et $p(z, f(x))$

Exercice en classe

- Dire si un UPG existe pour les littéraux suivants, et si oui l'identifier
  - $p(x, f(x))$ et $p(x, y)$ $\Rightarrow$ $\{ y = f(x) \}$
  - $p(x, z)$ et $p(z, f(x))$ $\Rightarrow$ n'existe pas
  - $p(f(y), z)$ et $p(z, f(x))$ $\Rightarrow$ $\{ z = f(x), y = x \}$

Mettre une formule sous forme normale conjonctive

1. **Élimination de l'implication**
   - Utiliser l'équivalence

     $$\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$$

     pour enlever toutes les implications de la formule
2. **Réduire la portée de ¬**
   - Utiliser les **lois de Morgan**, c-à-d.
     i. $\neg (f_1 \vee f_2) \equiv \neg f_1 \wedge \neg f_2$
     ii. $\neg (f_1 \wedge f_2) \equiv \neg f_1 \vee \neg f_2$
     iii. $\neg \neg f \equiv f$,

     de sorte que ¬ est toujours suivi d'un prédicat
3. **Standardiser les variables**
   - renommer les variables de telle sorte qu'aucune paire de quantificateurs ne porte sur la même variable

Mettre une formule sous forme normale conjonctive

4. **Éliminer les quantificateurs existentiels**
   - chaque quantificateur existentiel est éliminé, en **remplaçant sa variable par une fonction des quantificateurs universels englobants**
     - ex. : $\forall\, x\ \forall\, y\ \exists\, z\ p(x, y, z)$ est remplacé par $\forall\, x\ \forall\, y\ p(x, y, f(x,y))$
     - on appelle ces fonctions (ex. $f(x,y)$ ci-haut) des **fonctions de Skolem**
     - le symbole de la fonction doit être unique (ne pas utiliser f à chaque fois)
     - si aucun argument, on utilise une constante unique
       - ex. : $\exists\, x\ q(x)$ devient $q(a)$ (où a n'est pas une constante déjà définie)
5. **Mettre en forme prénexe**
   - mettre tous les quantificateurs universels en tête
6. **Distribuer les disjonctions dans les conjonctions**
   - mettre sous forme de conjonction ($\wedge$) de disjonctions ($\vee$) de littéraux, en utilisant les équivalences de distributivité :

$$f_1 \vee (f_2 \wedge f_3) \equiv (f_1 \vee f_2) \wedge (f_1 \vee f_3)$$

Mettre une formule sous forme normale conjonctive

7. **Éliminer les symboles de quantificateurs universels**
   - on ne laisse que les variables

8. **Éliminer les conjonctions (**$\wedge$**)**
   - on génère des clauses séparées (sur des lignes différentes)

9. **Standardiser les variables à part**
   - renommer les variables de telle sorte que deux clauses différentes n'aient pas les même variables

Preuve par résolution

- Pour prouver que f_1 implique f_2
  - transformer f_1 en un ensemble de clauses en forme normale conjonctive
  - y ajouter les clauses pour $\neg f_2$ (comme dans preuve par contradiction)
  - appliquer répétitivement la **règle de résolution** jusqu'à aboutir à la clause vide, notée $\square$ (on a prouvé que f_1 implique f_2)
  - s'il n'est plus possible d'appliquer la règle de résolution, f_1 n'implique pas f_2

- **Règle de résolution pour le cas propositionnel** (c.-à-d. prédicats sans arguments) :
  - étant données les **clauses parents** $(p_1 \vee \ldots \vee p_n)$ et $(\neg p_1 \vee q_1 \vee \ldots \vee q_m)$, on génère la **clause résolvante** $p_2 \vee \ldots p_n \vee q_1 \vee \ldots \vee q_m$
  - on retrouve la règle Modus Ponens, puisque $f_1 \rightarrow f_2$ est équivalent à $\neg f_1 \vee f_2$

Règle de résolution pour les prédicats

- **Règle de résolution pour le cas de prédicats généraux**
  - soit deux clauses parents $L = L_1 \vee \ldots \vee L_n$ et $M = M_1 \vee \ldots \vee M_m$:

  1. trouver un littéral L_k et un littéral M_l tel qu'il existe un UPG θ tel que $L_k\theta = \neg M_l\theta$
  2. la clause résolvante de $L_1 \vee \ldots \vee L_n$ et $M_1 \vee \ldots \vee M_m$ est

$$\underbrace{L_1\theta \vee \ldots \vee L_{k-1}\theta \vee L_{k+1}\theta \vee \ldots \vee L_n\theta}_{\boldsymbol{L\,\theta} \textbf{ sans } \boldsymbol{L_k\theta}} \vee \underbrace{M_1\theta \vee \ldots \vee M_{l-1}\theta \vee M_{l+1}\theta \vee \ldots \vee M_m\theta}_{\boldsymbol{M\,\theta} \textbf{ sans } \boldsymbol{M_l\theta}}$$

- Ex. :
  - **clauses parents:** $L = \neg dog(x) \vee animal(x)$, $M = \neg animal(y) \vee die(y)$
  - **clause résolvante:** $\neg dog(x) \vee die(x)$ (UPG = $\{ y = x \}$)
- Deux clauses parents peuvent avoir plusieurs résolvants selon le choix L_k et M_l

Règle de résolution pour les prédicats

- La règle de résolution telle que décrite jusqu'à maintenant est **correcte** (***sound***), mais elle n'est pas **complète**

- La règle de résolution combinée avec la **factorisation** est **complète**
  - si deux littéraux d'une même clause ont un UPG, on remplace ces littéraux par le résultat de leur unification
  - un **facteur** est le résultat de cette transformation
    - ex. : $q(z) \vee p(f(y))$ est un facteur de la clause $q(z) \vee p(x) \vee p(f(y))$ (obtenu par l'UPG $\{ x = f(y) \}$
  - on peut utiliser la factorisation au besoin, avant ou après l'application de la règle de résolution, afin de générer de nouvelles clauses

Répondre à des questions

- La résolution permet de savoir si oui ou non, f_2 est une conséquence logique de f_1
- 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, ... , x_n) à chaque clause de f_2, où les x_i sont les variables apparaissant dans la clause
  - on applique la preuve par résolution
  - on arrête lorsqu'on a une clause composée uniquement du littéral *Rep*

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

Formules

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. $\neg\ die(Fido)$

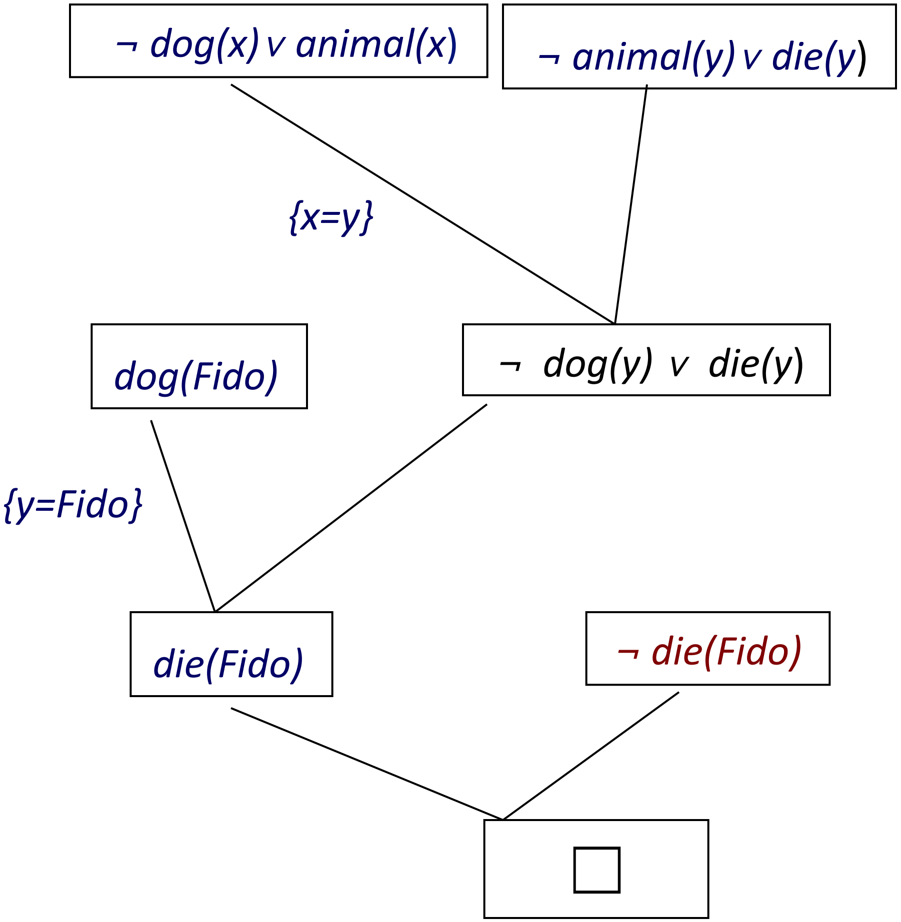
- **Format clausale**

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

---

5. $\neg\ dog(y) \lor die(y)$
   1, 2, $\{x=y\}$
6. $die(Fido)$
   3, 5 $\{y=Fido)$
7. $\square$
   4, 6

Exemple 1



- **Format clausale**
  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. □
   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 : $\boldsymbol{hait(Marcus,Cesar)}$

Etape 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. $assassiner(Marcus,Cesar)$

Etape 2 : réduire la porte de ¬

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*)

1. *personne*(*Marcus*)
2. *pompeien*(*Marcus*)
3. $\forall x \neg pompeien(\mathrm{x}) \vee romain(x)$
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*)

Etape 3 : standardiser les variables

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

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

Etape 4 : éliminer les quantificateurs existentiels

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

---

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

Etape 5 : mettre les formules en forme prénexe

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

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

Aucun changement dans ce cas-ci

Etape 6 : distribuer les disjonctions dans les conjonctions

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

Etape 7 : éliminer les quantificateurs universels

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. ¬ *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*)

Etape 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

Etape 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

Etape 10 : Ajouter les clauses de la négation de 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*)

Etape 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. ☐ (clause vide)
    1, 15

Example 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

Si on va plus loin : traiter l'égalité

- La logique du premier ordre inclue normalement la notion d'égalité (indiquée par le symbole « = ») entre les termes
- Une façon de gérer l'égalité est de la définir avec plusieurs formules qui décrivent le concept d'égalité (symétrie, transitivité, etc.)
  - on peut alors utiliser la preuve par résolution
- Pour d'autres façons de gérer l'égalité, voir la section 9.5.5 du livre

- Certains systèmes logiques utilisent **la supposition des noms uniques** (***unique-names assumption***) :
  - deux constantes ayant un symbole différent sont en fait des entités différentes
  - on ne désignera pas une même personne sous deux noms différents

Si on va plus loin : domaine ouvert vs. fermé

- La logique du premier ordre suppose aussi **un domaine ouvert**
  - il n'y a pas de limite connue sur l'ensemble des objets
- Dans ce cas, on ne peut pas remplacer la quantification universelle par une conjonction très longue
  - ex. : $\forall personne(x) \rightarrow mortel(x)$ veut dire que toutes les personnes **qui ont existées ou vont exister** sont mortelles

- Certains logiciels de raisonnement logique vont plutôt supposer un domaine fermé
  - on doit alors définir explicitement l'ensemble des objets de notre « monde »
  - une longue conjonction exhaustive et la quantification universelle sont alors équivalentes
- Un raisonnement similaire s'applique pour la quantification existentielle

Si on va plus loin : monde ouvert vs. fermé

- La logique de premier ordre suppose aussi **un monde ouvert**
  - si un $p(x)$ n'est pas dans la base de formules, ceci n'implique pas que $\neg p(x)$ est vraie
- Là encore, certains logiciels de raisonnement logique vont plutôt supposer le contraire, c.-à-d. un domaine fermé
- **Prolog** est un exemple de langage logique qui suppose
  - **noms uniques** (***unique-names assumption)***
  - **domaine fermé** (***closed-world assumption***)
  - **monde fermé** (***domain closure assumption***)
- Ces suppositions sont appelées **sémantiques des bases de données**

Si on va plus loin : satisfiabilité

- On a supposé que la base de connaissance initiale ne contenait pas de contradictions
  - ex. : elle ne contient pas $p(x)$ et $\neg$p(x)
  - si une conjonction de clauses (comme $p(x) \land \neg p(x)$) ne peut jamais être vraie, on dit qu'elle n'est pas **satisfaisable**
  - déterminer la satisfiabilité d'une conjonction de clauses est un problème NP-complet en général
    - » **3-SAT** : le problème de déterminer si une conjonction de clauses de 3 littéraux chacune est satisfaisable est le premier problème NP-complet ayant été découvert

Si on va plus loin : automatisation

- Une procédure automatique de résolution consisterait à grandir constamment la base de connaissance en appliquant la règle de la résolution sur chaque paire de clause possible
  - aussitôt qu'on génère la clause vide, on a réussit à déduire la clause requête
  - lorsqu'il n'est plus possible d'ajouter une nouvelle clause à l'aide de la règle de résolution, on sait qu'on ne peut déduire la clause requête

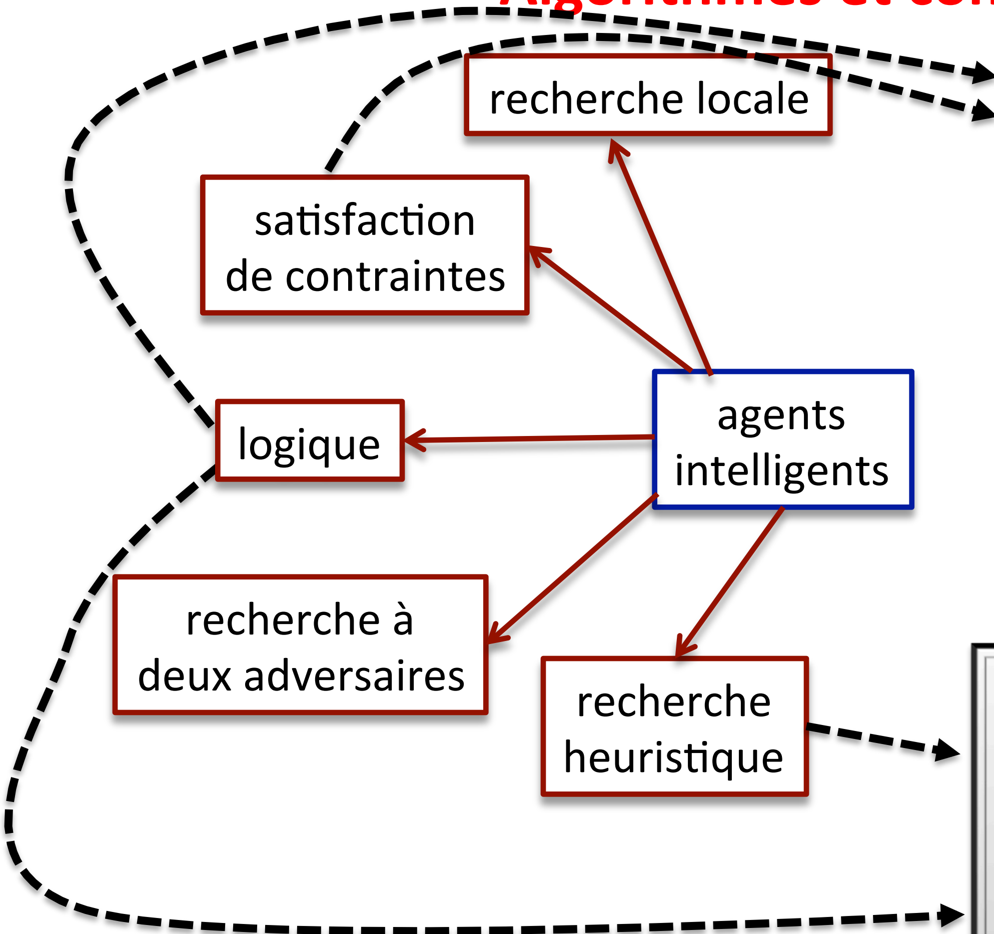
- Cette procédure pourrait être très lente
  - voir la section 9.5.6 pour des stratégies pour faire des preuves plus efficacement

Applications

- Vérification de logiciel informatique
  - la base de connaissance contient l'information sur l'effet de chaque instructions et leurs conditions pour être exécutées
- Synthèse de logiciel
  - répondre à la question « existe-t-il un programme *p* satisfaisant une certaine spécification »
- Systèmes experts
  - diagnostic médical, automobile : étant donné des « symptômes », quelle est la « maladie »
  - nécessite qu'un expert mette sous forme logique toutes ses connaissances
- Preuve de théorèmes mathématiques

Objectifs du cours

Algorithmes et concepts

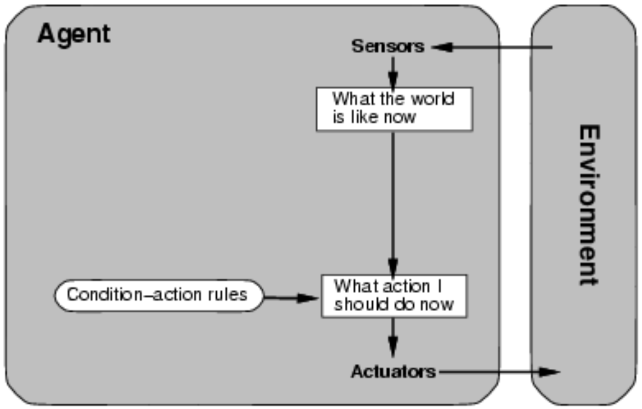


- ✓ comme en satisfaction de problème, la logique introduit une façon de décrire l'état de recherche pour une classe de problème spécifique

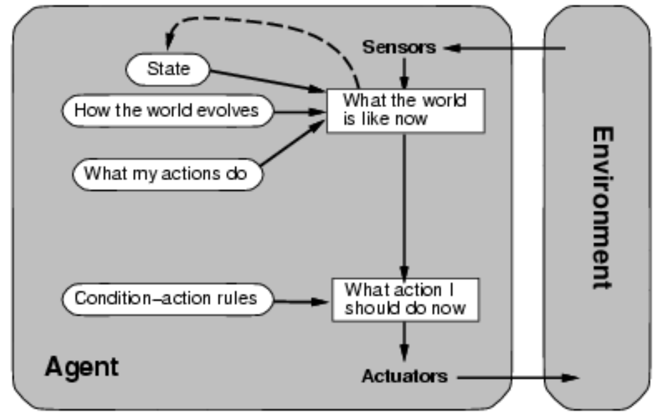
- ✓ les algorithmes basés sur la preuve par résolution font de la recherche heuristique
- ✓ un état correspond à l'ensemble des clauses dérivées
- ✓ le but est d'atteindre la clause vide

Logique du premier ordre : pour quel type d'agent?

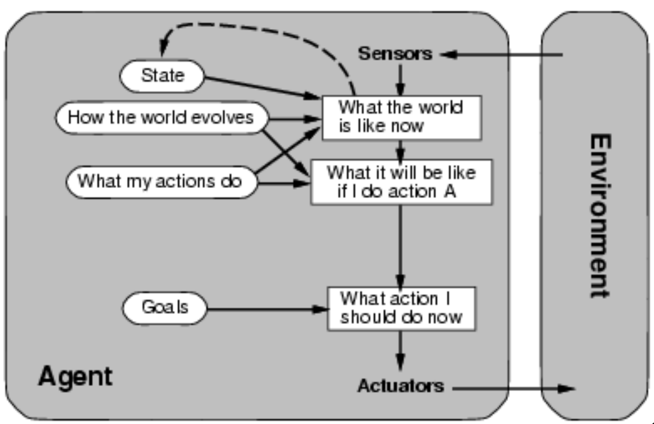
Simple reflex



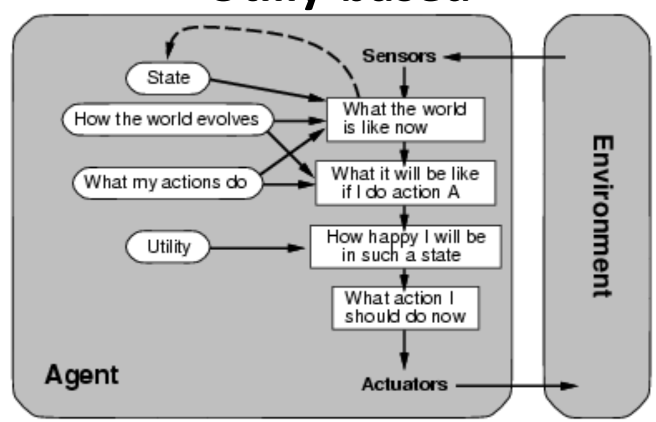
Model-based reflex



Goal-based



Utiliy-based



Conclusion

- La logique du premier ordre est un langage qui permet de modéliser le raisonnement logique
- Les applications sont variées, du diagnostique à la planification
- Bien que le problème d'inférence soit un problème NP-complet, certains problèmes peuvent être résolus en un temps raisonnable
- Dans une application donnée, une large partie du travail consiste à écrire la base de connaissance pour notre problème sous forme de logique du premier ordre

Vous devriez être capable de...

- Écrire des formules en logique de premier ordre
  - connaître la syntaxe
  - traduire une assertion en français sous forme de logique

- Faire une preuve par résolution
  - appliquer une substitution
  - identifier l'unificateur le plus général (UPG)
  - mettre sous forme normale conjonctive

Exercice en classe

- Faits (tirés de *Monty Python and the Holy Grail*) :
  1. Toute personne faite en bois est une sorcière.
  2. Tous les canards sont faits en bois.
  3. Toute chose qui pèse la même chose qu'un canard est faite en bois.
  4. La dame (A) pèse la même chose que le canard (D).
  5. Le Roi Arthur est une personne.
  6. Sir Bedevere est une personne.
  7. La dame (A) est une personne.
  8. D est un canard.
  9. Le canard (D) n'est pas une personne.

- Exercice : convertir sous forme logique de premier ordre

Exercice en classe (suite)

- Convertir la forme logique des faits extraits de *Monty Python and the Holy Grail* sous forme normale conjonctive

- Utiliser la preuve par résolution pour **prouver que la dame est une sorcière**

- Utilisez la preuve par résolution pour identifier **qui parmi la dame, Sir Bedevere et du canard est la sorcière**