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

Calcul des prédicats: 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

- Le **calcul des prédicats** (appelé aussi « **logique du premier ordre** ») 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

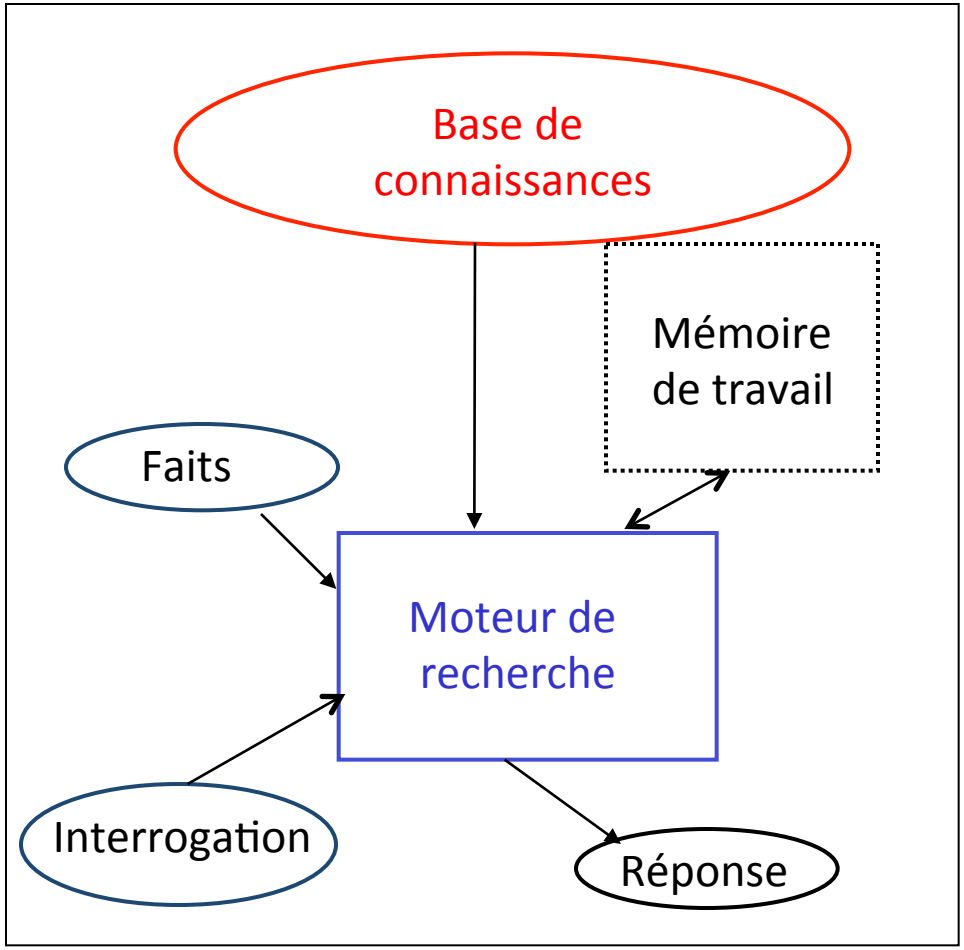
- Trouver un plan d'action permettant d'accomplir un but, à partir de:
  1. Actions possibles d'un agent (robot, personnage d'un jeu comme le *wumpus world*, etc.).
  2. But à accomplir.

- 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

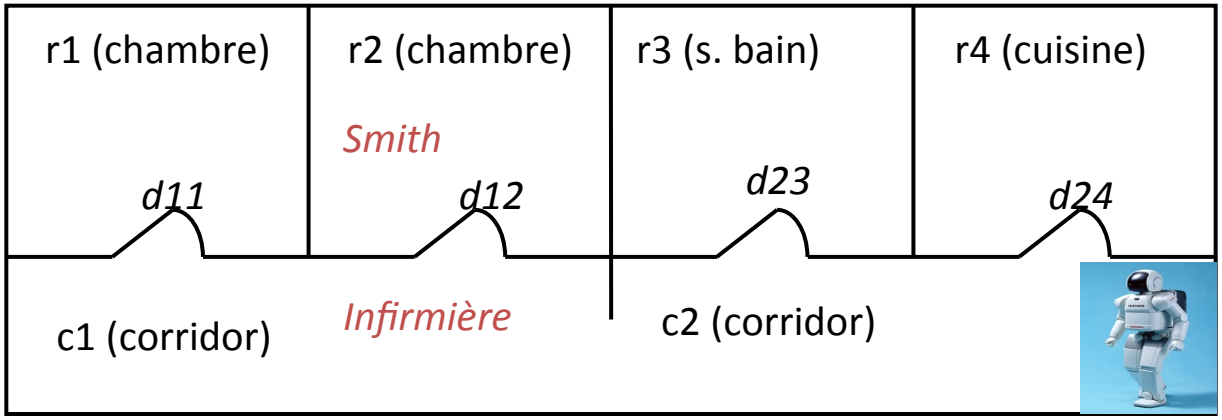
Exemple: Schéma d'un système expert à base de règles de production



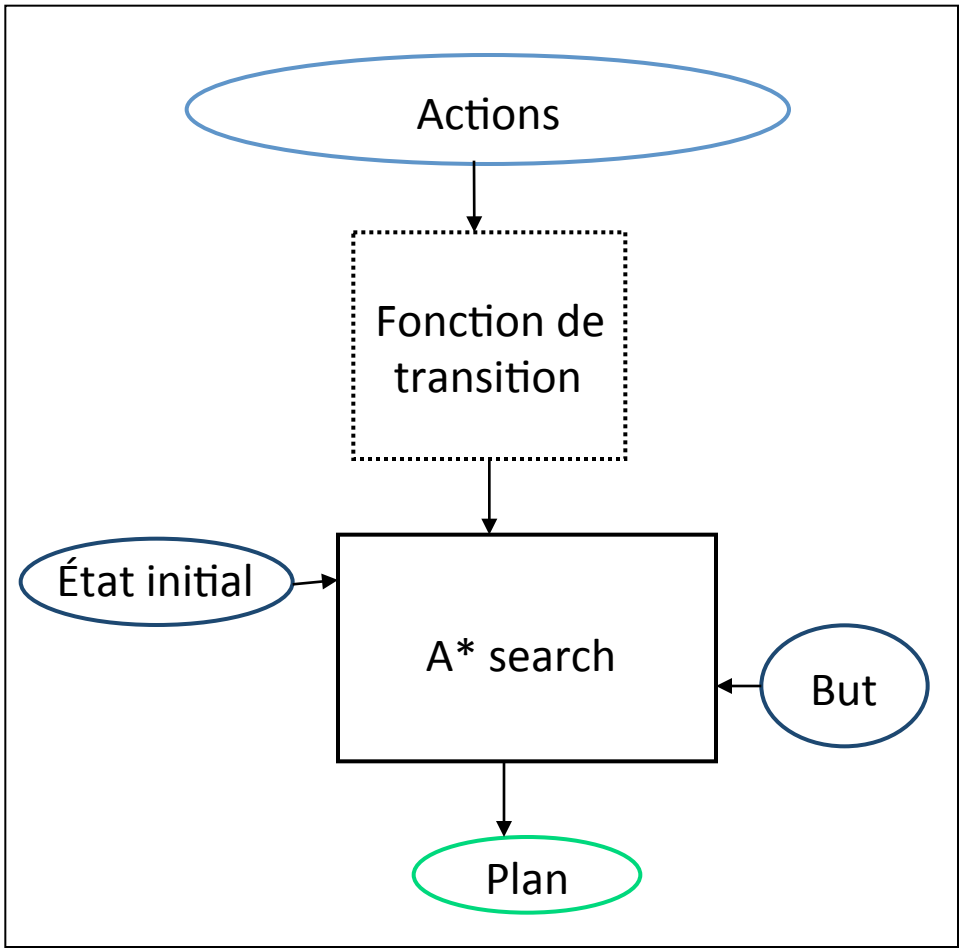
- La base de connaissances est spécifiée par des règles logiques
- Les faits sont des propositions logiques
- La mémoire de travail peut être vue comme un état
- Le moteur de recherche est une exploration de l'espace d'états
- Exemple:
  - Java Expert System Shell (JESS)

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

Le calcul des prédicats: un modèle mathématique du raisonnement déductif

- À l'origine, le calcul des prédicats 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

Trois niveaux pour évaluer une expression logique

- Interface: on évalue d'abord les **objets** impliqués dans l'expression
  - cela équivaut à la saisie des données

- On évalues ensuite les relations entre les objets apparaissant dans la relation. Ces relations sont aussi appelées des **faits** ou **propositions**. Elles sont représentées par des **prédicats**
  - cela équivaut au traitement des données

- Enfin, on évalue la façon dont les faits sont liés entre eux par des liens logiques: **et**, **ou**, **implique**, **pour tout**, **il existe**
  - cela revient à l'inférence logique

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 le calcul des prédicats

- Ensuite nous considérons la **sémantique**, c'est-à-dire comment on détermine si une expression est logiquement vraie étant donné ce que l'on a inféré jusqu'à maintenant

- 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** (*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**
  - 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 (ils sont à la base des formules)
- **Variables:** *x*, *y*, *z*
- **Connecteurs:** ¬ (non), ∧ (et), ∨ (ou), ⟶ (implique)
- **Quantificateurs:** ∀ (pour tout), ∃ (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 *t1*, …, *tn* sont des termes et *f* une fonction à n arguments, alors *f*(*t1*, …, *tn*) est aussi un terme.
  - par exemple: *pere*(*John*), *pere*(*x*), *pere*(*pere*(*x*))

Syntaxe des formules

- Un prédicat est une formule
  - plus précisément, si *t1*, …, *tn* sont des termes et *p* est un prédicat à n arguments, alors *p*(*t1*, …, *tn*) est une formule
  - c'est 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 $\neg \forall v \neg \alpha$

Notations

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

- Dans un programme par exemple, on utilisera 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 $\exists$.

Commentaire

- Les **termes dénotent des objets** alors que les **prédicats dénotent 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 a choisis
- C'est l'ensemble des valeurs couvertes par les variables dans les quantificateurs
- Par exemple, supposons que l'univers du discours soit 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*, *Jean-François*, *Frod*, etc.
- On aurait aussi des fonctions pour décrire de nouveaux objets en fonction d'autres objets
  - *age*(*Eric*), *taille*(*Jean-François*), *position*(*Robot*), etc.
- Nous aurions aussi des prédicats pour décrire des relations entre ces objets
  - *ACoteDe*(*Tableau*,*Frod*), *Regarde*(*Eric*,*Frod*), etc.

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
    - » $Aime(Jean, coursIA) \wedge \neg Aime(Jean, TP(coursIA))$
  - 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
    - » $\forall x\ \text{Inscrit}(x) \rightarrow \exists y\ Aime(x, y)$
- Les quantificateurs ne font pas partie des connecteurs
- Le calcul des prédicats est un langage
- Les formules précises qu'on va écrire dépendront de l'application

Commentaire

- On pourrait vouloir 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:
  - $Mortel(Personne1) \wedge \ldots Mortel(Personnen)$
- On doit alors on supposer qu'on a un **domaine fermé**
- Avec les quantificateurs, l'expression devient plus simple et s'applique aussi aux domaines ouverts
  - $\forall x \; Personne(x) \rightarrow Mortel(x)$

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. $assassiner(Marcus,Cesar)$

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

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 des 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

Programmer l'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 (définis) 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*(*t1*, …, *tn*) est vrai si *p*(*t1*, …, *tn*) 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

Sémantique des bases de données

- Dans la procédure qui suit, on va faire les suppositions suivantes concernant l'interprétation sémantique de la logique du premier ordre
  - **supposition de noms uniques** (***unique-names assumption***):
    deux constantes ayant un symbole différent sont en fait des entités différentes
    - » ex.: on ne désignera pas une même personne sous deux noms différents
  - **supposition d'un monde fermé** (***closed-world assumption***):
    un prédicat est considéré comme faux s'il n'est pas défini explicitement comme vrai ou ne peut pas être inféré comme vrai
  - **supposition de domaine fermé** (***domain closure assumption***):
    il n'y a pas d'autres entités que celles désignées par les constantes
- Ces suppositions font partie de ce qu'on appelle la **sémantique des bases de données** (***database semantics***)

Programmer l'interprétation des formules

- Pour évaluer une formule arbitraire, nous utilisons la fonction EVALFORMULA en considérant les prédicats comme le cas de base
- Étant donnée une formule arbitraire *f*, EVALFORMULA(*f*) fonctionne comme suit:
  - si *f* est un prédicat: retourne EVALPREDICATE(*f*)
  - si *f* est de la forme ¬ *f1*: si EVALFORMULA(*f1*) retourne « vrai » alors retourne « faux » sinon retourne « vrai »;
  - si *f* est de la forme *f1* ∧ *f2*: si EVALFORMULA(*f1*) retourne « vrai » et EVALFORMULA(*f2*) retourne « vrai », alors retourne « vrai »; sinon retourne « faux »
  - si *f* est de la forme ∀ *x f1*: si EVALFORMULA(*f2*) retourne « vrai » pour chaque constante *c* et chaque formule *f2* obtenue de *f1* en remplaçant chaque occurrence libre de *x* par *c*, alors retourne « vrai »; sinon retourne « faux »;
- Les cas ∨, → et ∃ sont traités par des macros.

Validité, satisfaisabilité 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 est **valide**
  - exemple: $P(a) \vee \neg P(a)$
- Si une formule est vraie dans quelques interprétations, on dit qu'elle est **satisfaisable** (par les interprétations qui la rendent vraie)
  - exemple: $\exists x\ Inscrit(x) \wedge Aime(x, coursIA)$
- Si une formule est fausse dans toutes les interprétations possibles (c'est-à-dire, quelque soit la valeur de EVALPREDICATE()), on dit qu'elle **n'est pas satisfaisable**
  - exemple: $P(a) \wedge \neg P(a)$
- Une formule *f2* est une **conséquence logique** d'une formule *f1* si toute interprétation satisfaisant *f1* satisfait aussi *f2*

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, instantiation universelle

- **Modus Ponens**
  - à partir de *f1* et *f1* $\rightarrow$ *f2*, on déduit *f2*
    - » si on a (*WumpusAhead* $\wedge$ *WumpusAlive*) $\rightarrow$ *Shoot* et a (*WumpusAhead* $\wedge$ *WumpusAlive*), alors *Shoot* peut être inféré
- **Instantiation universelle**
  - à partir de $\forall$ *x f1* on déduit *f2* obtenu de *f1* en remplaçant toutes les occurrences libres de *x* par un terme n'ayant pas de variable en commun avec *f1*
    - » 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
- 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

- Un **littéral** est un prédicat ou la négation d'un prédicat
  - ex.: $p_1(x, y)$, $\neg\, p_1(x, y)$
- Une **clause** est un ensemble de litéraux.
  - ex.: $p_1(x, y) \vee p_2(x, y, z) \vee \neg\, p_1(x, z)$
  - on va utiliser la notation $\{p_1(x, y), p_2(x, y, z), \neg\, p_1(x, z)\}$ pour les clauses
- 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 litéral α donne un litéral $\alpha\theta$ 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: $\alpha = p(x, y, f(a))$, $\theta = \{x = b, y = x\}$
    $\alpha\theta = p(b, x, f(a))$

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

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 $\theta\sigma$ 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 les identités, c-à-d., les paires pour lesquelles $s_i\sigma$ 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

- 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 littéraux
- Une substitution θ est appélé **unificateur** de S (de α_1 et α_2) si $\alpha_1\theta = \alpha_2\theta$
- Un unificateur θ de S est appelé **unificateur le plus général** (**UPG**) de S si pour toute unificateur σ de S, il existe une substitution γ telle que $\sigma = \theta\gamma$
- Par exemple:
  - { $p(f(x),a)$, $p(y,f(w))$ }, où a est une constante, n'est pas unifiable
  - { $p(f(x),z)$, $p(y,a)$ } a pour UPG $\sigma = \{y = f(x), z = a\}$.
- On appelle **ensemble de désaccord** entre deux littéraux la paire des premiers termes pour lesquels les deux littéraux diffèrent

Unification

Algorithme UNIFICATEUR(S)

1. k=0; $\sigma_0 = \varepsilon$
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.

Exercice en classe

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

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 de Skolem**, dont les arguments sont les variables quantifiées par des quantificateurs universels contenant le quantificateur qu'on veut éliminer dans leur portée
     - » ex.: $\forall x \forall y \exists z\ p(x, y, z)$ est remplacé par $\forall x \forall y \exists z\ p(x, y, f(x,y))$, et $f(x,y)$ est appelée un fonction de Skolem
5. **Mettre en forme prénexe**
   - mettre tous les quantificateurs universels en tête (la partie sans quantificateur est appelée une **matrice**)
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 quantificateurs universels**
   - on ne laisse que les variables

8. **Éliminer les conjonctions ($\wedge$)**
   - on obtient ainsi un ensemble de clauses séparées

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

Exercice en classe

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

Preuve par résolution

- Pour prouver que f_1 implique f_2
  - transformer f_1 en un ensemble de clauses
  - y ajouter les clauses pour $\neg f_2$
  - 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: 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 $f1 \rightarrow f2$ et équivalent à $\neg f1 \vee f2$.

Règle de résolution pour les prédicats

- Dénotons deux clauses parents par les ensembles de littéraux $\bigcup_i \{L_i\}$ et $\bigcup_j \{M_j\}$:
  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 $\bigcup_i \{L_i\}$ et $\bigcup_j \{M_j\}$ est
     $$\bigcup_i \{L_i\theta\} \setminus \{L_k\theta\} \quad \bigcup_j \{M_j\theta\} \setminus \{M_l\theta\}$$

- 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 est **correcte** (***sound***)
- Mais elle n'est pas **complète**
- La règle de résolution combinée avec la **factorisation** est **complète**
  - Si deux ou plus littéraux d'une clause *C* ont un UPG θ, alors la clause Cθ de laquelle on enlève les littéraux dupliqués est appelé **facteur** de C
  - La **factorisation** consiste à générer un facteur à partir d'une clause
- La résolution telle qu'on la voit dans le cours suppose un **domaine et un monde ouvert**

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 redondants
  - ex.: 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 énoncées précédemment pour qu'elle soit complète

Preuve par résolution avec factorisation

- Pour prouver que f_1 implique f_2
  - transformer f_1 en un ensemble de clauses
  - appliquer la **factorisation à chaque clause**
  - y ajouter les clauses **factorisées** pour $\neg f_2$
  - appliquer répétitivement la **règle de résolution** **suivi de la factorisation de la clause résolvante** jusqu'à aboutir à la clause vide, notée $\square$
  - s'il n'est plus possible d'appliquer la règle de résolution, f_1 n'implique pas f_2

Répondre à des questions

- La résolution permet de savoir si oui ou non, f_1 est une conséquence logique de f_2
- On peut aussi exploiter les traces du processus de preuve pour trouver des valeurs (instantiations) 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 résultant 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

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

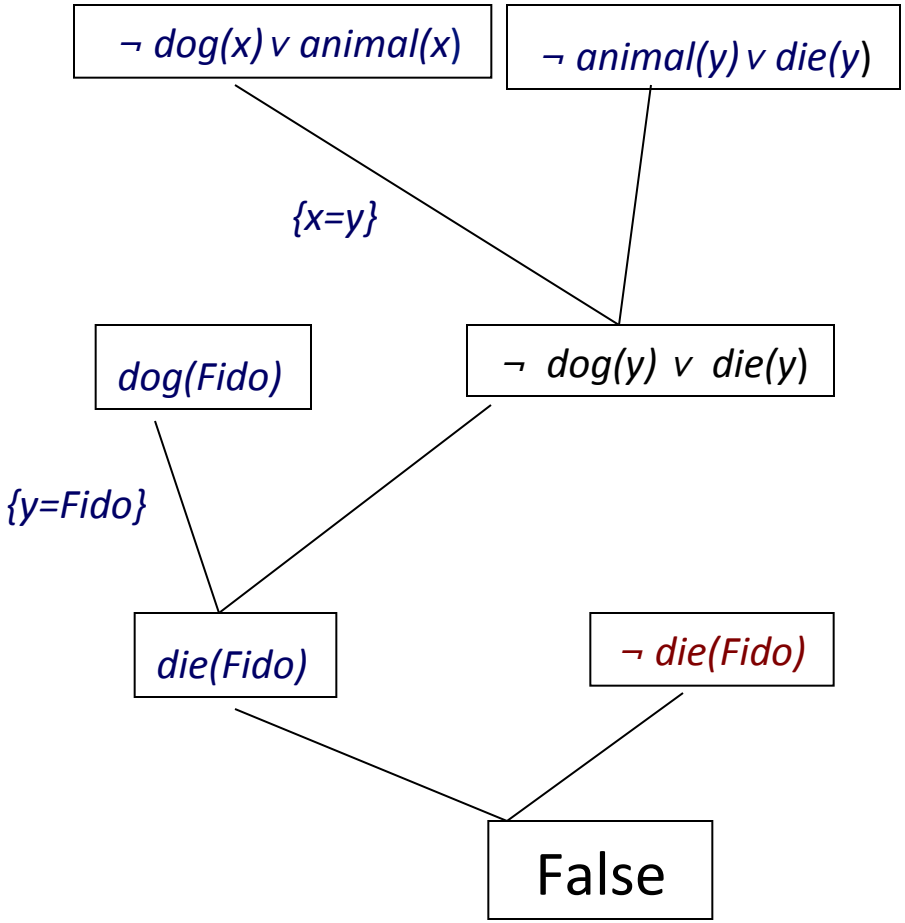
4. $\neg\ die(Fido)$

- **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



- **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. 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: $hait(Marcus,Cesar)$

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. ∀ *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*)

Etape 3: standardiser les variables

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. ∀ *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*)

Etape 4: éliminer les quantificateurs existentiels

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. $assassiner(Marcus,Cesar)$

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. $assassiner(Marcus,Cesar)$

Etape 5: mettre les formules en forme prénexe

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

Etape 6: distribuer les disjonctions dans les conjonctions

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

Etape 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. $assassiner(Marcus,Cesar)$

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. False
    1, 15 (clause vide)

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

Exercice en classe

- Dans l'exemple tiré de *Monty Python and the Holy Grail*, 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**

Égalité

- La logique du premier ordre inclue normalement la notion d'égalité (indiquée par le symbole « = ») entre les termes
- Dans ce cours, on a plutôt utilisé **la supposition des noms uniques** (***unique-names assumption***):
  - deux constantes ayant un symbole différent sont en fait des entités différentes
- 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 utilisé la preuve par résolution
- Pour d'autres façon de gérer l'égalité, voir la section 9.5.5 du livre

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