Neuvième partie IX

Inférence en logique du premier ordre

Plan

En bref ...

Réduction de la logique du premier ordre à la logique propositionnelle

Unification

Modus Ponens généralisé

Chaînage avant

Chaînage arrière

Résolution

Un bref rappel sur l'histoire des raisonnements

450 av. JC	Stoïcisme	Logique propositionnelle, inférence (peut être)
320 av .JC	Aristote	Syllogisme (règles d'inférence), quantifieurs
1565	Cardano	Théory des probabilité (logique propositionnelle + incertitude)
1847	Boole	Logique propositionnelle (encore)
1879	Fredge	Logique du premier ordre
1922	Wittgenstein	Preuve avec les tables de vérité
1930	Gödel	$\exists$ un algoritme complet pour la logique du premier ordre
1931	Herbrand	Algorithme complet pour la logique du premier ordre (réduit à la logique propositionnelle)
1960	Davis/Putnam	Algorithme utilisable pour la logique propositionnelle
1965	Robinson	Algorithme utilisable pour résolution de la logique du premier ordre

Réduction à la logique propositionnelle

Rappel : Substitution

- Définition : Une substitution est un ensemble de couples $\{(v_1/t_1), \ldots, (v_n/t_n))$
  Étant donné une formule S et une substitution σ, $S\sigma$ est le resultat de l'application de σ à S
- Exemple
  $S = PlusIntelligent(x, y)$ et $\sigma = \{x/Sophie, y/Paul\}$
  $S\sigma = PlusIntelligent(Sophie, Paul)$
- Remarques
  - Un terme qui ne contient plus de variable est un terme fermé
  - Un formule atomique qui ne contient plus de variable est appelé une proposition ou formule complètement instanciée

Instanciation universelle

- Instanciation universelle (UI) : Chaque instanciation d'un énoncé universellement quantifié peut être inféré :

$$\frac{\forall v, \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

  pour toute variable v et pour tout terme fermé g
- Exemple
  $\forall x\ Roi(x) \wedge Cupide(x) \Rightarrow Mechant(x)$
  $Roi(Jean) \wedge Cupide(Jean) \Rightarrow Mechant(Jean)$
  $Roi(Richard) \wedge Cupide(Richard) \Rightarrow Mechant(Richard)$
  $Roi(Pere(Jean)) \wedge Cupide(Pere(Jean)) \Rightarrow Mechant(Pere(Jean))$

Instanciation existentielle

- Instanciation existentielle (EI) : Pour toute énoncé α, pour toute variable v et pour tout symbole de contante k qui n'apparaît pas dans la base de connaissances, on a :

$$\frac{\exists v, \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

- Exemple
  $\exists x\ Couronne(x) \wedge SurTete(x, Jean)$
  $Couronne(C_1) \wedge SurTete(C_1, Jean)$
  C_1 est un nouveau symbole de constante, appelé constante de Skolem

Instanciation universelle et existentielle

- L'instanciation universelle peut être appliquée plusieurs fois pour ajouter de nouvelles formules dans la base de connaissances
  - La nouvelle base de connaissances est logicallement équivalente à l'ancienne
- L'instanciation existentielle peut être appliquée une fois pour remplacer les formules existentielle
  - La nouvelle base de connaissances n'est est pas logicallement équivalente à l'ancienne mais reste satisfiable ssi l'ancienne base de connaissances l'était

Exemple

- Supposons que nous ayons la base de connaissances suivante :

  $\forall x \ Roi(x) \wedge Cupide(x) \Rightarrow Mechant(x)$
  $Roi(Jean)$
  $Cupide(Jean)$
  $Frere(Richard, Jean)$

- Instanciation universelle : toutes les substitutions possibles :

  $Roi(Jean) \wedge Cupide(Jean) \Rightarrow Mechant(Jean)$
  $Roi(Richard) \wedge Cupide(Richard) \Rightarrow Mechant(Richard)$
  $Roi(Jean)$
  $Cupide(Jean)$
  $Frere(Richard, Jean)$

- La nouvelle base de connaissances est en logique propositionnelle
  - $Roi(John)$, $Cupide(John)$, $Mechant(John)$, $Roi(Richard)$, etc.

Propriétés et résultats

- Toute base de connaissances en logique du permier ordre peut être reduit en logique propositionnelle de manière à préserver la relation de conséquence
  - un énoncé est déduit de la nouvelle base de connaissances ssi il peut être déduit de la base de connaissances originale
- Idée pour la résolution : réduire une base de connaissances ainsi que la requête en logique propositionnelle, appliquer la résolution puis retourner un résultat
- Problème : Avec les symboles de fonction, l'ensemble des substitutions possibles des termes fermés est infini

  $Pere(Pere(Pere(Jean)))$

Propriétés et résultats

Théorème de Herbrandt

Si un énoncé est conséquence de la base de connaissances exprimée en logique du premier ordre, alors il existe une preuve qui ne fait appel qu'à un sous ensemble fini de la base de connaissances réduite en logique propositionnelle

- Idée
  - instancier d'abord avec toutes les constantes, e.g., $Richard$, $Jean$
  - puis les termes de profondeur 1, e.g., $(Pere(Richard), Pere(Jean))$
  - puis les termes de profondeur 2 ...
- Problème : fonctionne si l'énoncé est conséquence, mais boucle si l'énoncé n'est pas conséquence

Propriétés et résultats

Théorème de Turing et Church

En logique du premier ordre, la question de la conséquence logique est semi-décidable

- Il existe des algorithmes qui disent "oui" à tout énoncé conséquence, mais il n'en existe pas qui disent "non" à tout énoncé non-conséquence

Unification

Problèmes de réduction en logique propositionnelle

- La réduction en logique propositionnelle génére beaucoup d'énoncés inutiles
  - Exemple :

    $\forall x\ Roi(x) \wedge Cupide(x) \Rightarrow Mechant(x)$
    $Roi(Jean)$
    $\forall y\ Cupide(y)$
    $Frere(Richard, Jean)$

  On déduit $Mechant(Jean)$, mais également beaucoup d'énoncés comme $Cupide(Richard)$ qui ne sont pas pertinents
- Avec p prédicats $k-$aires et n constantes, il y a $p \cdot n^k$ instanciations
- Avex les symbols de fonctions, c'est encore bien pire !

Unification

- On peut obtenir l'inférence immédiatement si l'on peut trouver une substitution σ telle que $Roi(x)$ et $Cupide(x)$ correspondent à $Roi(Jean)$ et $Cupide(y)$

  $\sigma = \{x/John, y/John\}$
- $\textsc{Unify}(\alpha, \beta) = \sigma$ if $\alpha\sigma = \beta\sigma$

p	q	σ
$Connait(jean, x)$	$Connait(Jean, Jeanne)$	$\{x/Jeanne\}$
$Connait(jean, x)$	$Connait(y, Bill)$	$\{x/Bill, y/Jean\}$
$Connait(jean, x)$	$Connait(y, Mere(y))$	$\{y/Jean, x/Mere(Jean)\}$
$Connait(jean, x)$	$Connait(x, Bille)$	Echec

- Remarque :
  - Avant d'unifier il est nécessaire de renommer les variables pour éviter les interférence de nom
  - Cette étape s'appelle la normalisation

Unification

- Il peut y avoir plusieurs subtitutions ou unificateurs :
  - Exemple :
    Le résultat de l'unification de $Connait(Jean, x)$ et $Connait(y, z)$ peut être :
    $\sigma = \{y/Jean, x/z\}$
    $\sigma = \{y/Jean, x/Jean, z/Jean\}$
    Le premier unificateur est plus général que le second
- Il existe un seul unificateur plus général (MGU pour *Most General Unifier*) au renommage des variables près
  MGU $= \sigma = \{y/Jean, x/z\}$

Algorithme d'unification de Robinson (1/2)

Algorithme

funtion UNIFY(*x, y, sigma*) **return** *a subtitution to make x and y identical or failure*
  inputs : *x, a terme, i.e, a variable, a constant, a function, or a list*
    y, a terme, i.e, a variable, a constant, a function or a list
    σ, a subtutution initially empty $\{\}$
  if σ = *failure* **then return** *failure*
  if $x = y$ **then return** σ
  if VARIABLE ?(*x*) **then return** UNIFY-VAR(*x,y,σ*)
  if VARIABLE ?(*y*) **then return** UNIFY-VAR(*y,x,σ*)
  if FUNCTION ?(*x*) *and* FUNCTION ?(*x*) **then**
    if FUNCTOR(*x*) $\neq$ FUNCTOR(*y*) **then return** *failure*
    return UNIFY(AGRS[*x*],AGRS[*y*],*σ*)
  if LIST ?(*x*) *and* LIST ?(*x*) **then**
    if LENGTH(*x*) $\neq$ LENGTH(*y*) **then return** *failure*
    return UNIFY(REST[*x*], REST[*y*], FIRST[*x*], FIRST[*y*], *sigma*))
  return *failure*

Algorithme d'unification de Robinson (2/2)

Algorithme

funtion UNIFY-VAR(*var, x, sigma*) **return** *a subtitution*
  inputs : *var, a variable*
    x, a terme, i.e, a variable, a constant, a function or a list
    σ, a subtutution
  if $\{var/val\} \in sigma$ **then return** UNIFY(*val,x,σ*)
  if $\{x/val\} \in sigma$ **then return** UNIFY(*var,val,σ*)
  if OCCUR-CHECK ?(*var,x*) **then return** *failure*
  else return *add* $\{var/x\}$ *to* σ

Modus Ponens généralisé

Modus Ponens généralisé

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\sigma} \quad \text{ou} \quad p_i'\sigma = p_i\sigma \text{ for all } i$$

- Exemple :
  - p_1' est $Roi(Jean)$, p_1 est $Roi(x)$
  - p_2' est $Cupide(y)$, p_2 est $Cupide(x)$
  - q est $Mechant(x)$
  - σ est $\{x/Jean, y/Jean\}$
  - σq est $Mechant(Jean)$
- Remarques :
  - Le Modus Ponens généralisé est utilisé sur des bases de connaissances composées de clauses définies (exactement un litéral positif)
    - E.g, $\neg p \vee \neg q \vee \ldots \vee \neg t \vee u$ plus souvent écrit $p \wedge q \wedge \ldots \Rightarrow u$
  - Toutes les variables sont supposées universellement quantifiées

Chaînage avant

Exemple

Exemple : La vente d'arme aux États Unis

La loi stipule que c'est un crime pour un américain de vendre des armes à des nations hostiles. Le pays Nono, un ennemi de l'Amérique, a des missiles, et tous ses missiles lui ont été vendus par le colonel West, qui est américain

$\Rightarrow$ Prouvons que West est un criminel

Exemple

... c'est un crime pour un américain de vendre des armes à des nations hostiles :

$$\forall x, y, z \ Americain(x) \wedge Arme(y) \wedge Vend(x, y, z) \wedge Hostile(z) \Rightarrow Criminel(x)$$

Nono ... a des missiles, i.e., $\exists x \ Possede(Nono, x) \wedge Missile(x)$:

$$Possede(Nono, M_1) \text{ and } Missile(M_1)$$

... tous les missiles lui ont été vendus par le Colonel West

$$\forall x \ Missile(x) \wedge Possede(Nono, x) \Rightarrow Vend(West, x, Nono)$$

Les missiles sont des armes :

$$\forall x \ Missile(x) \Rightarrow Arme(x)$$

Un ennemi de l'Amérique est considérée comme hostile :

$$\forall x \ Ennemi(x, Amerique) \Rightarrow Hostile(x)$$

West, qui est un américain ...

$$Americain(West)$$

Le pays Nono est un ennemi de l'Amérique

$$Ennemi(Nono, Amerique)$$

Algorithme en chaînage avant

Algorithme

funtion FOL-FC-ASK ?(KB, α) **return** *a subtitution or false*

  input: KB, *a knowledge base, a entence in propositional logic*

   α, *the query, a sentence in propositional logic*

  repeat

   new $\leftarrow \{\}$

   foreach *sentence r in KB* **do**

    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARIZE(r)

    foreach σ *such that* $(p_1 \wedge \ldots \wedge p_n)\sigma = (p'_1 \wedge \ldots \wedge p'_n)\sigma$ *for some* $p'_1, \ldots, p'_n$ *in KB* **do**

     $q' \leftarrow$ SUBST(σ,q)

     if q' *os not a remaining of the sentence already in KB or new* **then**

      add q' *to new*

      $\phi \leftarrow$ UNIFY(q',α)

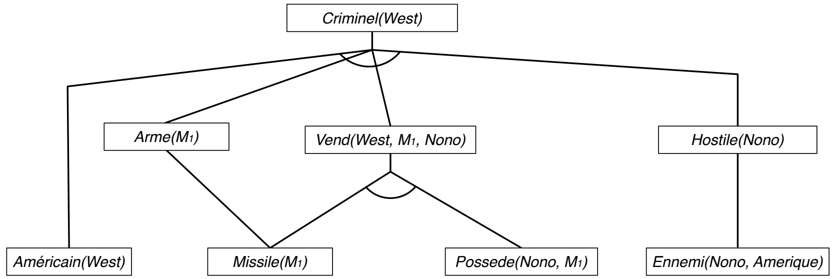
      if ϕ *is not fail* **then return** ϕ

   add new to KB

  until *new is empty*

  return *false*

Exemple



Propriétés du chaînage avant

- Valide et complet pour les bases de connaissances de clauses définies
- Le chaînage avant termine en un nombre fini d'itérations dans le cas des bases de connaissances Datalog
  - Datalog : base de connaissances de clauses définies sans symboles de fonctions
- Peut ne pas terminer dans le cadre général si α n'est pas conséquence (Cf. Théorème de Turing)
- Inévitable : la conséquence logique pour des clauses définies est semi-décidable

Chaînage arrière

Algorithme en chaînage arrière

Algorithme

funtion FOL-BC-ASK ?(*KB*, *goals*, α) **return** *substitution or false*

input : *KB, a knowledge base*

goals, a list of conjuncts forming a query (σ already applied)

σ, the current substitution, initially the empty subsitution $\{\}$

local variables : *answers, a set of substitutions, initially empty*

if *goal is empty* **then return** $\{\sigma\}$

$q' \leftarrow$ SUBST(σ, FIRST(*goals*))

foreach *sentence r in KB where*

STANDARDIZE(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$

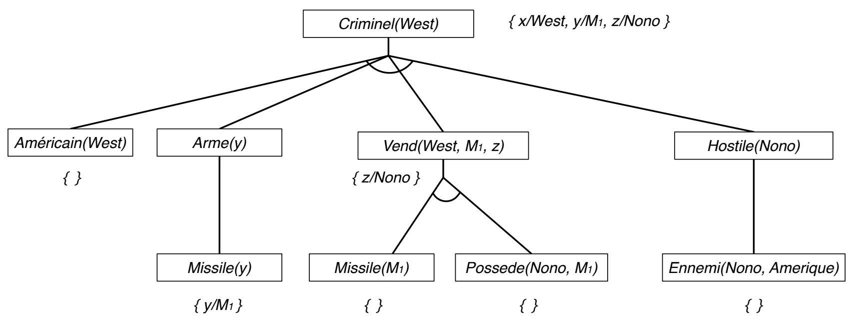
and $\sigma' \leftarrow$ UNIFY(q,q') *succeeds* **do**

newGoals $\leftarrow [p_1, \ldots, p_n|$ REST(*goals*)]

answers $\leftarrow$ FOL-BC-ASK(*KB*, *newGoals*, COMPOSE(σ',σ)) $\cup$*answers*

return *answers*

Exemple



Propriétés du chaînage arrière

- Chaînage arrière en profondeur d'abord : la complexité spatiale est linéaire en la taille de la preuve
- Incomplet : boucles infinies
  - Pour résoudre le problème il faut comparer le but actuel avec tous les buts empilés
- Inefficace à cause des sous-buts redondants
  - Pour résdoudre le problème il faut mettre en cache les résultats précédents
- Utilisé pour la programmation logique

Résolution

Résolution

- Règle de résolution pour la logique du premier ordre :

$$\frac{l_1 \vee \ldots l_k, m_1 \vee \ldots m_n}{(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)\sigma}$$

  ou $\text{UNIFY}(li, \neg m_i) = \sigma$.
- Par exemple :

$$\frac{\neg Rich(x) \vee Unhappy(x), Rich(Ken)}{Unhappy(Ken)}$$

  avec $\sigma = \{x/Ken\}$
- Remarques :
  - Les clauses doivent être normalisée, i.e., ne partager aucun nom de variable
  - La résolution appliquée sur une base de connaissances sous la forme CNF est complère pour la logique du premier ordre

Conversion en CNF

Soit l'énoncé suivant :

"Toute personne qui aime tous les animaux est aimée par quelqu'un"

qui se traduit en logique du premier ordre de la manière suivante :

$$\forall x\ (\forall y\ Animal(y) \Rightarrow Aimer(x, y)) \Rightarrow (\exists y\ Aimer(y, x))$$

La converstion de cet énoncé en forme normale conjonctive (CNF) s'effectue en 6 étapes :

1. Élimination des implications
2. Déplacement des négations vers l'intérieur des formules
3. Normalisation des variables
4. Skolémisation
5. Suppression des quatifieurs universels
6. Distribution de $\vee$ sur $\wedge$

Conversion en CNF

$$\forall x\ (\forall y\ Animal(y) \Rightarrow Aimer(x, y)) \Rightarrow (\exists y\ Aimer(y, x))$$

1. Élimination des implications

$$\forall x\ \neg(\forall y\ \neg Animal(y) \vee Aimer(x, y)) \vee (\exists y\ Aimer(y, x))$$

2. Déplacement des négations vers l'intérieur des formules

$$\forall x\ (\exists y\ \neg(\neg Animal(y) \vee Aimer(x, y))) \vee (\exists y Aimer(y, x))$$
$$\forall x\ (\exists y\ Animal(y) \wedge \neg Aimer(x, y)) \wedge (\exists y\ Aimer(y, x))$$

Rappel
- $\neg \forall x, p \equiv \exists x\ \neg p$
- $\neg \exists x, p \equiv \forall x\ \neg p$

Conversion en CNF

$$\forall x\ (\exists y\ Animal(y) \wedge \neg Aimer(x, y)) \wedge (\exists y\ Aimer(y, x))$$

3. Normalisation des variables : chaque quantifieur doit utiliser une variable différente

$$\forall x\ (\exists y\ Animal(y) \wedge \neg Aimer(x, y)) \wedge (\exists z\ Aimer(z, x))$$

4. Skolémisation : suppression des quantificateurs existentiels par élimination. Chaque variable existentielle est remplacée par une fonction de Skolem dont les arguments sont les variables universelles dans la portée du quantifieur universel

$$\forall x\ (Animal(F(x)) \wedge \neg Aimer(x, F(x))) \wedge Aimer(G(x), x)$$

Conversion en CNF

$$\forall x\ (Animal(F(x)) \wedge \neg Aimer(x, F(x))) \wedge Aimer(G(x), x)$$

5. Suppression des quantifieurs universels

$$(Animal(F(x)) \wedge \neg Aimer(x, F(x))) \wedge Aimer(G(x), x)$$

6. Distribution de $\vee$ sur $\wedge$

$$(Animal(F(x)) \vee Aimer(G(x), x)) \wedge (\neg Aimer(x, F(x)) \vee Aimer(G(x), x))$$