

Bases de la logique des prédicats du premier ordre (deuxième partie)

Ce document fait suite à « bases de la logique du premier ordre ».

D'une part, il introduit des termes complexes : les fonctions.

D'autre part, il présente une méthode de preuve : la méthode de résolution.

1. Les fonctions

Dans cette section, nous introduisons les symboles fonctionnels, au niveau syntaxique et au niveau sémantique. Les changements par rapport aux définitions de la première partie du cours sont en rouge (voir par exemple la version en ligne sur l'espace pédagogique).

1.1 Syntaxe

Définition (langage du premier ordre)

Un langage du premier ordre $L = (C, P, F)$ est formé :

- d'un ensemble C de **constantes** (ou **fonctions 0-aires**) ;
- d'un ensemble P de symboles de relation (ou **prédicats**) ;

chaque prédicat possède une arité, qui est un entier ≥ 0 ;

- et d'un ensemble F de symboles de **fonctions** (non 0-aires) ;

chaque symbole de fonction possède une arité, qui est un entier ≥ 1 .

Un prédicat d'arité 0 correspond à un symbole propositionnel (de la logique des propositions).

Une constante peut être vue comme un symbole fonctionnel d'arité 0. Dans la suite, lorsqu'on parle de symbole de fonction ou de fonction sans autre précision, on ne considère pas les constantes.

Définition (terme)

Soit L un langage du premier ordre, où C est l'ensemble de ses constantes et F est l'ensemble de ses symboles fonctionnels d'arité non nulle. L'ensemble de **termes** construits sur L peut être défini inductivement de la manière suivante :

(base) les **variables** et les **constantes de C** sont des termes ;

(règle de construction) si f est un **symbole de fonction** de F , et $t_1 \dots t_n$ ($n > 0$) sont des termes construits sur L , alors $f(t_1 \dots t_n)$ est un terme.

Les définitions de fbf, arborescence syntaxique d'une fbf, formule fermée, etc. ne changent pas (seule la définition des termes est étendue).

1.2 Sémantique

Définition (interprétation)

Soit $L = (\mathcal{C}, \mathcal{P}, \mathcal{F})$ un langage du premier ordre. Une *interprétation* I de L est constituée d'un ensemble D **non vide** appelé **domaine** et d'une définition du sens des symboles de L sous forme d'applications dans D ou de relations sur D :

- pour tout a de \mathcal{C} , $I(a)$ est un **élément de D** ;
- pour tout p de \mathcal{P} , $I(p) \subseteq D^{\text{arité}(p)}$
si p est d'arité 0, $I(p)$ est égal à vrai ou faux
- pour tout f de \mathcal{F} , $I(f)$ est une **application de $D^{\text{arité}(f)}$ dans D** .

Un symbole de constante s'interprète donc comme un élément du domaine d'interprétation, **un symbole de fonction comme une application de D à la puissance l'arité de la fonction, dans D** , et un prédicat comme une relation sur D (une "table" sur D) ayant pour arité celle du symbole. Remarque : puisqu'un symbole de fonction d'arité k s'interprète comme une application, *tout* k -uplet de D^k a une image par cette application.

Définition (valeur de vérité d'une fbf pour une interprétation I et une assignation s)

Si I est une interprétation et s est une assignation, l'application $v(A, I, s)$ dans $\{\text{vrai}, \text{faux}\}$ est définie de la manière suivante :

- si A est un *atome* $p(t_1, t_2, \dots, t_n)$, $v(p(t_1, t_2, \dots, t_n), I, s) = \text{vrai}$ si :

il existe $(d_1 \dots d_n) \in I(p)$ avec, pour tout i , $d_i = Is(t_i)$

où Is associe un élément de D à tout terme :

si t_i est une constante, $Is(t_i) = I(t_i)$ (*on prend l'interprétation de la constante*)

si t_i est une variable, $Is(t_i) = s(t_i)$ (*on prend l'assignation de la variable*)

si $t_i = f(t_1, \dots, t_k)$ alors $Is(t_i) = I(f)(Is(t_1), \dots, Is(t_k))$

$[v(\perp, I, s) = \text{faux pour tout } I \text{ et pour tout } s]$

- si $A = \neg B$ alors $v(A, I, s) = \text{NON}(v(B, I, s))$, la négation étant interprétée comme en logique des propositions
- si $A = (B \wedge C)$ alors $v(A, I, s) = \text{ET}(v(B, I, s), v(C, I, s))$, la conjonction étant interprétée comme en logique des propositions,
... *de même pour les autres connecteurs*
- si $A = \forall x B$ alors $v(A, I, s) = \text{vrai}$ ssi pour *tout* élément d de D , $v(B, I, s+[x \leftarrow d]) = \text{vrai}$, où $s+[x \leftarrow d]$ est l'assignation obtenue à partir de s en donnant à la variable x la valeur d ;
[si $A = \exists x B$ alors $v(A, I, s) = \text{vrai}$ ssi pour (*au moins*) un élément d de D , $v(B, I, s+[x \leftarrow d]) = \text{vrai}$].

2. Une méthode de preuve : la méthode de résolution

Jusqu'à présent, nous avons étudié la **syntaxe** et la **sémantique** de la logique du premier ordre. Pour prouver qu'un raisonnement est correct, nous avons utilisé la notion de *conséquence logique*, qui repose la notion d'interprétation, donc sur la signification des formules. Nous allons maintenant nous intéresser à des méthodes *purement syntaxiques*, que l'on appelle **méthodes de preuve**, ou **systèmes déductifs** : ces méthodes sont basées sur des règles syntaxiques qui permettent de dériver des formules à partir d'autres formules. On dit que les nouvelles formules sont **déduites** des premières. Ces méthodes ont été développées dans le but de modéliser le raisonnement mathématique (les formules de départ sont des *axiomes*, les formules dérivées sont des *théorèmes*, et la suite d'applications de règles permettant de dériver une formule à partir de formules de départ est appelé une *preuve*). Bien sûr, un système déductif n'est intéressant que si la notion de déduction correspond *exactement* à la conséquence logique : une formule C se *déduit* d'un ensemble de formules H_1, \dots, H_n , *si et seulement si* C est *conséquence logique* de H_1, \dots, H_n . On peut donc parler indifféremment de déduction ou de conséquence logique. Parmi les méthodes de preuve les plus importantes en logique du premier ordre, citons : les systèmes de Gentzen ou systèmes de déduction naturelle et la méthode de résolution (due à Robinson). La méthode de résolution permet le développement de démonstrateurs (ou "prouveurs") efficaces. Elle est à la base des langages de programmation logique (comme Prolog).

Dans cette section, nous étudierons en détail la **méthode de résolution**. Le but de cette méthode est de montrer qu'un ensemble de formules est insatisfiable (elle correspond donc à un **raisonnement par l'absurde** : pour prouver que C est conséquence de H_1, \dots, H_n , supposons que l'on ait H_1, \dots, H_n et $\neg C$ et montrons que l'on arrive à une contradiction (la clause vide)). Cette méthode s'applique à n'importe quelle formule de la logique du premier ordre, après que cette formule ait été mise sous une forme spéciale appelée *forme clausale*.

2.1. Forme prénexe

Définition (forme prénexe)

Une fbf A est sous forme *prénexe* lorsqu'elle a la forme suivante :

$A = Q_1x_1Q_2x_2\dots Q_nx_n B$, où les Q_i sont des quantificateurs et B est une fbf sans quantificateur.

Théorème

Toute fbf est équivalente à une fbf sous forme prénexe.

Démonstration

On se limite ici à des fbf fermées (mais le principe est le même pour des fbf quelconques).

Voici un **schéma d'algorithme** pour transformer une fbf fermée en une fbf équivalente sous forme prénexe :

- (1) Renommer les variables de façon à ce qu'aucune variable (x_i) n'apparaisse liée dans plusieurs quantificateurs (Qx_i)
- (2) Eliminer les connecteurs \rightarrow et \leftrightarrow en utilisant les équivalences $(A \leftrightarrow B) \equiv (A \rightarrow B) \wedge (B \rightarrow A)$ puis $(A \rightarrow B) \equiv (\neg A \vee B)$; les seuls connecteurs binaires restants sont \wedge et \vee
- (3) Faire "rentrer" les négations : déplacer les négations juste devant les symboles de prédicats en utilisant les équivalences ci-dessous, et supprimer au passage les doubles négations :

$\neg\neg A \equiv A,$
 $\neg(A \vee B) \equiv \neg A \wedge \neg B,$
 $\neg(A \wedge B) \equiv \neg A \vee \neg B,$
 $\neg\forall x A \equiv \exists x \neg A$
 $\neg\exists x A \equiv \forall x \neg A$

(4) Faire "remonter" les quantificateurs (les déplacer vers la gauche), c'est-à-dire répéter : choisir un quantificateur (qui n'est pas dans la portée d'un quantificateur non encore remonté) et le faire remonter, de façon à ce qu'il porte sur tous les littéraux, en utilisant les équivalences suivantes (où Q désigne \exists ou \forall , et A et B sont des sous-formules) :

A op QxB \equiv Qx (A op B), si x n'apparaît pas dans A

QxA op B \equiv Qx (A op B), si x n'apparaît pas dans A.

Comme on a renommé les variables de façon à ce qu'une même variable ne soit pas liée deux fois, ces équivalences sont toujours applicables.

Remarque : on peut parfois faire remonter plus ou moins un quantificateur parmi les autres quantificateurs. L'étape (4) peut donc donner des résultats différents. Il est toujours possible de faire remonter les quantificateurs en respectant l'ordre gauche-droite. Par la suite (skolémisation), on s'apercevra qu'il est avantageux de faire remonter les quantificateurs existentiels le plus possible (tout en respectant les règles ci-dessus !).

Exemples :

- la formule $\forall x(p(x) \wedge \exists y p(y))$ peut se mettre sous deux formes prénexes en utilisant les équivalences données en (4):

$\forall x(p(x) \wedge \exists y p(y)) \equiv \forall x(\exists y (p(x) \wedge p(y))) \equiv \forall x\exists y (p(x) \wedge p(y))$

$\forall x(p(x) \wedge \exists y p(y)) \equiv \forall x p(x) \wedge \exists y p(y) \equiv \exists y\forall x (p(x) \wedge p(y))$

(idée : les sous-formules $p(x)$ et $p(y)$ sont indépendantes, donc peu importe l'ordre entre les quantificateurs x et y ; essayez de bien repérer les équivalences utilisées à chaque étape)

- par contre, la formule $\forall x(p(x) \wedge \exists y q(x,y))$ a une seule forme prénexe :

$\forall x(p(x) \wedge \exists y q(x,y)) \equiv \forall x(\exists y (p(x) \wedge q(x,y))) \equiv \forall x\exists y (p(x) \wedge q(x,y))$

(idée : les sous-formules contenant x et y ne sont pas indépendantes, $\exists y$ ne peut donc pas « dépasser » $\forall x$).

D'autres exemples ont été vus en cours.

2.2. Forme clausale et théorème de Skolem

Définition (littéral, clause, forme clausale)

Un *littéral* est un atome $p(\dots)$ ou la négation d'un atome $\neg p(\dots)$. Une *clause* est une disjonction de littéraux, quantifiée universellement : $\forall x_1 \forall x_2 \dots \forall x_p (L_1 \vee L_2 \vee \dots)$, où chaque L_i est un littéral. Une fbf est sous *forme clausale* si elle est sous la forme d'une *conjonction de clauses* ; de façon équivalente, on peut aussi dire qu'une forme clausale est la fermeture universelle d'une conjonction de disjonctions de littéraux : $\forall x_1 \forall x_2 \dots \forall x_n ((L_{11} \vee L_{12} \vee \dots) \wedge (L_{21} \wedge L_{22} \vee \dots) \wedge \dots \wedge (L_{k1} \vee L_{k2} \vee \dots))$, où chaque L_{ij} est un littéral (en effet, on peut toujours renommer les variables des clauses de façon à pouvoir déplacer tous les quantificateurs universels en tête de la formule).

Pour les besoins algorithmiques, on voit souvent une clause comme un *ensemble* de littéraux (elle ne contient donc pas deux fois le même littéral), et une forme clausale comme un *ensemble* de clauses.

Comment mettre une fbf sous forme clausale ? On peut d'abord la mettre sous forme prénexe $A = Q_1 x_1 Q_2 x_2 \dots Q_n x_n B$. Comme B n'a pas de quantificateur, on peut ensuite mettre B sous la

forme d'une conjonction de disjonctions de littéraux. Il reste à faire **disparaître** les **quantificateurs existentiels**.

Si A est une fbf quelconque, on ne peut généralement pas trouver une forme clausale qui soit équivalente à A , sauf si A est insatisfiable. Mais on peut trouver une forme clausale qui est insatisfiable si et seulement si A est insatisfiable. C'est le cas de la forme clausale appelée **forme de Skolem** :

Voici un **schéma d'algorithme**, permettant de calculer une forme clausale associée à une fbf (fermée) A , que l'on appelle forme de Skolem :

Donnée : une fbf (fermée) A

Résultat : une forme de Skolem de A

(1-2-3-4) Mettre A sous forme prénexe $Q_1x_1Q_2x_2...Q_nx_n B$

(5) Distribuer les \wedge et \vee de façon à transformer B en une conjonction de disjonction de littéraux : on obtient B' .

(6) Faire disparaître les quantificateurs existentiels en introduisant des fonctions de Skolem : supprimer successivement tous les quantificateurs existentiels de la gauche vers la droite de la manière suivante : si x_i est la première variable quantifiée existentiellement, on considère un *nouveau symbole de fonction* f n'appartenant pas au langage, d'arité $i-1$ (donc une *constante* si $i = 1$), on fait la substitution $(x_i, f(x_1, \dots, x_{i-1}))$ dans B' , et on supprime x_i . Remarque : comme on supprime les quantificateurs existentiels de la gauche vers la droite, quand on traite $\exists x_i$, les quantificateurs qui portent sur x_1, \dots, x_{i-1} sont tous des quantificateurs *universels*.

Exemple :

$B = \exists x \forall y \exists z \forall t \exists u (r(g(x),y) \wedge s(x,y,z) \wedge s(a,t,u))$ où a est une constante

à pour forme de Skolem :

$\forall y \forall t (r(g(b),y) \wedge s(b,y,f_1(y)) \wedge s(a,t,f_2(y,t)))$

où b (constante), f_1 et f_2 sont les nouveaux symboles fonctionnels

Théorème de Skolem

Si S est une forme de Skolem d'une fbf A alors A est (in-)satisfiable ssi S est (in-)satisfiable. Plus précisément tout modèle de S est un modèle de A et on peut étendre tout modèle de A en un modèle de S .

Démonstration.

On montre d'abord le lemme suivant :

Lemme

Soient $A = \forall x_1 \forall x_2 \dots \forall x_{r-1} \exists x_r Q_{r+1}x_{r+1} \dots Q_n x_n B[x_1, \dots, x_n]$ une fbf fermée sous forme prénexe et

$A' = \forall x_1 \forall x_2 \dots \forall x_{r-1} Q_{r+1}x_{r+1} \dots Q_n x_n B[x_1, \dots, x_{r-1}, f(x_1, \dots, x_{r-1}), x_{r+1}, \dots, x_n]$, la fbf obtenue à partir de A par skolémisation de x_r , en introduisant le nouveau symbole fonctionnel $(r-1)$ -aire f , alors A est satisfiable ssi A' est satisfiable. Plus précisément tout modèle de A' est un modèle de A , et tout modèle de A peut être étendu en un modèle de A' .

Démonstration du lemme

Soit L le langage de A . Soit I une interprétation de L , de domaine D , qui est un modèle de A , donc : pour tout d_1 de D , pour tout d_2 de D ... pour tout d_{r-1} de D il existe un élément d_r de D tel que :

$v(C, I, s+[x_1 \leftarrow d_1, \dots, x_{r-1} \leftarrow d_{r-1}, x_r \leftarrow d_r]) = 1$, où $C = Q_{r+1}x_{r+1} \dots Q_n x_n B[x_1, x_2, \dots, x_n]$.

Considérons l'interprétation I' de $L + \{f\}$, de domaine D , obtenue en interprétant les symboles de L

comme dans I , et en définissant $I'(f)$ de la manière suivante : pour tout d_1 de D , pour tout d_2 de D ... pour tout d_{r-1} de D $I'(f)(d_1, d_2, \dots, d_{r-1}) = d_r$. On peut vérifier que I' est un modèle de A' . Réciproquement, si I' est un modèle de A' la restriction de I' à L est un modèle de A , il suffit de prendre pour valeur de la variable x_r , $d_r = I'(f)(d_1, d_2, \dots, d_{r-1})$. **cqfd**

On peut ensuite faire une démonstration du théorème par récurrence sur le nombre de quantificateurs existentiels :

- S'il n'y a pas de quantificateur existentiel alors une fbf sous forme prénexe est une forme clausale qui est donc équivalente à A .
- Supposons que la propriété soit vraie pour toute formule avec $k \geq 0$ quantificateurs existentiels et soit $Q_1 x_1 Q_2 x_2 \dots Q_n x_n B[x_1, x_2, \dots, x_n]$ avec $k + 1$ quantificateurs existentiels, le premier étant au rang r . Soit A' obtenue par skolémisation de x_r , en introduisant le nouveau symbole fonctionnel $(r-1)$ -aire f , on peut utiliser le lemme, et A' ayant k quantificateurs existentiels, on peut appliquer l'hypothèse de récurrence. **cqfd**

Quelques conséquences du théorème de Skolem

Soit S une forme de Skolem d'une fbf A .

(1) Si S est valide alors A est valide

(2) La réciproque est fausse comme le montre l'exemple suivant :

soit $A = (\exists x p(x) \rightarrow \exists y p(y))$;

A est valide, et en la skolémisant (et en faisant remonter le quantificateur sur y le plus possible), on obtient une formule non valide, équivalente à $(\exists x p(x) \rightarrow p(a))$.

Exercice : détailler ce passage de A à une forme clausale équivalente à $(\exists x p(x) \rightarrow p(a))$.

(3) $(A \wedge G)$ est insatisfiable ssi $(S \wedge G)$ est insatisfiable, où G est une fbf quelconque (et G ne contenant aucun des symboles fonctionnels introduits dans S , sinon ce n'est pas nécessairement vrai, par exemple : $A = \exists x p(x)$, $S = p(a)$, et $G = \neg p(a)$).

(4) Si, pour tout $i = 1, \dots, n$, S_i est une forme de Skolem de A_i , alors :

$A_1, A_2, \dots, A_n \models C$ ssi $S_1, S_2, \dots, S_n \models C$

En effet, $A_1 \wedge A_2 \wedge \dots \wedge A_n \wedge \neg C$ est insatisfiable ssi $S_1 \wedge S_2 \wedge \dots \wedge S_n \wedge \neg C$ est insatisfiable.

2.3. Unification de termes

Définition (substitution)

Une *substitution* est un ensemble de couples $s = \{(x_1, t_1), (x_2, t_2), \dots, (x_k, t_k)\}$ où les x_i sont des **variables** et les t_i des termes (variables, constantes, fonctions d'arité non nulle avec leurs arguments), avec :

pour tout i et tout $j \neq i$, on a $x_i \neq x_j$ (« on ne remplace pas 2 fois une même variable »)

Appliquer $s = \{(x_1, t_1), (x_2, t_2), \dots, (x_k, t_k)\}$ à une formule A consiste à remplacer toute variable x_i apparaissant dans A par t_i . Le résultat est noté $s(A)$. On peut de la même façon appliquer s à un terme, ou à une liste de termes. Attention, lors de l'application d'une substitution, on remplace **simultanément** (autrement dit "en parallèle") chaque variable. *Exemple :*

Si $A = p(x) \wedge p(y)$ et $s = \{(x, y), (y, f(a))\}$,

alors $s(A) = p(y) \wedge p(f(a))$ et pas $p(f(a)) \wedge p(f(a))$

Définition (unificateur)

Des listes de termes l_1, \dots, l_n sont *unifiables* s'il existe une substitution s qui les rend identiques : $s(l_1) = \dots = s(l_n)$. s est appelé un *unificateur* de ces listes de termes.

On étend cette définition à des atomes : des atomes sont unifiables s'ils ont même prédicat et si leurs listes de termes sont unifiables.

Notons E l'ensemble des éléments à unifier (ce sont des atomes, ou des listes de termes, ...). Il existe en général plusieurs unificateurs de E : on cherchera un unificateur qui "transforme" le moins possible les éléments de E : on l'appelle un *unificateur le plus général*.

Définition (upg)

Un unificateur u d'un ensemble E est un *unificateur le plus général (upg)* si tout autre unificateur u' de E s'obtient par une substitution supplémentaire s : $u' = s(u(E))$, ou encore $u' = s \circ u$.

E peut avoir *plusieurs* upg : dans ce cas, ils s'obtiennent les uns des autres avec un *renommage de variables* (c'est-à-dire : dans la définition ci-dessus, s ne comporte que des couples de variables, et deux variables différentes sont remplacées par des variables différentes).

Définition (composition de substitutions)

Soient les substitutions $s_1 = \{(x_1, t_1), (x_2, t_2), \dots, (x_k, t_k)\}$ et $s_2 = \{(y_1, t'_1), (y_2, t'_2), \dots, (y_k, t'_k)\}$. La composition de s_2 et s_1 est (on applique s_1 puis s_2) :

$s_2 \circ s_1 = \{(x_i, s_2(t_i)) \mid x_i \neq s_2(t_i)\} \cup \{(y_j, t'_j) \mid \text{il n'existe pas } x_i \text{ avec } x_i = y_j\}$

Remarque : la définition ci-dessus est une définition générale ; dans le contexte de l'algorithme de Robinson, on a juste à calculer $s_2 \circ s_1 = \{(x_i, s_2(t_i)) \mid \text{il n'existe pas } y_j \text{ avec } y_j = x_i\}$, sans aucune vérification.

L'algorithme de Robinson permet, étant donné un ensemble de termes (ou listes de termes, atomes, ...) E , de déterminer si E est unifiable, et le cas échéant de construire un upg de E .

On appelle **ensemble de désaccord** D d'un ensemble de termes $E = \{t_1, t_2, \dots, t_k\}$ l'ensemble de termes obtenu de la manière suivante :

voyons les t_i comme des mots ; on calcule p le plus grand préfixe commun à tous les t_i (plus précisément : pour tout i , t_i s'écrit $p.t'_i$ et p est maximal avec cette propriété) ; l'ensemble de désaccord D est l'ensemble des **sous-termes** commençant à la première lettre de chaque t'_i .

Remarque : en cours, nous avons considéré l'algorithme pour *deux* termes, on voit ici sa généralisation à un ensemble de k termes, généralisation qui est évidente.

Schéma de l'algorithme de Robinson

Donnée : $E = \{t_1, t_2, \dots, t_k\}$

Résultat : un upg de E , ou échec si les éléments de E ne sont pas unifiables

Début

$u \leftarrow \emptyset$ // initialisation de u qui sera, s'il existe, l'upg de E

Calculer D , l'ensemble de désaccord de E

tantque D n'est pas vide faire

 s'il existe dans D une **variable** x et un terme t avec $x \notin \text{variables}(t)$ alors

$s \leftarrow \{(x, t)\}$

$u \leftarrow s \circ u$ // composition des deux substitutions

$E \leftarrow s(E)$

 sinon retourner *échec*

fintantque

retourner u (qui est l'upg de E)

Fin

2.4. La méthode de résolution

Notation : Etant donné un littéral l , on note l^c son complémentaire (c'est-à-dire le littéral négatif obtenu en ajoutant une négation devant l si l est positif, et le littéral positif obtenu en supprimant la négation de l si l est négatif). Etant donné un ensemble L de littéraux, L^c est l'ensemble obtenu à partir de L en remplaçant tous les littéraux par leur complémentaire.

Définition (règle de résolution générale)

Soient C_1 et C_2 deux clauses sans variable en commun, et soient $L_1 \subseteq C_1$ et $L_2 \subseteq C_2$ tels que L_1 et L_2^c sont unifiables par un upg u . La **résolvante** de C_1 et C_2 selon L_1 , L_2 et u est :

$$\text{res}(C_1, C_2, L_1, L_2, u) = (u(C_1) - u(L_1)) \cup (u(C_2) - u(L_2)).$$

Remarques :

- " L_1 et L_2^c sont unifiables" entraîne que L_1 ne contient que des littéraux positifs et L_2 que des littéraux négatifs, ou inversement.
- on considère que C_1 et C_2 n'ont **pas de variable en commun** : sinon, la méthode de résolution n'est pas complète (on peut "rater" des déductions).
- on parle de règle de résolution **binaire** si L_1 et L_2 ne contiennent chacun qu'un seul littéral. La règle de résolution binaire n'assure pas la complétude de la méthode de résolution, comme le montre l'exemple suivant : soit la forme clausale $= C_1 \wedge C_2$, où $C_1 = p(x) \vee p(y)$ et $C_2 = \neg p(u) \vee \neg p(v)$; S est insatisfiable (se souvenir que les variables sont quantifiées **universellement**), mais on ne peut pas dériver la clause vide en utilisant seulement la résolution binaire. Il faut utiliser la règle de résolution générale pour unifier d'un seul coup les deux littéraux positifs de C_1 et les deux littéraux négatifs de C_2 . Attention, les littéraux de L_1 sont tous positifs et ceux de L_2 tous négatifs, ou inversement.

Propriété

- (1) Si u est une substitution quelconque et C une clause alors $C \models u(C)$
- (2) Si C est une résolvante de C_1 et C_2 alors $C_1, C_2 \models C$

La méthode de résolution (due à Robinson) permet de construire une preuve de la correction d'un raisonnement de la forme $H_1, H_2, \dots, H_n \models C$ de la manière suivante :

- on sait que $H_1, H_2, \dots, H_n \models C$ ssi $H_1 \wedge H_2 \wedge \dots \wedge H_n \wedge \neg C$ est insatisfiable.
- le théorème de Skolem dit qu'une fbf est insatisfiable ssi une forme clausale de cette fbf est insatisfiable (par exemple, la forme de Skolem).
- enfin, le théorème de Robinson (ci-dessous) dit que la méthode de résolution est complète :

Théorème de Robinson

Une forme clausale S est **insatisfiable** ssi on peut obtenir la **clause vide** en utilisant un **nombre fini** de fois la règle de résolution à partir des clauses de S .

Schéma d'algorithme (méthode de résolution)

Donnée : S une forme clausale

Résultat : retourne vrai si S est insatisfiable ; si S est satisfiable, retourne faux **ou bien ne s'arrête pas**

Début

EnsClauses $\leftarrow S$

tantque vrai faire

si EnsClauses contient deux clauses C_1 et C_2 (avec éventuel renommage de variables si C_1 et C_2 ont des variables en commun), avec $L_1 \subseteq C_1$, $L_2 \subseteq C_2$ et L_1 et L_2^c sont unifiables par un upg u , et telles que $\text{Res}(C_1, C_2, L_1, L_2, u) \notin \text{EnsClauses}$ (à un renommage des variables près)

alors

ajouter C à EnsClauses

si C est la clause vide

alors retourner vrai

sinon retourner faux // on a produit toutes les clauses possibles sans trouver la clause vide

fin tant que

Fin

Dans le cas où S est insatisfiable, si on considère l'arborescence des résolutions qui ont conduit à la clause vide, cette arborescence donne une explication de l'insatisfiabilité de S, puisque la règle de résolution est une règle de déduction.

- **Pourquoi l'algorithme peut-il ne jamais s'arrêter ? Est-ce un « défaut » de la méthode de résolution ?**

En logique des propositions, l'ensemble des interprétations possibles d'un ensemble fini de symboles propositionnels est fini. On peut donc décider si une formule est valide (ou si une formule est conséquence d'une autre, ce qui revient au même) en construisant une table de vérité qui énumère ces interprétations et vérifie que chacune rend la formule vraie (il existe des méthodes plus efficaces en pratique bien sûr). On peut donc construire un algorithme qui s'arrête quelle que soit la formule qui lui est fournie : le problème de la validité d'une formule est donc **décidable**. Un algorithme qui s'arrête dans tous les cas est appelé **procédure de décision**.

En logique du premier ordre, l'ensemble des interprétations d'un langage est infini (il existe une infinité de domaines possibles, et un domaine peut lui-même être infini). En 1936, Alonzo Church a prouvé le résultat suivant :

Théorème de Church

Le problème de la **validité** d'une formule en logique du premier ordre n'est **pas décidable**.

Le problème de la satisfiabilité ou de l'insatisfiabilité d'une formule n'est donc **pas décidable** non plus. Le fait que l'algorithme de la méthode de résolution puisse ne pas s'arrêter n'est donc pas un « défaut » de cet algorithme : d'après le théorème de Church, aucun algorithme ne s'arrête dans tous les cas. Cependant, on a des procédures de décision pour des sous-ensembles de la logique du premier ordre, par exemple les *clauses de Horn* (clauses ayant au plus un littéral positif). Ce type de formules est notamment utilisé dans les moteurs de règles (qui forment la base des systèmes experts) et en programmation logique (voir Prolog).

2.5. Stratégies de résolution

Il existe de nombreuses manières *d'appliquer* la méthode de résolution, et qu'on appelle des *stratégies de résolution*. Nous décrivons très brièvement quelques stratégies.

Une stratégie est dite **complète** si elle permet d'obtenir la clause vide lorsque la forme clausale initiale est insatisfiable (et il n'y a bien sûr aucune garantie d'arrêt, en logique du premier ordre, si la forme clausale initiale n'est pas insatisfiable).

Stratégies en largeur

Le principe général d'une telle stratégie est de considérer l'ensemble initial de clauses comme le niveau 0 et de construire l'ensemble de clauses de niveau i en résolvant, de toutes les manières

possibles, une clause de niveau $i - 1$ et une clause de niveau $\leq i - 1$. Cette stratégie est complète mais inefficace.

Stratégies linéaires, SLD résolution et Prolog

Une stratégie est **linéaire** si à chaque application de la règle de résolution, on utilise au moins une clause de l'ensemble initial S . Ce type de stratégies n'est pas complet en général.

La **SLD-résolution** est une stratégie linéaire particulière : on applique la règle de résolution à partir de la *dernière* clause obtenue et d'une clause initiale. Cette stratégie est complète dans le cas où toutes les clauses initiales sont des clauses de Horn, c'est-à-dire des clauses contenant au plus un littéral positif.

C'est la stratégie de PROLOG, qui, de plus, précise la première étape : on considère la *clause but* (cette clause est la négation de la question ; la question étant une conjonction d'atomes quantifiée existentiellement, sa négation est donc une clause qui n'a que des littéraux négatifs) et une des *clauses règles*. La résolvente obtenue est le nouveau but. A chaque étape, on considère ainsi le but courant et l'une des règles du programme.

Prolog construit un **arbre SLD**, dont la racine est étiquetée par la clause but (négation de la question). Les fils d'un sommet x de l'arbre sont chacun étiquetés par la résolvente obtenue en unifiant le premier sous-but étiquetant x avec la tête d'une règle. Les feuilles sont soit des sommets succès (cas où la clause vide a été obtenue), soit des sommets échecs (cas où le but courant ne peut pas être *effacé* : aucune unification n'est possible entre le premier sous-but courant et une tête de règle). L'arbre peut avoir des chemins infinis. La substitution correspondant à une feuille "succès" définit une réponse à la question.

L'arbre SLD construit par Prolog peut également être vu comme l'application en **chaînage arrière** des règles du programme (rappelons qu'en Prolog, les faits sont vus comme des règles réduites à une tête, autrement dit avec un corps vide). Le chaînage arrière appliqué à un programme Prolog consiste en effet à itérer l'opération suivante :

- unifier le premier sous-but $B1$ du but courant B avec la tête de la première règle $R = C :- H$ qui convient ; soit u l'unificateur ;
- le nouveau but est $(u(B) - \{u(B1)\}) \cup u(H)$; si H est vide (R est donc un fait), le nouveau but est $(u(B) - \{u(B1)\})$.

Lorsque le but est complètement effacé (le nouveau but est vide), le but initial a été prouvé ; la composition des unificateurs u définit une réponse au but initial.

Prolog construit l'arbre SLD en *profondeur d'abord*, ce qui rend sa stratégie de résolution incomplète : Prolog peut en effet partir dans l'exploration d'une branche infinie et ne jamais explorer le reste de l'arbre. S'il explorait l'arbre SLD en largeur, sa stratégie serait complète. L'ordre des règles dans un programme Prolog a donc son importance, ainsi que l'ordre des atomes dans une règle. Le comportement du programme peut être très différent selon l'ordre choisi. En conséquence, bien qu'il soit basé sur la logique du premier ordre, Prolog s'en écarte : deux programmes correspondant au même ensemble de formules peut avoir un comportement différent (étant donné une question, l'ensemble des réponses obtenu n'est pas le même).