Cours de logique pour l'informatique

Prof. Jean-François Raskin
Département d'Informatique
Faculté des Sciences
Université Libre de Bruxelles

Année académique 2007-2008

Organisation pratique du cours :

Références – Logic in Computer Science : Modeling and Reasoning about Systems, M. R. A. Huth and M. Ryan, Cambridge University Press, 1999.

– Mathematical Logic for Computer Science, M. Ben-Ari, Prentice Hall, 1993.

Matériel Les slides utilisés lors des cours seront disponibles sur la page web du cours.

Travaux pratiques Assistant : Mr Cédric Meuter

Contact – email : `Jean-Francois.Raskin@ulb.ac.be`

– tel : `650 55 92`

– http : `www.ulb.ac.be/di/ssd/jfr`

Introduction - Motivations

Les logiques sont utilisées en informatique pour

Modéliser de manière *formelle* des "objets" rencontrés par les informaticiens ;

Ex : Bases de données, bases de connaissances, pré-post conditions d'une procédure, etc.

Raisonner Après une phase de modélisation, l'informaticien doit être cable de se servir du modèle et raisonner sur celui-ci.

Ex : validation d'un modèle de données, prise de décision à partir de faits et d'une base de connaissances, preuve de correction d'une procédure/d'un programme ;

La logique est à la base de l'étude des raisonnements, c'est-à-dire des déductions que l'on peut faire sur les modèles formels.

Introduction - Motivations

Bénéfices à l'utilisation des formalismes logiques :

- modèles précis et non-ambigus, demarche rigoureuse, etc.
- possibilité d'automatiser/semi-automatiser l'analyse des modèles.

Introduction - Motivations
Exemples

Considérons la situation décrite par les affirmations suivantes :

1. **Si** le train arrive en retard et il n'y a pas de taxis à la gare **alors** l'invité arrive en retard.
2. L'invité n'est pas en retard.
3. Le train est arrivé en retard.

Et la déduction suivante :

donc, il y avait des taxis à la gare.

Introduction - Motivations
Exemples

Pourquoi peut-on déduire qu'il y avait des taxis à la gare ? Premièrement, si on met l'affirmation 1 et l'affirmation 3 ensemble, on peut affirmer que s'il n'y avait pas eu de taxis à la gare, alors l'invité serait arrivé en retard. Deuxièmement, cette dernière affirmation n'est compatible avec le fait 2 que s'il y avait des taxis à la gare. Donc il est consistant de déduire qu'il y avait des taxis à la gare.

Introduction - Motivations
Exemples

Considérons un autre exemple :

1. **Si** il pleut **et** l'invité a oublié son parapluie **alors** l'invité est trempé.
2. L'invité n'est pas trempé.
3. Il pleut.

Et la déduction suivante :

donc, l'invité n'a pas oublié son parapluie.

Introduction - Motivations
Exemples

Pour justifier la déduction “l’invité n’a pas oublié son parapluie”, on peut “réutiliser” le raisonnement fait précédement.

Pourquoi ?
Les deux situations décrites ont la même **structure logique**. Cela est reflété par les opérateurs logiques utilisés dans les phrases : **si** ... **alors**, **alors**, **donc**, ...

Malheureusement les langages naturels comme le français, l’anglais ou encore certains langages graphiques (UML par exemple), manquent de précision et sont ambigus. C’est pourquoi il y a un besoin pour des langages mathématiques capables de **modéliser** et d’aider au **raisonnement**.

Introduction - Motivations
Exemples

Voyons comment on pourrait formaliser les situations décrites dans les deux exemples précédents. On peut les formaliser à l'aide de la logique propositionnelle. Considérons que

- la proposition p est utilisée pour représenter la situation "le train arrive en retard",
- q pour "il y a des taxis à la gare"
- r pour "l'invité est en retard"

Introduction - Motivations
Exemples

Alors nous pouvons *formaliser* la situation décrite dans le premier exemple par la *formule logique* suivante :

$$((p \wedge \neg q) \rightarrow r) \wedge \neg r \wedge p$$

Et nous pouvons *formaliser* la déduction par :

$$((p \wedge \neg q) \rightarrow r) \wedge \neg r \wedge p \models q$$

Où "$\models$" se lit : "q est une conséquence logique de $((p \wedge \neg q) \rightarrow r) \wedge \neg r \wedge p$".

Introduction - Motivations
Exemples

Considérons le test suivant :

$$\textbf{if } (x \geq 3 \textbf{ and } z \leq 3) \textbf{ or } (y \leq 2 \textbf{ and } z \leq 3) \textbf{ then } ...$$

Il peut-être remplacé par

$$\textbf{if } (x \geq 3 \textbf{ or } \ y \leq 2) \textbf{ and } z \leq 3 \textbf{ then } ...$$

Car $(p \vee q) \wedge r$ est logiquement équivalent à $(p \wedge r) \vee (q \wedge r)$.

Introduction - Motivations
Un autre exemple

Considérons les règles suivantes :

1. les parents d'un enfant sont des ancètres de cet enfant ;
2. les ancêtres d'une personne qui est l'ancêtre d'une deuxième personne sont également ancètres de cette deuxième personne ;
3. deux personnes sont d'une même famille si et seulement si elles ont un ancêtre commun.

Introduction - Motivations
Un autre exemple

- à la différence des deux exemples précédents, on parle ici de propriétés qui sont vraies pour des catégories : les enfants, les personnes, ...
- la logique propositionnelle n'est pas suffisante pour modéliser cette situation, c'est pourquoi nous étudierons une extension de la logique propositionnelle qui est la *logique du premier ordre.*

Introduction - Motivations
La logique du premier ordre

Les objets sont représentés par des variables (du premier ordre) : $x, y, z, \ldots$

Formalisation de la règle :

1.

$$\forall x, y : Parent(x, y) \rightarrow Ancetre(x, y)$$

2.

$$\begin{aligned} \forall x, y : \quad & \exists z : Ancetre(x, z) \wedge Ancetre(z, y) \\ & \rightarrow Ancetre(x, y) \end{aligned}$$

3.

$$\begin{aligned} \forall x, y : \quad & Apparente(x, y) \\ & \leftrightarrow \exists z : Ancetre(z, x) \wedge Ancetre(z, y) \end{aligned}$$

Introduction - Motivations
La logique du premier ordre

La formalisation de cet exemple permet maintenant de faire des "calculs" pour raisonner sur cette règle. Par exemple, il est possible de poser de manière précise des questions comme :

– est-ce qu'un parent x et son enfant y sont apparentés ?

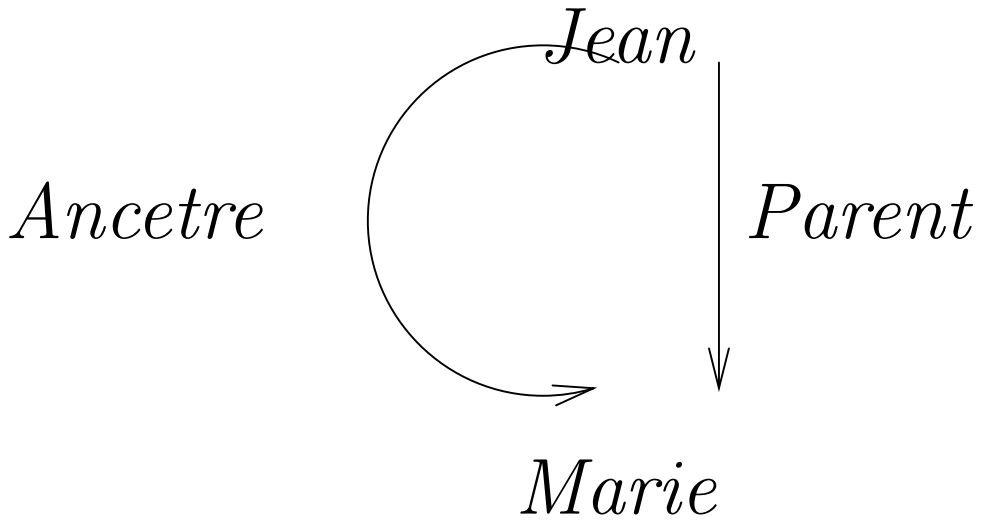
$$SpecApparente \models^{?}$$
$$\forall x, y : Parent(x, y) \rightarrow Apparente(x, y)$$

– est-ce que la relation $Apparente$ est symétrique ?

$$SpecApparente \models^{?}$$
$$\forall x, y : Apparente(x, y) \rightarrow Apparente(y, x)$$

Note : répondre à ces questions de manière formelle revient à faire une preuve qui établit la propriéte ou à construire un contre-exemple.

Considérons le situation suivante :



On n'a pas $Apparente(Jean, Marie)$. C'est assez contre-intuitif, il faut peut-être ajouter :

$$\forall x : Ancetre(x, x)$$

Donc une formalisation ne représente pas toujours ce que l'on veut vraiment, et on est parfois amené à l'améliorer.

Introduction - Motivations
Programmation logique

La formule *SpecApparente* peut être vue comme une *spécification* d'une requête sur une base de données qui peut-être traduite en `COBOL`, par exemple.

Mais étant donnée une base de faits, établir la liste des personnes apparentées à un individu revient à faire des preuves.

Donc si on peut automatiser ces preuves, on a une alternative à la traduction en `COBOL` : le résultat de la requête est simplement l'ensemble des paires (x, y) pour lesquelles, le *theorem prover* peut établir :

$$SpecApparente \wedge Faits \models Apparente(x, y)$$

Introduction - Motivations
Prolog : programmation logique

Il y a un langage qui est basé sur cette idée : PROLOG. Quelques caractéristiques de PROLOG :

- programmation déclarative ;
- beaucoup de “calculs à la ligne” ;
- permet de développer rapidement des prototypes ;
- utilisé pour construire des systèmes experts, etc.

Introduction - Motivations

Le but de ce cours est d'étudier en détails les fondements de la logique et vous donner une formation suffisante pour que vous puissiez vous familiariser avec d'autres logiques que vous rencontrerez plus tard. Et également vous sensibiliser au fait que la logique peut être très utile pour automatiser/semi-automatiser les tâches de raisonnement rencontrées lors de la construction/analyse de modèles et de programmes.

Plan du cours

- la logique propositionnelle ;
- la logique du premier ordre ;
- la programmation logique ;
- questions avancées.

La logique propositionnelle
Plan

- Syntaxe de la logique propositionnelle ;
- Sémantique de la logique propositionnelle ;
- Procédure de décision : tableaux sémantiques ;
- La déduction naturelle ;
- La résolution propositionnelle.

La logique propositionnelle
Syntaxe - Construction des formules (I)

Le *vocabulaire du langage de la logique propositionnelle* est composé :

- d'un ensemble, fini ou dénombrable, de *propositions* notées $p, q, r, \ldots$

  Dans la suite, nous notons un sous-ensemble de propositions par les lettres P, Q,... ;
- de deux constantes : vrai (notée $\top$) et faux (notée $\bot$) ;
- d'un ensemble de *connecteurs logiques* : et (noté $\wedge$), ou (noté $\vee$), non (noté $\neg$), implique (noté $\rightarrow$), équivalent (noté $\leftrightarrow$) ;
- les parenthèses : (,).

La logique propositionnelle
Syntaxe - Construction des formules (II)

Les *formules de la logique propositionnelle* respectent la règle de formation `BNF` suivante :

$$\phi ::= \top \mid \bot \mid p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$$

$$\mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \leftrightarrow \phi_2 \mid \neg\phi_1 \mid (\phi_1)$$

Où $\top$, $\bot$ sont respectivement les constantes vrai et faux, $p \in P$ est une proposition et ϕ_1, ϕ_2 sont des formules propositionnelles bien formées. L'ensemble des formules propositionnelles bien formées sera noté FormulesProp.

La logique propositionnelle
Quelques exemples de formules

Voici quelques exemples de formules et leur lecture intuitive :

- la formule propositionnelle "$p \rightarrow (q \wedge r)$", peut être lue de la façon suivante "p implique q et r", ou peut être également lue comme "si p est vrai alors q et r doivent être vrais" ;
- la formule propositionnelle "$\neg(p \wedge q)$", peut-être lue de la façon suivante "il est faux que p et q soient vrais (en même temps)".

La logique propositionnelle
Ambiguïtés (I)

L'utilisation de la *notation infixe* s'accompagne de problèmes de parsing :

$$
\begin{aligned}
& e_1\, op_1\, e_2\, op_2\, e_3 \\
=^{?}\ & (e_1\, op_1\, e_2)\, op_2\, e_3 \\
=^{?}\ & e_1\, op_1 (e_2\, op_2\, e_3)
\end{aligned}
$$

La logique propositionnelle
Ambiguïtés (II)

Pour lever les ambiguïtés, on utilise les parenthèses ou des règles de priorité entre opérateurs :

– si op_1 a une plus grande précédence (priorité) que op_2 alors

$$e_1\, op_1\, e_2\, op_2\, e_3$$

est équivalent à $((e_1\, op_1\, e_2)\, op_2\, e_3)$

– si op est associatif à gauche alors

$$e_1\, ope_2\, ope_3$$

est équivalent à $((e_1\, ope_2)\, ope_3)$

Une fois ces règles fixées, à chaque formule correspond un et un seul arbre de parsing.

La logique propositionnelle
Règles de précédence

Ordre de précédence sur les opérateurs :

$$\neg$$
$$\wedge$$
$$\vee$$
$$\rightarrow$$
$$\leftrightarrow$$

et associativité à gauche.

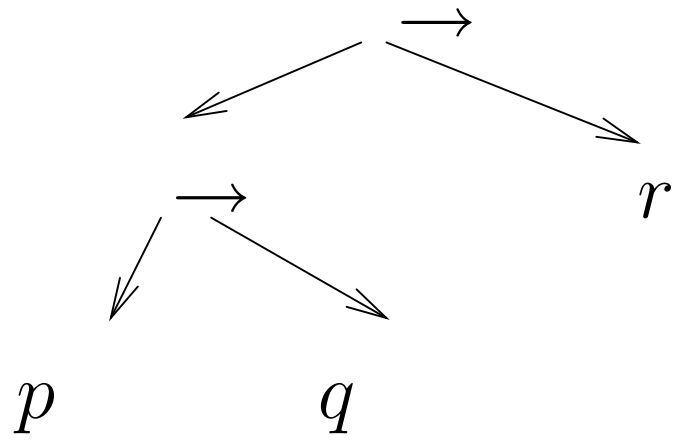
Les parenthèses permettent de contrecarrer ces règles, si elles ne conviennent pas. Elles permettent aussi de rendre une formule plus lisible, ou de ne pas devoir retenir les règles de précédence.

La logique propositionnelle
Arbre correspondant à une formule

La formule $p \to q \to r$ est donc équivalente à

$$((p \to q) \to r)$$

et donc son arbre de parsing est :



La logique propositionnelle
Profondeur d'une formule

La *profondeur d'une formule* ϕ est la profondeur de l'arbre de parsing A_ϕ associé à la formule.

Elle se définit de manière inductive comme suit :

- <u>cas de base</u> : si $\phi = p, \top, \bot$ où p est une proposition alors $Prof(\phi) = 0$;
- <u>cas inductif</u> : si $\phi = \phi_1 \, op \, \phi_2$ alors
  $Prof(\phi) = 1 + \mathbf{max}(Prof(\phi_1), Prof(\phi_2))$;
  si $\phi = (\phi_1)$ alors $Prof(\phi) = Prof(\phi_1)$.

Nous notons $\textsc{FormulesProp}_i$ les formules propositionnelles de profondeur i. Notons que

$$\textsc{FormulesProp} = \bigcup_{i \geq 0} \textsc{FormulesProp}_i$$

La logique propositionnelle
Rappel : l'induction mathématique (I)

Il est bien connu que pour $n \geq 1$, on a

$$1 + 2 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Pour démontrer que cette égalité, notée $M(n)$, est valide pour n'importe quel $n \geq 1$, on applique le principe d'*induction mathématique* .

La logique propositionnelle
Rappel : l'induction mathématique (II)

SI

- Cas de base : on établit que la propriété est vraie pour $n = 1$, c'est-à-dire que l'on prouve que $M(1)$ est vraie.
- Cas inductif : en faisant l'hypothèse que la propriété est vraie pour $n = i$ (ou pour $n \leq i$), on établit que la propriété est également vraie pour $n = i + 1$.

ALORS

Le principe d'induction affirme que $M(n)$ est vraie pour tout $n \geq 1$.

La logique propositionnelle
Rappel : l'induction mathématique (III)

Nous utiliserons le principe d'induction mathématique pour prouver des propriétés qui sont vraies pour toutes formules de la logique propositionnelle, et ce en raisonnant par induction sur la profondeur des formules.

La logique propositionnelle
La sémantique (I)

Etant donné un ensemble de propositions P, une *fonction d'interprétation* V pour P est une fonction du type $V : P \rightarrow \{\text{vrai}, \text{faux}\}$, c'est-à-dire une fonction qui assigne à chaque proposition de P la valeur vrai ou la valeur faux. L'ensemble des fonctions d'interprétation propositionnelle sera noté FonctInterProp.

Dans la suite, nous utiliserons parfois le terme *valuation* ou bien de fonction d'interprétation.

La logique propositionnelle
La sémantique (II)

La *valeur de vérité* d'une formule propositionnelle ϕ formée à partir (d'une des propositions de l'ensemble P, évaluée avec la fonction d'interprétation V, est notée $[\![\phi]\!]_V$. Formellement, $[\![\cdot]\!]. : \textsc{FormulesProp} \times \textsc{FonctInterProp} \rightarrow \{\text{vrai}, \text{faux}\}$. La fonction $[\![\phi]\!]_V$ est définie par induction sur la syntaxe de ϕ de la façon suivante :

- $\phi = \top$, dans ce cas, $[\![\phi]\!]_V = \text{vrai}$;
- $\phi = \bot$, dans ce cas, $[\![\phi]\!]_V = \text{faux}$;
- $\phi = p$, dans ce cas, $[\![\phi]\!]_V = V(p)$.
- $\phi = \neg\phi_1$, $[\![\phi]\!]_V = \begin{cases} \text{faux} & \underline{\text{si}}\ [\![\phi_1]\!]_V = \text{vrai} \\ \text{vrai} & \underline{\text{si}}\ [\![\phi_1]\!]_V = \text{faux} \end{cases}$

- $\phi = \phi_1 \vee \phi_2$,

$$[\![\phi]\!]_V = \text{vrai } \underline{\text{ssi}} \quad [\![\phi_1]\!]_V = \text{vrai}$$
$$\underline{\text{ou}}\ [\![\phi_2]\!]_V = \text{vrai}$$

- $\phi = \phi_1 \wedge \phi_2$,

$$[\![\phi]\!]_V = \text{vrai } \underline{\text{ssi}} \quad [\![\phi_1]\!]_V = \text{vrai}$$
$$\underline{\text{et}}\ [\![\phi_2]\!]_V = \text{vrai}$$

- $\phi = \phi_1 \rightarrow \phi_2$,

$$[\![\phi]\!]_V = \text{vrai } \underline{\text{ssi}} \quad [\![\phi_1]\!]_V = \text{faux}$$
$$\underline{\text{ou}}\ [\![\phi_2]\!]_V = \text{vrai}$$

– $\phi = \phi_1 \leftrightarrow \phi_2$,

$$[\![\phi]\!]_V = \text{vrai } \underline{\text{ssi}}$$
$$[\![\phi_1]\!]_V = \text{faux } \underline{\text{et}}\ [\![\phi_2]\!]_V = \text{faux}$$
$$\underline{\text{ou}}\ [\![\phi_1]\!]_V = \text{vrai } \underline{\text{et}}\ [\![\phi_2]\!]_V = \text{vrai}$$

– $\phi = (\phi_1)$, $[\![\phi]\!]_V = [\![\phi_1]\!]_V$.

Nous notons $V \models \phi$ <u>si et seulement si</u> $[\![\phi]\!]_V = \text{vrai}$.

La logique propositionnelle
La sémantique (III)

L'information contenue dans la définition précédente est souvent présentée sous forme de tables, appelées *tables de vérité* :

$\top$	$\bot$
vrai	faux

p	$\neg p$	(p)
vrai	faux	vrai
faux	vrai	faux

La logique propositionnelle
La sémantique (IV)

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
vrai	vrai	vrai	vrai	vrai	vrai
vrai	faux	faux	vrai	faux	faux
faux	vrai	faux	vrai	vrai	faux
faux	faux	faux	faux	vrai	vrai

La logique propositionnelle
La sémantique (V)

Voici deux définitions importantes :

- Une formule propositionnelle ϕ est *valide* si et seulement si pour toute fonction d'interprétation V pour les propositions de ϕ, on a $V \models \phi$.
- Une formule propositionnelle ϕ est *satisfaisable* si et seulement si il existe une fonction d'interprétation V pour les propositions de ϕ, telle que $V \models \phi$.

Une conséquence directe de ces deux définitions est que :

Une formule propositionnelle ϕ est valide ssi sa négation $\neg\phi$ n'est pas satisfaisable.

Theorème : Une formule propositionnelle ϕ est valide ssi sa négation $\neg\phi$ n'est pas satisfaisable.

Preuve. On applique simplement les définitions. Par définition ϕ est valide si, pour toute fonction d'interprétation V, $V \models \phi$, ce qui est équivalent à $V \not\models \neg\phi$, par définition de la sémantique de $\neg$ et donc si ϕ est valide, il n'existe pas de fonction d'interprétation V telle que $V \models \neg\phi$ et donc $\neg\phi$ n'est pas satisfaisable. L'autre direction est laissée comme exercice.

♣

La logique propositionnelle
La sémantique (VI)

Une formule propositionnelle ψ est une *conséquence logique* d'un ensemble de formules propositionnelles $\{\phi_1, \ldots, \phi_n\}$
si et seulement si on a que pour toute fonction d'interprétation V (sur les propositions de $\psi, \phi_1, \ldots, \phi_n$), telle que $V \models \phi_i$ pour tout i, $1 \leq i \leq n$, on a également $V \models \psi$; ce fait est noté $\phi_1, \ldots, \phi_n \models \psi$.

Deux formules ϕ et ψ sont *equivalentes* si et seulement si on a $\phi \models \psi$ et $\psi \models \phi$.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (I)

Algorithme pour établir la satisfaisabilité/validité de formules de la logique propositionnelle.

Le principe est très simple : pour prouver la satisfaisabilité d'une formule de la logique propositionnelle, on cherche systématiquement un modèle pour cette formule.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (II)

On a besoin de quelques nouvelles définitions :

Une formule propositionnelle ϕ est un *littéral* si et seulement si ϕ est une proposition ou la négation d'une proposition.

Deux formules ϕ et $\neg\phi$ sont des *formules complémentaires* .

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (III)

Considérons la formule $\phi = p \wedge (\neg q \vee \neg p)$

Essayons de construire systématiquement une fonction d'interprétation V telle que

$$[\![\phi]\!]_V = \text{vrai}$$

Par définition on a

$$[\![\phi]\!]_V = \text{vrai}$$

$$\underline{\text{ssi}}$$

$$[\![p]\!]_V = \text{vrai} \;\; \underline{\text{et}} \;\; [\![\neg q \vee \neg p]\!]_V = \text{vrai}$$

et

$$[\![\neg q \vee \neg p]\!]_V = \text{vrai}$$
$$\underline{\text{ssi}}$$
$$[\![\neg q]\!]_V = \text{vrai} \;\; \underline{\text{ou}} \;\; [\![\neg p]\!]_V = \text{vrai}$$

et donc,

$$[\![\phi]\!]_V = \text{vrai}$$
$$\underline{\text{ssi}}$$
$$[\![p]\!]_V = \text{vrai} \; \underline{\text{et}} \; [\![\neg q]\!]_V = \text{vrai}$$
$$\underline{\text{ou}} \; [\![p]\!]_V = \text{vrai} \; \underline{\text{et}} \; [\![\neg p]\!]_V = \text{vrai}$$

Pour ϕ, on construit la fonction d'interprétation V de la façon suivante :

– $V(p) = \text{vrai}$;

– $V(q) = \text{faux}$.

et on a bien que $V \models \phi$.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (IV)

Nous avons réduit le problème de satisfaction de ϕ à un problème de satisfaction d'ensembles de littéraux.

Un ensemble S de littéraux est satisfaisable ssi il ne contient pas une paire de *littéraux complémentaires* .

Vu que chaque formule contient un ensemble fini de propositions, et donc un ensemble fini de littéraux, il y a un nombre fini d'ensembles de littéraux.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (V)

Un autre exemple : $\phi = (p \vee q) \wedge (\neg p \wedge \neg q)$.

Par définition, on a

$$[\![\phi]\!]_V = \text{vrai}$$
$$\underline{\text{ssi}}$$
$$[\![p \vee q]\!]_V = \text{vrai } \underline{\text{et}} \;\; [\![\neg p \wedge \neg q]\!]_V = \text{vrai}$$

et donc,

$$[\![\phi]\!]_V = \text{vrai}$$
$$\underline{\text{ssi}}$$
$$[\![p \vee q]\!]_V = \text{vrai } \underline{\text{et}} \;\; [\![\neg p]\!]_V = \text{vrai } \underline{\text{et}} \;\; [\![\neg q]\!]_V = \text{vrai}$$

et donc,

$$[\![\phi]\!]_V = \text{vrai}$$
$$\underline{\text{ssi}}$$

$$\text{soit } [\![p]\!]_V = \text{vrai } \underline{\text{et}} \;\; [\![\neg p]\!]_V = \text{vrai } \underline{\text{et}} \;\; [\![\neg q]\!]_V = \text{vrai}$$
$$\underline{\text{ou}} \; [\![q]\!]_V = \text{vrai } \underline{\text{et}} \;\; [\![\neg p]\!]_V = \text{vrai } \underline{\text{et}} \;\; [\![\neg q]\!]_V = \text{vrai}$$

Vu que les deux ensembles de littéraux ont des paires complémentaires, la formule ϕ n'est donc pas satisfaisable.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (VI)

L'information présente dans les développements précédents est plus facilement représentée sous forme d'un arbre.

Exemple : $\phi = (p \vee q) \wedge (\neg p \wedge \neg q)$ (tableau donné au cours)

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (VII)

Remarques :

- quand on applique le "pas de simplification" à un noeud où on sélectionne une conjonction, alors on obtient un seul fils ; on appelle ces règles, des α-règles ;
- quand on applique le "pas de simplification" à un noeud où on sélectionnne une disjonction, alors on obtient deux fils ; on appelle ces règles, des β-règles.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (VIII)

α	α_1	α_2
$\neg\neg\phi$	ϕ	
$\phi_1 \wedge \phi_2$	ϕ_1	ϕ_2
$\neg(\phi_1 \vee \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\neg(\phi_1 \rightarrow \phi_2)$	ϕ_1	$\neg\phi_2$
$\phi_1 \leftrightarrow \phi_2$	$\phi_1 \rightarrow \phi_2$	$\phi_2 \rightarrow \phi_1$

α-règles .

Remarque : toutes les formules α peuvent être considérées comme équivalentes à des conjonctions. Par exemple :

$\neg(\phi_1 \vee \phi_2)$ est équivalent à $\neg\phi_1 \wedge \neg\phi_2$

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (IX)

β	β_1	β_2
$\phi_1 \vee \phi_2$	ϕ_1	ϕ_2
$\neg(\phi_1 \wedge \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\phi_1 \rightarrow \phi_2$	$\neg\phi_1$	ϕ_2
$\neg(\phi_1 \leftrightarrow \phi_2)$	$\neg(\phi_1 \rightarrow \phi_2)$	$\neg(\phi_2 \rightarrow \phi_1)$

β-règles .

Remarque : toutes les formules β peuvent être considérées comme équivalentes à des disjonctions. Par exemple :

$\neg(\phi_1 \wedge \phi_2)$ est équivalent à $\neg\phi_1 \vee \neg\phi_2$

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (X)

Algorithme de construction d'un *tableau sémantique* , quelques notations :

- un arbre représentant le tableau sémantique d'une formule ϕ sera noté T_ϕ ;
- l, n, m représentent des noeuds d'un arbre ;
- $\mathsf{Lab}(l)$ dénote l'étiquette du noeud l, c'est un ensemble de formules ;
- $\mathsf{Status}(l)$ est le statut du noeud l, ce statut est $-$ pour tout noeud qui n'est pas une feuille et $\mathsf{Status}(l) \in \{ouvert, ferm\}$ si l est une feuille ;
- Q, R dénotent des ensembles de noeuds.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XI)

Algorithme :

- Initialisation : $R := \{l\}$; $\mathsf{Lab}(l) := \{\phi\}$;
- Itérer tant que $R \neq \emptyset$: choisir $l \in R$:
  - **si** $\mathsf{Lab}(l)$ est un ensemble de littéraux **alors** :
    - Si $\mathsf{Lab}(l)$ contient une paire de littéraux complémentaires : $\mathsf{Status}(l) := \mathit{ferm}$ et $R := R \setminus \{l\}$;
    - Si $\mathsf{Lab}(l)$ ne contient pas de paires de littéraux complémentaires : $\mathsf{Status}(l) := \mathit{ouvert}$ et $R := R \setminus \{l\}$;
  - **si** $\mathsf{Lab}(l)$ n'est pas un ensemble de littéraux **alors** : choisir $\psi \in \mathsf{Lab}(l)$ qui n'est pas un littéral,
    - si ψ est une α-formule alors créer un fils m pour l et :
      - $R := R \cup \{m\} \setminus \{l\}$;
      - $\mathsf{Lab}(m) := (\mathsf{Lab}(l) \setminus \{\psi\}) \cup \{\alpha_1, \alpha_2\}$;
    - si ψ est une β-formule alors créer deux fils, m et n, pour l et :
      - $R := R \cup \{m, n\} \setminus \{l\}$;
      - $\mathsf{Lab}(m) := (\mathsf{Lab}(l) \setminus \{\psi\}) \cup \{\beta_1\}$;
      - $\mathsf{Lab}(n) := (\mathsf{Lab}(l) \setminus \{\psi\}) \cup \{\beta_2\}$;

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XI)

Un tableau dont la construction est terminée est appelé un *tableau complet* .

Un *tableau fermé* est un tableau complet dont toutes ses feuilles sont étiquettées avec *fermé*. C'est un *tableau ouvert* sinon.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XII)

Théorème : Si T_ϕ est le tableau complet de la formule ϕ, alors ϕ est non satisfaisable ssi T_ϕ est fermé.

Avant de prouver la correction de l'algorithme par rapport au théorème ci-dessus observons quelques conséquences de ce théorème :

– ϕ est satisfaisable ssi T_ϕ est ouvert ;

– ϕ est valide ssi $T_{\neg\phi}$ est fermé.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XII)

Correction de l'algorithme : qu'est-ce que cela veut dire ?

- *terminaison* : l'algorithme termine pour toutes les formules de la logique propositionnelle ;
- *adéquation* : si l'algorithme construit un tableau qui ferme pour une formule ϕ alors la formule ϕ est non satisfaisable ;
- *complétude* : pour toute formule ϕ non satisfaisable, l'algorithme construit un tableau fermé.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XIII)

Terminaison de l'algorithme :

On définit une fonction de "poids" qui associe à chaque ensemble de formules un nombre naturel ; on note $\mathsf{Poids}(l)$ le poids de l'étiquette du noeuds l.

On montre que si m est un fils de l alors

$$\mathsf{Poids}(m) < \mathsf{Poids}(l).$$

Vu que $\mathsf{Poids}()$ est une fonction des ensembles de formules vers les naturels alors la profondeur de tout tableau sémantique est finie et donc l'algorithme termine.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XIII)

Définition de la fonction $\mathsf{Poids}(\Psi)$. On définit d'abord cette fonction pour une formule ψ par induction sur la structure de la formule.

- $\mathsf{Poids}(p) = 1$.
- $\mathsf{Poids}(\neg\psi) = 1 + \mathsf{Poids}(\psi)$;
- $\mathsf{Poids}(\psi_1 \wedge \psi_2) = 2 + \mathsf{Poids}(\psi_1) + \mathsf{Poids}(\psi_2)$;
- $\mathsf{Poids}(\psi_1 \vee \psi_2) = 2 + \mathsf{Poids}(\psi_1) + \mathsf{Poids}(\psi_2)$;
- $\mathsf{Poids}(\psi_1 \rightarrow \psi_2) = 2 + \mathsf{Poids}(\psi_1) + \mathsf{Poids}(\psi_2)$;
- $\mathsf{Poids}(\psi_1 \leftrightarrow \psi_2) = 2 \times (\mathsf{Poids}(\psi_1) + \mathsf{Poids}(\psi_2)) + 5$

et finalement, $\mathsf{Poids}(\Psi) = \sum_{\psi \in \Psi} \mathsf{Poids}(\psi)$.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XIII)

Propriétés :

– la fonction Poids() attribue un poids fini et strictement positif à chaque formule ;

– le poids de l'étiquette d'un noeud l est toujours strictement supérieur aux poids des étiquettes de ses fils.

Preuves laissées comme exercices.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XIV)

Adéquation de l'algorithme :

On doit montrer que "Si T_ϕ est fermé alors ϕ est non satisfaisable".

On montre une propriété plus forte : "Si un sous-arbre de racine l est fermé alors l'ensemble de formules $\mathsf{Lab}(l)$ est non satisfaisable". Pour établir cette propriété, on raisonne par induction sur la profondeur du sous-arbre.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XIV)

Lemme : Si un sous-arbre de racine l est fermé alors $\mathsf{Lab}(l)$ est non satisfaisable.

Preuve : Par induction sur la profondeur du sous-arbre.

CB : $Prof(l) = 0$. Alors $\mathsf{Lab}(l)$ est un ensemble de littéraux. Si l est fermé, $\mathsf{Lab}(l)$ contient une paire de littéraux complémentaires et donc $\mathsf{Lab}(l)$ est non satisfaisable.

CI : dans ce cas $\mathsf{Prof}(l) > 0$, et l'algorithme a sélectionné $\psi \in \mathsf{Lab}(l)$. Deux cas sont possibles :

1. ψ est une α-formule. Traitons le cas $\psi \equiv \neg\neg\phi$. Dans ce cas, l a un fils m. On a $\mathsf{Lab}(m) = \mathsf{Lab}(l) \setminus \{\neg\neg\phi\} \cup \{\phi\}$. On sait que le sous-arbre de racine m est fermé (dans le cas contraire l ne pourrait pas l'être). Par **HI**, on a que $\mathsf{Lab}(m)$ est non satisfaisable et donc pour toute valuation V, on a soit $V \not\models \mathsf{Lab}(l) \setminus \{\neg\neg\phi\}$ ou $V \not\models \phi$. Donc pour chaque valuation V

on sait que $V \not\models \mathsf{Lab}(l) \setminus \{\neg\neg\phi\}$ ou $V \not\models \neg\neg\phi$. Par conséquent, nous avons bien que pour toute valuation V, $V \not\models \mathsf{Lab}(l)$, et $\mathsf{Lab}(l)$ est non satisfaisable.

2. ψ est une β-formule. Traitons le cas $\psi \equiv \phi_1 \vee \phi_2$. Dans ce cas, l a deux fils m et n. On a $\mathsf{Lab}(m) = \mathsf{Lab}(l) \setminus \{\phi_1 \vee \phi_2\} \cup \{\phi_1\}$ et $\mathsf{Lab}(n) = \mathsf{Lab}(l) \setminus \{\phi_1 \vee \phi_2\} \cup \{\phi_2\}$. Par **HI** on sait que $\mathsf{Lab}(m)$ et $\mathsf{Lab}(n)$ sont non satisfaisables. Donc toute valuation V qui satisfait $\mathsf{Lab}(l) \setminus \{\phi_1 \vee \phi_2\}$ ne satisfait ni ϕ_1 ni ϕ_2, et donc ne satisfait pas $\phi_1 \vee \phi_2$. On a donc bien que $\mathsf{Lab}(l) \setminus \{\phi_1 \vee \phi_2\} \cup \{\phi_1 \vee \phi_2\} = \mathsf{Lab}(l)$ est non satisfaisable.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XV)

Complétude de l'algorithme :

On doit montrer que "Si ϕ est non satisfaisable alors T_ϕ est fermé".

Prouver cette propriété revient à établir la contraposée : "Si T_ϕ est ouvert alors la formule ϕ est satisfaisable".

Pour établir cela, nous avons besoin de quelques notions supplémentaires.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XVI)

Un ensemble de formules Ψ (on considère des littéraux, conjonctions ou disjonctions) a la *propriété de Hintikka* ssi les trois propriétés suivantes sont vérifiées :

1. pour toute proposition p, on a soit $p \notin \Psi$ ou $\neg p \notin \Psi$;
2. si $\{\phi_1 \vee \phi_2\} \in \Psi$ alors on a $\phi_1 \in \Psi$ ou $\phi_2 \in \Psi$;
3. si $\{\phi_1 \wedge \phi_2\} \in \Psi$ alors on a $\phi_1 \in \Psi$ et $\phi_2 \in \Psi$.

Propriété des ensembles de Hintikka :

Tout ensemble de formules Ψ qui a la propriété de Hintikka est satisfaisable.

Lemme Tout ensemble de formules Ψ qui a la propriété de Hintikka est satisfaisable.

Preuve. On construit une valuation V de la façon suivante :

$$V(p) = Vrai \text{ si } p \in \Psi$$

$$V(p) = Faux \text{ si } \neg p \in \Psi$$

$$\text{et (arbitrairement) } V(p) = Vrai$$
$$\text{si } p \notin \Psi \text{ et } \neg p \notin \Psi.$$

Montrons que toute formule $\phi \in \Psi$ est satisfaite par V.

On raisonne par induction sur la structure de ϕ :

CB : $\begin{cases} \phi = p, [\![\phi]\!]_V = V(p) = Vrai & (p \in \Psi) \\ \phi = \neg p, [\![\phi]\!]_V = \neg V(p) = Vrai & (\neg p \in \Psi) \end{cases}$

CI :

– $\phi = \phi_1 \vee \phi_2$.

Par définition des ensembles de Hintikka on sait que soit $\phi_1 \in \Psi$, soit $\phi_2 \in \Psi$. Par **HI**, si $\phi_1 \in \Psi$ on a $[\![\phi_1]\!]_V = \text{vrai}$, et si $\phi_2 \in \Psi$ on a $[\![\phi_2]\!]_V = \text{vrai}$. Donc $[\![\phi_1 \vee \phi_2]\!]_V = \text{vrai}$.

– $\phi = \phi_1 \wedge \phi_2$. On sait que $\phi_1 \in \Psi$ et $\phi_2 \in \Psi$. Par **HI** on a $[\![\phi_1]\!]_V = [\![\phi_2]\!]_V = \text{vrai}$ et donc $[\![\phi_1 \wedge \phi_2]\!]_V = \text{vrai}$.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XVII)

On appelle l'ensemble des noeuds que l'on parcourt depuis la racine d'un arbre jusqu'à une feuille l la *branche* de l ; on note $\mathsf{Branche}(A, l)$ la branche qui mène de la racine de l'arbre A à la feuille l.

On a le lemme suivant :

Soit l une feuille ouverte de T_ϕ alors

$$\bigcup_{m \in \mathsf{Branche}(A,l)} \mathsf{Lab}(m)$$

est un ensemble de formules qui a la propriété de Hintikka.

Preuve. on montre que chaque propriété est vérifiée :

1. Pour tout p, $p \notin \Psi$ ou $\neg p \notin \Psi$. En effet l est une feuille ouverte et donc elle ne contient pas de littéraux complémentaires.
2. Si $\phi_1 \vee \phi_2 \in \Psi$ alors on a appliqué une β-règle et soit $\phi_1 \in \Psi$ ou $\phi_2 \in \Psi$.
3. Si $\phi_1 \wedge \phi_2 \in \Psi$ alors on a appliqué une α-règle et $\phi_1 \in \Psi$ ainsi que $\phi_2 \in \Psi$.

La logique propositionnelle
Procédure de décision - Tableaux sémantiques (XVIII)

On obtient la preuve de complétude par le raisonnement suivant :

Si T_ϕ est ouvert alors il contient une feuille l qui est ouverte et donc

$$\bigcup_{m \in \mathsf{Branche}(A,l)} \mathsf{Lab}(m)$$

est un ensemble de formules qui a la propriété de Hintikka. Donc toutes les formules de

$$\bigcup_{m \in \mathsf{Branche}(A,l)} \mathsf{Lab}(m)$$

sont satisfaisables et en particulier

$$\phi \in \bigcup_{m \in \mathsf{Branche}(A,l)} \mathsf{Lab}(m).$$

La logique propositionnelle
Déduction naturelle (I)

L'algorithme introduit précédemment ne convient pas bien pour faire des raisonnements "à la main" sur des formules propositionnelles. Quand on fait des preuves en mathématique, on utilise des règles qui nous permettent, pas par pas, de déduire des conclusions à partir de prémisses.

La *déduction naturelle* est une formalisation de ce type de raisonnement.

La logique propositionnelle
Déduction naturelle (II)

Supposons données un ensemble de formules $\phi_1, \ldots, \phi_n$ que nous appelons *prémisses* et une formule ψ que nous appelons *conclusion* . Et que l'on désire montrer que ψ peut être dérivée de $\phi_1, \ldots, \phi_n$; on notera cette intention $\phi_1, \ldots, \phi_n \vdash \psi$, cette dernière expression est appelée un *séquent* . Cette notion de dérivation syntaxique sera intimement liée à la notion sémantique de *conséquence logique* ($\models$), en effet, on aura :

$$\phi_1, \ldots, \phi_n \vdash \psi \text{ ssi } \phi_1, \ldots, \phi_n \models \psi$$

La logique propositionnelle
Déduction naturelle (III)

Règles pour la conjonction :

– Introduction :

$$\frac{\phi \quad \psi}{\phi \wedge \psi} \wedge_i$$

– Elimination :

$$\frac{\phi \wedge \psi}{\phi} \wedge_{e_1} \qquad \frac{\phi \wedge \psi}{\psi} \wedge_{e_2}$$

La logique propositionnelle
Déduction naturelle (IV)

Exemples d'utilisation de ces règles pour la conjonction :

– $p \wedge q, r \vdash^{?} q \wedge r$.

1	$p \wedge q$	*prémisse*
2	r	*prémisse*
3	q	$\wedge_{e_2}, 1$
4	$q \wedge r$	$\wedge_i, 3, 2$

– $(p \wedge q) \wedge r, s \wedge t \vdash q \wedge s$;
Laissé comme exercice.

– $(p \wedge q) \wedge r \vdash p \wedge (q \wedge r)$;
Laissé comme exercice.

La logique propositionnelle
Déduction naturelle (IV)

Règles pour la (double) négation :

– Introduction :

$$\frac{\phi}{\neg\neg\phi}\neg_i$$

– Elimination :

$$\frac{\neg\neg\phi}{\phi}\neg_e$$

Exemple : $p, \neg\neg(q \wedge r) \vdash \neg\neg p \wedge r$

La logique propositionnelle
Déduction naturelle (V)

Règle pour l'élimination de l'implication (*Modus Ponens*) :

$$\frac{\phi \qquad \phi \rightarrow \psi}{\psi} \rightarrow_e$$

Illustration :

1. il pleut ;
2. si il pleut alors la route est mouillée.

alors on peut déduire que la route est mouillée.

La logique propositionnelle
Déduction naturelle (VI)

Etant données les prémisses

1. p ;
2. $p \rightarrow q$;
3. $p \rightarrow (q \rightarrow r)$

Peut-on déduire r ?

La logique propositionnelle
Déduction naturelle (VII)

Supposons que $p \rightarrow q$ et $\neg q$ soient vraies, si p est vrai alors par la règle du *modus ponens* on peut dériver q ce qui est en contradiction avec le fait que $\neg q$ est vrai, donc on en déduit que $\neg p$ est vrai. Ce raisonnement est résumé dans la règle suivante, appelée règle de *Modus Tollens* :

$$\frac{\phi \rightarrow \psi \quad \neg\psi}{\neg\phi} MT$$

La logique propositionnelle
Déduction naturelle (VIII)

Prouvons le séquent suivant :

$$p \rightarrow (q \rightarrow r), p, \neg r \vdash \neg q$$

1	$p \rightarrow (q \rightarrow r)$	*prémisse*
2	p	*prémisse*
3	$\neg r$	*prémisse*
4	$q \rightarrow r$	$\rightarrow_e 1et2$
5	$\neg q$	$MT4, 3$

La logique propositionnelle
Déduction naturelle (XI)

MT nous a permis de montrer :

$$p \rightarrow q, \neg q \vdash \neg p$$

et le séquent

$$p \rightarrow q \vdash \neg q \rightarrow \neg p$$

semble d'une certaine façon dire la même chose. Mais jusqu'à présent, nous n'avons pas de règles pour introduire le connecteur "implication".

La logique propositionnelle
Déduction naturelle (XII)

Le raisonnement à partir d'hypothèses :

On sait que $p \rightarrow q$ est vrai. Si on suppose (on fait l'hypothèse) que q est faux alors on peut déduire grâce à la règle MT que p est faux. Donc en supposant $\neg q$ on peut déduire $\neg p$ donc "$\neg q$ implique $\neg p$", exprimé de manière symbolique $\neg q \rightarrow \neg p$ est déduit.

L'argument pour $p \rightarrow q \vdash \neg q \rightarrow \neg p$ peut être résumé de la façon suivante :

1	$p \rightarrow q$	prémisse
2	$\neg q$	hyp.
3	$\neg p$	$MT 1, 2$, fin hyp.2
4	$\neg q \rightarrow \neg p$	$\rightarrow_i, 2-3$

La logique propositionnelle
Déduction naturelle (XIII)

Règle pour l'introduction de l'implication :

$$\dfrac{\begin{array}{c}\phi \quad \text{hyp.} \\ \vdots \\ \psi \quad \text{fin hyp.}\end{array}}{\phi \rightarrow \psi} \rightarrow_i$$

La logique propositionnelle
Déduction naturelle (XIV)

Prouvons le séquent $\neg q \rightarrow \neg p \vdash p \rightarrow \neg\neg q$:

1	$\neg q \rightarrow \neg p$	prémisse
2	p	hyp.
3	$\neg\neg p$	$\neg\neg_i, 2$
4	$\neg\neg q$	$MT, 1, 3,$ fin hyp.2
5	$p \rightarrow \neg\neg q$	$\rightarrow_i, 2-4$

La logique propositionnelle
Déduction naturelle (XV)

Preuves de formules vraies sans prémisse :

1	p	hyp.
2	$\neg\neg p$	$\neg\neg_i, 1,$ fin hyp.1
3	$p \rightarrow \neg\neg p$	$\rightarrow_i, 1-2$

Donc on a établi $\vdash p \rightarrow \neg\neg p$

Note : les formules que l'on peut établir sans prémisses sont les formules valides.

La logique propositionnelle
Déduction naturelle (XVI)

Prouvons le séquent suivant :

$$\vdash^{?} (q \to r) \to ((\neg q \to \neg p) \to (p \to r))$$

1	$q \rightarrow r$	hyp.
2	$\neg q \rightarrow \neg p$	hyp.
3	p	hyp.
4	$\neg\neg p$	$\neg\neg_i, 3$
5	$\neg\neg q$	$MT 2, 4$
6	q	$\neg\neg_e 5$
7	r	$\rightarrow_e 1, 6,$ fin hyp.3
8	$p \rightarrow r$	$\rightarrow_i 3-7,$ fin hyp.2
9	$(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)$	$\rightarrow_i 2, 8,$ fin hyp.1
10	$(q \rightarrow r)$	
	$\rightarrow ((\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r))$	$\rightarrow_i 1, 9$

La logique propositionnelle
Déduction naturelle (XVII)

Règle pour introduire la disjonction :

$$\frac{\phi}{\phi \vee \psi} \vee_{i_1} \qquad \frac{\psi}{\phi \vee \psi} \vee_{i_2}$$

La logique propositionnelle
Déduction naturelle (XVIII)

Comment éliminer la disjonction ?

Imaginons que l'on puisse :

1. établir une formule ϕ sous hypothèse que ψ_1 est vrai,
2. et que l'on puisse également établir ϕ sous hypothèse que ψ_2 est vrai,
3. et qu'il existe une preuve pour $\psi_1 \vee \psi_2$

alors on devrait pouvoir déduire ϕ.

Cette stratégie de preuve est formalisée dans la règle suivante :

$$\dfrac{\begin{array}{ccc} \psi_1 \vee \psi_2 & \begin{array}{c}\psi_1 \textsf{hyp.}\\ \vdots \\ \phi \textsf{fin hyp.}\end{array} & \begin{array}{c}\psi_2 \textsf{hyp.}\\ \vdots \\ \phi \textsf{fin hyp.}\end{array} \end{array}}{\phi} \vee_e$$

Attention : dans chacun des deux cas, on ne peut pas utiliser l'hypothèse temporaire faite pour l'autre cas, sauf si cette hypothèse a été établie avant.

La logique propositionnelle
Déduction naturelle (XIX)

Prouvons le séquent

$$p \vee q \vdash q \vee p$$

1	$p \vee q$	prémisse
2	p	hyp.
3	$q \vee p$	$\vee_{i_2}$, fin hyp.2
4	q	hyp.
5	$q \vee p$	$\vee_{i_1}$, fin hyp.4
6	$q \vee p$	$\vee_e 1, 2-3, 4-5$

La logique propositionnelle
Déduction naturelle (XX)

Règle de copie pour conclure un raisonnement sous-hypothèse.

$$\frac{\phi}{\phi}\, copie$$

Considérons le séquent :

$$\vdash p \rightarrow (q \rightarrow p)$$

Et la preuve suivante :

1	p	hyp.
2	q	hyp.
3	p	$copie 1$, fin hyp.2
4	$q \rightarrow p$	$\rightarrow_i$ 2, 3, fin hyp.1
5	$p \rightarrow (q \rightarrow p)$	$\rightarrow_i$ $1-4$

La logique propositionnelle
Déduction naturelle (XXI)

Règles pour la négation.

Les *contradictions* sont des formules de la forme :

$$\neg\phi \wedge \phi \text{ ou } \phi \wedge \neg\phi$$

Notons qu'aucune valuation ne satisfait une contradiction. Donc on a $\neg\phi \wedge \phi \models \psi$ pour n'importe quel ψ. Donc si la méthode de déduction est complète, on devrait avoir : $\neg\phi \wedge \phi \vdash \psi$ pour n'importe quel ψ.

On peut également voir ce phénomène sous l'angle suivant : l'information contenue dans $\phi \wedge \psi$ est au moins le contenu de ϕ et de ψ ensemble, alors que l'information contenue dans $\phi \vee \psi$ a au plus l'information contenue dans ϕ et au plus l'information contenue dans ϕ, et si ϕ a au moins autant d'information que ψ alors on devrait pouvoir dériver $\phi \vdash \psi$. Mais quelle est la quantitée

d'information détenue dans $\psi \wedge \neg\psi$. Intuitivement il y a surcharge d'information, ce qui nous permet de dériver toutes les formules à partir d'une contradiction.

Vu que toutes les contradictions sont équivalentes, on utilisera $\perp$ pour représenter une contradiction.

Le fait que l'on peut tout déduire à partir d'une contradiction est formalisé par la règle suivante :

$$\frac{\bot}{\phi}\bot_e$$

Le fait que $\bot$ represente une contradiction est formalisé par la règle suivante :

$$\frac{\phi \qquad \neg\phi}{\bot}\neg_e$$

Comment introduire une négation ?

Supposons que l'on fasse une hypothèse et que l'on arrive a déduire une contradiction, dans ce cas l'hypothèse est fausse. Ceci est formalisé par la règle de preuve suivante :

$$\dfrac{\begin{array}{l}\phi \quad \text{hyp.}\\ \vdots \\ \bot \quad \text{fin hyp.}\end{array}}{\neg\phi}\neg_i$$

La logique propositionnelle
Déduction naturelle (XXI)

On est maintenant en mesure de prouver le sequent de l'exemple introductif (taxi-train-invité) :

$$p \wedge \neg q \rightarrow r, \neg r, p \vdash q$$

1	$p \wedge \neg q \rightarrow r$	prémisse
2	$\neg r$	prémisse
3	p	prémisse
4	$\neg q$	hyp.
5	$p \wedge \neg q$	$\wedge_i 3, 4$
6	r	$\rightarrow_e 1, 5$
7	$\perp$	$\neg_e 6, 2,$ fin hyp.4
8	$\neg\neg q$	$\neg_i 4 - 7$
9	q	$\neg\neg_e 8$

Règles pour l'équivalence

$$\frac{\phi_1 \rightarrow \phi_2 \wedge \phi_2 \rightarrow \phi_1}{\phi_1 \leftrightarrow \phi_2}(\leftrightarrow_i)$$

$$\frac{\phi_1 \leftrightarrow \phi 2}{\phi_1 \rightarrow \phi_2}(\leftrightarrow_{e_1})$$

$$\frac{\phi_1 \leftrightarrow \phi 2}{\phi_2 \rightarrow \phi_1}(\leftrightarrow_{e_1})$$

La logique propositionnelle
Déduction naturelle (XXII)

Règles dérivées :

Notons que MT

$$\frac{\phi \rightarrow \psi \quad \neg\psi}{\neg\phi} MT$$

est une règle dérivée. En effet :

1	$\phi \rightarrow \psi$	prémisse
2	$\neg\psi$	prémisse
3	ϕ	hyp.
4	ψ	$\rightarrow_e 1, 3$
5	$\bot$	$\neg_e 4, 2,$ fin hyp.3
6	$\neg\phi$	$\neg_i 3 - 5$

La logique propositionnelle
Déduction naturelle (XXIII)

Autres règles dérivées :

$$\begin{array}{ll} \neg\phi & \text{hyp.} \\ . & \\ . & \\ . & \\ \bot & \text{fin hyp.} \end{array}$$
$$\frac{}{\phi} RAA$$

RAA=reductio ad absurdum

$$\frac{\phi}{\neg\neg\phi}\neg\neg_i$$

$$\frac{}{\phi\vee\neg\phi} LEM$$

La logique propositionnelle
Déduction naturelle (XXIII)

A la fin du cours, nous prouverons que les règles de la déduction naturelle forme une méthode de preuves adéquate et complète.

La logique propositionnelle
Les formes normales (I)

Une formule est en *forme normale conjonctive* (FNC) ssi c'est une conjonction de disjonctions de littéraux, c'est-à-dire une formule de la forme :

$$\bigwedge_i (\bigvee_j (\neg) p_{i,j})$$

Une formule est en *forme normale disjonctive* (FND) ssi c'est une disjonction de conjonctions de littéraux, c'est-à-dire une formule de la forme :

$$\bigvee_i (\bigwedge_j (\neg) p_{i,j})$$

La logique propositionnelle
Les formes normales (II)

Exemples :

$$p \wedge (q \vee \neg r) \wedge (\neg p \vee r)$$

est une formule en forme normale conjonctive. Tandis que

$$p \vee (q \wedge \neg r) \vee (\neg p \wedge r)$$

est une formule en forme normale disjonctive.

La logique propositionnelle
Les formes normales (III)

Notons $Prop(\phi)$ l'ensemble des propositions qui apparaîssent dans la formule ϕ.

Fonction associée à une formule :

$$F_\phi : Valuation \rightarrow \{0,1\}$$

ou encore

$$F_\phi : \{0,1\}^{Prop(\phi)} \rightarrow \{0,1\}$$

F_ϕ assigne à chaque valuation possible pour les propositions de ϕ la valeur 0 ou 1. On définit F_ϕ de la façon suivante :

$$F_\phi(V) = [\![\phi]\!]_V$$

La logique propositionnelle
Les formes normales (III)

Proposition : deux formules ϕ et ψ sont équivalentes ssi elles définissent la même fonction, c'est-à-dire $F_\phi = F_\psi$.

(Preuve laissée comme exercice)

Proposition : il existe au plus 2^{2^n} formules, deux à deux non équivalentes, du calcul propositionnel construit sur n propositions.

La logique propositionnelle
Les formes normales (IV)

Théorème : Toute fonction F de $\{0,1\}^n$ dans $\{0,1\}$ peut être représentée par une formule propositionnelle à n variables.

Preuve. Preuve par induction sur le nombre n de propositions. (i) $n = 1$: les 4 fonctions possibles sont équivalentes à :
$p, \neg p, \neg p \vee p, \neg p \wedge p$. (ii) $n > 1$. Par **HI**, la propriété est vraie pour n-1. $P = \{p_1, ..., p_n\}$ et $F : \{0,1\}^P \rightarrow \{0,1\}$ peut être exprimée comme :

$$F = \begin{cases} F\mid_{p_n=1} & si\, p_n = 1 \\ F\mid_{p_n=0} & si\, p_n = 0 \end{cases}$$

Par **HI** on a $\phi_{F|_{p_n=1}}$ et $\phi_{F|_{p_n=0}}$ et on construit la formule recherchée de la façon suivante :

$$\phi_F = \left(p_n \rightarrow \phi_{F|_{p_n=1}}\right) \wedge \left(\neg p_n \rightarrow \phi_{F|_{p_n=0}}\right)$$

CQFD

Théorème : Toute formule de la logique propositionnelle est équivalente à une forme normale disjonctive et à une forme normale conjonctive.

Preuve. On raisonne par induction sur le nombre de propositions :
(i) $|Prop(\phi)| = |\{p\}| = 1$
4 fonctions possibles exprimées par les formules suivantes :
$p, \neg p, \neg p \vee p, \neg p \wedge p$, formules qui sont à la fois FNC, FND.

(ii) $|Prop(\phi)| = |\{p_1, ..., p_n\}| = n > 1$
Par **HI** le théorème est vrai pour n-1.

$$\begin{aligned} F_\phi &= (\neg p_n \rightarrow \psi_1) \wedge (p_n \rightarrow \psi_2) \\ &= (\neg p_n \wedge \psi_1) \vee (p_n \wedge \psi_2) \end{aligned}$$

où

$$\begin{aligned} \psi_1 &= \psi_{11} \vee ... \vee \psi_{1k} \\ \psi_2 &= \psi_{21} \vee ... \vee \psi_{2l} \end{aligned}$$

de plus $\neg p_n \wedge \psi_1 = (\neg p_n \wedge \psi_{11}) \vee ... \vee (\neg p_n \wedge \psi_{1k})$ est en FND, et $p_n \wedge \psi_2 = (p_n \wedge \psi_{21}) \vee ... \vee (p_n \wedge \psi_{2l})$ est en FND.

Pour la FNC, on utilise l'équivalence :

$$\begin{aligned} F_\phi &= (\neg p_n \rightarrow \psi_1) \wedge (p_n \rightarrow \psi_2) \\ &= (p_n \vee \psi_1) \wedge (\neg p_n \vee \psi_2) \end{aligned}$$

CQFD

La logique propositionnelle
Les formes normales (V)

En pratique, on utilise les transformations successives suivantes pour obtenir les formes normales :

1. élimination des connecteurs $\rightarrow$ et $\leftrightarrow$ grâce aux équivalences suivantes :

$$(\phi \rightarrow \psi) \equiv (\neg\phi \vee \psi)$$
$$(\phi \leftrightarrow \psi) \equiv (\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi)$$

2. entrer les négations le plus à l'intérieur possible :

$$\neg(\phi \wedge \psi) \equiv (\neg\phi \vee \neg\psi)$$
$$\neg(\phi \vee \psi) \equiv (\neg\phi \wedge \neg\psi)$$

3. utilisation des distributivité de $\wedge$ et $\vee$ l'un par rapport à l'autre :

$$(\phi \wedge (\psi \vee \chi)) \equiv (\phi \wedge \psi) \vee (\phi \wedge \chi)$$
$$(\phi \vee (\psi \wedge \chi)) \equiv (\phi \vee \psi) \wedge (\phi \vee \chi)$$

Exercices :

– appliquer ces transformations à :

$$\neg(p \leftrightarrow (q \rightarrow r))$$

– prouver l'adéquation des équivalences ;

– prouver la complétude des règles de transformation.

La logique propositionnelle
Résolution (I)

Rappel : Une formule propositionnelle ϕ est un *littéral* si et seulement si ϕ est une proposition ou la négation d'une proposition.

Donc, p et $\neg p$ sont des littéraux.

Une formule propositionnelle ϕ est une *clause* si $\phi = \psi_1 \vee \psi_2 \vee \ldots \vee \psi_n$ où $n \geq 1$ et chaque ψ_i, $1 \leq i \leq n$, est un *littéral* .

Par exemple, $p \vee \neg q \vee r$ est une clause.

La logique propositionnelle
Résolution (II)

Une clause

$$\neg a_1 \vee \neg a_2 \vee \ldots \vee \neg a_m \vee b_1 \vee b_2 \vee \ldots \vee b_n$$

où $a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n$ sont des propositions peut être écrite comme l'implication :

$$a_1 \wedge a_2 \wedge \ldots \wedge a_m \rightarrow b_1 \vee b_2 \vee \ldots \vee b_n$$

ou encore comme une paire d'ensembles (Γ, Δ) où $\Gamma = \{a_1, a_2, \ldots, a_m\}$ et $\Delta = \{b_1, b_2, \ldots, b_n\}$. Donc Γ contient l'ensemble des propositions qui apparaissent négativement dans la clause, alors que Δ contient l'ensemble des propositions qui apparaissent positivement dans la clause.

Proposition : Pour toute formule ϕ, il existe un ensemble fini de clauses equivalentes.

(preuve laissée comme exercice. Indice : utiliser la résultat sur les formes normales.)

La logique propositionnelle
Résolution (IV)

Une clause (Γ, Δ) est

- une *clause négative* si $\Delta = \emptyset$;
- une *clause positive* si $\Gamma = \emptyset$;
- une *clause vide* si $\Gamma = \Delta = \emptyset$;
- une *clause tautologique* si $\Gamma \cap \Delta \neq \emptyset$.

La logique propositionnelle
Résolution (V)

La fonction d'interprétation V satisfait la clause $C = (\Gamma, \Delta)$ si et seulement si il existe $a \in \Gamma$ tel que $V(a) =$ faux ou si il existe $b \in \Delta$ tel que $V(b) =$ vrai. Formellement $[\![(\Gamma, \Delta)]\!]_V =$ vrai ssi $\exists a \in \Gamma : V(a) =$ faux ou $\exists a \in \Delta : V(a) =$ vrai.

Et donc,

Une clause vide n'est satisfaite par aucune fonction d'interprétation.

Une clause tautologique est satisfaite par toutes les fonctions d'interprétation.

La logique propositionnelle
Résolution (VI)

Nous étendons maintenant les notions de satisfaction, satisfaisabilité, validité et conséquence logique aux *ensembles de clauses* :

Un ensemble de clauses S est satisfait par la fonction d'interprétation V si et seulement si on a que $V \models C$ pour toute clause $C \in S$.

Un ensemble de clauses S est satisfaisable si et seulement si il existe une fonction d'interprétation qui satisfait S.

Un ensemble de clauses S est valide si et seulement si toute fonction d'interprétation V satisfait S.

Une clause C est une conséquence logique d'un ensemble de clauses S, ce que l'on note $S \models C$ si et seulement si pour toute valuation V qui satisfait S, V satisfait C.

La logique propositionnelle
Résolution (VII)

Et donc,

L'ensemble vide de clause est valide et donc satisfaisable.

Soit S un ensemble de clauses et C une clause tautologique, $S \cup \{C\}$ est satisfaisable (respectivement valide) si et seulement si S est satisfaisable (respectivement valide).

Ce dernier corollaire nous indique que, lorsque l'on veut vérifier la satisfaisabilité ou la validité d'un ensemble de clauses, on peut toujours supprimer les clauses tautologiques.

La logique propositionnelle
Résolution (VIII) : Règle de coupure

Etant données deux clauses $C_1 = (\Gamma_1, \Delta_1)$ et $C_2 = (\Gamma_2, \Delta_2)$ et une proposition p telles que $p \in \Delta_1$ et $p \in \Gamma_2$, c'est-à-dire p apparaît positivement dans C_1 et négativement dans C_2, on obtient la clause $C_3 = (\Gamma_3, \Delta_3)$ par *coupure* avec p où :

– $\Gamma_3 = \Gamma_1 \cup (\Gamma_2 \setminus \{p\})$;

– $\Delta_3 = (\Delta_1 \setminus \{p\}) \cup \Delta_2$.

On note par $C_1, C_2 \vdash^c_p C_3$ le fait que C_3 est déductible de C_1, C_2 par coupure sur la proposition p.

La logique propositionnelle
Résolution (IX) : Règle de coupure (suite)

Théorème : Si $C_3 = (\Gamma_3, \Delta_3)$ est une clause obtenue par la règle de coupure à partir des clauses $C_1 = (\Gamma_1, \Delta_1)$ et $C_2 = (\Gamma_2, \Delta_2)$ alors C_3 est une conséquence logique de C_1 et C_2.

Preuve. Pour montrer que la règle de coupure est adéquate, nous devons montrer que

$$\text{si } C_1, C_2 \vdash_p^c C_3 \text{ alors } C_1, C_2 \models C_3.$$

Par définition de $\models$, cela revient à montrer que pour toute fonction d'interprétation V telle que $V \models C_1$ et $V \models C_2$, on a $V \models C_3$. Par contraposition, cela revient à montrer que si $V \not\models C_3$ alors soit $V \not\models C_1$ soit $V \not\models C_2$. Par définition de $\vdash_p^c$, nous savons que $\Gamma_3 = \Gamma_1 \cup (\Gamma_2 \setminus \{p\})$ et $\Delta_3 = (\Delta_1 \setminus \{p\}) \cup \Delta_2$. On sait également que si V ne satisfait pas C_3, alors

$$\forall q \in \Gamma_3 : V(q) = \text{vrai et } \forall q \in \Delta_3 : V(q) = \text{faux}$$

Par définition de Γ_3 et Δ_3, on a en particulier :

- $\forall q \in \Gamma_1 : V(q) = \text{vrai}$ (2) ;
- $\forall q \in \Gamma_2 \setminus \{p\} : V(q) = \text{vrai}$;
- $\forall q \in \Delta_1 \setminus \{p\} : V(q) = \text{faux}$;
- $\forall q \in \Delta_2 : V(q) = \text{faux}$ (1).

Il suffit maintenant de considérer les deux valeurs possibles pour $V(p)$:

- si $V(p) = \text{vrai}$ alors on a $\forall q \in \Gamma_2, V(q) = \text{vrai}$ et (1), donc on a $V \not\models C_2$;
- si $V(p) = \text{faux}$ alors on a $\forall q \in \Delta_1, V(q) = \text{faux}$ et (2), donc on a $V \not\models C_1$;

♣

La logique propositionnelle
Résolution (X)

Soit S un ensemble de clauses et C une clause. Une preuve de C par coupure à partir de S est une suite finie de clauses $C_1, C_2, \ldots, C_n$ où $C_n = C$ et pour tout i, $1 \leq i \leq n$, on a que :

– soit $C_i \in S$;

– ou il existe k, l, $1 \leq k < l < i$ et une proposition p tels que $C_k, C_l \vdash^c_p C_i$.

On note par $S \vdash^c C$ le fait que C est déductible de S par coupure.

La logique propositionnelle
Résolution (XI)

Théorème : Pour tout ensemble de clauses S et pour toute clause C, si $S \vdash^c C$ alors $S \models C$.

Preuve. La preuve est laissée en exercice. Indices : utiliser la validité de la règle de coupure et raisonner par induction sur la longueur de la preuve.

♣

La logique propositionnelle
Résolution (XII) : Complétude de la méthode de preuve par coupure

Une *réfutation* d'un ensemble S de clauses par coupure est une dérivation de la clause vide à partir de S.

Théorème : Si il existe un réfutation de S par coupure alors l'ensemble de clauses S est non satisfaisable.

Preuve. La preuve de cette propriété est laissée comme exercice.

♣

La logique propositionnelle
Résolution (XIII)

Comment peut-on se servir de la notion de réfutation pour vérifier qu'une formule ϕ est une conséquence logique d'un ensemble fini de formules $\psi_1, \psi_2, \ldots, \psi_n$? Il suffit de procéder de la façon suivante :

1. transformer la conjonction

   $$\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_n$$

   en un ensemble fini S de clauses (cette transformation est toujours possible) ;
2. de la même façon, transformer $\neg\phi$ en un ensemble S' de clauses ;
3. construire l'ensemble de toutes les clauses D déductibles par coupure de $S \cup S'$.

La logique propositionnelle
Résolution (XIV)

Si la clause vide appartient à l'ensemble D alors l'ensemble D n'est pas satisfaisable et donc l'ensemble $S \cup S'$ ne l'est pas non plus, ce qui revient à dire que la formule $\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_n \wedge \neg\phi$ n'est pas satisfaisable et donc ϕ est une conséquence logique de l'ensemble $\{\psi_1, \psi_2, \ldots, \psi_n\}$.

La logique propositionnelle
Résolution (XV)

Théorème : Si un ensemble S de clauses est non satisfaisable et ne contient pas la clause vide, alors il existe une proposition p et deux clauses $C_1, C_2 \in S$ telles que $p \in \Delta_2$ et $p \in \Gamma_1$.

Preuve. Nous établissons ce lemme en raisonnant par l'absurde. Faisons l'hypothèse que S est non satisfaisable, ne contient pas la clause vide et ne contient pas une proposition p avec la propriété requise. Par conséquent, on sait que toute proposition p apparaissant dans S a la propri été suivante :
– pour toute clause $C_i \in S$, $p \in \Gamma_i$ ou $p \notin \Gamma_i \cup \Delta_i$;
– ou pour toute $C_i \in S$, $p \in \Delta_i$ ou $p \notin \Gamma_i \cup \Delta_i$;
Autrement dit, p apparaît soit toujours négativement ou toujours positivement. Notons que c'est exactement la négation de la propriété que l'on veut établir. Montrons maintenant que l'on peut établir la satisfaisabilité de S, ce qui constitue une contradiction.

Pour satisfaire S, il suffit de prendre la fonction d'interprétation V tel que $V(p) =$ vrai si p apparaît toujours positivement dans S et $V(p) =$ faux si p apparaît toujours négativement dans S.

Une fois que V est définie pour chaque proposition comme expliqué ci-dessus et vu que S ne contient pas la clause vide, il est facile d'établir que $V \models S$, ce qui est une contradiction avec l'hypothèse de départ qui dit que S est non satisfaisable.

La logique propositionnelle
Résolution (XVI)

Les propositions d'une clause $C = (\Gamma, \Delta)$ sont l'ensemble des propositions qui apparaissent dans l'ensemble $\Gamma \cup \Delta$; cet ensemble est noté $\mathsf{Prop}(C)$. Les propositions d'un ensemble S de clauses est l'ensemble $\bigcup_{C \in S} \mathsf{Prop}(C)$; cet ensemble est noté $\mathsf{Prop}(S)$.

Etant donné un ensemble S de clauses et une proposition p, on note S_p le sous-ensemble de clauses de S qui contiennent la proposition p, c'est-à-dire, $S_p = \{C \in S \mid p \in \mathsf{Prop}(C)\}$.

La logique propositionnelle
Résolution (XVII)

Soit S un ensemble de clauses qui ne contient pas de clauses tautologiques et une proposition p telle qu'il existe C_1, C_2 et $p \in \Gamma_1$ et $p \in \Delta_2$, donc p apparaît négativement dans C_1 et positivement dans C_2. On définit le résolvant de S par p comme l'ensemble des clauses

$$\{C_3 \mid \exists C_1, C_2 \in S_p, p \in \Gamma_1, p \in \Delta_2 \text{ et } C_1, C_2 \vdash^c_p C_3\}$$

c'est-à-dire l'ensemble des clauses que l'on peut obtenir à partir de deux clauses de S par coupure sur p ; nous notons par $\mathsf{Res}(S, p)$ cet ensemble de clauses et par $\mathsf{RRes}(S, p)$ le sous-ensemble de clauses non tautologiques de $\mathsf{Res}(S, p)$.

La logique propositionnelle
Résolution (XVII)

Etant donné un ensemble S de clauses et une proposition p, l'ensemble de clauses $S' = S \setminus \{S_p\} \cup \mathsf{RRes}(S, p)$ est l'ensemble obtenu par un pas de résolution sur p à partir de S.

La logique propositionnelle
Résolution (XVIII)

Proposition : Si S' est un ensemble de clauses obtenu à partir de S (qui ne contient pas de clauses tautologiques) par un pas de résolution sur p alors :

1. S' ne contient pas de clauses tautologiques ;
2. S' est satisfaisable si et seulement si S est satisfaisable ;
3. $\mathsf{Prop}(S') = \mathsf{Prop}(S) \setminus \{p\}$.

(preuve laissée comme exercice)

La logique propositionnelle
Résolution (XVIII)

Proposition : Si un ensemble S de clauses est non satisfaisable, alors on peut déduire la clause vide à partir de S avec un nombre fini de pas de résolution.

(preuve laissée comme exercice)

Et donc,

Pour tout ensemble de clauses S et pour toute clause C tel que $S \models C$ alors on a $S \vdash^c C$.

La logique du premier ordre
Introduction (I)

Les situations suivantes ne sont pas “modélisables” en logique propositionnelle :

- certains étudiants assistent à tous les cours ;
- aucun étudiant n’assiste à un cours non intéressant ;
- dans toute salle d’examen, il y a un étudiant qui, s’il échoue, alors tout le monde échoue.

La logique du premier ordre
Introduction (II)

Ce qui manque à la logique propositionnelle, c'est de pouvoir parler d'objets, de propriétés d'objets et de mettre en relation des objets.
Par exemple :

- tel cours est intéressant ;
- tel étudiant assiste au cours de logique.

La logique du premier ordre
Introduction (III)

La logique propositionnelle ne permet pas de quantifier sur les objects. On ne peut pas par exemple exprimer :

– tous les cours sont intéressants ;

– il existe des cours intéressants.

La logique du premier ordre nous permettra de modéliser ces situations. Par exemple, l'énoncé "tous les cours sont intéressants" sera modélisé par :

$$\forall x : \mathsf{Cours}(x) \rightarrow \mathsf{Interessant}(x)$$

La logique du premier ordre
Plan

1. Langages du premier ordre
   - construction des termes
   - construction des formules
   - variables libres, variables liées
2. Sémantique
   - structures et langages
   - structures et satisfaction des formules
   - formules valides, formules équivalentes
   - substitution
3. Formes prénexes et formes de Skolem
4. Déduction naturelle pour le premier ordre
5. Procédure de résolution pour le premier ordre
   - mise sous forme de clauses
   - unification
   - méthode de résolution
   - modèles de Herbrand

Logique du premier ordre
Langages du premier ordre (I)

Pourquoi “langages” du premier ordre ?

– car ils partagent des caractéristiques avec beaucoup de *langages* comme les langages de programmation, les langues naturelles, les langages d'interrogation de bases de données, ...
– il existe une grande variété de langages du premier ordre, chacun étant déterminé par son vocabulaire ;
– ces langages permettent de représenter des relations mais également des fonctions.

Logique du premier ordre
Langages du premier ordre (II)

L'expression "premier ordre" différencie ces langages des langages "d'ordre supérieur", dans lesquels il est possible de quantifier sur d'autres objects que les variables. (par ex. sur les fonctions, prédicats, ...)

Logique du premier ordre
Langages du premier ordre (III) : alphabet

L'alphabet d'un langage du premier ordre comporte d'abord les symboles suivants qui sont *communs à tous ces langages* :

- les connecteurs $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$;
- les parenthèses $(,)$;
- le *quantificateurs universel* $\forall$ et le *quantificateur existentiel* $\exists$;
- un ensemble infini V de symboles de *variables* $x, y, z, ...$

Logique du premier ordre
Langages du premier ordre (IV) : alphabet

Un *langage* $\mathcal{L}$ de la logique du premier ordre est caractérisé par :

- des *symboles de relations* ou *prédicats* , $p, q, r, s...$;
- des *symboles de fonctions* , $f, g, h, ...$;
- des *symboles de constantes* , $c, d, e, ...$

A chaque prédicat p, respectivement fonction f, on associe un entier strictement positif appelé l'*arité* de p, respectivement de f, c'est-à-dire le nombre d'arguments de p, respectivement f.

On utilise le prédicat "=" pour dénoter l'égalité. Si "=" fait partie du vocabulaire du langage $\mathcal{L}$, on dit que $\mathcal{L}$ est *égalitaire* .

Logique du premier ordre
Langages du premier ordre (V) : alphabet

Exemples de langages :

– $\mathcal{L}_1$ contient un prédicat unaire r et une constante c ;

– $\mathcal{L}_2$ contient un prédicat binaire r, une fonction unaire f, deux symboles de fonctions binaires g et h, et deux constantes c et d.

Logique du premier ordre
Langages du premier ordre (VI) : construction des termes

L'ensemble des *termes d'un langage* $\mathcal{L}$ est le plus petit ensemble qui contient les symboles de constantes et de variables et qui est clos par application des fonctions.

L'ensemble des termes, noté $\mathcal{T}$, est le plus petit ensemble satisfaisant :

1. tout symbole de constante ou variable est un terme ;
2. si f est un symbole de fonction d'arité n et $t_1, t_2, \ldots, t_n$ sont des termes alors $f(t_1, t_2, \ldots, t_n)$ est un terme.

Remarque : les prédicats ne fournissent pas des termes ; ils serviront pour construire les formules.

Logique du premier ordre

Langages du premier ordre (VII) : construction des termes

Exemples :

- les seuls termes du langage $\mathcal{L}_1$ sont la constante c et les variables ;
- les expressions suivantes sont des termes du langage $\mathcal{L}_2$:
  - $f(c)$;
  - $f(h(f(c), d))$;
  - $f(y)$;
  - $f(h(f(x), f(d)))$.

Un terme est *clos* s'il est sans variable. Par exemple $f(c)$ est clos.

Logique du premier ordre
Langages du premier ordre (VIII) : construction des formules

L'ensemble des *formules atomiques* d'un langage $\mathcal{L}$ est l'ensemble des formules de la forme :

- $p(t_1, t_2, \ldots, t_n)$ ou p est un prédicat d'arité n et $t_1, t_2, \ldots, t_n$ sont des termes du langage $\mathcal{L}$;
- $t_1 = t_2$ si $\mathcal{L}$ est égalitaire et t_1, t_2 sont des termes du langage $\mathcal{L}$.

Logique du premier ordre
Langages du premier ordre (IX) : construction des formules

L'ensemble des *formules du langage* $\mathcal{L}$, que l'on désigne par $\mathcal{F}(\mathcal{L})$, est le plus petit ensemble qui satisfait :

1. toute formule atomique est une formule ;
2. si ϕ est une formule alors $\neg\phi$ est une formule ;
3. si ϕ et ψ sont des formules alors $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$ sont des formules ;
4. si ϕ est une formule et x une variable, alors $\forall x \cdot \phi$ et $\exists x \cdot \phi$ sont des formules.

Logique du premier ordre

Langages du premier ordre (X) : exemples de formules

La formule $r(c) \vee \neg\exists x \cdot r(x)$ est une formule du langage $\mathcal{L}_1$.

Exemples de formules du langage $\mathcal{L}_2$:

– $\forall x \cdot \exists y(g(x, y) = c \wedge g(y, x) = c)$

– $\forall x \cdot \neg(f(x) = c)$

Logique du premier ordre
Langages du premier ordre (XI) : décomposition d'une formule

Toute formule d'un langage du premier ordre se décompose de manière unique sous l'une, et une seule, des formes suivantes :
– une formule atomique,
– $\neg\phi$, où ϕ est une formule,
– $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, $\phi \leftrightarrow \psi$ où ϕ et ψ sont des formules,
– $\forall x \cdot \phi$ ou $\exists x \cdot \phi$ où ϕ est une formule et x est une variable.
Une formule ϕ est une *sous-formule* de ψ si ϕ apparaît dans la décomposition de ψ.

A nouveau, les parenthèses utilisées pour contrecarrer les règles de précédence ou rendre les formules plus lisibles seront omises dans les définitions (mais utilisées dans les exemples).

Logique du premier ordre
Langages du premier ordre (XII) : variables libres et liées

Une *occurrence d'une variable* dans une formule est un couple constitué de cette variable et d'une place effective, c'est-à-dire qui ne suit pas un quantificateur.

Par exemple, dans la formule

$$r(x, z) \rightarrow \forall z \cdot (r(y, z) \vee y = z)$$

la variable x possède une occurrence, la variable y deux et la variable z trois.

Logique du premier ordre

Langages du premier ordre (XIII) : variables libres et liées

Une occurrence d'une variable x dans une formule ϕ est une *occurrence libre* si elle ne se trouve dans aucune sous-formule de ϕ, qui commence par une quantification $\forall x$ ou $\exists x$. Dans le cas contraire, l'occurrence est dite *liée*.

Une variable est *libre* dans une formule si elle a au moins une occurrence libre dans cette formule.

Une *formule close* est une formule sans variable libre.

Logique du premier ordre : Sémantique
Structures et langages (I)

Une structure $\mathcal{M}$ pour un langage $\mathcal{L}$ se compose d'un ensemble non vide M, appelé le *domaine* et :

- d'un sous-ensemble de M^n, noté $r^{\mathcal{M}}$, pour chaque symbole de prédicat r d'arité n dans $\mathcal{L}$;
- d'une fonction de M^m dans M, notée $f^{\mathcal{M}}$, pour chaque symbole de fonction f d'arité m dans $\mathcal{L}$;
- d'un élément de M, noté $c^{\mathcal{M}}$, pour chaque symbole de constante c dans $\mathcal{L}$.

Logique du premier ordre : Sémantique
Structures et langages (II)

Un ensemble D muni d'une relation unaire E et d'un élément distingué a est une structure pour le langage $\mathcal{L}_1 = \{r, c\}$, notée (D, E, a).

On peut également donner un exemple plus concret d'interprétation pour le langage $\mathcal{L}_1 = \{r, c\}$ en prenant l'ensemble des nombres naturels comme domaine d'interprétation, l'ensemble des nombres premiers pour interpréter le prédicat r et le nombre 2 pour interpréter la constante c.

L'ensemble des réels $\mathbb{R}$ permet de construire une structure pour $\mathcal{L}_2 = \{r, f, g, h, c, d\}$ de la façon suivante :

- on interprète le prédicat r comme l'ordre $\leq$ sur les réels ;
- on interprète f comme la fonction $+1$, g comme $+$, et h comme $\times$;
- on interprète les constantes c et d comme 0 et 1.

On note cette structure

$$\mathcal{M}_2 = (\mathbb{R}, \leq, +1, +, \times, 0, 1)$$

Logique du premier ordre : Sémantique
Structures et satisfaction des formules (I)

Etant donné un ensemble de variables $\mathcal{V}$ et un domaine M, une *valuation* pour les variables de $\mathcal{V}$ dans M est une fonction $v : \mathcal{V} \to M$ qui attribue à chaque variable $x \in \mathcal{V}$, une valeur $v(x) \in M$.

Etant données une structure $\mathcal{M}$ et une valuation v pour $\mathcal{V}$, on définit inductivement la valeur d'un terme t (dont les variables sont dans $\mathcal{V}$), que l'on note $t^{\mathcal{M},v}$, de la façon suivante :

- si t est une constante c, alors $t^{\mathcal{M},v}$ est $c^{\mathcal{M}}$;
- si t est la variable x alors $t^{\mathcal{M},v}$ est $v(x)$;
- si t est de la forme $f(t_1, t_2, \ldots, t_n)$ et si $t_i^{\mathcal{M},v}$ est b_i alors $t^{\mathcal{M},v} = f^{\mathcal{M}}(b_1, b_2, \ldots, b_n)$;

Logique du premier ordre : Sémantique
Structures et satisfaction des formules (II)

Soit le langage $\mathcal{L}_2 = \{r, f, g, h, c, d\}$ et $\mathcal{M}_3$ la structure $(\boldsymbol{N}, \leq, +1, +, \times, 0, 1)$, où $\boldsymbol{N}$ est l'ensemble des entiers positifs (nombres naturels). La valeur du terme $t_1 \equiv g(y, h(c, x))$ pour une valuation v telle que $v(x) = 3$, $v(y) = 4$, $v(z) = 6$ est :

$$t_1^{\mathcal{M}_3, v} = 4 + (0 \times 3) = 4$$

La valeur du terme $t_2 \equiv f(g(d, h(y, z)))$ est :

$$t_2^{\mathcal{M}_3, v} = (1 + (4 \times 6)) + 1 = 26$$

Logique du premier ordre : Sémantique
Structures et satisfaction des formules (III)

Une formule ϕ construite sur un langage $\mathcal{L}$ est satisfaite dans une structure $\mathcal{M}$ et pour une valuation v donnant une valeur aux variables de l'ensemble $\mathcal{V}$, noté $\mathcal{M}, v \models \phi$, si et seulement si :

- si $\mathcal{L}$ est égalitaire et $\phi \equiv t_1 = t_2$ alors ϕ est vrai ssi $t_1^{\mathcal{M},v} = t_2^{\mathcal{M},v}$;
- si $\phi \equiv r(t_1, t_2, \ldots, t_n)$ et $t_i^{\mathcal{M},v} = b_i$ pour $i = 1, 2, \ldots, n$, alors ϕ est vraie ssi $(b_1, b_2, \ldots, b_n) \in r^{\mathcal{M}}$;

Logique du premier ordre : Sémantique
Structures et satisfaction des formules (IV)

- si $\phi \equiv \neg\psi_1$, $\phi \equiv \psi_1 \vee \psi_2$, $\phi \equiv \psi_1 \wedge \psi_2$, $\phi \equiv \psi_1 \rightarrow \psi_2$, $\phi \equiv \psi_1 \leftrightarrow \psi_2$ alors la valeur de ϕ est calculée à partir des valeurs de ϕ_1 et ψ_2 comme dans le cas propositionnel ;
- si $\phi \equiv \exists x \cdot \psi$, alors ϕ est vraie ssi il existe une valuation v' et une valeur $u \in M$ telles que

$$\begin{cases} v'(y) = v(y) & \text{si } y \not\equiv x \\ v'(y) = u & \text{si } y \equiv x \end{cases}$$

et ψ est vraie dans $(\mathcal{M}, v')$, c'est-à-dire $\mathcal{M}, v' \models \psi$;

Logique du premier ordre : Sémantique
Structures et satisfaction des formules (V)

– si $\phi \equiv \forall x \cdot \psi$, alors ϕ est vraie ssi pour toute valuation v' et valeur $u \in M$ telles que

$$\begin{cases} v'(y) = v(y) & \text{si } y \not\equiv x \\ v'(y) = u & \text{si } y \equiv x \end{cases}$$

ψ est vraie dans $(\mathcal{M}, v')$;

La *valeur* d'une formule ϕ dans une structure $\mathcal{M}$ est l'ensemble $\mathcal{W}$ des valuations telles que pour toute valuation $v \in \mathcal{W}$, $\mathcal{M}, v \models \phi$.

Logique du premier ordre : Sémantique
Structures et satisfaction des formules (VI)

Notons que, si ϕ est une formule close, alors sa valeur de vérité dans un couple $(\mathcal{M}, v)$, ne dépend pas de v.

Dans le cas où une formule close ϕ est vraie dans une structure $\mathcal{M}$, ce que l'on note,

$$\mathcal{M} \models \phi$$

On dit que $\mathcal{M}$ est un *modèle* pour ϕ.

Logique du premier ordre : Sémantique
Structures et satisfaction des formules (VII)

Quelques exemples :

– La structure (D, R, a) du langage $\mathcal{L}_3 = \{r, c\}$ est un modèle de la formule

$$\forall x \cdot \forall y \cdot (r(x, y) \rightarrow r(y, x))$$

ssi la relation R est symétrique.

– La formule close suivante

$$\begin{aligned} &\forall x \cdot r(x, x) \\ \wedge &\forall x \cdot \forall y \cdot (r(x, y) \rightarrow r(y, x)) \\ \wedge &\forall x \cdot \forall y \cdot \forall z \cdot (r(x, y) \wedge r(y, z) \rightarrow r(x, z)) \end{aligned}$$

exprime que si une structure (D, R, a) est un modèle de la formule, alors R est une relation d'équivalence.

– la formule $\forall y \cdot r(x, y)$ est vraie dans la structure $(\mathbb{N}, \leq, +1, +, \times, 0, 1)$ et la valuation v ssi $v(x) = 0$.
– la formule $\exists x \cdot \forall y \cdot r(x, y)$ est vraie dans la structure $(\mathbb{N}, \leq, +1, +, \times, 0, 1)$;
– la formule close

$$\forall x \cdot \forall z \cdot \exists y \cdot (x = c \vee g(h(x, y), z) = c)$$

du langage $\mathcal{L}_2$ est vraie dans la structure $(\mathbb{R}, \leq, +1, +, \times, 0, 1)$ et fausse dans la structure $(\mathbb{N}, \leq, +1, +, \times, 0, 1)$.

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (I)

La relation d'équivalence sur les formules permet de les classifier et de regrouper celles qui possèdent la même signification. L'une de ces classes de formules est particulièrement intéressante : les formules vraies dans toute structure. On appellera ces formules, les *formules valides*.

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (II)

Soit $\phi(x_1, x_2, \ldots, x_n)$ une formule du premier ordre avec les variables libres $x_1, x_2, \ldots, x_n$. On appelle *clôture universelle* de ϕ la formule

$$\forall x_1 \cdot \forall x_2 \cdot \ldots \forall x_n \cdot \phi(x_1, x_2, \ldots, x_n)$$

Soit $\mathcal{L}$ un langage du premier ordre, on définit les notions suivantes :

– une structure $\mathcal{M}$ *satisfait* une formule

$$\phi(x_1, x_2, \ldots, x_n)$$

si elle satisfait la clôture universelle de ϕ ;

– une formule ϕ est *valide* si elle est satisfaite par toutes les structures du langage $\mathcal{L}$;

– deux formules ϕ et ψ sont *équivalentes* si, pour toute structure $\mathcal{M}$ et pour toute valuation v, on a $(\mathcal{M}, v) \models \phi$ ssi $(\mathcal{M}, v) \models \psi$.

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (III)

Théorème : deux formules ϕ et ψ sont equivalentes si et seulement si la formule $\phi \leftrightarrow \psi$ est valide.

Théorème : si, dans une formule, on remplace une sous-formule par une formule équivalente, alors on obtient une formule équivalente à la formule de départ.

(Preuves à faire comme exercices).

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (IV)

Exercices : démontrer que

- si ϕ et $\phi \rightarrow \psi$ sont des formules valides, alors ψ est une formule valide ;
- la formule $(\exists x \cdot \forall y \cdot \phi(x, y)) \rightarrow (\forall y \cdot \exists x \cdot \phi(x, y))$ est valide ;
- la formule $(\forall y \cdot \exists x \phi(x, y)) \rightarrow (\exists x \cdot \forall y \cdot \phi(x, y))$ n'est pas valide.

Montrons que la formule

$$\exists x.\forall y.\phi(x,y) \rightarrow \forall y.\exists x.\phi(x,y)$$

est valide

Rappel : soient

$$\begin{aligned} \Psi &\equiv \Psi_1 \rightarrow \Psi_2 \\ \Psi_1 &\equiv \exists x.\forall y.\phi(x,y) \\ \Psi_2 &\equiv \forall y.\exists x.\phi(x,y) \end{aligned}$$

Ψ valide ssi pour toute structure μ : $\mu \models \Psi$

Ψ valide ssi pour toute structure μ : si $\mu \models \Psi_1$ alors $\mu \models \Psi_2$

Ψ valide ssi pour toute structure μ : soit $\mu \not\models \Psi_1$, ou $\mu \models \Psi_1$ et

$\mu \models \Psi_2$

or si $\mu \models \Psi_1$ alors pour toute valuation v,

$$\exists u \in D^{\mu} : \mu, v_{[x \mapsto u]} \models \forall y.\phi(x, y)$$

si $\mu \models \Psi_1$ alors pour toute valuation v,

$$\exists u \in D^{\mu} \forall w \in D^{\mu} : \mu, v_{[x \mapsto \mu, y \mapsto w]} \models \phi(x, y)$$

On a donc que $\forall w \in D^{\mu} : \mu, v_{[x \mapsto u, y \mapsto w]} \models \phi(x, y)$. Donc on sait que $\phi(x, y)$ est vérifié pour n'importe quelle valeur attribuée à y si on attribue à x la valeur u. Il suffit donc de choisir cette valeur lors de l'évaluation de la quantification existentielle dans la formule

$$\forall y \exists x \cdot \phi(x, y)$$

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (V)

Notion de *substitution* : étant donné un terme t et une variable x apparaissant dans ce terme, on peut remplacer toutes les occurrences de x par un autre terme t'. Le nouveau terme est dit obtenu par *substitution* de t' à x et est noté $t(t'/x)$.

Exemple :

- le résultat de la substitution de $f(y, z)$ à x dans le terme $h(x, y)$ est le terme $h(f(y, z), y)$;
- le résultat de la substitution de $g(y, z)$ à x dans le terme $g(x, x)$ est $g(g(y, z), g(y, z))$.

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (VI)

Substitution dans les formules : avant d'effectuer la substitution d'un terme à une variable libre dans une formule, il est nécessaire de prendre quelques précautions. Sinon, la signification de la formule peut être complètement modifiée par un phénomène appelé *capture de variables* .

Exemple : soit $\phi(x)$ la formule $\exists y \cdot g(y, y) = x$. Dans la structure $\mathcal{M} = (\mathbb{N}, \leq, +1, +, \times, 0, 1)$, où g est interprété par l'addition, la signification de $\phi(x)$ est claire : $(\mathcal{M}, v) \models \phi(x)$ ssi $v(x)$ est pair. Si l'on remplace la variable x par z, cette signification est conservée. Par contre si l'on remplace la variable x par y, on obtient : $\exists y \cdot g(y, y) = y$ qui est une formule close qui est vraie dans $\mathcal{M}$ (en effet, il suffit de choisir la valeur 0 pour la variable y).

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (VII)

La substition d'un terme t à une variable libre x dans une formule ϕ est obtenue en remplaçant toutes les occurrences libres de cette variable par le terme t, sous réserve que la condition suivante soit vérifiée : pour chaque variable y apparaissant dans t, x n'a pas d'occurrence libre qui se trouve dans une sous-formule de ϕ commençant par une quantification $\forall y$ ou $\exists y$. Le résultat de cette substitution, si elle existe, est une formule notée $\phi(t/x)$.

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (VIII)

Exemple : le résultat de la substitution du terme $f(z)$ à la variable x dans la formule $\phi(x)$ suivante :

$$((r(c,x) \wedge \neg x = c) \wedge (\exists y g(y,y) = x))$$

est la formule

$$((r(c,f(z)) \wedge \neg f(z) = c) \wedge (\exists y g(y,y) = f(z)))$$

Notons que la substitution de y à x n'est pas possible dans cette même formule.

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes (IX)

Théorème : si ϕ est une formule, x est une variable libre dans ϕ et t un terme tel que la substitution de t à x dans ϕ soit définie, alors les formules

$$(\forall x \cdot \phi(x)) \rightarrow \phi(t/x)$$

et

$$\phi(t/x) \rightarrow (\exists x \cdot \phi(x))$$

sont valides.

Preuve. Etant donnée une structure μ, on montre par induction sur la formule ϕ que la satisfaction de la formule $\phi(t/x)$ dans (μ, v) est équivalente à celle de la formule $\phi(x)$ dans (μ, v_1) où v_1 est obtenu à partir de v en donnant à x la valeur $t^{\mu,v}$. Les seuls cas qui nécessitent une justification sont ceux où :

$$\phi \equiv \exists y.\Psi$$

D'après l'hypothèse de la substitution de t à x, la quantification porte sur une variable y distincte à la fois de x et de toutes variables de t.

Il suffit donc d'examiner la satisfaction de $\Psi(t/x)$ pour une interprétation v' égale à v sauf sur y. Compte tenu de l'hypothèse d'induction sur Ψ, la formule $\Psi(t/x)$ est satisfaite par v' ssi Ψ est satisfaite par $v_1^{'}$ où $v_1^{'}(x) = t^{\mu,v'}$, qui est égale à $t^{\mu,v}$ vu que v et v' sont égales pour toutes les variables qui apparaissent dans t (qui sont différentes de y).

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes(X)

Les couples de formules suivants sont des exemples de formules équivalentes :

- ϕ et $\forall x \cdot \phi$ si x n'est pas libre dans ϕ ;
- ϕ et $\exists x \cdot \phi$ si x n'est pas libre dans ϕ ;
- $\forall x \cdot (\phi \wedge \psi)$ et $(\forall x \cdot \phi) \wedge (\forall x \cdot \psi)$
- $\exists x \cdot (\phi \vee \psi)$ et $(\exists x \cdot \phi) \vee (\exists x \cdot \psi)$
- $\exists x \cdot (\phi \rightarrow \psi)$ et $\exists x \cdot (\neg\phi \vee \psi)$
- $\exists x \cdot \phi$ et $\exists y \cdot \phi(y/x)$ si x est libre dans ϕ et y n'apparaît pas dans ϕ
- $\forall x \cdot \phi$ et $\forall y \cdot \phi(y/x)$ si x est libre dans ϕ et y n'apparaît pas dans ϕ

Si x n'est pas libre dans ψ, on obtient les équivalences suivantes :

- $\forall x \cdot (\phi \wedge \psi)$ et $(\forall x \cdot \phi) \wedge \psi$
- $\exists x \cdot (\phi \wedge \psi)$ et $(\exists x \cdot \phi) \wedge \psi$

Logique du premier ordre : Sémantique
Formules valides, formules équivalentes(X)

Les équivalences suivantes nous permettent de faire passer en tête de formule tous les quantificateurs. Soit ϕ une formule, x une variable et ψ une formule dans laquelle x n'est pas libre :

- $\neg\forall x \cdot \phi$ et $\exists x \cdot \neg\phi$
- $\neg\exists x \cdot \phi$ et $\forall x \cdot \neg\phi$
- $(\forall x \cdot \phi) \vee \psi$ et $\forall x \cdot (\phi \vee \psi)$
- $(\exists x \cdot \phi) \wedge \psi$ et $\exists x \cdot (\phi \wedge \psi)$
- $(\psi \rightarrow \forall x \cdot \phi)$ et $\forall x \cdot (\psi \rightarrow \phi)$
- $(\psi \rightarrow \exists x \cdot \phi)$ et $\exists x \cdot (\psi \rightarrow \phi)$
- $(\forall x \cdot \phi) \rightarrow \psi$ et $\exists x \cdot (\phi \rightarrow \psi)$
- $(\exists x \cdot \phi) \rightarrow \psi$ et $\forall x \cdot (\phi \rightarrow \psi)$

Notons également que les formules suivantes sont équivalentes :

- $\forall x \cdot \forall y \cdot \phi$ et $\forall y \cdot \forall x \cdot \phi$
- $\exists x \cdot \exists y \cdot \phi$ et $\exists y \cdot \exists x \cdot \phi$

Logique du premier ordre : Sémantique
Formules prénexes , formes de Skolem (I)

Objectif : définir une forme standard pour les formules du premier ordre.

Une formule ϕ est *prénexe* si et seulement si elle est de la forme

$$Q_1 x_1 \cdot Q_2 x_2 \cdot \ldots Q_n \cdot x_n \cdot \phi$$

où chaque Q_i est un quantificateur (existentiel ou universel), tous les x_i sont différents et ϕ est une formule sans quantificateur.

Rappel : Formules équivalentes

1. $\exists x \cdot \phi$ et $\exists y \cdot \phi(y/x)$ (x libre dans ϕ et y n'apparaît pas dans ϕ)
2. $\forall x \cdot \phi$ et $\forall y \cdot \phi(y/x)$ (idem)
3. $\forall x \cdot (\phi \wedge \Psi)$ et $\forall x \cdot \phi \wedge \Psi$ (x non libre dans Ψ)
4. $\exists x \cdot (\phi \vee \Psi)$ et $\exists x \cdot \phi \vee \Psi$ (idem)
5. $\neg \forall x \cdot \phi$ et $\exists x \cdot \neg \phi$
6. $\neg \exists x \cdot \phi$ et $\forall x \cdot \neg \phi$
7. $(\forall x \cdot \phi) \vee \Psi$ et $\forall x \cdot (\phi \vee \Psi)$ (x non libre dans Ψ)
8. $(\exists x \cdot \phi) \wedge \Psi$ et $\exists x \cdot (\phi \wedge \Psi)$ (x non libre dans Ψ)

Théorème Pour toute formule de la logique du premier ordre, il existe une formule prénexe équivalente.

Preuve. Par induction sur la structure des formules.

CB : Si ϕ est une formule atomique. Evident.

CI :

- $\phi \equiv \neg\Psi$ et Ψ est supposée équivalente à une forme prénexe $Q_1x_1.Q_2x_2...Q_nx_n.\Psi'$. Après application de (5) ou (6) on obtient :

$$\phi \equiv Q_1'x_1.Q_2'x2...Q_n'x_n.\neg\Psi'$$

  où $Q_i' = \exists$ si $Q_i = \forall$ et $Q_i' = \forall$ si $Q_i = \exists$; cette formule est en forme prénexe.
- Si $\phi \equiv \forall x \cdot \Psi$ alors on sait qu'il existe $\Psi' \equiv \Psi$ en forme prénexe et donc $\phi \equiv \forall x \cdot \Psi'$ est en forme prénexe.
- Si $\phi \equiv \exists x \cdot \Psi$, justification similaire.

- $\phi \equiv \Psi_1 \wedge \Psi_2$. On sait par hypothèse d'induction qu'il existe 2 formules prénexes Ψ_1' et Ψ_2' telles que $\Psi_1 \equiv \Psi_1'$ et $\Psi_2 \equiv \Psi_2'$.
  Il y a plusieurs cas à considérer. Soit $\Psi_1' \equiv \forall x \cdot \Psi_1''$ ou $\Psi_1' \equiv \exists x \cdot \Psi_1''$. Si x n'est pas libre dans Ψ_2' on applique (3) ou (8) pour sortir la quantification. Si x est libre dans Ψ_2' on applique d'abord les propriétés (2) ou (1) (renommage) et on applique (3) ou (8) comme au dessus.
  Finalement ϕ est équivalente à $\forall x \cdot (...)$ ou $\exists x \cdot (...)$ avec 1 quantificateur en moins à l'intérieur. On itère ensuite cette transformation.
- $\phi \equiv \Psi_1 \vee \Psi_2$, idem mais avec (1), (2), (4), (7).
- $\phi \equiv \Psi_1 \rightarrow \Psi_2$
  on sait que $\Psi_1 \equiv \Psi_1'$ et $\Psi_2 \equiv \Psi_2'$ or $\phi \equiv \neg\Psi_1 \vee \Psi_2$. On utilise les règles pour $\neg$ et $\vee$.
- $\phi \equiv \Psi_1 \leftrightarrow \Psi_2$ est équivalent à $\phi \equiv (\Psi_1 \rightarrow \Psi_2) \wedge (\Psi_1 \leftarrow \Psi_2)$.

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (II)

Exercice : donner la formule prénexe équivalente à la formule

$$\begin{aligned}\forall x \cdot \forall y \cdot (\quad & (r(x,y) \wedge \neg x = y) \\ & \rightarrow \exists z (y = g(x, h(z,z))))\end{aligned}$$

Pour cela, on transforme successivement la formule de la façon suivante :

$$\begin{aligned}\forall x \cdot \forall y \cdot (\quad & (r(x,y) \wedge \neg x = y) \\ & \rightarrow \exists z (y = g(x, h(z,z))))\end{aligned}$$

$$\forall x \cdot \forall y \cdot ((\neg r(x,y) \vee x = y) \vee \exists z (y = g(x, h(z,z))))$$

$$\forall x \cdot \forall y \cdot \exists z \cdot ((\neg r(x,y) \vee x = y) \vee (y = g(x, h(z,z))))$$

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (III)

Nous adaptons les notions de littéral, clause et forme normale conjonctive/disjonctive pour la logique du premier ordre :

- un *littéral* est une formule atomique ou une négation d'une formule atomique ;
- une *clause* est une disjonction de littéraux ;
- une formule prénexe

$$Q_1 x_1 \cdot Q_2 x_2 \cdot \ldots Q_n x_n \cdot \phi$$

est sous *forme normale conjonctive*, respectivement *forme normale disjonctive*, si la formule sans quantificateur ϕ est une clause ou une conjonction de clauses, respectivement une disjonction de conjonctions de littéraux.

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (IV)

Par exemple, la formule

$$\forall x \cdot \forall y \cdot \exists z \cdot ((\neg r(x, y) \vee x = y) \wedge (y = g(x, h(z, z))))$$

est sous forme normale conjonctive.

Théorème : toute formule de la logique du premier ordre est équivalente à une formule prénexe sous forme normale conjonctive (respectivement disjonctive).

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (V)

Skolémisation : suppression des quantificateurs existentiels.

Note : cette suppression ne fournit pas une formule équivalente, mais préserve la **satisfaisabilité**.

Quelques définitions :

- une formule ϕ est dite *universelle* si elle est prénexe et si tous les quantificateurs qui apparaissent dans ϕ sont universels ;
- soit $\mathcal{L}$ un langage du premier ordre, un langage $\mathcal{L}'$ est une *extension de Skolem* de $\mathcal{L}$ si $\mathcal{L}'$ est obtenu en ajoutant à $\mathcal{L}$ une infinité de symboles de fonctions de chaque arité et une infinité de symboles de constantes.

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (VI)

Une formule prénexe close ϕ de $\mathcal{L}'$ est soit universelle, soit de la forme $\forall x_1 \cdot \ldots \forall x_k \cdot \exists x \cdot \psi$ où ψ est prénexe. Dans ce dernier cas, il se peut que $k = 0$ et ϕ est de la forme $\exists x \cdot \psi$. La transformation que l'on applique à ϕ si elle contient au moins un quantificateur existentiel, consiste à lui associer une formule

$$\forall x_1 \cdot \ldots \forall x_k \cdot \psi(f(x_1, \ldots, x_k)/x),$$

où f est un symbole de fonction n'apparaissant pas dans la formule ψ. Dans le cas particulier où $k = 0$, c'est-à-dire $\phi \equiv \exists x \cdot \psi$, on lui associe la formule $\psi(c/x)$ où c est une constante n'apparaissant pas dans ψ. La formule ϕ_1 ainsi obtenue possède un quantificateur existentiel en moins que ϕ. Cette transformation est appelée *pas de Skolémisation* .

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (VII)

Quand on écrit

$$\psi \equiv \forall x \cdot \exists y \cdot p(x, y)$$

en rapport avec une structure $\mathcal{M}$, on exprime que pour chaque valeur a donnée à x, on peut choisir une valeur b pour y telle que

$$(a, b) \in p^{\mathcal{M}}$$

Notons que l'on peut "approximer" cela en associant à chaque valeur pour x une valeur fixe $f(x)$ pour y, où f est une nouvelle fonction. Si on interpréte f comme

$$f^{\mathcal{M}'}(a) = b \text{ alors on aura } (a, f^{\mathcal{M}'}(a)) \in p^{\mathcal{M}}$$

où $\mathcal{M}'$ est une inteprétation qui est "compatible" avec $\mathcal{M}$ et qui interpréte la nouvelle fonction f.

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (VIII)

Nous avons parlé d'approximation, pourquoi ?

Considérons à nouveau la formule

$$\psi \equiv \forall x \cdot \exists y \cdot p(x, y)$$

et une interprétation $\mathcal{M}_1$ telle que

$$M^{\mathcal{M}_1} = \{1, 2\} \text{ et}$$
$$p^{\mathcal{M}_1} = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$$

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (IX)

et considérons

$$\psi' \equiv \forall x \cdot p(x, f(x))$$

avec f un nouveau symbole de fonction, et nous étendons $\mathcal{M}_1$ en $\mathcal{M}'_1$ pour interpréter f :

$$f^{\mathcal{M}'_1}(1) = 2, f^{\mathcal{M}'_1}(2) = 1$$

alors on a $\mathcal{M}'_1 \models \psi'$. Mais on considère $\mathcal{M}_2$ avec

$$p^{\mathcal{M}_2} = \{(1,1), (2,2)\}$$

et $\mathcal{M}'_2$ l'extension de $\mathcal{M}_2$ avec à nouveau :

$$f^{\mathcal{M}'_2}(1) = 2, f^{\mathcal{M}'_2}(2) = 1$$

alors on a

$$\mathcal{M}_2 \models \psi \text{ mais } \mathcal{M}'_2 \not\models \psi'$$

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (X)

Intuitivement, en remplaçant la quantification existentielle par une fonction, on se limite à une dépendance fonctionnelle entre les variables x et y alors qu'en général on a des dépendances relationnelles entre variables (donc on limite le choix possible des modèles d'une formule).

On va maintenant formaliser cette notion de suppression des quantificateurs existentiels, appelée *Skolémisation* , et montrer que cette transformation préserve la satisfaisabilité mais pas la validité des formules.

Logique du premier ordre : Sémantique
Formules prénexes, formes de Skolem (XI)

Soit ϕ une formule close en forme prénexe du langage $\mathcal{L}'$, possédant n quantificateurs existentiels :

- une *forme de Skolem* de ϕ est une formule obtenue en appliquant n fois successivement le pas de skolémisation ;
- les nouvelles fonctions et constantes introduites au cours de ces transformations s'appellent les *fonctions* et *constantes de Skolem* .

Théorème : le pas de skolémisation préserve la satisfaisabilité des formules : soit ϕ une formule prénexe et ϕ_1 une formule obtenue de ϕ par un pas de Skolémisation, alors ϕ est satisfaisable ssi ϕ_1 est satisfaisable.

Preuve. Supposons que μ est un modèle de ϕ_1.

1. Si $\phi \equiv \forall x_1 \cdot ... \cdot \forall x_k \cdot \exists x \cdot \psi$ et
   $\phi_1 \equiv \forall x_1 \cdot ... \cdot \forall x_k \cdot \psi(f(x_1, ..., x_k)/x)$ et le résultat provient de la validité de la formule

   $$\forall x_1 \cdot ... \cdot \forall x_k \cdot \psi(f(x_1, ..., x_k)/x) \rightarrow \forall x_1 \cdot ... \cdot \forall x_k \cdot \exists x \cdot \psi$$

2. et si $\phi \equiv \exists x \cdot \psi$ et $\phi_1 \equiv \psi(c/x)$ et le résultat de la validité de la formule $\psi(c/x) \rightarrow \exists x \cdot \psi$.

Supposons que μ soit un modèle de ϕ on va enrichir μ pour interpréter les termes de Skolem et satisfaire ϕ_1.

1. Soit $\phi \equiv \forall x_1 \cdot ... \cdot \forall x_k \cdot \exists x \cdot \psi$ et
   $\phi_1 \equiv \forall x_1 \cdot ... \cdot \forall x_k \cdot \psi(f(x_1, ..., x_k)/x)$.
   On sait que $\mu \models \phi$ et donc, pour n'importe quel vecteur de valeurs $a_1, ..., a_k \in M^\mu$, il existe une valeur a

   $$\mu \models \phi(a_1/x_1, ..., a_k/x_k, a)$$

   on construit μ' en attribuant l'interprétation $f^{\mu'}(a_1, ..., a_k) = a$ pour chaque vecteur $a_1, ..., a_k$.
   Il est claire que $\mu' \models \phi_1$
2. $\phi \equiv \exists x \cdot \psi$ et $\phi_1 \equiv \psi(c/x)$
   On sait qu'il existe une valeur v pour x telle que $\mu, v \models \psi$.
   Donc il suffit de prendre $c^{\mu'} = v(x)$ et on a $\mu' \models \psi(c/x)$

Logique du premier ordre
Résolution (I)

La résolution est une méthode de démonstration automatique pour les formules du premier ordre sous forme de clauses.

Pour faire de la résolution dans le premier ordre on a besoin de la notion d'*unification* .

La méthode de résolution et l'unification sont utilisées dans les algorithmes sous-jacents à la programmation logique (par exemple Prolog).

Logique du premier ordre
Résolution (II)

Un préalable à l'application de la méthode de résolution est la mise sous *forme clausale* .

Considérons un ensemble Σ de formules closes et une formule close ϕ, la question "ϕ est une conséquence logique de Σ ?" est équivalente par réfutation à la question suivante : "l'ensemble de formules $\Sigma \cup \{\neg\phi\}$ est-il non satisfaisable ?". Une manière de répondre à cette question consiste à transformer cet ensemble de formules en un ensemble S de clauses, puis appliquer les règles de résolution aux clauses de S.

Logique du premier ordre
Résolution (III)

La suite de transformations qui permet de passer d'un ensemble de formules à un ensemble de clôtures universelles de clauses, est constitué par :

- la mise sous forme prénexe et normale de chacune des formules de Σ ; notons que cette transformation préserve l'équivalence ;
- la mise sous forme de Skolem de chacune des formules obtenues ; cette transformation préserve la satisfaisabilité mais pas l'équivalence ;
- la distribution des quantificateurs universels par rapport aux conjonctions dans chacune des formules ; cette transformation préserve l'équivalence ;
- la décomposition des conjonctions en ensembles de clôtures universelles de clauses ; cette dernière transformation préserve l'équivalence.

Logique du premier ordre
Résolution (V)

Un exemple de mise sous forme de clauses : soit $\mathcal{L}$ un langage comportant deux symboles de prédicats unaires p, q et

$$\Sigma = \{(\exists x \cdot p(x)) \rightarrow \forall y \cdot p(y), \forall x \cdot (p(x) \vee q(x))\}$$
$$\text{et } \phi \equiv (\exists x \cdot \neg q(x)) \rightarrow \forall y \cdot p(y)$$

On applique la suite des transformations à l'ensemble $\Sigma \cup \{\neg\phi\}$:

– Formes prénexe :

$$\forall x \cdot \forall y \cdot (\neg p(x) \vee p(y)),$$
$$\forall x \cdot (p(x) \vee q(x)),$$
$$\exists x \cdot \exists y \cdot (\neg q(x) \wedge \neg p(y))$$

Logique du premier ordre
Résolution (VI)

– Skolémisation :

$$\forall x \cdot \forall y \cdot (\neg p(x) \vee p(y)),$$
$$\forall x \cdot (p(x) \vee q(x)),$$
$$\neg q(c) \wedge \neg p(d)$$

où c et d sont de nouveaux symboles de constantes,

– les formules restent les mêmes,

– les clauses obtenues sont :

$$(\neg p(x) \vee p(y)),$$
$$(p(x) \vee q(x)),$$
$$\neg q(c),$$
$$\neg p(d)$$

Logique du premier ordre
Résolution (VII)

Théorème : si S est l'ensemble des formules obtenues à partir de Σ en appliquant la suite des transformations précédentes, alors S est satisfaisable si et seulement si Σ est satisfaisable.

Preuve : les transformations 1,3 et 4 préservent l'équivalence des formules et la transformation 2 préserve le fait d'être satisfaisable.

Logique du premier ordre
Résolution (VIII) : unification

Pour pouvoir appliquer la règle de résolution aux clauses de S, il est nécessaire de savoir reconnaître si deux, ou plusieurs, formules atomiques peuvent être unifiées, c'est-à-dire s'il existe une substitution des variables de ces formules par des termes du langage, qui permette de les rendre égales. L'algorithme utilisé pour résoudre ce problème s'appelle l'algorithme d'unification.

Logique du premier ordre
Résolution (IX) : unification

Définitions préalables à la présentation de l'algorithme d'unification :

- une *substitution* α est une fonction de l'ensemble des variables dans l'ensemble des termes ;
- dans la suite, le domaine d'une substitution, c'est-à-dire $\{x \mid \alpha(x) \neq x\}$, est supposé fini. Si le domaine est $\{x_1, x_2, \ldots, x_n\}$ et si pour $i = 1, 2, \ldots, n$, t_i est le terme $\alpha(x_i)$, la substitution α est notée :
$$(t_1/x_1, t_2/x_2, \ldots, t_n/x_n)$$
- la substitution α appliquée à une formule $\phi(x_1, x_2, \ldots, x_n)$, notée $\alpha(\phi)$, est la formule obtenue en substituant le terme $\alpha(x_i)$ à chaque occurrence de la variable x_i dans ϕ, pour tout $i = 1, 2, \ldots, n$.

Notons que cette notion de substitution généralise la notion de substitution d'un terme à une variable que nous avons introduite précédement.

Logique du premier ordre
Résolution (XI) : unification

Exemple : dans la suite nous considérons un langage $\mathcal{L}$ comportant un symbole de prédicat binaire r, un symbole de fonction unaire f, deux symboles de fonctions binaires g, h, un symbole de fonction ternaire k et deux constante c, d. Soit la substitution

$$\alpha : (g(d, z)/x, f(z)/y, d/z)$$

et la clause

$$C \equiv r(x, f(y)) \vee \neg r(g(y, c), z)$$

la clause $\alpha(C)$ est

$$r(g(d, z), f(f(z))) \vee \neg r(g(f(z), c), d)$$

Logique du premier ordre
Résolution (XII) : unification

Un ensemble de formules atomiques

$$\{A_1, A_2, \ldots, A_n\}$$

est *unifiable* s'il existe une substitution α telle que $\alpha(A_1) = \alpha(A_2) = \ldots = \alpha(A_n)$.

Ainsi, les formules atomiques

$$r(x, f(y)), r(g(y, c), z), r(g(d, v), f(d))$$

sont unifiables à l'aide de la substitution

$$(g(d, c)/x, d/y, f(d)/z, c/v)$$

où x, y, z, v sont des variables et c, d des constantes.

Logique du premier ordre
Résolution (XIII) : unification

Proposition : si α est une substitution, alors il existe une fonction unique $\bar{\alpha}$ de l'ensemble des termes dans lui-même, qui prolonge α et qui respecte la structure des termes.

En effet, la valeur de $\bar{\alpha}(t)$ est définie par induction sur le terme t :

- si t est une constante c, on pose $\bar{\alpha}(c) = c$;
- si t est une variable x, on pose $\bar{\alpha}(x) = \alpha(x)$;
- si t est de la forme $f(t_1, t_2, \ldots, t_n)$ et si on suppose définis $\bar{\alpha}(t_1)$, $\bar{\alpha}(t_2)$, ..., $\bar{\alpha}(t_n)$ on pose $\bar{\alpha}(t) = f(\bar{\alpha}(t_1), \bar{\alpha}(t_2), \ldots, \bar{\alpha}(t_n))$.

Logique du premier ordre
Résolution (XIV) : unification

Conséquences de la définition de $\bar{\alpha}$:

– pour calculer l'effet d'une substitution à un terme $t(x_1, x_2, \ldots, x_n)$, il suffit de substituer le terme $\alpha(x_i)$ à chaque occurrence de la variable x_i dans le terme t ;

– si α est une substitution et t un terme clos, c'est-à-dire sans variable, alors $\bar{\alpha}(t) = t$.

Dans la suite, on notera α au lieu de $\bar{\alpha}$.

Logique du premier ordre
Résolution (XV) : unification

- une substitution α est un *unificateur* de deux termes t et t' si $\alpha(t) = \alpha(t')$;
- soit un ensemble fini de couples de termes, noté $S = \{(t_i, t'_i) \mid i = 1, 2, \ldots, m\}$; une substitution α est un *unificateur* de S si, pour tout $i = 1, 2, \ldots, m$, $\alpha(t_i) = \alpha(t'_i)$;
- un unificateur σ de S est *principal* si pour tout unificateur α de S, il existe une substitution β telle que $\alpha = \beta \circ \sigma$;
- une substitution σ est un *unificateur principal* d'un ensemble fini de formules atomiques $\{A_1, A_2, \ldots, A_n\}$ si $\sigma(A_1) = \sigma(A_2) = \ldots = \sigma(A_n)$ et si pour tout unificateur α de cet ensemble de formules, il existe une substitution β telle que $\alpha = \beta \circ \sigma$.

Logique du premier ordre
Résolution (XVI) : unification

Les termes $t = h(x, g(y, z))$ et $t' = h(f(z), v)$ sont-ils unifiables ?

Si α est un unificateur de t, t', il doit satisfaire : $\alpha(t) = \alpha(t')$, c'est-à-dire

$$h(\alpha(x), g(\alpha(y), \alpha(z))) = h(f(\alpha(z)), \alpha(v))$$

La substitution α doit donc satisfaire :

$$\alpha(x) = f(\alpha(z)),\ \alpha(v) = g(\alpha(y), \alpha(z))$$

Il suffit donc de prendre deux termes arbitraire u, u' respectivement pour $\alpha(y), \alpha(z)$ et α satisfaisant :

$$\alpha(x) = f(u') \text{ et } \alpha(v) = g(u, u')$$

pour obtenir un unificateur de t et t'.

Logique du premier ordre
Résolution (XVII) : unification

La *hauteur* i d'un terme d'un langage du premier ordre mesure la complexité de ce terme.

Formellement : soit $\mathcal{L}$ un langage du premier ordre. La suite des ensembles $\mathcal{T}_n$ est définie par récurrence sur l'entier n :

- $\mathcal{T}_0$ est la réunion de l'ensemble des constantes et de celui des variables ;
- $\mathcal{T}_n$ est supposé défini ; on prend $t \in \mathcal{T}_{n+1}$ si $t \in \mathcal{T}_n$ ou si t est de la forme $f(t_1, t_2, \ldots, t_k)$ pour un symbole de fonction f d'arité k et chaque $t_i \in \mathcal{T}_n$.

La hauteur d'un terme t est le plus petit entier n tel que $t \in \mathcal{T}_n$.

Notons la propriété suivante :

Théorème : si α est une substitution, alors pour tout terme t, la hauteur de $\alpha(t)$ est supérieure ou égale à la hauteur de t.

Logique du premier ordre
Résolution (XVIII) : unification

Théorème : Si σ et σ' sont des unificateurs principaux de deux termes t et t', alors ils sont égaux à une permutation des variables près.

Preuve. Si σ et σ' sont des unificateurs principaux de t et t' alors il existe γ et β tels que

$$\sigma = \gamma \circ \sigma' \text{ et } \sigma' = \beta \circ \sigma$$

$$\text{et donc } \sigma = \gamma \circ \beta \circ \sigma$$

$$\text{et donc } \gamma \circ \beta \text{ est l'identité}$$

Comme β n'agit que sur les variables de $\sigma(t)$, l'action de $\gamma \circ \beta$ sur ces variables est l'identité.

Notons que β transforme une variable x en une autre variable. En effet, comme nous venons de le montrer, une substitution ne peut associer un terme fonctionnel à une variable de σ sinon γ ne pourrait pas réattribuer

une variable au terme ainsi créé. En effet cela contredirait le résultat qu'une substitution ne peut pas faire descendre la hauteur d'un terme. De même l'association d'une constante serait irréversible.

Le même raisonnement peut être fait sur γ.

Logique du premier ordre
Résolution (XIX) : unification

Quelques définitions supplémentaires :

- la *hauteur d'un couple de termes* (t, t') est le minimum des hauteurs des termes t, t' ;
- la *hauteur d'un ensemble fini S de couples* de termes est le maximum des hauteurs des couples de S ;
- un système est un ensemble fini de couples de termes.

Logique du premier ordre
Résolution (XX) : unification

Lemme : Etant donné un système S de couples de termes, il existe une suite finie de transformations qui préservent les unificateurs de S, s'il en existe, et le transforme en un système S_1 de hauteur nulle, appelé *système simplifié* .

Preuve. Soit S un système de hauteur $h > 0$. La transformation que nous allons définir ci-dessous permet soit de montrer que S n'est pas unifiable, soit de déterminer un système S' qui est de hauteur $h' < h$ et qui a les mêmes unificateurs que S.

La transformation peut-être définie de la façon suivante :

- remplacer si possible chaque couple (t, t') de termes de hauteur $h > 0$ par un ensemble de couples de termes de hauteur $h' < h$
  - Soient deux termes t et t' tels que

$$t \quad = \quad f(t_1, t_2, ..., t_n)$$

$$t' \quad = \quad f'(t'_1, t'_2, ..., t'_m)$$

alors soit :

- $f \neq f'$ et les termes t et t' ne sont pas unifiables et donc S n'a pas d'unificateur
- $f = f'$ et donc $n = m$ et on remplace le couple (t, t') par l'ensemble des couples (t_i, t'_i) pour $i = 1, 2, ..., n$, qui sont de hauteurs $h' < h$ et ont les mêmes unificateurs que (t, t').

Après applications successives de ces transformations, on obtient, si S est unifiable, un ensemble S' de hauteur $h = 0$. On peut simplifier cet ensemble en appliquant les deux règles suivantes :

- s'il existe plusieurs couples identiques dans S', on n'en conserve qu'un ;
- on élimine les couples de la forme (t, t) où t est un terme.

Ces transformations préservent les unificateurs.

Logique du premier ordre
Résolution (XXI) : unification

Exemple : soit le système constitué par le seul couple (t, t') avec

$$t = k(f(c, g(x_4, x_5)), f(c, g(x_5, x_4)), x_2) \text{ et}$$
$$t' = k(x_2, x_2, x_6)$$

le système simplifié est l'ensemble des couples :

- $(f(c, g(x_4, x_5)), x_2)$
- $(f(c, g(x_5, x_4)), x_2)$
- (x_2, x_6)

Logique du premier ordre
Résolution (XXI) : unification

Définition : Soit S' un système simplifié. Un couple de termes dans S' est dit *réductible* s'il se compose d'une variable x et d'un terme t dans lequel x n'apparaît pas. Sinon il est dit *irréductible* .

Lemme : Si S' est un système simplifié qui possède un couple irréductible, alors S' n'est pas unifiable.

Preuve. Un couple irréductible peut être composé des trois manières suivantes

- (c, d) deux constantes distinctes ;
- $(f(t_1, ..., t_n), c)$ un terme fonctionnel et une constante ;
- $(x, f(..., x, ...))$ par l'absurde, considérons σ qui unifie x et $f(..., x, ...)$ on a $\sigma(x) = f(..., x, ...)$ or la hauteur de $\sigma(x)$ est strictement inférieure à la hauteur de $\sigma(f(..., x, ...))$ et donc les deux termes ne peuvent pas être égaux.

Logique du premier ordre
Résolution (XXII) : unification

Algorithme d'unification :
$\sigma := \{(x, x) \mid x \in \mathit{Var}\}$; $\mathit{Unifiable} := \mathit{true}$
Tant que $S \neq \emptyset$ et $\mathit{Unifiable} = \mathit{true}$

1. la phase de simplification transforme S, si c'est possible, en un système simplifié S', sinon elle s'arrête (et il n'existe pas d'unificateur) : $\mathit{Unifiable} := \mathit{false}$;
2. la phase de test permet d'arrêter l'algorithme en présence de termes irréductibles (et il n'existe pas d'unificateur) : $\mathit{Unifiable} := \mathit{false}$;
3. la phase de réduction consiste à prendre un couple réductible (x, t), ou (t, x), dans S', à considérer la $\sigma := \sigma[x \to t]$ et $S_1 := \{(\sigma(t), \sigma(t')) \mid \sigma(t) \neq \sigma(t') \wedge (t, t') \in S'\}$; [a]
4. $S := S_1$

Si $\mathit{Unifiable}=\mathit{true}$ alors retourner σ.

[a] La variable x n'apparaissant pas dans t, la substitution σ vérifie $\sigma(x) = t$.

Logique du premier ordre
Résolution (XXIII) : unification

Lemme : si S' est un système simplifié de couples réductibles, (x, t) l'un de ces couples de S' et σ la substitution définie par $\sigma(x) = t$: si α est un unificateur du système S', alors il existe une substitution β telle que $\alpha = \beta \circ \sigma$.

Théorème : l'algorithme d'unification termine et produit, dans le cas ou le système S est unifiable, une suite de substitutions $\sigma_1, \sigma_2, \ldots, \sigma_n$ dont la composée est un unificateur principal du système S.

Preuve. L'algorithme termine soit dans une phase de simplification ou de test par impossibilité de réaliser une unification, soit au après un nombre fini n d'étapes par un système vide. Dans ce dernier cas, il est facile de voir, par induction sur $i \leq n$, que tout unificateur de S est de la forme :

– $\beta \circ \sigma_i \circ \sigma_{i-1} \circ \ldots \circ \sigma_1$ où β est un unificateur de S_i : en particulier, tout

unificateur de S est de la forme $\gamma \circ \sigma_n \circ \sigma_{n-1} \circ ... \circ \sigma_1$ où γ est un unificateur de S_n.

Le système (S_n) étant vide, n'importe quelle substitution γ convient et $\sigma_n \circ ... \circ \sigma_1$ est alors un unificateur principal de S.

Logique du premier ordre
Résolution (XXIV) : unification

Dans l'exemple précédent, tous les couples de S' sont réductibles et l'on peut choisir le premier

$$(f(c, g(x_4, x_5)), x_2)$$

La substitution σ_1 de la phase de réduction est définie par :

$$\sigma_1(x_2) = f(c, g(x_4, x_5))$$

et σ_1 est l'identité sur les autres variables. Le système $S_1 = \sigma_1(S')$ est égal à :

– $(f(c, g(x_5, x_4)), f(c, g(x_4, x_5)))$
– $(f(c, g(x_4, x_5)), x_6)$

Logique du premier ordre
Résolution (XXIV) : unification

On applique à nouveau une phase de simplification au système S_1, et on obtient le système :

$$\{(c, c), (x_5, x_4), (x_4, x_5),$$
$$(f(c, g(x_4, x_5)), x_6)\}$$

que l'on peut simplifier en
– (x_4, x_5)
– $(f(c, g(x_4, x_5)), x_6)$
Ces deux couples sont réductibles.

Logique du premier ordre
Résolution (XXV) : unification

En choisissant la substitution

$$\sigma_2(x_4) = x_5$$

le système réduit est S_2 égal à :

$$(f(c, g(x_5, x_5)), x_6)$$

Ce système étant de hauteur nulle et constitué d'un seul couple réductible, la réduction s'effectue à l'aide de la substitution

$$\sigma_3(x_6) = f(c, g(x_5, x_5))$$

Le système réduit associé est vide et un unificateur principal du système S est

$$\sigma = \sigma_3 \circ \sigma_2 \circ \sigma_1$$

Logique du premier ordre
Résolution (XXVI) : règle

Etant donnée une clause C, les formules atomiques de C apparaissant sans négation sont dites positives et celles apparaissant précédées d'une négation sont dites négatives. La clause C peut être notée (Γ, Δ) où Γ, respectivement Δ, est l'ensemble des formules atomiques négatives, respectivement positives de C.

La *règle de résolution* permet de déduire une nouvelle clause à partir de deux clauses C_1 et C_2. Elle opère en deux temps : d'abord unification d'un ensemble de formules atomiques négatives de C_1 et positives de C_2, puis coupure sur la formule atomique obtenue.

Logique du premier ordre
Résolution (XXVII) : règle

Soit $C_1 = (\Gamma_1, \Delta_1)$ et $C_2 = (\Gamma_2, \Delta_2)$ deux *clauses séparées* , c'est-à-dire sans variables communes (il suffit de renommer des variables dans une des clauses si nécessaire).

S'il existe $P_1 \subseteq \Delta_1$ et $N_2 \subseteq \Gamma_2$ tels que l'ensemble des formules atomiques $P_1 \cup N_2$ soit unifiable et si σ est un unificateur principal, la nouvelle clause C, déduite des deux clauses C_1, C_2 par résolution, est définie par

$$\Gamma = \sigma(\Gamma_1) \cup \sigma(\Gamma_2 \setminus N_2) \text{ et}$$
$$\Delta = \sigma(\Delta_1 \setminus P_1) \cup \sigma(\Delta_2)$$

La clause C est dite *déduite par résolution* des clauses C_1 et C_2 relativement à l'unificateur σ (et P_1, N_2). On dit que la clause C est un *résolvant* de C_1 et C_2 relativement à σ.

Logique du premier ordre
Résolution (XXVII) : règle

Exemple : soit $\mathcal{L}$ un langage comportant les symboles de prédicats p, q, s unaires, r binaire, f un symbole de fonction unaire et les clauses :

- $C_1 \equiv \neg s(z) \vee p(z) \vee q(z)$
- $C_2 \equiv \neg p(f(y)) \vee r(x, y)$

Logique du premier ordre
Résolution (XXVIII) : règle

La clause suivante est le résolvants de C_1 et C_2 :

$$P_1 = \{p(z)\},\ N_2 = \{\neg p(f(y))\}$$

Donc, nous devons unifier $\{p(f(y)), p(z)\}$

On considère donc le système $S = \{(z, f(y))\}$ de hauteur zéro qui est unifié par $\sigma(z) = f(y)$.

Nous pouvons maintenant calculer C_3 (la clause résolvante) comme

$$C_3 \equiv (\sigma(C_1) \setminus \sigma(P_1)) \cup (\sigma(C_2) \setminus \sigma(N_2))$$

et donc

$$C_3 \equiv \neg s(f(y)) \vee q(f(y)) \vee r(x, y)$$

Logique du premier ordre
Résolution (XXIX) : règle

Soit S un ensemble de clauses et C une clause. Une *preuve par résolution* de C à partir de S est une suite finie de clauses $C_1, C_2, \ldots, C_n$ telles que $C_n = C$ et pour tout $i = 1, 2, \ldots, n$:
– soit C_i est une clause de S ;
– soit il existe $1 \leq j, k < i$ tels que C_i soit un résolvant de C_j, C_k.
L'existence d'une preuve C par résolution à partir de S est notée $S \vdash_R C$. Une *réfutation* de S est une preuve de la *clause vide* à partir de S. La clause vide est notée $\times$.

Logique du premier ordre
Résolution (XXX) : règle

Exercice. Montrez que l'ensemble de clauses :

- $\neg p(x) \vee p(y)$
- $p(z) \vee q(z)$
- $\neg p(c)$
- $\neg q(d)$

possède une réfutation.

Logique du premier ordre
Résolution (XXXI) : règle

La correction de la méthode de résolution est exprimée par la propriété suivante : l'existence d'une réfutation de S implique que S n'a pas de modèle. Pour établir ce résultat, nous prouvons d'abord l'adéquation d'un pas de coupure (formalisée dans le lemme suivant).

Lemme : si C est un résolvant de C_1, C_2 alors $\bar{C}$ est une conséquence logique de $\bar{C_1} \wedge \bar{C_2}$.

(la clôture universelle de C est notée $\bar{C}$)

Preuve. On a besoin de la propriété suivante dans la démonstration du lemme.

Soit $B(x_1, ..., x_k)$ une clause et μ une structure pour le langage $\mathcal{L}$.

Si σ est une substitution telle que pour $i = 1, 2, ..., k$:

$$\sigma(x_i) = t_i(y_1, ..., y_l)$$

et $b_1, ..., b_l \in M^\mu$, on pose : $a_i = t_i^\mu [b_1, ..., b_l]$

La clause B satisfait alors : $\mu, [a_1, ..., a_k] \models B$ ssi
$\mu, [b_1, ..., b_l] \models \sigma(B)$.

Nous pouvons maintenant aborder la preuve du lemme. Les clauses C_1, C_2 et C, résolvant de C_1, C_2 relativement à l'unificateur σ (pour L_1 et L_2) peuvent être notées sous la forme suivante :

$$\begin{aligned} C_1 &\equiv G_1 \rightarrow (F_1 \vee L_1) \\ C_2 &\equiv (F_2 \wedge L_2) \rightarrow D_2 \end{aligned}$$

$$\text{et } C \equiv (\sigma(G_1) \wedge \sigma(F_2)) \rightarrow (\sigma(F_1) \vee \sigma(D_2))$$

où L_1 est la disjonction des formules atomiques positives et L_2 la

conjonction des formules atomiques négatives qui sont unifiées par σ, donc :

$$\sigma(L_1) = \sigma(L_2)$$

On va maintenant établir la contraposée du lemme. On suppose donc qu'il existe μ tel que $\mu \not\models \bar{C}$. En d'autres termes, nous supposons qu'il existe μ et $b_1, b_2, ..., b_l \in M^\mu$ tels que

$$\mu, [b_1, ..., b_l] \models \neg C$$

On va maintenant établir que $\mu \not\models \bar{C}_1$ ou $\mu \not\models \bar{C}_2$.

Par hypothèse, $\mu, [b_1, \ldots, b_l] \models \neg C$, donc on a que μ et $[b_1, \ldots, b_l]$ sont tels que :

$$\mu, [b_1, ..., b_l] \models (\sigma\,(G_1) \wedge \sigma\,(F_2))$$

$$\text{et } \mu, [b_1, ..., b_l] \models \neg\,(\sigma\,(F_1) \vee \sigma\,(D_2))$$

Soient $x_1, x_2, ..., x_k$ les variables apparaissant dans C_1, C_2 et t_i les

termes définis par

$$\sigma\left(x_i\right) = t_i\left(y_1, ..., y_l\right) \text{ pour } i = 1, 2, ..., k.$$

La suite des a_i est définie par

$$a_i = t_i^{\mu}\left[b_1, ..., b_l\right]$$

Vu qu'on a $\sigma(L_1) = \sigma(L_2)$, deux cas sont possibles :

1. Soit $\mu, [b_1, ..., b_l] \models \sigma(L_2)$ et donc $\mu, [a_1, ..., a_k] \models L_2$ d'après la propriété rappelée précédemment.
   De la même manière on sait que $\mu, [a_1, ..., a_k] \models F_2$ et également $\mu, [a_1, ..., a_k] \models \neg D_2$.
   Donc $\mu, [a_1, ..., a_k]$ satisfait la négation de C_2.
2. Soit $\mu, [b_1, ..., b_l] \models \neg\sigma(L_1)$ et donc $\mu, [a_1, ..., a_k] \models \neg L_1$.
   De même $\mu, [a_1, ..., a_k] \models G_1$ et $\mu, [a_1, ..., a_k] \models \neg F_1$.
   et donc $\mu, (a_1, a_2, ..., a_k)$ satisfait la négation de la clause C_1.

Dans les deux cas, l'une des formules de $\left\{\bar{C}_1, \bar{C}_2\right\}$ est fausse dans μ.

Corollaire : Si $S \vdash_R C$, alors $\bar{C}$ est une conséquence logique de S. En particulier, s'il existe une réfutation de S, alors S n'a pas de modèle.

Logique du premier ordre
Résolution (XXXIV) : Modèle de Herbrand

Soit $\mathcal{T}$ l'ensemble des termes du langage $\mathcal{L}$.

Le *domaine de Herbrand* H est l'ensemble des termes clos, c'est-à-dire sans variables, de $\mathcal{L}$. Donc H est le plus petit sous-ensemble de $\mathcal{T}$ contenant les constantes et clos par application des fonctions. Si $\mathcal{L}$ ne contient pas de constante, on lui en ajoute une.

La *base de Herbrand* $\mathcal{B}$ est l'ensemble des formules atomiques closes de $\mathcal{L}$.

Logique du premier ordre
Résolution (XXXIV) : Modèle de Herbrand

Le domaine et la base de Herbrand vérifient les conditions suivantes :

1. si $\mathcal{L}$ a un nombre fini de prédicats et de constantes, mais sans fonctions, alors H et $\mathcal{B}$ sont finis ;
2. si l'ensemble des fonctions de $\mathcal{L}$ est non vide, alors H et $\mathcal{B}$ sont infinis dénombrables.

Logique du premier ordre
Résolution (XXXIV) : Modèle de Herbrand

Exemples de domaines et de bases de Herbrand :

- si $\mathcal{L}$ comporte deux prédicats unaires p, q et deux constantes c, d, alors
  - $H = \{c, d\}$;
  - $\mathcal{B} = \{p(c), p(d), q(c), q(d)\}$.
- si $\mathcal{L}$ comporte deux prédicats unaires p, q, une fonction unaire f et une constante c,
  - $H = \{c\} \cup \{f^n(c) \mid n \geq 1\}$;
  - $\mathcal{B} = \{p(c), q(c)\} \cup \{p(f^n(c)), q(f^n(c)) \mid n \geq 1\}$.

Logique du premier ordre
Résolution (XXXV) : Modèle de Herbrand

Une *structure de Herbrand* $\mathcal{H}$ pour le langage $\mathcal{L}$ est une structure telle que :

- l'ensemble de base est le domaine de Herbrand ;
- pour tout symbole de fonction f d'arité m, l'interprétation de f est la fonction qui à $(t_1, t_2, \ldots, t_m) \in H^m$ associe le terme $f(t_1, t_2, \ldots, t_m)$ (chaque terme est interprété par lui-même).

Pour compléter l'interprétation, il suffit pour chaque formule atomique close $r(t_1, t_2, \ldots, t_n)$ de préciser quelle est la valeur de vérité de cette formule dans $\mathcal{H}$: il faut définir une fonction $V : \mathcal{B} \to \{0, 1\}$. Une telle fonction définie sur un sous-ensemble de la base de Herbrand est appelée une *structure partielle*.

Soit ϕ une *formule close* . Un *modèle de Herbrand* de ϕ est une structure de Herbrand qui satisfait ϕ.

Implication : vu que $\mathcal{B}$ est fini ou dénombrable, on peut énumérer les structures partielles de Herbrand.

Logique du premier ordre
Résolution (XXXVI) : Modèle de Herbrand

Lemme : Soit ϕ une *formule universelle close* , ϕ possède un modèle si et seulement si elle possède un modèle de Herbrand.

Preuve. Montrons tout d'abord que si ϕ possède un modèle μ alors on peut construire un modèle de Herbrand H pour cette formule. Rappelons que, pour définir un modèle de Herbrand, il suffit de définir quelles sont les formules de la base de Herbrand qui sont vraies ou celles qui sont fausses. On sait que

$$\phi \equiv \forall x_1 ... \forall x_k \cdot \psi$$

où ψ est une formule sans quantificateur. On définit la structure de Herbrand H de la façon suivante : pour toute formule atomique close $p(t_1, ..., t_n)$, on pose

$$V(p(t_1, ..., t_n)) = 1 \text{ ssi } \mu \models p(t_1, ..., t_n).$$

Il est alors facile de montrer que $\mu \models \phi$ ssi $H \models \phi$, par induction sur la structure des formules.

L'autre direction est triviale car un modèle de Herbrand est un modèle particulier.

Implication de ce théorème : pour tester si une formule universelle close est satisfaisable ou non, on peut se restreindre à lui chercher un modèle de Herbrand.

Logique du premier ordre
Résolution (XXXVII) : Modèle de Herbrand

Etant donnée une formule close universelle ϕ de la forme $\forall x_1 \cdot \forall x_2 \cdot \ldots \forall x_n \cdot \psi$, une formule ξ est appelée instance de ψ si elle est obtenue en substituant dans ψ des termes clos $t_1, t_2, \ldots, t_n$ respectivement aux variables $x_1, x_2, \ldots, x_n$.

Lemme : Soit Σ un ensemble de fomules universelles closes. Si Σ n'a pas de modèle de Herbrand, alors il existe un nombre fini d'instances de formules de Σ qui n'ont pas de modèle.

(preuve omise)

Logique du premier ordre
Résolution (XXXVII) : Modèle de Herbrand

Représentation de la classe des modèles de Herbrand par des *arbres sémantiques* :

Un arbre sémantique est associé à une énumération $\{A_n\}$ des formules atomiques closes ($\mathcal{B}$ est fini ou dénombrable).

On va utiliser les arbres sémantiques pour énumérer les structures partielles de Herbrand pour une base $\mathcal{B}$ donnée (rappel : il suffit de déterminer la valeur de vérité de chaque formule atomique close de $\mathcal{B}$).

Logique du premier ordre
Résolution (XXXVII) : Modèle de Herbrand

Un *arbre sémantique* pour $\mathcal{B}$ est un arbre binaire dont les arêtes sont étiquetées par des formules atomiques closes de telle sorte que :

- il existe un seul noeud de niveau 0, appelé racine ;
- pour chaque noeud u de niveau $n-1$, une arrête partant de ce noeud est étiqueté par la formule A_n et l'autre arête par la formule $\neg A_n$.

Une feuille est un noeud terminal de l'arbre. Une branche est un chemin constitué de noeuds, qui conduit de la racine à une feuille de l'arbre.

Logique du premier ordre
Résolution (XXXVIII) : Modèle de Herbrand

A chaque noeud u d'un arbre sémantique est associée une valuation partielle $\mathcal{V}_u$ sur $\mathcal{B}$, et donc une structure de Herbrand partielle, notée $\mathcal{M}_u$: si u est un noeud de niveau n, c'est la valuation partielle définie en prenant la valeur 1 sur les formules atomiques étiquetant les arêtes menant de la racine au noeud u.

D'autre part, il existe un ordre partiel sur l'ensemble des noeuds de l'arbre : $u < u'$ si et seulement si la valuation $\mathcal{V}_{u'}$ prolonge strictement la valuation $\mathcal{V}_u$.

Logique du premier ordre
Résolution (XXXIX) : Modèle de Herbrand

Exemple d'arbre sémantique, pour la base de Herbrand
$\mathcal{B} = \{p(a), p(b)\}$ (arbre donné au cours)

Notons que, si la base de Herbrand considérée est infinie, alors l'arbre sémantique est infini également.

Logique du premier ordre
Résolution (XL) : Modèle de Herbrand

Soit A un arbre sémantique et S un ensemble (fini) de clauses.

- un noeud u de l'arbre A réfute une clause C de S s'il existe une substitution σ qui associe à chaque variable de la clause C un terme clos telle que la structure partielle $\mathcal{M}_u$ rend fausse la clause $\sigma(C)$;
- un noeud u est un noeud d'échec pour S si
  - il existe une clause de S qui est réfutée par u ;
  - pour tout noeud $u' < u$, il n'existe pas de clause dans S qui soit réfutée par u'.
- un arbre sémantique A est fermé pour l'ensemble des clauses de S si sur toute branche de l'arbre, il existe un noeud d'échec pour S.

Logique du premier ordre
Résolution (XLI) : Modèle de Herbrand

Exemple d'arbre sémantique fermé. Considérons la base de Herbrand suivante :

$$\mathcal{B} = \{p(c), q(c), p(f(c)), q(f(c)), \ldots, p(f^n(c)), q(f^n(c)), \ldots\}$$

et considérons les trois clauses suivantes :

- $C_1 \equiv \neg p(x) \vee q(x)$
- $C_2 \equiv p(f(y))$
- $C_3 \equiv \neg q(f(c))$

Nous allons construire une partie suffisante de l'arbre sémantique (infini) associé à $\mathcal{B}$ pour montrer que la conjonction des clauses C_1, C_2 et C_3 est non satisfaisable.

Logique du premier ordre
Résolution (XLI) : Modèle de Herbrand

(arbre donné au cours)

Logique du premier ordre
Résolution (XLIII) : Modèle de Herbrand

Théorème : un ensemble S de clauses est non satisfaisable si et seulement si il existe un arbre sémantique fermé pour S.

Preuve. D'après la définition des arbres sémantiques, toute structure de Herbrand correspond à une branche de l'arbre. On a également vu qu'un ensemble de clauses S est satisfaisable ssi S a un modèle de Herbrand.

S'il existe un arbre sémantique fermé pour S cela veut dire que chaque branche contient un noeud d'échec et toute structure de Herbrand réfute au moins une clause de S. Aucune structure de Herbrand n'est un modèle de S. Dans le cas contraire, au moins une branche de l'arbre semantique ne contient pas de réfutation (ne contient pas de noeud d'échec) et donc cette branche constitue un modèle de Herbrand pour l'ensemble de clauses de l'ensemble S.

Remarque : contrairement au cas propositionnel, la propriété, pour un ensemble de clauses du premier ordre, d'être satisfaisable n'est pas décidable en général : en effet le problème revient à vérifier si un arbre sémantique, qui peut être infini, est fermé ou non. Notons tout de même que, si l'on se restreint aux langages ne possédant qu'un nombre fini de symboles de relations et de constantes et pas de symboles fonctionnels, alors H et $\mathcal{B}$ sont finis et donc le problème devient décidable et se ramène au cas propositionnel.

Logique du premier ordre
Résolution (XLIV) : Modèle de Herbrand

Théorème : Un ensemble de clauses S est non satisfaisable si et seulement si il existe une réfutation de S par résolution.

Programmation Logique
Résolution et dérivations SLD (I)

La programmation logique repose sur une stratégie spécifique de la résolution, appelée *Sélection Linéaire Définie* . La stratégie SLD est une stratégie basée sur la résolution linéaire où, à chaque étape, une clause de l'ensemble de départ est utilisée.

Notons que la méthode n'est pas complète en général, mais elle est complète lorsqu'elle est appliquée à une classe particulière de clauses, appelées *clauses définies* .

Soit S un ensemble de clauses et C une clause. Une preuve de C par *résolution linéaire* est dite *par entrée*, si à chaque étape, le résolvant est obtenu en utilisant une clause de S.

Programmation Logique
Résolution et dérivations SLD (II)

Quelques notions nécessaires :

– une clause définie (aussi appelée *clause de Horn*) est une clause de la forme :

$$\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \rightarrow \psi$$

où $\phi_1, \phi_2, \ldots, \phi_n$ avec $n \geq 0$, et ψ sont des formules atomiques. La formule ψ s'appelle la **tête** de la clause et la conjonction $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ le **corps**.

– Un **programme logique** est un ensemble fini de clauses définies.

Programmation Logique
Résolution et dérivations SLD (III)

– Soit P un programme logique, G une clause négative, nommée *but* , et une clause C. Une preuve de C par résolution à partir de $P \cup \{G\}$ est dite $\mathcal{LD}$ si elle est linéaire par entrée et si la première étape de résolution utilise le but G. L'existence d'une telle preuve est notée

$$P \cup \{G\} \vdash_{\mathcal{LD}} C$$

– Une réfutation $\mathcal{LD}$ de $P \cup \{G\}$ est une preuve $\mathcal{LD}$ de la clause vide à partir de $P \cup \{G\}$.

Programmation Logique
Résolution et dérivations SLD (IV)

Exemple : soit $\mathcal{L}$ un langage comportant deux symboles de relations binaires q, r et deux symboles de constantes a, b, P le programme logique constitué des clauses $\{C_1, C_2, C_3\}$ et G le but $\neg q(x, b)$.

- $C_1 \equiv (r(x, y) \wedge q(y, z)) \rightarrow q(x, z)$
- $C_2 \equiv q(x, x)$
- $C_3 \equiv r(a, b)$

Programmation Logique
Résolution et dérivations SLD (V)

L'ensemble $P \cup \{G\}$ possède deux réfutations $\mathcal{LD}$:

– la première :

$$\dfrac{\neg q(x,b) \qquad q(x,x)}{\times}$$

– la seconde :

$$\dfrac{\dfrac{\dfrac{\neg q(x,b) \qquad (r(x,y) \wedge q(y,z)) \rightarrow q(x,z)}{\neg(r(x,y) \wedge q(y,b))} \qquad r(a,b)}{q(b,b)} \qquad q(x,x)}{\times}$$

Programmation Logique
Résolution et dérivations SLD (VI)

Le théorème suivant rend valide l'utilisation de la résolution $\mathcal{LD}$ dans le cadre de la programmation logique (clauses définies) :

Théorème : tout programme logique P possède un modèle de Herbrand. De plus, si G est une clause négative telle que $P \cup \{G\}$ ne possède pas de modèle, alors il existe une réfutation $\mathcal{LD}$ de $P \cup \{G\}$.

Montrons d'abord qu'un programme logique a toujours un modèle, et en particulier un modèle de Herbrand. En effet, le résolvant de deux clauses définies est une clause définie :

$$\phi_1^1 \wedge ... \wedge \phi_{n_1}^1 \rightarrow \Psi_1$$

$$\phi_1^2 \wedge ... \wedge \phi_{n_2}^2 \rightarrow \Psi_2$$

$$\sigma(\phi_1^1 \wedge ... \wedge \phi_{n_1}^1 \wedge \phi_1^2 \wedge ... \wedge \phi_{i-1}^2 \wedge \phi_{i+1}^2 \wedge ... \wedge \phi_{n_2}^2) \longrightarrow \sigma(\Psi_2)$$

En particulier, on ne pourra jamais déduire la clause vide. La résolution étant complète, on sait qu'un programme logique est toujours satisfaisable.

La résolution $\mathcal{LD}$ est complète. Si $P \cup G$ n'a pas de modèle alors $P \cup G$ possède une réfutation linéaire (par complétude de cette méthode). Une étape de résolution ne peut utiliser deux clauses négatives et le résolvant de l'application d'une clause définie avec une clause négative est une clause négative. Un résolvant négatif ne peut provenir que d'une clause négative et d'une clause définie. C'est le cas de la clause vide dans une réfutation. Dans un arbre de réfutation linéaire de $P \cup G$ par réfutation linéaire, il existe donc une branche dont toutes les étiquettes sont des clauses négatives : cette branche commence par G, qui est la seule clause négative de $P \cup G$. Et donc on doit utiliser à chaque étape une clause de P vu qu'on a que des clauses négatives.

Programmation Logique
Résolution et dérivations SLD (VII)

Une **dérivation SLD** de $P \cup \{G\}$ est une suite $(G_i, C_i, \sigma_i)_{i \geq 1}$ éventuellement infinie, telle que :

- $G_0 = G$ et $(G_i)_{i \geq 1}$ est une suite de buts,
- $(C_i)_{i \geq 1}$ est une suite de clauses de P,
- $(\sigma_i)_{i \geq 1}$ est une suite d'unificateurs principaux telle que pour chaque $i \geq 1$, G_{i+1} est un résolvant de G_i, C_{i+1} relativement à l'unificateur σ_i.

Programmation Logique
Résolution et dérivations SLD (VIII)

Une dérivation SLD finie de $P \cup \{G\}$ peut :

– soit *réussir* si c'est une réfutation de $P \cup \{G\}$,

– soit *échouer* si elle se termine par un but, qui n'est pas la clause vide, tel que l'atome sélectionné dans le but ne s'unifie avec la tête d'aucune clause de P.

Programmation Logique
Prolog (I)

Nous allons nous intéresser à une implémentation particulière de la résolution SLD. Cette implémentation de la programmation logique est appelée Prolog et est très répandue.

Prolog voit la résolution SLD comme un processus opérationnel de calcul appliqué à un programme logique P et à un but G, dont le résultat, appelé aussi *réponse*, est :

- soit une réponse, positive ou négative, si le but G est sans variable, suivant que G est une conséquence logique ou non du programme P ;
- soit une substitution obtenue par composition des unificateurs principaux d'une réfutation de $P \cup \{G\}$, puis par restriction aux variables de G ;
- soit une absence de réponse dans le cas d'une dérivation infinie.

Programmation Logique
Prolog (III) : syntaxe

La clause $(p(x, y) \wedge q(z, w)) \rightarrow r(x, w)$ est exprimé en Prolog avec la syntaxe suivante :

$$r(X, W) : -p(X, Y), q(Z, W).$$

La clause positive $p(a, x)$ est notée

$$p(a, X).$$

Programmation Logique
Prolog (IV) : un programme

1. $equivalent(X, X).$
2. $equivalent(X, Y) :- equivalent(Y, X).$
3. $equivalent(X, Z) :- \begin{array}{l} equivalent(X, Y), \\ equivalent(Y, Z). \end{array}$
4. $equivalent(a, b)$
5. $equivalent(b, c)$

Programmation Logique
Prolog (V) : implémentation de SLD

Prolog implémente une stratégie particulière de recherche qui peut être spécifiée à l'aide :

– d'une règle de calcul ;

– d'une stratégie de recherche.

Dans les interpréteurs Prolog standards, la règle de calcul consiste à sélectionner l'atome *le plus à gauche* dans un but et la recherche s'effectue en *profondeur d'abord*, en examinant les clauses du programme dans l'ordre d'écriture.

Programmation Logique
Prolog (VI) : implémentation de SLD

L'implémentation utilise une pile des buts. Un état courant de la pile représente la dérivation examinée actuellement :

- lorsque l'atome sélectionné dans le but est au sommet de la pile et s'unifie avec la tête d'une clause, le résolvant est empilé et devient le nouveau but courant ;
- lorsqu'il existe pas de clause dont la tête puisse s'unifier avec l'atome sélectionné dans le but courant, ce but est dépilé et il y a alors *backtracking* pour examiner à nouveau le but précédent avec les clauses suivantes, dans l'ordre d'écriture.

Programmation Logique
Prolog (VII) : implémentation de SLD

Illustrons cette stratégie de recherche sur un exemple. Considérons le programme P suivant :

1. $p(X, X) : -q(X, Y), r(X, Z).$
2. $p(X, X) : -s(X).$
3. $q(b, a).$
4. $q(a, a).$
5. $q(X, Y) : -r(a, Y).$
6. $r(b, Z).$
7. $s(X) : -q(X, a).$

(traité au cours)

Programmation Logique
Prolog (VIII) : implémentation de SLD

Considérons le but $G = \{\neg p(X, X)\}$. Rappelons-nous que, si nous arrivons à dériver la clause vide de $P \cup \{\neg p(X, X)\}$, alors on sait que $p(X, X)$ est une conséquence logique de P.

Une dérivation est représentée à l'aide d'un arbre, appelé **arbre SLD**. Le long des branches de l'arbre, nous noterons les clauses de P qui sont utilisées pour obtenir le résolvant. On respectera également la convention suivante : les successeurs sont listés de gauche à droite dans l'ordre de leur apparition dans le programme. Un chemin est un *succès* s'il se termine par une réponse positive, que l'on notera $\times$. Un chemin est un *échec* s'il se termine avec une clause G' telle qu'il n'existe aucune clause du programme avec laquelle on peut résoudre sur l'atome le plus à gauche de G'.

Programmation Logique
Prolog (IX) : implémentation de SLD

Nous avons vu dans l'exemple précédent que, parmi les six chemins possibles, trois terminent en échec, deux terminent avec un succès et la substitution $\{X/b\}$, et un autre succès avec la substitution $\{X/a\}$.

Notons encore que le prouveur de théorèmes au coeur de Prolog cherche dans l'arbre SLD après une branche à succès en essayant d'abord le chemin le plus à gauche. C'est-à-dire que Prolog tente de résoudre le but courant avec la première clause de P qui permet l'application d'un pas de résolution. Dans notre exemple, Prolog utilisera donc les clauses $1, 3, 6$ du programme pour obtenir la réponse correcte : la substitution $\{X/b\}$.

Programmation Logique
Prolog (X) : implémentation de SLD

Si le prouveur de théorèmes atteint un noeud d'échec (failure) alors il *backtrack*. C'est-à-dire qu'il remonte le chemin qu'il vient de parcourir jusqu'au moment où il trouve une branche à droite qui doit encore être parcourue. S'il y a plus d'une de ces branches, alors le prouveur de théorèmes prend celle qui est la plus à gauche. Le prouveur de théorèmes répète l'opération jusqu'au moment où il trouve un chemin à succès. (il se peut qu'il ne s'arrête jamais)

Programmation Logique
Prolog (XI) : implémentation de SLD

Considérons P sans la clause 3 :

1. $p(X, X) : -q(X, Y), r(X, Z).$
2. $p(X, X) : -s(X).$
3.
4. $q(a, a).$
5. $q(X, Y) : -r(a, Y).$
6. $r(b, Z).$
7. $s(X) : -q(X, a).$

(traité au cours)

Programmation Logique
Prolog (XII) : implémentation de SLD

La procédure de backtracking est également utilisée lorsque nous demandons à Prolog une seconde réponse.

Par exemple, avec le programme original P, si nous posons la question

$$? - p(X, X).$$

On obtient la réponse $X = b \rightarrow$. Si nous demandons une deuxième réponse (en entrant " ;"), le prouveur de théorèmes backtrack du noeud à succès jusqu'au moment où il rencontre un noeud avec un chemin alternatif. Quand il trouve un tel noeud il recommence la recheche à partir de ce noeud sur la branche la plus à gauche non encore empruntée.

Programmation Logique
Prolog (XIII) : implémentation de SLD

Complétude de la résolution SLD et Prolog.

Si le prouveur de théorèmes répond "non". Nous savons que $P \cup \{G\}$ est satisfaisable et donc il n'existe pas de substitution pour laquelle le but est une conséquence logique de P.

Malheureusement, du point de vue de la complétude, Prolog est implémenté suivant une procédure de *recherche en profondeur d'abord*. En effet, le prouveur de théorèmes tente d'aller le plus profond possible dans un chemin avant d'essayer un autre chemin. Une alternative à la recherche en profondeur d'abord serait la *recherche en largeur d'abord*. En général, la recherche en profondeur d'abord est plus efficace. Malheureusement, elle entraîne l'incomplétude de l'implémentation de Prolog.

Programmation Logique
Prolog (XIV) : implémentation de SLD

Le théorème général de complétude garantit que si $P \cup \{G\}$ est insatisfaisable, alors il existe une réfutation-SLD (nécessairement finie) qui commence avec G. Notons que s'il y a une réfutation de longueur n alors on peut toujours la découvrir avec une recherche en largeur d'abord qui parcourt tous les chemins de longueur n. Malheureusement, on n'a pas une telle garantie avec une recherche en profondeur d'abord. Le problème est que certains chemins peuvent être **infinis**. Ce phénomène est illustré sur l'exemple suivant.

Programmation Logique
Prolog (XV) : implémentation de SLD

Considérons le programme P'' (seul la clause 6 diffère) :

1. $p(X, X) : -q(X, Y), r(X, Z).$
2. $p(X, X) : -s(X).$
3. $q(b, a).$
4. $q(a, a).$
5. $q(X, Y) : -r(a, Y).$
6. $r(W, Z) : -r(b, Z).$
7. $s(X) : -q(X, a).$

Programmation Logique
Prolog (XVI) : implémentation de SLD

Nous pouvons constater en analysant l'exemple précédent que l'ordre des clauses a de l'importance. En effet, si on réordonne le programme P'' en intervertissant les clauses 1 et 2, le prouveur de théorèmes trouvera la réponse $\{X/b\}$ et ensuite $\{X/a\}$ et ce n'est que si on demande une troisième réponse que le prouveur de théorèmes entrera sur un chemin infini.

Malheureusement, un réordonnancement des clauses ne suffit pas toujours. La recherche en profondeur d'abord est **intrinsèquement incomplète**.

Programmation Logique
Prolog (XVII) : implémentation de SLD

En effet, reconsidérons l'exemple suivant :

1. $equivalent(X, X).$
2. $equivalent(X, Y) :- equivalent(Y, X).$
3. $equivalent(X, Z) :- \begin{array}{l} equivalent(X, Y), \\ equivalent(Y, Z). \end{array}$
4. $equivalent(a, b)$
5. $equivalent(b, c)$

et le but $G \equiv \neg equivalent(c, a).$

Programmation Logique
Prolog (XVIII) : implémentation de SLD

Notons d'abord que $equivalent(c, a)$ est clairement une conséquence logique du programme. Le problème est que, peu importe dans quel ordre apparaissent les clauses 1 et 2 dans le programme ci-dessus, une recherche en profondeur d'abord ne pourra utiliser qu'une seule des deux clauses. Il est également facile de voir que si on supprime une des deux clauses, $equivalent(c, a)$ n'est plus une conséquence logique du programme.

Programmation Logique
Prolog : exemples de programmes (I)

Considérons le programme Prolog suivant :

$parent(luc, anne).$

$parent(marie, anne).$

$parent(henry, marie).$

$parent(henry, thierry).$

$parent(thierry, noel).$

$parent(julie, helene).$

$ancetre(X, X).$

$ancetre(X, Y) : -parent(X, Y).$

$ancetre(X, Y) : -parent(X, Z), ancetre(Z, Y).$

$ancetre(X, Y) : -parent(Z, Y), ancetre(X, Z).$

$apparente(X, Y) : -ancetre(Z, X), ancetre(Z, Y).$

Programmation Logique
Prolog : exemples de programmes (II)

Que donnent les questions suivantes :

– $?apparente(anne, noel).$

– $?apparente(anne, X).$

Calculez l'arbre de dérivation.

Programmation Logique
Prolog : exemples de programmes (III)

Que se passe-t-il si on rend le programme plus “déclaratif” ? Par exemple si on prend le programme suivant ?

$parent(luc, anne).$

$parent(marie, anne).$

$parent(henry, marie).$

$parent(henry, thierry).$

$parent(thierry, noel).$

$parent(julie, helene).$

$ancetre(X, X).$

$ancetre(X, Y) : -parent(X, Y).$

$ancetre(X, Y) : -ancetre(X, Z), ancetre(Z, Y).$

$apparente(X, Y) : -ancetre(Z, X), ancetre(Z, Y).$

Que se passe-t-il si on pose les mêmes questions sur ce nouveau programme ?

Programmation Logique
Prolog : exemples de programmes (IV)

Dans la deuxième formulation, on a exprimé plus “déclarativement” que la relation ancêtre est transitive.

Même si la sémantique “déclarative” est identique (c’est-à-dire que l’ensemble des valuations qui rendent la formule vraie sont les mêmes), la sémantique opérationnelle est différente.

Quand on écrit un programme Prolog, on est malheureusement obligé de tenir compte de la façon dont le programme va être exécuté.

Programmation Logique
Prolog : exemples de programmes (V)

Quelques règles générales quand on écrit un programme Prolog :

- quand on utilise des définitions récursives (cfr *ancetre*), le cas de base doit précéder le cas inductif.
- les faits connus pour un prédicat précèdent généralement les règles définies sur ce prédicat.

Attention ce ne sont que des heuristiques...

Programmation Logique
Prolog : exemples de programmes (V)

Une structure de donnée de Prolog : la liste.

- cas de base : [] empty list.
- constructeur : $.(b, T)$, on construit une nouvelle liste par concaténation de la constante b à la liste T.

Les programmes qui manipulent les listes seront naturellement récursifs.

Programmation Logique
Prolog : exemples de programmes (V)

Exemples de listes :

- $.(b, [])$ liste d'un seul élément b
- $[a, b, c, d]$, abbréviation pour $.(a, .(b, .(c, .(d, []))))$
- on peut également écrire $[a|[b, c, d]]$
- $[b|X]$ définit l'ensemble des listes qui commencent avec l'élément b
- $[a, [a, b], b]$, une liste avec trois éléments dont le deuxième est lui-même une liste.

Programmation Logique
Prolog : exemples de programmes (V)

Exemple de programme qui manipule des listes : le programme *append*.

Le but du programme est de calculer la concaténation de deux listes. Voici le programme :

$append([], X, X).$

$append([X|Y], Z, [X|W]) : -append(Y, Z, W).$

Le prédicat *append* est défini par induction. Le cas de base dit que la concaténation de la liste vide avec une liste X est cette même liste X. Le cas inductif exprime que si la concaténation de Y et Z est W, alors la concaténation de la liste Y augmentée de l'élément X avec la liste Z est la liste W augmentée de l'élément X.

Programmation Logique
Prolog : exemples de programmes (V)

Comment la recherche Prolog répond-t-elle aux questions suivantes :

- $?append([a], [b], [a, b])$.
- $?append([a, b], [c], [a, b, c])$.
- $?append([a, b], [c], [a, b, b, c])$.
- $?append([a], X, [Y|Z])$.
- $?append(X, Y, [a, b, c])$.

Programmation Logique
Prolog : exemples de programmes (V)

Dans un langage de programmation impératif comme Pascal, les arguments d'une procédure peuvent toujours être classifiés comme *input* ou *ouput*. Ce n'est pas le cas en Prolog. Il y a dans un programme Prolog une symétrie inhabituelle entre input et output.

Considérons à nouveau le programme "append".

$append([], X, X).$

$append([X|Y], Z, [X|W]) :- append(Y, Z, W).$

Le prédicat $append(X, Y, Z)$ exprime que la concaténation des listes X et Y est égale à la liste Z.

Programmation Logique
Prolog : exemples de programmes (V)

Si on pose la question $?append([a, b], [c, d], Z).$, les deux premiers arguments sont des inputs à la procédure, qui calcule la concaténation des listes $[a, b]$ et $[c, d]$.

Par contre, si on pose la question $?append(X, Y, [a, b, c, d])$, le dernier argument devient input et les deux premiers outputs. Quand on pose cette question, la procédure calcule quelles sont les listes qui, quand elles sont concaténées, donnent la liste $[a, b, c, d]$.

Cette caractéristique de Prolog est appelée la multi-directionnalité. Cette multi-directionnalité donne une grande souplesse et expressivité au langage. Cela a parfois un prix au niveau de l'efficacité.

Programmation Logique
Prolog : exemples de programmes (V)

Un autre programme qui manipule des listes : $member$.

$member(X, [X|Y])$.

$member(X, [Y|T]) :- member(X, T)$.

A nouveau la définition est récursive. Calculez les réponses aux questions suivantes :

– $?member(a, [a, b])$.
– $?member(X, [a, b, c])$.
– $?member(a, [X|Y])$.
– $?member(X, Y)$.

A nouveau, ces questions illustrent la puissance de la multi-directionnalité de Prolog.

Programmation Logique
Prolog : exemples de programmes (V)

Parcours d'un graphe :

$a(namur, bruxelles).$

$a(bruxelles, ostende).$

$a(ostende, courtrai).$

$a(courtrai, mons).$

$a(mons, charleroi).$

$a(charleroi, namur).$

$a(mons, nivelles).$

$a(nivelles, bruxelles).$

$a(X, Y) :- a(Y, X).$

$go(From, To) :- a(From, To).$

$go(From, To) :- a(From, In), go(In, To).$

Programmation Logique
Prolog : exemples de programmes (V)

Comment retenir par où l'itinéraire passe ? On mémorise le chemin dans une liste.

$a(namur, bruxelles).$

...

$a(nivelles, bruxelles).$

$go(From, To, []) :- a(From, To).$

$go(From, To, [In|T]) :- a(From, In), go(In, To, T).$

On peut poser les questions suivantes :

– $?go(namur, ostende, T).$

– $?go(namur, X, [bruxelles]).$

Programmation Logique
Prolog : exemples de programmes (V)

La résolution n'est pas efficace lorsqu'il s'agit d'évaluer des opérations arithmétiques. On peut coder les entiers, les opérations (comme l'addition) et certaines de leur propriétés. Et ce, en utilisant des clauses comme :

$$add(s(X), Y, s(Z)) : -add(X, Y, Z).$$

Pour évaluer directement des expressions arithmétiques, Prolog offre un prédicat infixe **is** :

Result is Expression

Programmation Logique
Prolog : exemples de programmes (V)

Par exemple la clause suivante retrouve le prix catalogue d'un article à partir d'une base de faits ainsi qu'un pourcentage de remise et établit le prix de vente :

$$
\begin{array}{ll}
sellingprice & (Item, Price) :- \\
 & listeprice(Item, List), \\
 & discountpercent(Item, Discount), \\
 & Price\ is\ List - List * Discount/100
\end{array}
$$

Attention, les prédicats arithmétiques diffèrent des autres prédicats. Un prédicat arithmétique est unidirectionnel ! Par exemple, si "10 is $X + Y$" était un prédicat logique, alors X et Y pourraient être unifiées à 1 et 9 par exemple, ou encore à 2 et 8, via un backtracking. Néanmoins, ceci est illégal. Dans le prédicat *Result is Expression*, les variables de *"Expression"* doivent être instanciées à des valeurs numériques lorsque le prédicat est évalué.

Programmation Logique
Prolog : exemples de programmes (V)

Pour terminer, nous abordons un moyen (pas très élégant mais souvent efficace) pour contrôler l'exécution d'un programme logique, via le prédicat "!" appelé "*cut* ".

Syntaxiquement "!" peut apparaître comme n'importe quel autre prédicat.

$$p :- q_1, q_2, !, q_3, q_4$$

Cut n'a aucune sémantique (déclarative), mais il altère la recheche SLD de la façon suivante : quand la clause ci-dessus est appelée dans une recherche, le sous-but $q_1, q_2, !, q_3, q_4$ est inséré au début du but courant. La recherche tente de satisfaire q_1, puis q_2 comme d'habitude. Si ca réussit, on passe outre le cut et on continue avec q_3 et q_4. Si on réussit, on continue en ignorant le cut. Sinon, si q_3 puis q_4 échouent, on doit backtracker. Et en remontant sur le "cut",

on fait comme si p avait échoué (donc on ne remonte pas sur q_2). La recherche repart alors du but parent qui a appelé p et on cherche la branche suivante à droite de ce noeud.

Programmation Logique
Prolog : exemples de programmes (V)

Illustration : considérons le programme Prolog suivant.

$t :- p, r.$

$t :- s.$

$p :- q_1, q_2, !, q_3, q_4.$

$q_1.$

$q_2.$

$s.$

$u.$

Et le but $\neg t$. Calculons l'arbre de dérivation.

Programmation Logique
Prolog : exemples de programmes (V)

Une application du cut : le calcul de la factorielle.

$$factorial(0, 1).$$

$$\begin{aligned} factorial(N, F) :- \quad & N1 \ is \ N - 1, \\ & factorial(N1, F1), \\ & F \ is \ N * F1. \end{aligned}$$

Maintenant faisons l'hypothèse que l'on appelle factorielle dans d'une autre procedure.

$$check(N) :- factorial(N, F), property(F).$$

Si *check* est appelé avec la valeur $N = 0$, il y a un appel de $factorial(0, F)$ qui va renvoyer $F = 1$ et appeler $property(1)$. Supposons que cet appel échoue. Alors il y a backtracking, et on "défait" la substitution $F = 1$. On essaye à nouveau un appel à

factorielle. On n'utilise plus le cas de base mais le cas inductif avec $F = 0$. Cela à pour effet de générer l'appel :

$$\begin{aligned} factorial(0, F) :- \quad & N1 \ is \ 0 - 1, \\ & factorial(-1, F1), \\ & F \ is \ 0 * F1. \end{aligned}$$

et cela entraine un boucle infinie. Pour éviter cela, on peut utiliser le cut de la façon suivante :

$$factorial(0, 1) :- !.$$

$$\begin{aligned} factorial(N, F) :- \quad & N1 \ is \ N - 1, \\ & factorial(N1, F1), \\ & F \ is \ N * F1. \end{aligned}$$

Le cut évite ainsi de backtracker sur la procédure *factorial* lorsque celle-ci a été appelée avec 0.

Logique du premier ordre
déduction naturelle (I)

Pour raisonner dans le logique du premier ordre, on utilise les règles de déduction naturelle de la logique propositionnelle et on y ajoute des règles pour (i) l'égalité, (ii) la quantification existentielle, et (iii) la quantification universelle.

Règles de preuves pour l'égalité

$$\frac{}{t=t}\ =_i$$

$$\frac{t_1=t_2 \qquad \phi[t_1/x]}{\phi[t_2/x]}\ =_e$$

Logique du premier ordre
déduction naturelle
Exemples

$t_1 = t_2 \vdash t_2 = t_1$

1	$t_1 = t_2$	*prémisse*
2	$t_1 = t_1$	$=_i$, rem : $\phi \equiv x = t_1$
3	$t_2 = t_1$	$=_e, 1, 2$

Logique du premier ordre
déduction naturelle
Exemples

$t_1 = t_2, t_2 = t_3 \vdash t_1 = t_3$

1	$t_2 = t_3$	*prémisse*
2	$t_1 = t_2$	*prémisse* , rem : $\phi \equiv t_1 = x$
3	$t_1 = t_3$	$=_e, 1, 2$

Logique du premier ordre
déduction naturelle (I)

Règles de preuves pour la quantification universelle

$$\frac{\forall x \cdot \phi}{\phi[t/x]} \quad \forall x_e$$

avec t libre pour x

$$\frac{\begin{array}{|c|}\hline x_0 \\ \vdots \\ \phi[x_0/x] \\ \hline \end{array}}{\forall\, x\, \phi} \quad \forall_i$$

x_0 doit être une variable *fraîche*, c'est-à-dire que x_0 n'apparaît nulle part en dehors de la boîte. En effet, on ne peut faire aucune hypothèse sur x_0 (on veut faire une preuve quel que soit x_0).

Logique du premier ordre
déduction naturelle
Exemples

$\forall x \cdot (p(x) \to q(x)), \forall x \cdot p(x) \vdash \forall x \cdot q(x)$

1	$\forall x \cdot (p(x) \to q(x))$	*prémisse*
2	$\forall x \cdot p(x)$	*prémisse*
3	x_0	x_0 fresh
4	$p(x_0) \to q(x_0)$	$\forall_e, 1$
5	$p(x_0)$	$\forall_e, 2$
6	$q(x_0)$	$\to_e, 4, 5$
7	end hypothesis on x_0	
8	$\forall x \cdot q(x)$	$\forall x_i$

Logique du premier ordre
déduction naturelle (I)

$$\frac{\phi[t/x]}{\exists x \cdot \phi}\ \exists x_i$$

$$\frac{\exists x \cdot \phi \qquad \boxed{\begin{array}{c} x_0\ \phi[x_0/x] \\ \vdots \\ \chi \end{array}}}{\chi}\ \exists\, x_e$$

où x_0 n'apparaît pas dans χ et n'est pas utilisé ailleurs.

Logique du premier ordre
déduction naturelle
Exemples

$\forall x \cdot \phi \vdash \exists x \cdot \phi$

1	$\forall x \cdot \phi$	*prémisse*
2	$\phi[x/x]$	$\forall_e, 1$
3	$\exists x \cdot \phi$	$\exists x_i$

Logique du premier ordre
déduction naturelle
Exemples

$\forall x \cdot (p(x) \to q(x)), \exists x \cdot p(x) \vdash \exists x \cdot q(x)$

1	$\forall x \cdot (p(x) \to q(x))$	*prémisse*
2	$\exists x \cdot p(x)$	*prémisse*
3	x_0	
4	$p(x_0)$	$\exists_e, 2$
5	$p(x_0) \to q(x_0)$	$\forall_e, 1$
6	$q(x_0)$	$\to_e, 4, 5$
7	$\exists x \cdot q(x)$	$\exists_i, 5$
8	end hypothesis on x_0	
9	$\exists x \cdot q(x)$	$\exists x_e$

Logique du premier ordre
Théorème de complétude (Goëdel)

Complétude du système de preuves de la logique du premier ordre :

$$\Delta \models \phi \text{ implique } \Delta \vdash \phi$$

Note : l'adéquation, i.e. $\Delta \vdash \phi$ implique $\Delta \models \phi$ est établie de manière classique.

Théorème de complétude
Définition d'un ensemble de formules consistantes

Δ est consistant s'il n'existe pas de ϕ tel que :

$$\Delta \vdash \phi$$

$$\Delta \vdash \neg\phi$$

Donc, la notion de consistance est une notion *syntaxique*. Voici deux exemples d'ensembles non consistants :

ex :

– $\Delta = \{\phi, \neg\phi\}$: Δ est non consistant.

– $\Delta = \{\phi, \phi \rightarrow \Psi, \neg\Psi\}$ est également non consistant.

Théorème de complétude
Définition de la cohérence

Une formule ϕ est cohérente avec Δ si

$$\Delta \cup \{\phi\} \nvdash \bot$$

Cela revient à dire que $\Delta \cup \{\phi\}$ est consistant.

Or nous voulons montrer que :

$$\Delta \models \phi \text{ implique } \Delta \vdash \phi$$

ce qui est équivalent, par contraposée, à :

$$\equiv \Delta \nvdash \phi \text{ implique } \Delta \not\models \phi$$

$$\equiv \Delta \cup \{\neg\phi\} \text{ est consistant}$$
$$\text{implique}$$
$$\Delta \cup \{\neg\phi\} \text{ a un modèle.}$$

Théorème de complétude

Pour passer de la deuxième formulation à la troisième, on utilise le fait que :

$$\Delta \nvdash \phi \text{ ssi } \Delta \cup \{\neg\phi\} \text{ est consistant}$$

Prouvons cette équivalence :

- $\Delta \nvdash \phi$ implique que $\Delta \cup \{\neg\phi\}$ est consistant
  On prouve la contraposée. Si $\Delta \cup \{\neg\phi\}$ est inconsistant, $\Delta \cup \{\neg\phi\} \vdash \bot$, et donc $\Delta \cup \{\neg\phi\} \vdash \phi$, par la règle de raisonnement sous hypothèses, on obtient $\Delta \vdash \neg\phi \rightarrow \phi$, ce qui est équivalent à $\Delta \vdash \neg\neg\phi \vee \phi$, par raisonnement propositionnel, cela donne $\Delta \vdash \phi$.
- $\Delta \cup \{\neg\phi\}$ est consistant implique que $\Delta \nvdash \phi$
  On établit la contraposée. Si $\Delta \vdash \phi$, alors $\Delta \cup \{\neg\phi\} \vdash \phi \wedge \neg\phi$, donc $\Delta \cup \{\neg\phi\}$ est inconsistant.

Théorème de complétude

Prouver la complétude revient donc à montrer que tout ensemble consistant de formules Δ possède un modèle.

La solution est simple et ingénieuse : le modèle va être un modèle syntaxique (rappelez-vous les modèles de Herbrand).

Le domaine d'interprétation sera le domaine des termes sur $\mathcal{L}$ et l'interprétation des fonctions et prédicats sera définie au regard de Δ.

Il y a deux difficultés majeures :

1. Le domaine des termes peut ne pas être assez riche pour construire un modèle. Par exemple :

   $$\Delta = \{\exists x \cdot p(x)\} \cup \{\neg p(t) : t \text{ est un terme de } \mathcal{L}\}$$

   Bien que Δ soit satisfaisable, on ne peut pas construire un modèle pour Δ à partir des termes.

2. De plus, Δ peut ne pas nous donner assez d'information pour nous guider dans la construction du modèle. Par exemple, pour un terme t et un prédicat p, Δ peut ne pas contenir de manière explicite l'information pour décider si $p(t)$ doit être vrai ou faux.

Théorème de complétude

Solutions à ces deux problèmes (Léon Henkin).

Pour 1.

On va ajouter au langage $\mathcal{L}$ un nombre infini dénombrable de constantes

$$\mathcal{L}' = \mathcal{L} \cup \{c_1, c_2, \ldots, c_n, \ldots\}.$$

Maintenant, il faut montrer que cette modification n'altère pas la consistance de Δ. En effet, on a ajouté des constantes à $\mathcal{L}$. Donc il y a des nouvelles formules que l'on peut déduire de Δ. Peut-être peut-on déduire $\perp$ de Δ ?

On va voir que c'est faux, Δ est toujours consistant.

Théorème de complétude

Lemme : Si Δ est consistant en considérant $\mathcal{L}$ alors Δ le reste pour $\mathcal{L}' = \mathcal{L} \cup \{c_1, c_2, \ldots, c_n, \ldots\}$.

Proof. Si Δ n'est pas consistant pour $\mathcal{L}'$ alors on a une preuve

$$S = (\phi_1, \ldots, \phi_n) \qquad \phi_n \equiv \Psi \wedge \neg\Psi$$

On peut faire l'hypothèse qu'il y a un nombre infini dénombrable de variables qu'on n'utilise pas dans la preuve (en effet la preuve est finie).

Dans la preuve S qui aboutit à une contradiction, on a utilisé des constantes de $\{c_0, c_1, \ldots, c_n, \ldots\}$ (sinon on aurait déjà pu faire la preuve avec $\mathcal{L}$ et Δ serait non consistant pour $\mathcal{L}$).

Comme ces nouvelles constantes n'apparaissent pas dans Δ, on peut donc refaire la preuve S avec des variables qui n'apparaissent pas dans Δ on a suffisamment de variables (en effet dans $\mathcal{L}$, et comme dans tout langage du premier ordre, on a une infinité dénombrable de variables et vu que les nouvelles constantes n'apparaissent pas dans Δ on peut remplacer ces constantes par des variables) donc S' est une nouvelle preuve de la contradiction. Mais c'est également une preuve qu'on pouvait faire avec $\mathcal{L}$. Donc Δ est inconsistant pour $\mathcal{L}$. Ce qui est absurde.

Théorème de complétude

Solution pour le problème 2

Pour résoudre le problème du manque d'information, nous allons ajouter des formules à Δ pour qu'il contienne l'information nécessaire.

On va énumérer les formules $\phi_1, \phi_2, \ldots, \phi_n, \ldots$ du langage $\mathcal{L}'$ et construire des ensembles successifs $\Delta_0, \Delta_1, \ldots, \Delta_n, \ldots$.

Pour construire les Δ_i, on procède par induction :

Cas de base : $\Delta_0 = \Delta$

Cas d'induction : Supposons que $\Delta_1, \ldots, \Delta_{i-1}$ soient construits.

1. si $\Delta_{i-1} \cup \{\phi_i\}$ est consistant et ϕ_i n'est pas de la forme $\exists x \cdot \Psi$ alors $\Delta_i = \Delta_{i-1} \cup \{\phi_i\}$.
2. si $\Delta_{i-1} \cup \{\phi_i\}$ est consistant et ϕ_i est de la forme $\exists x \cdot \Psi$ alors

$\Delta_i = \Delta_{i-1} \cup \{\exists x \cdot \Psi, \Psi[c/x]\}$, avec c qui n'apparaît pas dans

$$\Delta_0, \ldots, \Delta_{i-1}$$

Intuitivement, c est un témoin pour x qui montre que $\exists x \cdot \Psi$ est vrai.

3. si $\Delta_{i-1} \cup \{\phi_i\}$ est inconsistante et ϕ_i n'est pas de la forme $\forall x \cdot \Psi$ alors $\Delta_i = \Delta_{i-1} \cup \{\neg\phi_i\}$.
4. $\Delta_{i-1} \cup \{\phi_i\}$ est inconsistant et ϕ_i est de la forme $\forall x \cdot \Psi$ alors $\Delta_i = \Delta_{i-1} \cup \{\neg\phi_i, \neg\Psi[c/x]\}$ avec c une constante qui n'apparaît pas dans $\Delta_0, \Delta_1, \ldots, \Delta_{i-1}$.

Théorème de complétude

On va maintenant montrer que les ensembles de formules Δ_i que l'on vient de construire sont tous consistants.

Lemme. $\forall\, i \geq 0 : \Delta_i$ est consistant.

Preuve. A nouveau, la preuve est par induction.

C.B. : $i = 0$ on sait $\Delta_0 = \Delta$ et donc Δ_0 est consistant.

C.I. : Supposons que Δ_{i-1} est consistant et montrons que Δ_i l'est également.

1. Si Δ_i est obtenu par le cas 1. Alors trivialement Δ_i est consistant.
2. Même preuve que pour le cas 4 ci-dessous..
3. Si Δ_i est obtenu par le cas 3 alors on sait que $\Delta_{i-1} \cup \{\phi_i\}$ est inconsistant $(\Delta_{i-1} \cup \{\phi_i\} \vdash \bot)$.

Par la règle $\boxed{\neg_i}$ on sait donc que $\Delta_{i-1} \vdash \neg\phi_i$. Par l'adéquation du système de preuve, on sait que $\Delta_{i-1} \cup \{\neg\phi_i\} = \Delta_i$ est consistant.

4. Considérons le cas 4, par définition on a que
   $\Delta_i = \Delta_{i-1} \cup \{\neg\forall\, x \cdot \Psi\} \cup \{\neg\Psi[c/x]\}$.
   Faisons l'hypothèse (raisonnement par l'absurde) que Δ_i est inconsistant. On sait également que $\Delta_{i-1} \cup \{\neg\forall\, x \cdot \Psi\}$ est consistant car on sait que $\Delta_{i-1} \cup \{\forall\, x \cdot \Psi\} \vdash \bot$ et donc la règle $\boxed{\neg_i}$ nous permet de d'affirmer $\Delta_{i-1} \vdash \neg\forall\, x \cdot \Psi$.
   L' inconsistance ne peut donc venir que de l'ajout de $\neg\Psi[c/x]$!
   Donc faisons l'hypothèse que $\Delta_{i-1} \cup \{\neg\Psi[c/x]\}$ est inconsistant.
   A nouveau par $\neg_i$, nous obtenons :

$$\Delta_{i-1} \vdash \Psi[c/x]$$

   Donc on a une preuve de Ψ où on a remplacé x par c, car lors de la construction de Δ_i on a utilisé un c qui n'apparaissait pas

dans $\Delta_0, \ldots, \Delta_{i-1}$.
On peut donc remplacer la constante c dans cette preuve par une nouvelle variable y.

$$\Delta_{i-1} \vdash \Psi[y/x]$$

où y est une nouvelle variable
En utilisant, $\boxed{\forall\, x_i}$ On obtient :

$$\Delta_{i-1} \vdash \forall\, y \cdot \Psi$$

y étant quantifiée, elle peut être renommée, ce qui donne :

$$\Delta_{i-1} \vdash \forall\, x \cdot \Psi$$

Or, on avait montré précédemment que :

$$\Delta_{i-1} \vdash \neg \forall\, x \cdot \Psi$$

On obtient donc une contradiction. Et donc l'hypothèse Δ_i non

consistant est fausse.

C.Q.F.D.

Théorème de complétude

Nous sommes maitenant en mesure de définir l'ensemble de formules Δ'.

$$\Delta' = \bigcup_{i \geq 0} \Delta_i$$

Quelles sont les propriétés de Δ' ?

1. Pour toute formule ϕ sur $\mathcal{L}'$ on a
   – soit $\phi \in \Delta'$
   – soit $\neg\phi \in \Delta'$
   On dit que Δ' est *complet*. En effet, par construction, pour chaque formule ϕ_i de l'énumération, on a toujours ajouté soit ϕ_i, soit $\neg\phi_i$.
2. Pour toute formule de la forme $\exists\, x \cdot \phi \in \Delta'$, on a une formule $\phi[c/x] \in \Delta'$: on dit que Δ' est *clos*.
3. De plus, Δ' est *consistant*. En effet, faisons l'hypothèse que Δ' est non consistant.

$$\Delta' \vdash \bot$$

Il existe donc une preuve $S = (\phi_1, \ldots, \phi_n)$ telle que $\phi_n \equiv \bot \equiv \Psi \wedge \neg\Psi$. La preuve a une longueur finie : on sait donc que pour chaque ϕ_i il existe un j tel que $\phi_i \in \Delta_j$, ce qui voudrait dire que $\Delta_j \vdash \bot$, ce qui est absurde puisque nous avons montré que la séquence des Δ_i est consistante.

Théorème de complétude

Nous sommes maintenant en position pour définir le modèle Δ.

Définissons tout d'abord le domaine d'interprétation M_Δ.

M_Δ va être défini à partir des termes de $\mathcal{L}'$. On a besoin d'une relation d'équivalence sur les termes de $\mathcal{L}'$:

$$t \equiv t' \text{ ssi } \ t = t' \in \Delta'$$

$\equiv$ est bien une relation d'équivalence.

En effet, nous avons déjà montré que les propriétés de l'égalité peuvent être établies dans le système de preuves :

- réflexive ($\vdash t_1 = t_1$)
- transitive ($t_1 = t_2 \wedge t_2 = t_3 \vdash t_1 = t_3$)
- symétrique ($t_1 = t_2 \vdash t_2 = t_1$)

On prend un représentant par classe d' équivalence :

$$t \rightarrow [t]$$

$[t]$ est le représentant de sa classe d'équivalence.

Le domaine d'interprétation est constitué des seuls représentants des classes d'équivalence :

$$M_{\Delta} = Term(\mathcal{L}') \ \mathbf{mod} \ \equiv$$

Théorème de complétude

Nous allons maintenant fournir l'interprétation μ_Δ des fonctions et prédicats.

1. Si f est une fonction d'arité k

$$f^{\mu_\Delta}([t_1], \ldots, [t_k]) = [f(t_1, \ldots, t_k)]$$

2. Si R est un prédicat d'arité k alors

$$R^{\mu_\Delta}([t_1], \ldots, [t_k]) \text{ ssi } \; R(t_1, \ldots, t_k) \in \Delta'$$

On a choisi de ne garder dans le domaine de M_Δ qu'un seul représentant par classe d' équivalence. On va montrer que la définition de f^{μ_Δ} et R^{μ_Δ} est indépendante du choix d'un représentant particulier :

Formellement, cela revient à montrer que :

$$\begin{array}{c} t_1 \equiv t'_1 \\ \text{si} \qquad \vdots \qquad \text{alors} \\ t_k \equiv t'_k \\ f(t_1, \ldots, t_k) \equiv f(t'_1, \ldots, t'_k) \end{array}$$

En effet, on sait par définition que

$$t_i \equiv t'_i \text{ ssi } \; t_i = t'_i \in \Delta'$$

On veut montrer que

$$\Delta' \vdash f(t_1, \ldots, t_k) = f(t'_1, \ldots, t'_k)$$

On peut obtenir cette preuve de la façon suivante :

$$\boxed{=_i} \quad \Delta' \quad \vdash f(t_1, \ldots, t_k) = f(t_1, \ldots, t_k)$$
$$\boxed{=_e} \quad \Delta' \quad \vdash f(t_1, \ldots, t_k) = f(t'_1, \ldots, t'_k)$$

et $t_i = t'_i \in \Delta'$

Ici on a traité le cas de base. Pour les termes de hauteur > 1, on utilise un raisonnement par induction et le même schéma de preuve.

On peut faire le même raisonnement pour les prédicats.

On a donc : si $t_1 \equiv t'_i$ alors

$$R(t_1, \ldots, t_k) \in \Delta' \text{ ssi } R(t'_1, \ldots, t'_k) \in \Delta'$$

Théorème de complétude

Nous avons défini l'interprétation $\mu_{\Delta'}$. On va maintenant prouver que :

$$\mu_\Delta \models \Delta'$$

On va montrer que l'interprétation μ_Δ est un modèle pour toute formule $\phi \in \Delta'$, i.e.

$$\mu_\Delta \models \phi.$$

On va raisonner par induction sur la structure de ϕ pour montrer que $\mu_\Delta \models \phi$.

C.B. : $\phi \equiv R(t_1, \ldots, t_n)$

Par définition de R^{μ_Δ} on a que

$$([t_1], \ldots, [t_n]) \in R^{\mu_\Delta} \text{ ssi } R(t_1, \ldots, t_n) \in \Delta'.$$

On a donc $\mu_\Delta \models R(t_1, \ldots, t_n)$.

C.I :

– $\phi \equiv \neg\Psi$, par hypothèse d'induction on a que $\Psi \notin \Delta'$ ssi $\mu_\Delta \not\models \Psi$, or on sait que $\neg\Psi \in \Delta'$ et donc $\Psi \notin \Delta'$ par construction de Δ' et donc $\mu_{\Delta'} \not\models \Psi$. On a donc établi :

$$\mu_\Delta \models \neg\Psi$$

– $\phi \equiv \Psi_1 \vee \Psi_2$ par hypothèse d'induction on a :

$$\Psi_1 \in \Delta' \text{ ssi } \mu_\Delta \models \Psi_1$$

$$\Psi_2 \in \Delta' \text{ ssi } \mu_\Delta \models \Psi_2$$

Or par définition de la sémantique de l'opérateur $\vee$, on a

également :

$$\mu_\Delta \models \phi \quad \text{ssi} \quad \mu_\Delta \models \Psi_1$$
$$\text{ou} \quad \mu_\Delta \models \Psi_2$$

Vu que $\Psi_1 \vee \Psi_2 \in \Delta$ et que l'on peut montrer que

$$\Psi_1 \vee \Psi_2, \neg\Psi_1 \vdash \Psi_2$$
$$et \quad \Psi_1 \vee \Psi_2, \neg\Psi_2 \vdash \Psi_1$$

On a donc soit $\Psi_1 \in \Delta'$ ou $\Psi_2 \in \Delta'$. (on se sert donc des règles syntaxiques pour établir cela).
Et donc, on a :

$$\mu_\Delta \models \Psi_1$$

, ou

$$\mu_\Delta \models \Psi_2$$

et donc, par sémantique de $\vee$,

$$\mu_\Delta \models \Psi_1 \vee \Psi_2$$

– $\phi = \forall x \cdot \Psi$

Montrons que $\phi \in \Delta'$ ssi $\Psi[t/x] \in \Delta'$ pour tout terme t. Par hypothèse d'induction on a :

$$\Psi[t/x] \in \Delta'$$

ssi

$$\mu_\Delta \models \Psi[t/x]$$

Donc si on prouve ce résultat, alors on obtient bien $\forall x \cdot \Psi \in \Delta'$ ssi $\mu_\Delta \models \forall\, x\, \Psi$ (sémantique du $\forall$). En effet, le domaine M_Δ ne contient que les termes du langage $\mathcal{L}'$ (plus exactement, les classes d'équivalence de cet ensemble de termes).

1. supposons que $\forall x \cdot \Psi \in \Delta'$ alors en utilisant la règle de preuve

(syntaxique) $\forall_e$:

$$\frac{\forall x \cdot \phi}{\phi[t/x]}$$

$$\Delta' \vdash \Psi[t/x] \text{ et } \Delta' \text{ est consistant}$$

Donc $\neg\Psi[t/x] \notin \Delta'$. Par complétude de Δ', $\Psi[t/x] \in \Delta'$ pour tout terme t. Par hypothèse d'induction :

$$\Delta' \models \Psi[t/x] \text{ pour tout terme } t$$

et donc par sémantique du $\forall$:

$$\mu_\Delta \models \forall x \cdot \Psi$$

2. supposons $\forall x \cdot \Psi \notin \Delta'$

   par complétude on sait que

   $$\neg\forall x \cdot \Psi \in \Delta'$$

   par construction de Δ' on a $\neg\Psi[c/x] \in \Delta'$ (témoin) et donc

$\Psi[c/x] \notin \Delta'$.

Par hypothèse d'induction on a

$$\mu_\Delta \not\models \Psi[c/x]$$

et donc par la sémantique du $\forall$ on a

$$\mu_\Delta \not\models \forall x \cdot \Psi$$

et donc

$$\mu_\Delta \models \neg\forall x \cdot \Psi$$

Théorème de complétude

Donc, pour toute formule $\phi \in \Delta'$, on a montré que

$$\mu_\Delta \models \phi$$

et donc, vu que $\Delta \subseteq \Delta'$, on a que pour toute formule $\phi \in \Delta$

$$\mu_\Delta \models \phi$$

et Δ a donc un modèle !

C.Q.F.D.

Théorème d'incomplétude (Goëdel)

En général lorsque l'on fait des preuves en mathématique, on fait des raisonnements informels.

Hilbert : Peut-on formaliser les mathématiques, c'est-à-dire prouver formellement tous les théorèmes des mathématiques ?

Goëdel va établir que c'est impossible !

Théorème d'incomplétude (Goëdel)

Soit μ un modèle mathématique (les graphes, l'arithmétique, ...) au sujet duquel on veut faire des raisonnements, par exemple démontrer un théorème formalisé par une formule ϕ. On veut donc établir :

$$\mu \models \phi$$

Remarque : vu que l'on fixe l'interprétation μ, chaque formule ϕ est soit vraie soit fausse.

Théorème d'incomplétude (Goëdel)

Pour faire des raisonnements formels sur μ, on utilise la *méthode axiomatique* . On parlera d'*axiomatisation* d'une théorie mathématique.

μ est **complètement axiomatisé** ("décrit") par Ψ (un ensemble de formules) si :

1. pour tout $\psi \in \Psi$, on a $\mu \models \psi$;
2. si $\mu \models \phi$ alors $\models \Psi \rightarrow \phi$, et donc (par le théorème de complétude, si Ψ et ϕ sont des formules du premier ordre) $\vdash \Psi \rightarrow \phi$, ce qui est équivalent à $\Psi \vdash \phi$.

Théorème d'incomplétude (Goëdel)

Est-ce que la méthode axiomatique est complète ? C'est-à-dire applicable à tout modèle mathématique μ ?

Malheureusement, la réponse est négative. C'est ce que nous apprend le théorème d'incomplétude de Goëdel.

Plus précisement, Goëdel montre qu'il ne peut pas exister d'axiomatisation complète (dans le sens précis que l'on vient de définir) de l'arithmétique (théorie mathématique somme toute assez simple).

Théorème d'incomplétude (Goëdel)

Pour formaliser l'arithmétique, nous utilisons le language

$$\mathcal{L}^N = (0^N, s, +^N, \times^N, <^N)$$

Voici quelques exemples de formules dans ce langage :

- $\forall x \cdot \exists y \cdot \neg(x > y)$
- $\forall x, y \cdot \exists d, x \cdot$
  $x \times d + r = y \wedge r < d \rightarrow x \times (d + 1) > y$

Théorème d'incomplétude (Goëdel)

On obtient l'interprétation standard, en interprétant les symboles de $\mathcal{L}^N$ de la façon suivante :

- la constante 0^N est interprétée par $0 \in N$;
- la fonction s (pour $s$uccesseur) est interprétée par la fonction $+1 \in N \to N$;
- la fonction $+^N$ est interprétée par la fonction d'addition sur les nombres naturels ;
- la fonction $\times^N$ est interprétée par la fonction de multiplication sur les nombres naturels ;
- la relation $<^N$ est interprétée par l'ordre "plus petit que" sur les nombres naturels.

Notons que, puisque nous avons fixé le modèle d'interprétation μ, on aura donc $\mu \models \phi$ ou $\mu \models \neg\phi$ pour toute formule close ϕ.

Théorème d'incomplétude (Goëdel)

Des ensembles d'axiomes ont été proposés pour formaliser l'arithmétique. Il y a par exemple les axiomes de l'arithmétique de **Peano** :

- $\forall x \cdot (s(x) \neq 0^N)$
- $\forall x \cdot \forall y \cdot (s(x) = s(y) \rightarrow x = y)$
- $\forall x \cdot (x = 0 \vee \exists y \cdot x = s(y))$
- ...

Plus un schéma d'axiomes pour l'induction :

$$
\begin{aligned}
&(\phi[0/x] \\
&\wedge \forall x \cdot \phi(x) \rightarrow \phi[s(x)/x]) \\
&\rightarrow \forall y \cdot \phi[y/x]
\end{aligned}
$$

Ces axiomes et ce schéma d'axiomes nous permettent de déduire certains théorèmes de l'arithmétique, mais pas tous.

Théorème d'incomplétude (Goëdel)

Enoncé du théorème de Goëdel :

Tout ensemble fini ou énumérable d'axiomes Δ pour l'arithmétique est tel que :

Si Δ est consistant
Alors Δ est incomplet.

Théorème d'incomplétude (Goëdel)

Comment Goëdel établit-il ce résultat ?

La preuve de ce théorème est très ardue mais l'idée principale est simple et élégante. C'est une application ingénieuse du procédé de la diagonale de Kantor.

L'idée principale est de créer une correspondance, appelée *numéro de Goëdel*, entre les nombres naturels et les objets logiques comme les formules et les preuves. A chaque formule et à chaque preuve (séquence finie de formules) est attribué un numéro, le *numéro de Goëdel*.

Cette numérotation peut à son tour être définie par des formules du langage $\mathcal{L}^N$ quand celui-ci est interprété dans le modèle standard. Ces formules, avec les axiomes de Peano, sont suffissantes pour définir cette numérotation.

Théorème d'incomplétude (Goëdel)

En particulier, on peut montrer qu'il existe une formule

$$\phi(i, j)$$

dont l'interprétation est la suivante.

Pour tout i, j, on a que :

$\phi(i, j)$ est vrai

ssi

- i est le numéro de Goëdel d'une formule $\rho(x)$ avec une variable libre x ;
- j est le numéro de Goëdel de la preuve de $\rho[i/x]$.

Théorème d'incomplétude (Goëdel)

Lemme de complétude partielle pour les formules sans variable :

Lemme : si ψ est une formule valide de l'arithmétique sans variable, alors ψ peut-être dérivée grâce aux axiomes de Peano.

En particulier, si $\phi(n, m)$ est vraie, alors

$$\vdash_{Peano} \phi(n, m).$$

Théorème d'incomplétude (Goëdel)

Considérons maintenant la formule β suivante :

$$\beta \equiv \forall y \cdot \neg\phi(x, y)$$

Cette formule a une variable libre x et nous noterons son numéro de Goëdel m.

Maintenant, considérons la formule α définie comme suit :

$$\alpha \equiv \beta[m/x] \equiv \forall y \cdot \neg(\phi(m, y))$$

α exprime donc "Il n'y a pas de preuve de α".

Théorème d'incomplétude (Goëdel)

On établit maitenant le théorème suivant :

Théroème : Si Δ est *consistant*, alors :

$$\Delta \nvdash \alpha$$

et

$$\Delta \nvdash \neg\alpha$$

(et donc Δ est incomplet).

La preuve est établie par *contradiction*.

Supposons que $\boxed{\Delta \vdash \alpha}$. Notons par n le numéro de Goëdel de cette preuve. Par définition on a que $\phi(m, n)$ est vraie. Vu que $\phi(m, n)$ est sans variable, par le théorème de complétude partielle on a :

$$\Delta \vdash \phi(m, n)$$

or $\alpha \equiv \forall y \cdot \neg\phi(m, y)$; en utilisant la règle $\forall_e$, et l'hypothèse $\Delta \vdash \alpha$, je peux déduire :

$$\Delta, \alpha \vdash \neg\phi(m, n)$$

et donc

$$\boxed{\Delta \vdash \neg\phi(m, n)}$$

Ce qui contredit la consistance de Δ.

Théorème d'incomplétude (Goëdel)

Supposons maintenant que $\boxed{\Delta \vdash \neg\alpha}$. On a donc

$$\Delta \models \exists y \cdot \phi(m, y)$$

Notons n le nombre qui rend $\phi(m, y)$ vrai. Donc $\phi(m, n)$ est vrai. Cette formule signifie que la formule α a une preuve de numéro n et donc $\Delta \vdash \alpha$. Ce qui contredit l'hypothèse de consistance de Δ si l'on suppose $\Delta \vdash \neg\alpha$.

[

Index

]