LINGI1101

Logique et Structures Discrètes

Titulaire : Peter Van Roy

2014

Table des matières

Remerciements

Je tiens à remercier les étudiants de LINGI1101 pour avoir pris des notes pendant mon cours, ce qui faisait la base de ce syllabus. Les contributeurs sont : Antoine Walsdorff, Goeric Huybrechts, Romane Schelkens, Nicolas Van Wallendael, Kilian Verhetsel, Cyril de Vogelaere, Jonathan Legat.

Introduction au cours LINGI1101

Le cours *Logique et structures discrètes* a deux buts importants :

— Donner la motivation et l'intuition de la logique, pour que cette matière devienne véritablement utile pour les étudiants.
— Donner les concepts et les formalismes mathématiques nécessaires pour utiliser la logique à bon escient.

L'intuition est donc importante pour ce cours, mais néanmoins la connaissance des formalismes mathématiques reste essentielle. Le cours sera coté sur les deux : intuitions (un tiers) et formalismes (deux tiers).

Déroulement du cours

Le cours est composé de deux parties. La première partie, *logique formelle*, représentera deux tiers du cours. La seconde partie, *structures discrètes sur Internet*, comptera quant à elle pour un tiers du cours.

L'évaluation de ce cours se compose de trois parties. Il y aura tout d'abord une interrogation au milieu du quadrimestre portant sur 5 points. Il vous sera également demandé de prendre note pendant une heure de cours par groupe de trois, ceci afin de contribuer au syllabus. Ces notes prises au cours rapporteront au maximum 2 points de la note finale à chacun des participants. L'examen sera divisé en deux parties. La première partie sur 5 points portera sur la matière de l'interrogation. La note retenue sera le maximum entre la note de l'interrogation et celle obtenue à la question de l'examen. La seconde partie de l'examen sera donc cotée sur 13 points et portera sur le reste de la matière.

Afin de suivre ce cours, nous nous baserons sur deux livres de références

correspondants chacun à une partie du cours :

— Introductory Logic and Sets for Computer Scientists, by *Nimal Nissanke.*
— Networks, Crowds, and Markets : Reasoning About a Highly Connected World, by *David Easley and Jon Kleinberg.* [1]

La première partie sera complétée par des sujets et exercices plus avancés qui approfondissent le traitement du livre.

Plan du cours

Cette partie va parler du rôle des raisonnements et des différentes formes de raisonnement. Nous prendrons en exemple la méthode scientifique.

Logique des propositions

La logique des propositions est un langage formel constitué d'une syntaxe et d'une sémantique. La syntaxe décrit l'ensemble des formules qui appartiennent au langage. La sémantique permet de donner un sens aux formules de langage. C'est une logique très ancienne qui vient de l'antiquité.

Logique des prédicats

C'est la logique la plus expressive et la plupart des travaux mathématiques peuvent être écrits dans ce langage. Elle est aussi définie comme la logique du premier ordre. [2] En logique des prédicats, les éléments de base du langage ne sont plus des propositions mais des prédicats.

Interprétations et modèles

La logique a besoin d'un langage, de phrases pour la décrire. Cette section couvrira donc la sémantique à utiliser.

Théorie de preuve

Nous pouvons manipuler une phrase en logique pour obtenir un résultat. Par exemple, si A et B sont vrais, nous pouvons en déduire que A est vrai. Il y a des règles d'inférences à utiliser pour prendre une phrase en logique et en

1. Quelques chapitres.
2. Il existe d'autres formes de logiques plus expressives mais plus difficiles à utiliser. Exemple : la logique du deuxième ordre, ...

déduire une autre. Une preuve mathématique est une séquence de phrases liées par des règles d'inférences.

Algorithme de preuve

C'est l'algorithme le plus puissant qui existe en logique des prédicats. Néanmoins, il est inefficace seul. Afin de le rendre efficace, il faut poser des hypothèses. Nous approfondirons ce problème dans le cadre de cette section.

Théorie logique

Il est possible de formaliser tout objet mathématique avec une théorie logique qui lui est propre. En exemple, citons la théorie des ensembles, des fonctions et des ordres partiels.

Programmation logique

Le rêve serait de pouvoir exprimer toute chose logique en langage de programmation efficace. Il s'agira d'appliquer ce principe avec l'algorithme de preuve, sur base d'hypothèses.

Première partie

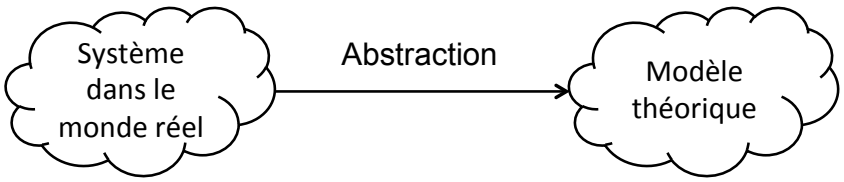
logique formelle

Chapitre 1

Contexte : la méthode scientifique

1.1 Formalisation d'un système

Comment pouvons-nous formaliser un système dans le monde réel tel que les champs magnétiques ou la gravitation ?

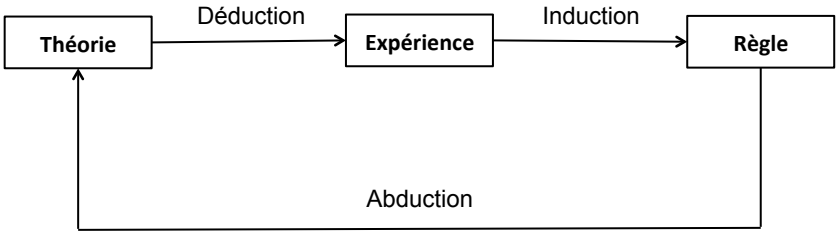


Afin de formaliser un système dans le monde réel, nous devons faire une abstraction vers un modèle théorique. Ce modèle théorique est un ensemble de phrases logiques dont il est possible de tirer des prédictions. Il n'est intéressant que s'il se comporte comme le vrai système.

Un exemple de cette formalisation pourrait être les équations de Maxwell qui sont le modèle théorique correspondant à la trajectoire des électrons et protons.

1.2 Boucle de raisonnement

Il existe trois formes de raisonnement : la déduction, l'induction et l'abduction. Ces trois formes de raisonnement peuvent être liées dans une boucle de raisonnement de la façon suivante :



1.2.1 Déduction

Il s'agit de faire des calculs et des raisonnements logiques par rapport à la théorie. On en conclut une expérience.

1.2.2 Induction

L'induction est le fait de trouver une règle générale qui décrit les résultats d'expériences. On choisit en général une règle moyenne qui deviendra la règle générale. Il faut souligner que les résultats expérimentaux ne sont pas totalement fiables ou incomplets. Dès lors, la règle trouvée n'est pas nécessairement exacte. Par exemple, si par induction nous avons trouvé la règle, "les oiseaux volent", cela est vrai tant que l'on a pas vu un pingouin. Autre exemple, nous pouvons supposer que demain le soleil va se lever comme depuis des milliers d'années, même si rien ne l'assure.

1.2.3 Abduction

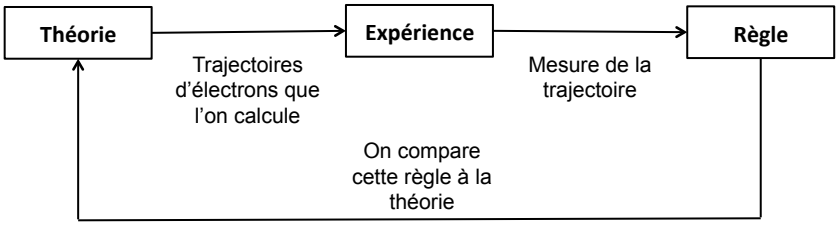
On compare la règle générale trouvée lors de l'induction avec la théorie. Si cela se révèle différent, comme l'on suppose que tout est parfait dans les calculs/expériences, c'est la théorie qui est fausse. Il faut alors corriger la théorie existante ou en inventer/deviner une nouvelle. Ce type de raisonnement est l'abduction. On applique l'abduction couramment dans notre vie de tous les jours ; par exemple, lorsqu'un élève entre trempé dans la classe, nous supposons qu'il pleut dehors.

Sur ces trois concepts seul la déduction est un raisonnement sûr, les deux autres sont encore mal compris. Nous nous focaliserons dans ce cours uniquement sur la déduction.

1.3 Exemples

1.3.1 Loi de Maxwell

Nous illustrons dès à présent le fonctionnement de la boucle de raisonnement à l'aide de l'exemple cité plus haut, c'est-à-dire les équations de Maxwell :



Par déduction, grâce à la théorie et aux conditions initiales que nous fixons, nous trouvons la trajectoire d'un électron. Nous effectuons ensuite des mesures dans le monde réel. Nous allons, par exemple, mesurer la trajectoire plusieurs fois avec des méthodes différentes et, par induction, nous trouvons une règle qui est la loi de comportement de la particule. Nous comparons ensuite, par abduction, cette règle à la théorie, et nous la corrigeons si besoin.

1.3.2 Sac de billes

Afin d'illustrer les 3 formes de raisonnement de manière plus formelle, considérons un sac de billes pouvant contenir des billes noires ou blanches. Notons que sac(x) signifie "la bille x est dans le sac" et que blanc(x) signifie "la bille x est blanche".

Déduction

1. Règle : $\forall\ x,\ sac(x) \Rightarrow blanc(x)$
2. Cas : $sac(a),\ sac(b),\ \cdots$

---

3. Résultat : $blanc(a),\ blanc(b),\ \cdots$

Si toutes les billes se trouvant dans le sac sont blanches et que l'on pioche une bille de ce sac, cette bille sera blanche. Cette déduction est forcément correcte.

Induction

1. Cas : $sac(a)$, $sac(b)$,$\cdots$
2. Résultat : $blanc(a)$, $blanc(b)$, $\cdots$

---

3. Règle : $\forall\ x$, $sac(x) \Rightarrow blanc(x)$

Si toutes les billes qu'on l'on pioche du sac sont blanches, alors nous pouvons établir comme règle que toutes les billes dans le sac sont blanches. Cette induction n'est pas forcément correcte.

Abduction

1. Règle : $\forall\ x$, $sac(x) \Rightarrow blanc(x)$
2. Résultat : $blanc(a)$, $blanc(b)$, $\cdots$

---

3. Cas : $sac(a)$, $sac(b)$, $\cdots$

Si toutes les billes se trouvant dans le sac sont blanches et que nous trouvons des billes blanches à côté du sac, nous pouvons penser qu'elles viennent du sac. Cette abduction n'est pas forcément correcte.

Chapitre 2

La logique propositionnelle

> « *They don't even need to know what they're talking about.* » — Richard Feynman à propos des mathématiciens.

La logique propositionnelle est la plus simple des formes de logique. Elle permet de formaliser des expressions telles que les suivantes :

1. « S'il fait beau, alors je vais dehors. »
2. « Cet homme est grand et fort. »
3. « Il fait jour mais pas nuit. »

Plus précisément, nous partons d'un ensemble de propositions premières. Par exemple :

1. il fait beau ;
2. je vais dehors ;
3. cet homme est grand ;
4. cet homme est beau ;
5. il fait jour ;
6. il fait nuit.

Le contenu exact de ces propositions n'a en fait aucune importance. C'est pourquoi elles seront remplacées par des lettres majuscules :

1. A = « il fait beau » ;
2. B = « je vais dehors » ;
3. C = « cet homme est grand » ;
4. D = « cet homme est beau » ;
5. E = « il fait jour » ;

6. F = « il fait nuit ».

Une proposition logique est alors :

— soit une des propositions premières ;
— soit une combinaison de propositions logiques connectées par des connecteurs logiques.

Ainsi, les exemples de propositions précédentes peuvent être réécrites comme ceci (la signification précise des différents symboles sera décrite plus loin) :

1. $A \Rightarrow B$;
2. $C \wedge D$;
3. $E \wedge \neg F$.

L'avantage de cette notation par rapport aux phrases en français est qu'elle nous permet d'effectuer des raisonnements formels sur les propositions logiques. En particulier, nous pouvons précisément définir :

1. une **grammaire** qui définit ce qui est une proposition logique et ce qui ne l'est pas ;
2. une **sémantique** qui donne un sens à chaque proposition logique ;
3. une **théorie de preuve** permettant, en sachant qu'une proposition est vraie, de trouver d'autres propositions vraies (par exemple à partir de $A \Rightarrow B$ on peut trouver $\neg B \Rightarrow \neg A$).

2.1 La grammaire

La logique propositionnelle est un **langage** formel. Il peut être défini à l'aide d'une grammaire sur un **alphabet**. L'alphabet est l'ensemble des symboles qui composent une proposition logique, c'est-à-dire :

— les lettres majuscules représentant les différentes propositions premières : A, B, C, etc. ;
— true et false qui représentent des propositions qui sont respectivement toujours vraies et toujours fausses ;
— les différents connecteurs logiques :
  — Conjonction : $\wedge$
  — Disjonction : $\vee$
  — Négation : $\neg$
  — Implication : $\Rightarrow$
  — Equivalence : $\Leftrightarrow$
— les caractères de ponctuation « (» et «) ».

Cependant, toutes les séquences composées de ces caractères ne sont pas des phrases propositionnelles. La grammaire suivante permet de donner les règles que les phrases propositionnelles doivent respecter :

$$
\begin{aligned}
\text{<identificateur>} &= A \mid B \mid C \mid D \mid \ldots \\
\text{<proposition>} &= \text{true} \\
&\mid \text{false} \\
&\mid \text{<identificateur>} \\
&\mid (\text{<proposition>}) \\
&\mid \neg\text{<proposition>} \\
&\mid \text{<proposition>} \wedge \text{<proposition>} \\
&\mid \text{<proposition>} \vee \text{<proposition>} \\
&\mid \text{<proposition>} \Rightarrow \text{<proposition>} \\
&\mid \text{<proposition>} \Leftrightarrow \text{<proposition>}
\end{aligned}
$$

Remarquez que seules les séquences de symboles qui respectent cette grammaire sont des phrases propositionnelles. Ainsi, $p \Leftrightarrow q$ n'est pas une phrase propositionnelle parce que les propositions premières sont des lettres majuscules ; de même les phrases en français — ou en klingon, ou encore dans d'autres formalismes mathématiques – comme « s'il fait beau alors je vais dehors » ne respectent pas la grammaire précédente et ne sont donc pas des phrases propositionnelles.

Notez aussi que le discours à propos des phrases propositionnelles n'est pas une phrase propositionnelle. La grammaire précédente, la description de l'alphabet, et cette explication parlent de propositions logiques sans en être, et font donc partie de ce qui est appelé le **métalangage**.

2.2 Les tables de vérité

La grammaire permet de définir précisément l'ensemble des phrases propositionnelles, mais pas de leur donner un sens, c'est à dire de définir quand une proposition est vraie ou fausse.

Pour cela, il faut décider lesquelles des propositions premières sont vraies et lesquelles sont fausses. Rappelez-vous que la signification des propositions premières n'a aucune importance, ce choix est donc complètement arbitraire (le choix qui décrit au mieux le monde réel n'est qu'un des choix possibles parmi tous les autres). On sait également que les propositions true et false sont, respectivement, toujours vraies et toujours fausses.

Les autres propositions sont construites à partir de propositions plus simples. Leur véracité est fonction de celle des propositions qui les composent. Prenons par exemple $p \wedge q$, où p et q sont d'autres propositions. La véracité de $p \wedge q$ est une fonction de celle de p et de q : $p \wedge q$ est vrai si et seulement si p et q sont vrais aussi ($\wedge$ est un « et » logique). Cette relation

peut être exprimée à l'aide de la table de vérité suivante :

p	q	$p \wedge q$
true	true	true
true	false	false
false	true	false
false	false	false

Voici la table de vérité des autres connecteurs logiques :

p	q	$p \vee q$
true	true	true
true	false	true
false	true	true
false	false	false

p	q	$p \Leftrightarrow q$
true	true	true
true	false	false
false	true	false
false	false	true

p	$\neg p$
true	false
false	true

p	q	$p \Rightarrow q$
true	true	true
true	false	false
false	true	true
false	false	true

Remarquez que le dernier tableau est correct. Il n'est parfois pas intuitif que la proposition $A \Rightarrow B$ (qui pourrait s'exprimer en français par « si A, alors B ») soit toujours vraie quand A est faux, mais c'est pourtant le cas : la proposition dit que quand A est vrai, B doit l'être aussi, mais elle ne donne aucune information sur le cas où A est faux.

Les parenthèses, quant à elles, servent à distinguer des propositions telles que $A \wedge (B \vee C)$ et $(A \wedge B) \vee C$, qui s'écriraient de la même manière sans parenthèses alors qu'elles n'ont pas la même table de vérité :

A	B	C	$A \wedge (B \vee C)$	$(A \wedge B) \vee C$
true	true	true	true	true
true	true	false	true	true
true	false	true	true	true
true	false	false	false	false
false	true	true	false	true
false	true	false	false	false
false	false	true	false	true
false	false	false	false	false

2.3 Les interprétations

Une autre façon plus puissante de définir si une proposition est vraie ou fausse est d'utiliser une **interprétation**. Si E_P est l'ensemble des propositions premières, alors une interprétation I définit la fonction $\text{val}_I : E_P \to \{\text{true}, \text{false}\}$ [1] qui permet de savoir si ces propositions premières sont vraies ou fausses. Par exemple, on pourrait écrire ceci :

$$\text{val}_I(A) = \text{true}$$
$$\text{val}_I(B) = \text{false}$$
$$\text{val}_I(C) = \text{true}$$

Étant donné la fonction val_I, il est possible de définir la fonction $\text{VAL}_I : P \to \{\text{true}, \text{false}\}$, qui est une extension de val_I à P, l'ensemble de toutes les propositions. L'équivalent des tables de vérité pourrait être des expressions telles que :

$$\forall p \in E_P . \text{VAL}_I(p) = \text{val}_I(p)$$

$$\text{VAL}_I(p \wedge q) = \begin{cases} \text{true si VAL}_I(p) \text{ et VAL}_I(q) = \text{true} \\ \text{false sinon} \end{cases}$$

$$\text{VAL}_I(p \vee q) = \begin{cases} \text{false si VAL}_I(p) \text{ ou VAL}_I(q) = \text{false} \\ \text{true sinon} \end{cases}$$

$$\text{VAL}_I(\neg p) = \begin{cases} \text{false si VAL}_I(p) = \text{true} \\ \text{true si VAL}_I(p) = \text{false} \end{cases}$$

1. La notation $f : A \to B$ signifie que f est une fonction depuis l'ensemble A vers l'ensemble B.

$$\mathrm{VAL}_I(p \Leftrightarrow q) = \begin{cases} \text{true si } \mathrm{VAL}_I(p) = \mathrm{VAL}_I(q) \\ \text{false sinon} \end{cases}$$

$$\mathrm{VAL}_I(p \Rightarrow q) = \begin{cases} \text{false si } \mathrm{VAL}_I(q) = \text{false alors que } \mathrm{VAL}_I(p) = \text{true} \\ \text{true sinon} \end{cases}$$

Prenons un exemple concret, utilisons une interprétation pour étudier la phrase suivante : « S'il fait beau à midi, j'irai promener le chien ». Nous devons d'abord traduire cette phrase en l'une des proposition du formalisme que nous avons défini, en commençant par identifier les propositions premières :

1. B = « Il fait beau » ;
2. M = « Il est midi » ;
3. P = « Je vais promener le chien » ;

Identifions également les connecteurs à employer : « s'il fait beau $\wedge$ qu'il est midi $\Rightarrow$ j'irai promener le chien ». En combinant ces deux résultats, nous obtenons la phrase propositionnelle $(B \wedge M) \Rightarrow P$.

Interprétons désormais notre proposition à l'aide de l'interprétation suivante :

1. Il fait beau : $\mathrm{val}_I(B) = \text{true}$;
2. Il est midi : $\mathrm{val}_I(M) = \text{true}$;
3. Je n'irai pas promener le chien : $\mathrm{val}_I(P) = \text{false}$.

Nous pouvons alors effectuer le développement suivant :

$$\begin{aligned} \mathrm{VAL}_I((B \wedge M) \Rightarrow P) &= \begin{cases} \text{false si } \mathrm{VAL}_I(P) = \text{false alors que } \mathrm{VAL}_I(B \wedge M) = \text{true} \\ \text{true sinon} \end{cases} \\ \text{or } \mathrm{VAL}_I(B \wedge M) &= \begin{cases} \text{true si } \mathrm{VAL}_I(B) \text{ et } \mathrm{VAL}_I(M) = \text{true} \\ \text{false sinon} \end{cases} \\ &= \text{true} \\ \text{donc } \mathrm{VAL}_I((B \wedge M) \Rightarrow P) &= \begin{cases} \text{false si } \mathrm{VAL}_I(P) = \text{false} \\ \text{true sinon} \end{cases} \\ &= \text{false} \end{aligned}$$

Nous pouvons donc conclure que, dans cette interprétation, la personne ayant fait cette affirmation a menti. Notons néanmoins que notre homme n'aurait pas mentit en partant promener le chien alors qu'il pleuvait à midi, rien n'ayant été dit sur ce qu'il ferait dans le cas où il ne ferait pas beau.

2.4 Les modèles logiques

À partir de la notion d'interprétation, nous pouvons définir ce qu'est un **modèle**. Soit $B = \{b_1, b_2, \ldots, b_n\}$ un ensemble de propositions logiques. Une interprétation I est un modèle de B si et seulement si $\forall b_i \in B.\mathrm{VAL}_I(b_i) =$ true. Autrement dit, I décrit un univers qui respecte toutes les règles se trouvant dans l'ensemble B.

Donc, dans notre exemple précédent, l'interprétation choisie n'est pas un modèle de la proposition analysée, celle-ci n'étant pas validée, mais par exemple l'interprétation telle que $\mathrm{val}_I(B) =$ true, $\mathrm{val}_I(M) =$ true et $\mathrm{val}_I(P) =$ true est bien un modèle de la proposition utilisée comme exemple. Remarquez aussi que cela ne change rien au fait que l'interprétation choisie est le modèle d'autres propositions que celle étudiée (par exemple $B \wedge M$).

Les tautologies : Pour certaines propositions, toute interprétation est un modèle, c'est à dire que ces propositions sont toujours vraies. Par exemple, true est évidemment une tautologie, de même que $A \Rightarrow A$ ou encore $A \vee \neg A$. Le fait qu'une proposition p est une tautologie se note $\vDash p$.

Les contradictions : Pour d'autres propositions, il n'existe aucun modèle, c'est à dire qu'elles sont toujours fausses. Par exemple $A \wedge \neg A$ est une contradiction. Le fait qu'une proposition p est une contradiction se note $\nvDash p$.

Les contingences : Toutes les autres propositions sont des contingences. Il existe des interprétations qui sont des modèles et d'autres qui n'en sont pas. Par exemple $A \wedge B$ est vrai pour l'interprétation I telle que $\mathrm{val}_I(A) = \mathrm{val}_I(B) =$ true, mais faux dans tous les autres cas.

2.4.1 Conséquence logique

p est conséquence logique de q si et seulement si $p \Rightarrow q$ est une tautologie. En d'autres termes, si

$p \models q$	q est valide dans tous les modèles de p
alors	
$\models (p \Rightarrow q)$	$p \Rightarrow q$ est une tautologie.
On peut donc écrire	
$p \Rrightarrow q$	p est conséquence logique de q.

Cependant, la conséquence logique ($\Rrightarrow$) n'est pas une proposition logique (cf. syntaxe d'une proposition).

2.4.2 Équivalence logique

Par le raisonnement ci-dessus, on peut dire que p est logiquement équivalent à q si et seulement si

$p \models q$	q est valide dans tous les modèles de p
$q \models p$	p est valide dans tous les modèles de q

et donc

$\models (p \Rightarrow q)$	$p \Rightarrow q$ est une tautologie et
$\models (p \Rightarrow q)$	$p \Rightarrow q$ est une tautologie.

On peut donc écrire

$p \Leftrightarrows q$	p est conséquence logique de q.

L'équivalence logique n'est pas non plus une proposition logique.

Il ne faut pas non plus oublier la différence entre phrase propositionnelle (p, q, s,...) et propositions primaires (P, Q, S,...) (cf. syntaxe d'une proposition) :

$p \Rightarrow q$	n'est pas une proposition
mais	
$P \wedge Q \Rightarrow R \wedge \neg S$	en est bien une.

Chapitre 3

Les preuves

Une preuve est une manière de partir d'une formule et de dire si celle-ci est vraie ou fausse.
il y a 3 approches possibles :

- Table de vérité
- Preuve transformationnelle
- Preuve déductive (la plus générale)

3.1 Preuve avec table de vérité

Prouvons que $\neg(P \wedge Q) \Leftrightarrow (\neg P \vee \neg Q)$ est vrai :

P	Q	$\neg P$	$\neg Q$	$(\neg P \vee \neg Q)$	$P \wedge Q$	$\neg(P \wedge Q)$
F	F	T	T	***T***	F	***T***
T	F	F	T	***T***	F	***T***
F	T	T	F	***T***	F	***T***
T	T	F	F	***F***	T	***F***

On voit bien ici, que la table de $\neg(P \wedge Q)$ est équivalente à celle de $\neg(P \wedge Q)$.
La preuve a donc vérifié la véracité de la proposition.
Il y a néanmoins un problème avec cette preuve car s'il y a N propositions premières, on a 2^N lignes dans la table.

3.2 Preuve transformationnelle

On va utiliser des "Lois", c'est-à-dire des équivalences.

$p \Lleftrightarrow p \vee p$	Idempotence
$p \vee q \Lleftrightarrow q \vee p$	Commutativité
$(p \vee q) \vee r \Lleftrightarrow p \vee (q \vee r)$	Associativité
$\neg\neg p \Lleftrightarrow p$	Double Négation
$p \Rightarrow q \Lleftrightarrow \neg p \vee q$	Implication
$\neg p \wedge q \Lleftrightarrow \neg p \vee \neg q$	1^{ere} loi de De Morgan
$p \Leftrightarrow q \Lleftrightarrow (p \Rightarrow q) \wedge (q \Rightarrow p)$	Équivalence

On y ajoute 2 règles supplémentaires pour cette technique :

Transitivité de l'équivalence

Si $p \Lleftrightarrow q$ et $q \Lleftrightarrow r$, alors $p \Lleftrightarrow r$.

Substitution

Il est autorisé de remplacer une formule par une formule équivalente à l'intérieur d'une autre formule. Autrement dit :

Soit p,q,r des formules propositionnelles.

Si $p \Leftrightarrow q$ et $r(p)$, alors $r(p) \Lleftrightarrow r(q)$.

On peut remplacer p par q car elles sont équivalentes. Il faut faire attention à ces règles, car elles sont en méta-langage. Elles expliquent seulement ce qui est autorisé faire.

Exemple

On veut prouver : $p \wedge (q \wedge r) \Lleftrightarrow (p \wedge q) \wedge r$

$$
\begin{aligned}
p \wedge (q \wedge r) \quad & \Lleftrightarrow p \wedge \neg\neg(q \wedge r) \\
& \Lleftrightarrow p \wedge \neg(\neg q \vee \neg r) \\
& \Lleftrightarrow \neg\neg(p \wedge \neg(\neg q \vee \neg r)) \\
& \Lleftrightarrow \neg(\neg p \vee \neg\neg(\neg q \vee \neg r)) \\
& \Lleftrightarrow \neg(\neg p \vee (\neg q \vee \neg r)) \\
& \Lleftrightarrow \neg((\neg p \vee \neg q) \vee \neg r) \\
& \vdots \\
& \text{effectuer les mêmes lois dans le sens contraire} \\
& \vdots \\
& \Lleftrightarrow (p \wedge q) \wedge r
\end{aligned}
$$

Le problème dans cette méthode de preuve c'est qu'il faut de l'idée, de la créativité et donc ce n'est pas forcément mieux que les tables de vérités, surtout si c'est compliqué de trouver l'astuce.

3.3 Preuve déductive

A partir des lois logiques (lois d'équivalence) et des règles d'inférence, On va construire des preuves.
Une preuve est un objet formel défini avec précision.

Lois de logique :

$p \Longleftrightarrow p \vee p$	Idempotence de $\vee$
$p \vee q \Longleftrightarrow q \vee p$	Commutativité de $\vee$
$(p \vee q) \vee r \Longleftrightarrow p \vee (q \vee r)$	Associativité de $\vee$
$\neg\neg p \Longleftrightarrow p$	Double Négation
$p \Rightarrow q \Longleftrightarrow \neg p \vee q$	Implication
$\neg p \wedge q \Longleftrightarrow \neg p \vee \neg q$	1^{ere} loi de De Morgan
$\neg p \vee q \Longleftrightarrow \neg p \wedge \neg q$	2^{eme} loi de De Morgan
$(p \wedge q) \vee r \Longleftrightarrow (p \vee r) \wedge (q \vee r)$	Distributivité de $\vee$

Mais également l'idempotence, la commutativité, l'associativité et la distributivité de $\wedge$.

Règle d'inférence :

A la différence de la preuve transformationnelle, les règles d'inférences ont une direction : Elles commencent par les prémisses et se terminent par la conclusion.

Conjonction : $\dfrac{\begin{array}{ll} \text{p} & \text{prémisse} \\ \text{q} & \text{prémisse} \end{array}}{\begin{array}{ll} p \wedge q & \text{Conclusion} \end{array}}$

Simplification : $\dfrac{p \wedge q}{p}$

Addition : $\dfrac{p}{p \vee q}$

Contradiction : $\dfrac{\begin{array}{l} p \\ \neg p \end{array}}{q}$

Double Négation : $\dfrac{\neg\neg p}{p}$

Transitivité de l'équivalence : $\dfrac{\begin{array}{l} p \Leftrightarrow q \\ q \Leftrightarrow r \end{array}}{p \Leftrightarrow r}$

Modus Ponens : $\dfrac{\begin{array}{c} p \Rightarrow q \\ p \end{array}}{q}$

Modus Tollens : $\dfrac{\begin{array}{c} p \Rightarrow q \\ \neg p \end{array}}{\neg q}$

Loi d'équivalence : $\dfrac{p \Leftrightarrow q}{q \Leftrightarrow p}$

On y ajoute 2 régles spéciales :
— Le théorème de déduction,
— La preuve par contradiction.

Théorème de déduction

Pour prouver $s \Rightarrow t$, On suppose s vrai. On déduit t et donc on sait que l'hypothèse $s \Rightarrow t$ est vraie et on l'évacue.

On note ce théorème $s \vdash t$ (t peut être prouvé à partir de s, ce qui signifie qu'on peut construire une preuve (objet mathématique) qui est vraie si on applique bien les règles ci-dessus).

Remarque : $p \models t \neq p \vdash t$
— $p \models t$ est une notion de vérité, et donc de sémantique,
— $p \vdash t$ est une notion syntaxique, on peut prouver qu'une preuve est vraie.

Notion de prouvabilité : on peut créer une preuve, et donc on peut créer une séquence de prémisses et de conclusions.

$$\frac{p, ..., r, s \vdash t}{p, ..., r \vdash s \Rightarrow t}$$

Ceci permet notamment de formaliser des théorèmes.

Preuve par contradiction

C'est une preuve indirecte :
Implicitement, on suppose que p jusqu'à q n'a pas de problème, c'est-à-dire qu'on ne peut pas prouver une contradiction pour ces formules propositionnelles.

$$\frac{p, ..., q, r \vdash s \quad p, ..., q, r \vdash \neg s}{p, ..., q \vdash \neg r}$$

Ceci signifie qu'il y a une erreur dans les prémisses, et on suppose ici que c'est la formule propositionnelle r qui est fautive. On justifie qu'il n'y a aucune contradiction dans $p, ..., q$ car on part du principe qu'il existe un modèle de $p, ...q$.

Tout ceci est une formalisation de choses qu'on connaît déjà, mais ça nous donne une notion précise du raisonnement à avoir.

3.4 Exemple de preuve propositionnelle

Voici les propositions premières :
A : tu manges bien
B : ton système digestif est en bonne santé
C : tu pratiques une activité physique régulière
D : tu es en bonne forme physique
E : tu vis longtemps

On peut maintenant faire une théorie et on espère qu'elle aura un modèle.
A⇒B, C⇒D, B∨D ⇒ E, ¬E
On aimerait prouver que ¬A ∧ ¬C est vrai.

Preuve :

1. A⇒B	prémisse
2. C⇒D	prémisse
3. B∨D ⇒E	prémisse
4. ¬E	prémisse
5. A	hypothèse
6. B	modus ponens (1)
7. B∨D	addition (6)
8. E	modus ponens (7)
9. ¬A	preuve indirecte
10. C	hypothèse
11. D	modus ponens (2)
12. D∨B	addition (11)
13. B∨D	commutativité (12)
14. E	modus ponens (9)
15. ¬C	preuve indirecte
16. ¬A ∧ ¬C	conjonction (9,15)

Grâce à la déduction on a donc pu prouver que tu ne manges pas bien et que tu ne pratiques pas d'activité physique régulière. On aimerait maintenant pouvoir automatiser les preuves quand elles existent. Mais il faut savoir s'il peut tout résoudre ou pas. On va donc construire un algorithme nous permettant de résoudre automatiquement les preuves en logique propositionnelle.

3.5 Deux règles plus sophistiquées

3.5.1 Théorème de déduction

— Pour prouver s ⇒ t
— On suppose s vrai
— On déduit t
— Ensuite on évacue l'hypothèse

Notation : $p \vdash t$ (on peut prouver t à partir de p)

Prémisse :

$$\frac{p.....r, s \vdash t}{p.....r, s \vdash (s \Rightarrow t)} \tag{3.1}$$

Conclusion :

Déduire une implication

3.5.2 Preuve par contradition (ou preuve indirecte)

On prend un hypothèse, et on peut prouver qu'elle est vraie ou fausse, d'où l'hypothèse n'est pas bonne.

Prémisse :

on suppose que p...q n'a pas de problème

$$\frac{\begin{array}{c} p.....q, r, s \vdash s \\ p.....q, r, s \vdash \neg s \end{array}}{p.....q \vdash \neg r} \tag{3.2}$$

Conclusion :

si p...q n'a pas de problème, on se focalise alors sur r

3.6 Exemples de Preuves

Une preuve est une séquence de pas où chaque pas est une application de règles d'inférences et de lois logiques Il faut justifier à chaque étape le nom de

la règle / loi, et indenter les éléments de la preuve en preuve conditionnelle indirecte.

3.6.1 Prémisse :

p $\wedge$ q $\vee$ r

3.6.2 Conclusion :

$\neg$ p $\Rightarrow$ r

3.6.3 Exemple sans preuve conditionnelle

1. (p $\wedge$ q) $\vee$ r *Prémisse*
2. r $\vee$ (p $\wedge$ q) $\vee$ r *Commutativité en 1*
3. (r $\vee$ p) $\wedge$ (r$\vee$q) *Associativité en 2*
4. (r $\vee$ p) *Simplification en 3*
5. (p $\vee$ r) *Commutativité en 4*
6. $\neg\neg$ p $\vee$ r *Loi de la négation en 5*
7. $\neg$ p $\Rightarrow$ r *Implication en 6*

3.6.4 Exemple avec preuve conditionnelle

1. (p $\wedge$ q) $\vee$ r *Prémisse*
2. $\neg$ $\neg$(p $\wedge$ q) $\vee$ r *Double négation en 1*
3. $\neg$ ($\neg$ p $\wedge$ $\neg$ q) *Loi De Morgan en 2*
4. $\neg$ p $\vee$ $\neg$ q $\Rightarrow$ r *Implication en 3*
   (a) $\neg$ p *Hypothèse*
   (b) $\neg$ p $\vee$ $\neg$ p *Addition sur 5*
   (c) r *Modus Ponens sur 4 et 6*
5. $\neg$ p $\Rightarrow$ r *Evacuation de l'hypothèse*

3.6.5 Exemple de Preuve par contradiction

1. (p $\wedge$ q) $\vee$ r *Prémisse*
2. (p $\vee$ r) $\wedge$ (q $\vee$ r) *Distributivité sur 1*
3. (p $\vee$ r) *Simplification en 2*
   (a) $\neg$ ($\neg$ p $\Rightarrow$ r) *Hypothèse*

(b) $\neg$ ($\neg$ $\neg$ p $\vee$ r)*Implication en 4*

(c) $\neg$ (p $\vee$ r) *Négation en 5*

4. $\neg$ $\neg$ ($\neg$ p $\Rightarrow$ r) *Preuve par contradiction*
5. $\neg$ p $\Rightarrow$ r *Négation en 7*

3.7 Quelques concepts

3.7.1 Principe de dualité

Dans les formules sans $\rightarrow$:

$$1 \leftrightarrow T \qquad\qquad true \leftrightarrow false \tag{3.3}$$

$$\models \neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$$
$$\models \neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$$

Formule quelconque :

$$1 \leftrightarrow T \qquad\qquad true \leftrightarrow false \tag{3.4}$$

Justification en raisonnant sur les modèles :

$$p1, ..., pn \models ssi \models (p1, ..., pn \wedge \neg q) \leftrightarrow false$$
$$ssi \models (\neg p1 \vee ... \vee pn) \leftrightarrow true$$

3.7.2 Forme Normale

— Conjonctive : $(p \vee q) \wedge (q \vee a) \wedge (s \vee r)$
— Disjonctive

terminologie

— Littéral :$P \vee \neg P \cong L$
— Clause : $\vee Li = (L1 \vee L2 \vee L3... \vee Li)$

Forme normale conjonctive FNC

3.7.3 Algorithme de normalisation

1. Eliminer les $\rightarrow$ et $\leftrightarrow$

2. Déplacer les négations vers l'intérieur (dans les propositions premières) De Morgan
3. Déplacer les disjonctions ($\vee$) vers l'intérieur
4. Simplifier $(P \vee \neg P)$

Exemple

$(p \to (Q \to R)) \to ((P \wedge S) \to R)$
$\neg(...) \vee (...)$
$\neg(\neg P \vee (\neg Q \vee R)) \vee (\neg(P \wedge S) \vee R)$
$(\neg\neg P \wedge \neg(\neg Q \vee R)) \vee ((\neg P \vee \neg S) \vee R)$
$(P \wedge (Q \wedge \neg R)) \vee (\neg P \vee \neg S \vee R)$
$(P \vee \neg P \vee \neg S \vee R) \wedge (Q \vee \neg P \vee \neg S \vee R) \wedge (\neg R \vee \neg P \vee \neg S \vee R)$
$(Q \vee \neg P \vee \neg S \vee R)$

3.8 Algorithme de preuve

Nous allons maintenant introduire un algorithme qui permet de trouver une preuve en logique des prédicats. Cet algorithme est une automatisation de la *démonstration par l'absurde* qui est basé sur une seule règle d'inférence, la *résolution*.

3.8.1 La résolution

On veut quelque chose de simple, sans toutes les règles que nous avons vues auparavant, mais le plus puissant possible. Nous n'utiliserons qu'une seule règle : la **résolution**. On peut faire des résolutions de preuves propositionnelles rien qu'en ayant cette règle. Cette règle utilise la forme normale conjonctive. On utilise les preuves indirectes (preuves par l'absurde), car c'est le plus simple.
Commençons par un exemple de résolution.

Exemple de résolution

Prenons comme propositions premières :

P_1 : il neige
P_2 : la route est dangereuse
P_3 : on prend des risques
P_4 : on va vite
P_5 : on va lentement
P_6 : on prend le train

$$\left.\begin{array}{l} 1.\ P_1 \Rightarrow P_2 \\ 2.\ P_2 \Rightarrow \neg P_3 \\ 3.\ P_4 \Rightarrow P_3 \vee P_6 \\ 4.\ P_4 \vee P_5 \\ 5.\ P_1 \end{array}\right\} \text{B : notre théorie}$$

On va utiliser B + modus ponens + résolution..

1. $P_1 \Rightarrow P_2$	
2. $P_2 \Rightarrow \neg P_3$	
3. $P_4 \Rightarrow P_3 \vee P_6$	1-5 : B : notre théorie que l'on utilise comme prémisse
4. $P_4 \vee P_5$	
5. P_1	
3'. $\neg P_3 \Rightarrow \neg P_4 \vee P_6$	réécriture de 3
6. P_2	modus ponens (1,5)
7. $\neg P_3$	modus ponens (6,2)
8. $\neg P_4 \vee P_6$	modus ponens (7,3')
9. $P_5 \vee P_6$	résolution (4,8)

La ligne 3 n'étant pas symétrique, nous pouvons la transformer pour obtenir une proposition symétrique et donc choisir le membre qui est à gauche de l'implication. Pour rappel, $P_4 \Rightarrow P_3 \vee P_6$ peut être réécrit : $\neg P_4 \vee P_3 \vee P_6$ (loi de l'implication), qui est logiquement équivalent à $\neg P_3 \Rightarrow \neg P_4 \vee P_6$ (loi de l'implication). C'est de cette manière que nous avons obtenu la ligne 3'.

On peut fusionner les lignes 4 et 8 grâce à la résolution. La **résolution** est une règle qui prend deux disjonctions avec une proposition première et sa négation, et qui les fusionne en retirant cette proposition première. On peut prouver que cela fonctionne de plusieurs manières.
Par exemple : si P_4 est vrai, P_6 doit être vrai. Si P_4 est faux, P_5 doit être vrai. Donc on sait que P_5 ou P_6 doit être vrai car on sait que dans tous les cas de figure, c'est soit l'un soit l'autre qui doit être vrai.

Principe de résolution

$p_1 \vee q$
$p_2 \vee \neg q$

$p_1 \vee p_2$

Cette règle représente la base de l'algorithme de résolution. On peut la vérifier en utilisant le métalangage. De plus, cette règle est aussi utilisée dans la logique des prédicats.

La résolution préserve les modèles

Tout ce qui est modèle des deux premières disjonctions sera aussi modèle de la résultante.

$p : \bigwedge_{1 \leq i \leq n} C_i$ $\quad$ C_i : disjonction : $\bigvee_{1 \leq j \leq n} L_j$ $\quad$ $\{C_1, ..., C_n\}$

C_1, C_2 = deux disjonctions

On doit prouver : $\{C_1, ..., C_n\} \models r$ avec $r = C_1$ - $\{P\} \vee C_2$ - $\{\neg P\}$. r est une nouvelle disjonction à partir de deux autres disjonctions. On doit prouver que r est toujours vrai.

On considère que P est dans C_1 et que $\neg$P est dans C_2.

Pour prouver cela, on utilise la sémantique. On fait une preuve en métalangage, ce n'est pas formalisé.
$\text{Val}_I(P) = \begin{cases} T \\ F \end{cases}$

Dans les deux cas de figure, on doit démontrer que quand on a un modèle, une interprétation qui rend vrai p, le r sera vrai aussi. Si P est vrai alors $\neg$P est faux, donc C_2 sera vrai et donc r sera vrai. Quand P est faux, le C_1 doit être vrai, donc r est vrai. r est donc vrai dans les deux cas.

3.8.2 Algorithme

C_i : clause $\bigvee_i L_i$
L_i : P ou $\neg$P
$\{C_i,...,C_n\} \models C$
s.s.i
$\{C_i,...,C_n,\neg C\} \models$ false
C_i : axiomes
C : candidat théorème
Ce que nous voulons prouver : $\{C_i,...,C_n\} \vdash C$
Il existe une preuve avec les règles d'inférence $\{C_i,...,C_n\}$ tel qu'on obtient C.
$S = \{C_i,...,C_n,\neg C\}$
But : Déterminer si S est inconsistant. On veut faire des déductions jusqu'à arriver sur false.

Pseudocode

while	false $\notin$ S et $\exists$? clauses résolvables non-résolues
do	choisir C_1, $C_2 \in$ S tel que $\exists$ P $\in C_1$, $\neg$P $\in C_2$
	calculer r := C_1 - $\{P\} \vee C_2$ - $\{\neg P\}$
	calcul S := S $\cup$ {r}

end

if false $\in$ S then *C est prouvé* else *C n'est pas prouvé*

La subtilité dans cet algorithme est de choisir correctement les clauses C_1 et C_2 car l'efficacité de l'algorithme en dépend.

3.8.3 Exemples

Exemple 1

C_1 : P $\vee$ Q
C_2 : P $\vee$ R
C_3 : $\neg$Q $\vee$ $\neg$R
C : P $\{C_1, C_2, C_3, \neg C\}$

Quelques pas de résolution :

C_1 + $\neg$C $\rightarrow$ Q (C_5)
C_2 + $\neg$C $\rightarrow$ R (C_6)
C_3 + C_5 $\rightarrow$ $\neg$R (C_7)
C_6 + C_7 $\rightarrow$ false ($\in$ S donc C est prouvé)

Exemple 2

p_1 : Mal de tête $\wedge$ Fièvre $\Rightarrow$ Grippe
p_2 : Gorge blanche $\wedge$ Fièvre $\Rightarrow$ Angine
p_3 : Mal de tête
p_4 : Fièvre

Algorithme
- Normalisation en forme normale
- Pseudocode avec résolution

Question : Grippe ?

3.8.4 Conclusion

Nous pouvons tirer des conclusions sur la logique des propositions et sur notre algorithme.

Pour toute théorie $B = \{c_1, \ldots, c_n\}$ et p,

— si $B \vdash p$ alors $B \models p$ (Adéquat - *Soundness*) ;
— si $B \models p$ alors $B \vdash p$ (Complet - *Completeness*) ;
— $\forall\ B, p$, l'exécution de l'algorithme se termine après un nombre fini d'étapes. (Décidable - *Decidable*)

Cet algorithme est donc très puissant, même s'il n'est peut être pas très efficace. Mais au moins on est sûr qu'il s'arrêtera toujours à un moment.

Malheureusement, la logique des propositions n'est pas très expressive. On va donc essayer de faire la même chose mais avec une logique plus puissante : la logique des prédicats. Mais nous n'arriverons pas à avoir un algorithme aussi puissant pour la logique des prédicats, car c'est une logique trop forte.

Chapitre 4

La logique des prédicats

4.1 Introduction

Nous allons maintenant étudier une logique beaucoup plus expressive que la logique propositionnelle, la logique des prédicats, qui est aussi appelée la logique de premier ordre. [1] Voici un premier tableau qui montre les différences entre la logique propositionnelle vue jusqu'à présent et la logique des prédicats que nous allons étudier.

Logique Propositionnelle	Logique des prédicats
Propositions premières P, Q, R ↪ Pas de Relations	Prédicats P(x,y) Quantifieurs : ∃x, ∀y ↪ Relation

On note dans la logique des prédicats P(x,y) avec x,y qui sont des variables, les arguments du prédicat P. Dans la logique propositionnelle, chaque proposition est "toute seule" alors que dans les prédicats on peut lier plusieurs prédicats ensemble.

Exemple	Logique propositionnelle	Logique des prédicats
Socrate est un philosophe	P	Phil(Socrate)
Platon est un philosophe	Q	Phil(Platon)

En logique propositionnelle il n'y a aucunes relations entre P et Q, alors qu'en logique des prédicats on peut lier Socrate et Platon avec le prédicat Philosophe qui prend en argument Socrate ou Platon. Phil(Socrate) est donc vrai. On peut donc dire grâce aux prédicats que Socrate et Platon

1. Il existe des logiques d'ordres supérieures mais ils ne feront pas l'objet de ce cours.

sont "la même chose", des philosophes.

Un autre exemple de prédicat :

$$\forall \alpha \ \text{Phil}(\alpha) \Rightarrow \text{Savant}(\alpha)$$

$\hookrightarrow$ *... c'est un résumé d'un très grand nombre de faits. ça marche pour tous arguments, ça peut être un ensemble infini !*

Comme Socrate est un philosophe, on peut déduire que Socrate est un savant aussi !

Pour dire la même chose en logique de proposition cela est beaucoup plus compliqué :

"Socrate est un savant" Proposition "R"
"Platon est un savant" Proposition "S"

On va donc noter en logique propositionnelle

$$(\text{P} \Rightarrow \text{R}) \cup (\text{Q} \Rightarrow \text{S}) \cup ...(\textit{potentiellement infini})$$

On doit tout énumérer et il n'y a aucune relations entre les différentes propositions. Si il y a un nombre infini ça ne marche pas. Il y a donc de grandes limitations dans la logique propositionnelle.

Néanmoins parfois la logique propositionnelle est utile. Il existe des outils informatiques qui utilisent la logique propositionnelle et donc parfois ça suffit. Typiquement on peut prendre l'exemple de "SAT solver" qui donne equation booleenes très compliquées et va trouver des valeurs propositions primitives qui rendent vrais cette porposition. C'est donc assez utilisé ! La logique propositionnelle est utile mais si on veut faire du raisonnement sur plus que vrai et faux, avec des relations plus que vrai faux et avec des relations entre des propositions alors la logique propositionnelle ne marche pas. Si on veut faire un logiciel qui montre une certaine intelligence alors la logique des propositions ne marche pas et il faut utiliser la logique des prédicats.

Autre exemple :

Exemple	Logique propositionnelle	Logique des prédicats
Tout adulte peut voter	P	$\forall$x adulte(x) $\Rightarrow$ voter(x)
John est un adulte	Q	adulte(john)
———	—	———
John peut voter	?R ?	voter(john)

Ce genre de raisonnement est très difficile à faire en logique propositionnelle alors qu'en logique des prédicats c'est beaucoup plus simple ! Le john en ligne 3 et en ligne 4 sont les même john ! Ceci montre donc bien qu'il nous faut la logique des prédicats pour faire des relations de ce type.

4.2 Quantificateurs

Les expressions "pour tout x" ($\forall x$) et "il existe x tel que" ($\exists x$) sont appelés des quantificateurs en logique des prédicats. Les quantificateurs permettent d'instancier les variables dans une formule. La notion de portée d'un quantificateur est un concept très important auquel il faut faire très attention, car il peut changer complètement le sens d'une formulation.

$\forall x$ (enfants(x) $\wedge$ intelligents(x) $\Rightarrow$ $\exists y$ aime(x,y))

$\forall x$ (enfants(x) $\wedge$ intelligents(x)) $\Rightarrow$ $\exists y$ aime(x,y)

Ces deux formules peuvent paraître équivalente, mais en réalité elles ont un sens tout à fait différent. En effet, dans le deuxième cas on remarque que le quantifficateurs $\forall x$ ne porte pas sur la dernière variable x qui est en argument du prédicat aime(x,y).

Il faut donc faire bien attention à quel quantificateur une variable s'identifie lorsqu'on manipule des formules.

- $\forall x$ P(x) $\wedge$ $\exists x$ Q(x) : contient deux variables différentes
- $\forall x$ $\exists x$ P(x) $\wedge$ Q(x) : est une forme incorecte, conflit des noms de variables

Pour résoudre ces conflits on fait appel à une nouvelle opération, le renomage. Cette opération permet de changer le nom des variables tout en conservant le sens de la formule. Ainsi on obtient :

$\forall x$ $\exists z$ P(x) $\wedge$ Q(z) *renomage (2)*

Le concept de variables, de leur portées ainsi que d'opérateurs en logique des prédicats fait fortement penser au langage de programmation

Une comparaison entre un code et une formule est tout à fait envisageable. Prenons un code tout à fait banal comprenant des variables dif-

férentes avec des portées différentes qui ont le même identificateur ainsis qu'une formule correspondante.

```
1.  begin {
2.      var x,y: int;
3.      x := 4;
4.      y := 2;
5.
6.      begin {
7.          var x: int;
8.          x := 5;
9.          x := x*y;
10.     end }
11.     x := x*y;
12. end }
```

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

En analysant morceau par morceau de la formule :

- " $\forall x\ \forall y$ p(x) $\wedge$ " correspond aux points $\{2, 3, 4\}$ du code
- " $\exists x$ q(x,y) $\vee$ " correspond aux points $\{7, 8, 9\}$
- " r(x,y)) " correspond au point $\{11\}$

Cet exemple ilustre parfaitement la ressemblance et le lien entre le monde de la programmation et celui de la logique des prédicats

4.3 Syntaxe

Symboles logiques	quantificateurs connecteurs logiques parenthèses variables true, false	$\forall\ \exists$ $\wedge\ \vee\ \neg\ \Rightarrow\ \Leftrightarrow$ ( ) x, y, z
Symboles non-logiques	symboles de prédicats symboles de fonction	$P\ \varphi\ R$ + arguments ≥ 0 + $arguments \geq 0$

4.4 Grammaire

4.4.1 Règles de formation

$$
\begin{aligned}
< formule >::= \quad & < formule\ atomique > \\
& | \neg < formule > \\
& | < formule > < connecteur > < formule > \\
& | \forall < var > . < formule > \\
& | \exists < var > . < formule > \\
< formule\ atomique >::= \quad & \text{true, false} \\
& | < predicat > (< terme > *) \\
< terme >::= \quad & < constante > \\
& | < var > \\
& | < fonction > (< terme > *) \\
< connecteur\ binaire >::= \quad & \wedge | \vee | \Rightarrow | \Leftrightarrow
\end{aligned}
$$

4.5 Sémantique

Dans la logique des prédicats, nous gardons les notions de modèle et d'interprétation qui sont déjà définis dans la logique propositionelle. Même si la logique des prédicats est beaucoup plus puissante, nous gardons une sémantique assez similaire à la logique propositionelle.
L'idée dans l'interprétation c'est de dire si la formule est vraie ou fausse. Tout comme pour la logique propositionelle on va faire la même chose pour les prédicats avec les relations.

Illustrons par un exemple :

$$p : P(b, f(b)) \Rightarrow \exists y P(a, y)$$

On suppose que le a et le b sont des constantes et que le f est une fonction. Il faut donner un sens à cette formule et pour cela il faut donner un sens à :

- P : $val_I(P) = \geq$ (*Considéré comme un vrai prédicat*)
- a : $val_I(a) = 2$
- b : $val_I(b) = \pi$
- f : $val_I(f) = f_i \quad f_i = \Re \to \Re : d \to \dfrac{d}{2}$

Avec ces éléments, on peut donc trouver l'interprétation :

$$Si\ \pi \geq \frac{\pi}{2},\ alors\ \exists\ d \in \Re\ tel\ que\ \sqrt{2} \geq d$$

Cette phrase n'est pas très utile mais avec cette interprétation, la phrase logique donne ce sens. Une autre interprétation donnerais un sens totale-

ment différent à la phrase logique. Voici une autre interprétation totalement différente :

- a : $val_I(a)$ = "Barack Obama"
- b : $val_I(b)$ = "Vladimir Putin"
- f : $val_I(f) = f_i$ $\quad f_i \rightarrow$ père(d)
- P : $val_I(P) = P_I$ $\quad d_1$ est enfant de d_2

Avec ce nouveau sens, on trouve l'interprétation suivante :

Si Vladimir Putin est l'enfant du père de Vladimir Putin alors $\exists$ *une personne telle que Barack Obama est l'enfant de cette personne.*

La seconde interprétation est très différente de la première malgré le fait que ce soit la même formule à l'origine ! La connexion entre une formule et son sens permet de garder une certaine souplesse dans le sens où l'on peut choisir ça. C'est un peu comme dans la logique propositionelle mais avce encore plus de souplesse. Si l'on fait ce raisonnement-ci, ce dernier sera toujours vrai pour toutes les interprétations qui sont vraies.

On peut se demander si ces deux interprétations sont des modèles de la formule ?
La première interprétation est un modèle de la formule car le sens de la formule est vrai dans l'interprétation. En effet, $\pi \geq \frac{\pi}{2}$ et $\exists\ d \in \Re$ tel que $\sqrt{2} \geq d$. On voit donc que le modèle est vrai.

La deuxième interprétation est aussi un modèle de la formule car l'interprétation trouvée est vrai aussi. Cela peut paraître bizarre mais c'est correct.

Chapitre 5

Logique des prédicats

5.1 Détails techniques

La première question que l'on se pose est : Comment faire des preuves en logique des prédicats ? Il faut noter que les quantificateurs $\forall$ et $\exists$ rendent le raisonnement plus subtile. Il faudra des règles pour pouvoir raisonner sur ces quantificateurs.

5.2 Sémantique

La sémantique en logique des prédicats est très proche de celle en logique propositionnelle. Cependant, l'interprétation, I, est plus précise qu'en logique propositionnelle mais est également plus compliquée à utiliser à cause des variables et des symboles de fonction. cette interprétation peut être décrite comme une paire : $I = pair(D_I, val_I)$ avec D le domaine de discours I et la fonction val qui est l'interprétation de tous les symboles. $D_I \neq \emptyset$ $\forall s \in S$ avec s soit un symbole de prédicat soit une fonction. $val_I(S) = P_I$ une fonction $P_I : D_I^n \rightarrow (True, False)$ avec S symbole d'une fonction. Il existe aussi une vrai fonction $f_I : D_I^n \rightarrow D_I$ avec n le nombre d'arguments tel que $val_I(S) = f_I$. Cela implique que dans le domaine de discours chaque fonction correspond à une vrai fonction et chaque prédicat correspond à un vrai prédicat.

Si on rajoute une variable, x par exemple, l'expression devient : $(var, x) \rightarrow val_I(x) = x_I \in D_I$. L'interprétation d'une variable, x_i est un élément (n'importe lequel en fonction de l'interprétation) de D_I. La fonction VAL_I est la même fonction mais pour les formules et pas uniquement pour les symboles comme avant. Néanmoins on ne doit pas redéfinir VAL_I car elle existe à partir du moment ou $val_I et D_I$ existent. Définition : $VAL_I : TERM \cup PRED \rightarrow D_I \cup (True, False)$ avec TERM l'ensemble des termes et PRED l'ensemble des prédicats (toutes les formules en logique des prédicats). Les termes peuvent être définis comme $t \rightarrow VAL_I(t)$ et les prédicats comme $P \rightarrow VAL_I(P)$ avec P une formule.Il y à une relation entre

val_I et VAL_I : $val_I((P(T_1, ..., T_n)) = P_I(VAL_I(t_1), ..., VAL_I(t_n))$ avec $P_I = VAL_I(P)$ La première partie de l'égalité est un prédicat avec des arguments tandis que la seconde est formée de plusieurs prédicats avec un argument.

Définissons d'autres formules. $VAL_I(P \wedge Q)$ est true si $VAL_I(P) = True$ et $VAL_I(Q) = True$. false sinon. Donc il faut $VAL_I(\forall x.P) = True$ avec P une formule pouvant dépendre de x (il faut remplacer toutes les occurrences de x dans P et la formule doit restée vraie. Si pour chaque d $\in D_I$ $I' = I \cup (x \leftarrow d)$ et $val_I(x) = d$ alors l'interprétation de x doit être : $VAL'_I(P) = True$. Pour $VAL_I(\exists x.P) = True$ le raisonnement est identique en remplacant $\forall$ par $\exists$.

5.3 Différence avec la logique des propositions

Les trois différences entre les deux logiques sont : les variables, les quantificateurs et les symboles de fonction (moins important). Imaginons un modèle $B : \left\{ P_1, ...P_n \right\}$ Si nous utilisons une interprétation I pour B cela donne : $\forall P_I \in BVAL_I(P_I) = True$ qui est très générale car P_I peut avoir des variables, des quantificateurs, etc...

5.4 Preuves avec règles

Il est possible de faire des preuves avec des règles de preuve. Une preuve est une simple manipulation de symboles qui sont des règles. Mais il faut

justifier ces règles pour obtenir des résultats vrais. Elles sont justifiées en raisonnant sur les interprétations pour vérifier si elles sont correctes ou non. Tout cela est la sémantique. C'est la base pour pouvoir faire des preuves.

Beaucoup de choses restent les mêmes que dans la logique des propositions. La preuve est toujours un objet mathématique, une séquence avec des formules, des justifications, une application des règles. Elle commence avec des prémisses et finit par une conclusion.

Il est possible de faire des preuves manuelles, mais aussi des preuves automatisées. C'est une généralisation de l'approche de la logique des propositions.

Il y a toujours :

— Une règle de résolution pour les preuves automatisées, mais celle-ci est plus générale. Elle va utiliser un concept appelé "unification". Ce nouveau concept est introduit à cause des variables. En effet, celle-ci peuvent être différentes, il faut donc trouver un nouveau moyen de les fusionner.
— Une forme normale, mais elle est plus compliquée à cause des quantificateurs et des symboles de fonctions. Mais il est encore possible de l'obtenir.
— un algorithme avec ses propriétés. Mais il est moins fort/complet que l'algorithme développé pour la logique des propositions. Il ne sera plus décidable mais seulement semi-décidable. C'est-à-dire que parfois il tournera en boucle. Cela est dû aux variables et aux quantificateurs. La logique des prédicats est beaucoup plus riche que la logique des propositions, mais en contrepartie l'algorithme arrive à prouver moins de choses. Cependant, l'aglorithme sera toujours adéquat mais pas forcément complet. Il ne sera pas toujours possible de trouver une preuve, même quand elle existe parce qu'elle sera trop compliquée.

Qu'est-ce qu'il est possible de faire avec ce genre d'algorithme qui est moins fort ?

Il y a deux possibilités :

1) Faire un assistant de preuve

C'est un outil qui aide les gens à faire des preuves formelles. Deux exemples d'assistants de preuves sont Coq et Isabelle. C'est un outil très sophistiqué mais qui a permis de prouver des choses de manière totalement formelle, alors qu'avant des preuves prenaient des dizaines voire des centaines de pages de preuves mathématiques. Mais cet assistant ne fait pas tout parce que l'algorithme est moins bon. Cependant il aide beaucoup. C'est à l'être humain de lui donner des coups de pouce sous forme de lemmes, hypothèses, chemins, stratégies ... Ensuite l'algorithme s'occupe de la manipulation des symboles.

Un exemple très célèbre est le théorème de la coloration d'une carte. La question est : Est-il toujours possible de colorier chaque pays avec une couleur, de façon à ce que deux pays limitrophes n'aient pas la même couleur et en utilisant un certain nombre de couleurs différentes ? Ce n'est pas évident à prouver et ça a demandé beaucoup de travail aux mathématiciens. Mais récemment, Georges Gonthier (un informaticien) a réussi à formuler ce problème avec l'assistant de preuves. Ce fût un tour de force. Désormais, il existe une preuve complètement formalisée, sans erreur pour ce théorème.

2) L'utiliser dans les langages de programmation

L'algorithme peut être considéré comme le moteur d'un programme. C'est ce qu'on appelle maintenant la programmation logique. Elle consiste à utiliser la logique dans un programme. Le langage le plus célèbre qui a suivi cette approche est Prolog. Ce fût un énorme succès car les gens ne

croyaient pas que c'était possible de faire un programme en logique qui pouvait tourner. Cela a donné naissance à la programmation par contraintes (une contrainte est une relation logique). Cette discipline est très utile pour les optimisations, par exemple dans le cas du "voyageur de commerce".

La logique des prédicats n'est donc pas quelque chose de seulement théorique, destiné uniquement aux mathématiciens. Les gens ont vraiment essayé avec succès d'utiliser la logique dans l'exécution des programmes.

Chapitre 6

Preuves en logique des prédicats

On va généraliser l'approche de la logique propositionnelle car le langage des prédicats est beaucoup plus riche. Il ajoute entre autres :

— Des variables
— Des constantes
— Des fonctions (détaillé plus tard)
— Des prédicats
— Des quantificateurs

Les preuves en logique des prédicats ressemblent très fort aux preuves en logique propositionnelles. Il y a encore des prémisses, des formules avec leurs justificatifs et une conclusion. On peut aussi utiliser des preuves indirectes et des preuves conditionnelles. Cela reste un objet formel.

1.	$\boxed{\ldots}$	*Premisses*
2.	Formule, regle	$Justification$
...	...	...
n.	Conclusion	$Justification$

6.1 Exemple

La méthode pour passer de $\forall x \cdot P(x) \wedge Q(x)$ (prémisse) à $\forall x \cdot P(x) \wedge (\forall x \cdot Q(x))$ (conclusion) est la suivante :

1. Enlever les quantificateurs pour avoir des variables libres
2. Raisonner sur l'intérieur
3. Remettre les quantificateurs

Les étapes difficiles à réaliser correctement sont les étapes 1 et 3. Voici la preuve en "français" :

En retirant les quantificateurs des prémisses, cela donne : "Alors, $P(x) \wedge Q(x)$ est vrai pour tout x donc $P(x)$ est vrai pour tout x". De là on peut remettre les quantificateurs pour obtenir $\forall x \cdot P(x)$. De façon similaire, on obtient $\forall x \cdot Q(x)$. Et on conclut en remettant les quantificateurs : $\forall x \cdot P(x) \wedge \forall x \cdot Q(x)$, en utilisant la conjonction.

En preuve formelle cela donne :

1.	$\forall x \cdot P(x) \wedge Q(x)$	Prémisses
2.	$P(x) \wedge Q(x)$	Elimination de $\forall$
3.	$P(x)$	Simplification
4.	$\forall x \cdot P(x)$	Introduction de $\forall$
5.	$Q(x)$	Simplification $\forall$
6.	$\forall x \cdot Q(x)$	Introduction de $\forall$
7.	$\forall x \cdot P(x) \wedge \forall x \cdot Q(x)$	Conjonction

On a donc utilisé 4 règles en plus par rapport aux preuves formelles en logique propositionnelle (les règles de la logique propositionnelle restent valables en logique des prédicats) :

— Elimination de $\forall$
— Introduction de $\forall$
— Elimination de $\exists$
— Introduction de $\exists$

Certaines de ces règles sont simples d'utilisation, d'autres sont plus difficiles. Il est également possible d'utiliser d'autres règles (certaines plus générales que d'autres [1]).

Note :

Il est possible d'utiliser les quantificateurs dans les formules mathématiques. Typiquement, on ne les note pas, car ils sont présents de manière implicite. Par exemple :

— $\forall x \cdot \sin(2x) = 2 \cdot \sin(x) \cdot \cos(x)$
— $\forall x \cdot x + x = 2x$
— $\exists x \cdot \sin(x) + \cos(x) = 0,5$
— $\exists x \cdot x + 5 = 9$

On peut remarquer que pour les deux premiers cas, x est une véritable variable, on peut donc ajouter un quantificateur universel $\forall$.

1. Voir "Inference logic" ou "Predicate logic"

Pour les deux cas suivants, on remarque que x est une inconnue, car il y a une équation à résoudre et une solution à trouver, on peut donc ajouter un quantificateur existentiel $\exists$.

Dans certains cas, les quantificateurs existentiels et universels sont utilisés au sein de la même formule mathématique.

$\forall a \cdot \forall b \cdot \forall c \cdot \exists x \cdot ax^2 + bx + c = 0$

Dans l'exemple ci-dessus, nous avons 4 variables : a, b, c, x. Les 3 premières sont des véritables variables, on peut les affecter à n'importe quelle valeur, tandis que la dernière est une inconnue, c'est la solution à trouver. Il faut donc trouver x pour toutes les valeurs possibles de a, b, c.

6.2 Règles en logique des prédicats

En logique des prédicats, pour trouver une preuve, on va faire des manipulations de formules.

6.2.1 La substitution

Une manipulation fréquente en logique des prédicats est la **substitution**. Elle consiste à prendre une formule et remplacer une partie par une autre.

Si on a une formule $p[x/t]$ (où p veut dire "toutes les formules"). On va remplacer toutes les occurrences libres de x par t.
On peut aussi écrire cela de la manière suivante :

$p[x/t]$ possède deux portées :
La première se note $p[x]$ et correspond à une partie de la règle.
La seconde se note $p[t]$ et correspond à l'autre partie de la règle.

Exemple :

1.	$P(x) \to \forall y \cdot (P(x) \land R(y))$	$[x/y]$ veut dire qu'on va remplacer toutes les occurences libres de x par y.
2.	$P(y) \to \forall y \cdot (P(y) \land R(y))$	En remplaçant x par y, on a changé le sens de la formule car avant, x n'était pas dans la portée du quantificateur alors que maintenant il l'est. Ce changement de sens s'appelle une **capture de variable**, car la variable y est capturée par le quantificateur. Pour résoudre ce problème, on va effectuer un **renommage**.
3.	$P(y) \to \forall z \cdot (P(y) \land R(z))$	Résultat après renommage (pour éviter la capture de variable).

6.3 Elimination de ∀

∀x•P(x) → P(a) a est une constante
→ P(y) y est une variable ($P_I(y_I)$ est vrai $y_I \in P_I$)
∀ = pour tout $x_I \in P_I$: $P_I(x_I)$ est vrai

Règle :

$$\frac{\forall x{:}p}{p[x/t]}$$

Substitution : t remplace x

/ !\ Il faut faire du renommage dans certains cas !

Exemple :

1. ∀x • ∀y • P(x,y) Prémisse
2. ∀y • P(x,y) Elimination de ∀
3. P(x,x) Elimination de ∀ → Pas de renommage car pas de capture
4. ∀x • P(x,x) Introduction de ∀

6.4 Elimination de ∃

∃x • P(x) → P(x) x = nouvelle constante qui apparait nulle part ailleurs ($val_I(a) = x_I$)
Il existe un $x_I \in D_I$ avec $P_I(x_I)$ est vrai
↛ P(y) y = variable qui existe déjà dans la preuve
→ P(z) z = nouvelle variable dans la preuve $val_I(z) = x_I$

Exemple 1 :

1. ∃x • chef(x) Prémisse
2. ∃x • voleur(x) Prémisse
3. chef(y) Elimination de ∃
4. ~~voleur(y)~~ Elimination de ∃ y n'est pas une nouvelle variable dans la preuve
5. chef(y) ∧ voleur(y) Conjonction
6. ∃y • chef(y) ∧ voleur(y) Introduction de ∃

Exemple 2 :

1. $\exists$x • chef(x) Prémisse
2. $\exists$x • voleur (x) Prémisse
3. chef(y) Elimination de $\exists$
4. voleur (z) Elimination de $\exists$
5. chef(y) $\wedge$ voleur(z) Conjonction
6. $\exists$y $\exists$z chef(y) $\wedge$ voleur(z) Introduction de $\exists$

6.5 Introduction de $\exists$

Règle :

$$\frac{p[t]}{\exists x \bullet p[x]}$$

Il y a une substitution p[x/t]

Exemple :

P(y,y)
$\exists$x • P(x,x)

~~P(y,x)~~
~~$\exists$x • P(x,x)~~

Ceci n'est pas correct !

$\Rightarrow$ Il doit être possible de retrouver la formule original en remplaçant.

6.6 Introduction de $\forall$

Règle :

$$\frac{p}{\forall x \bullet p}$$

— Si p n'a pas d'occurence libre de x alors c'est OK
— Si p contient une occurence libre de x : on doit s'assurer que la preuve jusqu'à cet endroit marchera pour toutes valeurs affectées à x
↪ Aucune formule dans la preuve juqu'à cet endroit ne doit mettre une contrainte sur x !

Deux conditions :

- x n'est pas libre dans une formule dans la preuve jusqu'à cet endroit obtenu par élimination de $\exists$
- x n'est pas libre dans une prémisse (x est déjà connu au début donc il possède déjà une valeur)

Exemple :

1. $\forall x\ \exists y$ parent(y,x) Prémisse
2. $\exists y$ parent(y,x) Elimination de $\forall$
3. parent(y,x) Elimination de $\exists$ $\rightarrow$ N'est valable que pour ce y et ce x, pas pour tous
4. ~~$\forall x$ parent(y,x)~~ Introduction de $\forall$ $\rightarrow$ On ne peut pas faire ça car il y a une contrainte sur x. Là on dit que ce y est parent de tous !

Terminons par un exemple un peu plus conséquent d'une preuve manuelle en logique des prédicats avant d'introduire l'algorithme permettant d'effectuer des preuves de manière automatisée.

6.6.1 Exemple de preuve manuelle

Il est important de pouvoir faire des preuves manuellement, car cela permet de bien comprendre toutes les étapes de raisonnement d'une preuve, même si par la suite on utilise un algorithme plutôt que de faire les preuves à la main.

L'exemple suivant est inspiré de l'Empire romain :

Prémisses

— Les maîtres et esclaves sont tous des hommes adultes
— toutes les personnes ne sont pas des hommes adultes

Note : on voit dans les prémisses qu'il y a des quantificateurs : tous, toutes.

A prouver

— il existe des personnes qui ne sont pas des maîtres

Preuve

1. $\forall x\ (maitre(x) \vee esclave(x) \implies adulte(x) \wedge homme(x))$ prémisse
2. $\neg\forall x\ (adulte(x) \wedge homme(x))$ prémisse
3. $\exists x\ \neg(adulte(x) \wedge homme(x))$ théorème négation
   S' il n'est pas vrai que toutes les personnes sont des hommes adultes alors il existe une personne qui n'est pas un homme adulte
4. $\neg(adulte(x) \wedge homme(x))$ $\exists$ elim
   On élimine le quantificateur existentiel : on peut le faire, car on introduit une variable x qu'on choisit comme étant une personne rendant vraie la proposition.
5. $(maitre(x) \vee esclave(x) \implies adulte(x) \wedge homme(x))$ $\forall$ elim
   On peut retirer le $\forall$ en réduisant le champ de x aux x rendant vraie la proposition.
6. $\neg(maitre(x) \vee esclave(x))$ modus tollens
7. $\neg maitre(x) \wedge \neg esclave(x)$ de Morgan
8. $\neg maitre(x)$ simplification

9. $\exists y\ \neg maitre(y)$ $\exists$ intro
   Comme dans l'interprétation, x est une personne qui rend valable cette proposition, on peut dire qu'il existe une personne rendant valable cette proposition et réintroduire le quantificateur $\exists$

C'était un exemple très simple ne faisant que quelques pas, mais la logique est assez expressive pour permettre des preuves plus complexes(par exemple, formaliser les mathématiques), le nombre de pas serait alors beaucoup plus important.

Instant Histoire :

A la fin du 19ème siècle, début du 20ème :

- création de la logique de 1er ordre (Gottlob Frege)
- Deux personnes ont essayé de formaliser toutes les mathématiques. Principa Mathématica (Alfred Whitehead, Bertrand Russell)

Lors que l'arrivée des ordinateurs, fin du 20ème siècle(années 50-60 et fin du siècle) on a essayé de formaliser la logique via des algorithmes :

- Création de l'Algorithme de Preuves (1965) :
  - Alan Robinson crée La Règle de Résolution (qui va être expliquée aujourd'hui)
  - Création de prouveurs (assistants de preuve) par exemple Coq et Isabelle en 1972
  - Création de la logique de programmation qui aide à l'élaboration de la programmation par contraintes : Prolog (1972)
- Création de la sémantique Web : OWL (Web Ontology Langage)

Chapitre 7

Algorithme de preuve pour la logique des prédicats

— Cet algorithme s'inspire de l'algorithme de réfutation de la logique des propositions [résolution forme clausale]
— Pour la logique des prédicats, c'est un peu plus complique, mais ça marche !

On peut le faire marcher malgré la complexité des variables et des quantificateurs ce qui est assez étonnant, car c'est une logique très expressive. Arriver à trouver un algorithme permettant de traiter la logique des prédicats était une sorte de Graal au 20ème siècle.

7.1 3 transformations

On va commencer par faire les transformations de normalisation. Il y a 3 transformations à effectuer :

1. formule → forme prénexe :

$$(\dots\forall\dots\exists\dots\forall) \implies \forall\exists\forall(\dots)$$

   Tous les quantificateurs sont mis en tête de la formule. Les quantificateurs étant très compliqués à gérer, on transforme la formule pour les extraire de celle-ci.
   Les modèles sont conservés durant cette transformation.

2. forme prénexe → forme Skolem (élimination des $\exists$) :

$$\forall\exists\forall(\dots) \implies \forall\forall\forall(\dots)$$

   Les quantificateurs existentiels sont très embêtants, car ils sont restrictifs. Ils disent qu'il existe des éléments, mais ne précisent pas

lesquels, on va donc les éliminer.
Cette transformation préserve l'existence des modèles, mais pas les modèles eux-mêmes. Ils doivent être modifiés pour conserver la même signification.

3. forme Skolem → forme normale conjonctive :

$$\forall \ldots \forall \wedge_i (\vee_j L_{ij})$$

(*meme algorithme que pour la logique des propositions*)

Cette transformation est la même que celle effectuée dans la logique des propositions.
Les modèles sont préservés lors de cette transformation.

7.2 Résolution

En logique des propositions :

$$\frac{L \vee C_1, \neg L \vee C_2}{C_1 \vee C_2}$$

Cela fonctionne en logique des propositions, car il n'y a pas de variables, mais ici, on peut avoir $L_1 \vee C_1 \quad \neg L_2 \vee C_2$ avec L_1 et L_2 qui ont des variables différentes. Par exemple P(x,a) et P(y,z).
Pour pouvoir faire la résolution, il va falloir en quelque sorte les rendre identiques.
On va donc dire : ce ne sont peut-être pas toujours les mêmes, mais, pour certaines valeurs, ils sont identiques. Si x=y et a=z alors on peut faire la résolution.

7.2.1 Unification

L1 : P(x,a)
L2 : P(y,z)

Pour que L_1 et L_2 soient identiques, on va restreindre les variables et faire une substitution.

$$(a : \textit{constante},\ x, yz : \textit{variables})$$

$$\sigma = \{(x, y), (z, a)\}$$

$$P(x, a) \rightarrow P(y, a)$$

$$P(y, z) \rightarrow P(y, a)$$

Cette résolution marche pour toutes les valeurs qui sont limitées par la substitution. Le résultat ne sera donc pas général. Cette opération s'appelle l'unification et utilise la substitution σ (sigma). On peut maintenant faire

la résolution en appliquant le même algorithme de réfutation que pour la logique des prédicats :

$$\frac{L_1 C_1, \neg L_2 \vee C_2}{(C_1 \vee C_2)\sigma}$$

7.3 Propriétés de cet algorithme

- Cet algorithme est moins fort que pour la logique des propositions, car la logique des prédicats est beaucoup plus expressive.
- adéquat : Si $B \vdash T \rightarrow B \models T$
  si l'on trouve une preuve de T avec les axiomes B alors T sera vrai dans tous les modèles de B
- complet : Si $B \models T \rightarrow B \vdash T$
  Si quelque chose est vrai dans tous les modèles alors on va trouver une preuve
- L'algorithme possède les propriétés d'un algorithme semi-décidable :
  - Si $B \vdash T \rightarrow$ l'algorithme trouve une preuve.
  - Si $B \nvdash T \rightarrow$ il peut tourner en rond indéfiniment.

  Si ce qu'on tente de prouver est vrai dans tous les modèles, l'algorithme va finir par trouver une preuve, mais si ce n'est pas vrai, l'algorithme va tourner en rond et ne jamais se terminer. Le problème est donc que quand l'algorithme prend trop de temps à trouver une preuve, on doit l'arrêter et l'on n'est jamais certain du résultat. On ne peut jamais être sûr que l'algorithme n'aurait pas trouvé une preuve si on l'avait laissé tourner plus longtemps. Il est donc semi-décidable, car ses résultats ne sont totalement fiables que dans le cas ou une preuve est trouvée.

7.4 Transformation de la formule de base vers la forme prénexe

Etapes de la transformation en formule logiquement équivalente :

1. Éliminer $\Leftrightarrow$ et $\Longrightarrow$
2. Renommer les variables.
   - Chaque quantificateur ne porte que sur une variable, il faudra en créer de nouvelles si besoin en prenant soin de conserver l'équivalence de la formule .
   - Attention : ne jamais garder le même nom de variable pour une variable libre et une variable liée.
   - Supprimer les quantificateurs si possible.

3. Migrer les négations ($\neg$) vers l'intérieur, vers les prédicats. On peut faire cela, car $\neg\exists$ peut être transformé en $\forall\neg$ et vice versa.
4. On peut mettre tous les quantificateurs de la logique des prédicats à l'avant de la formule.

7.4.1 Exemple d'une transformation en forme prénexe

1. $\forall x[p(x) \wedge \neg(\exists y)\forall x(\neg q(x,y)) => \forall z\exists v \bullet p(a,x,y,v))]$
   Expression de base
2. $\forall x[p(x) \wedge \neg(\exists y)(\forall x)(\neg\ \neg q(x,y) \vee\ \forall z\exists v \bullet r(a,x,y,v)]$
   Suppression des $\Longrightarrow$
3. $\forall x[p(x) \wedge \neg(\exists y)(\forall u)(\neg\neg q(u,y) \vee \not{\forall z}\exists v \bullet r(a,u,y,v)]$
   Renommage des variables et suppression des quantificateurs inutiles
4. $\forall x[p(x) \wedge \forall y\neg\ (\forall u)(\ \not{\neg\neg}\ q(u,y) \vee \exists v \bullet r(a,u,y,v)]$
   $\neg\exists y$ *devient* $\forall\ y\ \neg$ *et simplification des* $\neg$
5. $\forall x[p(x) \wedge \forall y\ \exists u\neg\ (q(u,y) \vee \exists v \bullet r(a,u,y,v)]$
   $\neg\forall u$ *devient* $\exists\ u\ \neg$
6. $\forall x[p(x) \wedge \forall y\exists u\ (\neg q(u,y) \wedge \neg(\exists v) \bullet r(a,u,y,v))]$
   Distribution des $\neg$*(de Morgan)*
7. $\forall x[p(x) \wedge \forall y\exists u(\neg q(u,y) \wedge\ (\forall v) \bullet \neg\ r(a,u,y,v))]$
   $\neg\exists y$ *devient* $\forall\ y\ \neg$
8. $\forall x\ \forall y\exists u\forall v\bullet\ [p(x) \wedge (\neg q(u,y) \wedge \neg r(a,u,y,v))]$
   Extraction des quantificateurs.

7.5 Transformation en forme Skolen

7.5.1 Intuition

Cette transformation consiste à éliminer toutes les occurences de quantificateurs existentiels.

$(\forall x)(\forall y)(\exists u)(\forall v)\big[P(x) \wedge \neg Q(u,y) \wedge \neg R(a,u,y,v)\big]$

Dans ce cas-ci, la valeur de u de dépend des valeurs de x et y. Lorsqu'on a choisi x et y, on est alors libres de choisir u. On peut donc supposer qu'une fonction $g(x,y)$ fournit cet élément de façon à conserver la satisfaisabilité de la formule tout en supprimant $(\exists u)$.

$(\forall x)(\forall y)(\forall v)\big[P(x) \wedge \neg Q(\mathbf{g(x,y)},y) \wedge \neg R(a,\mathbf{g(x,y)},y,v)\big]$

Après la transformation, l'existence des modèles est préservée.

7.5.2 Règle

Pour chaque élimination d'un quantificateur existentiel $(\exists x)$, on remplace sa variable quantifiée par une fonction $f(x_1,...,x_n)$ dont les arguments sont les variables des quantificateurs universels dont x est dans la portée.

Justification par un exemple :

$p : \forall x \forall y \exists z\big[\neg P(x,y) \vee Q(x,z)\big]$

$p_s : \forall x \forall y\big[\neg P(x,y) \vee Q(x,\mathbf{f(x,y)})\big]$

Les modèles de p $\neq$ modèles p_s.

Pour p
— Interprétation I
— D_I = Professeur $\cup$ Université
— $Val_I(P) = P_i$ = "a enseigné à l'université"
— $Val_I(Q) = Q_i$ = "est diplômé de l'université"
— $Val_I(f)$ n'existe pas.

Pour p_s
— Etendre I
— $I' = \{F \vdash F_i\} \circ I$
— $f(a,b)$ = "l'université ayant dû diplômer a pour que a puisse enseigner à b"

p admet un modèle (I) SSI p_s admet un modèle (I').

Que ça ne soit exactement le même modèle ne pose pas de problème pour notre algorithme. L'algorithme par réfutation continue à itérer jusqu'à

trouver une contradiction ($false$). S'il n'y a pas de modèle pour p_s, il n'y a pas de modèle pour p et ça suffit.

7.6 Transformation en forme normale conjonctive

Même manipulations qu'en logique des propositions.

7.7 La règle de résolution

$$\frac{L_1 \vee C_1, \neg L_2 \vee C_2}{(C_1 \vee C_2)\sigma}$$

Cette règle de résolution ne fonctionne que si L_1 et L_2 sont identiques.

— $L_1 = P_1(a, y, z)$
— $L_2 = P_1(x, b, z)$

Dans un modèle il y a un prédicat qui correspond au symbole P_1 et il y a un ensemble de triplets qui rendent vrai P_1.

L'unification de L_1 et L_2 donne L qui représente l'intersection des deux ensembles. On écrit :

— $L_1\sigma = L$
— $L_2\sigma = L$

L_1 et L_2 sont unifiables s'il existe une substitution σ telle que $L_1\sigma = L_2\sigma$.

— $\sigma = \{(x, a), (y, b)\}$
— $L_1\sigma = P(a, b, z)$
— $L_2\sigma = P(a, b, z)$

Exemple :

— $L_1 = P_1(a, x)$
— $L_2 = P_2(b, x)$

Il n'y pas de substitution qui existe car on a deux constantes différentes, l'intersection des deux ensembles est vide.

Exemple 2 :

— $L_1 = P(f(x), z)$
— $L_2 = P(y, a)$

Dans ce cas-ci, il y a beaucoup de subtitutions possibles telles que :

— $\sigma_1 : \{(y, f(a)), (x, a), (z, a)\} => L_1\sigma_1 = L_2\sigma_1 = p(f(x), a)$
— $\sigma_2 : \{(y, f(x)), (z, a)\} => L_1\sigma_2 = L_2\sigma_2 = p(f(x), a)$

Dans ce cas-ci, σ_2 est plus général.

— On préfère alors la substitution σ la plus générale.
— On peut démontrer qu'il existe un unificateur plus général U.P.G.
— U.P.G. est calculable.

Règle de résolution :
— p_1, p_2 clauses
— $p_1 = L^+ \vee C_1$
— $p_2 = \neg L^- \vee C_2$
— L^+ et L^- ont les mêmes symboles de prédicat.
— $\{L^+, L^-\}$ unifiable par σ U.P.G.
Alors

$$\frac{L^+ \vee C_1, \neg L^- \vee C_2}{(C_1 \vee C_2)\sigma}$$

7.8 Algorithme

$S := \{\Delta x_1, ..., \Delta x_i, \neg Th\}$ dont chaque formule est en forme normale conjonctive (FNC).

while : **false** $\ni S$ et il existe une paire de clauses résolvables et non-résolues.

do : choisir p_i, p_j dans S et L tel que :
— L^+ dans p_i
— L^- dans p_i
— $\{L^+, L^-\}$ unifiable par σ U.P.G.
Calculer :
— $r := (p_i - [L^+] \vee p_j - [\neg L^-])\sigma$
— $S_i = S \cup \{r\}$
end

If **false** $\ni S$ then Th prouvé. else Th non prouvé. end

7.9 Exemple

— $(\forall x) homme(x) \wedge fume(x) \Rightarrow mortel(x)$
— $(\forall x) animal(x) \Rightarrow mortel(x)$
— $homme(Ginzburg)$
— $fume(Ginzburg)$
— **Candidat-théorème :** $mortel(Ginzburg)$

7.9.1 Initialisation de S

— P1 : $(\forall x) homme(x) \wedge fume(x) \Rightarrow mortel(x)$
— P2 : $(\forall x) animal(x) \Rightarrow mortel(x)$
— P3 : $homme(Ginzburg)$
— P4 : $fume(Ginzburg)$
— P5 : $\neg mortel(Ginzburg)$

7.9.2 Itérations

A
— P1 + P5
— $\sigma = \{(x, Ginzburg)\}$
— $r = P6 = \neg homme(Ginzburg) \vee \neg fume(Ginzburg)$
B
— P3 + P6
— $\sigma = \{\}$
— $r = P7 = \neg fume(Ginzburg)$
C
— P4 + P7
— $\sigma = \{\}$
— $r = false$. Inconsistence donc le candidat théorème est prouvé.

7.9.3 Non-déterminisme

Importance des choix qu'on fait.

A
— P2 + P5
— $\sigma = \{(x, Ginzburg)\}$
— $r = \neg animal(Ginzburg)$
Si on avait fait ce choix-ci pour la première itération, l'algorithme ne peut plus continuer et on doit faire marche-arrière.

7.10 Stratégies

— Quelles paires p_i, p_j choisir ?
— Quelles L^+, L^- choisir ?

Les assistants de preuves utilisent des stratégies existantes, avec l'input de l'humain.

Le language **Prolog**, inventé en 1972 par Alain Colmerauer et Robert Kowalski, utilise volontairement des stratégies naives qui permettent de rendre l'algorithme prévisible. Les axiomes deviennent un programme, c'est

la programmation logique. Un exemple de stratégie naïve est la stratégie LUSH qui choisi de haut vers le bas les paires p_i, p_j, et de gauche à droite dans p_i.

Chapitre 8

Théorie logique

8.1 Etude des structures discrètes

Exemples de structures discrètes : entiers positifs, chaines, arbres, ensembles, relations, fonctions, ...

On peut définir ces structures avec la logique des prédicats et faire des raisonnements sur celles-ci en utilisant des règles d'inférence.

Utilisation :

- On peut programmer avec ces structures. *Prolog* permet d'écrire les axiomes directement. Il faut cependant faire attention de bien les choisir ;
- On peut les utiliser dans les assistants de preuve (exemples d'assistants de preuve : *Coq, Isabelle*)

8.2 Théorie du premier ordre

Intuition : définition logique d'une structure mathématique

8.2.1 Définition d'une théorie

- Sous-language de la logique du premier ordre
  - **vocabulaire** : constantes, fonctions, prédicats ;
  - règles syntaxiques et sémantiques sur ce vocabulaire ;
- Ensemble d'axiomes (formules fermées, c'est-à-dire formules ne contenant pas de variables libres) ;

— Ensemble de règles d'inférence.

8.2.2 Exemple : théorie des liens familiaux (fam)

1. Vocabulaire :
   — 2 symboles de fonctions : $p/1$, $m/1$
   — 3 symboles de prédicats : $P/2$, $GM/2$, $GP/2$

   On peut interpréter les fonctions p et m comme "père de" et "mère de", et les prédicats P, GM et GP comme "parent de", "grand-mère de" et "grand-père de". On donnera plus de précisions sur cette interprétation dans la suite.

2. Axiomes :

$$(\forall x)\,(P(x, p(x))) \qquad \text{(père)}$$
$$(\forall x)\,(P(x, m(x))) \qquad \text{(mère)}$$
$$(\forall x)(\forall y)\,(P(x, y) \Rightarrow GP(x, p(y)))$$
$$(\forall x)(\forall y)\,(P(x, y) \Rightarrow GM(x, p(y)))$$

3. Règles :
   Les règles sont uniquement celles de la logique des prédicats.

Première interprétation

D_I : personnes	
$\text{val}_I(p)$ = "père de"	père de : Pers $\to$ Pers : $d \to$ "père de" d
$\text{val}_I(m)$ = "mère de"	mère de : Pers $\to$ Pers : $d \to$ "mère de" d
$\text{val}_I(P)$ = "Parent"	Parent$(d_1, d_2) = V$ ssi d_2 est un parent de d_1
$\text{val}_I(GP)$ = "Grand-père"	Grand-père$(d_1, d_2) = V$ ssi d_2 est un grand-père de d_1
$\text{val}_I(GM)$ = "Grand-mère"	Grand-mère$(d_1, d_2) = V$ ssi d_2 est une grand-mère de d_1

Cette interprétation est un modèle de FAM car les axiomes sont tous vérifiés.
On remarque la ressemblance avec une théorie scientique, où les axiomes correspondent à la théorie et l'interprétation à ce que celle-ci signifie dans le monde réel.
Une théorie peut avoir plusieurs modèles.

Seconde interprétation

Cette interprétation est également un modèle de FAM.

$D_J : \mathbb{N}$	
$\text{val}_J(p) = "p_J"$	$p_J : \mathbb{N} \to \mathbb{N} : d \to 2d$
$\text{val}_J(m) = "m_J"$	$m_J : \mathbb{N} \to \mathbb{N} : d \to 3d$
$\text{val}_J(P) = "P_J"$	$P_J(d_1, d_2)$ ssi $d_2 = 2d_1$ ou $d_2 = 3d_1$
$\text{val}_J(P) = "GP_J"$	$GP_J(d_1, d_2)$ ssi $d_2 = 4d_1$ ou $d_2 = 6d_1$
$\text{val}_J(GM) = "GM_J"$	$GM_J(d_1, d_2)$ ssi $d_2 = 6d_1$ ou $d_2 = 9d_1$

8.3 Propriétés des théories

— Une formule fermée p est **valide** dans la théorie Th si elle est vraie dans chaque modèle de Th. On écrit :

$$\vDash_{Th} p$$

Soit l'ensemble des axiomes $Ax = \{Ax_1, \ldots, Ax_n\}$. On a bien que $\vDash_{Th} Ax_i$.

— q est une **conséquence logique** de p dans la théorie Th si q est vraie dans tous les modèles de Th qui rendent p vraie. On écrit :

$$p \vDash_{Th} q$$

— Une théorie est **consistante** si elle a au moins un modèle (> 0 modèles).
— Une théorie est **inconsistante** si elle n'a pas de modèle (0 modèles).

Comment peut-on faire pour établir $\vDash_{Th} p$? Il y a deux approches différentes :

1. **l'approche sémantique** : on prend un modèle quelconque de Th et on évalue $\text{VAL}_I(p)$ en utilisant le fait que $\text{VAL}_I(Ax_i) = V$;
2. **l'approche syntaxique** : théorie de preuve : on essaye de construire une preuve de p à partir des axiomes, en appliquant les règles de Th.

En pratique, le deuxième approche est utilisée beaucoup plus souvent.

Preuve (esquisse)

La preuve qui suit est une esquisse. Il manque plusieurs étapes.
On veut montrer :

$$\vDash_{\text{FAM}} (\forall x)(\exists z) GM(x, z)$$

$(\forall x)(\forall y)\,(P(x,y) \Rightarrow GM(x,m(y)))$	(Ax)
$(\forall x)\,(P(x,p(x)) \Rightarrow GM(x,m(p(x))))$	(Elimination de $\forall y$ et substitution $y/p(x)$)
$(\forall x\ P(x,p(x))) \Rightarrow \forall x\ GM(x,m(p(x)))$	(Distribution $\forall/\Rightarrow$)
$\forall x\ GM(x,m(p(x)))$	(Modus ponens)
$\forall x\ \exists y\ GM(x,y)$	(Introduction de $\exists$)

8.4 Qualité d'une théorie

Certaines qualités sont directement issues de la logique :

1. **consistante** : il est impossible de déduire p et $\neg p$ de la même théorie.
2. **minimale** : les axiomes son indépendants : $\{Ax_1, \ldots, Ax_n\} \nvdash Ax_k$
   Vérification : Construction d'interprétations.
   $\mathrm{VAL}_J(Ax_k) = False$
   $\mathrm{VAL}_J(Ax_i) = True$ pour $i \neq k$
3. **complète** : les axiomes suffisent pour prouver la propriété d'intérêt. Sinon il faut en ajouter.

8.4.1 Exemple : qualité de deux théories

1. **Système de Copernic** : Chaque axiome de chaque planète est indépendant car elles tournent toutes autour du soleil.
2. **Système de Ptolémée** : Chaque axiome de chaque planète dépendent de ceux de la Terre car elle représente le centre du l'univers mais est également une planète et dispose donc d'axiomes.

8.5 Extension d'une théorie

Lorsqu'on a une théorie déjà existante, on souhaite parfois l'étendre afin de la rendre plus complète. Pour ce faire, on ajoute des axiomes et on étend le vocabulaire. Voici deux exemples d'extension de théorie pour mieux comprendre comment effectuer cette opération. Ils étendent tous les deux la théorie des liens familiaux (FAM) décrite dans une section précédente.

Exemple 1

Considérons le nouvel axiome suivant, que nous noterons Ax.

$$(\forall x)\neg P(x,x)$$

La nouvelle théorie ainsi étendue que nous noterons FAM* possède un axiome de plus : Ax. Cette théorie FAM* est **consistante**, c'est-à-dire qu'il existe au moins un modèle qui valide cette théorie. Pour s'en convaincre, il suffit de considérer la première interprétation de la théorie FAM de la section précédente, qui utilise les liens familiaux.

En revanche, si on considère la deuxième interprétation (deuxième modèle, noté J) de FAM qui associe les symboles p et m aux fonctions mathématiques $p_J : \mathbb{N} \to \mathbb{N} : d \to 2d$ et $m_J : \mathbb{N} \to \mathbb{N} : d \to 3d$, on observe une contradiction. En effet, dans le modèle J on a la définition suivante du prédicat P :

$$P_J(d_1, d_2) \text{ ssi } d_2 = 2d_1 \text{ ou } d_2 = 3d_1$$

Il suffit de choisir $x = 0$ dans notre nouvel axiome Ax pour constater que le modèle J ne valide pas la théorie étendue FAM*. De manière générale, l'extension d'une théorie peut donc réduire l'ensemble des modèles de celle-ci.

Exemple 2

Considérons à présent le nouvel axiome suivant, que nous noterons Adam.

$$(\forall y)\neg P(a,y)$$

où a est une constante arbitraire. Notons la théorie étendue FAM' = FAM + Adam. Dans cet exemple, on peut observer que FAM' est **inconsistante** car aucun modèle ne peut valider cette théorie. En effet, en partant du premier axiome de FAM (appelé "père"), nous effectuons quelques étapes

pour obtenir une contradiction.

$$\begin{aligned} & (\forall x)P(x, p(x)) & \\ \iff & P(a, p(a)) & \text{Elimination } \forall \\ \iff & (\exists y)P(a, y) & \text{Intro } \exists \end{aligned}$$

Ci-dessus, le premier axiome de FAM reformulé (père), qui est en contradiction avec le nouvel axiome (Adam), que nous reformulons ci-dessous.

$$\begin{aligned} & (\forall y)\neg P(a, y) \\ \iff & \neg(\exists y)P(a, y) \end{aligned}$$

Par la règle de preuve par contradiction, on démontre qu'aucun modèle n'est possible pour la théorie étendue FAM'. Étendre une théorie équivaut à faire de la manipulation syntaxique. Il est important de bien vérifier ce que notre modification a comme conséquence sur le nombre de modèles que la théorie accepte.

8.6 Liens entre théories

Dans cette section nous abordons la comparaison de différentes théories : inclusion, équivalence et quelques corollaires, ainsi que la théorie des ordres partiels stricts. Dans ce qui suit, on note Th_1 et Th_2 deux théories.

Inclusion

On dit que Th_1 est **contenue** dans Th_2 si

- Le vocabulaire de Th_1 est inclus dans le vocabulaire de Th_2.
- Toute formule valide dans Th_1 l'est aussi dans Th_2.

Attention, on peut donc avoir deux théories qui "parlent de la même chose" mais qui possèdent des axiomes totalement différents. TODO : expliquer pourquoi / Ajouter un exemple

Equivalence

On dit que Th_1 et Th_2 sont **équivalentes** si elles sont contenues l'une dans l'autre. Cela signifie que les deux théories "disent la même chose" et que tout modèle d'une des théorie est également modèle de l'autre.

Il est important de bien faire la différence entre les **liens** entre les théories et l'**extension** d'une théorie. Le premier concept exprime ce que modélisent les théories tandis que le deuxième n'est que de la manipulation syntaxique.

Corollaires

On note respectivement V_i, M_i et Ax le vocabulaire, les modèles et les axiomes d'une théorie i.

Si $V_{Th_1} \subseteq V_{Th_2}$ et $M_{Th_2} \subseteq M_{Th_1}$ alors Th_1 est contenue dans Th_2.

Si $V_{Th_1} \subseteq V_{Th_2}$ et tout axiome de Th_1 est aussi axiome de Th_2 alors Th_1 est contenue dans Th_2.

Si $V_{Th_1} = V_{Th_2}$ et $\forall i, j \quad \vDash_{Th_2} Ax_{i,1}$ et $\vDash_{Th_1} Ax_{j,2}$ alors Th_1 et Th_2 sont équivalentes.

Si p une formule fermée telle que $\vDash_{Th_1} p$ et $Th_2 = Th_1 \bigcup \{p\}$ alors Th_1 et Th_2 sont équivalentes.

8.7 Théorie des ordres partiels stricts

Nous allons donner un autre exemple de théorie ainsi que deux interprétations différentes de cette théorie.

Vocabulaire

- Le symbole P

Axiomes

- $(\forall x)\neg P(x, x)$ (irréflexivité) (OPS1)
- $(\forall x)(\forall y)(\forall z)(P(x, y) \wedge P(y, z) \Rightarrow P(x, z))$ (transitivité) (OPS2)

Première interprétation

Soit l'interprétation I de cette théorie. On a :

- $D_I = \mathbb{Z}$
- $val_I(P) =$ " $<$ "

Lorsque l'on écrit $val_I(<) =$ " $<$ ", le premier symbole $<$ est un mot de vocabulaire dans la théorie tandis que le deuxième est une fonction. Cette fonction confère un sens au symbole. On remarque que cette interprétation est un modèle de notre théorie. En effet, la fonction $<$ sur les entiers est

irréflexive et transitive. Si on a $x < y$ et $y < z$, on peut en déduire que $x < z$.

Deuxième interprétation

Soit l'interprétation J de cette théorie. On a :

- $D_J = \mathbb{Z}$
- $val_J(P) = " \neq "$

Ici on change le sens du symbole $<$:

$$val_J(<) = " \neq "$$

Cette interprétation n'est pas un modèle. En effet, par exemple, pour $x = 5, y = 3, z = 5$, on a :

$$5 < 3 \wedge 3 < 5 \nRightarrow 5 < 5$$

Ceci est un contre-exemple car 5 n'est pas différent de 5 (irréflexivité), et donc cette interprétation ne vérifie pas l'axiome sur la transitivité !

8.8 Théorie de l'égalité (EG)

Il existe différents symboles pour représenter l'égalité :
— E(x,y)
— $x = y$
— $x == y$
Nous allons utilisé $'==$' dans la suite de ce cours pour représenter l'égalité.

8.8.1 Axiomes

Pour définir ce qu'est une égalité, nous avons d'abord besoin de définir 3 axiomes et 2 schémas d'axiomes.

(eg1) Réflexivité

$\forall x, x == x$

(eg2) Symétrie

$\forall x, \forall y, x == y \Rightarrow y == x$

(eg3) Transitivité

$\forall x, \forall y, \forall z, (x == y \wedge y == z) \Rightarrow x == z$

(eg4) Substituabilité dans les fonctions

$\forall x_1, ..., x, ..x_n \ , \ x_i == x \Rightarrow f(x_1, ..., x_i, ..x_n) == f(x_1, ..., x, ..x_n)$

(eg5) Substituabilité dans les prédicats

$\forall x_1, ..., x, ..x_n \ , \ x_i == x \Rightarrow P(x_1, ..., x_i, ..x_n) == P(x_1, ..., x, ..x_n)$

8.8.2 Régles d'inférences

En plus de ces 5 axiomes, il nous faut aussi définir deux régles d'inférences.

Substituabilité fonctionnelle

$$\frac{s_1 == t_1 \wedge s_2 == t_2 \wedge ... \wedge s_n == t_n}{f(s_1, s_2, ..., s_n) == f(t_1, t_2, ..., t_n)}$$

Subsituabilité prédicative

$$\frac{s_1 == t_1 \wedge s_2 == t_2 \wedge ... \wedge s_n == t_n}{P(s_1, s_2, ..., s_n) == P(t_1, t_2, ..., t_n)}$$

Grâce aux règles sémantiques de l'égalité, on peut raionner sur des formules mais aussi bien sur une interprétation.

Si $VAL_I(t_1) == VAL_I(t_2)$ alors $VAL_I(t_1 == t_2) = true$

Preuve (Métalangage)

Soit I, un modèle de EG ($'=='$) pour t_1 et t_2.

$VAL_I(t_1 == t_2) = VAL_I(==)(VAL_I(t_1), VAL_I(t_2))$

On pose $VAL_I(==) = E_I$ et $VAL_I(t_1) = e$ et $VAL_I(t_2) = e$

Donc $VAL_I(t_1 == t_2) = E_I(e, e)$

$\Rightarrow$ preuve en regardant les axiomes

$VAL_I(\forall x, x == x) = true$ (eg1)

On cherche le $\forall x$ dans la sémantique :

On sait que pour tout d $\in$ Domaine de I, $VAL_I(d == d) = true$,

Si $d = e$

alors $E_I(e, e) = true$

8.8.3 Remarque

La théorie de l'égalité a un utilité très limitée, elle doit être étendue pour pouvoir servir à quelque chose. On appelle une telle théorie un template.

8.9 Théorie de l'ordre partiel (OP)

On ajoute un deuxième symbole dans le langage en plus du symbole d'égalité.

— $==$

— $\leq$

8.9.1 Axiomes

Aux axiomes et schémas d'axiomes de la théorie de l'égalité, on rajoute de nouveaux axiomes

(op1) Réflexivité

$\forall x,\, x \leq x$

(op2) Anti-symétrie

$\forall x,\, \forall y,\, x \leq y \wedge y \leq x \Rightarrow y == x$

(op3) Transitivité

$\forall x,\, \forall y,\, \forall z,\, (x \leq y \wedge y \leq z) \Rightarrow x \leq z$

(op4) Substituabilité à gauche

$\forall x_1,\, \forall x_2,\, \forall x,\, x_1 == x \Rightarrow x_1 \leq x_2 \Leftrightarrow x \leq x_2$

(op5) Substituabilité à droite

$\forall x_1,\, \forall x_2,\, \forall x,\, x_2 == x \Rightarrow x_1 \leq x_2 \Leftrightarrow x_1 \leq x$

Théorème : $\models \forall x, \forall y, [x == y \Leftrightarrow (x \leq y) \wedge (y \leq x)]$

8.9.2 Preuve

La preuve va être démontrée en partie en métalangage et en partie en preuve formelle.

$\Leftarrow$: est démontré par l'axiome anti-symétrique

$\Rightarrow : \models_{op} \forall x, \forall y, [x \leq y \wedge y \leq x]$

preuve formelle :

(1)$\forall x, \forall y, x == y \Rightarrow x \leq x \Leftrightarrow y \leq x$ substituabilité à gauche

(2)$\forall x, \forall y, x == y \Rightarrow x \leq x \Leftrightarrow x \leq y$ substituabilité à droite

(3)$x == y \Rightarrow x \leq x \Leftrightarrow y \leq x$ $\forall$ élimination

(3)$x == y \Rightarrow x \leq x \Leftrightarrow x \leq y$ $\forall$ élimination

preuve conditionnelle :

(4)$x == y$ supposition

(5)$y \leq x$ modus ponens (1,4)

(6)$x \leq y$ modus ponens (2,4)

(7)$y \leq x \wedge x \leq y$ conjonction (5,6)

(8)$x == y \Rightarrow x \leq x \Leftrightarrow y \leq x \wedge x \leq y$

(9)$\forall x, \forall y, x == y \Rightarrow x \leq x \Leftrightarrow y \leq x \wedge x \leq y$ $\forall$ Introduction

8.9.3 Exemples de Modèles d'OP

$\underline{I_1} : D_{I_1} = \mathbb{Z}$

$val_{I_1}(==) = '='$: égalité d'entiers

$val_{I_1}(\leq) = '\leq'$: plus petit ou égal pour les entiers

cette interprétation va satisfaire tous les entiers.

$\underline{I_2} : D_{I_2} = P(E)$ ensemble des sous-ensembles de E

$val_{I_2}(==) = '='$: égalité d'ensemble

$val_{I_2}(\leq) = '\subseteq'$: inclusion d'ensemble

$\underline{I_3} : D_{I_3} = ALPH^2 = (l_1, l_2), ..$ doublons de lettres de l'alphabet

$val_{I_3}(==) =$ égalité des paires

$val_{I_3}(\leq) =$ suivant ordre lexicographique

exemple : $(l_i, l_j) \leq (l_p, l_q)$ si $l_i < l_p$ ou $l_i = l_p$ et $l_j < l_q$ ou $l_j = l_q$

$\underline{I_3}$: D_{I_4} = ensemble de listes

$val_{I_4}(==)$ = = égalité de liste (se elles possèdent les même composants à la même position

$val_{I_3}(\leq)$ = "suffixe de"

exemple : l_1 est suffixe de l_2

l_1 = [d, e, f, g, h] et l_2 = [a, b, c, d, e, f, g, h, i]

I_5 : Soit D_{I_5} l'ensemble des formules en logique des propositions. On commence à utiliser la logique pour parler d'elle-même
$VAL_{I_5}(==)$ = "<≡>" → L'égalité est synonyme d'équivalence en logique des propositions
$VAL_{I_5}(\leq)$ = "⊨" → Le signe d'inclusion entre les ensembles de modèles
Maintenant qu'on a défini la sémantique de l'ordre partiel en utilisant la notion de modèles, on peut prendre quelques exemples en raisonnant sur la logique :

$p \leq_{I_5} q$ $\quad$ $p \models_{I_5} q$ $\quad$ $\models p \Rightarrow q$

I_6 : D_{I_6} = l'ensemble des unificateurs d'un ensemble S de termes ou de formules
$VAL_{I_6}(==)$ = égalité entre substitutions { (x_i,t_i)... } : un ensemble de paires avec une variable et un terme
$VAL_{I_6}(\leq)$ = "moins général que"

Exemple : P(x,f(y)) $\quad$ $\underbrace{P(x,z)}_{\sigma}$
σ' = $\sigma \cup$ {(z, f(y)} donc $\sigma' \leq \sigma$

Ces exemples montrent qu'on peut raisonner sur tout, y compris sur la logique et les algorithmes eux-mêmes, à partir du moment où on respecte les axiomes.

8.10 Théorie des ensembles

Elle sera utilisé pour la spécification des systèmes $\mathbb{Z}$.

Ensemble : on peut définir les ensembles de façon informelle (c'est-à-dire sans les axiomes) :

- Un ensemble peut être défini comme une énumération d'éléments
  Ex : { 0,20,40,60,80,100 } ou {Mercure, Vénus, Terre, ..., Neptune}
- Il peut également être défini comme le prédicat d'un argument qui est vrai pour les membres. On appelle cette définition informelle "compréhension)". Certains langages comme Python possèdent des syntaxes particulières dédiées à la création de listes remplies par des compréhensions
  Ex : $\{ n \mid n \in \mathbb{N} \wedge \exists k, k \in \mathbb{N} \wedge 20k = n \wedge 0 \leq n \leq 100 \}$

La définition formelle des ensembles passe par 8 axiomes :
Axiomes

- **Egalité** (1 ère définition)
  $A = B \Leftrightarrow (\forall\ x)\ (x \in A \Leftrightarrow x \in B)$

- **Ensemble vide** (sans éléments)
  $\emptyset = \{x \mid x \neq x\}$

- **Ensemble universel** (tous les éléments du domaine)
  $\mathcal{U}$ (par exemple : $\mathbb{N}$, $\mathbb{R}$, $\mathbb{Z}$)

- **Sous-ensemble**
  $A \subseteq B \Leftrightarrow (\forall\ x)\ (x \in A \Rightarrow x \in B)$
  $A \nsubseteq B \Leftrightarrow \neg(A \subseteq B)$
  $A \subset B \Leftrightarrow A \subseteq B \wedge A \neq B$

- **Construction de sous ensemble**
  $A \subseteq B \Leftrightarrow (x \in A \Leftrightarrow x \in B \wedge \varphi(x))$
  On va ici prendre des éléments de B tels que $\varphi(x)$ est vrai, et les mettre dans A. C'est un peu une formulation de la notion de compréhension.

- **Propriétés**
  $\emptyset \subseteq A$
  $A \subseteq A$
  $A \subseteq B \wedge B \subseteq C \Rightarrow A \subseteq C$

- **Egalité** (2 ième définition)
  $(A = B) \Leftrightarrow (A \subseteq B) \wedge (B \subseteq A)$

- **Taille d'un ensemble**
  - Prédicat à 2 arguments
    $\#(A,n)$
    $\#\emptyset = 0 \rightarrow \#(\emptyset, 0)$
  - $\mathbb{P}A = \{a \mid a \subseteq A\}$ (prédicat à 2 arguments)
    $\#\mathbb{P}A = 2^{\#A}$

Spécification formelle des systèmes avec l'approche $\mathbb{Z}$

Cette approche est "Orientée modèle" : On ne donne pas que des équations, mais aussi des structures

Deux concepts en $\mathbb{Z}$:

Types :

— Types génériques (ensembles) : par exemple : Book, Location
— Types énumérés : par exemple : Statuts := $\underbrace{In}_{\textit{Dans la bibliothèque, disponible}}$

$| \overbrace{Out}^{\textit{Prêté}} | \quad \underbrace{Ref}_{\textit{Livre de référence}}$

Schémas :

— Etat du système : → logique des prédicats
— Comportement actions : préconditions et posconditions → également logique des prédicats

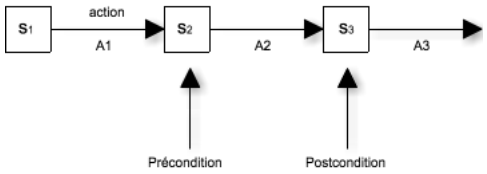
Exemple d'un état

nom	LIB_STOCK
signature	stock on_loan on_shelf ref_coll : $\mathbb{F}$ Book
prédicat	stock = on_loan ∪ on_shelf on_loan ∩ on_shelf = ∅ ret_call ⊆ on_shelf

$\mathbb{F}$ Book représente un sous-ensemble fini de l'ensemble Book.

Exemple du comportement

— Il y a un "avant" et un "après"
Les variables "avant", on les écrit normalement (sans rien changer) alors qu'on ajoute une apostrophe aux variables "après" pour pouvoir les distinguer. Ex : Stock et Stock'
— Une exécution est une séquence d'états



— Notation $\mathbb{Z}$
Δ Lib_Stock $\triangleq$ Lib_Stock $\wedge$ Lib_Stock'
$\equiv$ Lib_Stock $\triangleq$ Lib_Stock $\wedge$ (Lib_Stock' | Lib_Stock = Lib_Stock')
Cette deuxième ligne d'exemple veut dire la même chose, mais dans la condition où Lib_stock n'a pas changé.

nom	ADD_STOCKo
signature	Δ lib_stock new ? : $\mathbb{F}$ Book
prédicat	new ? $\neq \emptyset \rightarrow$ précondition new ? $\wedge$ stock $= \emptyset \rightarrow$ précondition stock' = stock $\cup$ new ? $\rightarrow$ postcondition on_loan' = on_loan $\rightarrow$ postcondition

Le " ?" signifie qu'il s'agit d'une entrée.
On peut déduire de ces prédicats que new ? sera dans on_shelf.

— Si la (les) précondition(s) ne sont pas satisfaites, on aura des erreurs que l'on devra corriger via la gestion d'erreurs. Si la (les) précondition(s) n'échoue(nt) jamais, on a affaire à une opération totale.

Exemple d'opération totale :

ADD_STCK = AdDD_STCKo $\cup$ ERRONEOUS_NEW_STOCK

nom	ERRONEOUS_NEW_STOCK
signature	$\equiv$ lib_stock new ? : # Book rep ! : Report
prédicat	(new ? $= \emptyset \cup$ new ? $\cap$ stock $\wedge \emptyset$) ... $\rightarrow$ préconditions rep ! = "Attention stock problem"

Le " !" signifie qu'il s'agit d'une sortie.

8.11 Introduction à la programmation logique

8.11.1 Introduction à la programmation logique

Prolog est l'un des principaux langages de programmation logique. Il est à la base de nombreux fondements.

La programmation logique fait de la déduction sur les axiomes. On utilise la logique comme un langage de programmation : on va adapter l'algorithme de réfutation vu précédemment

Le programme (ressemble à une théorie) :
— Axiomes en logique des prédicats
— Une requête, un but (=goal)→ le but du système est d'apporter une preuve
— Un prouveur de théorème → attention : il faut des conditions sur le prouveur car il faut être capable de prévoir le temps et l'espace utilisé par le programme.

Exécuter un programme = faire des déductions en essayant de prouver le but. Mais est-ce que cette idée peut donner un système de programmation pratique ?

Il y a une compromis entre expressivité et efficacité : Si c'est trop expressif, ça devient moins efficace, par contre si c'est trop peu expressif, on ne peut rien programmer, ça ne sert à rien non plus. Il faut donc être expressif tout en restant efficace. Le Prolog offre un bon équilibre entre expressivité et efficacité.

Mais pour arriver à cela, il y a quelques problèmes à surmonter :

a. un prouveur est limité :
vérité = $p \models q$ (= q est vrai dans tous les modèles de p)
preuve = $p \vdash q$
$p \models q \Rightarrow p \vdash q$ (= Si c'est vrai dans tous les modèles, on peut trouver une preuve)
Si $p \models q$ alors l'algorithme se terminera. Cependant, on ne peut pas trouver les preuves que pour des choses vraies dans tous les modèles. (Comme c'est impossible on ne prend qu'une partie des modèles, ce qui limite le programme).
b. Même si on peut trouver une preuve, le prouveur est peut-être inefficace (utilise trop de temps ou de mémoire) ou imprévisible. → On ne peut pas raisonner sur l'efficacité du prouveur.
c. La déduction faite par le prouveur doit être constructive

Si le prouveur affirme : $(\exists X)P(X)$ alors le prouveur doit donner une valeur de x (c'est quoi x).

Il faut construire un résultat.

Pour résoudre ces problèmes...

1. Restrictions sur la forme des axiomes.
   typiquement :

$$(\forall X_1)...(\forall X_n)A_1 \wedge A_2 \wedge ... \wedge A_n \Rightarrow A \tag{8.1}$$

$$C_i \neg A_1 \vee \neg A_2 \vee ... \vee \neg A_n \vee A \tag{8.2}$$

   Il n'y a qu'un seul littéral sans négation. (Pour prouver A, il faut prouver A_1 ; A_2, ..., A_n).
   C_i est la clause. Le programme tout entier est une série de clauses :

$$C_1 \wedge C_2 \wedge ... \wedge C_k \tag{8.3}$$

2. Le programmeur va aider le prouveur. Par exemple : il faut commencer par prouver A_1 puis A_2... dans cet ordre là.
   $\rightarrow$ Le programmeur donnera des heuristiques. Attention : ces heuristiques ne changent pas la sémantique logique du programme. Elles ont seulement un effet sur l'efficacité !

$$C_1 = \neg A_1 \vee \neg A_2 \vee ... \vee A_n \vee A \tag{8.4}$$

$$C_2 = \neg B_1 \vee \neg B_2 \vee ... \vee B_k \vee A \tag{8.5}$$

Le langage Prolog utilise ces 2 ordres.

Bref historique :

1. 1965 : La règle de résolution a été inventée par A. Robinson
2. 1972 : Invention du langage Prolog / premier interprète (de Prolog) par A. Calmerauer, R. Kowalski et Ph. Roussel. Ils voulaient faire un langage de programmation logique, et connaissaient les différentes formes de logique ainsi que la résolution. Ils ont donc inventé un langage très simple qu'ils ont appelé Prolog (pour Programmation Logique). Il s'avère que ce langage a un compromis très intéressant par rapport à la tension entre efficacité et expressivité. Aujourd'hui, on peut faire une implémentation extrêmement efficace de Prolog. Il est extrêmement expressif, ce qui permet de faire des programmes complexes. C'est un langage à part entière.

8.11.2 Introduction à Prolog

En Prolog, on a des clauses (règles) :

$$A1 \leftarrow B1, \ldots, Bn \tag{8.6}$$

(On peut prouver A en prouvant $B1$ jusqu'à Bn. Remarque :←ou :-)

$$\neg(B_1 \wedge ... \wedge B_n) \vee A_1 \tag{8.7}$$

$$(\neg B_1 \vee \neg B_2 \vee ... \vee \neg B_n \vee A_1) \tag{8.8}$$

Programme = ensemble de clauses.

Exemple d'un petit programme en Prolog : (extrait du livre « The Art of Prolog » par L. Sterling et E. Shapine)

Règle : $grandpere(x, z) \leftarrow pere(x, y), pere(y, z)$. ($x$, y,... sont des variables. En Prolog, elles sont souvent en majuscule)

Faits : $pere(terach, abraham)$ (terach, abraham,... sont des constantes)
$pere(abraham, isaac)$
$pere(haram, lot)$
...

→ Syntaxe clausale : $pere(terach, abraham) \wedge pere(abraham, isaac) \wedge pere(haram, lot) \wedge (\neg pere(x, y) \vee \neg pere(y, z) \vee grandpere(x, z))$

Il existe une corrélation évidente entre Prolog et les bases de données. Elles ont été inventées quasi au même moment et aujourd'hui Prolog est utilisé comme une sémantique pour les bases de données déductibles. Ici, on peut avoir une relation avec deux colonnes qui auraient l'argument père. le grand-père serait une combinaison de ces deux relations.

Prolog peut être vu comme une sorte de base de données relationnelle mais beaucoup plus puissante : on peut faire des programmes qui sont plus que des simples requêtes, avec des calculs beaucoup plus complexes.

8.11.3 Algorithme d'exécution de Prolog

Dans la version de l'algorithme de preuve par résolution, l'ensemble S grandit (ce qui n'est pas très efficace).

L'idée :

— On commence par mettre le but (G) que l'on veut prouver dans r (sans négation).

— Ensuite, jusqu'à ce que r soit vide, on prend le premier littéral dans $r(A_1)$.
— Puis, on parcourt un à un les axiomes de P (P est le programme, la base de faits) pour trouver une clause Ax_i unifiable avec A_1 au moyen de $\sigma(u.p.g)$
  — Si on trouve une telle clause, on ajoute à r les littéraux de Ax_i après unification avec A_1 (et on recommence au début).
  — Si on ne trouve pas de clause unifiable, on revient sur le dernier choix (Par exemple, pour un A_1 qui aurait plusieurs clauses unifiables, on a dû en choisir une. Et bien, on retourne en arrière pour en choisir une autre, sans oublier de modifier r. (En effet, il faut éliminer les résultats de toutes les unifications qui ont été réalisées entre le moment où le point de choix a été mémorisé et le moment du retour en arrière.))
  — Si on épuise tous les choix sans que r soit vide alors nous sommes en cas d'échec.
— Lorsque le programme s'arrête, si r est vide, on a un résultat.

Programme : $P = Ax_1, ..., Ax_n$

Un « but » (un goal, une « requête ») G ($\simeq$ théorie)

$r :=< G > \ldots$ résolvante (une séquence de littéraux $\rightarrow$
$r =< A_1, A_2, ..., A_m >$ Il n'y a qu'un seul r.)

while *r est non vide* **do**
  — Choisir un littéral A_1 dans r (on prend le premier littéral)
  — Choisir une clause $Ax_1 = (A \leftarrow B_1, ..., B_k)$ dans P. (D'abord on prend la première clause, puis la suivantes jusqu'à ce qu'on trouve une clause unifiable avec A_1. Si aucune clause n'est unifiable on revient sur le dernier choix (backtrack))
  — Nouvelle résolvante $:=< B_1, ..., B_k, A_2, ..., A_m > \sigma$
  — $G' = G\sigma$

end

if *r est vide* **then**
  le résultat est le dernier G'

end

if *on épuise les choix sans que r soit vide* **then**
  le résultat est NON. (On n'a pas prouvé G). (Attention : G est peut-être vrai, mais les heuristiques ne suffisent pas pour le prouver.)

end

On peut également avoir une boucle infinie (l'algorithme est non déterministe).

8.12 Exemples de programmes Prolog

Programme (code Prolog)

Ce petit programme permet de calculer la factorielle d'un nombre. Il s'agit de clauses exprimant des faits, des règles et des questions. Le code commence par un fait : 0!==1. Ensuite, il défini une clause pour les factorielles de façon généralisée. Attention, les virgules et les points sont importants. Les virgules sont les séparateurs entre les littéraux dits négatifs tandis que le point marque la fermeture de la clause.

```
fact(0,1)
fact(N,F) :- N>0,
 N1 is N-1
 fact(N1,F1),
 F is N* F1 .
```

Requête

Nous allons maintenant exécuter le programme fact afin de trouver la factorielle de 5 et stocker la réponse dans la variable F. Le prompt Prolog est représenté par " |?- " et la sortie standard par "->". Les commentaire sont après les "%" ou entre "/* */". Il ne faut pas oublier le point à la fin de la requête.

```
|?- fact (5,F). %requête
-> F=120 % réponse
```

Forme clausale

Le programme réécrit sous forme clausale.

$$
\begin{aligned}
& fact(0,1) \\
& \quad \wedge \\
& \quad (\neg n > 0 \\
& \quad \vee \neg minus(n1, n, 1) \\
& \quad \vee \neg fact(n1, f1) \\
& \quad \vee \neg times(f, n, f1) \\
& \quad \vee fact(n, f))
\end{aligned}
$$

Exécution

Le but de l'exécution est de prouver que $G \equiv fact(5, r)$. Pour y arriver, Prolog va faire une suite de substitutions σ afin de vérifier les affirmations contenues dans r. La substitution finale contiendra le résultat final.

$G \equiv fact(5, r)$
$r =< fact(5, r) >$
%(1)
$\sigma = \{(n, 5), (f, r)\}$
$r =< (5 > 0), minus(n1, 5, 1), fact(n1, f1), times(r, 5, f1) > \sigma$
%(2)
$r =< minus(n1, 5, 1), fact(n1, f1), times(r, 5, f1) >$
$\sigma' = \{(n1, 4) \cup \sigma\}$
$r =< fact(4, f1), times(r, 5, f1) >$
[...]
$\sigma_{res} = \{(r, 120), ...\}$

(1) avec clause 2
(2) a>b existe dans le système prédéfini -> true

Exemple 2

Le but de cet exemple est de faire un "append" de deux listes :
append(L1, L2, L2)

Prolog :
- append ([] , L , L)
- append ([X|L1], L2, [X|L3]) :- append (L1,L2,L3)

Clausal Le programme réécrit sous forme clausale.
append(nil,l',l')
$\wedge$
($\neg$append(l_1, l_2, l_3) $\wedge$ append (cons(x, l_1), l_2, cons(x,l_3))

Execution Attention, l'exécution du programme est une preuve en soit.
G = append (cons(1,nil), cons(2, nil), l)

1. r= < append (cons(1,nil), cons(2,nil, l)> et σ_1 = (x,1), (l_1,nil), (l_2, cons(2,nil)), (cons(x,l_3),l)
2. r=< append(nil,cons(2,nil),l_3)> et σ_2 = (l', cons(2, nil), (l_3,l')
3. r=<>

Résultat l = con (1, l_3) l_3 = l' l' = cons(2,nil) l=con(1,cons(1,nil))

Exemple 3

Intéressons-nous à la manière dont prolog exécute la requête **|?- append (L1,L2,[1])** en nous basant sur la syntaxe définie lors de l'exemple 2.
Pour information, la fonction cons prend comme premier argument une valeur et comme second argument une liste. Elle place le premier argument en tête du second argument.

Requête

```
|?- append (L_{1},L_{2},[1]). %requête
-> r = <append (l_{1}',l_{2}',cons(1,nil))> % réponse

L_{1} = [],
L_{2} = [1]. %clause 1
ou
L_{1} = [1],
L_{2} = []. %clause 2
```

Forme clausale Le programme réécrit sous forme clausale.
append(nil,l',l')
$\wedge$
($\neg$append(l_1, l_2, l_3) $\vee$ append (cons(x, l_1), l_2, cons(x,l_3))

Exécution Dans cette exemple, il y a plusieurs chemins pour arriver au résultat

Si on choisi la clause 1 :

$$\sigma = (l'_1, nil), (l'_2, l'), (l', cons(1, nil))$$
$$r = <>$$
$$l'_1 = nil$$
$$l'_2 = cons(1, nil)$$

Si on choisi la clause 2 :

$$\sigma = (l'_1, cons(x, l_1)), (l'_2, l_2), (x, 1), (l_3, nil)$$
$$r =< append(l_1, l'_2, nil) >$$
$$\sigma = (l_1, nil), (l_2, l'_2), (nil, l_3)$$
$$r =<>$$
$$l'_1 = nil$$
$$l'_2 = cons(1, nil)$$

Il existe deux manières de percevoir une exécution Prolog.

1 Approche "impérative" : l'exécution est vue comme une séquence de calculs
2 Approche logique : l'exécution est vue comme une preuve. Cette approche n'existe pas dans les langages classiques, c'est ce qui fait la force de Prolog.

Prolog est présent dans beaucoup de domaine de programmation :

— Dans la gestion de base de données avec Datalog
— Dans la semantic web
— Dans la programmation par contrainte. Les substitutions sont remplacées par des relations quelconques.

Deuxième partie

Structures discrètes sur Internet

Chapitre 9

Structures discrètes sur l'internet

9.1 Ressources

Livre : sur iCampus
- Chapitre 1-5 (Graphe = modèle des réseaux) définitions, concepts (liens forts et faibles), contexte des réseaux, relations positives et négatives
- Chapitre 13 (Structure du Web)
- Chapitre 14 (Analyse des liens sur le web)
- Chapitre 18 (Lois de puissances)
- Chapitre 20 (Les phénomènes de petit monde)

9.2 Exemple et analyse de graphes

Voici quelques commentaires réalisés sur les figures du 1^{er} chapitre :
- **Figure 1.1** (ci dessous : Figure 9.2 Illustration des relations amicales entre 34 personnes dans un club de karaté. Chaque nœud représente une personne et chaque lien, un lien d'amitié entre ces personnes. On constate que tout le monde n'est pas ami avec tout le monde. Grâce à cette structure, on peut déduire certaines choses, par exemple : deux personnes ont beaucoup de liens d'amitié avec d'autres personnes mais pas entre eux.

- **Figure 1.2** Des employés dans un laboratoire de recherche (nœud) ont des liens entre eux : Lignes claires = communication e-mail. Lignes foncées = hiérarchie, organisation du laboratoire. On voit que la communication entre les gens suit relativement bien la structure

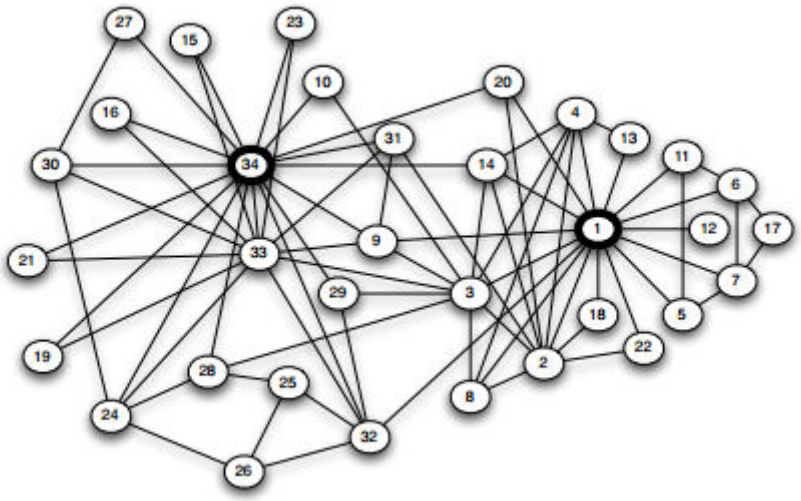


FIGURE 9.1 – Relations amicales dans un club de karaté

hiérarchique, mais pas complètement. On peut voir comment les gens collaborent, leurs degrés de collaboration, etc...

— **Figure 1.3** On constate que dans cette illustration, il y a beaucoup plus de nœuds. Chaque nœud est une institution financière (banque par exemple). Chaque nœud a un chemin vers un autre nœud (il y a des liens entre toutes les banques). Le centre est très dense, ça montre une faiblesse du système financier : si une banque dans le centre fait faillite par exemple, toutes les autres banques liées à elle sont également mises en danger . Donc si le noyau central est trop grand, c'est une faiblesse. En regardant cette structure, on peut trouver les faiblesses et les comprendre. Ca peut être très important.

— **Figure 1.4** Un nœud représente un blog politique et un lien, une référence vers un autre blog. Nous avons deux partis qui représentent chacun un noyau : les démocrates et les républicains. On constate qu'il y a moins de connexions entre les deux noyaux qu'à l'intérieur de ceux-ci. On peut visualiser cette structure et se poser des questions : est-ce que ce monde bipolaire est un problème ?

Ce sont divers exemples que nous allons essayer d'analyser. Sur Internet, il y a beaucoup de nœuds avec de grandes capacités de calcul et de stockage. On peut maintenant regarder ces structures (pas avant).

9.3 Introduction

- Structure des réseaux (Facebook, Twitter, réseau économique,...).
- Comportement des participants (interactions : chaque nœud sera un participant et va interagir).En principe chaque nœud ne voit que son voisinage et interagit en conséquence.
  - Interactions LOCALES avec conséquence GLOBALES. Il faut faire le lien entre ces deux choses.
  - Effets non attendus. Ex : réseaux routiers (nœud = automobilistes, lien = routes) :
    S'il y a des bouchons, on augmente la capacité du réseau (ajouter une voie par exemple) Le résultat peut être non intuitif, ça peut être :
    - Une réduction des transferts.
    - Une réduction du trafic. (résultat contre celui attendu)
  - Le **Paradoxe de Braess** nous dit que l'ajout d'une nouvelle capacité à un réseau peut réduire la performance globale (effet non attendu). Il faut donc comprendre comment le réseau fonctionne au lieu de faire n'importe quoi et avoir des effets non attendus.

9.4 Nouvelle discipline

Les graphes et leurs propriétés évoluent avec le temps, ce n'est pas statique.
⇒Nouvelles disciplines pour analyser des graphes Youtube, Flicker, etc...
Synthèse de 3 disciplines :

1. La théorie des graphes => mathématique
2. La théorie des jeux => mathématique Exemple : Youtube impose ses règles et ceux qui utilisent Youtube sont des joueurs.
3. La sociologie (étude des groupes sociaux) Les participants sont humains ou guidés par un humain. Ce n'est pas simplement des maths, il faut aussi comprendre les humains.

Dans ce cours, nous nous concentrerons principalement sur la théorie des graphes. La théorie des jeux sera très intuitif, et nous parlerons un peu de la sociologie.

9.4.1 Théorie des jeux

On a un ensemble de participants qui jouent à "un jeu" (un ensemble de règles suivies par tous les participants). Chaque participant doit agir :

— Simple à spécifier (comme les échecs : 2 participants et 1 action en alternance).
— Compliqué : pas d'alternance, tout le monde agit en même temps. C'est un système concurrent.

Exemple d'action simple : la vente aux enchères : n participants, règles simples (différentes techniques)

On va rester intuitif sur ce sujet, mais si on veut être plus précis, il y a des mathématiques pour ça.

— **Figure 1.8** Réseau d'interaction économique entre pays. Structure de l'économie mondial : Honkong a un gros avantage, il a une porte d'entrée vers la Chine (à l'époque). Certains pays sont des partenaires privilégie des Etats-Unis,...
— **Figure 1.9** Chemins de commerces médiévaux en Europe. L'endroit comporte des avantages : la position dans le graphe. On a toute une série d'avantages qui viennent de la structure du réseau (le comportement d'un participant peut dépendre de la structure). Si on est malin et qu'on comprend le réseau dans lequel on est, on peut essayer de se mettre dans une structure ou on a plus de pouvoir.

Si on connaît la structure, une petite action peut suffire pour arrêter une épidémie.

9.4.2 Théorie des graphes

Définitions

Graphe G = ensemble de liens et de nœuds. Un lien est une paire de deux nœuds.

G = (N,E)
N = nœud
E = edge (lien, arête)

Deux types de graphes :

— **Les graphes orientés** (avec flèches et principe de direction). Une arête/lien est une paire de nœuds. L'ordre a de l'importance. Dans cet exemple, les paires de nœuds sont : (A,B), (A,C), (D,A) et (C,D).

— **Les graphes non orientés** (sans flèche, sans direction). Une arête est un ensemble de deux nœuds. On ne parle pas de "paire" vu que l'ordre des nœuds n'a pas d'importance. Ici, parler de l'arête (A,B) ou (B,A) revient à la même chose.

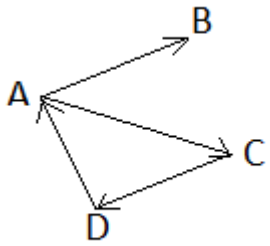


FIGURE 9.2 – Graphe orienté

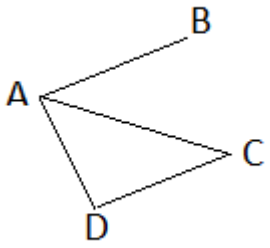


FIGURE 9.3 – Graphe non orienté

Dans le cours, on aura surtout affaire à des graphes non orientés.

Notions de chemins et de connectivité

- **Chemin** : séquence (un ordre) de nœud. Chaque paire consécutive est une arête. Un chemin peut passer plusieurs fois par la même arête (boucle), ce qui n'est pas toujours ce qu'on veut.
- **Chemin simple** : chaque nœud est au maximum une fois dans la séquence. On ne peut plus faire le tour plusieurs fois.
- **Cycle** : chemin qui arrive au même endroit d'où il est parti.
  - Le premier et le dernier nœud sont les mêmes.
  - Le chemin a au moins 3 liens (puisqu'on ne peut pas passer plusieurs fois par la même arête).

Chapitre 10

Théorie des Graphes

10.1 Chemins et connectivité

Nous allons commencer par rappeler la notion de chemin :

Chemin : c'est une séquence de nœuds dont chaque paire consécutive est une arrête.

Chemin simple : c'est un chemin dont chaque nœud se trouve au maximum une fois dans la séquence de nœuds.

Cycle : c'est un chemin dont le premier et le dernier nœud sont les mêmes. Un cycle possède au moins 3 liens (arrêtes).

Maintenant, nous allons définir la notion de connectivité d'un graphe.

Connectivité : un graphe est dit connexe si pour toutes paires de nœuds A et B, il existe au moins un chemin de A vers B.

Tous les graphes ne sont en effet pas connexes. La figure 10.1 montre un exemple de graphe non connexe. Ce graphe possède deux composants, AB et CDE. Ces composants (qui sont maximaux) sont les parties connexes du graphe. On a les définitions suivantes :

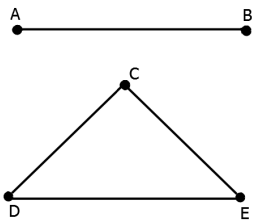


FIGURE 10.1 – Graphe non connexe

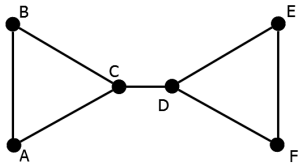


FIGURE 10.2 – Graphe connexe

Composant d'un graphe : c'est un sous-graphe (partie de graphe) qui est connexe. Il est maximal s'il ne fait pas partie d'un composant plus grand.

Composant géant : il s'agit d'un composant qui contient une fraction significative de l'ensemble des nœuds contenus dans le graphe.

Par exemple, le graphe des amitiés mondiales n'est sûrement pas connexe. En effet, il peut y avoir des habitants d'une île reculée qui se connaissent entre eux, mais qui n'ont pas de connection avec le reste du monde. Cependant, la majorité du monde est connectée ; on a donc un énorme composant géant.

C'est une propriété générale des grands graphes complexes : il est rare qu'ils soient connexes, mais ils ont très souvent un composant géant. Avoir plusieurs composants géants est instable : il arrive vite qu'un lien se forme entre les deux composants, formant ainsi un seul composant géant. Si avant le XV$^{\text{ème}}$ siècle, il y avait un composant géant eurasien et un autre américain, il n'a fallu qu'un lien (la découverte de l'Amérique par Christophe Colomb) pour assembler les deux composants, avec toutes les conséquences que cela a entrainé (maladies, exploitation, etc.).

Avec les éléments que nous venons de définir, on est en mesure d'analyser tout un graphe. On peut le partitionner en composants et regarder la structure interne de chacun d'entre eux. Par exemple, dans la figure 10.2, on peut partitionner en deux composantes, ABC et DEF. On constate que le lien CD est un lien spécial car si on l'enlève, il déconnecte de graphe.

10.2 Distance entre nœuds

Nous allons maintenant définir la longueur d'un chemin ainsi que la distance entre deux nœuds :

Longueur d'un chemin : le nombre d'arêtes consécutives sur ce chemin.

Distance entre 2 nœuds : le chemin le plus court entre ces deux nœuds.

La méthode de calcul de distance entre deux nœuds est la traversée en largeur d'abord (*breadth-first traversal*). Cela consiste à débuter par un nœud, puis à regarder tous les nœuds qui sont à une distance de 1 du nœud de départ. À partir de tous ceux-là, on regarde les nœuds qui sont à une distance 1, et donc à une distance 2 du nœud de départ. On continue jusqu'à arriver au nœud d'arrivée. On a donc, à chaque étape, constitué des couches de nœuds se trouvant à une certaine distance. Cet algorithme sera expliqué plus en détails par après.

10.3 Phénomène du petit monde

Le « phénomène du petit monde », aussi connu sous le vocable « paradoxe de Milgram » (du nom du psycho-sociologue *Stanley Milgram*), est l'hypothèse que chacun puisse être relié à n'importe quel autre individu par une courte chaîne de relations sociales. La figure 10.3 montre la probabilité qu'ont deux personnes d'être reliés par x intermédiaires. Cette courte chaîne de relations sociales a été approfondie par la théorie des « six degrés de séparation » affirmant qu'en prenant deux personnes, il est possible de trouver une chaîne d'amis entre eux de taille maximum 6. Différents types de distances entre personnes existent comme :

— Le nombre d'Erdos est la distance de collaboration avec le célèbre mathématicien *Paul Erdös* qui a réalisé de très nombreuses co-publications. Avoir réalisé une publication en collaboration avec Erdös correspond au nombre d'Erdös 1. Avoir écrit une publication avec quelqu'un qui a co-publié avec Erdös équivaut au nombre 2, etc.
— Le nombre de Bacon est la distance de collaboration dans un film avec l'acteur *Kevin Bacon*.

Ce phénomène de petit monde est particulièrement vrai pour les réseaux créés dynamiquement. Nous expliquerons plus en détails pourquoi cette affirmation est vraie dans la suite du cours.

10.4 Liens forts et faibles

Un réseau peut évoluer de différentes manières et selon différents mécanismes. Considérons un exemple où chaque nœud correspond à une personne et les arcs correspondent à un lien d'amitié (figure 10.4). Dans cet exemple, on voit aue A est ami à la fois avec B et avec C. Dans cette condition, il est fort probable que B et C deviennent eux-mêmes amis.

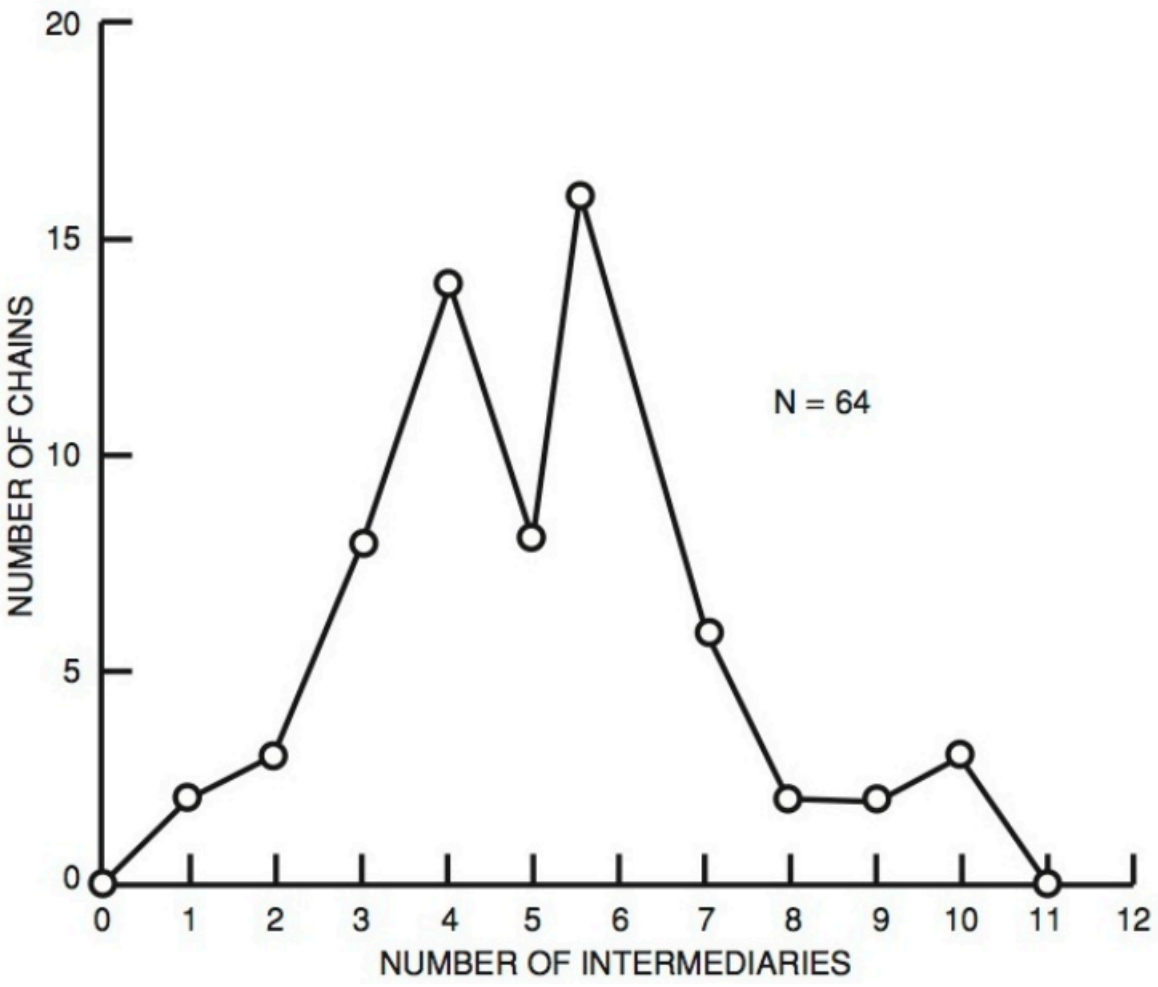


FIGURE 10.3 – Statistiques du phénomène du petit monde

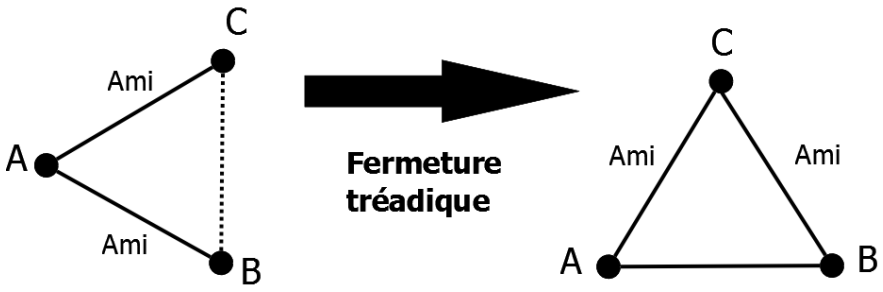


FIGURE 10.4 – Exemple de fermeture triadique

C'est ce qu'on appelle la fermeture triadique.

Cette situation nous amène à nous intéresser à la notion de coefficient de regroupement qui reflète la probabilité qu'un arc se crée entre deux nœuds dans un graphe dynamique. Dans l'exemple ci-dessus, cela correspondrait au fait que C et B deviennent amis. On peut démontrer qu'au plus on fait de fermetures triadiques, au plus le coefficient de regroupement est elevé.

Exemple

Une étude a été faite dans les années 1960, par le sociologue américain *Mark Granovetter*, dans laquelle il s'est interessé aux personnes qui changent de travail et plus particulièrement à la manière dont ils trouvent un nouveau travail. Il a remarqué que les personnes trouvent du travail plutôt via des connaissances que via des amis. Cela s'explique par la structure des graphes des amis et nous amène à définir les notions de **liens forts** et de **liens faibles** (figure 10.5). La suite de cet exemple sera expliquée plus tard.

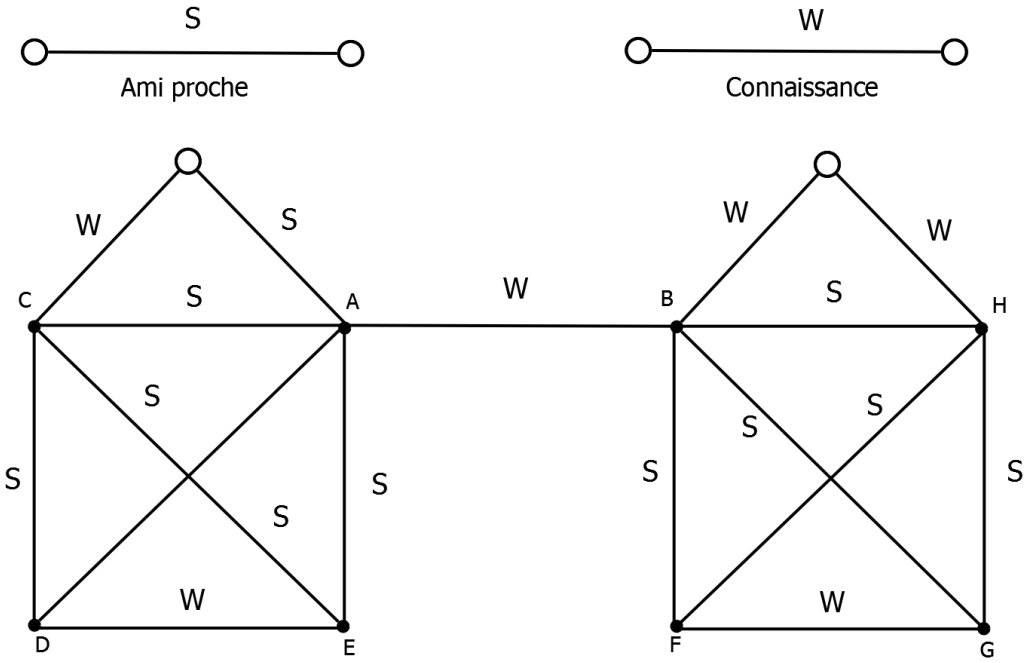


FIGURE 10.5 – Liens forts et faibles

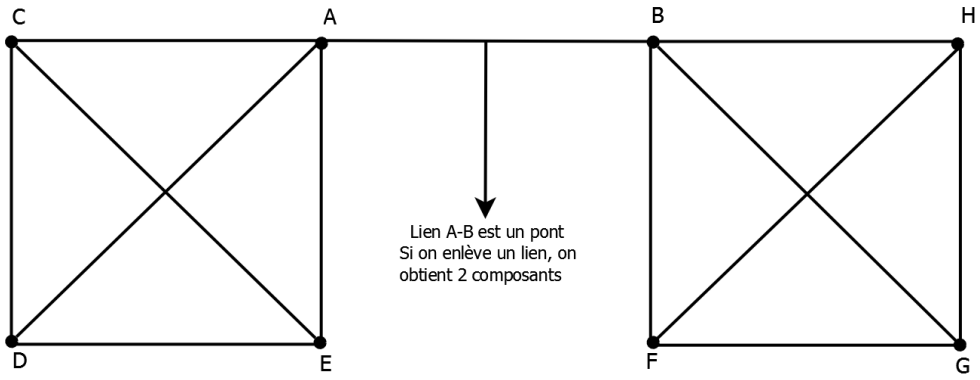


FIGURE 10.6 – Pont

10.5 Ponts

Pour expliquer l'exemple précédent, nous avons besoin de la notion de pont (figures 10.6 et 10.7).

Pont Un lien entre A et B est un pont si l'enlèvement de ce lien aboutit à la séparation du graphe en deux composants disjoints.

Pont local Un lien entre A et B est un pont local si l'enlèvement de ce lien aboutit au fait que deux composantes sont reliées par un chemin significativement plus long.

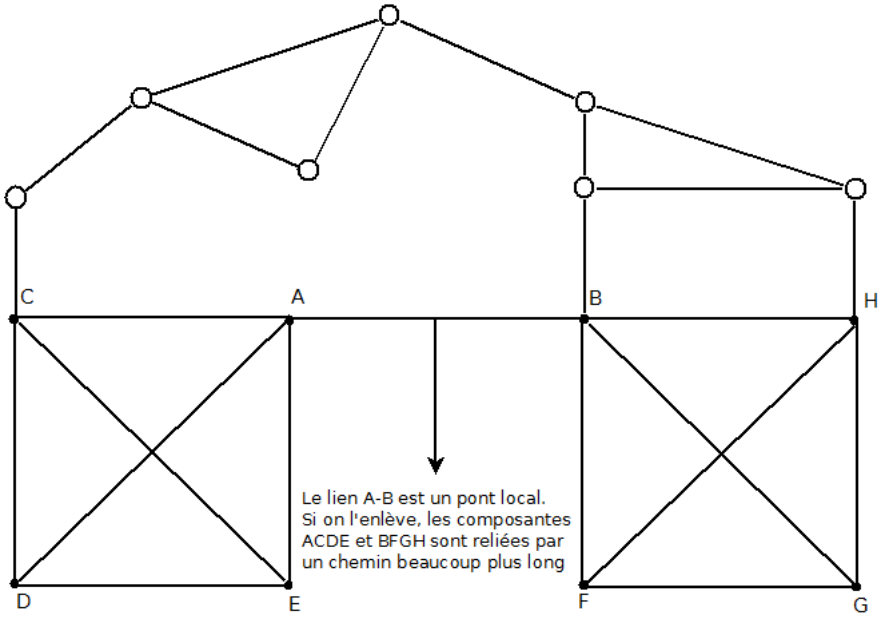


FIGURE 10.7 – Pont local

Reprenons l'exemple de la section 31.4, si l'on s'intéresse à la personne A, on sait que s' il cherche du travail c'est fort probable que ce soit via B qu'il en trouve (B est une connaissance de A).Socialement, on sait que si deux amis ont un ami en commun, il y a des chances qu'ils deviennent ami aussi ou tout du moins connaissance (propriété de fermeture triadique forte).On peut donc en déduire que chaque pont local tel que celui entre A et B dans la figure 8 sera un lien faible (conséquence de la propriété de fermeture triadique forte).En effet si le lien entre A et B était fort, la propriété de fermeture triadique forte nous dirais que d'autres liens se formeraient. Par exemple entre E et B et entre A et F ce qui aurait pour conséquence que le lien A-B ne serait plus un pont.B fourni donc de nouvelles informations à A via le pont local.

10.6 Force d'un lien dans un grand réseau

Pour caractériser la force d'un lien dans un grand réseau, il va falloir généraliser les concepts relatifs aux liens forts et faibles vus précédemment. La force d'un lien sera caractérisée en y attribuant une quantité numérique, on va utiliser la notion de chevauchement de voisinage (neighbourhood overlapping) pour ça. Chevauchement de voisinage = (Nombre de noeuds voisins de A et B)/(Nombre de noeuds voisins de A ou B) La quantité numérique augmentera en fonction de ce chevauchement. Plus cette valeur de chevauchement de voisinage tendra vers 0, plus ce lien deviendra un pont local.

10.7 Force des liens en pratique dans un réseau téléphonique

Une étude portée sur les conversations cellulaires nous prouve que plus les liens entre des invidus sont fort, plus ces personnes passeront du temps au téléphone à communiquer. Effectivement, on remarque que la durée augmente au fur et à mesure que le chevauchement de voisinage augmente.

10.8 Force des liens en pratique sur facebook

Facebook est un réseau social développé sur internet permettant à des indivdus de communiquer avec des personnes de leur entourage. Les liens

entre individus sur Facebook sont nommés les liens d'amitié. Facebook est ainsi notamment un bon exemple où la force des liens est utilisée au maximum. D'après une étude réalisée sur une période d'un mois, on trouve trois catégories de liens sur Facebook, caractérisant soit : - une communication réciproque ; - une communication orientée ; - une relation maintenue.

On peut faire une comparaison avec la force d'un lien, le nombre de personnes ayant un lien d'amitié avec quelqu'un augmente en fonction de la taille du voisinage. On remarque également que la propagation d'informations est plus rapide avec la 3 catégorie de lien.

10.9 Twitter

Twitter est un microblog qui permet de partager de petits messages (140 catactères au maximum). Il est basé sur une relation de Followers à Followees. Force du lien : - Faible si on est follower - Forte si on envoie un message ciblé Si on "suit" plus de personne, on a plus d'amis (jusqu'à un certain point, c'est une fonction logarithmique). Effectivement, cela s'explique par le fait que pour entretenir des liens forts, ceci demande un effort conditionnel. Il y a une limite physique aux nombres de liens forts que l'on peut posséder. A contrario, il ne demande pas de temps ou d'effort pour 'suivre' quelqu'un. Il y a moins de contraintes, c'est un engagement passif et c'est ce qui est proné sur Twitter.

Chapitre 11

Structure du Web

11.1 Mémoire associative - Hypertexte

Liens hypertextes : chaque élément a des liens vers et depuis d'autres éléments. Un contenu hypertexte est un contenu auquel il est fait référence dans un document.

Deux exemples de contenus hypertextes :

- **Graphe de citations**
  Précurseur du web : nœuds indépendants, liens strictement vers le passé
- **Encyclopédie**
  Les articles renvoient vers d'autres articles
  (Wikipédia)

} Réseaux d'informations

Problème de cohérence

- Web : Cohérence *"a posteriori"*
  Le web n'ayant pas d'organisation, on va devoir en trouver une : un index ou un moteur de recherche
- Wikipedia : Cohérence *"a priori"*
  Une structure existe déjà, définie par les rédacteurs, les modérateurs et les règles de structure.

Le succès du Web réside dans la possibilité de trouver une structure à ce dernier, notamment grâce à l'algorithme PageRank. Altavista, Google, Lycos, ... proposent tous leur version de moteur de recherche. C'est grâce à l'efficacité de PageRank que Google s'est imposé largement comme moteur de recherche dominant.

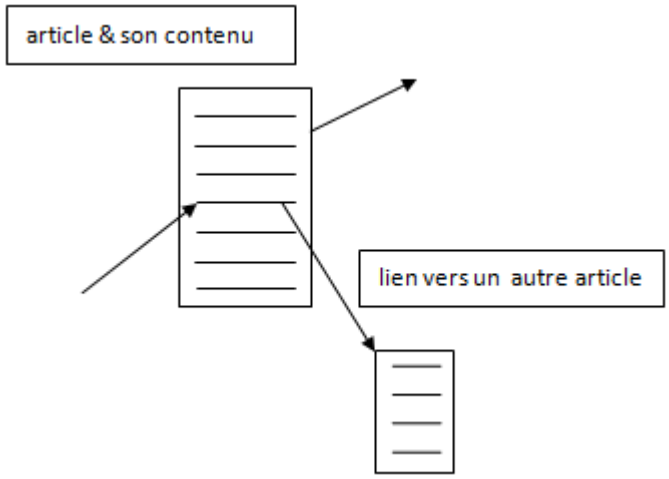


FIGURE 11.1 – Hypertexte -liens vers des articles

11.2 Le Web est un graphe orienté

— Chemin dans un graphe orienté : $A - \rightarrow B$

Séquence de nœuds qui commence avec A et termine avec B et où chaque paire consécutive correspond à un lien orienté.

— Connectivité :

Un graphe orienté est connexe s'il existe un chemin orienté entre chaque paire de nœuds.

11.3 Composant fortement connexe (CFC) *Strongly Connected Component (SCC)*

À l'intérieur d'un graphe orienté, un composant fortement connexe est :

— Un ensemble de nœuds tel qu'il existe un chemin orienté entre chaque paire.

— L'ensemble ne fait pas partie d'un plus grand environnement qui à la même propriété.

Les CFCs forment un genre de *"super noeud"*. Pour la connectivité, on peut ignorer la structure interne des CFCs.

On peut donc transformer le graphe en un graphe réduit : le CFC devient un unique nœud. Pour trouver un chemin dans le graphe original, il suffit de trouver un chemin dans le graphe réduit. Un exemple de transformation de ce type est illustré sur la figure 11.3.

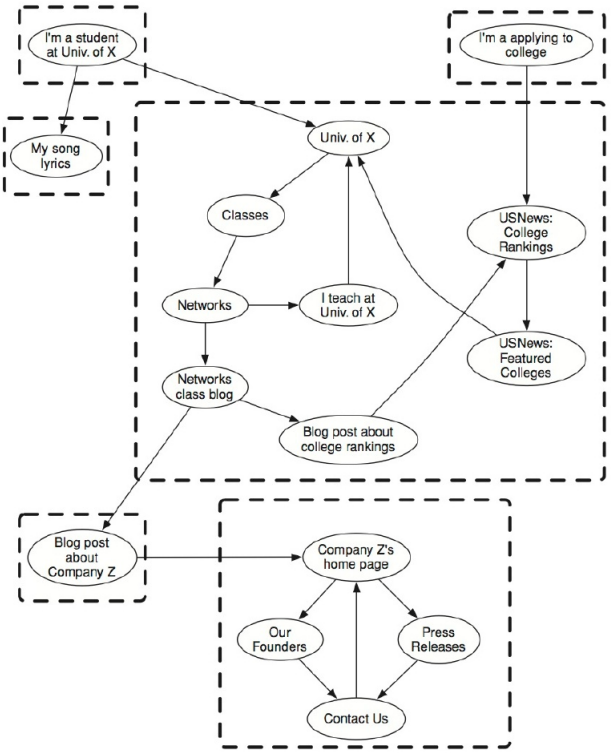


FIGURE 11.2 – Un graphe dirigé avec ses CFCs identifiés

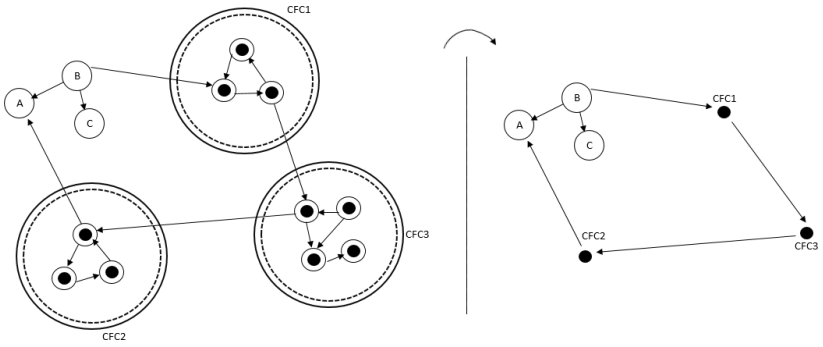


FIGURE 11.3 – Exemple de transformation du graphe

11.4 Nœud papillon (≈ *années 2000*)

Maintenant, à quoi ressemble le graphe réduit du web ? Autour des années 2000, sa structure était proche de celle d'un nœud papillon, tel que celui représenté sur la figure 11.4.

On repère trois composants principaux :

— Un composant "in", qui contient des liens hypertextes sortants.

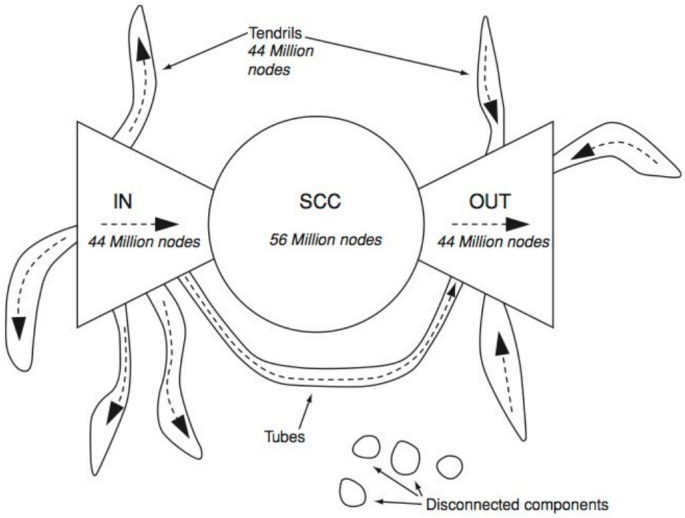


FIGURE 11.4 – Structure en nœud papillon du web

— Un composant fortement connecté principal, qui forme un "noyau".
— Un composant "out", qui contient beaucoup de liens hypertextes entrants.

11.5 Émergence du Web 2.0 ($\geq$ *années 2000*)

L'émergence du Web 2.0 se déroule entre 2000 et 2010. Il s'agit en fait d'un changement d'attitude général des certains acteurs importants du web conduisant à une structuration de la toile basée sur trois grands principes.

1. Création collaborative de contenu plutôt que des pages personnelles.
2. Services vers lesquels sont transférés les données personnelles. Au lieu de publier du contenu sur des sites personnels, les gens se tournent vers des services pour publier leur contenu (par exemple Youtube, Flicker, Github...)
3. Personnes au lieu des documents. On n'identifie plus un document en fonction de son contenu, mais en fonction de son auteur ou de la personne qui en est le sujet.

Technologies utilisées :
— Blogs (Skyblog), ensuite détrônés par les réseaux sociaux (Facebook, Twitter, MySpace, Friendster...)
— Deep Web : création de page à la demande (créer un nouveau profil, une page Wikipédia, un groupe d'intérêt...)

— Cloud : regroupement et partage de ressources "à la demande", permet plus d'élasticité (adaptation en fonction des besoins)

Un petit bout d'histoire

Quelques précurseurs :

— 1945 - Vannevar Bush : Conseiller du président des États-Unis, Roosevelt
Il a écrit un article intitulé "As We May Think" dans lequel il explique son appareil électronique relié à une bibliothèque et capable d'afficher des livres et de projeter des films appelé "Memex".

— 1934 - Paul Otlet : Documentaliste
Il avait imaginé un système où l'on pourrait faire des recherches et consulter le résultat de ces recherches sur un écran. Cette intuition d'un pre-internet est à l'origine d'une structuration des ressources des bibliothèques. Il est aussi un pionnier des microfiches, des fiches indépendantes qui stockent des informations, ayant une ressemblance non-négligeable avec les pages web d'aujourd'hui.

— 1990 - Tim Berners-Lee et Robert Cailliau - CERN : Inventeur du World Wide Web
Ils se sont inspirés des ecrits de Vannevar Bush pour inventer le WWW, avec sa structure de pages indépendantes reliées par des liens hypertextes.

Chapitre 12

Recherche dans le Web

Comment trouver une information ?

1960 : Concept de mot-clé → Limitations fortes

— Limitations dues au concept de mot
  — Synonymie : plusieurs mots pour le même concept.
  — Polysémie : un mot pour plusieurs concepts.
    Par exemple les noms propres peuvent référer à plusieurs personnes, des lieux et des organisations.

  Autrement dit, il n'y a pas de bijection entre les mots et les concepts.
— Limitations dues à l'abondance d'informations.
  — Avant : Les informations sur le web étaient rares.
  — Maintenant : Il y a beaucoup trop d'informations.

  } Plus grand problème du web

Pour résoudre le problème de surabondance d'informations, il faut trouver une façon de savoir quelle page est la meilleure. Comment faire ce choix ?

La meilleure solution est d'utiliser l'information contenue dans la structure du réseau.

Bibliographie

[Nis] Nimal Nissanke. *Introductory Logic and Sets for Computer Scientists.*

[LPP] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets : Reasoning About a Highly Connected World.*