
UCL

**Université
catholique
de Louvain**



LINGI1101

Logique et Structures Discrètes

(version préliminaire du 10 juillet 2015)

Titulaire : Peter VAN ROY

2015

Table des matières

Remerciements	7
I Logique formelle	11
1 Contexte : la méthode scientifique	12
1.1 Formalisation d'un système	12
1.2 Boucle de raisonnement	12
1.2.1 Déduction	13
1.2.2 Induction	13
1.2.3 Abduction	13
1.2.4 Conclusion	14
1.3 Exemples	14
1.3.1 Loi de Maxwell	14
1.3.2 Sac de billes	14
2 La logique propositionnelle	16
2.1 La syntaxe	17
2.2 Les tables de vérité	19
2.3 Les interprétations	20
2.4 Les modèles logiques	22
2.4.1 Conséquence logique	23
2.4.2 Équivalence logique	23
3 Preuves en logique propositionnelle	25
3.1 Preuve avec table de vérité	25

3.2	Preuve transformationnelle	26
3.3	Preuve déductive	27
3.3.1	Equivalences logiques	27
3.3.2	Règles d'inférence	28
3.3.3	Schémas de preuve	28
3.4	Exemple de preuve déductive	29
3.5	Exemples de l'utilisation des schémas	31
3.5.1	Exemple sans schéma	31
3.5.2	Exemple de preuve conditionnelle	31
3.5.3	Exemple de preuve par contradiction	32
3.6	Principe de dualité	32
3.7	Algorithme de preuve	34
3.7.1	Transformation en forme normale conjonctive	34
3.7.2	La résolution	35
3.7.3	Algorithme	37
3.7.4	Exemples	38
3.7.5	Conclusion	39
4	La logique des prédictats	40
4.1	Introduction	40
4.2	Quantificateurs	42
4.3	Syntaxe	43
4.3.1	Symboles	43
4.3.2	Règles de Grammaire	44
4.4	Sémantique	44
4.4.1	Exemple	44
4.4.2	Interprétation	45
4.5	Preuves en logique des prédictats	47
4.5.1	Preuves manuelles	47
4.5.2	Preuves automatisées	47
5	Preuves en logique des prédictats	50
5.1	Exemple	51

5.2	Règles en logique des prédicats	52
5.2.1	La substitution	52
5.2.2	Élimination de \forall	53
5.2.3	Élimination de \exists	54
5.2.4	Introduction de \exists	55
5.2.5	Introduction de \forall	55
5.3	Exemple plus conséquent	56
5.4	Note historique	57
6	Algorithme de preuve pour la logique des prédicats	58
6.1	Propriétés de l'algorithme	58
6.2	Concepts de base	59
6.2.1	Forme normale conjonctive (forme clausale)	59
6.2.2	Résolution	60
6.3	Transformation en forme prénexe	61
6.3.1	Exemple d'une transformation en forme prénexe	61
6.4	Transformation en forme Skolem	62
6.4.1	Intuition	62
6.4.2	Règle	62
6.5	Transformation en forme normale conjonctive	63
6.6	La règle de résolution	63
6.7	Algorithme	64
6.7.1	Définition de l'algorithme	65
6.7.2	Exemple d'exécution	65
6.7.3	Stratégies	66
7	Théories logiques	68
7.1	Théorie du premier ordre	69
7.1.1	Définition d'une théorie	69
7.1.2	Exemple : théorie des liens familiaux (FAM)	69
7.2	Propriétés des théories	71
7.2.1	Établir la validité	71
7.2.2	Qualité d'une théorie	72

7.3	Opérations sur les théories	72
7.3.1	Extension d'une théorie	72
7.3.2	Comparaison des théories	74
7.3.3	Corollaires	74
7.4	Théorie des ordres partiels stricts (OPS)	75
7.5	Théorie de l'égalité (EG)	76
7.5.1	Axiomes	76
7.5.2	Règles d'inférence	77
7.5.3	Théorème sur la sémantique de l'égalité	77
7.6	Théorie de l'ordre partiel (OP)	78
7.6.1	Axiomes	78
7.6.2	Exemples de modèles d'OP	79
7.7	Théorie des ensembles	80
7.7.1	Spécification formelle avec l'approche Z	81
8	Introduction à la programmation logique	85
8.1	La voie vers la programmation logique	86
8.1.1	Limitations théoriques du prouveur	86
8.1.2	Efficacité du prouveur	87
8.1.3	Construction du résultat	87
8.1.4	Bref historique	87
8.2	Introduction au langage Prolog	88
8.2.1	Exemple de programme Prolog	88
8.3	Algorithme d'exécution de Prolog	89
8.3.1	Explication de l'algorithme	89
8.3.2	Définition de l'algorithme	90
8.3.3	Gestion des choix	91
8.4	Exemples de programmes Prolog	91
8.4.1	Exemple 1 : Factorielle	92
8.4.2	Exemple 2 : Append de deux listes	94
8.4.3	Exemple 3 : Append avec plusieurs solutions	95
8.4.4	Dernières remarques	97

II Structures discrètes sur Internet	98
9 Structures discrètes sur l'Internet	99
9.1 Ressources	99
9.2 Exemple et analyse de graphes	99
9.3 Introduction	102
9.4 Nouvelle discipline	102
9.4.1 Théorie des jeux	103
10 Théorie des Graphes	106
10.1 Définitions	106
10.2 Chemins et connectivité	107
10.3 Distance entre noeuds	109
10.4 Phénomène du petit monde	109
10.5 Liens forts et faibles	110
10.6 Ponts	111
10.7 Force d'un lien dans un grand réseau	113
10.8 Notion de trou structurel et capital social	115
10.8.1 L'enclavement d'un lien (embaddness)	115
10.8.2 Notion de capital social	117
10.9 Similitude des noeuds	117
10.9.1 Principe de similitude	117
10.9.2 Nouveaux mécanismes de fermeture	119
10.9.3 Réseaux dans leurs contextes	122
10.10 La formation des liens (selon les 3 approches)	122
10.10.1 Algorithme	122
10.10.2 Modèle pour expliquer ce résultat (fermeture triadique)	123
10.11 Quantifier les rôles relatifs de sélection d'influence sociale	125
10.11.1 Comment quantifier cela ?	125
10.12 Les relations positives et négatives	127
10.12.1 Théorie de l'équilibre de structure (Équilibre structurale fort)	127
10.12.2 Caractérisation de l'équilibre structurel	128

10.12.3 Théorème d'équilibre [Frank Harary 1953]	129
10.12.4 Équilibre structurel faible	132
11 Structure du Web	134
11.1 Mémoire associative - Hypertexte	134
11.2 Le Web est un graphe orienté	135
11.3 Composant fortement connexe (CFC) <i>Strongly Connected Component (SCC)</i>	135
11.4 Web en nœud papillon (\approx années 2000)	136
11.5 Émergence du Web 2.0 (\geq années 2000)	137
12 Recherche dans le Web	140
12.1 L'analyse des liens	141
12.2 PageRank	145
Références	148

Remerciements

Je tiens à remercier les étudiants de LINGI1101 de l'année académique 2014-2015 pour avoir pris des notes pendant le 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, Siciliano Damiano-Joseph, Aghakhani Ghazaleh, Kühn Alexandre, Maas Dylan, Paulus Aloïs, De Droogh Joachim, André William, Vandepitte Cassandre, Julémont Léonard, Surquin Corentin, Ahad Ivan, Thuin Florian, Malingreau Alexandre, Vaessen Tanguy, De Cocq Aymeric, Powell Jonathan, Pignolet Aurélien, Bollen Thomas, Mondry Laurent, Leurquin Guillaume, Bertaux Jérôme, Palumbo François, Clémenti Florent, Lambin Grégoire, Lambot Sue, de Saint-Hubert Olivier, Istasse Maxime, Sayez Niels, De Grove Gil, De Maeyer Julien, Lepinois Loïc, Vander Schueren Grégory, Van Den Eeckhaut Kim, De Bels Tanguy, Cambier Rodolphe, Demesmaeker Florian, Deplasse Victor, Colard Pierre-Olivier, Sauvetelet Caroline, Lejoly Florent, Vanden Bulcke Cédric, Demaude Guillaume, Ivinza Mbe Scott, Georges Benjamin, Dizier Romain, Kerger Zacharie, Larigaldie Nathanaël, Russello Helena, Visschers Marie, Grynczel Wojciech, Hauet Alexandre, Jacquet Charles, Lemaire Jérôme, Degryse Baptiste, Moubarak Joey, Wenders Audrey, Vrielynck Nicolas, Henneton Romain, Gusbin Quentin, Gerniers Alexander, Ndizera Eddy, El Jilali Solaiman, Dhillon Sundeep, Rens Maxime, Hardy Maxime, Haven David, Francotte Florian, de Potter d'Indoye Aurian, Dechamps Anthony, Ninane Charles, Mokaddem Sami, Deconinck Guillaume, Dubois Robin, Pierret Alexis, De Mol Maxime, De Ryck Aurélien, Marinx Olivier, Marinx Denis, Candaele Simon, Dagnely Vincent, Dethise Arnaud, Bellenger Jordan, Schmitz Loic, Hofs Sylvian.

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, 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.

En année académique 2014-2015, 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.

Le cours se base principalement sur trois ouvrages :

- Introductory Logic and Sets for Computer Scientists, by *Nimal Nisanke*.
- Mathématiques discrètes : bases logiques de l'informatique, by *Philippe Delsarte and Axel van Lamsweerde*.
- Networks, Crowds, and Markets : Reasoning About a Highly Connected World, by *David Easley and Jon Kleinberg*.¹

Néanmoins, ce syllabus peut être lu indépendamment de ces ouvrages. Le syllabus complémente ces ouvrages par d'autres informations afin que le tout donne une vue équilibrée du domaine.

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 prédictats

C'est une logique beaucoup plus expressive et la plupart des travaux mathématiques peuvent être écrits dans ce langage². Elle est aussi définie comme la logique du premier ordre.³ En logique des prédictats, les éléments de base du langage ne sont plus des propositions, mais des prédictats.

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 la 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

-
1. Quelques chapitres.
 2. Elle est un effet un parfait compromis entre expressivité et efficacité.
 3. Il existe d'autres formes de logiques plus expressives, mais plus difficiles à utiliser.
Exemple : la logique du deuxième ordre.

y a des règles d'inférences à utiliser pour prendre une phrase en logique et en 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 prédictats. 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

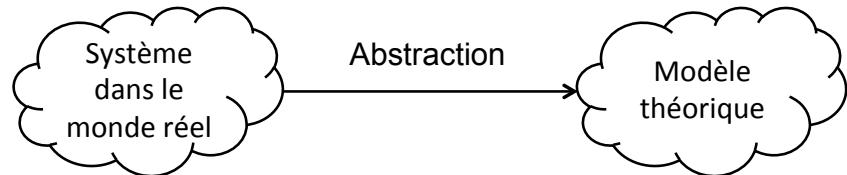
Logique formelle

Chapitre 1

Contexte : la méthode scientifique

1.1 Formalisation d'un système

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

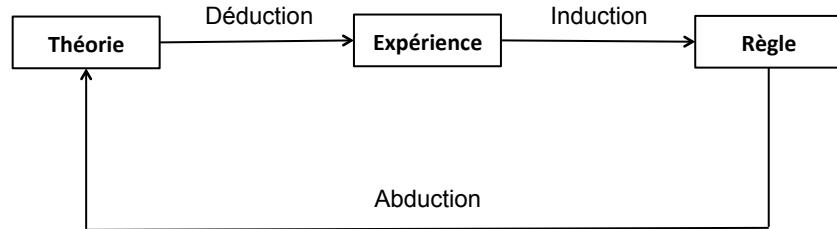


Afin de formaliser un système dans le monde réel, nous devons faire une abstraction vers un modèle théorique. Ce modèle théorique, aussi appelé théorie, est un ensemble de phrases logiques dont il est possible de tirer des prédictions en utilisant le raisonnement déductif. Il n'est intéressant que s'il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

Un exemple de cette formalisation pourrait être les équations de Maxwell qui sont le modèle théorique correspondant pour l'électromagnétisme.

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 une théorie. Avec ces raisonnements, on déduit le résultat qu'une expérience donnerait selon la théorie. Par exemple, en utilisant les équations de Maxwell on peut déduire le trajet d'un objet avec une charge électrique dans un champ électromagnétique.

1.2.2 Induction

L'induction est le fait de trouver une règle générale à partir des expériences répétées. 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 complets. 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 n'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. S'il y a une incohérence entre la règle générale et la théorie qui ne rentre pas dans la marge d'erreur expérimentale, on suppose qu'il y a une erreur dans la théorie. Il faut alors corriger la théorie existante ou en inventer/deviner une nouvelle. Ce type de raisonnement s'appelle l'abduction : trouver une *explication* (= la théorie corrigée) pour une règle ou un fait. On applique l'abduction couramment dans la vie de tous les jours ; par exemple, lorsqu'un élève entre trempé dans la classe, nous supposons qu'il pleut dehors. La pluie est une explication possible pour l'état de l'élève.

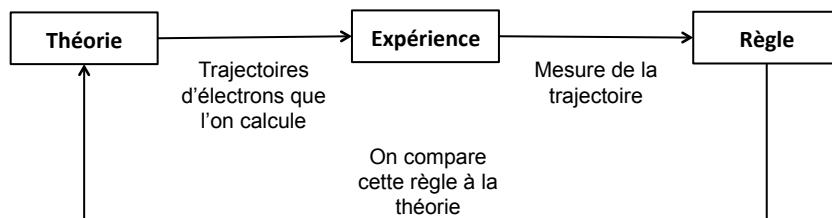
1.2.4 Conclusion

Sur ces trois formes de raisonnement, seule la déduction est un raisonnement sûr. Les deux autres, l'induction et l'abduction, peuvent donner des erreurs. Malgré cela, les trois formes sont tout aussi importantes. Par exemple, il faut les trois pour expliquer comment marche la méthode scientifique. Dans l'état actuel de la science du raisonnement, nous comprenons beaucoup mieux la déduction que l'induction et l'abduction. Toute la logique mathématique est un approfondissement de la science de la déduction. 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 calculons 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 cette règle à la théorie, et nous la corrigons si besoin. La création d'une théorie qui explique la règle trouvée est une abduction.

1.3.2 Sac de billes

Afin d'illustrer les 3 formes de raisonnements 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 : $sac(a), sac(b), \dots$
-
3. Résultat : $blanc(a), blanc(b), \dots$

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 : $sac(a), sac(b), \dots$
 2. Résultat : $blanc(a), blanc(b), \dots$
-
3. Règle : $\forall x, sac(x) \Rightarrow blanc(x)$

Si toutes les billes que 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 : $blanc(a), blanc(b), \dots$
-
3. Cas : $sac(a), sac(b), \dots$

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. L'explication pour la couleur des billes trouvées est 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 connexions logiques entre des propositions. Par exemple, prenons les expressions suivantes :

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

Dans ces expressions, nous pouvons définir des propositions premières :

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 sens de ces propositions en langue naturelle n'a aucune importance dans la logique propositionnelle. 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. $E = \text{« il fait jour »};$
6. $F = \text{« il fait nuit »}.$

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 réécrits 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 *syntaxe* (définie par 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 syntaxe

La logique propositionnelle est un *langage formel*. Ce langage 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 (« et ») : \wedge
 - Disjonction (« ou ») : \vee
 - Négation : \neg
 - Implication : \Rightarrow
 - Équivalence : \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 :

```
<identificateur>   ::=   A | B | C | D | ...
<proposition>    ::=   true
                      |
                      | false
                      |
                      | <identificateur>
                      |
                      | (<proposition>)
                      |
                      |  $\neg$ <proposition>
                      |
                      | <proposition>  $\wedge$  <proposition>
                      |
                      | <proposition>  $\vee$  <proposition>
                      |
                      | <proposition>  $\Rightarrow$  <proposition>
                      |
                      | <proposition>  $\Leftrightarrow$  <proposition>
```

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 doivent *toujours* être représentées par 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.

Métalangage Notez que notre discours (en français et en notation mathématique) à propos des phrases propositionnelles n'est pas une phrase propositionnelle. La grammaire précédente, la description de l'alphabet, et cette explication en français parlent de propositions logiques sans en être, et font donc partie de ce qui est appelé le *métalangage*. Un métalangage est un deuxième langage utilisé pour parler d'un premier langage. Dans notre discours, le premier langage est la logique propositionnelle, et le deuxième langage est le français augmenté avec des notations mathématiques.

Le concept de métalangage est important pour distinguer le raisonnement formel (en utilisant les opérations logiques définies formellement) et le raisonnement informel (typiquement en langage naturel augmenté par des notations mathématiques). Le raisonnement informel reste très important, même si le but ultime est de faire le plus possible en raisonnement formel, parce qu'il est plus facile d'éviter des erreurs de raisonnement dans un raisonnement formel.

2.2 Les tables de vérité

La grammaire définie dans la section précédente permet d'écrire les propositions en logique propositionnelle, mais elle ne leur donne pas un sens, c'est-à-dire de définir quand une proposition est vraie ou fausse. Plus précisément, le sens d'une proposition s'appelle la sémantique de la proposition. Pour savoir si une proposition est vraie ou fausse, il faut commencer par choisir pour chacune de ses propositions premières si elle est vraie ou fausse. Ensuite on peut déterminer si la proposition est vraie ou fausse. Il y a deux approches principales pour faire cela : les *tables de vérité* et les *interprétations*. Dans cette section nous expliquerons les tables de vérité. Dans la section suivante nous expliquerons les interprétations.

Rappelez-vous que la signification des propositions premières n'a aucune importance, le choix de sa véracité 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 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 $\text{val}_I : E_P \rightarrow \{\text{true}, \text{false}\}$ qui permet de savoir si ces propositions premières sont vraies ou fausses.¹ Par exemple, on pourrait écrire ceci :

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

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

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

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

Étant donné la fonction val_I , il est possible de définir la fonction $\text{VAL}_I : P \rightarrow \{\text{true}, \text{false}\}$, 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 :

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

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

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

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

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

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

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 propositions du formalisme que nous avons défini, en commençant par identifier les propositions premières :

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

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 : $\text{val}_I(B) = \text{true} ;$

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

Nous pouvons alors effectuer le développement suivant :

$$\text{VAL}_I((B \wedge M) \implies P) = \begin{cases} \text{false} & \text{si } \text{VAL}_I(P) = \text{false} \text{ alors que } \text{VAL}_I(B \wedge M) = \text{true} \\ \text{true} & \text{sinon} \end{cases}$$

$$\text{VAL}_I(B \wedge M) = \begin{cases} \text{true} & \text{si } \text{VAL}_I(B) = \text{true} \text{ et } \text{VAL}_I(M) = \text{true} \\ \text{false} & \text{sinon} \end{cases}$$

Dans notre cas :

$$\text{val}_I(B \wedge M) = \text{true} \wedge \text{true} = \text{true}$$

Donc :

$$\text{val}_I((B \wedge M) \implies P) = \text{true} \implies \text{false} = \text{false}$$

Nous pouvons donc en conclure que, dans cette interprétation, la personne ayant fait cette affirmation a menti. Notons néanmoins que notre homme n'aurait pas menti 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

Dans le premier chapitre, nous avons parlés de modèles théoriques (ou théories) d'un système dans le monde réel. Dans le contexte de la méthode scientifique, nous avons fait de la déduction à partir de ces modèles théoriques. Maintenant que nous avons introduit notre première logique et sa sémantique, nous pouvons rendre ces notions plus concrètes.

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

Dans l'exemple de la section précédente, l'interprétation choisie n'est donc pas un modèle de la proposition analysée, celle-ci n'étant pas validée. Par contre, l'interprétation telle que $\text{val}_I(B) = \text{true}$, $\text{val}_I(M) = \text{true}$ et $\text{val}_I(P) = \text{true}$ est bien un modèle de la proposition utilisée comme exemple. Remarquez aussi que cela ne change rien au fait que l'interprétation choisie soit le modèle d'autres propositions que celle étudiée ou non (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 $\models p$.

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 $\not\models p$.

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

$$p \models q \quad q \text{ est valide dans tous les modèles de } p$$

$$\begin{aligned} &\text{alors} \\ &\models (p \Rightarrow q) \quad p \Rightarrow q \text{ est une tautologie.} \end{aligned}$$

$$\begin{aligned} &\text{On peut donc écrire} \\ &p \Rightarrow q \quad p \text{ est conséquence logique de } q. \end{aligned}$$

Cependant, la conséquence logique (\Rightarrow) n'est pas une proposition logique, mais fait partie du métalangage (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

$$\begin{array}{ll} p \models q & q \text{ est valide dans tous les modèles de } p \\ q \models p & p \text{ est valide dans tous les modèles de } q \end{array}$$

et donc

$$\begin{array}{ll} \models (p \Rightarrow q) & p \Rightarrow q \text{ est une tautologie et} \\ \models (q \Rightarrow p) & q \Rightarrow p \text{ est une tautologie.} \end{array}$$

On peut donc écrire

$$p \Leftrightarrow q \quad p \text{ sont logiquement équivalents } q.$$

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, \dots) et propositions premières (P, Q, S, \dots) (cf. syntaxe d'une proposition) :

$$\begin{array}{ll} p \Rightarrow q & \text{n'est pas une proposition} \\ \text{mais} & \\ P \wedge Q \Rightarrow R \wedge \neg S & \text{en est bien une.} \end{array}$$

Chapitre 3

Preuves en logique propositionnelle

Une preuve est un raisonnement déductif qui démontre si une proposition est vraie ou fausse. On distingue des preuves informelles et des preuves formelles. Une preuve informelle est un raisonnement en langage naturel, parfois augmenté avec des notations mathématiques. Une preuve formelle est un objet mathématique qui formalise le raisonnement déductif. Un des buts de la logique mathématique est de prouver le plus possible des résultats mathématiques avec des preuves formelles.

Au 20ème siècle les mathématiciens sont arrivés à prouver la plupart des mathématiques classiques (telles qu'utilisées par des ingénieurs) avec des preuves formelles. Un des résultats les plus célèbres est la preuve formelle du théorème des quatre couleurs, fait par Georges Gonthier et Benjamin Werner avec l'assistant de preuve Coq (un logiciel qui automatise la plupart des manipulations formelles nécessaires). Ce théorème dit que toute carte découpée en régions connexes peut être colorée avec seulement quatre couleurs, de sorte que deux régions adjacentes ont toujours des couleurs distinctes.

Dans ce chapitre nous allons définir des preuves formelles pour la logique propositionnelle. Nous présenterons trois approches :

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

3.1 Preuve avec table de vérité

La preuve formelle la plus simple est une table de vérité. Prouvons que $\neg(P \wedge Q) \Leftrightarrow (\neg P \vee \neg Q)$ est vrai :

P	Q	$\neg P$	$\neg Q$	$(\neg P \vee \neg Q)$	$P \wedge Q$	$(\neg P \vee \neg Q)$
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 peut constater que le vecteur de vérité de $\neg(P \wedge Q)$ est équivalent à celui de $(\neg P \vee \neg Q)$. La preuve a donc vérifié la véracité de la proposition. Notez qu'une table de vérité est un objet mathématique en métalangage parce qu'elle n'est pas une proposition.

L'inconvénient de cette méthode de preuve est qu'elle devient rapidement très lourde quand le nombre de propositions premières augmente. Il faut en effet 2^n lignes dans la table pour n propositions.

3.2 Preuve transformationnelle

Une preuve transformationnelle est une séquence de transformations $p_1 \Leftrightarrow p_2 \Leftrightarrow \dots \Leftrightarrow p_n$, dans laquelle on a toujours $p_i \Leftrightarrow p_{i+1}$ (équivalence logique entre éléments adjacents dans la séquence). Une preuve transformationnelle est aussi un objet mathématique en métalangage. Pour faciliter la création d'une preuve transformationnelle, on utilise des "Lois", c'est-à-dire des équivalences connues.

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

On ajoute deux règles supplémentaires : la transitivité et la substitution.

Transitivité de l'équivalence

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

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 $r(p) \Leftrightarrow r(q)$.

On peut remplacer p par q car elles sont équivalentes.

Exemple

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

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

Le problème de cette méthode de preuve est qu'elle requiert de l'intuition, de la créativité. Elle n'est donc pas forcément plus efficace que les tables de vérité, surtout si "l'astuce" est difficile à trouver.

3.3 Preuve déductive

Une preuve déductive est un objet mathématique qui formalise une séquence de pas de raisonnement simples. Chaque pas doit être justifié avec le nom de la règle ou la loi qui est utilisée. Les pas utilisent trois techniques de raisonnement différentes : les équivalences logiques, les règles d'inférence et les schémas de preuve. Avec ces techniques, une preuve déductive est beaucoup plus expressive qu'une preuve transformationnelle.

3.3.1 Equivalences logiques

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

À ces équivalences nous ajoutons aussi l'idempotence, la commutativité, l'associativité et la distributivité de \wedge .

3.3.2 Règles d'inférence

À la différence de la preuve transformationnelle, les règles d'inférences ont une direction : elles commencent par les prémisses et se terminent par la conclusion. Pour chaque règle, si les prémisses sont vraies, alors la conclusion est vraie. Nous utilisons un raisonnement informel pour justifier chaque règle.

Conjonction : $\frac{\begin{array}{c} p \text{ prémissse} \\ q \text{ prémissse} \end{array}}{p \wedge q} \text{ Conclusion}$	Simplification : $\frac{p \wedge q}{p}$	
Addition : $\frac{p}{p \vee q}$	Contradiction : $\frac{\begin{array}{c} p \\ \neg p \end{array}}{q}$	
Double Négation : $\frac{\neg\neg p}{p}$	Transitivité de l'équivalence : $\frac{\begin{array}{c} p \Leftrightarrow q \\ q \Leftrightarrow r \end{array}}{p \Leftrightarrow r}$	
Modus Ponens : $\frac{\begin{array}{c} p \Rightarrow q \\ p \end{array}}{q}$	Modus Tollens : $\frac{\begin{array}{c} p \Rightarrow q \\ \neg q \end{array}}{\neg p}$	
Loi d'équivalence : $\frac{p \Leftrightarrow q}{q \Leftrightarrow p}$		

3.3.3 Schémas de preuve

En plus des équivalences logiques et des règles d'inférence, nous ajoutons deux schémas de preuve qui formalisent des techniques de raisonnement plus abstraites : le théorème de déduction et la démonstration par l'absurde. Ces schémas donnent à l'approche de preuve déductive une grande expressivité, beaucoup plus qu'une preuve transformationnelle.

Théorème de déduction (preuve conditionnelle)

Pour prouver la proposition $s \Rightarrow t$, on suppose s vrai. La proposition s s'ajoute donc aux prémisses utilisées dans la preuve. Ensuite, on fait une preuve de t : on peut construire une preuve (objet mathématique) de t en commençant de s . On note ce théorème $s \vdash t$. On écrit ce schéma un peu comme une règle d'inférence :

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

On déduit t et donc on sait que l'hypothèse $s \Rightarrow t$ est vraie et on l'évacue.

Remarque : Il ne faut pas confondre les deux notations $p \models t$ et $p \vdash t$.

- $p \models t$ est une notion de vérité (tout modèle de p est un modèle de t), et donc de sémantique ;
- $p \vdash t$ est une notion syntaxique (en commençant de p on peut construire une preuve de t), car une preuve est une séquence de manipulations syntaxiques.

Démonstration par l'absurde (preuve par contradiction)

On suppose que les prémisses p, \dots, q n'ont pas de problème, c'est-à-dire qu'on ne peut pas prouver une contradiction à partir de ces propositions. Ensuite, on ajoute r aux prémisses. S'il est possible de prouver s et aussi de prouver $\neg s$, cela signifie qu'il y a une erreur dans les prémisses. On suppose que c'est l'ajout r qui est fautif. On justifie qu'il n'y a aucune contradiction dans p, \dots, q car on part du principe qu'il existe un modèle de p, \dots, q . On écrit ce schéma ainsi :

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

3.4 Exemple de preuve déductive

Nous donnons un premier exemple de preuve déductive. 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 établir une théorie, c'est-à-dire, un ensemble de propositions, dont on espère qu'elle aura un modèle.

Théorie

1. $A \implies B$
2. $C \implies D$
3. $B \vee D \implies E$
4. $\neg E$

À prouver $\neg A \wedge \neg C$

Preuve Voici la preuve déductive. Nous la mettons dans un cadre pour souligner qu'elle est un objet mathématique. Chaque ligne est où une prémissé, où un pas de raisonnement (une équivalence ou une règle d'inférence). Pour chaque ligne il faut donner le nom de la règle qui est appliquée, cela s'appelle la *justification* et c'est une partie importante de la preuve. Les deux schémas se présentent avec des indentations ; la partie indentée d'une preuve contient une prémissé en plus (*s* pour la preuve conditionnelle, *r* pour la preuve indirecte).

1. $A \Rightarrow B$	prémissé
2. $C \Rightarrow D$	prémissé
3. $B \vee D \Rightarrow E$	prémissé
4. $\neg E$	prémissé
5. A	hypothèse
6. B	modus ponens (1)
7. $B \vee D$	addition (6)
8. E	modus ponens (7)
9. $\neg A$	preuve indirecte
10. C	hypothèse
11. D	modus ponens (2)
12. $D \vee B$	addition (11)
13. $B \vee D$	commutativité (12)
14. E	modus ponens (9)
15. $\neg C$	preuve indirecte
16. $\neg A \wedge \neg C$	conjonction (9,15)

Les quatre premières lignes introduisent les prémisses (les propositions de la théorie). La ligne 5 commence une première preuve indirecte : on fait l'hypothèse A et ensuite on déduit E (sur la ligne 8). C'est une contradiction avec la prémissé $\neg E$ et donc on vient de prouver $\neg A$ (sur la ligne 9). La ligne 10 commence une deuxième preuve indirecte : on fait l'hypothèse C et

ensuite on déduit E (sur la ligne 14). De nouveau, c'est une contradiction (avec la prémissse $\neg E$) et donc on vient de prouver $\neg C$ (sur la ligne 15). Les justifications pour les lignes 9 et 15 sont *preuve indirecte*.

Avec cette preuve déductive, nous avons pu prouver que tu ne manges pas bien et que tu ne pratiques pas d'activité physique régulière.

3.5 Exemples de l'utilisation des schémas

Pour illustrer l'utilisation des deux schémas, la preuve conditionnelle et la preuve par contradiction, nous allons prouver la même conclusion en trois manières, avec chaque schéma et sans schéma.

- Prémissse : $(p \wedge q) \vee r$
- Conclusion : $\neg p \Rightarrow r$

3.5.1 Exemple sans schéma

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

3.5.2 Exemple de preuve conditionnelle

1. $(p \wedge q) \vee r$	<i>Prémissse</i>
2. $\neg \neg(p \wedge q) \vee r$	<i>Double négation en 1</i>
3. $\neg(\neg p \vee \neg q) \vee r$	<i>Loi De Morgan en 2</i>
4. $\neg p \vee \neg q \Rightarrow r$	<i>Implication en 3</i>
5. $\neg p$	<i>Hypothèse</i>
6. $\neg p \vee \neg q$	<i>Addition sur 5</i>
7. r	<i>Modus Ponens sur 4 et 6</i>
8. $\neg p \Rightarrow r$	<i>Evacuation de l'hypothèse</i>

3.5.3 Exemple de preuve par contradiction

1. $(p \wedge q) \vee r$	<i>Prémisse</i>
2. $(p \vee r) \wedge (q \vee r)$	<i>Distributivité sur 1</i>
3. $(p \vee r)$	<i>Simplification en 2</i>
4. $\neg(\neg p \Rightarrow r)$	<i>Hypothèse</i>
5. $\neg(\neg\neg p \vee r)$	<i>Implication en 4</i>
6. $\neg(p \vee r)$	<i>Négation en 5</i>
7. $\neg\neg(\neg p \Rightarrow r)$	<i>Preuve par contradiction</i>
8. $\neg p \Rightarrow r$	<i>Négation en 7</i>

3.6 Principe de dualité

Le principe de dualité affirme que l'on peut transformer une propriété vraie en une autre propriété vraie en remplaçant systématiquement tout symbole par un autre selon un schéma de correspondances.

Dans les formules sans \rightarrow :

Les correspondances sont :

$$\begin{aligned}\wedge &\leftrightarrow \vee \\ \text{true} &\leftrightarrow \text{false}\end{aligned}$$

Par exemple, si on prend la tautologie suivante :

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

en remplaçant chaque symbole par le symbole correspondant on obtient une autre tautologie :

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

Dans une formule quelconque :

Les correspondances sont :

$$\begin{aligned}\wedge &\leftrightarrow \vee \\ \text{true} &\leftrightarrow \text{false} \\ p &\leftrightarrow \neg p\end{aligned}$$

Par exemple, si on prend la définition suivante :

$$\{p_1, \dots, p_n\} \models q \quad ssi \quad \models (p_1 \wedge \dots \wedge p_n \wedge \neg q) \leftrightarrow \text{false}$$

en remplaçant chaque symbole par le symbole correspondant on obtient une propriété vraie :

$$\{p_1, \dots, p_n\} \models q \quad \text{ssi} \quad \models (\neg p_1 \vee \dots \vee \neg p_n \vee q) \leftrightarrow \text{true}$$

3.7 Algorithme de preuve

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

3.7.1 Transformation en forme normale conjonctive

Toute formule peut être transformée en une formule équivalente, la forme normale conjonctive, qui a toujours la même forme. La forme normale conjonctive facilite la manipulation des formules nécessaire par l'algorithme de preuve.

Forme normale conjonctive

Il y a deux formes normales qui sont souvent utilisées : la forme normale conjonctive (FNC) et la forme normale disjonctive (FND). Pour l'algorithme de preuve, nous allons utiliser la FNC, mais comme la FND est parfois importante, nous les définissons toutes les deux. Dans la FNC, la formule est écrite comme une conjonction de disjonctions. Dans la FND, la formule est écrite comme une disjonction de conjonctions. À l'intérieur de chaque forme normale on trouve des propositions premières ou des négations des propositions premières. Voici un exemple de chaque forme normale :

- FNC : $(P \vee \neg Q) \wedge (Q \vee A) \wedge (\neg S \vee R)$
- FND : $(P \wedge \neg Q) \vee (Q \wedge A) \vee (\neg S \wedge R)$

Pour faciliter la discussion autour des formes normales, nous introduisons une terminologie :

- Un *littéral*, écrit L , est où une proposition première où la négation d'une proposition première. Pour une proposition première P on peut faire deux littéraux, P et $\neg P$.
- Une *clause*, écrite C , est (pour la forme normale conjonctive) une disjonction de littéraux. On écrit $\vee L_i$ ou $(L_1 \vee L_2 \vee L_3 \vee \dots \vee L_i)$.

L'algorithme de normalisation

L'algorithme de normalisation se fait en quatre phases :

1. Eliminer les \rightarrow et \leftrightarrow en les remplaçant par des formules équivalentes.
Par exemple, $p \rightarrow q$ sera remplacée par $\neg p \vee q$.
2. Déplacer les négations vers l'intérieur (jusqu'à dans les propositions premières) en utilisant les formules de De Morgan.
3. Déplacer les disjonctions (\vee) vers l'intérieur en utilisant les lois distributives.
4. Simplifier en éliminant les formes $(P \vee \neg P)$ dans chaque disjonction.

Exemple de normalisation

$$\begin{aligned}
& (P \rightarrow (Q \rightarrow R)) \rightarrow ((P \wedge S) \rightarrow R) \\
& \neg(\neg P \vee (\neg Q \vee R)) \vee (\neg(P \wedge S) \vee R) \\
& (\neg\neg P \wedge \neg(\neg Q \vee R)) \vee ((\neg P \vee \neg S) \vee R) \\
& (P \wedge (Q \wedge \neg R)) \vee (\neg P \vee \neg S \vee R) \\
& (P \vee \neg P \vee \neg S \vee R) \wedge (Q \vee \neg P \vee \neg S \vee R) \wedge (\neg R \vee \neg P \vee \neg S \vee R) \\
& (Q \vee \neg P \vee \neg S \vee R)
\end{aligned}$$

3.7.2 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

- | | |
|---|-------------------|
| 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 | B : notre théorie |
|---|-------------------|

On va utiliser B + modus ponens + résolution.

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

$$\begin{array}{l}
 p_1 \vee q \\
 p_2 \vee \neg q \\
 \hline
 p_1 \vee p_2
 \end{array}$$

Cette règle représente la base de l'algorithme de résolution. On peut la vérifier en utilisant le métalangage. Nous allons voir que cette règle sera

utilisée aussi 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.

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

C_1, C_2 = deux disjonctions

On doit prouver : $\{C_1, \dots, C_n\} \models r$ 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.

3.7.3 Algorithme

Prenons des axiomes C_i qui sont normalisés :

$$C_i = \bigvee_i L_i$$

$$L_i = P \text{ ou } \neg P$$

et un candidat théorème C à démontrer. Nous voulons donc démontrer :

$$\{C_1, \dots, C_n\} \vdash C$$

C'est-à-dire, qu'il existe une preuve avec les règles d'inférence $\{C_1, \dots, C_n\}$ tel qu'on obtient C . Nous savons que $\{C_1, \dots, C_n\} \models C$ ssi $\{C_1, \dots, C_n, \neg C\} \models \text{false}$. Il suffit donc de démontrer que $S = \{C_1, \dots, C_n, \neg C\}$ est inconsistant. Pour arriver à cela, nous allons combiner des éléments de S avec la résolution,

jusqu'à que l'on arrive sur le résultat false ou qu'il n'y a plus de possibilités d'utiliser la résolution. Dans le premier cas, nous avons prouvé C . Dans le deuxième cas, il n'existe pas de preuve de C .

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  $r := C_1 - \{P\} \vee C_2 - \{\neg P\}$ 
    — calculer  $S := S \cup \{r\}$ 
end
if false  $\in S$  then
    | C est prouvé
else
    | C n'est pas prouvé
end

```

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

3.7.4 Exemples

Exemple 1

$$\begin{aligned}
 C_1 &: P \vee Q \\
 C_2 &: P \vee R \\
 C_3 &: \neg Q \vee \neg R \\
 C &: P
 \end{aligned}
 \quad \{C_1, C_2, C_3, \neg C\}$$

Quelques pas de résolution :

$$\begin{aligned}
 C_1 + \neg C &\rightarrow Q \quad (C_5) \\
 C_2 + \neg C &\rightarrow R \quad (C_6) \\
 C_3 + C_5 &\rightarrow \neg R \quad (C_7) \\
 C_6 + C_7 &\rightarrow \underline{\text{false}} \quad (\in S \text{ donc } C \text{ est prouvé})
 \end{aligned}$$

Exemple 2

$$\begin{aligned}
 p_1 &: \text{Mal de tête} \wedge \text{Fièvre} \Rightarrow \text{Grippe} \\
 p_2 &: \text{Gorge blanche} \wedge \text{Fièvre} \Rightarrow \text{Angine} \\
 p_3 &: \text{Mal de tête} \\
 p_4 &: \text{Fièvre}
 \end{aligned}$$

Algorithme

— Normalisation en forme normale

— Pseudocode avec résolution

Question : Grippe ?

3.7.5 Conclusion

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

Pour toute théorie $B = \{c_1, \dots, c_n\}$ 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 très puissant mais n'est pas toujours très efficace. Par contre, quoi qu'il arrive, on peut au moins être sûr qu'il s'arrêtera toujours à un moment.

La logique des propositions n'est malheureusement pas très expressive. Elle ne permet pas de relations entre les propositions. Nous allons essayer d'appliquer la même démarche mais avec une logique plus puissante : la logique des prédictats. Il n'est par contre pas possible d'arriver à un algorithme aussi puissant avec la logique des prédictats, la logique étant trop forte.

Chapitre 4

La logique des prédictats

4.1 Introduction

Nous allons maintenant étudier une logique beaucoup plus expressive que la logique propositionnelle, la logique des prédictats, 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 prédictats que nous allons étudier.

Logique Propositionnelle	Logique des prédictats
Propositions premières P, Q, R \hookrightarrow Pas de Relations	Prédicats $P(x,y)$ Quantificateurs : $\exists x, \forall y$ \hookrightarrow Relation

On note $P(x,y)$ dans la logique des prédictats avec x,y , les arguments du prédicat P qui sont des variables. Dans la logique propositionnelle, chaque proposition est isolée/indépendante alors que dans les prédictats on peut lier plusieurs prédictats ensemble.

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

En logique propositionnelle il n'y a aucune relation entre P et Q , alors qu'en logique des prédictats on peut lier Socrate et Platon avec le prédicat *Philosophe* qui prend en argument le nom du philosophe (Socrate ou Platon dans ce cas). $Phil(Socrate)$ est donc vrai. On peut donc dire grâce aux

1. Il existe des logiques d'ordres supérieurs, mais elles ne feront pas l'objet de ce cours.

prédicts que Socrate et Platon sont "la même chose", des philosophes.

Un autre exemple de prédictat :

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

$\hookrightarrow \dots$ cette formulation permet de résumer un très grand nombre de faits.
L'ensemble des arguments α peut être infini

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

Dire la même chose en logique propositionnelle serait beaucoup plus compliqué :

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

On va donc noter en logique propositionnelle

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

On doit tout énumérer car il n'y a aucune relation entre les différentes propositions. S'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 peut être utile. Il existe des outils informatiques qui utilisent la logique propositionnelle. On peut prendre l'exemple de "SAT solver" à qui on donne des équations booléennes très compliquées et qui va trouver les valeurs des propositions primitives qui rendent vraie cette proposition. La logique propositionnelle est utile, mais si l'on veut faire du raisonnement sur plus que "vrai" et "faux" avec des relations entre des propositions, la logique propositionnelle ne marche pas. Si on veut faire un logiciel qui montre une certaine intelligence, il faut utiliser la logique des prédictats.

Autre exemple :

Exemple	Logique propositionnelle	Logique des prédictats
Tout adulte peut voter	P	$\forall x \text{ adulte}(x) \Rightarrow \text{voter}(x)$
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 prédictats c'est beaucoup plus simple. Le John

en ligne 3 et en ligne 4 correspond à la même personne, ou de manière plus général à la même variable. Ceci montre donc bien l'utilité de 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*. Les quantificateurs permettent de résumer un grand nombre de formules en 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. Par exemple, les deux formules suivantes :

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

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

Ces deux formules peuvent paraître équivalentes, mais en réalité elles ont un sens tout à fait différent. En effet, dans le deuxième cas on remarque que le quantificateur $\forall x$ ne porte pas sur la dernière variable x qui est l'argument du prédicat $\text{aime}(x,y)$. 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 incorrecte, conflit des noms de variables

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

$$\forall x \exists z P(x) \wedge Q(z) \text{ renommage (2)}$$

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

Une comparaison peut être faite entre un code de programme et une formule. Prenons un code tout à fait banal comprenant des variables différentes avec des portées différentes qui ont le même identificateur ainsi 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 }
```

$$\forall x \forall y p(\textcolor{violet}{x}) \wedge ((\exists x q(\textcolor{blue}{x}, \textcolor{red}{y})) \vee r(\textcolor{violet}{x}, \textcolor{red}{y}))$$

Cet exemple illustre la hiérarchie et la portée des variables et des quantificateurs.

En analysant la formule morceau par morceau :

- " $\forall x \forall y$ $p(\textcolor{violet}{x}) \wedge$ " correspond aux lignes {2,3,4} du code
- " $\exists x$ $q(\textcolor{blue}{x}, \textcolor{red}{y}) \vee$ " correspond aux lignes {7,8,9}
- " $r(\textcolor{violet}{x}, \textcolor{red}{y})$ " correspond à la ligne {11}

Cet exemple montre bien la correspondance entre le concept de portée dans le monde de la programmation et celui de la logique des prédictats.

4.3 Syntaxe

4.3.1 Symboles

Voici les différents symboles utilisés dans une formule de la logique des prédictats. Nous appelons *arité* d'un prédictat ou d'une fonction son nombre d'arguments (fonction unaire, prédictat binaire, etc.).

Symboles logiques	quantificateurs connecteurs logiques parenthèses variables	$\forall \exists$ $\wedge \vee \neg \Rightarrow \Leftrightarrow$ () x, y, z true, false
Symboles non logiques	symboles de prédicat symboles de fonction symboles de constante	$P Q R$ (avec arité ≥ 0) $f g h$ (avec arité ≥ 0) $a b c$ (si arité = 0)

4.3.2 Règles de grammaire

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

4.4 Sémantique

Dans la logique des prédictats, nous gardons les notions de modèle et d'interprétation déjà définies dans la logique propositionnelle. Même si la logique des prédictats est beaucoup plus puissante, sa sémantique reste similaire à la logique propositionnelle. Comme pour la logique propositionnelle, une interprétation peut avoir une valeur (true, false).

4.4.1 Exemple

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. Une interprétation possible de cette formule est la suivante :

- $P : \text{val}_I(P) = \geq$ (considéré comme un vrai prédicat)
- $a : \text{val}_I(a) = \sqrt{2}$
- $b : \text{val}_I(b) = \pi$

- $f : \text{val}_I(f) = f_i \quad f_i = \mathfrak{R} \rightarrow \mathfrak{R} : d \rightarrow \frac{d}{2}$

Avec ces éléments, on peut donc interpréter la formule p :

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

Avec cette interprétation, la phrase logique donne ce sens. Une autre interprétation donnerait un sens totalement différent à la phrase logique. Voici une autre interprétation totalement différente :

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

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

Si Vladimir Putin est un enfant du père de Vladimir Putin alors il existe une personne telle que Barack Obama est un 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 propositionnelle, mais avec encore plus de souplesse.

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 $\exists d \in \mathfrak{R}$ 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 vraie aussi. Cela peut paraître bizarre, mais c'est correct.

4.4.2 Interprétation

Interprétation des symboles

L'approche que nous utilisons pour définir une sémantique de la logique des prédictats est très proche de l'approche que nous avons utilisée pour la logique propositionnelle. Nous allons définir une interprétation I de chaque formule, qui nous permettra de calculer si la formule est vraie ou fausse. Cependant, l'interprétation en logique des prédictats est plus générale qu'en logique propositionnelle. En plus des propositions, nous avons des variables,

des fonctions et des prédicts avec des arguments, et des quantificateurs (\forall et \exists).

Une *interprétation* I en logique des prédicts est une paire $I = (D_I, \text{val}_I)$, avec un ensemble D_I qui s'appelle le *domaine de discours* et une fonction val_I qui s'appelle la *fonction de valuation* qui renvoie un élément de D_I pour chaque symbole. Nous avons donc pour chaque symbole s :

- Si s est un symbole de prédict, $\text{val}_I(s) = P_I$ une fonction $P_I : D_I^n \rightarrow \{\text{true}, \text{false}\}$.
- Si s est un symbole de fonction, $\text{val}_I(s) = f_I$ une fonction $f_I : D_I^n \rightarrow D_I$ avec n le nombre d'arguments.
- Si s est une constante (une fonction avec zéro arguments), $\text{val}_I(s) = c$ un élément de D_I .
- Si s est une variable, $\text{val}_I(x) = x_I$, un élément D_I .

Cela implique chaque fonction correspond à une vraie fonction, chaque prédict correspond à un vrai prédict, et chaque constante et chaque variable correspondent à une constante dans le domaine de discours.

Attention à la différence entre les variables et les constantes. Pour les deux, l'interprétation est un élément de D_I , mais on peut utiliser les variables dans les quantificateurs et pas les constantes. Cela veut dire que dans une formule, la valeur d'une variable dépend de l'endroit où se trouve la formule, ce qui n'est pas vrai pour une constante.

Interprétation des termes et formules

Avec la fonction val_I qui est définie sur tous les symboles, nous pouvons définir une fonction VAL_I sur les termes et les formules :

$$\text{VAL}_I : \text{TERM} \cup \text{PRED} \rightarrow D_I \cup \{T, F\} \quad (4.1)$$

Ici, TERM est l'ensemble des termes et PRED est l'ensemble des formules. Nous avons donc :

- Si t est un terme, $t \rightarrow \text{VAL}_I(t)$.
- Si p est une formule, $p \rightarrow \text{VAL}_I(p)$.

Nous pouvons définir VAL_I avec val_I , en suivant la définition de la syntaxe des formules :

- $\text{VAL}_I(P(t_1, \dots, t_m)) = (\text{val}_I(P))(\text{VAL}_I(t_1), \dots, \text{VAL}_I(t_m))$.
- $\text{VAL}_I(\forall x.p) = T$ si pour chaque $d \in D_I$, $\text{VAL}_{\{x \leftarrow d\} \circ I}(p) = T$. Sinon, c'est F . En clair, d est l'interprétation de x dans la formule p . Si pour tous les d , l'interprétation de p est vraie, alors le quantificateur universel est vrai aussi.
- $\text{VAL}_I(\exists x.p) = T$ s'il existe un élément $d \in D_I$ tel que $\text{VAL}_{\{x \leftarrow d\} \circ I}(p) = T$. Sinon, c'est F .
- $\text{VAL}_I(p \wedge q) = T$ si $\text{VAL}_I(p) = T$ et $\text{VAL}_I(q) = T$. Si au moins un des deux est F , c'est F .

- $\text{VAL}_I(p \vee q) = T$ si $\text{VAL}_I(p) = T$ ou $\text{VAL}_I(q) = T$. Si tous les deux sont F , c'est F .
- $\text{VAL}_I(c) = \text{val}_I(c)$ si c est un symbole de constante.
- $\text{VAL}_I(x) = \text{val}_I(x)$ si x est un symbole de variable.
- $\text{VAL}_I(f(t_1, \dots, t_n)) = (\text{val}_I(f))(\text{VAL}_I(t_1), \dots, \text{VAL}_I(t_n))$ si f est un symbole de fonction.
- $\text{VAL}_I(\text{true}) = T$.
- $\text{VAL}_I(\text{false}) = F$.
- Dans cette énumération, j'ai omis de mentionner l'implication \Rightarrow et l'équivalence \Leftrightarrow . Je vous les laisse en exercice.

En décomposant une formule p en ses symboles de base, nous pouvons donc calculer $\text{VAL}_I(p)$, c'est-à-dire si la formule est vraie ou fausse, à partir de la fonction val_I .

Modèle

Un modèle d'un ensemble de formules est une interprétation qui rend toutes les formules vraies. Formellement, si on a un ensemble de formules $B = \{p_1, \dots, p_n\}$, une interprétation I pour B est un *modèle* si et seulement si :

$$\forall p_i \in B : \text{VAL}_I(p_i) = T \quad (4.2)$$

Chapitre 5

Preuves en logique des prédictats

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

- Des variables
- Des constantes
- Des fonctions (détaillé plus tard)
- Des prédictats
- Des quantificateurs

Les preuves en logique des prédictats ressemblent très fort aux preuves en logique propositionnelle. 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. Une preuve est toujours un objet formel :

1.	...	<i>Prémisses</i>
2.	Formule, règle	<i>Justification</i>
...
<i>n.</i>	Conclusion	<i>Justification</i>

Mais il est vrai que les preuves en logique des prédictats sont parfois délicates à cause des variables : occurrences libres (par quantifiées), occurrences liées par \forall et occurrences liées par \exists . Dans ce chapitre nous allons surtout étudier comment manipuler les variables et les quantificateurs correctement dans une preuve.

5.1 Exemple

Nous allons prouver que s'il est vrai que $\forall x \cdot P(x) \wedge Q(x)$ (prémisses) alors il est vrai que $\forall x \cdot P(x) \wedge (\forall x \cdot Q(x))$ (conclusion). Nous utilisons l'approche suivante pour traiter les quantificateurs :

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 : "Comme $P(x) \wedge Q(x)$ est vrai pour tout x , alors $P(x)$ 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.	$P(x) \wedge Q(x)$	Élimination 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 prédictats) :

- Élimination de \forall
- Introduction de \forall
- Élimination 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'autres¹).

Note

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

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 . 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 . On appelle souvent a, b, c des *paramètres*.

5.2 Règles en logique des prédictats

Nous allons maintenant introduire les règles de déduction pour les quantificateurs. D'abord, nous définissons une manipulation symbolique importante qui est utilisée dans ces règles : la substitution. Ensuite nous définissons les quatre règles pour les quantificateurs.

Comment savons-nous que les règles sont correctes ? On ne peut pas le prouver dans la logique des prédictats : on ne peut pas prouver l'exactitude des règles avec les règles elles-mêmes ! Il faut faire un raisonnement en dehors de la logique des prédictats. Nous justifions chaque règle avec un raisonnement basé sur un modèle de la formule. Si la logique des prédictats est notre langage pour formaliser les raisonnements, on peut donc dire que la justification des règles est faite en dehors de ce langage, donc en *meta-langage*.

5.2.1 La substitution

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

Si on a une formule quelconque p , la notation $p[x/t]$ veut dire de rem-

placer toutes les occurrences libres de x par t . Une *occurrence libre* d'une variable est une occurrence qui n'est pas dans la portée d'un quantificateur. Il faut faire attention aux variables dans t , pour éviter qu'une variable dans t ne rentre dans la portée d'un quantificateur, ce qui s'appelle la *capture de variable*. Pour éviter la capture, il faut faire un *renommage* de variable, c'est-à-dire, changer les noms des variables dans p . Voici un exemple de $p[x/y]$ où $p = P(x) \rightarrow \forall y \cdot (P(x) \wedge R(y))$:

1.	$P(x) \rightarrow \forall y \cdot (P(x) \wedge R(y))$	$[x/y]$ veut dire qu'on va remplacer toutes les occurrences libres de x par y .
2.	$P(y) \rightarrow \forall y \cdot (P(y) \wedge R(y))$	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 <i>capture de variable</i> , car la variable y est capturée par le quantificateur. Pour résoudre ce problème, on va effectuer un <i>renommage</i> .
3.	$P(y) \rightarrow \forall z \cdot (P(y) \wedge R(z))$	Résultat après renommage (pour éviter la capture de variable). On a changé le y en z (renommage) et remplacé le x par y (substitution).

Voici un autre exemple avec $p = \exists y.P(x, y)$. Nous donnons plusieurs possibilités de substitution correctes :

- $p[x/y] = \exists z.P(y, z)$ (attention : y a été renommée)
- $p[x/f(x)] = \exists y.P(f(x), y)$
- $p[x/c] = \exists y.P(c, y)$ (c est une constante)
- $p[x/z] = \exists y.P(z, t)$

5.2.2 Élimination de \forall

Parce que $\forall x.P(x)$ veut dire pour tout $x_I \in D_I : P_I(x_I)$ est vrai, on peut remplacer sans contrainte :

$$\begin{aligned} \forall x \bullet P(x) \rightarrow P(a) & \text{ a est une constante quelconque} \\ \rightarrow P(y) & \text{ y est une variable (parce que } P_I(y_I) \text{ est vrai)} \end{aligned}$$

Règle :

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

⚠ N'oubliez pas de faire le renommage si nécessaire

Exemple :

- | | |
|---|---------------------------|
| 1. $\forall x \bullet \forall y \bullet P(x,y)$ | Prémisse |
| 2. $\forall y \bullet P(x,y)$ | Élimination de \forall |
| 3. $P(x,x)$ | Élimination de \forall |
| 4. $\forall x \bullet P(x,x)$ | Introduction de \forall |

Le renommage n'est pas nécessaire dans la ligne (3) parce qu'il n'y a pas de capture.

5.2.3 Élimination de \exists

Il faut faire attention avec cette règle, parce que $\exists x.P(x)$ veut dire qu'il existe un élément $x_I \in D_I$ pour lequel $P_I(x_I)$ est vrai. On ne connaît pas cet élément, mais on peut introduire un symbole qui le représente :

$$\exists x \bullet P(x) \rightarrow P(a)$$

a = nouvelle constante qui n'apparaît nulle part ailleurs ($\text{val}_I(a) = x_I$)

$$\exists x \bullet P(x) \rightarrow P(z)$$

z = nouvelle variable dans la preuve ($\text{val}_I(z) = x_I$)

$$\exists x \bullet P(x) \not\rightarrow P(y) \text{ (pas autorisé)}$$

y = variable qui existe déjà dans la preuve, elle a déjà une valeur

Exemple 1 :

- | | |
|---|--|
| 1. $\exists x \bullet \text{chef}(x)$ | Prémisse |
| 2. $\exists x \bullet \text{voleur}(x)$ | Prémisse |
| 3. $\text{chef}(y)$ | Élimination de \exists |
| 4. $\text{voleur}(y)$ | Élimination de \exists (y pas nouvelle) |
| 5. $\text{chef}(y) \wedge \text{voleur}(y)$ | Conjonction |
| 6. $\exists y \bullet \text{chef}(y) \wedge \text{voleur}(y)$ | Introduction de \exists (FAUSSE) |

Exemple 2 :

- | | |
|---|----------|
| 1. $\exists x \bullet \text{chef}(x)$ | Prémisse |
| 2. $\exists x \bullet \text{voleur}(x)$ | Prémisse |

- | | |
|---|------------------------------------|
| 3. chef(y) | Élimination de \exists |
| 4. voleur (z) | Élimination de \exists |
| 5. chef(y) \wedge voleur(z) | Conjonction |
| 6. $\exists y \exists z$ chef(y) \wedge voleur(z) | Introduction de \exists (EXACTE) |

5.2.4 Introduction de \exists

Si $p[t]$ est vrai cela veut dire qu'il existe un élément $t_I = \text{VAL}_I(t) \in D_I$ avec $(\text{val}_I(p))(t_I)$. On peut donc introduire $\exists x$ car un élément existe qui rend vrai $p[x]$. Mais attention, on doit pouvoir trouver la formule originale à partir de $\exists x.p[x]$, sinon l'introduction du quantificateur " \exists " a changé le sens de la formule (capture!).

Règle :

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

Autorisé si la substitution $p[x/t]$ retrouve la formule originale.

Exemple :

$$\frac{P(y,y)}{\exists x \bullet P(x,x)}$$

$$\frac{P(y,x)}{\exists x \bullet P(x,x)}$$

Ceci n'est pas correct !

Le deuxième exemple ne marche pas parce que $P(x,x)[x/y]$ ne retrouve pas la formule originale.

5.2.5 Introduction de \forall

Règle :

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

- Si p n'a pas d'occurrence libre de x alors c'est OK

- Si p contient une occurrence 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 jusqu'à cet endroit ne doit mettre une contrainte sur x !

Deux conditions :

- x n'était pas libre dans une formule contenant un quantificateur \exists qu'on a éliminé.
- x n'est pas libre dans une prémissse (dans ce cas, x serait connu depuis le début donc il possède déjà une valeur).

Exemple :

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

5.3 Exemple plus conséquent

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. Terminons donc 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.

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

Prémisses

- Les maîtres et les 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. $\forall x (\text{maître}(x) \vee \text{esclave}(x) \implies \text{adulte}(x) \wedge \text{homme}(x))$ Prémissse

2. $\neg\forall x (\text{adulte}(x) \wedge \text{homme}(x))$ Prémissse
3. $\exists x \neg(\text{adulte}(x) \wedge \text{homme}(x))$ Théorème de 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. $\neg(\text{adulte}(x) \wedge \text{homme}(x))$ Élimination de \exists
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. $(\text{maitre}(x) \vee \text{esclave}(x) \implies \text{adulte}(x) \wedge \text{homme}(x))$ Élim. de \forall
On peut retirer le \forall en réduisant le champ de x aux x rendant vraie la proposition.
6. $\neg(\text{maitre}(x) \vee \text{esclave}(x))$ Modus Tollens
7. $\neg\text{maitre}(x) \wedge \neg\text{esclave}(x)$ De Morgan
8. $\neg\text{maitre}(x)$ Simplification
9. $\exists y \neg\text{maitre}(y)$ Introduction de \exists
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.

Les mathématiciens ont fait de grands efforts pour formaliser les mathématiques avec la logique des prédictats. Nous mentionnons le Principia Mathematica de Whitehead et Russell, et les ouvrages nombreux de Nicolas Bourbaki.

5.4 Note historique

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.
Principia Mathematica (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'Algorithm de Preuves (1965) :
 - Alan Robinson invente la Règle de Résolution (qui va être expliquée au chapitre suivant)
 - 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 6

Algorithme de preuve pour la logique des prédicats

Dans le chapitre précédent nous avons introduit des règles pour faire des preuves manuelles. Avec l'arrivée des ordinateurs, les logiciens ont tout de suite essayé de faire des preuves mécanisées. Nous avons vu dans le chapitre 3 (Section 3.7) un algorithme de preuve pour la logique propositionnelle qui utilise la réfutation. La logique des prédicats est beaucoup plus expressive que la logique propositionnelle et les raisonnements sont plus délicats (voir chapitre précédent !).

Est-ce que nous pouvons faire un algorithme de preuve pour la logique des prédicats ? Cela ne semble pas évident. Beaucoup de logiciens ont travaillé sur cette question. La réponse affirmative à la question est un des grands résultats des mathématiques du 20ème siècle. Dans ce chapitre nous verrons un algorithme de preuve avec une grande puissance. 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.

6.1 Propriétés de l'algorithme

L'algorithme pour la logique des prédicats ne garde pas toutes les propriétés de l'algorithme pour la logique propositionnelle, car la logique des prédicats est beaucoup plus expressive. En particulier, là où l'algorithme pour la logique propositionnelle est décidable, l'algorithme pour la logique des prédicats n'est que semi-décidable.

Supposons B l'ensemble des axiomes (prémisses) et T le théorème que l'on veut prouver. On a les propriétés suivantes :

- L'algorithme est *adéquat* (“*consistent*”) : Si $B \vdash T$ alors $B \models T$

Si l'algorithme trouve une preuve de T avec les axiomes B alors T sera vraie dans tous les modèles de B .

- L'algorithme est *complet* (“*complete*”) : Si $B \models T$ alors $B \vdash T$
Si T est vrai dans tous les modèles de B alors l'algorithme trouvera une preuve de T avec les axiomes B .
- L'algorithme est *semi-décidable* :
 - Si $B \models T$ alors l'algorithme trouve une preuve.
 - Si $B \not\models T$ il peut tourner en rond indéfiniment.

Si T est vrai dans tous les modèles, l'algorithme trouvera une preuve, mais si T n'est pas vrai dans tous les modèles, l'algorithme peut tourner en rond et ne jamais se terminer. Le problème est donc que quand l'algorithme prend trop de temps à trouver une preuve, à un certain moment 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 où une preuve est trouvée.

6.2 Concepts de base

L'algorithme de preuve est basé sur deux idées :

- Simplification des prémisses. On transforme les prémisses dans une forme uniforme et simple qui s'appelle forme clausale.
- Simplification des règles d'inférence. On garde une seule règle d'inférence, la résolution, qui marche sur la forme clausale.

Avec ces deux simplifications, la définition de l'algorithme devient simple aussi.

6.2.1 Forme normale conjonctive (forme clausale)

Pour transformer les prémisses en forme clausale, il y a trois étapes :

1. Formule \rightarrow 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 préservés durant cette transformation, la formule avant la transformation a les mêmes modèles que la formule après la transformation. Cela est vrai parce que les deux formules sont équivalentes.

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 (la satisfaisabilité), mais pas les modèles eux-mêmes. Ils doivent être modifiés pour conserver la même signification.

3. Forme Skolem → forme normale conjonctive (forme clausale) :

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

Cette transformation consiste à transformer la formule en une conjonction de disjonctions (\wedge de \vee). Les règles pour cette transformation sont les mêmes règles qu'en logique propositionnelle.

Les modèles sont préservés lors de cette transformation.

6.2.2 Résolution

La seule règle d'inférence gardée par l'algorithme s'appelle la résolution. Cette règle existe déjà dans une forme plus simple dans la logique des propositions :

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

Cette règle est simple en logique des propositions, car il n'y a pas de variables : L et $\neg L$ sont directement comparables (il y a le même L dans les deux). En logique des prédictats ce n'est plus vrai : on peut avoir $L_1 \vee C_1$ et $\neg L_2 \vee C_2$ avec L_1 et L_2 qui ont des variables différentes. On ne peut donc pas faire la résolution immédiatement.

Pour pouvoir faire la résolution, il va falloir en quelque sorte "rendre L_1 et L_2 identiques". On va donc dire : ce ne sont peut-être pas toujours les mêmes, mais, pour certaines valeurs, ils sont identiques. L'idée sera donc de les rendre identiques en substituant leurs variables par d'autres judicieusement choisies. Cette opération s'appelle l'*unification*. Prenons un exemple :

$$\begin{aligned} L_1 &= P(x, a) \\ L_2 &= P(y, z) \end{aligned}$$

Pour que L_1 et L_2 deviennent identiques, on va restreindre leurs variables en faisant une substitution. Nous avons les variables x, y, z et une constante a . Une substitution possible est de remplacer x par y et z par a . On écrit $\sigma = \{(x, y), (z, a)\}$ où σ est la substitution. Si on applique σ à L_1 et L_2 on obtient :

$$L_1\sigma = P(x, a)\sigma = P(y, a)$$

$$L_2\sigma = P(y, z)\sigma = P(y, a)$$

Les deux formules sont maintenant identiques, et on peut donc faire la résolution. Cette résolution marche pour toutes les valeurs qui sont limitées par la substitution. Le résultat ne sera donc pas général. En résumant, on peut écrire la règle de résolution de façon plus générale en y mettant la substitution :

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

Pour faire l'inférence on choisit le σ le plus général possible, ce que nous appellons l'*unificateur le plus général* ("most general unifier"). On peut démontrer qu'un unificateur le plus général existe toujours. C'est cette règle d'inférence que nous allons utiliser dans l'algorithme de preuve.

6.3 Transformation en forme prénexe

Etapes de la transformation en formule logiquement équivalente :

1. Éliminer \Leftrightarrow et \Rightarrow
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 prédictats. 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 prédictats à l'avant de la formule.

6.3.1 Exemple d'une transformation en forme prénexe

1. $\forall x[p(x) \wedge \neg(\exists y)\forall x(\neg q(x, y))] \Rightarrow \forall z\exists v \bullet p(a, x, y, v))]$
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 \Rightarrow
3. $\forall x[p(x) \wedge \neg(\exists y)(\forall u)(\neg \neg q(u, y) \vee \forall z\exists v \bullet r(a, u, y, v))]$
Renommage des variables et suppression des quantificateurs inutiles
4. $\forall x[p(x) \wedge \forall y \neg (\forall u)(\neg q(u, y) \vee \exists v \bullet r(a, u, y, v))]$
 $\neg\exists y$ 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.

6.4 Transformation en forme Skolem

6.4.1 Intuition

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

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

Dans ce cas-ci, la valeur de u dépend des valeurs de x et y . Lorsqu'on a choisi x et y , on est alors libre 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 $(\exists u)$.

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

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

6.4.2 Règle

Pour chaque élimination d'un quantificateur existentiel $(\exists x)$, on remplace sa variable quantifiée par une fonction $f(x_1, \dots, x_n)$ dont les arguments sont les variables des quantificateurs universels dont x est dans la portée. Cette transformation ne conserve pas les modèles parce que la formule originale n'utilise pas le symbole f et la formule transformée utilise un symbole f . Mais la satisfaisabilité est conservée : la première formule a un modèle si et seulement si la deuxième formule en a un.

Justification par un exemple :

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

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

Les modèles de p ne sont les mêmes que les modèles de p_s .

Pour p

- Interprétation I
- $D_I = \text{Professeur} \cup \text{Université}$
- $\text{val}_I(P) = P_I = \text{"a enseigné à l'université"}$
- $\text{val}_I(Q) = Q_I = \text{"est diplômé de l'université"}$
- $\text{val}_I(f)$ n'existe pas (le symbole f n'existe pas dans p).

Pour p_s

- Interprétation $I' = \{f \leftarrow f_i\} \circ I$ (c'est une extension de I)
- $D_I = \text{Professeur} \cup \text{Université}$
- $\text{val}_I(P) = P_I = \text{"a enseigné à l'université"}$
- $\text{val}_I(Q) = Q_I = \text{"est diplômé de l'université"}$
- $\text{val}_I(f) = f_I : \text{Professeur} \times \text{Université} \rightarrow \text{Université}$
- $f_I(a, b) = \text{"l'université ayant dû diplômer } a \text{ pour que } a \text{ puisse enseigner à } b\text{"}$

p admet un modèle (I) si et seulement si 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.

6.5 Transformation en forme normale conjonctive

On fait les mêmes manipulations qu'en logique des propositions (voir Section 3.7.1). On transforme la formule en une conjonction de disjonctions.

6.6 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.

Exemple 1 :

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

Dans un modèle de $\{L_1, L_2\}$ 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 2 :

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

Il n'y pas de substitution qui existe, car on a deux constantes différentes, l'intersection des deux ensembles est vide. L_1 et L_2 ne sont donc pas unifiables.

Exemple 3 :

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

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

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

L_1 et L_2 sont donc unifiables. On peut démontrer que σ_2 est l'unificateur le plus général (souvent abrégé en *upg*, en anglais *mgu* : most general unifier). Il donne les ensembles les plus grands.

- Pour que la démonstration soit la plus générale possible, on préfère toujours l'unificateur σ le plus générale.
- On peut démontrer que l'unificateur le plus général est unique. La démonstration de ce fait est hors de portée pour ce cours.
- L'upg 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.
- $\{L^+, L^-\}$ sont unifiables par l'upg σ .

Alors

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

6.7 Algorithme

L'algorithme de preuve prend comme entrées un ensemble de formules (“axiomes”) $\{\text{Ax}_1, \dots, \text{Ax}_n\}$ et une formule à prouver Th . Chaque formule

est en forme normale conjonctive. L'algorithme utilise la preuve par contradiction (réfutation) : nous allons donc ajouter $\neg\text{Th}$ à l'ensemble d'axiomes et essayer de déduire une contradiction (false). Les déductions utilisent la règle de résolution, qui est bien adaptée à la forme normale conjonctive.

L'algorithme n'est pas garantie de terminer. S'il termine avec une contradiction, cela démontre Th. S'il termine sans contradiction, cela démontre que l'on ne peut pas prouver Th. S'il ne termine pas, on est dans le cas de la semi-décidabilité.

6.7.1 Définition de l'algorithme

```

Input:  $S := \{\text{Ax}_1, \dots, \text{Ax}_n, \neg\text{Th}\}$ 
while  $\text{false} \notin S$  et il existe une paire de clauses dans  $S$  résolvables et non résolues. do
    Choisir  $p_i, p_j$  dans  $S$  tel qu'il existe :
    —  $L^+$  dans  $p_i$ 
    —  $L^-$  dans  $p_j$ 
    —  $\{L^+, L^-\}$  unifiable par upg  $\sigma$ 
    Calculer :
    —  $r := (p_i - [L^+]) \vee p_j - [\neg L^-]\sigma$ 
    —  $S_i = S \cup \{r\}$ 
end
if  $\text{false} \in S$  then
    |  $S$  inconsistent, donc Th prouvé.
else
    |  $S$  consistent, donc Th non prouvé.
end
```

6.7.2 Exemple d'exécution

Voici un exemple avant la mise en forme clausale :

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

Initialisation de S

Après la mise en forme clausale cela donne :

- P1 : $(\forall x)\neg\text{homme}(x) \vee \neg\text{fume}(x) \vee \text{mortel}(x)$

- P2 : $(\forall x)\neg animal(x) \vee mortel(x)$
- P3 : $homme(Ginzburg)$
- P4 : $fume(Ginzburg)$
- P5 : $\neg mortel(Ginzburg)$

Itérations

1

- P1 + P5
- $\sigma = \{(x, Ginzburg)\}$
- $r = P6 = \neg homme(Ginzburg) \vee \neg fume(Ginzburg)$

2

- P3 + P6
- $\sigma = \{\}$
- $r = P7 = \neg fume(Ginzburg)$

3

- P4 + P7
- $\sigma = \{\}$
- $r = \text{false}$

Il y a une inconsistance donc le candidat théorème est prouvé.

Non-déterminisme

Cet algorithme a plusieurs sources de non-déterminisme, c'est-à-dire le choix des clauses et le choix des littéraux dans les clauses. Ces choix sont faits à chaque itération. Avec un mauvais choix, l'algorithme peut ne pas terminer ou faire des déductions inutiles. Par exemple, si on fait un autre choix lors de la première itération :

1

- P2 + P5
- $\sigma = \{(x, Ginzburg)\}$
- $r = \neg animal(Ginzburg)$

C'est une déduction inutile qui n'aide pas pour la démonstration de Th.

6.7.3 Stratégies

Pour que cet algorithme soit utilisable en pratique, il est très important de concrétiser des stratégies pour les choix :

- Quelles paires p_i et p_j choisir ?
- Quelles L^+ et L^- choisir ?

Il y a deux possibilités pour ces stratégies qui sont utilisées en pratique, qui donne lieu à deux sortes de programmes que l'on appelle *assistant de preuve*

ou *langage logique*. Dans ces deux cas, cela a donné lieu à des développements signifiants.

Un assistant de preuve

Un assistant de preuve est un outil qui aide un mathématicien humain 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 fut un tour de force. Désormais, il existe une preuve complètement formalisée, sans erreur pour ce théorème.

Un langage logique

Un langage logique est un langage de programmation basé sur la logique des prédictats. Le plus célèbre de ces langages est *Prolog*, inventé en 1972 par Alain Colmerauer et Robert Kowalski. Prolog utilise *volontairement* des stratégies naïves qui permettent de rendre l'algorithme prévisible. Les axiomes deviennent un *programme*. Cette approche donne un langage de programmation pratique. Un exemple de stratégie naïve est la stratégie LUSH utilisée par Prolog qui choisit de haut vers le bas les paires p_i, p_j , et de gauche à droite dans chaque p_i . LUSH est un acronyme qui veut dire *Linear resolution with Unrestricted Selection for Horn clauses*. La programmation logique sera expliquée plus en détail dans le chapitre 8.

Chapitre 7

Théories logiques

On peut utiliser la logique des prédicats pour définir et étudier des structures mathématiques. L'avantage est qu'une définition avec la logique est complètement précise. La définition d'une structure contiendra deux types de formules : des *axiomes* et des *règles d'inférence*. Quelques exemples simples des structures que l'on peut définir sont l'ordre partiel, les treillis, les groupes, les entiers, les chaînes, les arbres, les ensembles, les relations et les fonctions. Il y en a beaucoup plus que cela ; en fait quasi toutes les mathématiques peuvent être formalisées avec la logique des prédicats. Dans ce chapitre nous allons voir quelques exemples de ces formalisations pour des structures discrètes.

Ces formalisations peuvent être utilisées pour la programmation (par ex. avec Prolog) et aussi dans les assistants de preuve. Pour ces deux utilisations, il faut faire très attention à la manière dont on définit les axiomes et les règles : cela peut avoir une grande influence sur l'efficacité de l'exécution. Mais dans ce chapitre nous ne nous intéressons pas à l'efficacité, mais plutôt à la simplicité et la compréhension.

Logique du premier ordre

La logique des prédicats est parfois appelé la logique du premier ordre. Dans ce chapitre nous allons parfois utiliser cette terminologie. On l'appelle premier ordre parce que le domaine des quantificateurs $\forall x$ et $\exists x$ est D_I : les valeurs de $\text{val}_I(x)$ sont toujours dans D_I . Il existe aussi des logiques d'ordres supérieurs ; dans ce cas le domaine des quantificateurs peut être autre chose. Par exemple, dans une logique du second ordre, on peut quantifier sur les prédicats et pas simplement sur les éléments du domaine de discours. On pourra donc dire $\forall p$ ou $\exists p$ où p est un prédicat. Cela donne une expressivité supplémentaire par rapport à la logique du premier ordre, mais les raisonnements et les preuves deviennent beaucoup plus complexes.

Il y a aussi moins de résultats généraux sur cette logique et la plupart des mathématiques utilisées en pratique n'ont pas besoin de cette logique pour être formalisée. Nous n'abordons pas les logiques d'ordres supérieurs dans ce cours.

7.1 Théorie du premier ordre

Le concept de base est la *théorie du premier ordre*. Une théorie du premier ordre est un ensemble B de formules qui représente les axiomes et les règles d'inférence d'une structure mathématique. Nous allons nous restreindre au modèles : les interprétations de B où les axiomes et les règles sont valides.

7.1.1 Définition d'une théorie

Une théorie contient les parties suivantes :

- Un sous-langage de la logique du premier ordre :
- Vocabulaire : constantes, fonctions, prédictats ;
- Règles syntaxiques et sémantiques sur ce vocabulaire ;
- Ensemble d'axiomes
- Ensemble de règles d'inférence

Les axiomes et les règles d'inférence sont des formules *fermées*, c'est-à-dire des formules ne contenant pas de variables libres. L'idée est que ces formules expriment des faits sur tous les éléments du domaine.

7.1.2 Exemple : théorie des liens familiaux (FAM)

Comme premier exemple nous définissons une théorie qui permettra de raisonner sur les liens familiaux.

1. Vocabulaire :

- 2 symboles de fonctions à un paramètre : $p/1, m/1$
- 3 symboles de prédictats à deux paramètres : $P/2, GM/2, GP/2$

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

2. Axiomes :

$(\forall x) (P(x, p(x)))$	(père)
$(\forall x) (P(x, m(x)))$	(mère)
$(\forall x)(\forall y) (P(x, y) \Rightarrow GP(x, p(y)))$	(grand-père)
$(\forall x)(\forall y) (P(x, y) \Rightarrow GM(x, m(y)))$	(grand-mère)

3. Règles :

Les règles sont uniquement celles de la logique des prédictats.

Première interprétation

D_I : personnes

$\text{val}_I(p) = \text{"père de"}$

$\text{val}_I(m) = \text{"mère de"}$

$\text{val}_I(P) = \text{"Parent"}$

$\text{val}_I(GP) = \text{"Grand-père"}$

$\text{val}_I(GM) = \text{"Grand-mère"}$

père de : Pers \rightarrow Pers : $d \rightarrow \text{"père de" } d$

mère de : Pers \rightarrow Pers : $d \rightarrow \text{"mère de" } d$

$\text{Parent}(d_1, d_2) = T$ ssi d_2 est un parent de d_1

$\text{Grand-père}(d_1, d_2) = T$ ssi d_2 est un grand-père de d_1

$\text{Grand-mère}(d_1, d_2) = T$ 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 scientifique, 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

$D_J : \mathbb{N}$

$\text{val}_J(p) = "p_J"$

$\text{val}_J(m) = "m_J"$

$\text{val}_J(P) = "P_J"$

$\text{val}_J(GP) = "GP_J"$

$\text{val}_J(GM) = "GM_J"$

$p_J : \mathbb{N} \rightarrow \mathbb{N} : d \rightarrow 2d$

$m_J : \mathbb{N} \rightarrow \mathbb{N} : d \rightarrow 3d$

$P_J(d_1, d_2)$ ssi $d_2 = 2d_1$ ou $d_2 = 3d_1$

$GP_J(d_1, d_2)$ ssi $d_2 = 4d_1$ ou $d_2 = 6d_1$

$GM_J(d_1, d_2)$ ssi $d_2 = 6d_1$ ou $d_2 = 9d_1$

Cette interprétation est également un modèle de FAM.

Exemple de preuve

Voici une preuve dans la théorie FAM. Cette preuve est un exemple de l'approche syntaxique qui est mentionnée dans la section suivante. La propriété qui est prouvée est vraie dans les deux interprétations. On veut montrer :

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

1. $(\forall x)(\forall y) (P(x, y) \Rightarrow GM(x, m(y)))$ (Axiome de grand-mère)
2. $(\forall x) (P(x, p(x)) \Rightarrow GM(x, m(p(x))))$ (Elimination de $\forall y$ et substitution $y/p(x)$)
3. $(\forall x)P(x, p(x)) \Rightarrow (\forall x)GM(x, m(p(x)))$ (Distributivité \forall / \Rightarrow)
4. $(\forall x)P(x, p(x))$ (Axiome de père)
5. $(\forall x)GM(x, m(p(x)))$ (Modus ponens)
6. $(\forall x)(\exists y)GM(x, y)$ (Introduction de \exists)

Attention, cette preuve utilise le schéma de distributivité de \forall sur \Rightarrow dans la ligne (3). Ce schéma dit que pour toutes formules p et q l'on peut remplacer $(\forall x)(p \Rightarrow q)$ par $(\forall x)p \Rightarrow (\forall x)q$. Je vous laisse comme exercice de prouver la validité de ce schéma. Indice : on peut utiliser l'approche sémantique, comme définie la section suivante.

7.2 Propriétés des théories

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

$$\models_{Th} p$$

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

- 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 :

$$p \models_{Th} q$$

- 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).

7.2.1 Établir la validité

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

1. L'approche *sémantique* : on prend un modèle I quelconque de Th et on évalue $VAL_I(p)$ en utilisant le fait que $VAL_I(Ax_i) = T$.
2. L'approche *syntaxique* : on essaye de construire une preuve de p à partir des axiomes, en appliquant les règles de Th . Cette approche s'appelle aussi *théorie de la preuve*.

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

7.2.2 Qualité d'une théorie

Une théorie peut posséder certaines qualités qui la rend meilleure que d'autres théories qui ne les possèdent pas. Une "bonne" théorie est :

1. *consistante* : il est impossible de déduire p et $\neg p$ de la même théorie.
2. *minimale* : les axiomes sont indépendants : $\{Ax_1, \dots, Ax_n\} \models Ax_k$
On peut vérifier la minimalité en construisant une interprétation J telle que :
 $\text{VAL}_J(Ax_k) = \text{false}$
 $\text{VAL}_J(Ax_i) = \text{true}$ pour $i \neq k$
3. *complète* : les axiomes suffisent pour prouver les propriétés qui nous intéressent. Sinon il faut en ajouter.

Exemple dans l'astronomie

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

7.3 Opérations sur les théories

Nous allons définir une opération, l'extension, qui permet de créer une nouvelle théorie par rapport à une théorie existante, et deux opérations, l'inclusion et l'équivalence, qui permettent de comparer deux théories.

7.3.1 Extension d'une théorie

Lorsqu'on a une théorie déjà existante, on souhaite parfois l'étendre afin de la rendre plus complète. On dit que Th_2 est une *extension* de Th_1 si :

- Le vocabulaire de Th_1 est inclus dans le vocabulaire de Th_2 .
- Tout axiome de Th_1 est axiome de Th_2 .

On peut donc étendre le vocabulaire et ajouter des axiomes. 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 la section précédente.

Exemple 1

Considérons le nouvel axiome suivant, que nous noterons Ax :

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

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 : personne est parent de lui-même.

En revanche, si on considère la deuxième interprétation (deuxième modèle, noté J) de FAM qui associe les symboles p et m aux fonctions mathématiques $p_J : \mathbb{N} \rightarrow \mathbb{N} : d \mapsto 2d$ et $m_J : \mathbb{N} \rightarrow \mathbb{N} : d \mapsto 3d$, 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 le nouvel axiome suivant, que nous noterons Adam :

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

où a est une constante arbitraire. Notons la théorie étendue $\text{FAM}' = \text{FAM} + \text{Adam}$. 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 :

$(\forall x)P(x, p(x))$	Axiome père
$P(a, p(a))$	Elimination \forall
$(\exists y)P(a, y)$	Introduction \exists

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

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

Par la règle de preuve par contradiction, on démontre qu'aucun modèle n'est possible pour la théorie étendue FAM'. Nous avons vu que l'extension d'une théorie peut réduire le nombre de modèles : ici on voit qu'il faut bien vérifier que ce nombre ne devient pas 0.

7.3.2 Comparaison des théories

Dans cette section, nous abordons la comparaison de différentes théories : inclusion, équivalence et quelques corollaires. Dans ce qui suit, on prend 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 .

Tout modèle de Th_2 est donc aussi un modèle de Th_1 . Mais pas inversement : il est possible qu'il existe des modèles de Th_1 qui ne sont pas des modèles de Th_2 . C'est parce que Th_2 peut contenir des formules valides qui ne le sont pas dans Th_1 .

Équivalence

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 théories est également modèle de l'autre.

Il est important de bien faire la différence entre l'extension d'une théorie et l'inclusion d'une théorie dans une autre. La confusion est possible parce que dans les deux cas, le nombre de modèles peut être réduit. Mais il ne faut pas oublier que l'extension est définie par rapport aux axiomes et l'inclusion est définie par rapport aux modèles. Pour l'extension on ajoute des axiomes et on ne regarde pas les modèles. Pour l'inclusion on compare les modèles et on ne regarde pas les axiomes. On peut dire que l'extension est une manipulation *yntaxique* (on change les axiomes ce qui permette de nouvelles preuves) et l'inclusion est une manipulation *sémantique* (on fait une comparaison entre ce que modélisent les deux théories).

7.3.3 Corollaires

On note respectivement V_t , M_t et Ax_t le vocabulaire, les modèles et les axiomes d'une théorie t .

Si $V_{Th_1} \subseteq V_{Th_2}$ et $M_{Th_2} \subseteq M_{Th_1}$ (tout modèle de Th_2 est aussi modèle de Th_1) alors Th_1 est contenue dans Th_2 . Attention à la direction : M_{Th_2} est bien à gauche !

Si $V_{Th_1} \subseteq V_{Th_2}$ et $Ax_{Th_1} \subseteq Ax_{Th_2}$ (tout axiome de Th_1 est aussi axiome de Th_2) alors Th_1 est contenue dans Th_2 . Il est donc vrai que $M_{Th_2} \subseteq M_{Th_1}$. On peut donc dire qu'une extension donne lieu à une inclusion (mais pas inversément).

Si $V_{Th_1} = V_{Th_2}$ et pour tout axiome $p \in Ax_{Th_1}$ nous avons $\models_{Th_2} p$ et pour tout axiome $q \in Ax_{Th_2}$ nous avons $\models_{Th_1} q$, alors Th_1 et Th_2 sont équivalentes.

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

7.4 Théorie des ordres partiels stricts (OPS)

Le concept d'ordre partiel strict est un des concepts de base des mathématiques. Nous allons le définir par une théorie. Ensuite nous donnons deux interprétations différentes de cette théorie.

Vocabulaire

- Le symbole $P/2$

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 :

- $D_I = \mathbb{Z}$
- $\text{val}_I(P) = <$

Lorsque l'on écrit $\text{val}_I(<) = <$, le premier symbole $<$ est un mot de vocabulaire dans la théorie tandis que le deuxième est une fonction sur les entiers. 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 :

- $D_J = \mathbb{Z}$
- $\text{val}_J(P) = " \neq "$

Ici on change le sens du symbole $<$:

$$\text{val}_J(<) = " \neq "$$

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

$$x < y \wedge y < z \not\Rightarrow x < z$$

$$5 \neq 3 \wedge 3 \neq 5 \not\Rightarrow 5 \neq 5$$

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 !

7.5 Théorie de l'égalité (EG)

Le concept d'égalité est omniprésent dans les mathématiques. Nous allons définir une théorie pour le formaliser. D'abord, il faut faire attention à la notation. Il existe différents symboles pour représenter l'égalité :

- $E(x,y)$
- $x = y$
- $x == y$

Nous allons utiliser “==” dans la suite de ce cours pour représenter l'égalité.

La théorie de l'égalité est souvent *ajoutée* à d'autres axiomes pour être utile en pratique. Dans cette section on donne que les axiomes d'égalité. Dans la suite nous allons voir comment ajouter l'égalité à d'autres théories.

7.5.1 Axiomes

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

1. Réflexivité
 $\forall x, x == x$
2. Symétrie
 $\forall x, \forall y, x == y \Rightarrow y == x$

3. Transitivité

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

4. Substituabilité dans les fonctions

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

5. Substituabilité dans les prédictats

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

Les deux derniers axiomes sont des *schémas* : chaque schéma définit plusieurs axiomes, où on remplace f par tout symbole de fonction et P par tout symbole de prédictat.

7.5.2 Règles d'inférence

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

Substituabilité fonctionnelle

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

Substituabilité prédicative

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

7.5.3 Théorème sur la sémantique de l'égalité

Si $\text{VAL}_I(t_1) == \text{VAL}_I(t_2)$ alors $\text{VAL}_I(t_1 == t_2) = \text{true}$

L'égalité comme elle est définie ici est une propriété syntaxique qui nous permet de faire des preuves. Avec ce théorème, nous pouvons aussi raisonner sur un modèle (c'est-à-dire la sémantique) pour déterminer l'égalité. C'est-à-dire, si deux termes donnent la même entité dans le domaine de discours, alors on peut dire que les deux termes sont égaux.

Preuve

Voici la preuve étape par étape :

- Soit I un modèle de EG et deux termes quelconques t_1 et t_2 .
- Par définition $\text{VAL}_I(t_1 == t_2) = \text{VAL}_I(==)(\text{VAL}_I(t_1), \text{VAL}_I(t_2))$.
- On pose $\text{VAL}_I(==) = E_I$ et $\text{VAL}_I(t_1) = e_1$ et $\text{VAL}_I(t_2) = e_2$.
- La prémissse du théorème nous dit que $e_1 = e_2 = e$.
- Donc $\text{VAL}_I(t_1 == t_2) = E_I(e, e)$.

- L'axiome de réflexivité nous dit $\text{VAL}_I(\forall x. x == x) = \text{true}$.
- La sémantique de \forall nous dit que pour tout $d \in D_I$, il est vrai que $\text{VAL}_I(d == d) = \text{true}$.
- On prend $d = e$ et on peut conclure que $E_I(e, e) = \text{true}$.

Remarquez que ceci n'est pas une preuve en logique des prédictats, mais une preuve en métalangage, c'est-à-dire une preuve qui parle de la logique des prédictats. On ne peut pas formaliser cette preuve comme un objet formel qui est une preuve en logique des prédictats. Toute l'argumentation sur la sémantique de la logique des prédictats et la justification des règles de preuve sont des raisonnements en métalangage.

7.6 Théorie de l'ordre partiel (OP)

Pour définir la théorie de l'ordre partiel, nous prenons EG et nous ajoutons un symbole en plus du symbole d'égalité : On ajoute un deuxième symbole dans le langage en plus du symbole d'égalité :

\leq

Les axiomes d'égalité sont déjà présents. Il nous reste de faire les axiomes pour ce nouveau symbole.

7.6.1 Axiomes

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

1. Réflexivité
 $\forall x, x \leq x$
2. Anti-symétrie
 $\forall x, \forall y, x \leq y \wedge y \leq x \Rightarrow y == x$
3. Transitivité
 $\forall x, \forall y, \forall z, (x \leq y \wedge y \leq z) \Rightarrow x \leq z$
4. Substituabilité à gauche
 $\forall x_1, \forall x_2, \forall x, x_1 == x \Rightarrow x_1 \leq x_2 \Leftrightarrow x \leq x_2$
5. Substituabilité à droite
 $\forall x_1, \forall x_2, \forall x, x_2 == x \Rightarrow x_1 \leq x_2 \Leftrightarrow x_1 \leq x$

7.6.2 Exemple de théorème

Nous allons prouver le théorème suivant :

$$\models \forall x. \forall y. [x == y \Leftrightarrow (x \leq y) \wedge (y \leq x)]$$

Preuve

- \Leftrightarrow equivaut à $\Leftarrow \wedge \Rightarrow$.
- \Leftarrow est démontré immédiatement par l'axiome antisymétrique.
- \Rightarrow demande à démontrer $\models \forall x. \forall y. [x == y \Rightarrow x \leq y \wedge y \leq x]$.

Il nous reste de démontrer \Rightarrow .

1. $\forall x. \forall y. \forall z. x == y \Rightarrow z \leq x \Leftrightarrow y \leq x$	substituabilité à gauche
2. $\forall x. \forall y. x == y \Rightarrow x \leq x \Leftrightarrow y \leq x$	\forall Élimination
3. $x == y \Rightarrow x \leq x \Leftrightarrow y \leq x$	\forall Élimination (2 fois)
4. $\forall x. \forall y. \forall z. x == y \Rightarrow x \leq z \Leftrightarrow x \leq y$	substituabilité à droite
5. $\forall x. \forall y. x == y \Rightarrow x \leq x \Leftrightarrow x \leq y$	\forall Élimination
6. $x == y \Rightarrow x \leq x \Leftrightarrow x \leq y$	\forall Élimination (2 fois)
7. $x == y$	Supposition (conditionnel)
8. $y \leq x$	Modus ponens (3,7)
9. $x \leq y$	Modus ponens (6,7)
10. $y \leq x \wedge x \leq y$	Conjonction (8,9)
11. $x == y \Rightarrow y \leq x \wedge x \leq y$	Conclusion
12. $\forall x. \forall y. x == y \Rightarrow y \leq x \wedge x \leq y$	\forall Introduction (2 fois)

Les lignes (8) et (9) sont légèrement simplifiés : pour la rigueur il faut utiliser la réflexivité et la définition de l'équivalence. Je vous laisse cela en exercice.

7.6.3 Exemples de modèles d'OP

Les ordres partiels sont omniprésents dans les mathématiques et l'informatique. En voici quelques exemples.

- $I_1 : D_{I_1} = \mathbb{Z}$
 $val_{I_1}(==) = '='$: égalité d'entiers
 $val_{I_1}(\leq) = ' \leq '$: plus petit ou égal pour les entiers
Cette interprétation va satisfaire tous les entiers.
- $I_2 : D_{I_2} = P(E)$ ensemble des sous-ensembles de E
 $val_{I_2}(==) = '='$: égalité d'ensemble
 $val_{I_2}(\leq) = ' \subseteq '$: inclusion d'ensemble
- $I_3 : D_{I_3} = ALPH^2 = (l_1, l_2), \dots$ 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 $l_j = l_q$
- $I_4 : D_{I_4} =$ ensemble de listes
 $val_{I_4}(==) = =$ égalité de liste (si elles possèdent les mêmes 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 :

$VAL_{I_5}(==) = "\Leftrightarrow"$: L'égalité est synonyme d'équivalence en logique des propositions

$VAL_{I_5}(\leq) = "\sqsubseteq"$: 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 $\{ (x_i, t_i) \dots \}$: un ensemble de paires avec une variable et un terme $VAL_{I_6}(\leq)$ = "moins général que"

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

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. Ce qui est métalangage dans un raisonnement peut devenir langage dans un autre raisonnement apparenté.

7.7 Théorie des ensembles

Nous introduisons une version simple de la théorie des ensembles comme prérequis pour la spécification des systèmes avec l'approche Z. Informellement, il y a deux manières de définir des ensembles :

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

Les axiomes vont en partie suivre cette intuition. Nous introduisons maintenant les axiomes pour les ensembles.

Axiomes

— Égalité (première 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)

$$\mathcal{U} \text{ (par exemple : } \mathbb{N}, \mathbb{R}, \mathbb{Z})$$

— Sous-ensemble

$$A \subseteq B \Leftrightarrow (\forall x) (x \in A \Rightarrow x \in B)$$

$$A \not\subseteq B \Leftrightarrow \neg(A \subseteq B)$$

$$A \subset B \Leftrightarrow A \subseteq B \wedge A \neq B$$

— 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

$$\emptyset \subseteq A$$

$$A \subseteq A$$

$$A \subseteq B \wedge B \subseteq C \Rightarrow A \subseteq C$$

— Égalité (2e définition)

$$(A = B) \Leftrightarrow (A \subseteq B) \wedge (B \subseteq A)$$

— Taille d'un ensemble

— Prédicat à 2 arguments

$$\#(A, n) : A \text{ contient } n \text{ éléments}$$

$$\#\emptyset = 0 \rightarrow \#(\emptyset, 0)$$

— $\mathbb{P}A = \{a \mid a \subseteq A\}$ (prédicat à 2 arguments) : ensemble des sous-ensembles de A

$$\#\mathbb{P}A = 2^{\#A}$$

7.7.1 Spécification formelle avec l'approche Z

L'approche Z utilise la logique des prédicats et est basée sur la théorie des ensembles. C'est une approche "orientée modèle" : on ne donne pas que les équations, mais aussi la structure du système. Dans cette section nous

donnons une introduction à l'approche Z avec un exemple simple de gestion de stock dans une bibliothèque.

Concepts en Z

Type :

- *Types génériques* (ensembles) : par exemple : Book, Location
- *Types énumérés* : par exemple :

$$\text{Status} := \underbrace{\text{In}}_{\text{Dans la bibliothèque, disponible}} \mid \overbrace{\text{Out}}^{\text{Prêté}} \mid \underbrace{\text{Ref}}_{\text{Livre de référence}}$$

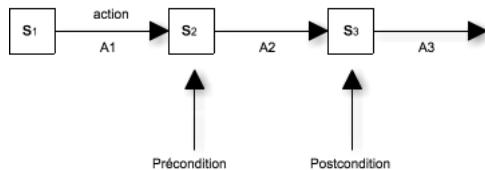
Schéma d'un état :

- *Nom* : le nom d'un état du système
- *Signature* : une énumération des variables utilisées avec leurs types
- *Prédicat* : formule en logique des prédicats qui décrit l'état du système

Schéma d'un comportement :

- *Nom* : le nom d'un comportement du système
- *Signature* : une description de la transformation faite par le comportement
- *Précondition* : formule en logique des prédicats, ce qui doit être vrai avant pour que le composant soit applicable
- *Postcondition* : formule en logique des prédicats, ce qui est vrai après l'application du composant

La définition d'un comportement utilise une sémantique et une syntaxe particulière pour décrire les transformations d'état. Une exécution est une séquence d'états et chaque application d'un composant avant l'exécution d'un pas avec un avant et un après :



Les variables avant sont écrites normalement (Lib_Stock) et les variables après sont écrites avec un apostrophe (Lib_Stock'). Il y a une syntaxe particulière pour décrire ce qui se passe avec les états avant et après :

- Δ Lib_Stock : état avant (Lib_Stock) et après (Lib_Stock')
- Ξ Lib_Stock : état avant et après s'ils sont identiques

Les définitions de ces deux symboles sont :

- $\Delta \text{Lib_Stock} \triangleq \text{Lib_Stock} \wedge \text{Lib_Stock}'$
- $\Xi \text{Lib_Stock} \triangleq \text{Lib_Stock} \wedge \text{Lib_Stock}' \mid \text{Lib_Stock} = \text{Lib_Stock}'$

Exemple d'un état

nom	LIB_STOCK
signature	stock on_loan on_shelf ref_coll : \mathbb{F} Book
prédicat	stock = on_loan \cup on_shelf on_loan \cap on_shelf = \emptyset ret_call \subseteq on_shelf

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

Exemple d'un comportement

nom	ADD_STOCK ₀
signature	$\Delta \text{Lib_Stock}$ 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 ou les préconditions ne sont pas satisfaites, on aura des erreurs que l'on devra corriger via la gestion d'erreurs. Si les préconditions n'échouent jamais, on a affaire à une opération totale.

Exemple d'opération totale

On peut définir une opération totale en ajoutant un traitement d'erreur :

$$\text{ADD_STOCK} = \text{ADD_STOCK}_0 \cup \text{ERRONEOUS_NEW_STOCK}$$

nom	ERRONEOUS_NEW_STOCK
signature	$\exists \text{ Lib_Stock}$ $\text{new?} : \# \text{ Book}$ $\text{rep!} : \text{Report}$
prédicat	$(\text{new?} = \emptyset \vee \text{new?} \cap \text{stock} \neq \emptyset) \rightarrow \text{préconditions}$ $\text{rep!} = \text{"Attention stock problem"}$

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

Chapitre 8

Introduction à la programmation logique

Depuis l'arrivée des ordinateurs, il y a eu la vision d'automatiser le raisonnement. La programmation logique est née dans le contexte de cette vision. Pour la programmation logique, l'exécution d'un programme correspond à un raisonnement logique dans une théorie logique. Elle propose l'approche suivante :

- Le programme est un *ensemble d'axiomes* en logique des prédicats.
- L'exécution du programme est fait par un *prouveur de théorèmes* (similaire à notre algorithme de preuve).
- L'exécution démarre avec une *requête*, une formule à prouver en logique des prédicats.

Est-ce que cette approche peut donner un système de programmation pratique ? C'est la grande question de la programmation logique. Une réponse affirmative à cette question pourrait réaliser plusieurs rêves :

- La vérification des programmes : si le programme est lui-même un ensemble d'axiomes, sa vérification devient facile.
- L'intelligence artificielle : si le prouveur est sophistiqué, l'exécution pourrait faire le travail d'un mathématicien.

La réponse pour le deuxième rêve est *oui, en partie* : les prouveurs de théorèmes sont devenus des assistants très utiles pour les mathématiciens, mais ils ne les remplacent pas (encore !). Mais la réponse pour le premier rêve est un *oui* très clair. On peut même dire que c'est là une des plus grandes réalisations de l'informatique au 20ème siècle. Les systèmes actuels (le langage Prolog n'est que la partie visible d'un iceberg très grand) sont complètement utilisables pour les applications pratiques : ils sont performants et robustes, avec une multitude d'outils pratiques.

8.1 La voie vers la programmation logique

Nous allons d'abord suivre le raisonnement qui a mené à l'invention de Prolog et nous allons montrer les bases théoriques de son exécution. Principalement, il y a un *compris entre expressivité et efficacité* : un langage trop expressif devient lent. Si le langage permettrait de résoudre la logique des prédictats complètement, dans certains cas l'algorithme d'exécution ne pourra rien déduire (parce qu'il est semi-décidable). Est-ce que l'on peut trouver un langage moins puissant, qui est assez expressif pour la programmation et assez *peu* expressif pour avoir une implémentation efficace ? La réponse à cette question est un oui très clair. Nous allons dans ce chapitre voir comment on obtient cette réponse. Il y a trois problèmes majeurs à résoudre :

1. *Limitations théoriques du prouveur.* Il y a des limites théoriques à ce qu'un prouveur peut faire en tant qu'algorithme. Nous avons vu une partie de ces limites avec notre algorithme de preuve : il est semi-décidable : si $p \models q$ l'algorithme termine mais si $p \not\models q$ il ne termine pas toujours. Si l'algorithme prend beaucoup de temps, on doit à un moment l'arrêter et on ne sait rien.
2. *Efficacité du prouveur.* Même si on peut trouver une preuve, le prouveur peut être inefficace (temps exponentiel). Le prouveur doit avoir une sémantique opérationnelle (= exécution) efficace et prévisible, pour être pratique.
3. *Construction du résultat.* Même si le prouveur est efficace, on doit construire une réponse et pas seulement dire que la réponse existe. Tout programme pratique calcule des résultats et ne dit pas simplement que le résultat existe.

8.1.1 Limitations théoriques du prouveur

Notre algorithme de preuve a les propriétés suivantes :

- Il est adéquat (consistent) : si $p \vdash q$ alors $p \models q$. C'est-à-dire, si l'algorithme peut prouver la requête q à partie du programme p , alors q est effectivement vrai quand p est vrai.
- Il est complet : si $p \models q$ alors $p \vdash q$. C'est-à-dire, si q est vrai quand p est vrai, alors l'algorithme peut trouver une preuve.

Où est donc le problème ? C'est le "si" dans le deuxième point : l'algorithme ne peut pas savoir si $p \models q$. Si ce n'est pas le cas, alors l'algorithme pourrait ne rien nous dire.

8.1.2 Efficacité du prouveur

Même si l'algorithme peut trouver un résultat, il est souvent hautement exponentiel. Pour pallier à ce problème on va faire deux choses :

- Mettre des restrictions sur la forme des axiomes. On ne permettra que des axiomes pour lequel l'algorithme sait raisonner efficacement. Typiquement, on ne permettra dans une clause C_i qu'un littéral sans négation :

$$C_i = (\forall X_1) \dots (\forall X_n) A_1 \wedge A_2 \wedge \dots \wedge A_n \Rightarrow A$$

En forme normale cela donne :

$$C_i = \neg A_1 \vee \neg A_2 \vee \dots \vee \neg A_n \vee A$$

Pour prouver A , il faut prouver A_1 et A_2 et ... et A_n). Le programme tout entier est donc une conjonction de clauses C_i :

$$C_1 \wedge C_2 \wedge \dots \wedge C_k$$

- Permettre au programmeur de donner des heuristiques (des “indices”) pour aider le prouveur. Typiquement, on donne de l’importance à l’ordre des clauses et l’ordre des littéraux dans les clauses. Le prouveur essaiera les clauses dans l’ordre et les littéraux dans l’ordre.

$$\text{ordre des axiomes} \left\{ \begin{array}{l} C_1 = \neg A_1 \vee \neg A_2 \vee \dots \vee \neg A_n \vee A \\ C_2 = \underbrace{\neg B_1 \vee \neg B_2 \vee \dots \vee \neg B_k \vee A}_{\text{ordre des littéraux dans un axiome}} \\ \dots \end{array} \right.$$

Le langage Prolog utilise ces deux techniques pour obtenir un langage efficace.

8.1.3 Construction du résultat

Notre algorithme de preuve construit de nouvelles clauses à partir des anciennes. La résolution utilise l’unification, ce qui instantie les variables. Pour obtenir un résultat final d’une requête, il suffit de garder la requête pendant le calcul et de l’instantier avec les mêmes substitutions utilisées par la résolution. À la fin du calcul, les variables de la requête contiendront alors le résultat.

8.1.4 Bref historique

1. 1965 : La règle de résolution est inventée par Alan Robinson. On commence à voir comment la logique peut devenir la base d’un langage de programmation.

2. 1972 : Invention du langage Prolog et implémentation de la première interprète par Alain Colmerauer, Robert Kowalski et Philippe Roussel. On démontre que Prolog est un langage utile avec lequel on peut écrire des vrais programmes avec les axiomes de logique. Ils inventent le nom Prolog qui vient de **Programmation Logique**.
3. 1977 : Premier compilateur Prolog par David H. D. Warren. Un énorme pas en avant, on démontre qu'une implémentation de Prolog peut être compétitif avec les implémentations des autres langages symboliques comme le Lisp.
4. 1990 : Premier compilateur Prolog compétitif avec C par Peter Van Roy et Andrew Taylor. On démontre qu'il n'y a rien dans le langage Prolog qui empêche une efficacité complète.

Le succès de Prolog vient du fait qu'il fait un compromis très intéressant par rapport à la tension entre efficacité et expressivité. Aujourd'hui, on peut faire une implémentation extrêmement efficace de Prolog, qui est même compétitif avec l'implémentation d'un langage de très bas niveau comme le C.

8.2 Introduction au langage Prolog

Un programme en Prolog est un ensemble de clauses (règles ; axiomes) :
En Prolog, on a des clauses (règles) :

$$A_1 \leftarrow B_1, \dots, B_n \quad (8.1)$$

L'intuition d'une clause est que l'on peut prouver A en prouvant B_1 jusqu'à B_n . En transformant à la forme normale, cette clause devient d'abord :

$$\neg(B_1 \wedge \dots \wedge B_n) \vee A_1 \quad (8.2)$$

et ensuite :

$$(\neg B_1 \vee \neg B_2 \vee \dots \vee \neg B_n \vee A_1) \quad (8.3)$$

Avant d'expliquer l'exécution de Prolog, nous allons montrer un exemple d'un programme.

8.2.1 Exemple de programme Prolog

Écrire un programme en Prolog, c'est formuler les algorithmes avec des axiomes en logique. En voici un premier exemple, extrait du livre « The Art of Prolog » par L. Sterling et E. Shapiro [Pro]. Le voici en syntaxe Prolog :

```
grandpere(X,Z) :- pere(X,Y), pere(Y,Z).
```

```

pere(terach, abraham).
pere(abraham, isaac).
pere(haram, lot).

```

Cet exemple utilise la syntaxe usuelle de Prolog introduit par David Warren. Les noms des variables sont en majuscule et le symbole \leftarrow est représenté par les deux caractères `:=`. Cette syntaxe représente la forme normale suivante :

$$\begin{aligned}
& (\forall x)(\forall y)(\forall z) \text{ grandpere}(x, z) \vee \neg \text{pere}(x, y) \vee \neg \text{pere}(y, z) \\
& \wedge \\
& \text{pere}(\text{terach}, \text{abraham}) \\
& \wedge \\
& \text{pere}(\text{abraham}, \text{isaac}) \\
& \wedge \\
& \text{pere}(\text{haram}, \text{lot})
\end{aligned} \tag{8.4}$$

En regardant ce programme, il est claire qu'il y a une correspondance entre Prolog et les bases de données relationnelles. Un programme Prolog peut être vu comme une base de données relationnelle augmentée avec la déduction. Le modèle relationnel a été inventé à peu près au même moment que Prolog par E. F. Codd. Par la suite il y a eu beaucoup d'influence mutuelle entre la programmation logique et les bases de données. En particulier, une forme simplifiée de Prolog sans symboles de fonction, appelée Datalog, est utilisée jusqu'à aujourd'hui pour modéliser la sémantique des bases de données relationnelles.

8.3 Algorithme d'exécution de Prolog

L'algorithme d'exécution de Prolog est basé sur l'algorithme de preuve de la logique des prédictats, modifié avec les techniques nécessaires pour obtenir l'efficacité et pour construire les résultats. Dans la version originale de cet algorithme, l'ensemble S grandit toujours, ce qui n'est pas très efficace. Pour augmenter l'efficacité, on va remplacer S par une seule clause r qui s'appelle la *résolvante*. Pendant l'exécution de l'algorithme, on va utiliser la résolution sur r pour la modifier. On ne va donc jamais augmenter le nombre de clauses. On ne garde que trois choses : les axiomes de base (le programme $P = \{Ax_1, \dots, Ax_n\}$), la résolvante r et la requête originale G (le “*but*” ou *goal*) qui correspond au théorème à prouver.

8.3.1 Explication de l'algorithme

L'exécution de l'algorithme se fait comme ceci :

- 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 (par exemple, A_1).
- Puis, on parcourt un à un les axiomes de P pour trouver une clause Ax_i unifiable avec A_1 au moyen de l'upg σ .
 - Si on trouve une telle clause, on ajoute à r les littéraux de Ax_i après unification avec A_1 , et on continue la boucle dès le début.
 - Si on ne trouve pas de clause unifiable, on revient sur le dernier choix. Par exemple, pour un littéral A_1 qui aurait plusieurs clauses unifiables dans P , on a dû en choisir une. On choisit la clause suivante, sans oublier d'abord de remettre r dans sa forme originale.
 - Si on épouse tous les choix sans que r soit vide alors nous sommes en cas d'échec. La requête G ne peut pas être prouvée.
- L'exécution s'arrête quand r est vide ou quand il n'y a plus de choix à faire. Si r est vide, on a un résultat qui se voit dans G .
- Il est possible aussi d'avoir une boucle infinie (l'algorithme ne s'arrête jamais). Ceci est considéré comme une erreur : le programmeur doit veiller à définir les axiomes pour que cela n'arrive pas.

8.3.2 Définition de l'algorithme

```

 $r := < G > \dots$  résolvante initiale (séquence initiale contient  $G$ )
 $\dots$  pendant l'exécution  $r = < A_1, A_2, \dots, A_m >$  (séquence de littéraux)
while  $r$  est non vide do
    — Choisir le premier littéral  $A_1$  dans  $r$ .
    — Choisir une clause  $Ax_1 = (A \leftarrow B_1, \dots, B_k)$  dans  $P$ . D'abord on prend la première clause, puis la suivante 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  $r := < B_1, \dots, B_k, A_2, \dots, A_m > \sigma$ 
    — Nouveau but  $G := G\sigma$ 
end
if  $r$  est vide then
    Le résultat est OUI. Le résultat de l'exécution est le tout dernier  $G$ . Si on le souhaite, on peut faire un retour en arrière pour que l'algorithme calcule d'autres solutions.
end
if On épouse 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
```

8.3.3 Gestion des choix

Il est important de comprendre comment cet algorithme gère les choix des clauses. L'algorithme fait parfois du retour en arrière (qui s'appelle *back-track* en anglais) pour changer un choix. Chaque fois que l'on choisit une clause à unifier avec le début de la résolvante, cela marque un endroit où l'on peut faire un retour en arrière. Pour faire le retour en arrière, il remet ses structures de données internes (r et G) à l'état avant le choix et il recommence son exécution à l'endroit du dernier choix. S'il n'y a plus de clauses possibles à ce choix, on revient à l'avant-dernier choix, et ainsi de suite. La gestion des choix est donc *récursive*. Il y a donc une *pile* de choix qui est gérée par l'algorithme. Pour simplifier la présentation, la gestion de cette pile n'est pas montrée dans la définition de l'algorithme.

Si on arrive au tout premier choix qui a été fait quand on a commencé la boucle et il n'y a plus de clauses unifiables, alors on sort de la boucle avec une résolvante qui n'est pas vide. Et dans ce cas l'algorithme n'a pas trouvé de solution. Il y a donc deux manières de sortir de la boucle : (1) r est vide, et (2) il n'y a plus de choix. La première manière correspond à une exécution réussie. La deuxième manière correspond à un échec (“failure”). Attention, cet échec fait partie de l'algorithme d'exécution, il n'y a rien d'anormal à ce que l'algorithme sort avec un échec.

Une conséquence intéressante de la gestion des choix est que l'algorithme permet à un programme de trouver plusieurs solutions, s'il existe plusieurs possibilités de solution pour G dans le programme P . Dans l'exemple `append(l, m, cons(1, nil))` plus bas, on voit comment cela marche en pratique. Pour cet exemple, on trouve une première solution avec $l = \text{nil}$ et $m = \text{cons}(1, \text{nil})$ et une deuxième solution avec $l = \text{cons}(1, \text{nil})$ et $m = \text{nil}$. Pour obtenir plusieurs solutions, il suffit après que l'algorithme ait trouvé une solution, de demander un retour en arrière, c'est-à-dire, de revenir sur le dernier choix et continuer l'exécution. Ainsi, l'algorithme trouvera plusieurs solutions si la requête G à plusieurs solutions avec le programme P . C'est une propriété très puissante de la programmation logique qui n'existe pas dans la programmation traditionnelle.

8.4 Exemples de programmes Prolog

Maintenant que nous avons vu l'algorithme d'exécution de Prolog, regardons quelques programmes pour voir comment on peut programmer en Prolog. L'idée de base est qu'un programme est un ensemble d'axiomes qui énoncent des propriétés vraies des algorithmes que l'on veut programmer, tout en permettant l'exécution de procéder de façon efficace.

L'invention de Prolog a donné naissance à un nouveau style de program-

mation. La programmation de Prolog est l'art de faire une spécification logique qui possède en même temps une exécution efficace. Dans cette section nous ne pouvons montrer qu'une toute petite partie de ce style, mais vous êtes encouragés à explorer d'autres programmes en Prolog et à télécharger un des nombreux systèmes Prolog pour les exécuter.

8.4.1 Exemple 1 : Factorielle

Ce premier exemple montre comment on peut faire des calculs avec des nombres en Prolog.

Programme en Prolog

Ce programme permet de calculer la factorielle d'un nombre. La définition est un ensemble de clauses exprimant des propriétés de la fonction factorielle. Le code commence par un fait : $0! == 1$. Ensuite, il définit 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.
```

Pour les mordus des langages de programmation, nous mentionnons que ce programme Prolog n'est pas récursif terminal. Comme pour la programmation fonctionnelle, il est avantageux aussi pour un prédictat en programmation logique d'être récursif terminal, parce que l'exécution dépend aussi d'une pile et la taille de la pile n'augmentera pas pendant l'exécution. Ceci ne se voit pas avec une interprète Prolog, comme notre algorithme d'exécution de Prolog, mais devient important pour un compilateur Prolog. Il est tout à fait possible d'écrire une autre définition de factorielle en Prolog qui est récursive terminale et qui garde la sémantique logique.

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 “`->`”. En Prolog on peut mettre des commentaires après les “`%`” ou entre “`/* */`”. Il ne faut pas oublier le point à la fin de la requête.

```
|?- fact(5,F). % Requête de l'utilisateur
-> F=120          % Réponse du système
```

Forme clausale

Voici le programme écrit en forme clausale :

$$\begin{aligned} &\text{fact}(0, 1) \\ &\wedge \\ &(\forall n)(\forall f)(\forall n_1)(\forall f_1) \quad (8.5) \\ &\text{fact}(n, f) \vee \neg(n > 0) \vee \neg\text{minus}(n_1, n, 1) \vee \\ &\neg\text{fact}(n_1, f_1) \vee \neg\text{times}(f, n, f_1) \end{aligned}$$

Les trois prédicats mathématiques font partie du système et sont prédéfinis : $a > b$ est vrai si $a > b$, $\text{minus}(a, b, c)$ est vrai si $a = b - c$, $\text{times}(a, b, c)$ est vrai si $a = b \times c$.

Exécution

Le but de l'exécution est de prouver la requête G avec $G = \text{fact}(5, r)$. Pour y arriver, Prolog construira au fur et à mesure de l'exécution une substitution finale σ_{res} en faisant les résolutions qui feront évoluer r . La substitution finale, appliquée à G , donnera le résultat final.

Requête initiale :

$$r = < \text{fact}(5, r) >$$

Résolution 1 :

$$\sigma = \{(n, 5), (f, r)\}$$

$$r = < (5 > 0), \text{minus}(n_1, 5, 1), \text{fact}(n_1, f_1), \text{times}(r, 5, f_1) >$$

Résolution 2 :

$$\sigma' = \sigma$$

$$r = < \text{minus}(n_1, 5, 1), \text{fact}(n_1, f_1), \text{times}(r, 5, f_1) > \quad (8.6)$$

Résolution 3 :

$$\sigma'' = \sigma' \cup \{(n_1, 4)\}$$

$$r = < \text{fact}(4, f_1), \text{times}(r, 5, f_1) >$$

...

$$\sigma_{res} = \{(r, 120), \dots\}$$

$$r = < >$$

Le résultat final est donc $\text{fact}(5, 120)$, ce qui donne en syntaxe Prolog `fact(5,120)`.

8.4.2 Exemple 2 : Append de deux listes

Cet exemple montre comment on peut faire des calculs avec des structures de données en Prolog. Le but de ce programme est de faire la concaténation de deux listes.

Programme en Prolog

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

Requête

La requête initiale $G = \text{append}(\text{cons}(1, \text{nil}), \text{cons}(2, \text{nil}), l)$. On veut faire la concaténation des listes [1] et [2] (en syntaxe Prolog). La paire de liste, écrite $[X|L1]$ en Prolog, s'écrit $\text{cons}(x, l_1)$ en syntaxe logique.

```
|?- append([1], [2], L).
-> L=[1,2]
```

Forme clausale

Voici le programme écrit en forme clausale :

$$\begin{aligned} &\text{append}(\text{nil}, l'', l') \\ &\wedge \\ &(\forall l_1)(\forall l_2)(\forall l_3)(\forall x) \\ &\quad \text{append}(\text{cons}(x, l_1), l_2, \text{cons}(x, l_3)) \vee \neg \text{append}(l_1, l_2, l_3) \end{aligned} \tag{8.7}$$

Exécution

La première valeur de r contient la requête initiale.

Requête initiale :

$$r = < \text{append}(\text{cons}(1, \text{nil}), \text{cons}(2, \text{nil}), l) >$$

Résolution 1 :

$$\begin{aligned} \sigma &= \{(x, 1), (l_1, \text{nil}), (l_2, \text{cons}(2, \text{nil})), (l, \text{cons}(x, l_3))\} \\ r &= < \text{append}(\text{nil}, \text{cons}(2, \text{nil}), l_3) > \end{aligned} \tag{8.8}$$

Résolution 2 :

$$\sigma_{res} = \sigma \cup \{(l', \text{cons}(2, \text{nil})), (l_3, l')\}$$

$$r = < >$$

Expliquons en plus de détail ce qui se passe lors de résolution 1 :

Il y a une unification entre le premier élément de r , c'est-à-dire $\text{append}(\text{cons}(1, \text{nil}), \text{cons}(2, \text{nil}), l)$, et la tête d'une clause (un littéral positif) dans le programme, c'est-à-dire ici $\text{append}(\text{cons}(x, l_1), l_2, \text{cons}(x, l_3))$. Cette unification donne lieu à une substitution : une liste de paires (variable, terme) nécessaire pour que les deux littéraux deviennent égaux. C'est la substitution σ . La nouvelle résolvante après la résolution 1 est le littéral négatif de la clause, c'est-à-dire $\text{append}(l_1, l_2, l_3)$, où l'on a remplacé les variables par leurs substitutions. C'est la valeur de r juste en dessous de σ .

Après la résolution 2, l'exécution s'arrête avec réussite parce que r est vide. Le résultat final est :

$$\begin{aligned}\sigma_{res} &= \{(x, 1), (l_1, \text{nil}), (l_2, \text{cons}(2, \text{nil})), (l, \text{cons}(x, l_3)), (l', \text{cons}(2, \text{nil})), (l_3, l')\} \\ G &= \text{append}(\text{cons}(1, \text{nil}), \text{cons}(2, \text{nil}), \text{cons}(1, \text{cons}(2, \text{nil})))\end{aligned}\tag{8.9}$$

Pour trouver le résultat G on a simplement appliqué σ_{res} à la requête initiale. La seule variable dans la requête initiale est l : la valeur de l est à trouver dans σ_{res} (il faut suivre les substitutions jusqu'au bout!).

Attention, chaque fois que l'on fait la résolution avec une clause du programme il faut *renommer* les variables de la clause. Dans le cas ci-haut ce sont l_1 , l_2 , l_3 et x . Ceci est nécessaire pour éviter d'utiliser le même nom pour deux variables différentes pendant l'exécution.

8.4.3 Exemple 3 : Append avec plusieurs solutions

Le programme de l'exemple précédent peut calculer plusieurs solutions, si plusieurs solutions existent et on demande à l'algorithme de nous les calculer. Pour notre dernier exemple on va regarder cela de plus près en donnant comme requête $\text{append}(l_1, l_2, \text{cons}(1, \text{nil}))$. On voit bien qu'il y a deux solutions :

- La première solution est $l_1 = \text{nil}$ et $l_2 = \text{cons}(1, \text{nil})$.
- La deuxième solution est $l_1 = \text{cons}(1, \text{nil})$ et $l_2 = \text{nil}$.

Regardons si l'algorithme les trouve aussi !

Requête

La requête initiale est $G = \text{append}(l_1, l_2, \text{cons}(1, \text{nil}))$. On la donne au système avec la syntaxe `append(L1,L2,[1])`. L'exécution du système donnera d'abord la première solution. L'utilisateur pourra demander une deuxième solution en tapant le caractère ; (point-virgule).

```
|?- append (L1,L2,[1]).  
-> L1=[] , L2=[1] ;  
-> L1=[1] , L2=[]
```

Exécution

La première valeur de r contient la requête initiale.

Requête initiale :

$$\begin{aligned} r &= < \text{append}(l_1, l_2, \text{cons}(1, \text{nil})) > \\ \text{Résolution 1 :} \\ \sigma &= \{(l_1, \text{nil}), (l_2, l'), (l', \text{cons}(1, \text{nil}))\} \\ r &= < > \end{aligned} \tag{8.10}$$

L'algorithme commence par choisir la première clause, $\text{append}(\text{nil}, l', l')$. Cela donne un résultat (obtenu en remplissant la requête avec σ) :

$$G = \text{append}(\text{nil}, \text{cons}(1, \text{nil}), \text{cons}(1, \text{nil})) \tag{8.11}$$

En syntaxe Prolog, c'est le résultat $\text{append}([], [1], [1])$. Si on demande un autre résultat, l'algorithme fera un retour en arrière pour faire un autre choix de clause. Le dernier choix fait par l'algorithme a été fait lors de la première résolution, où il a choisi la première clause. On peut re-exécuter cette résolution, en choisissant la deuxième clause. Ensuite on continue jusqu'à ce que la résolvante est vide :

Résolution 1bis :

$$\begin{aligned} \sigma &= \{(l_1, \text{cons}(x', l'_1)), (l_2, l'_2), (x', 1), (l'_3, \text{nil})\} \\ r &= < \text{append}(\text{cons}(1, l'_1), l'_2, \text{nil}) > \\ \text{Résolution 2bis :} \\ \sigma' &= \sigma \cup \{(l'', \text{cons}(1, l'_1)), (l'', l'_2)\} \\ r &= < \text{append}(l'_1, l'_2, \text{nil}) > \\ \text{Résolution 3bis :} \\ \sigma'' &= \sigma' \cup \{(l'_1, \text{nil}), (l'_2, l'''), (l''', \text{nil})\} \\ r &= < > \end{aligned} \tag{8.12}$$

Cela donne un autre résultat (obtenu en remplissant la requête avec σ'') :

$$G = \text{append}(\text{cons}(1, \text{nil}), \text{nil}, \text{cons}(1, \text{nil})) \tag{8.13}$$

En syntaxe Prolog, c'est le résultat $\text{append}([1], [], [1])$. Cela donne deux résultats qui satisfont tous les deux la requête initiale. On peut essayer de trouver un troisième résultat, en faisant un nouveau retour en arrière (c'est possible parce que 3bis a choisi la première clause), on verra que l'unification échouera : il n'y a que deux résultats pour notre requête.

8.4.4 Dernières remarques

Voici ce qui termine notre introduction à la programmation logique et au langage Prolog. Nous n'avons expliqué qu'une petite partie de ce que peut faire un système Prolog. Je voudrais terminer par attirer votre attention sur les deux manières de percevoir l'exécution d'un programme Prolog :

1. *La vue opérationnelle* : l'exécution est vue comme une séquence de calculs. On se concentre sur l'exécution de l'algorithme de preuve. Le résultat d'un calcul est une séquence d'opérations, avec ses problèmes de temps d'exécution et d'espace mémoire utilisé.
2. *La vue logique* : l'exécution est vue comme une preuve en logique des prédictats. On oublie l'algorithme et on se concentre sur la logique. Le résultat d'un calcul est un théorème, donc une conséquence logique des axiomes du programme.

Prolog est un des rares langages à offrir ces deux vues sur une même exécution. Cela donne beaucoup de puissance : on peut raisonner sur l'exactitude de façon (presque) complètement indépendante de l'efficacité.

L'équation de Kowalski

La relation entre la vue logique et la vue opérationnelle est résumée par la célèbre équation de Kowalski :

$$\text{Algorithm} = \text{Logic} + \text{Control}$$

La logique et le contrôle peuvent être vus de façon séparées. Pour une même définition logique, on peut changer l'efficacité en changeant le contrôle. Cela change l'algorithme mais ne change pas ce que calcule l'algorithme.

L'influence de Prolog

La programmation logique, le langage Prolog et ses successeurs, gardent une grande influence sur l'informatique. L'esprit de Prolog est toujours présent dans beaucoup de domaines de l'informatique, par exemple :

- Les bases de données (modèle relationnel et Datalog).
- La sémantique Web (basée sur un langage logique OWL et un modèle d'exécution).
- La programmation par contraintes (on remplace les substitutions par des relations quelconques).

Deuxième partie

Structures discrètes sur

Internet

Chapitre 9

Structures discrètes sur l'Internet

9.1 Ressources

Ressource pour cette partie du cours :

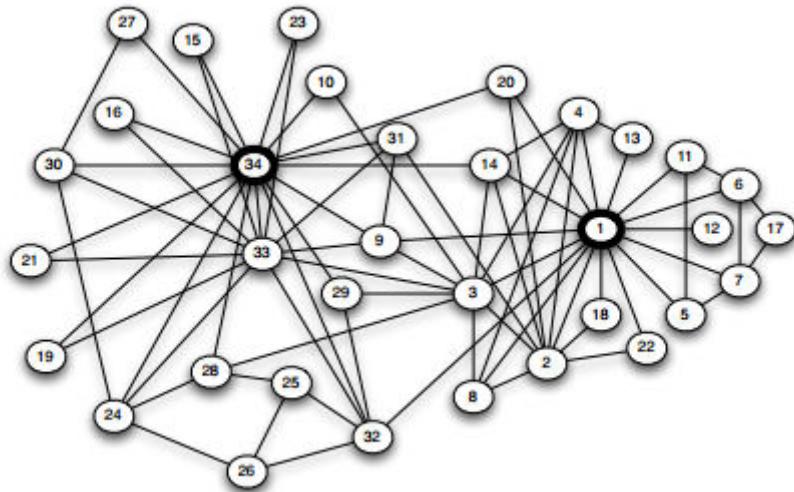
Networks, Crowds, and Markets : Reasoning About a Highly Connected World, by David Easley and Jon Kleinberg

- 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 noeud 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 (noeud) ont des liens entre eux : Lignes claires = communication e-mail.



Graphique 9.1 – Relations amicales dans un club de karaté

Lignes foncées = hiérarchie, organisation du laboratoire. On voit que la communication entre les gens suit relativement bien la structure hiérarchique, mais pas complètement. On peut voir comment les gens collaborent, leurs degrés de collaboration, etc.

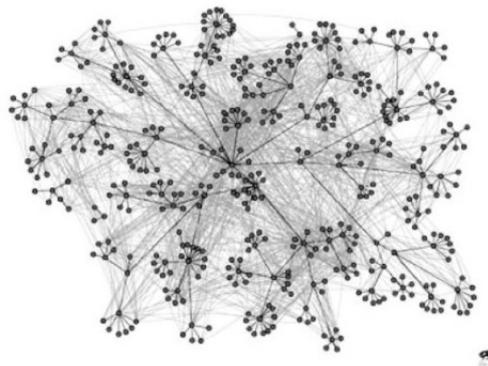


Figure 1.2. Social networks based on communication and interaction can be constructed from the traces left by online data. In this case, the pattern of e-mail communication among 436 employees of the Hewlett Packard Research Lab is superimposed on the official organizational hierarchy [6]. (Image from <http://www-personal.umich.edu/ladamic/img/hplabsemailhierarchy.jpg>, courtesy of Elsevier Science and Technology Journals.)

- **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). Les liens représentent des relations entre les banques, c'est à dire des prêts faits par une banque à une autre. Le graphe est connexe (il y a des chemins 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. Ça peut être très important.

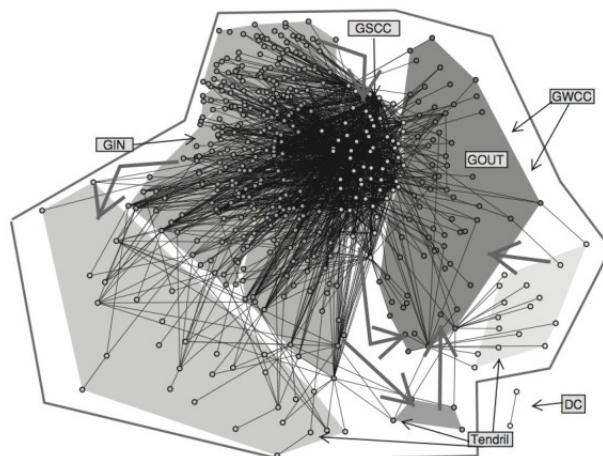


Figure 1.3. The network of loans among financial institutions can be used to analyze the roles that different participants play in the financial system and how the interactions among these roles affect the health of individual participants and the system as a whole. The network is annotated in a way that reveals its dense core, according to a scheme that we describe in Chapter 13. (Image from Bech and Atalay, [50].)

- **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 cette absence de contact entre les deux factions d'un 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. Grâce à de nouveaux outils utilisés sur internet pour analyser l'information, on peut observer la structure des graphes (ce qu'on ne pouvait pas faire avant).

9.3 Introduction

- Structure des réseaux (Facebook, Twitter, réseau économique,...).
- Comportement des participants (interactions : chaque noeud sera un participant et va interagir). En principe, chaque noeud ne voit que son voisinage et interagit en conséquence.
- Interactions *locales* avec conséquences *globales*. Il faut faire le lien entre ces deux choses.
- Effets non attendus. Par exemple les réseaux routiers : s'il y a des bouchons, on augmente la capacité du réseau (en ajoutant une voie). Le résultat peut être contre intuitif, ça peut être :
 - Une réduction des transferts.
 - Une augmentation du trafic. (résultat opposé à 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 pour prévoir la réaction à une modification plutôt que d'agir impulsivement 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 d'informations tirés de sites comme YouTube, Flickr, etc.

Synthèse de 3 disciplines :

1. Mathématiques : théorie des graphes

2. Mathématiques : théorie des jeux

Exemple : YouTube impose ses règles, c'est le croupier, et ceux qui utilisent YouTube sont des joueurs.

3. Sociologie : étude des groupes sociaux

les participants sont humains ou guidés par un humain. Il n'est pas uniquement question de mathématiques, il faut aussi comprendre les humains.

Dans ce cours, nous nous concentrerons principalement sur la théorie des graphes. La théorie des jeux sera abordée de façon intuitive, 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 : nombre fini de participants, règles simples et définies (différentes techniques sont possibles)

On va rester intuitif sur ce sujet, mais si on veut être plus précis, il y a des champs mathématiques qui étudient les systèmes concurrents.

Enfin, on peut aussi voir que la position dans le graphe peut aussi apporter des avantages.

- **Figure 1.8** Réseau d'interaction économique entre pays. Structure de l'économie mondiale : Hong Kong a un gros avantage, il a une porte d'entrée vers la Chine (à l'époque). Certains pays sont des partenaires privilégiés des États-Unis...
- **Figure 1.9** Chemins de commerces médiévaux en Europe. Une position stratégique donne des avantages. Tous ces avantages viennent de la structure du réseau (notons que le comportement d'un participant peut dépendre de la structure).

Si on observe correctement le réseau, on peut se placer en position d'avantage et se mettre à une position qui donne plus de pouvoir.

Si on connaît la structure, une petite action peut suffire pour arrêter une épidémie en coupant les liens par lesquels cette épidémie pourrait se répandre.

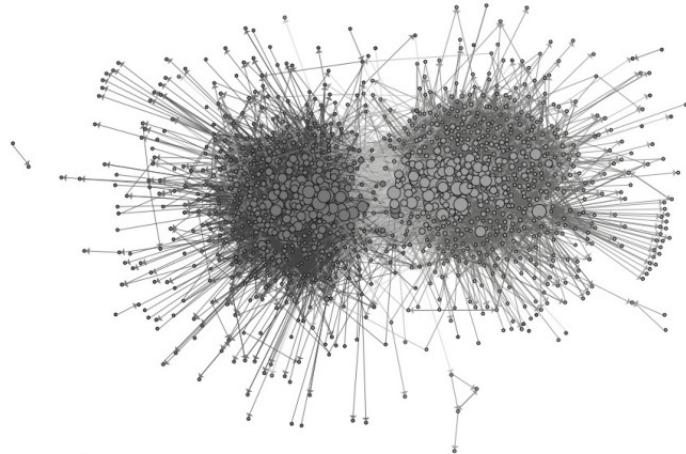


Figure 1.4. The links among Web pages can reveal densely knit communities and prominent sites. In this case, the network structure of political blogs prior to the 2004 U.S. presidential election reveals two natural and well-separated clusters [5]. (Image from Association for Computing Machinery, Inc.; <http://www-personal.umich.edu/~ladamic/img/politicalblogs.jpg>.)

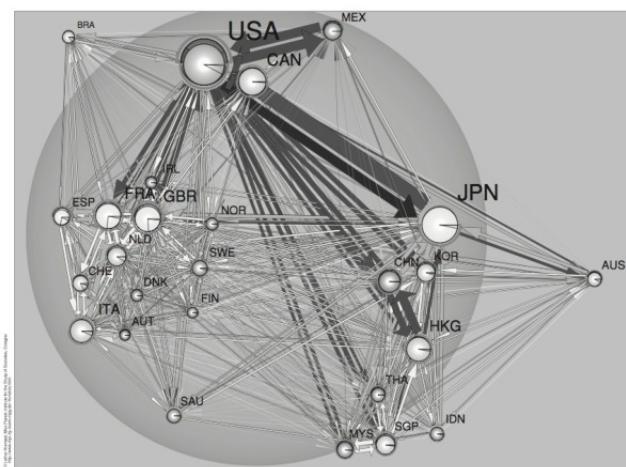


Figure 1.8. In a network representing international trade, one can look for countries that occupy powerful positions and derive economic benefits from these positions [262]. (Image from Carnegie Mellon University; <http://www.cmu.edu/joss/content/articles/volume4/KrempelPlummer.html>.)

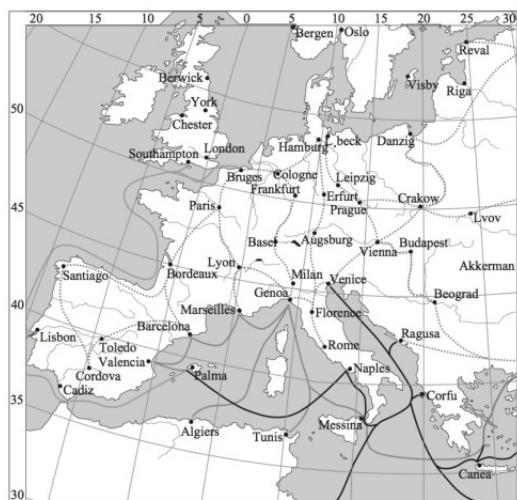


Figure 1.9. In some settings, such as this map of medieval trade routes, physical networks constrain the patterns of interaction, giving certain participants an intrinsic economic advantage based on their individual network positions. (Image from http://upload.wikimedia.org/wikipedia/commons/e/e1/Late_Medieval_Trade_Routes.jpg.)

Chapitre 10

Théorie des Graphes

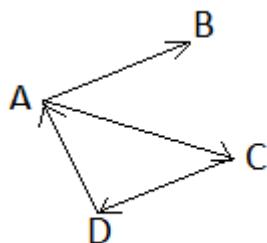
10.1 Définitions

Un graphe G est un ensemble de liens et de nœuds. Un lien est une paire de deux nœuds.

$$\begin{aligned} G &= (N, E) \\ N &= \text{nœud} \\ E &= \text{edge (lien, arête)} \end{aligned}$$

Deux types de graphes :

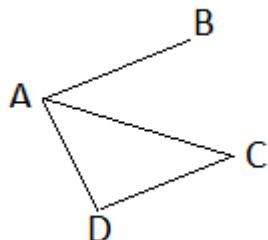
- **Les graphes orientés** (avec flèches et principe de direction). Une arête/lien est une paire de nœuds. L'ordre a de l'importance. Dans cet exemple, les paires de nœuds sont : (A,B), (A,C), (D,A) et (C,D).



Graphique 10.1 – Graphe orienté

- **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.



Graphique 10.2 – Graphe non orienté

Ce cours s'intéressera principalement aux graphes non orientés.

10.2 Chemins et connectivité

Voici la définition des différents types de chemins dans un graphe :

Chaîne : Dans un graphe non orienté, une chaîne reliant un sommet x à un sommet y est une suite finie d'arêtes consécutives, reliant x à y

Chemin : séquence de nœuds dont chaque paire consécutive est reliée par une arête.

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

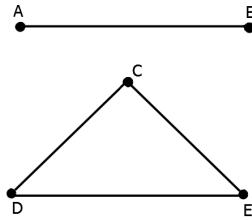
Cycle : c'est un chemin dont le premier et le dernier nœud sont les mêmes. Un cycle possède au moins 3 liens (arêtes).

Une fois la notion de chemin connue, on peut 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.

Il existe des graphes qui ne sont pas connexes. La figure 10.3 montre un exemple de graphe non connexe.

Ce graphe possède deux composants, AB et CDE . Ces composants sont les parties connexes du graphe. On ajoute donc deux définitions pour les composants.



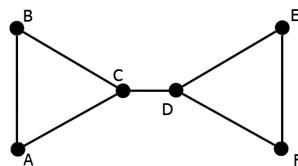
Graphique 10.3 – Graphe non connexe

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 noeuds 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 connexion avec le reste du monde. Il peut aussi y avoir un seul individu asocial qui n'a pas d'ami et forme un composant à lui seul. 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^{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 entraîné (maladies, exploitation, etc.).



Graphique 10.4 – Graphe connexe

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 10.4, 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 le graphe en

deux composants distincts.

10.3 Distance entre nœuds

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

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

Distance entre deux noeuds : le chemin le plus court entre ces deux noeuds.

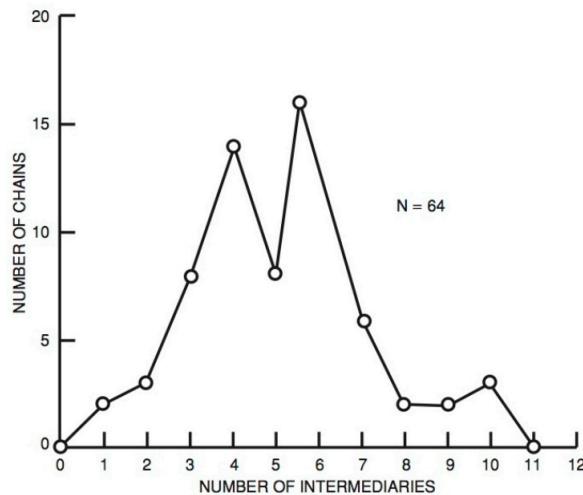
La méthode de calcul de distance entre deux noeuds est la traversée en largeur d'abord (*breadth-first traversal*). Cela consiste à débuter par un noeud, puis à regarder tous les noeuds qui sont à une distance de 1 du noeud de départ. À partir de tous ceux-là, on regarde les noeuds qui sont adjacents,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

10.4 Phénomène du petit monde

Le « phénomène du petit monde », aussi connu sous le vocable « paradoxe de Milgram » (du nom du psychosociologue *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 10.5 montre la probabilité qu'ont deux personnes d'être reliées 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 des personnes existent, comme :

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

1. xkcd.com/599/

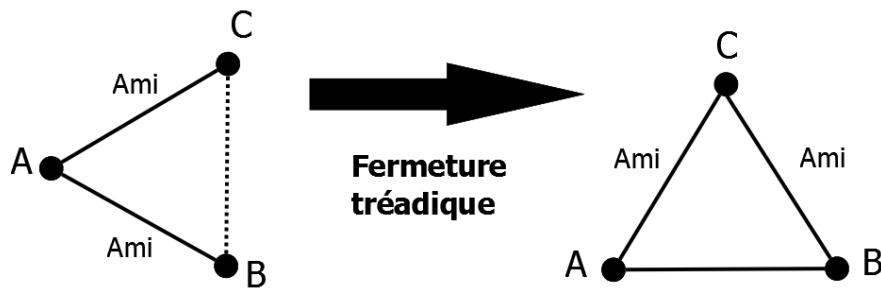


Graphique 10.5 – Statistiques du phénomène du petit monde

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

10.5 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 10.6). Dans cet exemple,



Graphique 10.6 – Exemple de fermeture triadique

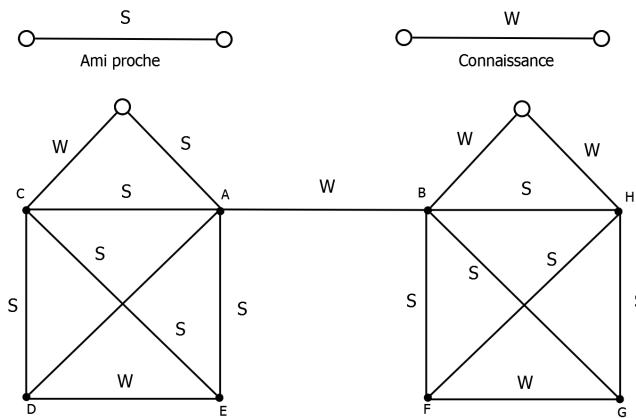
on voit que *A* est ami à la fois avec *B* et avec *C*. Dans cette condition, il est fort probable que *B* et *C* deviennent eux-mêmes amis. C'est ce qu'on appelle la fermeture triadique.

Propriété de fermeture triadique forte : si *A* a un lien fort avec *B* et avec *C*, alors un lien au moins faible va se créer entre *B* et *C*.

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 élevé.

Exemple

Une étude a été faite dans les années 1960, par le sociologue américain *Mark Granovetter*, dans laquelle il s'est intéressé 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 10.7). Nous reviendrons sur cet exemple plus tard.



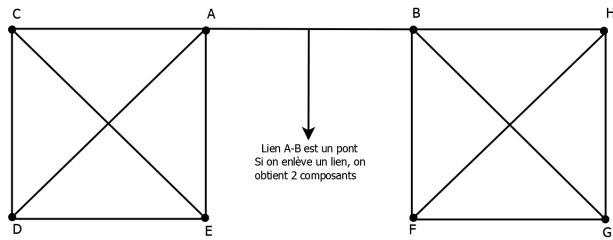
Graphique 10.7 – Liens forts et faibles

10.6 Ponts

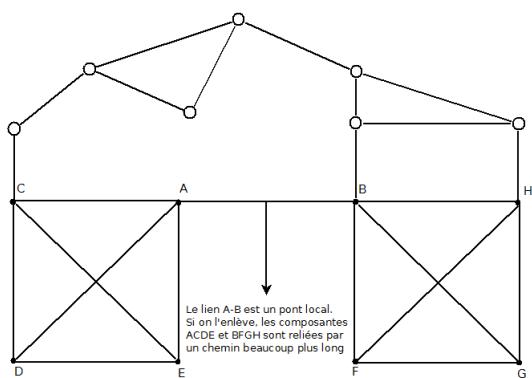
Pour expliquer l'exemple précédent, nous avons besoin de la notion de pont (figures 10.8 et 10.9).

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.



Graphique 10.8 – Pont



Reprendons l'exemple de la section 9.5, si l'on s'intéresse à la personne A, on sait que si 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 amis aussi ou tout du moins connaissance (propriété de fermeture triadique forte). On peut donc en déduire que chaque pont local est tel que le pont entre A et B dans la figure 9.9 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 dirait 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.

10.7 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.

$$\text{Chevauchement de voisinage} = \frac{\text{Nombre de noeuds voisins de A et B}}{\text{Nombre de noeuds voisins de A ou B}}.$$

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.

Force des liens en pratique dans un réseau téléphonique

Une étude portée sur les conversations cellulaires nous prouve que plus les liens entre des individus sont forts, 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.

Force des liens en pratique sur Facebook

Facebook est un réseau social développé sur internet permettant à des individus 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. Les observations montrent aussi que la propagation d'informations est plus rapide avec la 3e catégorie de lien.

Twitter

Twitter est un microblog qui permet de partager de petits messages (140 caractères 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 personnes, on a plus d'amis (jusqu'à un certain point, on suit une courbe 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 prôné sur Twitter.

10.8 Notion de trou structurel et capital social

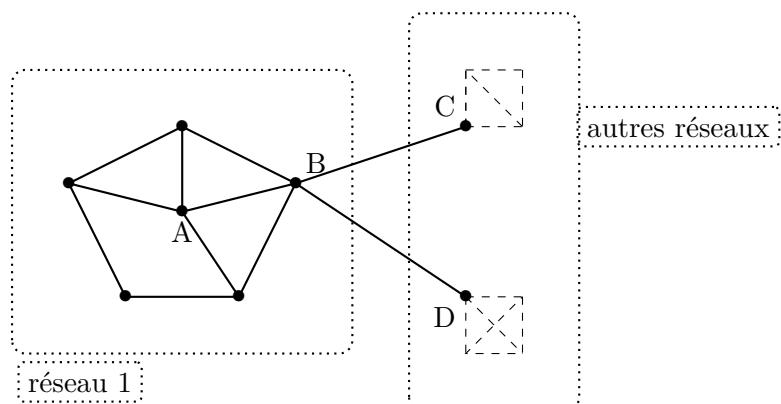
Comme nous l'avons vu précédemment, il existe deux types différents de liens. Les liens faibles et les liens forts, ces derniers étant beaucoup moins nombreux que les faibles.

Or lorsque l'on regarde un réseau, les noeuds ont aussi une grande importance. En fonction de la structure ou de l'organisation du graphe, certains noeuds posséderont quelques avantages et inconvénients. Par la suite, nous allons montrer que cette propriété propre aux noeuds est de posséder un "pouvoir".

10.8.1 L'enchâssement d'un lien (embaddness)

Attardons-nous tout d'abord à l'enchâssement d'un lien, c'est-à-dire le nombre de voisins commun entre les deux noeuds de ce lien. Par ailleurs, on peut observer que ce nombre est équivalent au dénominateur du coefficient de chevauchement. On peut dire qu'un lien est plus fortement incrusté dans le réseau si les 2 noeuds du lien possèdent un grand nombre de voisins.

Considérons le graphe 10.10 à la page 115. On peut remarquer que le noeud **A** admet quelques fermetures triadiques parmi ses voisins tandis que le noeud **B** permet au **réseau 1** de rejoindre les deux autres.

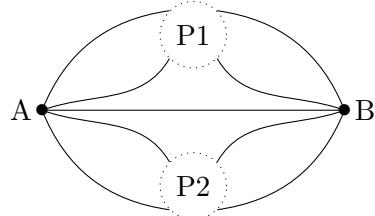


Graphique 10.10 – Un réseau central lié à deux autres sous-réseaux

On peut aussi apercevoir que le noeud **A** est localisé là où le coefficient de regroupement est assez élevé contrairement à **B** qui est dans une zone à faible densité . On dit que les liens de **A** sont 'incrustés', ils ont un enchaissement significatif.

Deux individus connectés par un lien possédant un grand enchaissement auront une plus grande confiance mutuelle. En plus de la simple force du lien, c'est également la structure du réseau qui va augmenter la confiance

entre deux noeuds. Il y aura une observation mutuelle, une surveillance de la part des amis communs aux deux noeuds. Une représentation de ce concept est donnée dans le graphe 10.11. Dans cette figure, une observation mutuelle entre **A** et **B** est mise.



Graphique 10.11 – Deux noeuds possédant un enchaînement significatif

En effet, imaginons que le noeud **A** effectue une action quelconque, tous les autres noeuds (par exemple, ceux de *P1* et *P2*) peuvent voir cette action. Ainsi, deux noeuds possédant un enchaînement suffisamment conséquent ont une certaine confiance mutuelle.

On peut donc en tirer la propriété suivante provenant directement de la structure du graphe :

Plus l'enchaînement augmente, plus la confiance mutuelle est grande.

En conséquence, **A** possède une grande influence car il a une forte confiance mutuelle avec tous ses amis.

Revenons maintenant au graphique 10.10. Bien que le lien **A** possède un grand enchaînement, il n'en est pas de même pour **B** avec **C** et **D**. En effet, ils ne peuvent pas avoir une confiance mutuelle, car les autres noeuds de leur réseau respectif ne peuvent plus les "observer".

Bien que d'un certain point de vue, **A** pourrait posséder plus d'avantages que **B** grâce à ses fermetures et son voisinage, ce n'est pas tout à fait exact. Le noeud **B** possède lui aussi des avantages aussi fondamentaux que **A**.

- **B** est au bord d'un pont local. Il joue le rôle d'intermédiaire entre plusieurs réseaux.
- **B** s'étend sur un trou structurel, il remplit le vide entre plusieurs ensembles de noeuds.
- **B** est le seul lien pour communiquer avec certains ensembles. Tous les chemins entre ces deux communautés passent par **B**. Il est donc incontournable.
- Il a des informations dispersées auxquelles les autres n'ont pas accès. Il va donc pouvoir amplifier sa créativité en observant les informations

ou en en faisant passer certaines pour siennes.²

- Il joue le rôle de gardiennage social, il peut réguler l'accès de l'extérieur.

Par ailleurs, ces quelques derniers avantages peuvent engendrer un conflit d'intérêts entre **B** et ses sous-ensembles. Par exemple, dans le cas où **B** voudrait laisser deux réseaux scindés alors que ceux-ci désireraient s'assembler.

⇒ On fait donc appel à des notions de *pouvoir* ou d'.

Ces notions sont aussi appelées *capital social* ou capacité à se procurer des avantages grâce à l'appartenance à une structure sociale.

10.8.2 Notion de capital social

Le capital social est la capacité à se procurer des avantages grâce à l'appartenance à une structure sociale. C'est une forme de capital, car elle représente une capacité que l'on possède et que l'on peut utiliser. Il existe d'autres notions de capitaux :

- Capital physique (exemple : technologie)
- Capital humain (exemple : expertise des personnes)
- Capital économique (exemple : monétaire)
- Capital culturel (exemple : les ressources accumulées dans une culture, les connaissances communes)

10.9 Similitude des noeuds

Après nous être intéressés aux liens entre les noeuds, nous allons nous concentrer sur les similitudes entre ces noeuds.

10.9.1 Principe de similitude

La notion sociologique de similitude s'observe par une augmentation du nombre de liens entre les noeuds. Exemple : Les étudiants de l'UCL présentent une similitude par le fait qu'ils font partie de la même université.

2. Le fait d'être présent sur un pont pour pouvoir espionner efficacement peut par exemple être observé dans les programmes de surveillance des la NSA (États-Unis) ou du GCHQ (Grande-Bretagne) qui profitent du pont internet entre l'Europe et l'Amérique.

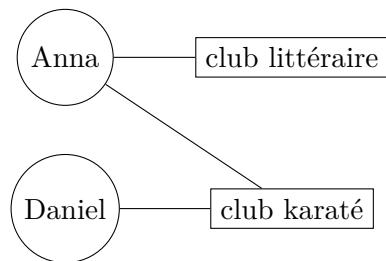
Ce principe implique de nouveaux mécanismes dans la manière où nous allons représenter les graphes :

- Sélection → *local* (Chacun choisit ses amis)
- Influence sociale → *global/extérieur* (Induite par les gens que l'on fréquente)

Exemple de la propriété d'influence sociale :

Le fait d'être dans le même auditoire qu'un autre individu peut constituer un lien dans le graphe.

Bien que la sociologie est un concept difficile à formaliser. Une idée pourrait être d'inclure les facteurs de similitudes dans le graphe.



Graphique 10.12 – Exemple d'un graphe personnes-intérêts

Les graphes sont maintenant constitués à partir de deux types de noeuds :

- noeud Individu : représente une personne physique.
- noeud Focus : activité ou centre d'intérêt.

Mais aussi de deux types de liens :

- liens Personnes - Personnes
- liens Personnes - Focus

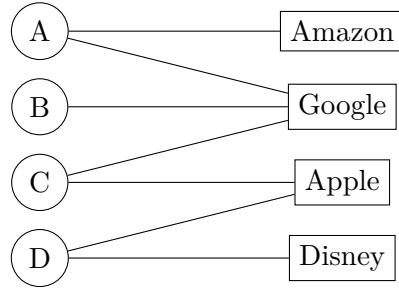
Ces graphes aussi appelés réseau d'affiliations ont une caractéristique bien particulière, ils sont toujours bipartis.

Définition *graphe biparti* : Soit un graphe $G=G(V,E)$ et V_1, V_2 deux ensembles de noeuds et $V_1 \cup V_2 = V$, un graphe est dit biparti s'il n'existe pas d'arête interne dans ces deux sous-ensembles.

$$\forall e = (u,v) \in E, u \in V_1 \text{ et } v \in V_2.$$

Le graphe ci-dessous représente l'affiliation des personnes (A, B, C, D) au conseil d'administration de différentes entreprises. Comme on peut le voir (graphe 10.13), il peut y avoir un conflit d'intérêts, par exemple entre Google et Apple. En effet, les deux compagnies sont concurrentes (sur le marché des smartphones), et pourtant une personne est membre des deux conseils d'administration.

Par ailleurs, ces graphes bipartis ne sont pas statiques. En effet, ils peuvent subir des évolutions dans le temps comme des ajouts de noeuds,



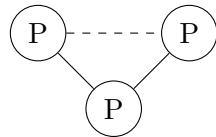
Graphique 10.13 – Exemple d'un graphe représentant un conseil d'administration

points d'intérêts, nouveaux liens ...

Ainsi, ces graphes nous fournissent plus de précisions que les précédents. Grâce à ce concept, on peut voir plus efficacement les changements et transformations lors d'évolutions.

10.9.2 Nouveaux mécanismes de fermeture

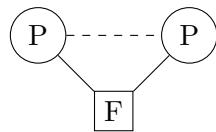
Nous avons déjà vu précédemment le principe de fermeture triadique.



Graphique 10.14 – Rappel de fermeture triadique

Grâce à la prise en compte des points d'intérêts (focus), deux nouveaux principes de fermeture peuvent être considérés :

Fermeture focale

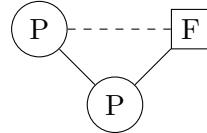


Graphique 10.15 – Exemple de fermeture focale

Comme on peut le voir sur le graphique 10.15, deux personnes ayant des similitudes ou mêmes centres d'intérêts, peuvent devenir amis.

Fermeture d'adhésion

De la même manière, une personne ayant une activité peut convertir son ami.

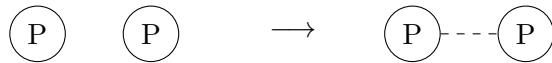


Graphique 10.16 – Exemple de fermeture d'adhésion

Ces deux nouveaux mécanismes de fermeture permettent de raisonner beaucoup plus fort sur ces graphes. Ils sont même essentiels ! Si on ne les prenait pas en compte, on ne pourrait jamais comprendre pourquoi un lien apparaît spontanément.

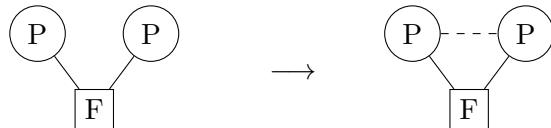
Ci-dessous, un exemple illustrant cette propriété :

Sans les concepts introduits dans cette partie, voici ce que l'on observerait :



Graphique 10.17 – Apparition spontanée d'un lien

Heureusement avec la propriété de fermeture focale, on peut raisonner sur la raison de cette apparition.



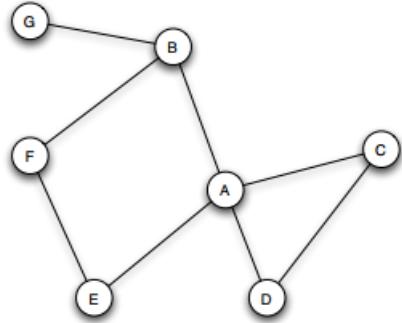
Graphique 10.18 – Apparition du lien dû au focus en commun

Coefficient de regroupement

Le coefficient de regroupement ou en anglais, *clustering coefficient*, représente la probabilité pour un nœud **A** que deux amis de celui-ci choisi aléatoirement, soit amis entre eux.

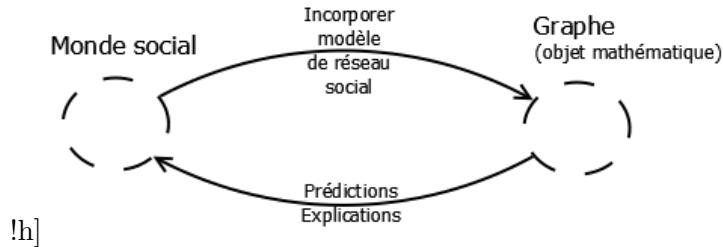
$$\text{Clustering coefficient} = \frac{\text{nombre d'arêtes existantes reliant les amis de A}}{\text{nombre total d'arêtes possible reliant les amis de A}}$$

Un exemple de ce concept est donné pour la figure suivante.



Graphique 10.19 – Graphe simple

Considérons la figure 10.19. Le coefficient de regroupement pour le nœud **A** est $\frac{1}{6}$ car il n'y a qu'une seule arête (**C-D**) parmi les six autres paires possibles (**B-C**, **B-D**, **B-E**, **C-D**, **C-E**, et **D-E**) reliées entre elles.



10.9.3 Réseaux dans leurs contextes

- réseau social : similitude entre amis (on voudrait mettre ces similitudes dans le réseau)
- réseau d'affiliation social ☺ qui permet d'expliquer les similitudes
 - personnes en contact
 - points d'intérêts communs

Idée générale

Exemples

Il y a trois formes de fermeture

1. Fermeture triadique
2. Fermeture focale
3. Fermeture d'adhésion

10.10 La formation des liens (selon les 3 approches)

Grâce à l'accès aux informations et aux outils d'analyse de réseau social, on peut observer les liens de plusieurs façons.

- On a des données réelles via les réseaux sociaux
- On a plus facilement un accès direct au graphes qu'avant.
- On peut mesurer empiriquement le taux de création des liens en fonction du nombres d'amis communs.

10.10.1 Algorithme

1. Capture à deux instants différents le réseau (appelons ces deux graphes (1) et (2))
2. Pour chaque entier "k" plus grand ou égal à 0 :

- On identifie les paires de noeuds qui ont "k" amis en communs dans (1)
3. On regarde dans (2) si pour chaque paire un lien s'est formé
 \Rightarrow On calcule $T(k) = \text{fraction des paires qui ont formé un lien}$

10.10.2 Modèle pour expliquer ce résultat (fermeture triadique)

Si on prend deux personnes : s'il y a 1 ami en commun, il y a une probabilité " p " qu'un lien se forme.

Quelle est la probabilité pour " k " amis en commun ?

La probabilité qu'aucun lien se forme quand il y a un ami en commun est de $(1 - p)$.

On en déduit que pour " k " amis en commun, la probabilité est de $(1 - p)^k$. Dès lors, la probabilité qu'au moins 1 lien se forme pour k amis en commun est de $1 - (1 - p)^k$. C'est ce qui est égal à $T(k)$.

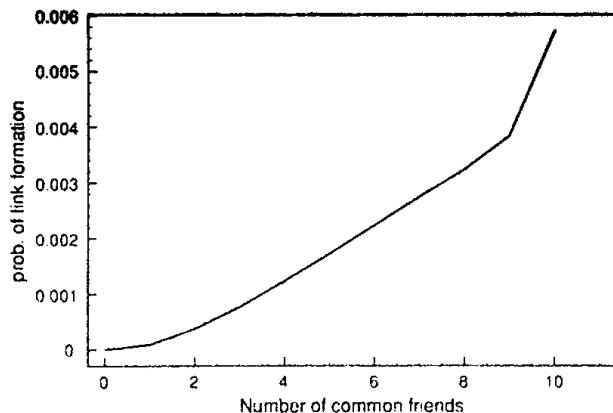


Figure 4.9. Quantifying the effects of triadic closure in an e-mail data set [259]. The curve determined from the data is shown in the solid black line; the dotted curves show a comparison to probabilities computed according to two simple baseline models in which common friends provide independent probabilities of link formation. (Image from the American Association for the Advancement of Science.)

On voit sur la figure 4.9 (fermeture triadique) que la probabilité qu'un lien se forme augmente exponentiellement avec le nombre d'amis en commun.

Attention ! Le comportement et donc les calculs sont différents pour la fermeture focale et d'adhésion !

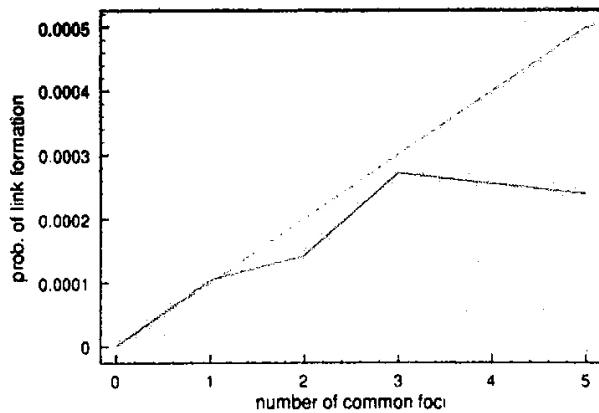


Figure 4.10. Quantifying the effects of focal closure in an e-mail data set [259]. Again, the curve determined from the data is shown as the solid black line, while the dotted curve provides a comparison to a simple baseline. (Image from the American Association for the Advancement of Science.)

La figure 4.10 (fermeture focale) nous montre que dans le cas où l'on considère le nombre d'intérêts en commun (ici, des cours), on arrive à un certains moment à saturation. Augmenter le nombre de points d'intérêts communs n'augmente plus la probabilité de création d'un lien à partir d'un certain point (ici 3 cours en communs).

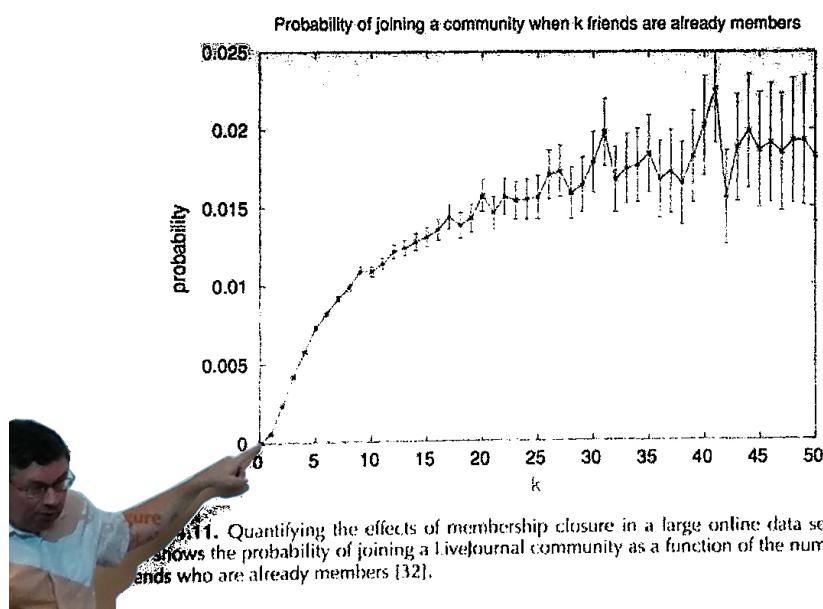


Figure 4.11. Quantifying the effects of membership closure in a large online data set: The graph shows the probability of joining a LiveJournal community as a function of the number of friends who are already members [32].

Enfin, la figure 4.11 (fermeture d'adhésion) présente aussi un saturation à partir d'un certains nombre d'amis qui ont le même point d'intérêt.

10.11 Quantifier les rôles relatifs de sélection d'influence sociale

Les mécanismes de similitude :

- La sélection (intérieur, c'est nous qui faisons le choix)
- L'influence sociale (extérieur, c'est les autres qui nous influencent)

10.11.1 Comment quantifier cela ?

Par exemple wikipédia Il peut exister une similitude de comportement entre rédacteurs. Par exemple les articles sur lesquels ils travaillent.

Raisonnement :

- On a des rédacteurs
- Un lien entre deux rédacteurs : ils communiquent par la "Talk page" (chaque article a une "Talk page"). Autrement dit, si un rédacteur B communique sur la page de A, alors il y a un lien.
- Les "points d'intérêts" ici sont les articles.
- Quantification de la similitude

$$\frac{\text{nombre d'articles rédigés par A ET B}}{\text{nombre d'articles rédigés par A OU B}}$$

Notons dès lors que la similitude ne peut pas être plus petite que 0 ni plus grand que 1. $0 \leq sim \leq 1$

- Le moment de rupture est le moment où le lien est créé.
- On compare la similitude avant et après cette rupture (figure 4.13). C'est surtout la sélection qui joue avant, car chaque rédacteur choisit ce qu'il doit ajouter ou modifier par rapport à ce qu'a mis l'autre rédacteur. Après la rupture, c'est principalement l'influence sociale qui entre en jeu car les deux rédacteurs mettent leurs idées en commun.

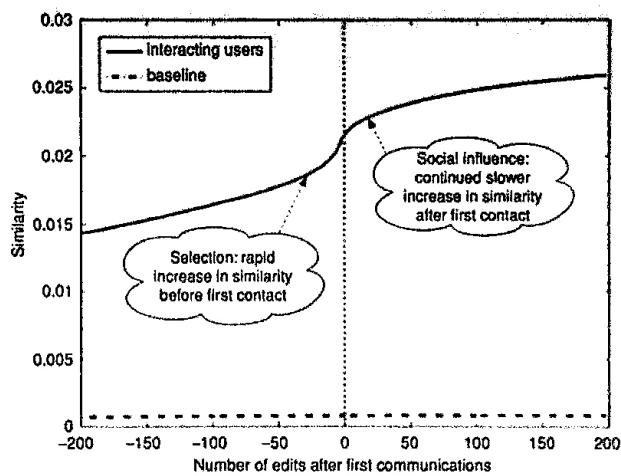
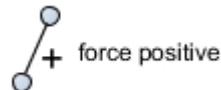


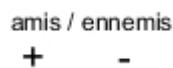
Figure 4.13. The average similarity of two editors on Wikipedia, relative to the time (0) at which they first communicated [122]. Time, on the x-axis, is measured in discrete units, where each unit corresponds to a single Wikipedia action taken by either of the two editors. The curve increases both before and after the first contact at time 0, indicating that both selection and social influence play a role; the increase in similarity is steepest just before time 0.

10.12 Les relations positives et négatives

Précédemment, les relations étudiées étaient considérées comme uniquement positives (on ne peut se faire que des amis).



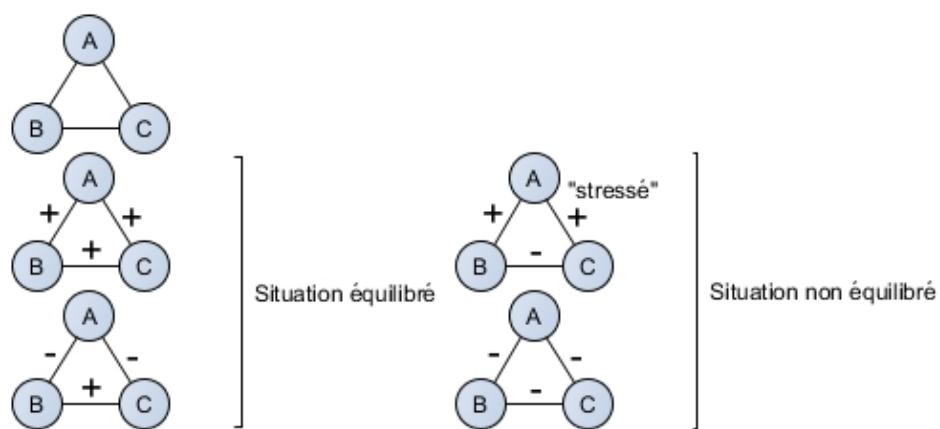
Dans ce chapitre, nous allons étudier les graphes comportant également des relations positives et des relations négatives (on peut se faire des amis et des ennemis).



10.12.1 Théorie de l'équilibre de structure (Équilibre structurel fort)

L'équilibre dans une structure va nous permettre d'identifier les situations stables et celles qui sont instables. Cette étude se fera dans un graphe complet (un graphe dont chaque paire a un lien et dont chaque lien peut être de type "+" ou "-").

L'idée cruciale est d'identifier dans le graphe les situations équilibrées et les situations qui ne le sont pas et qui engendrent un stress entre les nœuds.



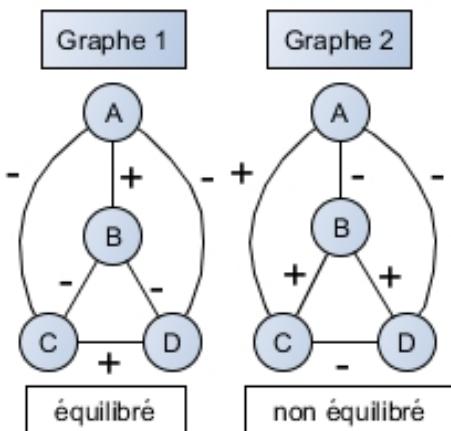
Graphique 10.20 – Relations équilibrées et non équilibrées

Dans la figure 10.12.1, les schémas de gauche sont considérés comme

équilibrés. En effet, soit A, B et C sont tous amis et il n'y aucun conflit entre eux, soit B et C sont amis, mais sont en conflit avec A.

Par contre, les schémas de droite sont des situations non équilibrées. Dans la 3, B et C sont amis avec A mais ennemis entre eux, on peut supposer que A doive alors choisir un camp. La 4 indique que tous sont ennemis, mais il est fort probable que le graphe évolue vers une situation où deux noeuds se liguent contre le troisième.

On peut maintenant généraliser cela pour un nombre quelconque de noeuds. Ainsi un graphe complet sera fortement équilibré si chaque ensemble de 3 noeuds est équilibré et donc possède des liens de type + + + ou - - -.



Graphique 10.21 – Généralisation des relations équilibrées et non équilibrées

Si un graphe n'est pas équilibré, il a tendance à s'équilibrer.

10.12.2 Caractérisation de l'équilibre structurel

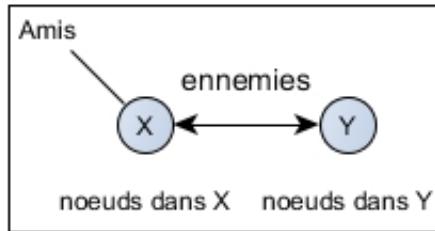
Dans le point précédent, nous avons réalisé une définition avec des conditions locales, mais il est difficile de comprendre ce que cela fait pour tout le graphe. Ainsi nous voudrions une définition globale (une caractérisation) sur l'ensemble du graphe.

Preuve :

Pour prouver qu'un graphe est bien équilibré, il faut pouvoir démontrer dans les deux sens :

1. \Rightarrow la structure est équilibrée \rightarrow direction facile à déterminer
2. \Leftarrow équilibrée est la structure \rightarrow direction plus complexe à déterminer

10.12.3 Théorème d'équilibre [Frank Harary 1953]



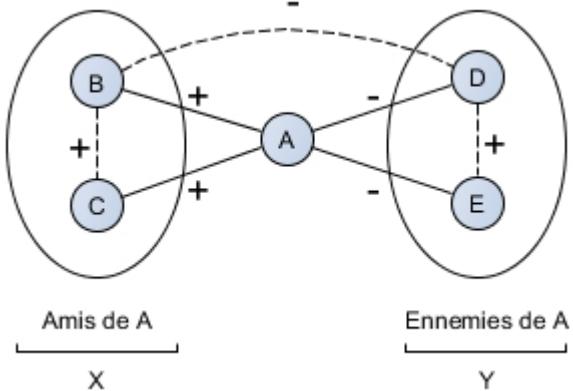
Graphique 10.22 – 2 groupes d'amis qui sont ennemis entre eux.

Si un graphe complet est équilibré, alors :

1. toutes les paires sont amies
2. on peut diviser les noeuds en deux groupes X et Y, tel que X et Y chacun contient des amis mutuels et chaque membre de X est ennemi de chaque membre de Y comme représenté à la figure : 10.12.3.

Preuve

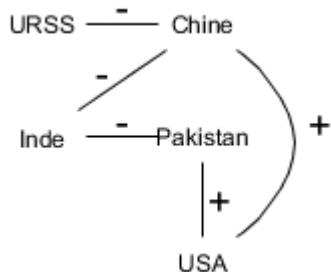
- graphe complet, annoté +, -
 - graphe équilibré
 1. si aucun lien négatif : forcément vrai car tous les triangles sont + + +
 2. sinon il existe un lien négatif
 - prenons un noeud A quelconque
 - Définissons :
 1. $X = A$ et tous ses amis
 2. $Y =$ les ennemis de A
 - Est-ce que X et Y satisfont la condition du théorème ?
- À démontrer*
1. chaque paire dans X = amis
 2. chaque paire dans Y = amis
 3. chaque noeud de X est ennemi de chaque noeud de Y



Exemple 1

On peut retrouver ce type de comportement de graphe dans les relations internationales : lors de la séparation du Bangladesh au Pakistan en 1972, on a assisté à un surprenant soutien des États-Unis pour le Pakistan alors que celui-ci n'était pas un allié des Américains.

Explication Comme indiqué dans la figure 10.12.3, les USA désiraient se rapprocher de la Chine, pour y arriver, ils ont analysé les relations des différents pays de la région. Ils ont ainsi constaté que la Chine avait des ennemis communs avec le Pakistan. Un rapprochement avec le Pakistan lui permettait de rentrer dans le groupe des amis de la Chine.



Graphique 10.23 – Relation lors de l'indépendance du Bangladesh

Une autre approche de ce conflit peut être réalisée d'un point de vue de la Chine :

- Vietnam du Nord $\xleftrightarrow{+}$ Inde
- Pakistan \longleftrightarrow pays du bloc EST
- Chine : vote l'abolition du Bangladesh à l'ONU

Exemple 2

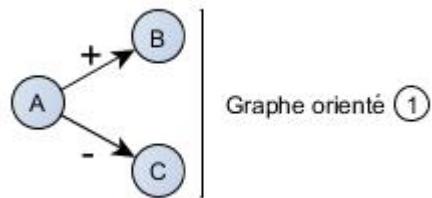
Un autre exemple de ce jeu diplomatique est donné à la figure 55 du livre de référence et représente le jeu des alliances des grandes puissances européennes à la veille de la Première Guerre mondiale.

On assiste à un équilibrage du graphe des relations avec des pays qui changent continuellement d'alliés jusqu'au moment de la triple entente et de la triple alliance qui sépara tous les pays d'Europe en deux camps ennemis.

Exemple 3

Dans ce troisième exemple, on touche directement les réseaux sociaux où il existe des relations *Like* ou *Trust* et *Dislike* ou *Distrust* dont les utilisateurs peuvent se servir pour juger le contenu des autres (par exemple Reddit, les reviews sur Amazon...)

Le classement de produits y est libre : un utilisateur est libre d'avoir des opinions positives (+) ou négatives (-) sur les autres



Transitivité

Il est intéressant de savoir si les relations sont transitives ou non, autrement dit si A croit en B et que B croit en C, A croira-t-il en C ?

$$A \xrightarrow{\text{trust}} B \xrightarrow{\text{trust}} C \xrightarrow{?} A \xrightarrow{\text{trust}} C$$

Ou alors, si A n'a pas confiance en B et que B n'a pas confiance en C, est-ce que A aura alors confiance en C ou non ?

$$\begin{aligned} A &\xrightarrow{\text{distrust}} B \xrightarrow{\text{distrust}} C \xrightarrow{?} A \xrightarrow{\text{trust}} C \\ &A \xrightarrow{\text{distrust}} C \end{aligned}$$

Finalement, tout est possible selon le cas rencontré :

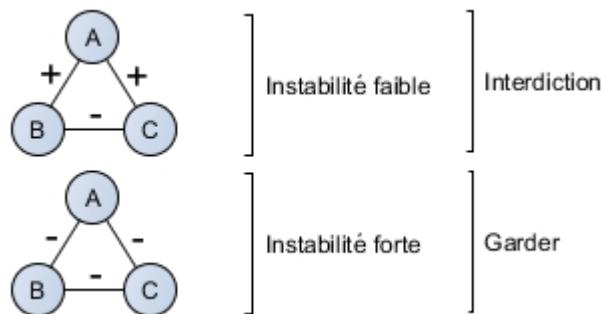
- Si le trust signifie une relation d'amitié, la transitivité sera respectée.

- Si le trust indique une confiance sur l'opinion, la transitivité ne peut être respectée.
- Si par contre trust en une proximité dans les opinions politiques, un rapprochement de A et C est probable.

Conclusion : La théorie de l'équilibre structurel sera gérée selon le cas

10.12.4 Équilibre structurel faible

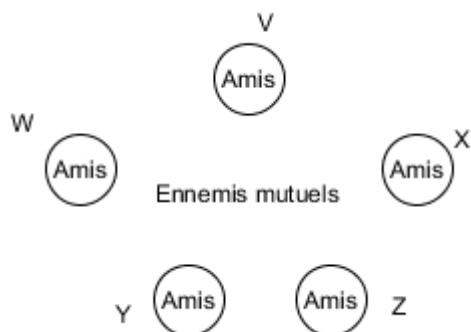
L'équilibre structurel faible, à l'instar de l'équilibre structurel fort, se base sur la notion de stabilité des triangles du graphe. Mais contrairement à l'équilibre fort, on ajoute deux cas instables



Un graphe complet annoté est faiblement équilibré si :

- aucun triplet n'est $++-$;
- tous les triplets sont de type $+++$, $+--$, $--+$.

L'équilibre faible permet une structure **multipolaire**.



Graphique 10.24 – Structure multipolaire

Théorème d'équilibre pour l'équilibre faible

Si graphe complet annoté est faiblement équilibré, alors on peut diviser ses nœuds en groupes :

- dans un groupe : amis
- autre groupe : ennemis

Preuve : (*analogue à la preuve sur l'équilibre structurel*)

1. Nœud A + tous ses amis appartiennent au groupe X
→ amis mutuels, car (+++).
2. A + ses amis sont ennemis avec tous les autres.
3. Enlever A + ses amis : nouveau graphe.
→ raisonnement récursif

Chapitre 11

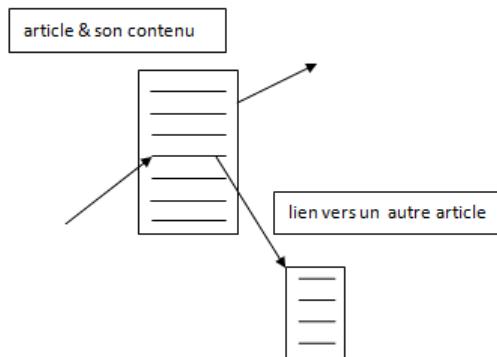
Structure du Web

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



Graphique 11.1 – Liens hypertexte vers des articles

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 *a posteriori* 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.

11.2 Le Web est un graphe orienté

- Chemin dans un graphe orienté : A → B
Séquence de noeuds 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 noeuds.

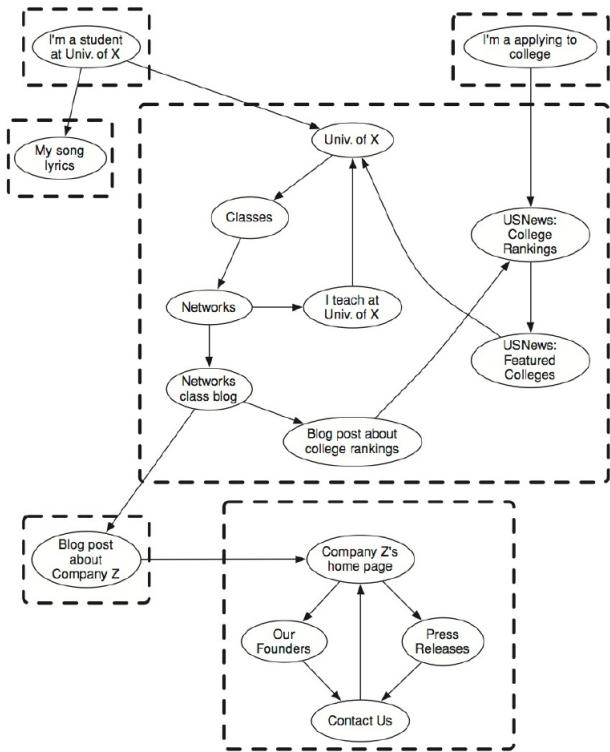
11.3 Composant fortement connexe (CFC) *Strongly Connected Component (SCC)*

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

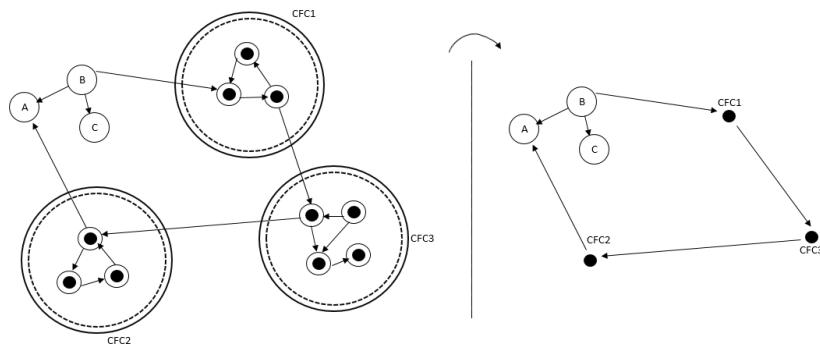
- Un ensemble de noeuds tel qu'il existe un chemin orienté entre chaque paire.
- L'ensemble ne fait pas partie d'un plus grand environnement ayant la même propriété.

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 noeud. Pour trouver qu'un chemin existe 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.



Graphique 11.2 – Un graphe dirigé avec ses CFCs identifiés



Graphique 11.3 – Exemple de transformation du graphe

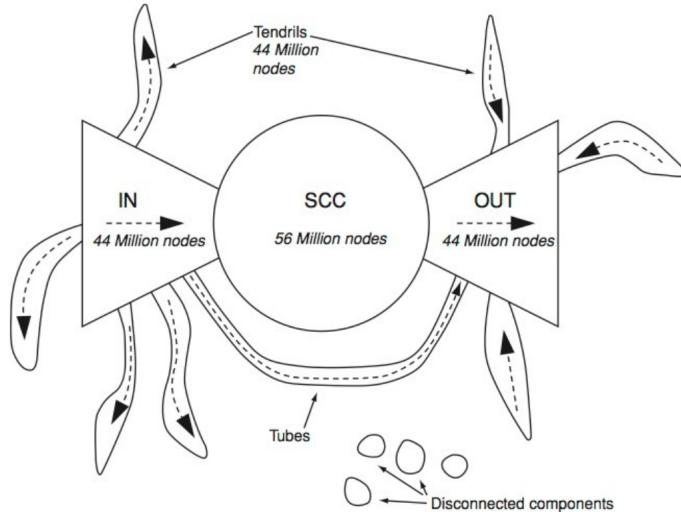
11.4 Web en nœud papillon (\approx années 2000)

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

On repère trois composants principaux :

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

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



Graphique 11.4 – Structure en nœud papillon du web

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, Flickr, 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 pré 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 écrits 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.

12.1 L'analyse des liens

- Concentrateurs (*Hubs*)
- Autorités (*Authorities*)
- Comment trouver la meilleure page ? (Voir figure 12.1)

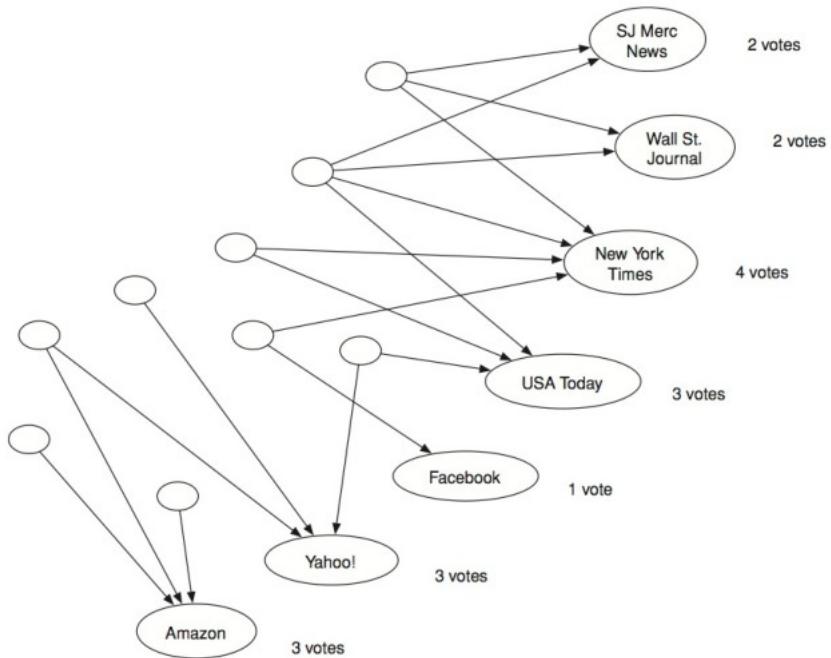


Figure 14.1. Counting in-links to pages for the query "newspapers."

Graphique 12.1 – Comptage des liens entrants pour la recherche "newspapers"

Requête "newspapers"

Pourquoi Facebook, Yahoo, Amazon, se retrouvent-ils dans la requête News Papers ? Car beaucoup d'utilisateurs ont des pages concentrées sur ces sites et comme dans cet exemple on utilise un algorithme qui n'est pas très sophistiqué, elles apparaissent.

Comment trouver la meilleure page ?

1. Compter les liens entrants comme une estimateur de la qualité d'une autorité.

2. Classer les concentrateurs en fonction de la qualité des pages qu'elles réfèrentent.
3. Mettre à jour la qualité des autorités en fonction du poids de la source des liens.
4. Mettre à jour les concentrateurs.

Algorithm

- Pages autorité (liens entrants) → auto (p)
- Pages concentrateurs (liens sortants) → conc (p)
- Mises à jour des liens entrants :

$$auto'(p) = \sum_{p' \rightarrow p} conc(p') \quad \text{avec } p' \rightarrow p \text{ les pages } p' \text{ qui ont un lien vers } P$$

- Mises à jour des liens sortants :

$$conc'(p) = \sum_{p \rightarrow p'} auto(p') \quad \text{avec } p \rightarrow p' \text{ les pages } p' \text{ qui sont référencées par } p$$

Iteration

$$\forall p \text{ in pages} : \begin{cases} auto(p) = 1 \\ conc(p) = 1 \end{cases}$$

Mise à jour, normalisation :

$$auto'(p) = \frac{auto(p)}{\sum auto(p')}$$

$$conc'(p) = \frac{conc(p)}{\sum conc(p)}$$

Cet algorithme converge

En appliquant maintenant cet algorithme, la requête "newspapers" nous donnerait les résultats suivants (12.2 et 12.3) :

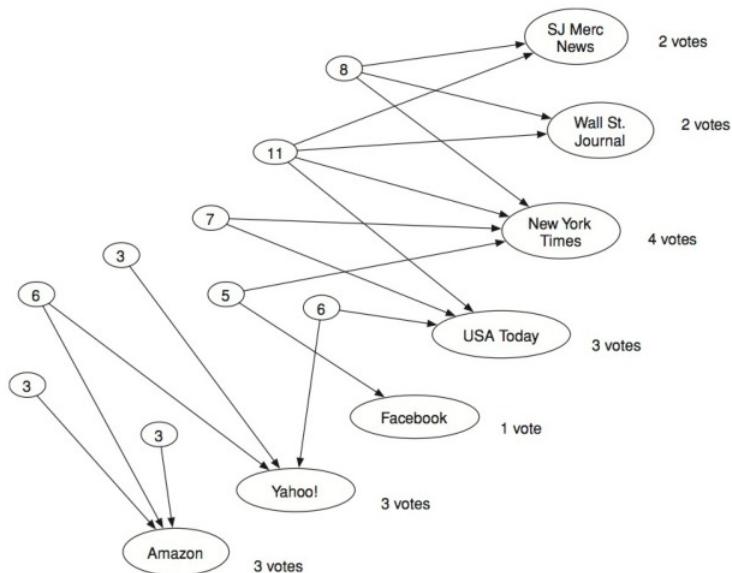


Figure 14.2. Finding good lists for the query “newspapers”: each page’s value as a list is written as a number inside it.

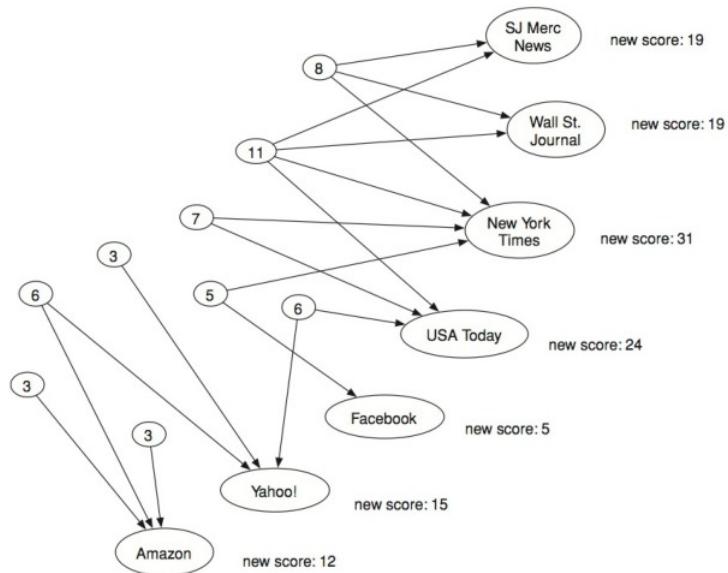


Figure 14.3. Reweighting votes for the query “newspapers”: each labeled page’s new score is equal to the sum of the values of all lists that point to it.

Graphique 12.2 – Page Rank

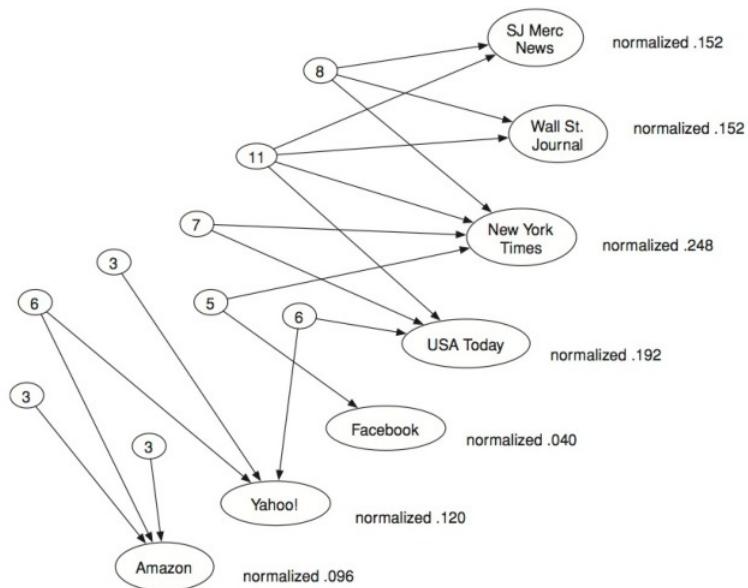


Figure 14.4. Reweighting votes after normalizing for the query "newspapers."

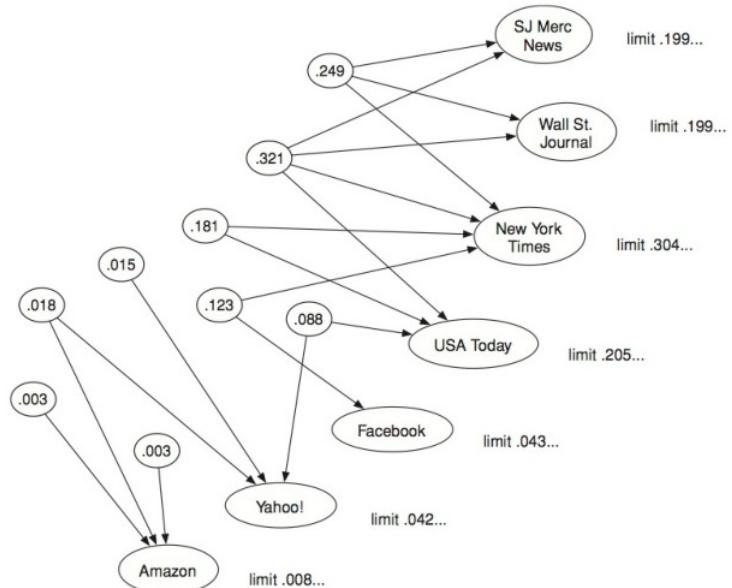


Figure 14.5. Limiting hub and authority values for the query "newspapers."

Graphique 12.3 – Page Rank

Comme nous pouvons voir, l'algorithme compte le nombre de liens entrants pour attribuer un poids à chaque place. Puis on normalise les valeurs trouvées, ce qui correspond au poids de la page. Au plus grand est le poids d'une page au plus son autorité sera grande.

12.2 PageRank

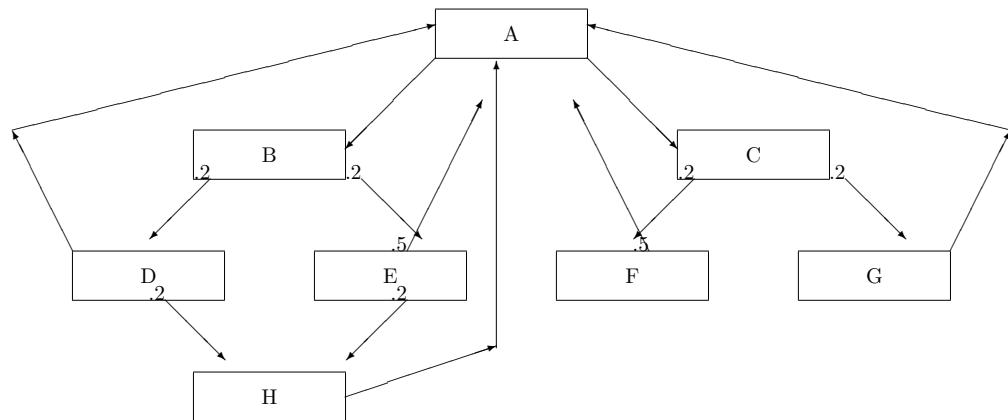
- Consolider autorités et concentrateurs.
- Une valeur par noeud : son "PageRank" que nous allons calculer.
- Intuition : Un "fluide" qui circule dans le réseau.

Algorithme PageRank :

1. N noeuds (chaque noeud représentant une page) : Initialisation $Pr(p) = \frac{1}{n}$
2. Choisir un nombre de pas k
3. K mises à jour :

$$Pr(p) = \sum_{p'} \frac{Pr(p')}{n(p')}$$
 avec $n(p')$ le nombre de liens sortant de p' et
 $Pr(p')$ le poids (ou PageRank) de p' à la $k^{\text{ème}}$ itération.

Exemple de PageRank :



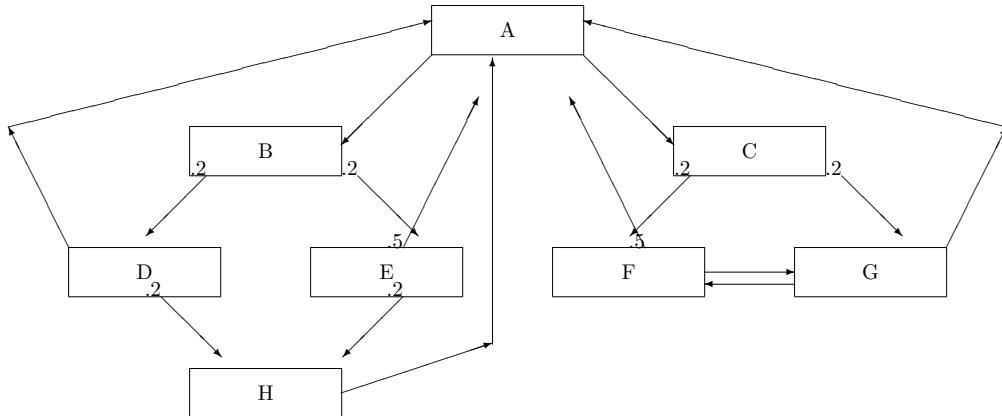
Noeuds/Itérations	0	1	2
A	1/8	1/2	5/16
B	1/8	1/16	1/4
C	1/8	1/16	1/4
D	1/8	1/16	1/32
E	1/8	1/16	1/32
F	1/8	1/16	1/32
G	1/8	1/16	1/32
H	1/8	1/8	1/16
Σ	1	1	1

Si nous continuons à laisser travailler l'algorithme, pour un nombre n fini d'itérations telles que k=n, il y aura convergence de l'algorithme. Dans cet exemple-ci, nous aurons :

$$Pr(A) = 4/13; Pr(B) = 2/13; Pr(C) = 2/13; Pr(Autres) = 1/13$$

Et la condition $\sum Pr(p) = 1$ est toujours vérifiée.

L'équilibre est vérifié si le graphe est connexe, par contre si il ne l'est pas un problème se pose : Le fluide peut arriver au mauvais noeud (analogie réseau d'eau) :



Solution du problème

La solution dans le cas où le PageRank s'amasse dans un noeuds serait de former des cycles afin d'assurer que le fluide continue à circuler.

En pratique, on va réinjecter un peu de fluide partout à chaque itération pour éviter que le fluide se concentre dans les noeuds qui n'ont que des liens entrants et pas de liens sortant. De cette façon on referme la boucle et le fluide peut continuer à circuler.

Cela peut s'apparenter à la circulation de l'eau dans l'atmosphère : le fluide s'évapore un peu partout et retombe de façon équitable.

Ancienne règle de mise à jour :

$$Pr(p) = \sum_{p'} \frac{Pr(p')}{n(p')}$$

Nouvelle règle de mise à jour :

$$Pr(p) = S * Pr(p) + (1 - S) \times \frac{1}{n}$$

Où S est un paramètre : $0 \leq S \leq 1$

Une autre manière de voir l'algorithme :

Cet algorithme correspond aussi au comportement des utilisateurs sur le web. La marche aléatoire d'un utilisateur sur le web peut être vue comme :

- Probabilité de S : Suivre un lien dans la page web ou l'on se trouve.
- (1-S) : Choisir un noeud au hasard, par exemple, taper une URL et accéder directement à un site.
- $Pr(p)$, tel que calculé dans la nouvelle règle de PageRank, est la probabilité de tomber sur la page p.

Historique de PageRank

PageRank a commencé à être utilisée au début des années 1990, mais a été en partie abandonné à partir de 2003-2004 pour bloquer les services de SEO (*Search Engine Optimisation*) et autres manipulations du système comme les Google bombs¹.

Ceux-ci consistent à altérer les résultats de recherche soit en créant de nombreuses pages contenant des liens vers une page en particulier, soit en utilisant de gros concentrateurs qui contiennent énormément de mots-clés.

Outre les modifications à l'algorithme de tri, certains sites utilisent aussi un attribut html "*nofollow*" qui a pour effet que ces liens n'entrent pas en compte dans le PageRank d'une page. C'est notamment utilisé par les sites de blogging et les sites collaboratifs comme les wikis afin que personne n'ait intérêt à saboter une page pour ajouter un lien et améliorer le classement d'un site en particulier.²

1. [wikipedia.org/wiki/Google_bomb](https://en.wikipedia.org/wiki/Google_bomb)

2. [wikipedia.org/wiki/Nofollow](https://en.wikipedia.org/wiki/Nofollow)

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*.
- [Pro] Leon Sterling and Ehud Shapiro. *The Art of Prolog*, MIT Press.