Logique et Preuves

Pierre Castéran
http ://www.labri.fr/∼casteran/L-et-P/

14 novembre 2017

Table des matières

1 Introduction **4**
- 1.1 De la logique en général . . . 4
- 1.2 La logique et l'informatique . . . 4
- 1.3 Cadre de ce cours . . . 5
  - 1.3.1 Références . . . 5
- 1.4 Divers . . . 5

I Le raisonnement logique **6**

2 Le vocabulaire de la logique **7**
- 2.1 Propositions . . . 7
  - 2.1.1 Notion intuitive et floue de proposition . . . 7
  - 2.1.2 Un cadre formel . . . 7
- 2.2 L'importance du contexte : la notation de séquent . . . 9
  - 2.2.1 Séquents . . . 10
- 2.3 Vérité et Preuve . . . 11
  - 2.3.1 Point de vue sémantique . . . 11
  - 2.3.2 Le point de vue syntaxique : notion de preuve . . . 13
  - 2.3.3 Notre point de vue (un rien subjectif) . . . 13

3 La Logique Minimale **14**
- 3.1 Les propositions . . . 14
  - 3.1.1 Remarques . . . 14
- 3.2 Approche sémantique de la logique minimale . . . 16
  - 3.2.1 Définitions . . . 16
- 3.3 Preuves en logique minimale . . . 16
  - 3.3.1 Règle d'hypothèse . . . 17
  - 3.3.2 Règle du modus ponens (élimination de l'implication) . . . 18
  - 3.3.3 Preuves en déduction naturelle . . . 19
  - 3.3.4 Règle d'abstraction (introduction de l'implication) . . . 20
- 3.4 Comment simplifier l'écriture des preuves . . . 24
  - 3.4.1 Règles dérivées . . . 24
  - 3.4.2 Règle d'affaiblissement . . . 28

3.4.3 Prouvabilité et règles dérivéees 29
3.4.4 Omission d'étapes triviales 29
3.5 Preuves *vs* validité 30
3.5.1 Correction 30
3.5.2 Incomplétude 31
3.5.3 Théorème de substitution uniforme 32
3.6 Stratégies de démonstration 33
3.7 Logique et Programmation Fonctionnelle 35
3.7.1 Types et propositions 35
3.7.2 Termes et Preuves 35

4 Logique Propositionnelle (intuitionniste, puis classique) 38
4.1 La contradiction et la négation 38
4.1.1 Introduction 38
4.1.2 Syntaxe 39
4.1.3 Aspects sémantiques de la contradiction 40
4.1.4 Le principe d'explosion 40
4.1.5 Règles associées à la négation 41
4.1.6 Définition 43
4.1.7 Peut-on prouver $\bot$? 43
4.1.8 Autour de la double négation 43
4.2 Les connecteurs binaires 44
4.2.1 La conjonction 44
4.2.2 La disjonction 45
4.2.3 L'équivalence logique 47
4.2.4 Équivalence entre propositions 48
4.3 Quelques méta-théorèmes 48
4.3.1 Cohérence et non-complétude 48
4.3.2 Relation avec la logique minimale 49
4.3.3 Remplacement d'une sous-formule 50
4.4 La logique propositionnelle classique 50
4.4.1 Règles dérivées pour la logique classique 51
4.5 Quelques méta-théorèmes pour la logique classique 51
4.5.1 Cohérence et complétude 51
4.5.2 Relation avec la logique intuitionniste 52
4.5.3 Quelques règles dérivées en logique classique 52

5 Calcul des prédicats (Logique du premier ordre) 55
5.1 Introduction 55
5.2 Formules du premier ordre 55
5.2.1 Types 55
5.2.2 Constantes et symboles de fonctions 56
5.2.3 Variables, Déclarations et Contextes 57
5.2.4 Termes 57
5.2.5 Règles de typage 58
5.2.6 Prédicats et Relations 58

5.2.7 Syntaxe des propositions 60
5.2.8 Variables libres et variables liées 60
5.2.9 Substitution 62
5.2.10 Substitution simultanée 63
5.2.11 Le coin du formaliste 63
5.3 Preuves en logique du premier ordre 63
5.3.1 Règles associéees à l'égalité 64
5.3.2 Règles associées au quantificateur universel 65
5.3.3 Notation de bloc pour l'introduction du quantificateur universel 66
5.3.4 Quantifications imbriquées 66
5.3.5 Règles associées au quantificateur existentiel 67
5.4 Pratique de la logique du premier ordre 69
5.4.1 Quantificateurs et négation 70
5.4.2 Exemples 71
5.4.3 Règles dérivées 71
5.5 Sémantique 72
5.5.1 Structures d'interprétation 72
5.5.2 Valuations 73
5.5.3 Sémantique des formules 73
5.6 Méta-théorèmes principaux 73

II Spécifier et prouver des programmes fonctionnels 75

6 Quelques types inductifs 77
6.1 Les valeurs booléennes 77
6.1.1 Règles spécifiques au type `bool` 77
6.1.2 Définitions de fonctions par cas 78
6.1.3 Exemples 78
6.2 Entiers naturels 78

III Annexes 79

7 TO DO 80

Chapitre 1

Introduction

1.1 De la logique en général

La logique est une science dont l'objet est l'étude du raisonnement, abstraction faite du domaine auquel ce raisonnement s'applique : linguistique, mathématiques, propriétés des programmes informatiques, etc. Cette discipline est très ancienne, les premiers ouvrages sur la logique ayant été écrits par Aristote (384-322 avant J.C.). On pourra lire avec profit les ouvrages de Gilles Dowek-[5] et de Jean-Pierre Belna [2].

Le côté obscur de la logique (la théorie de la mauvaise foi) est traité avec humour dans le petit livre d'Arthur Schopenhauer *L'Art d'avoir toujours raison* [7]). Trente-huit *stratagèmes* pour avoir raison dans une discussion y sont présentés. Un exercice intéressant est de voir lesquels de ces stratagèmes correspondent à des règles de raisonnement mal appliquées ou carrément dévoyées.

1.2 La logique et l'informatique

La logique, après avoir été au centre de la problématique des fondements des mathématiques [9], a pris une importance considérable avec le développement de l'informatique. En effet, la nécessité de *prouver* que tel logiciel ou composant matériel correspond bien à sa *spécification* conduit au développement d'outils informatiques permettant la vérification de programmes. On pourra trouver dans la préface de [3] un historique de ces outils.

Ces outils permettent de mener à bien des preuves qui, dans le cas de la vérification de programmes de taille courante, peuvent être très longues et complexes. Aucun être humain n'est en mesure d'en écrire et/ou en vérifier toutes le étapes.

Manipuler ces outils afin de *certifier* du logiciel ou du matériel (cartes, circuits) demande deux types de compétences :

— Savoir spécifier en langage non-ambigü (c.-à-d. mathématique) le comportement voulu du composant informatique considéré

— Savoir prouver de façon incontestable que le dit composant satisfait sa spécification.

La seconde de ces compétences ne peut être totalement automatisée dans la plupart des cas. On verra en master 1, dans le cadre de la théorie de la calculabilité, comment montrer cette impossibilité d'automatiser la preuve de programmes.

La preuve de correction d'un composant informatique peut donc exiger de la part des ingénieurs du logiciel la connaissance des règles de base du raisonnement, afin de guider l'outil en lui donnant des stratégies de démonstration (par exemple, suggérer un raisonnement par l'absurde ou par récurrence).

1.3 Cadre de ce cours

Ce cours est destiné aux étudiants de licence d'Informatique et de Mathématiques et Informatique. On privilégiera les exemples pris dans le domaine du logiciel, et plus particulièrement dans la sûreté du logiciel. Seules des connaissances de base de la programmation et de mathématiques discrètes seront nécessaires.

On étudiera des systèmes logiques de plus en plus expressifs, déterminés par les composants suivants :

— Un langage (formel) de *propositions* correspondant aux énoncés dont l'éventuelle vérité nous importe,

— Une *sémantique* permettant d'associer une *valeur de vérité* à toute proposition .

— Un langage (formel, aussi) permettant d'écrire des *preuves* de ces énoncés. On insistera sur la parenté entre langages de programmation et langages de description de preuves.

— Des *tactiques* et *stratégies* de démonstration, facilitant la recherche de la preuve d'un énoncé donné.

La plupart des séances se feront en salle de TD ; de plus, trois séances seront consacrées à la présentation et l'apprentissage de l'*assistant de preuve* *Coq*[8, 3].

1.3.1 Références

Nous conseillons la consultation régulière du site du cours de Mathématiques pour l'Informatique de Christine Paulin-Mohring [6].

1.4 Divers

Merci de me signaler par courrier électronique les (certainement nombreuses) erreurs dans ce document.

Première partie

Le raisonnement logique

Chapitre 2

Le vocabulaire de la logique

2.1 Propositions

2.1.1 Notion intuitive et floue de proposition

Il est très difficile de définir de façon générale la notion de *proposition*. On trouve souvent des descriptions genre [1]

> Une proposition est un énoncé (mathématique, informatique, mais pas que) susceptible d'être démontré ou réfuté, pour lequel il fait sens de parler de vérité.

En voici quelques exemples, pris dans des domaines divers : Certaines sont vraies, d'autres fausses, et la dernière est encore une conjecture.

— "Mon mot de passe est "password" "
— "42 est un nombre premier"
— "41 et 43 sont des nombres premiers jumeaux"
— "Le langage `C` comporte plusieurs failles de sécurité" [4]
— "*Quicksort* est un algorithme correct de tri de tableaux"
— "Le programme de recherche binaire de la figure 2.2 page suivante est incorrect"
— "La fonction de tri de la figure 2.3 page 9 est correcte"
— "La suite de Syracuse (Figure 2.1) finit toujours par le cycle $(1, 2, 4)$ quel que soit l'élément initial"

En revanche, voici des exemples de non-propositions :

— 42
— $b^2 - 4ac$
— L'algorithme du quicksort
— L'ensemble des entiers naturels

2.1.2 Un cadre formel

Pour ne pas nous disperser, la majorité des exemples que nous prendrons viendront des applications informatiques (spécification et vérification du logiciel)

initialisation On considère un nombre entier strictement positif.

calcul de l'élément suivant Soit n un élément de la suite : Par construction, il est strictement positif

- Si n est pair, l'élément suivant est $n/2$
- Sinon, l'élément suivant est $3n + 1$

Par exemple, si l'on démarre de 23, on obtient la suite suivante :

$$23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1 \ldots$$

FIGURE 2.1 – La suite de Syracuse

```
int binary_search(long t[], int n, long v) {
  int low = 0, high = n - 1;
  while (low <= high) {
    int middle = (low + high) / 2;
    if (t[middle] < v)
       low = middle + 1;
    else if (t[middle] > v)
     high = middle -1;
    else return middle;
 }
 return -1;
}
```

FIGURE 2.2 – Un programme buggé

et de mathématiques simples (inévitables en programmation).

De plus, les propositions que nous étudierons appartiendront à des *langages*, qui à l'instar du langage mathématique et des langages de programmation, disposent d'une *syntaxe* non-ambigüe. Il n'existe qu'une seule façon de lire une expression de ces langages. Comme pour les langages de programmation, des conventions (précédence et associativité des opérateurs) permettent de trouver un compromis entre lourdeur des notations et non-ambiguïté.

Il existe de nombreuses variantes de la notion de proposition, suivant les besoins à satisfaire. En voici une énumération non exhaustive.

- Logique *minimale* : permet de comprendre la notion d'implication (notée $\to$ dans ce cours).
- Logique *propositionnelle* : extension de la logique minimale par les connecteurs $\vee$, $\wedge$, $\neg$, $\leftrightarrow$, et les propositions $\bot$ (toujours fausse) et $\top$ (toujours vraie).
- Logique du premier ordre : définit les notions de prédicats, de relation (dont l'égalité), et la quantification sur des objets

```
let rec insert x l =
  match l with
    [] -> [x]
  | y::l' -> if x <= y
             then x::l
             else y::insert x l'

let rec sort l = match l with
    [] -> []
  | x::l' -> insert x (sort l')

let _ = sort [0; 2; 4; 3; 4; 5; 9]
```

FIGURE 2.3 – Un programme correct

— Logique d'ordre supérieur : on autorise la quantification sur des fonctions, des relations et des ensembles

Certaines logiques répondent à des besoins spécifiques :

— Logique de Hoare : preuve de programmes impératifs (contenant des affectations)
— Logique de séparation : extension de la logique de Hoare pour les programmes manipulant des pointeurs et des structures de données allouéees dynamiquement (comme en `C`)
— Logiques temporelles : permettent de considérer des systèmes *réactifs* et de raisonner sur des suites d'*événements*.
— Logiques épistémiques : tel processeur peut n'avoir qu'une connaissance *partielle* de l'état d'un réseau.

Chacun de ces systèmes, à l'instar des langages de programmation, est muni d'une syntaxe précise et non ambigüe, que nous présenterons pour chaque logique étudiée dans ce cours.

2.2 L'importance du contexte : la notation de séquent

Une proposition comme "$x = 2$", prise isolément, ne peut être considérée en absolu comme vraie ou fausse. Tout dépend en effet de ce qu'on sait ou suppose sur x. : Est-ce un entier ? un nombre réel ? la valeur courante d'une variable d'un programme `C` ?

Dès qu'on veut être rigoureux, on se doit de rendre *explicite* le *contexte* dans lequel on se place. Cela consiste à préciser dans quel univers (nombres, rela-

tions, programmes, etc.) on se place, et quelles propriétés sont admises (analyse mathématique, théorie des ensembles, spécification d'un langage de programmation, etc.), mais aussi des hypothèses de travail spécifiques à une situation donnée (tel tableau est trié jusqu'à l'indice i, par exemple).

2.2.1 Séquents

Définition

Un *séquent* est une structure composée :
— d'un *contexte* formé d'un ensemble Γ de propositions appelées *prémisses* ou *hypothèses*,
— d'une proposition A appelée *conclusion* du séquent
Un tel séquent se note $\Gamma \vdash A$.

Il est d'usage d'écrire les hypothèses séparées par des virgules, et sans accolades, sous la forme $A_1, \ldots, A_n \vdash A$ où les A_i sont les prémisses et A la *conclusion*.

Un tel séquent se lit de façon indifférente d'une des trois façons suivantes :
— "sous les hypothèses $A_1, \ldots, A_n$, la conclusion A est vraie",
— "La proposition A est une conséquence logique de $A_1, A_2, \ldots, A_n$ ",
— Si $A_1, \ldots,$ et A_n sont vraies, alors A est vraie".

La notion de "validité" d'un séquent sera prise de façon intuitive dans ces exemples, et précisée dans les chapitres suivants.

Exemples

Voici quelques exemples de séquents, plaçant chacun la proposition $x = 2$ dans un contexte différent :

— Le premier séquent affirme que si x est un nombre réel vérifiant l'égalité $2x^2 - 5x + 2 = 0$, alors $x = 2$:

$$x \in \mathbb{R},\ 2x^2 - 5x + 2 = 0 \vdash x = 2$$

Intuitivement, ce séquent ne peut être tenu pour « vrai ». En effet, si l'on prend pour x la valeur $1/2$, les hypothèses de ce séquent sont vérifiées, alors que la conclusion est fausse.

— En revanche, en *affaiblissant* la conclusion, on obtient un jugement valide :

$$x \in \mathbb{R},\ 2x^2 - 5x + 2 = 0 \vdash x = 2 \lor x = 1/2$$

— Si une hypothèse contraint x à être un entier, nous avons le séquent valide suivant.

$$x \in \mathbb{Z},\ 2x^2 - 5x + 2 = 0 \vdash x = 2$$

— Finalement, nous terminons par deux exemples de séquents valides *parce que leurs hypothèses sont incohérentes*.

$$x = 2,\ x = 3 \vdash x = 42$$

$$x \in \mathbb{R},\ x^2 + x + 1 = 0 \vdash x = 42$$

Notations diverses

Traditionnellement, nous utiliserons la méta-variable Γ pour désigner un contexte arbitraire. L'adjonction d'une hypothèse A à un contexte Γ sera notée Γ, A au lieu du plus lourd $\Gamma \cup \{A\}$. L'union de deux contextes Γ et Δ sera notée simplement Γ, Δ.

Si l'ensemble des hypothèses est vide, on notera simplement $\vdash A$ le séquent $\emptyset \vdash A$.

Remarque

Un séquent sans prémisse, de la forme $\vdash A$ se lit "La proposition A est vraie" (sans condition).

Prononciation : Le symbole $\vdash$ se prononce "thèse" en français. Sa ressemblance avec un tourniquet vu de haut le fait appeler « turnstyle » en anglais. Certaines régions vinicoles y voient une forme abstraite de tire-bouchon.

Remarque

Dans le calcul des séquents dit *classique*, on peut avoir un nombre quelconque de propositions à droite du symbole $\vdash$. Nous n'étudierons pas ce calcul dans ce cours, car il s'agit d'une formalisation de la logique très différente.

2.3 Vérité et Preuve

Nous voulons donner un sens précis à la notion de "correction" d'un séquent, juste abordée de façon informelle dans les exemples de la section 2.2.1 page précédente. Nous présentons deux façons de voir.

2.3.1 Point de vue sémantique

Le cas des propositions

Il est communément admis qu'une proposition est vraie ou fausse. La "sémantique" d'une logique précise comment déterminer la "valeur de vérité" d'une proposition. Cette valeur dépend en général d'une *valuation*, c'est à dire une fonction qui attribue une valeur de vérité aux formules les plus élémentaires (*e.g.* les formules atomiques de la logique propositionnelle) à partir desquelles on calcule la valeur de vérité de n'importe quelle proposition (voir par exemple la section 3.2 page 16).

Dans le cas de la logique propositionnelle, des méthodes de calcul comme les tables de vérité ou d'autres méthodes plus sophistiquées (diagrammes binaires de décision) permettent de calculer cette sémantique de façon automatique.

Le cas des séquents

Considérons un séquent $A_1, \ldots, A_n \vdash A$. Nous dirons qu'il est *valide* si, chaque fois que *toutes* les hypothèses sont vraies, alors la conclusion A est vraie. On notera $A_1, \ldots, A_n \vDash A$ la propriété « le séquent $A_1, \ldots, A_n \vdash A$ est valide ».

Il faut bien distinguer un *jugement* de la forme $\Gamma \vDash A$ ou sa négation $\Gamma \nvDash A$ du séquent (objet purement syntaxique) $\Gamma \vdash A$ dont les jugements ci-dessus portent sur la validité.

Ainsi, on a tout à fait le droit d'écrire le séquent $x = 2 \vdash x = 3$, bien qu'on ne puisse pas énoncer le jugement $x = 2 \vDash x = 3$.

Remarques

1. Soit un séquent sans hypothèses, c'est à dire de la forme $\vdash A$. Alors ce séquent est valide (c'est à dire $\vDash A$) si et seulement si A est une tautologie.
2. Pour montrer qu'un séquent n'est pas valide, il suffit de donner une valuation telle que toutes les hypothèses sont vraies et la conclusion fausse
3. Soit un séquent tel qu'il n'existe aucune valuation rendant *toutes* ses hypothèses vraies. Alors ce séquent est valide. *Autre formulation équivalente : pour toute valuation ν, il existe au moins une hypothèse fausse pour cette valuation*. C'est le cas du séquent $x{=}2,\ x{=}3 \vdash x{=}42$.

Limites de l'approche sémantique

Cette méthode fonctionne bien dans le cas de la logique propositionnelle : tables de vérité, diagrammes de décision binaire, etc.

Elle présente cependant quelques inconvénients :

— Elle est plus adaptée au calcul automatique qu'à la compréhension intuitive des énoncés, et du *sens* des formules. Qui peut lire une table de vérité sur plus de 5 variables propositionnelles ?

— En dehors de la logique propositionnelle, la détermination automatique de la valeur de vérité d'une proposition peut être soit difficile, soit impossible.

— Les algorithmes de calcul de la valeur de vérité d'une proposition peuvent utiliser des représentations qui risquent de masquer l'intuition d'un énoncé.

— La définition de la sémantique des connecteurs et quantificateurs montre une "circularité" déroutante. Par exemple :

  — "La proposition $A \wedge B$ est vraie si la proposition A et la proposition B sont vraies"

  — "La proposition $\forall\, x, P(x)$ est vraie si pour toute valeur t, la proposition $P(t)$ est vraie"

— Il est difficile de bien saisir le *sens* de l'implication $\rightarrow$. Dire que $A \rightarrow B$ est une abréviation de $\neg A \vee B$ n'aide en rien à comprendre la relation de causalité liée à ce connecteur.

— Finalement, rappelons la présence de propositions dont on ne sait encore si elles sont vraies ou fausses. Exemple : la conjecture sur la suite de Syracuse.

2.3.2 Le point de vue syntaxique : notion de preuve

En complément de la vue sémantique précédente (liée au nom de Tarski), la *théorie de la démonstration* consiste à définir des objets appelées *preuves*. Étant donné un énoncé logique, on définira quelles sont ses preuves éventuelles. On remplace donc la question « Cet énoncé est-il vrai ? » par « Comment démontrer cet énoncé ? ».

Comme pour un langage de programmation où la notion de "programme bien écrit" est définie, nous allons définir quelles sont les preuves correctes d'un séquent. Les règles de construction de ces preuves donneront un sens aux connecteurs, quantificateurs, etc. que nous espérons aussi intuitif que les méthodes sémantiques.

Il existe plusieurs notions de preuves en logique, notamment le *calcul des séquents* de Gentzen, les *Systèmes à la Hilbert*, la *déduction naturelle*. C'est cette dernière que nous allons utiliser, car très proche de la notion intuitive de raisonnement.

Quelques remarques :

— Ce n'est pas parce qu'on n'a pas pu démontrer un énoncé qu'il est faux.
— L'écriture de preuves correctes est très proche de l'écriture de programmes informatiques. Les informaticien[ne]s devront se sentir en terrain familier.
— Contrairement aux tables de vérité et autres techniques similaires, l'approche par la notion de preuve est tout à fait adaptée aux logiques à fort pouvoir d'expression (premier ordre, ordre supérieur, etc.)

2.3.3 Notre point de vue (un rien subjectif)

Il n'est pas question de prendre une position affirmée entre ces deux faces de la logique. Ce cours utilisera plutôt la notion de démonstration pour les raisons suivantes :

— Il est important de reconnaître et écrire des argumentations convaincantes
— La similitude entre la structure des démonstrations et celle des programmes informatiques rend ce choix bien adapté à un public informaticien.

Chapitre 3

La Logique Minimale

Afin d'aider à mieux comprendre le sens des propositions logiques, des notions de démonstration, d'hypothèse, de théorème, etc., nous commençons par une logique très réduite et très abstraite, appelée *logique minimale*, et ne contenant qu'un connecteur, l'implication.

Les logiques propositionnelle, du premier ordre, etc. seront des extensions de la logique minimale où l'on introduira des connecteurs, quantificateurs, etc.

3.1 Les propositions

La description d'une logique commence par définir quels énoncés seront manipulés, et comment les écrire sans ambiguïté.

On considère un ensemble arbitraire V_P de *variables propositionelles* souvent notées P, Q, etc.

L'ensemble des propositions sur V_P (ou plus simplement *propositions*) est défini de façon *inductive* par :

— Toute variable propositionnelle est une proposition (appelée également *proposition atomique* ou *atome*)

— si A et B sont deux propositions, alors l'implication $(A \to B)$ est une proposition

sous-entendu : Il n'y a pas d'autres propositions

On utilisera également le terme "formule" pour désigner une proposition.

La proposition $(A \to B)$ se lit « A implique B », ou « si A, alors B ».

Intuitivement, elle « signifie » que, si l'on peut prouver A, alors on peut aussi prouver B. Cette signification intuitive sera précisée plus loin sous forme de *règles de déduction*.

3.1.1 Remarques

1. Par convention, les méta-variables P, Q, R, etc., seront utilisées pour désigner des variables propositionnelles et A, B, C, etc, des propositions

quelconques.

2. Le fait d'utiliser, dans ce chapitre et le suivant, des noms de propositions "abstraits" nous permet à la fois de souligner l'universalité des schémas de raisonnements présentés, et de travailler avec des formules plus courtes et donc plus lisibles.

   Rien n'interdit au lecteur de prendre des exemples comme socrate_est_mortel ou mon_programme_boucle, mais ces exemples seront bien mieux traités dans le cadre de la logique des prédicats.

3. On trouve fréquemment la notation $A \Rightarrow B$ pour l'implication. Nous avons choisi la simple flèche $\to$ par compatibilité avec le logiciel *Coq*.

4. Les propositions $((A \to B) \to C)$ et $(A \to (B \to C))$ sont *différentes*. La structure de la première est représentée en figure 3.1, et celle de la seconde en figure 3.2.

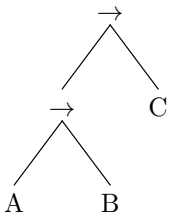

FIGURE 3.1 – structure de la proposition $((A \to B) \to C)$

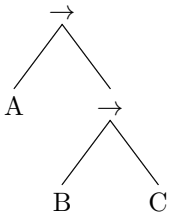

FIGURE 3.2 – structure de la proposition $(A \to (B \to C))$

5. On peut enlever des parenthèses inutiles en considérant que l'opérateur $\to$ est *associatif à droite* (aussi dit *de droite à gauche*), c'est à dire que la formule $A \to B \to C$ est interprétée comme $A \to (B \to C)$ et non comme $(A \to B) \to C$.

Exercice 1. *Dessiner la structure des formules suivantes :*

1. $P \to (P \to Q) \to Q$
2. $(P \to Q) \to (Q \to R) \to P \to R$
3. $(P \to Q \to R) \to Q \to P \to R$

3.2 Approche sémantique de la logique minimale

On considère toujours les propositions et séquents que l'on peut former à partir d'un ensemble V_P de variables propositionnelles. Dans la plupart de nos exemples, nous prendrons $V_p = \{P, Q, R, S, T, U\}$

3.2.1 Définitions

1. On considère l'ensemble $\mathbb{B}$ des *valeurs booléennes*. $\mathbb{B}$ ne contient que deux éléments : $\boldsymbol{v}$ ("vrai") et $\boldsymbol{f}$ ("faux"). *On évitera l'assimilation de $\boldsymbol{f}$ au nombre 0 et de $\boldsymbol{v}$ à 1.*
2. Une *valuation* est une fonction ν qui associe à toute variable propositionnelle une valeur dans $\mathbb{B} = \{\boldsymbol{v}, \boldsymbol{f}\}$. Par exemple, il y a $2^6 = 64$ valuations différentes sur l'ensemble V_p ci-dessus,
3. Soit ν une valuation. On l'étend à l'ensemble de toutes les propositions en posant :

$$\nu((A \to B) = \tag{3.1}$$
$$\boldsymbol{v} \text{ si } \nu(A) = \boldsymbol{f} \tag{3.2}$$
$$\nu(B) \text{ si } \nu(A) = \boldsymbol{v} \tag{3.3}$$

4. Une valuation ν *satisfait* un ensemble Γ de propositions si $\nu(A) = \boldsymbol{v}$ pour toute formule A de Γ. Notation : $\nu \models \Gamma$.
5. Un ensemble Γ de propositions est *satisfaisable* s'il existe une valuation ν qui satisfait $\nu \models \Gamma$.
6. Un séquent $A_1, A_2, \ldots A_n \vdash A$ est *valide* si pour toute valuation ν telle que $\nu(A_1) = \nu(A_2) = \cdots = \nu(A_n) = \boldsymbol{v}$, alors $\nu(A) = \boldsymbol{v}$. On exprime par le jugement $\Gamma \vDash A$ la validité du séquent $\Gamma \vdash A$.
7. Soit A une proposition. On dit que A est une *tautologie* si $\vDash A$.

La table de vérité de la figure 3.3 page suivante montre que le séquent $P \to Q, Q \to R \vdash P \to R$ est valide. En effet, toutes les valuations qui rendent vraies à la fois $P \to Q$ et $Q \to R$ rendent vraies la conclusion $P \to R$.

En revanche, le séquent $P \to Q, Q \to R \vdash R \to P$ n'est pas valide (voir Figure 3.4 page suivante, 2ème et 4éme lignes).

3.3 Preuves en logique minimale

Dans cette section, nous définissons quelles sont les éventuelles *preuves* d'un séquent, dans le cadre de la logique minimale.

Rappelons que le but est l'obtention d'une certitude que dans un séquent $\Gamma \vdash A$, A peut se déduire des hypothèses de Γ.

On définit une preuve de $\Gamma \vdash A$ comme une suite de séquents dont chaque élément s'obtient des précédents en appliquant une *règle d'inférence* parmi les

P	Q	R	$P \to Q$	$Q \to R$	$P \to R$
f	f	f	v	v	v
f	f	v	v	v	v
f	v	f	v	f	v
f	v	v	v	v	v
v	f	f	f	v	f
v	f	v	f	v	v
v	v	f	v	f	f
v	v	v	v	v	v

FIGURE 3.3 – Table de vérité du séquent $P \to Q, Q \to R \vdash P \to R$

P	Q	R	$P \to Q$	$Q \to R$	$R \to P$
f	f	f	v	v	v
f	f	v	v	v	f
f	v	f	v	f	v
f	v	v	v	v	f
v	f	f	f	v	v
v	f	v	f	v	v
v	v	f	v	f	v
v	v	v	v	v	v

FIGURE 3.4 – Table de vérité du séquent $P \to Q,\ Q \to R \vdash R \to P$

trois règles de la logique minimale, définies ci-dessous et dont le dernier élément est le séquent $\Gamma \vdash A$.

Nous donnerons quelques exemples de preuve au fur et à mesure de la présentation des règles de la logique minimale.

3.3.1 Règle d'hypothèse

La *règle d'hypothèse*[1], comme son nom l'indique, précise le rôle joué par les hypothèses d'un séquent : Si une proposition apparait parmi les hypothèses d'un séquent, on considère qu'elle est bien une conséquence logique triviale de ces hypothèses.

$$\text{Si } A \in \Gamma, \text{ alors } \Gamma \vdash A$$

On présente souvent une règle d'inférence de la façon suivante : la conclusion est placée sous une barre horizontale, le nom de la règle et d'éventuelles conditions d'application à droite de cette barre.

1. « assumption » en anglais, parfois appelée "axiome" mais bof

Voici la présentation habituelle de la règle d'hypothèse.

$$\frac{}{\Gamma \ \vdash A}\ \text{hyp}\ (A \in \Gamma)$$

Instances d'une règle

Dans la règle ci-dessus, les symboles A et Γ sont des variables pouvant être respectivement remplacées par n'importe quelle proposition et n'importe quel contexte.

Cette opération de remplacement s'appelle *instantiation* de la règle.

Soient par exemple trois variables propositionnelles P, Q et R.

Alors, la règle d'hypothèse nous permet de déduire le séquent :

$$\mathtt{P} \to \mathtt{Q},\ \mathtt{Q} \to \mathtt{R},\ \mathtt{P} \vdash \mathtt{P} \to \mathtt{Q}$$

On peut représenter la preuve associée comme une application de la règle d'hypothèse :

$$\frac{}{P{\to}Q, Q{\to}R, P \ \vdash P{\to}Q}\ \text{hyp}$$

En revanche, le séquent suivant *n'est pas* obtenu comme instance de la règle d'hypothèse (car la condition $A \in \Gamma$ n'est pas respectée)

$$\mathtt{P} \ \to\ \mathtt{Q} \vdash \mathtt{P}$$

3.3.2 Règle du modus ponens (élimination de l'implication)

Le sens de l'implication $A{\to}B$ est donné par la règle du modus-ponens (également appelée *règle d'élimination de l'implication*). Avoir prouvé $A{\to}B$, permet, si l'on a également prouvé A, d'en déduire B. Pour référencer cette règle, on pourra utiliser indifféremment la notion $\to_e$ (élimination de l'implication) ou *mp* (*modus ponens*).

La présentation usuelle de cette règle consiste, comme pour la règle d'hypothèse, à placer la conclusion sous une barre horizontale. Au dessus de cette barre, on place les séquents dont la preuve préalable conditionne l'application de la règle.

$$\frac{\Gamma \vdash A{\to}B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \text{mp}$$

Cette règle se lit :

> Si l'on peut prouver les séquents $\Gamma \vdash A{\to}B$ et $\Gamma \vdash A$, alors on peut prouver le séquent $\Gamma \vdash B$

En d'autres termes, cette règle permet d'obtenir une preuve de $\Gamma \vdash B$ à partir d'une preuve de $\Gamma \vdash A{\to}B$ et d'une preuve de $\Gamma \vdash A$.

Remarque

Il est important, lorsqu'on applique une règle d'inférence, de vérifier que l'on remplace toutes les occurrence d'une même méta-variable par la même formule ou le même contexte. Par exemple l'arbre suivant n'est pas une instance du modus-ponens :

$$\frac{P{\to}Q, R{\to}P \vdash P{\to}Q \qquad R, R{\to}P \vdash P{\to}P}{P{\to}Q, R{\to}P \vdash Q}\ \text{mp}$$

3.3.3 Preuves en déduction naturelle

Avec les deux règles : d'hypothèse et du modus-ponens, nous pouvons déjà écrire des preuves non-triviales. Les preuves en déduction naturelle prennent alors la forme de *dérivations*. Une dérivation est un arbre dont la racine est le séquent prouvé, et chaque noeud correspond à l'application d'une règle d'inférence. Les feuilles de cet arbre sont les applications de la règle d'hypothèse.

Exemple

La preuve suivante utilise les deux règles d'hypothèse et du modus ponens/ Afin d'alléger les notations nous posons $\Gamma = \{P{\to}\mathtt{Q},\ \mathtt{Q}{\to}\mathtt{R},\ P\}$

$$\dfrac{\dfrac{}{\Gamma \vdash Q{\to}R}\ \text{hyp} \qquad \dfrac{\dfrac{}{\Gamma \vdash P{\to}Q}\ \text{hyp} \qquad \dfrac{}{\Gamma \vdash P}\ \text{hyp}}{\Gamma \vdash Q}\ \text{mp}}{\Gamma \vdash R}\ \text{mp}$$

Présentation linéaire d'une preuve

Les arbres associés à des preuves complexes peuvent vite devenir illisibles. On pourra à la place représenter une preuve comme une suite de propositions. Les hypothèses seront signalées comme telles, et les propositions obtenues par l'application d'une règle d'inférence seront accompagnées d'une *justification* composée du nom de la règle appliquée, et de références permettant de retrouver à quelle formules ces règles sont appliquéees.

De façon plus précise :

— Chaque ligne de la preuve est soit une hypothèse, soit une proposition obtenue par l'application d'une règle d'inférence.

— À chaque ligne différente d'une hypothèse, on peut associer le séquent dont la conclusion est la formule courante et le contexte l'ensemble des hypothèses précédant cette ligne dans la preuve.

— Il va de soi que les références présentes dans une justification doivent porter sur des étapes précédant la ligne considérée.

Dans l'exemple ci-dessous, les lignes 1 à 3 sont des hypothèses. La ligne 4 est obtenue par modus ponens sur l'implication de la ligne 1 et l'hypothèse en ligne 3. De même pour la ligne 5, obtenue par modus ponens, à partir de la proposition $\mathtt{Q} \to \mathtt{R}$ et de la proposition $\mathtt{Q}$ prouvée en ligne 4.

Le séquent associé à la ligne 4 est

$$P{\to}Q, Q{\to}R, P \vdash Q$$

```
1 Supposons P → Q
2 Supposons Q → R
3 Supposons P
4 Q [mp, 1,3]
5 R [mp, 2,4]
```

Exercice 2. *Prouver le séquent suivant (dans les deux formats)*

$$P{\to}\mathtt{Q}{\to}\mathtt{R},\ P{\to}\mathtt{Q},\ P \vdash \mathtt{R}$$

(attention à l'associativité droite de $\to$ *!)*

3.3.4 Règle d'abstraction (introduction de l'implication)

Dans les exemples précédents, nous avons vu comment *utiliser* des propositions de la forme $A{\to}B$ présentes dans les hypothèses. La règle d'introduction de l'implication (souvent notée $\to_i$ permet de prouver des séquents de la forme $\Gamma \vdash A{\to}B$.

Cette règle formalise le mode de raisonnement intuitif suivant.

Pour prouver $A{\to}B$:

1. **Supposons** A
2. *Sous cette hypothèse*, prouvons B

La règle $\to_i$, dite d'*introduction de l'implication* formalise ce mode de raisonnement dit *raisonnement hypothétique*. Le fait de prouver B *en supposant* A se traduit simplement par "prouver le séquent $\Gamma, A \vdash B$ ".

Règle 1 (introduction de l'implication, $\to_i$)**.**

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A{\to}B}\ {\to_i}$$

Remarquons que l'hypothèse sur A n'est "active" que lors de la preuve de B. Cette remarque est explicitée par la représentation formelle de la règle $\to_i$, où la formule A n'est plus dans les hypothèses de la conclusion de la règle.

Exemple

Soit $\Gamma = \{P{\to}\mathtt{Q},\ \mathtt{Q}{\to}R\}$. La dérivation suivante utilise les trois règles de la logique minimale. *On remarquera que tous les séquents de cette preuve n'ont pas le même contexte.*

$$\dfrac{\dfrac{\dfrac{}{\Gamma, P \vdash Q{\to}R}\ \text{hyp} \qquad \dfrac{\dfrac{}{\Gamma, P \vdash P{\to}Q}\ \text{hyp} \qquad \dfrac{}{\Gamma, P \vdash P}\ \text{hyp}}{\Gamma, P \vdash Q}\ \text{mp}}{\Gamma, P \vdash R}\ \text{mp}}{\Gamma \vdash P{\to}R}\ {\to_i}$$

Présentation “à la C” d’une preuve

Une autre présentation du raisonnement hypothétique se rapproche de la structure de bloc des langages de programmation. La notion d’hypothèse s’apparente alors à celle de *déclaration de variable locale*.

Prenons par exemple la fonction 4.1 page 39. La variable local `i` est invisible depuis les lignes 9, 10 et 11 de la fonction.

Comme en **C**, une convention d’indentation permet de mieux visualiser la structure d’une dérivation. Nous adaptons donc la présentation de la section 3.3.3 page 19 en ajoutant une notation de bloc, délimité par des accolades. Chaque bloc peut contenir une hypothèse, dont la *portée* est limitée à ce bloc.

```
1  Supposons P → Q
2  Supposons Q → R
3  { Supposons P
4     Q [mp, 1, 3]
5     R [mp, 2, 4]
6  }
7  P → R [→ᵢ, 3, 5]
```

Il est important de comprendre la notion de *portée d’une hypothèse* et le rôle des blocs. Un bloc commençant par une ligne “**Supposons** A” contient des propositions prouvées en général à l’aide de l’hypothèse sur A. Afin de marquer cette dépendance, on convient qu’une proposition appartenant à ce bloc est inaccessible depuis l’extérieur de ce bloc.

Considérons par exemple la structure ci-dessous, comportant deux sous-dérivations :

```
1  { Supposons A
2      ...
3      B
4   }
5   ...
6   C
7   ...
8   { Supposons H
9      ...
10     D
11  }
12  E
```

— Une première sous-dérivation, sous l’hypothèse A ; la preuve de la justification de la proposition B peut utiliser cette hypothèse.
— De même avec un bloc d’hypothèse H
— Les justifications des formules C et E ne peuvent pas utiliser les propositions A, B, H et D.

— Le seul moyen d'utiliser par exemple la proposition B prouvée en ligne 3, dans une justification de E est de l'exporter par la règle d'introduction de l'implication.

```
1  { Supposons A
2      ...
3      B
4   }
5   ...
6   C
7   ...
8   { Supposons H
9      ...
10     D
11  }
12   A → B   [→ᵢ,1,3]
13  E
```

Dérivations imbriquées

Comme dans les langages de programmation, la structure de bloc associée au raisonnement hypothétique autorise l'imbrication de *sous-dérivations*. Chaque niveau d'imbrication pourra correspondre à autant d'applications de la règle d'introduction de l'implication.

La preuve suivante contient deux applications successives de la règle $\rightarrow_i$, traduites par deux blocs imbriqués.

$$\dfrac{\dfrac{\dfrac{}{P, Q \vdash P}\ \text{hyp}}{P \vdash Q \rightarrow P}\ \rightarrow_i}{\vdash P \rightarrow Q \rightarrow P}\ \rightarrow_i$$

```
1  { Supposons P
2    { Supposons Q
3     P [hyp,1]
4    }
5    Q → P [→ᵢ, 2, 3]
6  }
7  P → Q → P [→ᵢ, 1, 5]
```

Erreurs communes

La dérivation suivante est un exemple de non-respect de la structure de bloc. À la ligne 8, on tente d'appliquer l'introduction de l'implication sur une formule de la ligne 5, non accessible de la ligne 8.

```
1  Supposons P → Q
2   { Supposons Q → R
3    { Supposons P
4       Q [mp, 1, 3]
5       R [mp, 2, 4]
6      }
7    }
8  (Q → R) → R  [→ᵢ, 2, 5]
```

Cette dérivation incorrecte "prouve" le séquent non-valide suivant :

$$P \to Q \vdash (Q \to R) \to R$$

Une dérivation correcte (mais ne prouvant pas le même séquent) serait la suivante, où la proposition de la ligne 5 est "exportée" en tenant compte de l'hypothèse sur Q.

```
1  Supposons P → Q
2   { Supposons Q → R
3    { Supposons P
4       Q [mp, 1, 3]
5       R [mp, 2, 4]
6      }
7     P → R [→ᵢ, 3, 5]
8    }
9  (Q → R) → P → R  [→ᵢ, 2, 6]
```

Définitions

Soit $\Gamma \vdash A$ un séquent. On dira que ce séquent est *prouvable* (en logique minimale) s'il existe une dérivation en logique minimale concluant par se séquent. On utilisera la notation $\Gamma \vdash_M A$ pour exprimer le jugement "$\Gamma \vdash A$ est prouvable en logique minimale"

Notons que $\vdash_M$ *n'est pas un connecteur*. On ne confondra pas le *jugement* $A \vdash_M B$ avec la *proposition* $A \to B$.

Remarque Il faut bien distinguer la différence de statut linguistique entre un séquent $\Gamma \vdash A$ et un jugement $\Gamma \vdash_M A$. Le premier s'apparente à une structure de donnée (niveau "objet") et le second à une propriété mathématique (niveau "méta"). On trouvera des exemples de cette seconde catégorie dans la suite de ce document : jugements de validité $\Gamma \vDash A$, prouvabilité dans d'autres logiques : $\Gamma \vdash_J A$, $\Gamma \vdash_K A$, etc.

Introduction, élimination

Les vocables "règle d'introduction" et "règle d'élimination" sont utilisés en logique pour classifier les règles d'inférence associées aux connecteurs et quantificateurs.

— Une *règle d'introduction* permet de prouver un séquent de la forme $\Gamma \vdash A$ où l'opérateur principal de A est le connecteur ou quantificateur considéré.
— Une *règle d'élimination* permet de prouver un séquent de la forme $\Gamma \vdash A$, en utilisant un séquent $\Delta \vdash B$ où l'opérateur principal de B est le connecteur ou quantificateur considéré.

3.4 Comment simplifier l'écriture des preuves

3.4.1 Règles dérivées

Notion de règle dérivée

La preuve d'un séquent par les seules règles d'hypothèse, d'introduction et d'élimination peut se révéler très longue et fastidieuse. Une *règle dérivée* est par définition une règle qui n'apporte rien du point de vue de la prouvabilité, mais qui permet de remplacer plusieurs étapes d'application des règles de base. D'une certaine façon, l'usage de règles dérivées s'apparente à l'utilisation de fonctions dans un langage de programmation.

Avant d'utiliser une règle dérivée, il convient de montrer comment cette règle se traduit en termes de règles dont la validité est admise (règles de base de la logique, ou règles dérivées déjà validéees). En d'autres termes, une règle dérivée se *justifie* en donnant un algorithme permettant de traduire toute preuve utilisant cette nouvelle règle en une preuve n'utilisant que les règles de base ou les règles dérivées *déjà certifiées*.

Généralisation du modus-ponens

L'élimination de l'implication (modus-ponens) permet d'éliminer une implication de la formule $A \to B$. On rencontre fréquemment dans une preuve des propositions de la forme $A_1 \to A_2 \to \ldots \to A_n \to B$ (rappelons que l'implication est associative à droite).

Supposons que l'on soit capable de prouver chacun des séquents $\Gamma \to A_i$.

On peut obtenir la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A_1 \to A_2 \to \ldots A_{n-1} \to A_n \to B \quad \Gamma \vdash A_1}{\Gamma \vdash \to A_2 \to \ldots A_{n-1} \to A_n \to B}\,\text{mp} \quad \Gamma \vdash A_2}{\Gamma \vdash A_3 \to \ldots A_{n-1} \to A_n \to B}\,\text{mp}}{\vdots}}{\dfrac{\Gamma \vdash A_{n-1} \to A_n \to B \qquad \Gamma \vdash A_{n-1}}{\Gamma \vdash A_n \to B}\,\text{mp} \qquad \Gamma \vdash A_n}{\Gamma \vdash B}\,\text{mp}$$

Il est plus simple de court-circuiter toutes ces étapes en une seule, formalisée par une généralisation du modus-ponens :

$$\frac{\Gamma \vdash A_1 \to A_2 \to \ldots A_{n-1} \to A_n \to B \quad \Gamma \vdash A_1 \quad \Gamma \vdash A_2 \quad \ldots \quad \Gamma \vdash A_{n-1} \quad \Gamma \vdash A_n}{\Gamma \vdash B}\ \text{mp}$$

Exemple

Soit $\Gamma = \{P \to Q \to R, P, Q\}$ La dérivation suivante est correcte :

$$\frac{\dfrac{}{\Gamma \vdash P \to Q \to R}\ \text{hyp} \qquad \dfrac{}{\Gamma \vdash P}\ \text{hyp} \qquad \dfrac{}{\Gamma \vdash Q}\ \text{hyp}}{\Gamma \vdash R}\ \text{mp}$$

```
1 P → Q → R
2 P
3 Q
4 R [mp, 1,2,3]
```

Exemple

Cet autre exemple montre une application *partielle* du modus ponens généralisé :

$$\frac{\Gamma \vdash P \to Q \to R \to S \to T \quad \Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash R \to S \to T}$$

Généralisation de l'introduction de l'implication

De façon symétrique au modus-ponens, on peut dériver la règle suivante :

$$\frac{\Gamma, A_1, A_2, \ldots, A_n \vdash A}{\Gamma \vdash A_1 \to A_2 \to \ldots \to A_n \to A}\ \to_i$$

En voici un exemple d'instance :

$$\frac{\dfrac{}{P, Q \vdash P}\ \text{hyp}}{\vdash P \to Q \to P}\ \to_i$$

Exercice 3. *Montrer que cette généralisation de $\to_i$ est bien une règle dérivée de la logique minimale.*

Et les blocs ?

Dans la présentation structurées des preuves, il suffit de procéder aux modifications suivantes :

— Admettre plusieurs hypothèses $A_1, A_2, \ldots, A_n$ en tête de bloc
— Une proposition B située à l'intérieur d'un tel bloc s'exporte sous la forme $A_1 \to A_2 \to \ldots \to A_n \to B$.

```
{ Supposons P
    Supposons Q
   P [hyp,1]
}
P → Q → P [→ᵢ, 1-2, 3]
```

Remarque

Il est important d'exporter *toutes* les hypothèses d'un bloc. L'exemple suivant montre que le non-respect de cette recommandation peut conduire à la « preuve » de séquents non-valides.

```
{ Supposons P
    Supposons Q
   P [hyp,1]
}
Q → P [→ᵢ, 2, 3]
```

Exercice 4. *Prouver les séquents suivants (si possible dans les deux formats de preuve).*

```
⊢ P→P.

⊢ (P→P)→(P→P)

⊢ (P→Q)→(Q→R)→P→R

  P→Q→R ⊢ (Q→P→R)

P→R ⊢ P→Q→R

P→P→Q ⊢ P→Q

P→Q, P→R, Q→R→T ⊢ P→T
```

Règle de coupure

Supposons qu'on veuille prouver un séquent $\Gamma \vdash A$, et que l'on trouve cette tâche difficile. En revanche, vous trouvez une proposition B telle que :

— Vous vous sentez capable de prouver le séquent $\Gamma \vdash B$

— Vous pensez qu'une fois prouvée B, il est plus facile de prouver A.

Une stratégie possible peut consister à travailler en trois étapes :

1. Prouver d'abord un séquent $\Gamma \vdash B$, où B est une proposition appelée *lemme*.

2. Prouver l'implication $\Gamma \vdash B \to A$
3. appliquer le modus ponens pour en déduire $\Gamma \vdash A$.

En bref :

$$\dfrac{\Gamma \vdash B \qquad \dfrac{\Gamma, B \vdash A}{\Gamma \vdash B \to A}\ \to_i}{\Gamma \vdash A}\ \text{mp}$$

Nous pouvons associer une nouvelle règle dérivée, la *règle de coupure* à cette stratégie, de façon à court-circuiter la succession formée d'une introduction, immédiatement suivie d'une élimination de l'implication.

Nous utilisons (provisoirement) des couleurs pour distinguer les deux composantes d'une coupure.

$$\dfrac{\Gamma \vdash B \qquad \Gamma, B \vdash A}{\Gamma \vdash A}\ \text{cut}$$

Voici un premier exemple d'application de cette règle.
Soit $\Gamma = \{P \to Q \to R, P \to Q, P\}$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash P \to Q}\ \text{hyp} \qquad \dfrac{}{\Gamma \vdash P}\ \text{hyp}}{\Gamma \vdash Q}\ \text{mp} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, Q \vdash P \to Q \to R}\ \text{hyp} \qquad \dfrac{}{\Gamma, Q \vdash P}\ \text{hyp}}{\Gamma, Q \vdash Q \to R}\ \text{mp} \qquad \dfrac{}{\Gamma, Q \vdash Q}\ \text{hyp}}{\Gamma, Q \vdash R}\ \text{mp}}{\Gamma \vdash R}\ \text{cut}$$

Dans le format structuré des preuves, la règle de coupure peut se représenter très simplement. On commence par prouver la proposition B, dans un bloc dédié à cette preuve. Une fois prouvée, B peut être utilisée dans le reste de la démonstration.

Reprenons notre exemple :

```
1  Supposons P → Q → R
2  Supposons P → Q
3  Supposons P
4  Prouvons Q {
5    Q [mp, 2,3]
6  }
7  R [mp, 1, 3, 4]
8  }
```

L'exemple suivant, un peu plus complexe, montre comment la règle de coupure permet de prouver une seule fois la proposition `P → R` et de l'appliquer deux fois (en lignes 13 et 16). L'utilisation de lemmes (et donc de la règle de coupure) permet souvent de réduire la taille des preuves. En ce sens, la règle de coupure s'apparente aux techniques pour éviter la duplication de code en programmation. Là où le programmeur utilise des fonctions auxiliaires pour "factoriser" son code, le "logicien" invente des lemmes.

```
Supposons P → Q
Supposons Q → R
Supposons (P → R) → T → Q
Supposons (P → R) → T
Prouvons P → R {
  { Supposons P
     Q [mp, 1, 6]
     R [mp, 2, 7]
   }
  P → R [→ᵢ, 6,8]
  }
Prouvons T {
  T [mp, 4, 5]
}

Q [mp, 3, 5, 12]
```

3.4.2 Règle d'affaiblissement

Si l'on peut prouver un séquent $\Gamma \vdash A$, on peut *a fortiori* prouver sa conclusion en ajoutant un ensemble arbitraire d'hypothèses.

La *règle d'affaiblissement* tient compte de ce fait :

$$\frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A}\ \text{aff}$$

Exercice 5. *Montrer que la règle d'affaiblissement est bien une règle dérivée. On pourra procéder par récurrence sur [la taille de] la preuve de* $\Gamma \vdash A$

Exercice 6. *Montrer que la "règle" suivante est visiblement fausse.*

$$\frac{\Gamma, \Delta \vdash A}{\Gamma \vdash A}\ \textit{renforcement}$$

Exercice 7. *Le but de cet exercice est de comprendre comment créer une règle dérivée à partir de la preuve d'un séquent.*

Soient A *et* B *deux propositions quelconques. On suppose qu'on a une preuve en logique minimale*[2] *du séquent* $A \vdash B$*. Montrer qu'on peut alors dériver la règle suivante :*

$$\frac{\Gamma \vdash A}{\Gamma \vdash B}$$

2. Ce résultat s'étendra aux logiques présentéees dans les chapitres suivants.

3.4.3 Prouvabilité et règles dérivéees

Les règles dérivées que nous avons vues (ainsi, en général, que toutes celles que nous verrons par la suite), présentent plusieurs avantages :

— Elles permettent de réduire notablement la taille des preuves, et donc d'accroître leur lisibilité

— Elles facilitent la recherche d'une preuve d'un séquent donné (notamment la règle de coupure).

En revanche, elles n'ajoutent rien à l'ensemble des séquents prouvables : si la preuve d'un séquent utilise des règles dérivées, alors on peut construire une preuve du même séquent n'utilisant que les trois règles de base de la logique minimale.

Les règles dérivées : mode d'emploi

Les remarques qui suivent sont bien sûr immédiatement applicables à la logique minimale, mais seront utiles dans tout le document

— Les règles dérivées sont introduites dans le texte du document, mais aussi dans les exercices.

— La preuve de validité des ces règles est parfois laissée en exercice. Dans d'autres cas, cette preuve est trop longue ou complexe pour ce cours, et on pourra admettre le résultat ou chercher de la documenttaion sur Internet.

— Si par malheur, vous ne pouvez pas faire un tel exercice, vous pouvez cependant admettre la règle en question dans la suite du document.

— Une fois qu'une règle est présentée et prouvée ou admise, son utilisation permet de simplifier la recherche ou l'écriture de démonstrations.

— On peut trouver que de nombreux exercices se ressemblent ; cependant nous encourageons le lecteur à essayer de trouver les preuves les plus simples et courtes ; l'emploi de règles dérivées est un moyen d'obtenir cette simplicité.

3.4.4 Omission d'étapes triviales

On peut convenir de laisser implicites certaines feuilles (règle d'hypothèse) d'une dérivation.

Par exemple, soit la dérivation :

$$\dfrac{\dfrac{}{P{\to}Q, P \vdash P{\to}Q}\ \text{hyp} \qquad \dfrac{}{P{\to}Q, P \vdash P}\ \text{hyp}}{P{\to}Q, P \vdash Q}\ \text{mp}$$

On peut sans danger l'abréger ainsi :

$$\dfrac{}{P{\to}Q, P \vdash Q}\ \text{mp}$$

Dans la présentation d'une preuve, l'omission de sous-dérivations triviales permet de mieux visualiser la structure de la démonstration.

```
Supposons P → Q
Supposons Q → R
Supposons (P → R) → T → Q
Supposons (P → R) → T
Prouvons P → R {…}
Prouvons T {…}
Q [mp, 3, 5, 6]
```

3.5 Preuves *vs* validité

Nous avons deux notions différentes de vérité pour un séquent $\Gamma \vdash A$:
— Une vérité "sémantique", exprimée par le jugement $\Gamma \vDash A$
— Une vérité " syntaxique" exprimée par le jugement $\Gamma \vdash_M A$.
Il est temps de comparer ces deux approches.

3.5.1 Correction

Le (méta-) théorème de correction établit que les règles de déduction de la logique minimale ne permettent de prouver que des séquents valides.

Méta-théorème 1 (Correction)**.** *Si* $\Gamma \vdash_M A$*, alors* $\Gamma \vDash A$

Pour prouver ce thèorème, on peut procéder à une *récurrence* sur la structure des preuves.

1. Soit une preuve obtenue par la règle d'hypothèse :

$$\frac{}{\Gamma \vdash A}\ \text{hyp}$$

   Par construction, la proposition A appartient au contexte Γ. Donc, toute valuation ν qui satisfait toutes les hypothèses de Γ satisfait forcément A, et le séquent $\Gamma \vdash A$ est donc valide.

2. Soit une preuve se terminant par un modus-ponens. Elle a la forme suivante :

$$\frac{\begin{array}{cc}\vdots\,\pi_1 & \vdots\,\pi_2 \\ \Gamma \vdash A{\to}B & \Gamma \vdash A\end{array}}{\Gamma \vdash B}\ \text{mp}$$

   On suppose (hypothèses de récurrence) $\Gamma \vDash A{\to}B$ et $\Gamma \vDash A$
   Montrons le jugement $\Gamma \vDash B$:

   (a) Soit ν une valuation qui rend vraies toutes le hypothèses de Γ. Par hypothèse, on a $\nu(A) = \boldsymbol{v}$ et $\nu(A{\to}B) = \boldsymbol{v}$ Par conséquent, on a $\nu(B) = \boldsymbol{v}$.

3. Soit une preuve se terminant par une introduction de l'implication. Elle a la forme suivante :

$$\dfrac{\begin{array}{c}\vdots\,\pi\\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash A{\to}B}\ {\to_i}$$

On suppose (hypothèse de récurrence) $\Gamma, A \vDash B$ Montrons le jugement $\Gamma \vDash A{\to}B$:

(a) Soit ν une valuation satisfaisant toutes les hypothèses de Γ. Deux cas sont possibles :

i. Si $\nu(A) = \boldsymbol{v}$, alors ν satisfait toutes les hypothèses de Γ ainsi que l'hypothèse A, et par conséquent $\nu(B) = \boldsymbol{v}$, et $\nu(A{\to}B) = \boldsymbol{v}$

ii. Si $\nu(A) = \boldsymbol{f}$, alors $\nu(A{\to}B) = \boldsymbol{v}$

3.5.2 Incomplétude

On aurait aimé prouver la rèciproque du théorème de correction :

Tout séquent valide est prouvable en logique minimale

Eh bien non ! Au contraire on peut prouver le résultat suivant :

Méta-théorème 2 (Incomplétude de la logique minimale)**.** *Il existe au moins un séquent valide, et non-prouvable en logique minimale.*

Idée de démonstration

On considère la *Formule de Peirce* : $((P{\to}Q){\to}P){\to}P$.

Exercice 8. *Montrer que la formule de Peirce est une tautologie ; autrement dit, le séquent* $\vdash ((P{\to}Q){\to}P){\to}P$ *est valide.*

Une analyse des propriétés de la logique minimale (que nous ne détaillons pas ici) permet de démontrer qu'aucune preuve de la logique minimale ne permet de prouver le séquent $\vdash ((P{\to}Q){\to}P){\to}P$.

Commentaires sur le résultat d'incomplétude

Bon, ce dernier résultat semble montrer que la logique minimale est un peu faible, puisqu'elle est incapable de démontrer n'importe quelle tautologie. Par ailleurs, elle nous a permis de nous familiariser avec les notions de règles d'inférence, de preuve, de règle dérivée, etc.

Dans les chapitres suivants, nous procéderons à des extensions successives de cette logique, en enrichissant la syntaxe et l'ensemble de règles d'inférence.

3.5.3 Théorème de substitution uniforme

La logique propositionnelle permet, une fois qu'on a prouvé un théorème de la forme $\Gamma \vdash A$ d'en déduire une infinité d'autres sans effort ©. En fait, cela revient à transformer tout séquent en une régle. Pour ce faire, il suffit de remplacer *uniformément* des variables propositionnelles par des propositions à la fois dans Γ et dans A.

Plus précisément :

Définition 1 (Substitution)**.** *On considère un ensemble fini de variables propositionnelles $\{v_1, \ldots, v_n\} \subseteq V_p$, et n formules $B_1, \ldots, B_n$ On définit récursivement la substitution dans une proposition A des formules B_i aux variables v_i (pour $i \in 1..n$), notée $A[v_1/B_1, \ldots, v_n/B_n]$ par*

- $v_i[v_1/B_1, \ldots, v_n/B_n] = B_i; (i \in 1..n)$
- $v[v_1/B_1, \ldots, v_n/B_n] = v \ (v \in V_p \backslash \{v_1, \ldots, v_n\})$
- $(A \to B)[v_1/B_1, \ldots, v_n/B_n] = (A[v_1/B_1, \ldots, v_n/B_n] \to A[v_1/B_1, \ldots, v_n/B_n])$

Exercice 9. *Étendre la notion de substitution aux séquents, de façon à définir la notation $(\Gamma \vdash A)[v_1/B_1, \ldots, v_n/B_n]$*

Méta-théorème 3 (Théorème de substitution uniforme)**.** *Soit un séquent prouvable en logique minimale $\Gamma \vdash A$. Alors le séquent $(\Gamma \vdash A)[v_1/B_1, \ldots, v_n/B_n]$ (avec les notations ci-dessus) est aussi prouvable en logique minimale.*

Exemple

On considère le jugement $A \to B, B \to C \vdash_M A \to C$. Si l'on remplace *uniformément* A par $P \to Q$, B par R, et C par $S \to T$, on obtient le jugement suivant : $(P \to Q) \to R, R \to S \to T \vdash_M (P \to Q) \to S \to T$.

Remarques

Attention aux parenthèses !

Le jugement $P \to Q \to R, Q \to R \to S \vdash_M P \to S$ n'est pas une instance de $A \to B, B \to C \vdash_M A \to C$.

En effet si l'on veut substituer P à A, $Q \to R$ à B et S à C, nous obtenons plutôt le jugement ci-dessous, de structure différente :

$$P \to (Q \to R), (Q \to R) \to S \vdash_M P \to S$$

On se rappellera les problèmes liés à l'usage du `#define` du langage **C**, où une méconnaissance des priorités et associativité des opérateurs peut conduire à des substitutions malheureuses. En cas de doute, il vaut mieux définir les substitutions par des expressions bien parenthésées, quitte à enlever les parenthéses inutiles après avoir effectué la substitution.

Dans notre exemple, la substitution correcte est :

substituer P à A, $(Q \to R)$ à B et S à C

On obtient bien le jugement correct.

Par ailleurs, pour que le théorème de substitution uniforme soit correctement appliqué, il est important que la substitution soit *uniformément* appliquée dans le contexte et la conclusion du séquent.

Prenons comme exemple à ne pas suivre le séquent (prouvable)

$$P{\to}Q \vdash (Q{\to}R){\to}P{\to}R$$

Si l'on remplace Q par S *seulement dans la conclusion du séquent* on obtient le séquent non valide (et donc non prouvable)

$$P{\to}Q \vdash (S{\to}R){\to}P{\to}R$$

3.6 Stratégies de démonstration

Supposons que l'on cherche à écrire (si possible) une preuve d'un séquent $\Gamma \vdash A$.

Une démarche possible est de considérer la preuve de ce séquent comme un *but* à atteindre ; nous pourrons utiliser la notation $\Gamma \vdash^? A$ pour désigner un tel but, c'est à dire une preuve incomplète.

Par exemple, considérons le but $\Gamma \vdash^? P{\to}R$, avec $\Gamma = \{P{\to}Q, Q{\to}R\}$.

Au départ de notre recherche de preuve, nous avons la situation suivante :

$$\Gamma \vdash^? P{\to}R$$

La conclusion du but courant étant une implication, nous pouvons tenter d'appliquer $\to_i$, ce qui engendre un nouveau *sous-but*.

$$\frac{\Gamma, P \vdash^? R}{\Gamma \vdash P{\to}R}\ {\to}_i$$

On pense résoudre le but $\Gamma, P \vdash^? R$ en appliquant le modus ponens, ce qui nous donne deux sous-buts :

$$\frac{\dfrac{\Gamma, P \vdash^? Q{\to}R \qquad \Gamma, P \vdash^? Q}{\Gamma, P \vdash R}\ \text{mp}}{\Gamma \vdash P{\to}R}\ {\to}_i$$

Le sous-but de gauche se résoud immédiatement avec la règle d'hypothèse, celui de droite peut utiliser encore une fois le modus-ponens.

$$\frac{\dfrac{\dfrac{}{\Gamma, P \vdash Q{\to}R}\ \text{hyp} \qquad \dfrac{\Gamma, P \vdash^? P{\to}Q \qquad \Gamma, P \vdash^? P}{\Gamma, P \vdash Q}\ \text{mp}}{\Gamma, P \vdash R}\ \text{mp}}{\Gamma \vdash P{\to}R}\ {\to}_i$$

On obtient finalement une preuve complète (sans " ?") !

$$\dfrac{\dfrac{\dfrac{}{\Gamma, P \vdash Q \to R}\ \text{hyp} \qquad \dfrac{\dfrac{}{\Gamma, P \vdash P \to Q}\ \text{hyp} \qquad \dfrac{}{\Gamma, P \vdash P}\ \text{hyp}}{\Gamma, P \vdash Q}\ \text{mp}}{\Gamma, P \vdash R}\ \text{mp}}{\Gamma \vdash P \to R}\ \to_i$$

Si l'on veut généraliser cet exemple, on peut proposer l'heuristique suivante pour prouver un séquent $\Gamma \vdash A$:

1. Créer un but $\Gamma \vdash^? A$.
2. tant qu'il reste un but non résolu de la forme $\Gamma \vdash^? A$, appliquer l'une des tactiques suivantes :
   — Si $A \in \Gamma$, le but se résoud immédiatement à l'aide de la règle d'hypothèse.
   — Si A est de la forme $A_1 \to A_2 \to \ldots \to A_n \to A$, appliquer la règle d'introduction de l'implication, et engendrer un nouveau but $\Gamma, A_1, A_2, \ldots, A_n \vdash^? A$
   — Si l'on peut prouver $\Gamma \vdash A_1 \to A_2 \to \ldots \to A_n \to A$ [3], alors appliquer la règle d'élimination de l'implication, et engendrer les sous-buts $\Gamma \vdash^? A_1$, $\Gamma \vdash^? A_2$, …$\Gamma \vdash^? A_n$.

Remarque

Appliquer une telle stratégie de démonstration revient à dessiner un arbre de preuve en commençant par la racine (le séquent qu'on veut prouver) et en terminant par les feuilles (résolution de sous-buts triviaux).

Cette stratégie est celle utilisée par la tactique `auto` de l'assistant de preuve *Coq*.

Remarque

On peut bien sûr établir des stratégies plus élaborées (par exemple en appliquant des règles dérivées). On remarquera cependant le cas particulier de la règle de coupure :

Soit un but $\Gamma \vdash^? A$. On peut essayer de résoudre ce but en prouvant un lemme :

$$\dfrac{\Gamma \vdash^? B \qquad \Gamma, B \vdash^? A}{\Gamma \vdash A}\ \text{cut}$$

Il reste que la proposition B peut très bien ne pas être une sous-formule de $\Gamma \cup \{A\}$. C'est à l'auteur de la démonstration de trouver quelle formule B aidera à compléter la preuve en cours.

Exercice 10. *Reprendre les exercices 2 et 4, en utilisant la stratégie de recherche systématique de preuve.*

Exercice 11. *Il est difficile de trouver à la main une preuve du séquent suivant :*

3. en particulier, si la proposition $A_1 \to A_2 \to \ldots \to A_n \to A$ fait partie des hypothèses de Γ

$((((P \to Q) \to P) \to P) \to Q) \vdash Q$

En revanche, il ne posera pas de problème en séance sur machine.

3.7 Logique et Programmation Fonctionnelle

Dans cette section, nous commençons à explorer la correspondance entre la programmation fonctionnelle et la logique. Cette correspondance, *dite de Curry-Howard* est à la base de la réalisation d'assistants de preuves tels que *Coq*, dont nous allons utiliser les notations, très voisines de celles du langage *OCaML*.

3.7.1 Types et propositions

Considérons un ensemble de propositions atomiques P, Q, etc. Nous avons vu comment former les propositions de la logique minimale à l'aide du conncteur $\to$. Si nous considérons P, Q comme des types de base du lambda-calcul simplement typé, nous pouvons associer à toute proposition de la logique mimimale un type du lambda-calcul simplement typé.

— Si P est une proposition atomique, on lui associe le type de base P

— Soit une implication de la forme $A \to B$, on lui associe le type $A \to B$ des fonctions totales du type A vers le type B.

On remarque que jusqu'à présent, cette "correspondance" n'est que l'identité. Le plus important est de voir dans cette correspondance une assimilation entre deux notions de nature apparemment différente : l'implication (intuitionniste) logique, et le typage des fonctions dans un langage de programmation.

3.7.2 Termes et Preuves

Nous proposons une représentation des preuves de la logique minimale par des expressions (aussi appelés *termes*) de notre langage fonctionnel. Cette représentation est basée sur l'assimilation entre deux jugements : *t est une preuve de A* et *t est un terme de type de A*.

Plus précisément, nous devons prendre en compte les notions d'hypothèse et de séquent, et nous nous intéresserons aux jugements "*dans le contexte Γ, t est une preuve de A*" et "*dans le contexte Γ, t est un terme de type de A*" .

Dans le premier jugement, Γ est un ensemble de propositions considérées comme les prémisses du jugement, dans le second, Γ est un ensemble de déclarations de variables. Afin de rendre compatibles ces deux visions du contexte, on conviendra que chaque hypothèse d'un contexte doit avoir un nom (distinct de celui des autres hypothèses).

Du coup, un contexte de la logique minimale aura la forme $h_1 : A_1, h_2 : A_2, \ldots, h_n : A_n$, où les $h_i$sont des identificateurs deux à deux distincts. Chaque déclaration $h_i : A_i$peut se lire dans un cadre comme "*L'hypothèse h_i d'énoncé A_i*", dans l'autre comme "*La déclaration de la variable h_i de type A_i*".

Reprenons les trois règles de la logique minimale, en les mettant en correspondance avec les règles de typage de la programmation fonctionnelle.

Un jugement pourra s'écrire $\Gamma \vdash_M t : A$ et se lire "dans le contexte Γ, t est une preuve [4] de A", mais aussi "dans le contexte Γ, t est un terme de type A"

Règle d'hypothèse

On peut amender la règle telle que présentée en section 3.3.1 page 17 :

Si le contexte Γ contient la déclaration $a : A$, alors $\Gamma \vdash_M a : A$

Règle du modus ponens

La règle d'élimination de l'implication (modus-ponens) correspond à la règle de typage associée à l'application d'une fonction en lambda-calcul simplement typé.

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B}\ \text{mp}$$

Autrement dit : Une preuve de $A \to B$ est une fonction f qui, appliquée à une preuve a de A, retourne une preuve de B. Cette dernière preuve est juste l'application de la fonction f à l'argument a.

Par exemple, soit un contexte Γ contenant les déclarations $f : A \to B \to C$, $b : B$, et $c : C$. Alors le terme $f\ a$ aura pour type $B \to C$, et le terme $f\ a, b$ [5] aura pour type C. Donc, le terme $f\ a\ b$ est une preuve de la proposition C dans le contexte Γ.

Remarque

La généralisation du modus-ponens vue en section 3.4.1 page 24 s'interprête très facilement en programmation fonctionnelle. *Rappelons que dans les langages fonctionnels comme* OCaML *et* Coq*, l'application d'une fonction à son argument est considérée comme une opération associant à gauche : l'écriture* $f\ a_1\ a_2\ \ldots\ a_{n-1}\ a_n$ *est une abréviation de* $((f\ a_1)\ a_2)\ \ldots\ a_{n-1})\ a_n$.

$$\frac{\Gamma \vdash f : A_1 \to A_2 \to \ldots A_{n-1} \to A_n \to B \qquad \Gamma \vdash a_1 : A_1 \qquad \Gamma \vdash a_2 : A_2 \qquad \ldots \qquad \Gamma \vdash a_n : A_{n-1} \qquad \Gamma \vdash a_n : A_n}{\Gamma \vdash f\ a_1\ a_2\ \ldots\ a_n : B}\ \text{mp}$$

Règle d'abstraction

Puisque la preuve d'une implication $A \to B$ s'interprête comme une fonction du type fonctionnel $A \to B$, l'introduction de l'implication s'apparente à la règle de typage des fonction en lambda-calcul simplement typé.

$$\frac{\Gamma, a : A \vdash t : B}{\Gamma \vdash \text{fun}\ a : A => t : A \to B}\ \to_i$$

4. On dit aussi "est un terme de preuve de A"
5. avec les conventions syntaxiques de *Coq* et *OCaML* sur l'application de fonction

Exemple

Par exemple, considérons le but $\Gamma \vdash^? P \to R$, avec $\Gamma = \{P \to Q, Q \to R\}$.

On commence par nommer les hypothéses de Γ : $\Gamma = \{H : P \to Q, H0 : Q \to R\}$. Pour cosntruire un terme de type $P \to R$ dans Γ, on cherche à construire un terme de type R dans le contexte étendu $\Gamma' = \{H : P \to Q, H0 : Q \to R, p : P\}$.

L'application $H0\,(H\ p)$ est clairement de type R dans Γ',

et donc l'abstraction fun$p$: $P{=>}H0\,(H\ \ p)$ est bien une preuve de $P \to R$ dans Γ.

Résumé

On peut donc convenir qu'une preuve en logique minimale du séquent $h_1 : A_1, h_2 : A_2, \ldots, h_n : A_n \ \vdash\ A$ est un terme du lambda-calcul simplement typé de type A dans le contexte $h_1 : A_1, h_2 : A_2, \ldots, h_n : A_n$.

Exercice 12. *Reprendre les exercices 2 et 4, en donnant le terme du lambda-calcul simplement typé pour chacune des preuves.*

Chapitre 4

Logique Propositionnelle (intuitionniste, puis classique)

Dans ce chapitre, nous étudions une première extension de la logique minimale, afin d'étudier les schémas de raisonnement par l'absurde, puis nous introduirons les connecteurs $\wedge$, $\vee$, et $\Leftrightarrow$.

Finalement, nous considérons deux versions de la logique propositionnelle, selon que nous incluons ou pas le *principe du tiers exclu*.

4.1 La contradiction et la négation

4.1.1 Introduction

La *négation* est ommiprésente dans le langage courant : en voici quelques exemples :

"Socrate n'est pas immortel"

"je ne te hais point"

"Il ne pleuvra pas au sud d'une ligne Bordeaux Lyon"

En Mathématiques et en Informatique, elle apparait dans des situations très courantes, parfois masquée par des abréviations :

— La proposition $i \neq j$ est une abréviation de $\neg(i = j)$
— La proposition $i < j$ est une abréviation de $i \leq j \wedge i \neq j$, etc
— L'instruction `assert` de **C**, permet de stopper l'éxécution d'un programme si son argument est une expression fausse (voir figure 4.1 page suivante)

```c
double harmonic(long t[], int n) {
double s = 0.0;
  assert (n > 0);
  { int i = 0;
     for(i=0, i < n ; i++){
      assert(t[i] != 0.0);
      s += 1.0/t[i];
      }
     ...
  }
  return n/s ;
}
```

FIGURE 4.1 – Utilisation d'assert en C

La notion de contradiction, plus simplement de proposition fausse, est tellement ancrée dans nos neurones qu'il semble difficile d'en donner une *définition* précise. Des exemples de propositions fausses seraient :

— "$2 = 3$"
— "Bordeaux est une ville de montagne"
— " `C` est le seul langage de programmation existant"

4.1.2 Syntaxe

Pour des raisons de simplicité, nous procédons en deux étapes :

1. Nous introduidons une proposition "toujours fausse" : *la contradiction*, notée $\bot$.
2. Nous en dérivons la *négation* : La négation notée $\sim A$ d'une proposition A est juste une abréviation de la proposition $A \to \bot$.

La contradiction

Définition 2. *On ajoute une clause à la définition de l'ensemble des propositions donnée en section 3.1 page 14 :*

— *Toute variable propositionnelle est une proposition*
— *La contradiction $\bot$ est une proposition*
— *si A et B sont deux propositions, alors l'implication $(A \to B)$ est une proposition*

sous-entendu : Il n'y a pas d'autres propositions

La négation

En suivant la tradition des *macros* des langages de programmation, nous ne considérons pas la *négation* comme une nouvelle construction mais comme une abréviation.

> Soit A une proposition ; la proposition $\sim A$ est une abréviation de $(A \to \bot)$.

L'opérateur $\sim$ se prononce "non". La notation $\neg A$ est aussi très usitée.

Par exemple, la proposition $(P \to Q) \to \sim Q \to \sim P$ est une abréviation de $(P \to Q) \to (Q \to \bot) \to P \to \bot$ et donc une instance de $(A \to B) \to (B \to C) \to A \to C$

En reprenant un exemple antérieur, la proposition $i \neq j$ est donc une abréviation de $i = j \to \bot$

4.1.3 Aspects sémantiques de la contradiction

Pour définir la sémantique de propositions et de séquents contenant la proposition $\bot$, et, par conséquent, la négation $\sim$, il suffit de poser que pour n'importe quelle valuation ν, on a $\nu(\bot) = \boldsymbol{f}$ Par conséquent, pour toute proposition A, nous avons

$$\begin{aligned} \nu(\sim A) &= \nu(A \to \bot) && (4.1)\\ &= \boldsymbol{f} \text{ si } \nu(A) = \boldsymbol{v} && (4.2)\\ &= \boldsymbol{v} \text{ si } \nu(A) = \boldsymbol{f} && (4.3) \end{aligned}$$

Considérons maintenant un jugement de la forme $\Gamma \models \bot$. Nous pouvons en inférer que toute valuation ν qui satisfait (simultanément) toutes les hypothèses de Γ satisfait également $\bot$, ce qui est impossible.

Soit maintenant une proposition quelonque A. Il n'existe aucune valuation ν satisfaisant toutes les hypothèses de Γ et ne satisfaisant pas la conclusion A, donc il n'existe aucun contre-exemple possible à la validité de $\Gamma \vdash A$. On peut donc en conclure le jugement $\Gamma \models A$.

Nous venons donc de prouver le résultat suivant :

> Si $\Gamma \models \bot$, alors $\Gamma \models A$, pour n'importe quelle proposition A

4.1.4 Le principe d'explosion

La règle d'inférence ci-dessous dite ("*principe d'explosion*", mais aussi "*ex falso (sequitur) quodlibet*", ou "*élimination de la contradiction"* reflète le raisonnement du paragraphe précédent.

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot_e$$

Voici un exemple de preuve utilisant ce principe.

$$\dfrac{\dfrac{}{\bot \vdash \bot}\ \text{hyp}}{\bot \vdash P}\ \bot_e$$

Cette règle est la base du raisonnement par l'absurde.

Remarque

La règle $\bot_e$ peut sembler artificielle au premier abord. Remarquons cependant que le principe d'explosion (sous sa dénomination latine) est connu depuis le moyen âge (pour ne pas dire l'antiquité). Nous verrons plus loin (Section 4.1.5 page suivante) comment cette règle sert à valider le "raisonnement par l'absurde".

Une autre façon de concevoir la règle $\bot_e$ serait de considérer la contradiction $\bot$ comme une façon de dire "toute proposition est vraie". Si toute proposition est vraie, alors $2 = 2$, $2 \neq 2$ sont vraies, et la notion de vérité perd tout sens. Nous ne nous attarderons pas sur cet aspect, car cela demanderait d'étudier la *logique du second ordre* qui n'est pas au programme de licence ☹.

Exercice 13. *Faire des recherches sur Internet sur les termes "ex falso quodlibet sequitur" et "principle of explosion"*

4.1.5 Règles associées à la négation

Rappelons que toute proposition de la forme $\sim A$ est juste une abréviation de $A \to \bot$. Les règles d'inférences associées à la négation seront donc des règles dérivées.

Macro-expansion

Dans toute preuve, on peut remplacer n'importe quelle sous-formule de la forme $\sim A$ par $(A \to \bot)$ et réciproquement.

Introduction de la négation

Pour prouver un séquent $\Gamma \vdash \sim A$ il suffit d'expanser la négation et d'appliquer la règle d'introduction.

$$\dfrac{\dfrac{\Gamma, A \vdash \bot}{\Gamma \vdash A \to \bot}\ \to_i}{\Gamma \vdash \sim A}\ \sim\text{-expansion}$$

D'où la règle dérivée :

$$\dfrac{\Gamma, A \vdash \bot}{\Gamma \vdash \sim A}\ \sim_i$$

Contraposée

$$\dfrac{\Gamma \vdash A \to B}{\Gamma \vdash \sim B \to \sim A}\ \text{contraposée}$$

En voici la justification :

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A{\to}B}{\Gamma, {\sim}B, A \vdash A{\to}B}\ \text{aff} \qquad \dfrac{}{\Gamma, {\sim}B, A \vdash A}\ \text{hyp}}{\Gamma, {\sim}B, A \vdash B}\ \text{mp} \qquad \dfrac{}{\Gamma, {\sim}B, A \vdash {\sim}B}\ \text{hyp}}{\Gamma, {\sim}B, A \vdash B{\to}\bot}\ {\sim}\text{-expansion}}{\Gamma, {\sim}B, A \vdash \bot}\ \text{mp}$$
$$\dfrac{\Gamma, {\sim}B, A \vdash \bot}{\Gamma, {\sim}B \vdash {\sim}A}\ {\sim}_i$$
$$\dfrac{\Gamma, {\sim}B \vdash {\sim}A}{\Gamma \vdash {\sim}B{\to}{\sim}A}\ {\to}_i$$

Raisonnement par l'absurde (intuitionniste)

A l'aide de la règle $\bot_e$, nous pouvons implanter un premier mode de raisonnement par l'absurde : Pour prouver $\Gamma \vdash A$ il *suffit* de trouver une proposition B et de prouver les sequents $\Gamma \vdash {\sim}B$ et $\Gamma \vdash B$

$$\dfrac{\Gamma \vdash {\sim}B \qquad \Gamma \vdash B}{\Gamma \vdash A}\ \text{absurde}$$

Exercice 14. *Justifier (par une dérivation) cette nouvelle règle dérivée.*

Exercice 15. *En utilisant la règle* `absurde`*, prouver le séquent*

$$P{\to}Q, P{\to}{\sim}Q \vdash P{\to}S$$

.

Remarque

Du point de vue de la preuve interactive de théorèmes, la règle `absurde` peut s'utiliser comme une "tactique d'élimination de la négation" : Si l'on cherche à prouver un séquent $\Gamma \vdash A$ et qu'on dispose déjà d'une preuve de $\Gamma \vdash {\sim}B$, alors il *suffit* de prouver aussi le séquent $\Gamma \vdash B$.

Il est remarquable dans ce cas que la proposition A "disparait" du nouveau but. C'est encore une conséquence du principe d'explosion. Dans le cas considéré, la preuve de $\Gamma \vdash A$ revient plus à chercher une contradiction dans Γ que de chercher une preuve de A dans ce contexte.

Exercice 16. *Prouver les règles dérivées suivantes (une fois prouvées, on pourra les utiliser dans tout raisonnement ultérieur)*

$$\dfrac{\Gamma \vdash A}{\Gamma \vdash {\sim}{\sim}A}\ \textit{double négation}_i$$

$$\dfrac{\Gamma \vdash {\sim}{\sim}{\sim}A}{\Gamma \vdash {\sim}A}\ \textit{triple négation}_e$$

4.1.6 Définition

Un séquent $\Gamma \vdash A$ est prouvable en logique propositionnelle intuitionniste si l'on peut en construire une preuve utilisant les règles de la logique minimale ainsi que les règles associées à la contradiction et la négation (ainsi que les règles dérivées).

On note $\Gamma \vdash_J A$ le jugement "$\Gamma \vdash A$ est prouvable en logique propositionnelle intuitionniste".

Notons que $\vdash_J$, à l'instar de $\vdash_M$, *n'est pas un connecteur*. On ne confondra pas le *jugement* $A \vdash_J B$ avec la *proposition* $A \to B$.

4.1.7 Peut-on prouver $\bot$?

Il n'y a pas de règle d'introduction pour la contradiction.

Un séquent de la forme $\Gamma \vdash \bot$ ne peut être prouvable que si les hypothèses de Γ contiennent cette contradiction, soit directement, soit par la présence d'une négation.

Exemple

$$\dfrac{\dfrac{\dfrac{}{P \to Q \to \bot, P, Q \vdash P \to Q \to \bot}\ \text{hyp} \qquad \dfrac{}{P \to Q \to \bot, P, Q \vdash Q}\ \text{hyp} \qquad \dfrac{}{P \to Q \to \bot, P, Q \vdash P}\ \text{hyp}}{P \to Q \to \bot, P, Q \vdash \bot}\ \text{mp}}{P \to \sim Q, P, Q \vdash \bot}\ \sim\text{-expansion}$$

Voici la même preuve, dans un autre format :

```
1 Supposons P → ∼Q
2 Supposons P
3 Supposons Q
4 P → Q → ⊥ [expansion de ∼, 1]
5 ⊥ [mp, 4, 2, 3]
```

4.1.8 Autour de la double négation

On remarquera que dans l'exercice 16 page précédente, on ne demande pas de prouver de règle de la forme :

$$\dfrac{\Gamma \vdash \sim\sim A}{\Gamma \vdash A}\ \text{double négation}_e$$

La raison en est encore un problème d'incomplétude. En effet, bien que pour toute proposition A et tout contexte Γ, le jugement $\Gamma \vdash \sim\sim A$ implique le jugement $\Gamma \vdash A$, la règle ci-dessus n'est pas dérivable en logique intuitionniste. On verra dans la section 4.4 que cette règle est dérivable dans le cadre de la logique dite *classique*.

4.2 Les connecteurs binaires

4.2.1 La conjonction

La conjonction permet, à partir de deux propositions A et B de former une seule proposition qui est vraie si et seulement si A et B sont vraies.

Syntaxe

On étend de nouveau la définition de l'ensemble des propositions (Section 2 page 39), en ajoutant la clause :

Définition 3. *Si A et B sont deux propositions, alors $(A \wedge B)$ est une proposition, prononcée "$(A$ et $B)$"*

On considèrera que $\wedge$ est un opérateur associatif à droite (comme $\rightarrow$) et de priorité supérieure à $\rightarrow$ et inférieure à $\sim$.

Par exemple, la proposition $\sim P \wedge Q \rightarrow R \wedge S$ doit se lire comme $((\sim P) \wedge Q) \rightarrow (R \wedge S)$. Une proposition de la forme $A \wedge B \wedge C$ se lira comme $A \wedge (B \wedge C)$.

Sémantique

A	B	$A \wedge B$
f	f	f
f	v	f
v	f	f
v	v	v

Règle d'introduction

Pour prouver un séquent $\Gamma \vdash A \wedge B$, il suffit de prouver séparément les séquents $\Gamma \vdash A$ et $\Gamma \vdash B$:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_i$$

Règle d'élimination

La règle d'élimination de la conjonction peut paraitre un peu déroutante, mais la pratique de nombreux exemples en fera saisir l'intuition. Elle exprime que, pour prouver une proposition C à partir de $A \wedge B$, il suffit de prouver C à partir de A et B *prises séparément*.

$$\frac{\Gamma \vdash A \wedge B \quad \Gamma, A, B \vdash C}{\Gamma \vdash C} \wedge_e$$

Dérouté.e ? Prenons comme exemple la validation de deux règles dérivées plus intuitives.

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge_1$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge_2$$

Montrons que $\wedge_1$ est bien une règle dérivée. La preuve pour $\wedge_2$ est laissée en exercice :

$$\dfrac{\Gamma \vdash A \wedge B \qquad \dfrac{}{\Gamma, A, B \vdash A}\ \text{hyp}}{\Gamma \vdash A}\ \wedge_e$$

Exercice 17. *Montrer les règles suivantes :*

$$\frac{}{A \wedge B \vdash B \wedge A}\ \textit{commutativité de } \wedge$$

$$\frac{}{(A \wedge B) \wedge C \vdash A \wedge (B \wedge C)}\ \textit{associativité de } \wedge$$

$$\frac{}{A \wedge \bot \vdash \bot}\ \textit{loi d'absorption de } \wedge$$

4.2.2 La disjonction

Reprenons le même plan que pour la conjonction.

Syntaxe

On étend de nouveau la définition de l'ensemble des propositions (Section 2 page 39), en ajoutant la clause :

Définition 4. *Si A et B sont deux propositions, alors $(A \vee B)$ est une proposition, prononcée "$(A$ ou $B)$"*

On considèrera que $\vee$ est un opérateur associant à droite et de priorité supérieure à $\rightarrow$ et inférieure à $\wedge$.

Par exemple, la proposition $\sim P \wedge Q \rightarrow R \vee S \wedge T$ doit se lire comme $((\sim P) \wedge Q) \rightarrow (R \vee (S \wedge T))$

Sémantique

A	B	$A \vee B$
f	***f***	***f***
f	***v***	***v***
v	***f***	***v***
v	***v***	***v***

Règles d'introduction

La disjonction possède *deux* règles d'introduction : Pour prouver un séquent $\Gamma \vdash A \vee B$, on peut soit prouver le séquent $\Gamma \vdash A$ soit prouver le séquent $\Gamma \vdash B$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_{i,1}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_{i,2}$$

Règle d'élimination

La règle d'élimination de la disjonction formalise la notion de *preuve par cas*. Si l'on veut prouver $\Gamma \vdash C$ et que l'on sait prouver $\Gamma \vdash A \vee B$ alors il suffit de prouver C en considérant les deux cas possibles

— C peut être prouvée en supposant A
— C peut être prouvée en supposant B

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C} \vee_e$$

Sous un autre format, l'élimination de la disjonction se présente ainsi :

```
1  …
2  A ∨ B
3  …
4  { Supposons A
5     ...
6     C
7  }
8  { Supposons B
9     ...
10    C
11 }
12 C [∨e, 2, 4 − 6, 7 − 9]
```

Exemple

Voici une preuve, dans les deux formats, du séquent $A \vee \bot \vdash A$:

$$\dfrac{\dfrac{}{A \vee \bot \vdash A \vee \bot}\ \text{hyp} \qquad \dfrac{}{A \vee \bot, A \vdash A}\ \text{hyp} \qquad \dfrac{\dfrac{}{A \vee \bot, \bot \vdash \bot}\ \text{hyp}}{A \vee \bot, \bot \vdash A}\ \bot_e}{A \vee \bot \vdash A}\ \vee_e$$

```
1 Supposons A ∨ ⊥
2 { supposons A
3    A [hyp,2]
```

```
}
{ supposons ⊥
   A [⊥e,4]
}
A [∨e,1,2-3,5-6]
```

Exemple

Voici la preuve de la règle de "commutativité de $\vee$"

$$\dfrac{\dfrac{}{A \vee B \vdash A \vee B}\ \text{hyp} \qquad \dfrac{\dfrac{}{A \vee B, A \vdash A}\ \text{hyp}}{A \vee B, A \vdash B \vee A}\ \vee i, 2 \qquad \dfrac{\dfrac{}{A \vee B, B \vdash B}\ \text{hyp}}{A \vee B, B \vdash B \vee A}\ \vee i, 1}{A \vee B \vdash B \vee A}\ \vee_e$$

Voici la même preuve sous un autre format :

```
Supposons A ∨ B
{ supposons A
   A [hyp,2]
   B ∨ A [∨i,2,3]
}
{ supposons B
   B [hyp,6]
   B ∨ A [∨i,1,7]
}
B ∨ A [∨e,1,2-4,6-8]
```

Exercice 18. *Prouver les règles dérivéees suivantes :*

$$\dfrac{\Gamma \vdash A \vee B \qquad \Gamma \vdash \sim B}{\Gamma \vdash A}$$

$$\dfrac{}{\sim A \vee B \vdash A \rightarrow B}$$

$$\dfrac{}{\sim(A \vee B) \vdash \sim A \wedge \sim B}\ \textit{De Morgan}_1$$

4.2.3 L'équivalence logique

De même que pour la négation, nous allons considérer l'équivalence logique comme une abréviation plutôt qu'un nouveau connecteur.

Définition 5. *Si A et B sont deux propositions, alors $(A \leftrightarrow B)$ est une abréviation de $((A \rightarrow B) \wedge (B \rightarrow A))$.*

L'opérateur $\leftrightarrow$ sera d'une priorité équivalente à celle de $\rightarrow$. Il conviendra de mettre suffisamment de parenthèses dans toute formule comportant à la fois ces deux connecteurs.

On peut dériver les règles suivantes :

$$\frac{\Gamma \vdash A \to B \quad \Gamma \vdash B \to A}{\Gamma \vdash A \leftrightarrow B} \leftrightarrow_i$$

$$\frac{\Gamma \vdash A \leftrightarrow B}{\Gamma \vdash A \to B} \leftrightarrow_e$$

$$\frac{\Gamma \vdash A \leftrightarrow B}{\Gamma \vdash B \to A} \leftrightarrow_e$$

$$\frac{\Gamma \vdash A \leftrightarrow B}{\Gamma \vdash B \leftrightarrow A} \leftrightarrow\text{-sym}$$

4.2.4 Équivalence entre propositions

Soient A et B deux formules. On notera $A \equiv_J B$ le *jugement* $\vdash_J A \leftrightarrow B$.

Par exemple, la loi de De Morgan de l'exercice précédent s'exprime par le jugement $\sim(A \vee B) \equiv_J \sim A \wedge \sim B$.

Notons que $\equiv$ *n'est pas un connecteur*. On ne confondra pas le *jugement* $A \equiv_J B$ avec la *proposition* $A \leftrightarrow B$.

Exercice 19. *Montrer les jugements suivants (qu'on pourra par la suite utiliser comme règles dérivées) :*

$$A \wedge \bot \equiv_J \bot \tag{4.4}$$
$$A \vee \bot \equiv_J A \tag{4.5}$$
$$A \vee B \equiv_J B \vee A \tag{4.6}$$
$$A \wedge B \equiv_J B \wedge A \tag{4.7}$$
$$A \vee (B \vee C) \equiv_J A \vee B \vee C \tag{4.8}$$
$$A \wedge (B \wedge C) \equiv_J A \wedge B \wedge C \tag{4.9}$$
$$A \wedge (B \vee C) \equiv_J A \wedge B \vee A \wedge C \tag{4.10}$$
$$A \vee B \wedge C \equiv_J (A \vee B) \wedge (A \vee C) \tag{4.11}$$
$$\sim(A \to B) \equiv_J (A \wedge \sim B) \tag{4.12}$$
$$\sim(A \vee B) \equiv_J \sim A \wedge \sim B \text{ [loi de De Morgan]} \tag{4.13}$$
$$A \to \sim A \vdash_J \sim A \text{ [reductio ad absurdum]} \tag{4.14}$$
$$\sim A \to A \vdash_J \sim\sim A \tag{4.15}$$

4.3 Quelques méta-théorèmes

4.3.1 Cohérence et non-complétude

Les propriétés suivantes (que nous admettrons pour la plupart) permettent d'exprimer la cohérence de la logique propositionnelle intuitionniste.

Tout d'abord, nous avons un résultat similaire au thèorème 1 page 30

Méta-théorème 4 (Correction)**.** *Si* $\Gamma \vdash_J A$*, alors* $\Gamma \vDash A$

Nous en déduisons (par contraposition) le corollaire suivant :

Méta-théorème 5 (non-contradiction)**.** *Le séquent* $\vdash \bot$ *n'est pas démontrable en logique propositionnelle intuitionniste.*

En effet, le séquent $\vdash \bot$ n'est clairement pas valide, ce qui entraîne la non-prouvabilité de ce séquent.

Ce méta-théorème entraîne l'impossibilité de prouver une proposition *et* sa négation.

Méta-théorème 6. *Il n'existe aucune proposition* A *telle que* $\vdash_J A$ *et* $\vdash_J \sim A$*.*
Preuve laissée en exercice

Exercice 20. *L'énoncé suivant est-il vrai ou faux ? Pourquoi ?*

> *" Il n'existe aucun séquent* $\Gamma \vdash A$ *tel qu'on ait à la fois* $\Gamma \vdash_J A$ *et* $\Gamma \vdash_J \sim A$*."*

Remarque

Le théorème de correction nous donne un moyen de vérifier qu'un séquent $\Gamma \vdash A$ n'est pas prouvable : il suffit de montrer qu'il n'est pas valide.

Non-complétude

Nous donnons ci-dessous une liste de séquents valides (au sens sémantique) mais non prouvables en logique propositionnelle intuitionniste (nous admettrons ce dernier résultat).

$$\vdash A \vee \sim A \text{ [tiers exclu]} \tag{4.16}$$
$$\sim\sim A \vdash A \text{ [élimination de la double négation]} \tag{4.17}$$
$$\vdash \sim(A \wedge B) \leftrightarrow \sim A \vee \sim B \text{ [loi de de Morgan]} \tag{4.18}$$
$$\vdash \sim(A \to B) \leftrightarrow (A \wedge \sim B) \tag{4.19}$$
$$\vdash (A \to B) \leftrightarrow (\sim B \to \sim A) \tag{4.20}$$

4.3.2 Relation avec la logique minimale

La logique propositionnelle intuitionniste est une extension de la logique minimale, dans la mesure où toute règle d'inférence de la logique minimale est aussi une règle de la logique propositionnelle intuitionniste. Toute preuve en logique minimale peut alors être convertie ("castée") en une preuve de la logique propositionnelle intuitionniste :

Méta-théorème 7. *Si* $\Gamma \vdash_M A$*, alors* $\Gamma \vdash_J A$

Méta-théorème 8. *Toutes les règles dérivées de la logique minimale restent applicables dans la logique propositionnelle intuitionniste.*

Exercice 21. *Donner une idée de preuve de l'énoncé ci-dessus.*

4.3.3 Remplacement d'une sous-formule

Méta-théorème 9. *Si* $\Gamma \vdash_J A \leftrightarrow B$ *et* $\Gamma \vdash_J C$*, alors* $\Delta \vdash_J D$*, où* $Delta \vdash_J D$ *est obtenue en remplaçant dans* Γ *et* C *une ou plusieurs occurrences de* A *par* B*.*

Exercice 22. *Donner une idée de preuve de l'énoncé ci-dessus.*

Exemple

$$\dfrac{\dfrac{\dfrac{}{\vdash (P \vee Q) \leftrightarrow (Q \vee P)}}{P \vee Q, \sim P \vdash (P \vee Q) \leftrightarrow (Q \vee P)}\ \text{aff} \qquad P \vee Q, \sim P \vdash Q}{Q \vee P, \sim P \vdash Q}$$

Remarques

Dans l'énoncé du théorème 9, le contexte des trois séquents est le même. En fait, on peut relâcher un peu cette condition.

Méta-théorème 10. *Si* $\Delta \vdash_J A \leftrightarrow B$ *et* $\Gamma \vdash_J C$*, avec* $\Delta \subseteq \Gamma$*, alors* $\Gamma \vdash_J D$*, où* D *est obtenue en remplaçant dans* C *et* Γ *une ou plusieurs occurrences de* A *par* B*.*

Méta-théorème 11. *En particulier, si* $A \equiv_J B$*, et si* $\Gamma \vdash_J C$*, alors* $\Gamma \vdash_J D$*, où* D *est obtenue en remplaçant dans* C *et* Γ *une ou plusieurs occurrences de* A *par* B*.*

Exercice 23. *Pouvez-vous justifier ces variantes ?*

Exercice 24. *Inventez un exemple où le non-respect de la condition* $\Delta \subseteq \Gamma$ *entraîne la dérivation d'un séquent absurde.*

4.4 La logique propositionnelle classique

Nous avons vu en 4.3.1 page précédente une liste de schémas "classiques" que la logique propositionnelle intuitionniste ne permet pas de prouver.

Définition 6 (Logique classique)**.** *La logique propositionnelle classique s'obtient en ajoûtant à la logique propositionnelle intuitionniste la règle du tiers exclu*[1] *:*

$$\dfrac{}{A \vee \sim A}\ exm$$

1. "Excluded middle" en anglais

Nous noterons $\Gamma \vdash_K A$ le jugement "Le séquent $\Gamma \vdash A$ est prouvable en logique propositionnelle classique".

4.4.1 Règles dérivées pour la logique classique

Tous les séquents de la liste donnée en 4.3.1 page 49 deviennent prouvables en logique classique. Voici par exemple comment dériver l'élimination de la double négation :

$$\dfrac{\dfrac{}{\sim\sim A \vdash A \vee \sim A}\ \text{exm + aff} \qquad \dfrac{}{\sim\sim A, A \vdash A}\ \text{hyp} \qquad \dfrac{\dfrac{\dfrac{}{\sim\sim A, \sim A \vdash \sim\sim A}\ \text{hyp} \qquad \dfrac{}{\sim\sim A, \sim A \vdash \sim A}\ \text{hyp}}{\sim\sim A, \sim A \vdash \bot}\ \text{absurde}}{\sim\sim A, \sim A \vdash A}\ \bot_e}{\sim\sim A \vdash A}\ \vee_e$$

```
1  Supposons ∼∼ A
2  A ∨ ∼ A [exm]
3  { supposons A
4     A [hyp,3]
5  }
6  { supposons ∼ A
7     A [absurde, 1, 6]
8  }
9  A [∨e,2,3-4,6-7]
```

Exercice 25. *Prouver les autres régles dérivéees de 4.3.1 page 49*

4.5 Quelques méta-théorèmes pour la logique classique

4.5.1 Cohérence et complétude

Méta-théorème 12 (Correction)**.** *Si $\Gamma \vdash_K A$, alors $\Gamma \vDash A$*

Nous en déduisons (par contraposition) le corollaire suivant :

Méta-théorème 13 (non-contradiction)**.** *Le séquent $\vdash \bot$ n'est pas démontrable en logique classique.*

En effet, le séquent $\vdash \bot$ n'est clairement pas valide, ce qui entraîne la non-prouvabilité de ce séquent.

Ce méta-théorème entraîne l'impossibilité de prouver une proposition *et* sa négation.

Méta-théorème 14. *Il n'existe aucune proposition A telle que $\vdash_K A$ et $\vdash_K \sim A$.*
Preuve laissée en exercice

Remarque

Le thèorème de correction nous donne un moyen de vérifier qu'un séquent $\Gamma \vdash A$ n'est pas prouvable : il suffit de montrer qu'il n'est pas valide.

Méta-théorème 15 (Complétude)**.** *Si* $\Gamma \vDash A$ *alors* $\Gamma \vdash_K A$*.*

Preuve longue, assez technique, et ennuyeuse. On en trouve plein de versions sur Internet.

Équivalence en logique classique

On notera $A \equiv_K B$ le jugement $\vdash_K A \leftrightarrow B$.
Il est clair que si l'on a $A \equiv_J B$, alors $A \equiv_K B$.
Dans la mesure où le contexte — intuitionniste ou classique — est clair, on pourra simplifier en la notation $A \equiv B$.

4.5.2 Relation avec la logique intuitionniste

La logique propositionnelle classique est une extension de la logique intuitionniste. Toute preuve en logique propositionnelle intuitionniste peut être convertie ("castée") en une preuve de la logique propositionnelle classique :

Méta-théorème 16. *Si* $\Gamma \vdash_J A$*, alors* $\Gamma \vdash_K A$

Méta-théorème 17. *Toutes les règles dérivées de la logique intuitionniste restent applicables dans la logique propositionnelle classique.*

Exercice 26. *Donner une idée de preuve de l'énoncé ci-dessus.*

Méta-théorème 18. *Le théorème de substitution uniforme reste applicable en logique classique.*

4.5.3 Quelques règles dérivées en logique classique

Attention ! Les règles de cette section ne sont correctes qu'en logique classique et sont donc à éviter en logique propositionnelle intuitionniste.

Raisonnement par l'absurde classique

$$\frac{\Gamma, \sim A \vdash \bot}{\Gamma \vdash A}\ \text{absurde classique}$$

Exercice 27. *Montrer que cette règle est valide.*

Elimination du tiers-exclu

$$\frac{\Gamma, B \vdash A \qquad \Gamma, \sim B \vdash A}{\Gamma \vdash A}\ \mathrm{exm}_e$$

Exercice 28. *Montrer que cette règle est valide.*

Exercice 29. *Valider la règle dérivée ci-dessous :*

$$\frac{\Gamma, B \vdash A \qquad \Gamma \vdash \sim A}{\Gamma \vdash \sim B}$$

Exercice 30. *Montrer le théorème suivant :*

Méta-théorème 19. *Soient Γ un contexte et A une proposition, alors $\Gamma \vdash_K A$ si et seulement si $\Gamma \cup \{\sim A\}$ n'est pas satisfaisable.*

Exercice 31. *Prouver le jugement suivant*

$$\sim A \to A \vdash_K A$$

Aide : *On pourra d'abord prouver le jugement*

$$\sim A \to A \vdash_J \sim\sim A$$

puis utiliser l'élimination de la double négation.

Exercice 32. *Prouver le jugement suivant*

$$\vdash_K (A \to B) \vee (B \to A)$$

Remarque : *Nous avons là un exemple de jugement prouvable en logique classique, donc valide, mais dont l'intuition nous échappe.*

Que veut dire, et surtout comment utiliser l'instance suivante ?

$$\vdash_K (x = 2 \to x = 3) \vee (x = 3 \to x = 2)$$

Exercice 33. *Soient P et Q deux propositions atomiques.*

1. *Montrer l'équivalence $(\sim P \to P) \equiv_J \sim\sim P$.*
2. *En déduire l'équivalence $(\sim P \to P) \to P \equiv_J \sim\sim P \to P$.*
3. *(*) En déduire que si la formule de Peirce était prouvable en logique intuitionniste, alors on pourrait prouver l'élimination de la double négation c'est à dire prouver tout jugement de la forme $\sim\sim A \vdash_J A$. On pourra utiliser le thèorème de substitution uniforme.*
4. *Donner une preuve en logique classique de la formule de Peirce :*

$$((P \to Q) \to P) \to P$$

5. *Donner une preuve en logique intuitionniste du séquent suivant :*

$$P \vee \sim P \vdash ((P \to Q) \to P) \to P$$

6. *(*) Montrer que si l'on ajoute à la logique intuitionniste la règle d'élimination de la double négation :*

$$\frac{\Gamma \vdash \sim\sim A}{\Gamma \vdash A}\ \textit{double negation}_e$$

   Alors, on peut en dériver la règle du tiers exclu.

7. *Que peut-on en conclure sur le tiers-exclu, la loi de la double négation et la formule de Peirce ?*

Chapitre 5

Calcul des prédicats (Logique du premier ordre)

5.1 Introduction

Dans les chapitres précédents, nous n'avons vu que des formules très abstraites, engendréees à partir de simples variables propositionnelles. Cette abstraction nous a permis d'étudier les règles et structures de raisonnement sans être influencés par l'application à des domaines précis.

Nous allons procéder à une extension des logiques propositionnelles (intuitionniste et classique) dans deux directions, qui nous autoriseront à présenter des exemples plus réalistes :

— Remplacement des variables propositionnelles par des formules plus complexes composées à partir d'objets, de fonctions, de prédicats et de relations

— Introduction des deux *quantificateurs* "pour tout" et "il existe au moins"

Les règles (de base et dérivées) de la logique propositonnelle seront encore applicables. Quant aux méta-théorèmes, nous examinerons un par un s'il sont encore applicables dans ce nouveau cadre.

5.2 Formules du premier ordre

5.2.1 Types

L'étudiant en informatique a déjà rencontré la notion de type dans le cadre des langages de programmation. Cette notion permet de contrôler que les expressions apparaissant dans un programme sont bien formées : dans le sens où une fonction ne peut être appliquée qu'à des arguments du type prévu. Par exemple la fonction **C** `itoa` ne doit être appliquée qu'à un argument de type `int` et renvoie un résultat de type `char *`. On dispose d'une déclaration similaire pour `strlen`.

On en déduit que si `i` est une variable de type `int`, alors l'expression strlen (atoi (i)) est bien formée et de type `int`.

Syntaxe des types

On pourra considérer suivant les exemples des *types atomiques* qui seront des symboles déclarés comme tels. En voici quelques exemples, que nous reprendrons dans nos exemples :

bool Le type des valeurs booléennes

nat Le type des entiers naturels

Z Le type des nombres entiers

int Le type des entiers sur 32 bits (int31 en *Coq*)

char Le type des caractères

unit Le type singleton (contenant une seule valeur)

Empty_set Le type vide

individu Un type pour les syllogismes à la Aristote

5.2.2 Constantes et symboles de fonctions

Les constantes sont des identificateurs à la base de la construction des *termes*, les expressions dont le calcul des prédicats permet d'étudier les propriétés. À chaque constante c sera associé un type unique. On utilisera la notation de *typage* $c : A$ pour exprimer que A est le type de c

Les constantes sont en fait un cas particulier de *symboles de fonctions*. À chaque symbole de fonction est associé un type de la forme $A_1 \times A_2 \times \cdots \times A_k \rightarrow A$ ou les A_i et A sont des types atomiques.

Ce type se veut le type des fonctions totales qui prennent leurs arguments dans les types $A_1, \ldots, A_n$et retournent un résultat de type A. Si $k = 0$, on retrouve la notion de constante.

Remarque

Dans nombre d'ouvrages sur la logique mathématique, on ne considère qu'un seul type de base. Dans ce cas, un symbole de fonction est caractérisé par son *arité*, c'est à dire son nombre d'arguments. Nous avons choisi de donner une définition plus proche des langages de programmation.

Exemples

Nous donnons quelques exemples de constantes et symboles de fonctions qui seront utilisés plus loin :

— true : bool
— false : bool
— xor : bool × bool →bool

— O : nat *le nombre* 0
— S : nat → nat *(la fonction successeur)*
— + : nat × nat →nat
— * : nat × nat →nat
— 'a' : char
— Socrate : individu

Remarques

1. On ne confondra pas la constante false avec la valeur de vérité ***f***, ni avec la contradiction ⊥. Ce sont des objets de nature totalement différente :
   — false est une constante associée au type bool
   — ***f*** est une valeur de vérité, du domaine de la sémantique
   — ⊥ est une propositon logique
   De même pour true à ne pas confondre avec ***v***.
2. **À déplacer** On s'autorisera les abréviations suivantes : 0 pour O, 1 pour S O, 2 pour S (S O), etc.

5.2.3 Variables, Déclarations et Contextes

Afin de rester le plus compatibles possible avec *Coq*, nous allons légèrement modifier les notations de contextes et de séquents vus précédemment, le but étant d'unifier le traitement des hypothèses et des déclarations de variables.

Nous considérons dans toute la suite un ensemble infini de symboles appelés *variables*. En utilisant une politique de nommage appropriée (comme dans les bonnes règles de programmation) on s'arrangera à éviter toute confusion avec les noms de constantes, de fonctions ou de prédicats.

Définition 7 (Contexte)**.** *Un contexte est un ensemble de déclarations d'une des deux formes suivantes :*
— *Une déclaration de variable* $v : A$ *où* A *est un type.*
— *Une hypothèse* $H : A$ *où* A *est une proposition et* H *un identificateur.*
On suppose qu'un contexte ne contient pas deux déclarations différentes de la même variable ou du même nom d'hypothèse.

Remarque

Afin de rester compatible avec les notations usuelles et les chapitres précédents, on pourra continuer à ne pas nommer les hypothèses. En revanche, comme on l'a vu sur machine, le logiciel *Coq* demande que chaque hypothèse soit nommée.

5.2.4 Termes

Comme dans tout langage de programmation typé, l'écriture d'expressions correctes est guidée par des conventions syntaxiques et l'application de *règles*

de typage. Ces règles permettent de construire des *jugements de typage*, de la forme $\Gamma \vdash t : A$ qui se lisent " dans le contexte Γ, le terme t a pour type A".

5.2.5 Règles de typage

Les deux premières règles permettent de construire des termes de base, soit à partir d'une constante, soit à partir d'une variable déclarée dans le contexte considéré. La troisième permet de construire des termes plus complexes, en appliquant des fonctions à des arguments du type approprié.

Constantes

$$\frac{}{\Gamma \vdash c : A} \ (\ c \text{ est une constante de type } A)$$

Variables

$$\frac{}{\Gamma \vdash v : A} \ ((\mathrm{v} \ : \mathrm{A}) \in \Gamma)$$

Application d'une fonction

$$\frac{f : A_1 \times A_2 \times \cdots \times A_k \to A \quad \Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2 \quad \ldots \quad \Gamma \vdash t_k : A_k}{\Gamma \vdash (f(t_1, t_2, \ldots, t_k) : A}$$

Exemple

Reprenons les symboles de fonctions de la section 5.2.2 page 56. Soit un contexte Γ contenant deux variables i : nat et j : nat. Alors nous obtenons le jugement de typage $\Gamma \vdash *(i, (+(j, S(S(0))))) : nat$.

Exercice 34. *Vérifier l'exemple précédent règle par règle. Construire d'autres exemples simples.*

Remarque

Afin d'alléger les notations, nous conviendrons d'écrire i+j au lieu de +(i,j), i*j au lieu de *(i,j), 1 au lieu de S(0), 2 au lieu de S(S(0)), etc.

Ainsi notre exemple précédent peut-il s'écrire $\Gamma \vdash i * (j + 2) : nat$.

5.2.6 Prédicats et Relations

Les symboles de prédicats et de relations nous permettent de construire des propositions atomiques énonçant des propriétés des termes.

Un *symbole de prédicat* est un identificateur R (différent des symboles de constantes et de fonction), auquel on associe un type d'argument A.

Les symboles de prédicats sont des cas particuliers de *symboles de relation*, auxquels on associe un type de la forme $A_1 \times A_2 \times \cdots \times A_k$.

Notation

Nous proposons les notations suivantes (cohérentes avec *Coq*) pour déclarer le type des arguments d'un symbole de prédicat ou relation :

$$\mathrm{R} : A \to \mathbf{Prop}$$

et

$$\mathrm{R} : A_1 \times A_2 \cdots \times A_k \to \mathbf{Prop}$$

Cette notation revient à attribuer un type spécial aux propositions, ce qui permet de simplifier l'écriture de la phrase " P est une proposition "en "$P : \mathbf{Prop}$".

Exemples

Reprenons les types utilisés dans l'exemple 5.2.2 page 56.

Nous pouvons considérer des symboles de prédicat associés aux propriétés sur les entiers naturels : "être nul", "être positif", et un symbole de relation binaire "être strictement inférieur à" :

— $\mathrm{nul} : \mathrm{nat} \to \mathbf{Prop}$
— $\mathrm{positif} : \mathrm{nat} \to \mathbf{Prop}$
— $< : \mathrm{nat} \times \mathrm{nat} \to \mathbf{Prop}$

dans le même contexte que l'exemple 5.2.5 page précédente, nous pouvons former les propositions suivantes :

— $i < i + 1$ (autrement dit $< (i, i + 1)$
— $\mathrm{nul}(i) \vee \mathrm{positif}(i)$
— $0 < i \leftrightarrow \mathrm{positif}(i)$

L'égalité

On considère un symbole de relation binaire $=$ *polymorphe* : Pour tout type atomique A, et tout couple de termes t et t' *du même type* A, la proposition $t = t'$ est bien formée.

On considérera aussi $t \neq t'$ comme une abréviation de $\sim(t = t')$.

Notons que "$1 \neq \texttt{true}$" n'est pas une proposition, car l'égalité n'a pas de sens entre deux termes de type différent. Or autoriser l'écriture de "$1 \neq \texttt{true}$" reviendrait à autoriser cellle de "$\sim(1 = \texttt{true})$" qui contiendrait comme sous-formule "$1 = \texttt{true}$".

Définition 8 (Signature)**.** *En logique du premier ordre (calcul des prédicats) on appelle signature la donnée d'un ensemble de types, de symboles de constantes, de fonctions, et de relations (avec leur type). On supposera toujours que les symboles utilisés pour désigner les types, constantes, fonctions, relations sont deux à deux disjoints, afin d'éviter toute ambiguïté.*

5.2.7 Syntaxe des propositions

Les constructions de propositions vues en logique propositionnelle ont besoin d'être étendues.

Tout d'abord, les propositions atomiques, au lieu d'être réduites aux variables propositionnelles, sont maintenant formées à partir des termes et des symboles de relations (y compris l'égalité).

Les constructions à partir de connecteurs des chapitres précédents restent valables. Il nous reste à introduire les *quantificateurs* pour obtenir la logique des prédicats.

Définition 9 (Quantifications)**.** *Soit v une variable , A un type, et P une proposition. Les formules $\forall v : A,\ P$ et $\exists v : A,\ P$ sont des proposition. Les symboles $\forall$ et $\exists$ sont appelés respectivement quantificateur universel et quantificateur existentiel.*

On considère que les quantifications ont une priorité plus faible que tous les connecteurs. Par exemple la proposition "$\forall i : nat,\ positif(i) \vee i = 0$" se lit "$\forall i : nat,\ (positif(i) \vee i = 0)$".

On peut utiliser des parenthèses pour modifier cette priorité comme dans la formule suivante :

$$\forall i : nat,\ (\exists j : nat,\ j < i) \leftrightarrow 0 < i$$

Remarques

Dans le cas d'une signature à un seul type A, ou si le type des variables est rendu clair par le contexte, on pour omettre l'indication de type et noter "$\forall v,\ P$" et "$\exists v,\ P$" au lieu de "$\forall v : A,\ P$" et "$\exists v : A,\ P$".

De même, on pourra abréger des quantifications imbriquées de même nature et portant sur le même type en écrivant par exemple "$\forall v\, w\, x : A,\ P$" au lieu de "$\forall v : A, \forall w : A, \forall x : A,\ P$".

5.2.8 Variables libres et variables liées

Dans une formule de la forme "$\forall v : A,\ P$" ou "$\exists v : A,\ P$", on appelle P la *portée* de la quantification sur x. Par exemple, dans la formule vue précédemment :

$$\forall i : \mathrm{nat},\ (\exists j : \mathrm{nat},\ j < i) \leftrightarrow 0 < i$$

la portée de la quantification sur i est l'expression "$(\exists j : \mathrm{nat},\ j < i) \leftrightarrow 0 < i$" et la portée de la quantification sur j est l'expression "$j < i$".

On dit qu'une variable est *liée* dans une formule si elle apparait dans la portée d'un quantificateur de cette formule, *libre* sinon.

Par exemple, soit $\mathrm{i} : \mathrm{nat}$, et la formule "$(\exists j : \mathrm{nat},\ j < i) \leftrightarrow 0 < i$". La variable i est libre dans cette proposition alors que j est liée (par le quantificateur existentiel).

La figure 5.1 page suivante visualise par une flèche la liaison entre chaque occurrence de variable liée et *son* quantificateur.

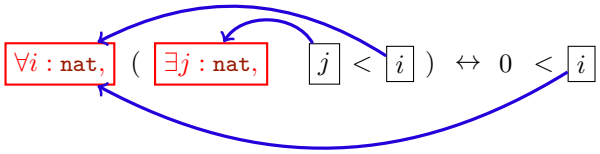

FIGURE 5.1 – géometrie des liaisons(1)

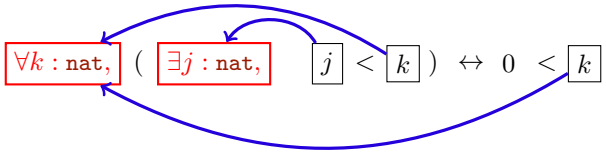

FIGURE 5.2 – géometrie des liaisons : résultat d'une α-conversion

On appelle α-conversion la transformation consistant à renommer toutes les variables liées à un même quantificateur *sans modifier la géométrie des liaisons*.

La figure 5.2 montre le résultat d'une α-conversion sur la formule précédente : la variable liée i a été renommée en k. On voit clairement que la structure des liaisons n'a pas été modifiée par rapport à la figure 5.1

En revanche si l'on renomme la même variable en j, on voit que la structure de la formule a considérablement changé (figure 5.3). En effet la deuxième occurrence de la variable j a été "capturée" par la quantification existentielle.

On considère que des formules égales à α-conversion près sont identiques. C'est le cas entre les formules des figures 5.1 et 5.2. En revanche, les figures 5.2 et 5.3 sont vraiment différentes.

On évitera d'avoir des variables à la fois libres et liées dans une même formule, par exemple comme dans " $(\exists\, i : \text{nat},\, i < j) \vee i = 2$ ". Dans ce cas, il est conseillé de procéder à une α-conversion, ce qui donnerait par exemple $(\exists\, k : \text{nat},\, k < j) \vee i = 2$ ". On appelle *polie* une formule dans laquelle aucune variable n'est à la fois libre et liée, et dans laquelle aucune variable liée n'est liée dans deux quantifications différentes.

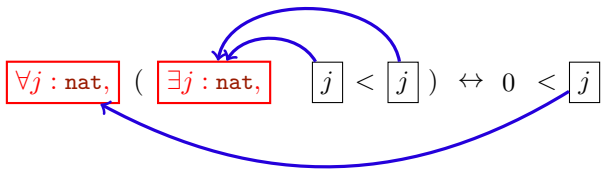

FIGURE 5.3 – géometrie des liaisons : mauvaise α-conversion

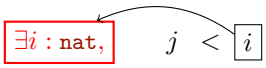

FIGURE 5.4 – phénomène de capture : avant

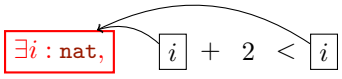

FIGURE 5.5 – phénomène de capture : après

On qualifie de *close* toute formule qui ne contient aucune variable libre. Par exemple "$\forall\, i : \texttt{nat},\ \exists\, j : \texttt{nat},\ i < j$" est close.

En revanche, une formule qui contient une ou plusieurs occurrences d'une variable v exprime une propriété de v. Par exemple soit $\texttt{n} : \texttt{nat}$. La proposition "$\exists\, p : \texttt{nat},\ n = p \times 2$" exprime la propriété "$n$ est pair".

De même pour une formule avec plusieurs variables libres : la formule "$\exists\, d : \texttt{nat},\ n = d + p$" exprime la relation "$n \leq p$".

5.2.9 Substitution

Soit P une formule, une variable $v : A$ et un terme $t : A$ (dans un contexte donné). On note $F[v \leftarrow t]$ le résultat de la substitution dans F de chaque occurrence libre de v par le terme t, *à condition qu'aucune variable de t ne devienne liée dans F*

Remarques

Dans la définition précédente, on supposera que le terme t ne contient aucune variable qui pourrait être "capturée" par une quantification.

Par exemple, prenons la proposition "$\exists\, i : \texttt{nat},\ j < i$"

Si nous avons dans le contexte considéré, la déclaration $\texttt{i} : \texttt{nat}$ le remplacement de j par $i + 2$ nous donnerait "$\exists\, i : \texttt{nat},\ i + 2 < i$".

Dans ce cas, on commencera par renommer la variable liée i, ce qui donne "$\exists\, k : \texttt{nat},\ j < k$", puis on opère le remplacement de j par $i + 2$, ce qui donne "$\exists\, k : \texttt{nat},\ i + 2 < k$" (voir Figure 5.6).

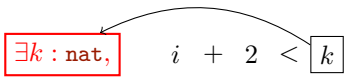

FIGURE 5.6 – substitution correcte de j par $i+2$ dans la formule de la figure 5.4

5.2.10 Substitution simultanée

On étend la notion de substitution au cas de plusieurs variables libres distinctes 2 à 2 : Notation $F[v_1 \leftarrow t_1; v_2 \leftarrow t_2; \ldots; v_n \leftarrow t_n]$

Exercice 35. *Donner une définition formelle de cette opération.*

5.2.11 Le coin du formaliste

On peut trouver une définition précise de la substitution, par exemple dans https ://www.lri.fr/∼paulin/Logique/html/cours005.html

1. On commence par définir la substitution de v par t *dans un terme* t' par récurrence sur ce terme.
   — Si c est une constante, alors $c[v \leftarrow t] = c$
   — Si w est une variable distincte de v, alors $w[v \leftarrow t] = w$
   — $v[v \leftarrow t] = t$
2. Si la formule P est atomique, la substitution se définit sans problème : $R(t_1, \ldots, t_n)[v \leftarrow t] = R(t_1[v \leftarrow t], \ldots, t_n[v \leftarrow t]$
3. Les connecteurs propositionnels de posent aucun problème :
   — $\bot[v \leftarrow t] = \bot$
   — $(\sim P)[v \leftarrow t] = \sim(P[v \leftarrow t]$
   — $(P \vee Q)[v \leftarrow t] = P[v \leftarrow t] \vee Q[v \leftarrow t]$
   — etc.
4. Les règles de substitution pour les quantificateurs doivent tenir compte des problèmes de capture de variables :
   Soit P une formule et w une variable :
   — Si $w = v$, alors $(\forall w : A, P)[v \leftarrow t] = (\forall w : A, P)$ et $(\exists w : A, P)[v \leftarrow t] = (\exists w : A, P)$
   — Si $w \neq v$ et *w n'a pas d'occurrence dans t*, alors

$$(\forall w : B, P)[v \leftarrow t] = \forall w : B, P[v \leftarrow t]$$

   et

$$(\exists w : B, P)[v \leftarrow t] = \exists w : B, P[v \leftarrow t]$$

5.3 Preuves en logique du premier ordre

On a tous les outils pour définir ce qu'est une preuve dans le calcul des prédicats. Il suffit d'ajouter aux règles de la logique propositionnelle des régles pour l'égalité et les deux quantificateurs. Suivant qu'on admet ou non la règle du tiers-exclu, on obtient une logique du premier ordre classique ou intuitionniste.

5.3.1 Règles associéees à l'égalité

Règle d'introduction de l'égalité

Soit t un terme de type A. La règle suivante applique la réflexivité de l'égalité au terme t.

$$\frac{}{\Gamma \vdash t = t} =_i \quad (\Gamma \vdash t : A)$$

Par exemple nous pouvons montrer le théorème suivant :

$$\frac{}{\vdash 36 = 36} =_i$$

On admet une légère extension de $=_i$, utile dans le raisonnement sur des calculs (par exemple les expressions d'un langage de programmation) :

$$\frac{}{\Gamma \vdash t = t'} =_i \quad (\Gamma \vdash t : A) \quad t \text{ et } t' \text{ ont la même valeur}$$

Ce qui nous permet d'obtenir le résultat suivant :

$$\frac{}{\vdash 6 \times 6 = 9 \times 4} =_i$$

Règle d'élimination de l'égalité

$$\frac{\Gamma \vdash t = t' \qquad \Gamma \vdash P[x \leftarrow t]}{\Gamma \vdash P[x \leftarrow t']} =_e$$

commentaires La variable x (libre dans P) dans cette règle sert uniquement à *marquer* les emplacements où l'on remplace le terme t par t'. Bien sûr le nom x utilisé dans la présentation de la règle peut être remplacé par n'importe quel nom non-utilisé par ailleurs. Cette variable est souvent laissée implicite dans les preuves.

Exemple

Voici une preuve de symétrie de l'égalité (qui devient donc une règle dérivée) : (a et b sont des termes d'un type A).

$$\frac{\Gamma \vdash a = b \qquad \dfrac{}{\Gamma \vdash a = a} =_i}{\Gamma \vdash b = a} =_e$$

Si l'on veut détailler l'application de la règle $=_e$, il suffit d'écrire $a = a$ comme $(x = a)[x \leftarrow a]$. Du coup, $(x = a)[x \leftarrow b]$ est bien la proposition $b = a$.

Exercice 36. *En utilisant les règles associées à l'égalité, prouver la règle de transitivité de l'égalité.*

$$\frac{\Gamma \vdash a = b \qquad \Gamma \vdash b = c}{\Gamma \vdash a = c}$$

5.3.2 Règles associées au quantificateur universel

Règle d'élimination

Soit A un type ; dans un contexte où l'on peut prouver la proposition $\forall\, x : A,\ P$ on peut en déduire une *instance* en instanciant la variable liée v par un terme de type A.

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash \forall\, x : A, P}{\Gamma \vdash P[x \leftarrow t]}\ \forall_e, t$$

Exemple

$$\frac{\dfrac{}{\Gamma \vdash 23 : \texttt{nat}} \qquad \Gamma \vdash \forall i : \texttt{nat}, i < i + 1}{\Gamma \vdash 23 < 23 + 1}\ \forall_e, 23$$

Ajouter l'exemple sur Socrate

Donner une version complète, puis l'élimination du quantificateur et de l'implication en un seul raccourci

Attention !

La règle d'élimination du quantificateur universel utilise l'opération de substitution. Ne pas respecter les précautions destinées à empêcher la capture de variables peut conduire à des déductions absurdes :

$$\frac{\Gamma \vdash j : \texttt{nat} \qquad \Gamma \vdash \forall\, i : nat, \exists\, j : \texttt{nat}, i \neq j}{\Gamma \vdash \exists\, j : \texttt{nat}, j \neq j}\ \forall_e, j$$

En revanche, si l'on renomme la variable liée j, tout se passe bien :

$$\frac{\Gamma \vdash j : \texttt{nat} \qquad \dfrac{\Gamma \vdash \forall\, i : nat, \exists\, j : \texttt{nat}, i \neq j}{\Gamma \vdash \forall\, i : nat, \exists\, k : \texttt{nat}, i \neq k}\ \alpha\text{-conversion}}{\Gamma \vdash \exists\, k : \texttt{nat}, j \neq k}\ \forall_e, j$$

Règle d'introduction

Soit x une variable non-libre dans le contexte Γ (*i.e.*, qui n'apparaît comme variable libre dans **aucune** des hypothèses de Γ).

$$\frac{\Gamma, x : A \vdash P}{\Gamma \vdash \forall\, x : A, P}\ \forall_i$$

Exemple

$$\frac{\dfrac{}{\Gamma, x : A \vdash x = x}\ =_i}{\Gamma \vdash \forall\, x : A, x = x}\ \forall_i$$

Attention !

Ne pas respecter la condition ci-dessus "*x non-libre dans le contexte* Γ conduit encore à des raisonnements absurdes :

$$\dfrac{\dfrac{\overline{i : \texttt{nat}, i = 2 \vdash i = 2}}{i : \texttt{nat}, i = 2 \vdash \forall\, i : \texttt{nat},\ i = 2}\ \forall_i}{i : \texttt{nat}, i = 2 \vdash 23 = 2}\ \forall_e, 23$$

Dans ce cas, on choisit une variable *"fraîche"* pour l'introduction du quantificateur universel :

$$\dfrac{\overline{i : \texttt{nat}, i = 2 \vdash i = 2}}{i : \texttt{nat}, i = 2 \vdash \forall\, k : \texttt{nat},\ i = 2}\ \forall_i$$

5.3.3 Notation de bloc pour l'introduction du quantificateur universel

Comme pour les hypothèses, les variables gérées dans la règle $\forall_i$ ont un comportement très similaire aux variables locales des langages tels que **C**. On créera donc un bloc pour chaque introduction de $\forall$ avec une mention "**Soit** $x : A$". La variable x est bien sûr *locale* à ce bloc. Une proposition P prouvée à l'intérieur du bloc est *exportée* sous la forme $\forall\, x : A, P$.

```
1 { Soit x : A
2     x = x  [=_i]
3 }
4 ∀ x : A,  x  =  x  [∀_i,  2,  x]
```

5.3.4 Quantifications imbriquées

Comme pour l'implication, nous généralisons l'introduction et l'élimination du quantificateurs à un nombre quelconque de variables.

Introduction du quantificateur universel (variante)

Soient $x_1, \ldots, x_n$ n variables non-libres dans le contexte Γ.

$$\dfrac{\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash P}{\Gamma \vdash \forall\, x_1 : A1, \ldots x_n : A_n,\ P}\ \forall_i$$

Elimination du quantificateur universel (variante)

$$\dfrac{\Gamma \vdash t_1 : A_1 \quad \ldots \quad \Gamma \vdash t_n : A_n \quad \Gamma \vdash \forall\, x_1 : A_1, \ldots, x_n : A_n,\ P}{\Gamma \vdash P[x1 \leftarrow t_1; \ldots; x_n \leftarrow t_n]}\ \forall_e, t_1, \ldots, t_n$$

Exemples

$$\dfrac{\dfrac{\dfrac{\overline{x:A, y:A, x=y \vdash x=y} \qquad \overline{x:A, y:A, x=y \vdash x=x}}{x:A, y:A, x=y \vdash y=x} =_e}{x:A, y:A \vdash x=y \to y=x} \to_i}{\vdash \forall x\ y:A, x=y \to y=x} \forall_i, x; y$$

```
1  { Soient x, y: A
2    { Supposons x = y
3      y = y  [=_i]
4      x  =  y  [=_e, 3,2]
5    }
6     x  =  y  →  y  =  x  [→_i, 2, 4]
7  ∀ x y : A,  x  =  y  →  y  =  x  [∀_i, 6, x,y]
```

Voici un exemple d'utilisation de $\forall_e$ sur deux quantifications imbriquées :

```
1  { Supposons ∀ x y : A, P(x, y)
2    { Soit x : A
3       P(x, x)   [∀_e,2,x,x]
4    }
5    ∀ x : A,  P(x, x)   [∀_i, 3,x]
6  }
7   (∀ x y : A, P(x, y)) → (∀ x A,  P(x, x)) [→_i,1,4]
```

5.3.5 Règles associées au quantificateur existentiel

Règle d'introduction

La règle ci-dessous exprime que pour prouver une formule $\exists x : A, P$, il suffit de fournir un terme $t : A$ tel que l'on puisse prouver la proposition $P[x \leftarrow t]$.

Le terme t est appelé le *témoin* de l'application de cette règle.

$$\dfrac{\Gamma \vdash t : A \qquad \Gamma \vdash P[x \leftarrow t]}{\Gamma \vdash \exists x : A, P} \exists_i, t$$

Exemple

```
1  Supposons ∀ x,  humain(x) → mortel(x)
2  Supposons humain(Socrate)
3  …
4  mortel(Socrate)
5  ∃ h : individu, mortel(h)  [∃_i,  4,  Socrates]
```

Exemple

```
{ Soit x:A
    x = x [=ᵢ]
    ∃ y : A, x = y [∃ᵢ, 2, x]
}
∀ x : A, ∃ y : A, x = y [∀ᵢ,3]
```

Attention !

La règle d'introduction du quantificateur utilise l'opération de substitution. Il est donc nécessaire de vérifier que les précautions d'usage de cette opération (voir Section 5.2.9 page 62 sont bien respectées.

Voici un exemple de mauvaise application de cette règle.

```
Supposons ∀ y : nat, y ≤ 2y
∃ x : nat, ∀ y : nat, y ≤ x [∃ᵢ,1,2y]
```

Règle d'élimination

On considère une variable x, non libre dans $\Gamma \cup \{Q\}$.

$$\frac{\Gamma, x : A, P \vdash Q \qquad \Gamma \vdash \exists x : A, P}{\Gamma \vdash Q}$$

Ce schéma peut s'écrire facilement en considérant un bloc contenent la variable x et l'hypothèse P :

```
...
∃ x : A, P
...
{ Soit x : A tel que P
    …
    Q
}
Q [∃ₑ,2,6]
```

Cela parait-il compliqué ? Intuitivement, le schéma d'élimination du quantificateur existentiel exprime le raisonnement suivant :

1. On considère qu'il existe un x vérifiant la propriété P
2. Prenons donc un élément quelconque x vérifiant P [1].
   (a) Dans ce contexte, nous arrivons à prouver Q
3. Puisque x était quelconque, Q est donc prouvé.

1. Cette quelconquitude est exprimée par la condition *" x, non libre dans* $\Gamma \cup \{Q\}$ *"*

Exemple

```
Supposons ∃ x : A, ∀ y : A,  x =  y
{ Soit x : A tel que ∀ y : A,  x =  y
    { soient a,  b : A
      x = a  [∀e,2,a]
      x = b  [∀e,2,b]
      a = b  [=e,5,4]
    }
     ∀ a b : A,  a = b  [∀i,6,a,b]
}
∀ a b : A,  a = b  [∃e,1,8]
```

Attention !

Qu'arrive-t-il si on ne respecte pas la condition x, non libre dans $\Gamma \cup \{Q\}$? Voici deux exemples de faux raisonnements :

```
Supposons x = 2
Prouvons ∃ x : nat,  x + x = x  {
  0  =  0  +  0   [=i]
  ∃ x : nat,  x + x = x [∃i,3,0]
}
{ Soit x : nat tel que x + x = x
    2 + 2 = 2  [=e, 1, 6]
    4  =  2 [calcul sur 7]
}
4  =  2 [∃e,2,7]
```

```
∃ x : nat,  x  =  0 [exercice]
{ soit x : nat tel que x =  0
    x + x = x  [exercice]
}
x + x = x  [∃e,2,7]
```

5.4 Pratique de la logique du premier ordre

Nous pouvons voir quelques exemples de preuve en logique du premier ordre, utilisant les règles de la logique propositionnelle et celles associées à l'égalité et aux deux quantificateurs. Suivant que l'on admet ou pas la règle du tiers exclu, on aura des théorèmes "classiques" ou "intuitionnistes". Nous étendons au calcul des prédicats l'usage des symboles $\vdash_K$ et $\vdash_J$.

5.4.1 Quantificateurs et négation

Soient A un type quelconque et P un prédicat sur A. Voici une preuve *en logique intuitionniste* du séquent $\sim(\exists x : A, P(x)) \vdash \forall x : A, \sim P(x)$

```
1  Supposons ∼(∃ x : A,  P(x))
2  { Soit x : A
3        { Supposons P(x)
4          ∃ x : A, P(x)  [∃_i, 4, x]
5          ⊥ [mp,1,5]
6        }
7        ∼P(x)  [→_i, 3,5]
8     }
9  }
10 ∀ x : A,  ∼  P(x)  [∀_i, 2, 7]
```

La preuve suivante est une preuve *logique classique* du séquent $\sim(\forall x : A, \sim P(x)) \vdash \exists x : A, P(x)$

```
1  Supposons ∼(∀ x : A,  ∼  P(x))
2  { Supposons ∼(∃ x : A,   P(x))
3    { Soit x : A
4       { Supposons P(x)
5           ∃ x : A,   P(x)  [∃_i,5,x]
6           ⊥ [→_e,2,5]
7       }
8       ∼  P(x)  [→_i, 4, 6]
9     }
10    ∀ x : A,  ∼  P(x), [∀_i,3, 8]
11    ⊥, [→_e,1,11]
12 }
13 ∃ x : A,   P(x)) [absurde classique]
```

On aurait pu aussi utiliser le tiers-exclu pour la même preuve :

```
1  Supposons ∼(∀ x : A,  ∼  P(x))
2  (∃ x : A,   P(x))  ∨  ∼  (∃ x : A,   P(x)) [tiers exclu]
3  { Supposons ∃ x : A,   P(x)
4      ∃ x,   P(x) [hypothèse]
5  }
6  { Supposons ∼(∃ x : A,   P(x))
7     même dérivation que ci-dessus
8    ∃ x : A,   P(x))
9  }
10 ∃ x : A,   P(x)) [∨_e,1,2-3,6-8]
```

5.4.2 Exemples

Exercice 37. *Prouver les jugements suivants :*

$\vdash_J \ \forall \ x \ : A, \ P(x) \ \rightarrow \ \exists \ y : \ A, \ P(y)$
$\sim(\forall\, x : \ A, \ P(x)) \ \equiv_K \ \exists\, x : A, \sim P(x)$
$\sim(\exists\, x : \ A, \ P(x)) \ \equiv_J \ \forall\, x : A, \sim P(x)$
$(\exists \ x : A, \ P(x)) \ \rightarrow \ Q \ \equiv_J \ \forall \ x : A, \ P(x) \ \rightarrow \ Q$
$\exists \ x : A, \forall \ y : B, \ Q(x, y) \ \vdash_J \ \forall \ y : B, \ \exists \ x, \ Q(x, y)$

5.4.3 Règles dérivées

Les règles dérivées applicables en logique propositionnelle intuitionniste [resp. classique] restent applicables en logique du premier ordre intuitionniste [resp. classique].

Il est toutefois intéressant de modifier le thèorème de substitution uniforme afin de réutiliser des jugements similaires à ceux de l'exercice 37.

Prenons par exemple le jugement suivant :
$\exists x : A, \forall y : B, \ Q(x, y) \vdash_J \forall y : B, \exists x, Q(x, y)$

Dans ce jugement, Q est un symbole de relation sur le type A $\times$ B.

On peut obtenir une instance de ce jugement en substituant à A un type X et à B un type Y, puis à Q une relation sur A $\times$ B.

Par exemple, substituant nat à A et B, et associons à Q le prédicat défini par $Q(n, p) = n < p$. L'opération de substitution consiste alors à remplacer toute sous-formule de la forme $Q(t_1, t_2)$ par $(n \leq p)[n \leftarrow t_1; p \leftarrow t_2]$ soit $t_1 < t_2$.

Nous obtenons donc le jugement :

$$(\exists x : \texttt{nat}, \ \forall y : \texttt{nat}, \ x \leq y \vdash_J \forall y : \texttt{nat}, \ \exists x, x \leq y$$

.

Nous admettons que, en logique des prédicats, la substitution uniforme peut s'appliquer à des symboles de prédicats et de relations, à condition que l'on remplace ces symboles en respectant lez contraintes de type.

Ce résultat permet d'utiliser les jugements de l'exercice 37 comme des règles dérivées.

Notons que l'utilisation de l'opération de substitution doit être correctement appliquée, notamment en ce qui concerne les captures de variables.

Par exemple, considérons une variable y :nat, et le jugement $\vdash_J \forall x \ : \ \texttt{nat}, P(x) \rightarrow \exists y \ : \ \texttt{nat}, P(y)$. Si nous associons au symbole P le prédicat défini par $P(\alpha) =_{\text{def}} y < \alpha$, une substitution appliquée sans précaution donnerait le jugement $\vdash_J \forall x : \texttt{nat}, y < x \rightarrow \exists y : \texttt{nat}, y < y$ au lieu du (correct) $\vdash_J \forall x : \texttt{nat}, y < x \rightarrow \exists z : \texttt{nat}, y < z$.

Exercice 38. *Démontrer en logique classique le paradoxe du buveur :*

"Soit un café avec au moins une personne (par exemple le patron). Alors, il existe une personne dans ce café qui, si elle boit, alors tout le monde boit"

On considère un type $presents$ *pour les personnes présentes dans le café, et une constante* $patron$ *:* $presents$

Il s'agit alors de montrer le jugement $\vdash_K \exists c : presents, boit(c) \rightarrow \forall x : presents, boit(x)$.

Aide : *On pourra utiliser le tiers-exclu avec la proposition* $\forall x : presents, boit(x)$.

Exercice 39. *Soit* A *un type. On peut exprimer que* A *est vide par la proposition* $vide(A) =_{def} \forall x : A, x \neq x$.

Montrer les jugements :

$$vide(A) \vdash_J \forall x : A, P(x) \tag{5.1}$$

$$\sim vide(A), \forall x : A, P(x) \vdash_K \exists x : A, P(x) \tag{5.2}$$

$$\forall\, x\, y : A,\ x \neq y \vdash_J vide(A) \tag{5.3}$$

5.5 Sémantique

5.5.1 Structures d'interprétation

Définition 10. *Soit* σ *une signature pour la logique du premier ordre. Une* $interprétation$ *pour* σ *est une structure* I *constituée des champs suivants :*

- *Pour chaque type* A*, un domaine* I_A [2]
- *Pour chaque constante* $c : A$*, un élément* $I_c \in I_A$,
- *Pour chaque symbole de fonction* $f : A_1 \times A_2 \times \cdots \times A_k \rightarrow A$*, une fonction totale* I_f *de* $I_{A_1} \times I_{A_2} \times \cdots \times I_{A_k}$ *dans* I_A,
- *Pour chaque symbole de relation* R *sur* $A_1 \times A_2 \times \cdots \times A_k$*, un sous-ensemble* I_R *du produit cartésien* $I_{A_1} \times A_2 \times \cdots \times A_k$.

Exemple

Prenons la signature vue en 5.2 page 55

Nous pouvons lui associer une interprétation I de la façon suivante :

- $I_{\texttt{nat}} =_{\text{def}} \mathbb{N}$
- $I_0 =_{\text{def}} 0$
- $I_S =_{\text{def}} \{(n, n+1) | n \in \mathbb{N}\}$
- $I_+ =_{\text{def}} +$
- $I_* =_{\text{def}} \times$
- $I_{nul} =_{\text{def}} \{0\}$
- $I_{positif} =_{\text{def}} \mathbb{N} \backslash \{0\}$
- $I_< =_{\text{def}} \{(n, p) | n,\ p \in \mathbb{N} \wedge n < p\}$

2. En théorie des modèles, on impose $I_A \neq \emptyset$. Nous n'imposerons pas cette hypothèse dans ce cours.

5.5.2 Valuations

On suppose dans ce paragraphe que pour chaque type A de la signature considérée, nous disposons d'une infinité de variables de ce type.

Définition 11. *Une valuation est une application ν qui à toute variable de type A associe une valeur $\nu(v) \in I_A$.*

Soit V un ensemble de variables ; on dit que deux valuations ν et ν' sont congruentes sur V (noté $\nu \equiv_V \nu'$) si pour toute variable v dans V, $\nu(v) = \nu'(v)$.

Sioit v une variable de type A et $a \in I_A$. On note $\nu\{v := a\}$ la valuation ν'' définie par

- *$\nu'(v) = a$*
- *$\nu'(w) = \nu(w)$ pour toute variable w différente de v.*

5.5.3 Sémantique des formules

Soit ν une valuation.

Pour tout terme t, notons $[t]_\nu$ la valeur de t dans la valuation ν.

Exercice : En donner la définition.

La *valeur de vérité* d'une formule φ dans ν, notée $[\varphi]_\nu$, se définit par récurrence sur la structure de φ.

- $[R(t_1, \ldots, t_n)]_\nu = \boldsymbol{v}$ si $([t_1]_\nu, \ldots, [t_n]_\nu) \in I_R$
- $[t_1 = t_2]_\nu = \boldsymbol{v}$ si $[t_1]_\nu = [t_2]_\nu$
- $[\varphi \to \varphi']_\nu =$
  - $\boldsymbol{v}$ si $[\varphi]_\nu = \boldsymbol{f}$ ou $[\varphi']_\nu = \boldsymbol{v}$
  - $\boldsymbol{f}$ si $[\varphi]_\nu = \boldsymbol{v}$ ou $[\varphi']_\nu = \boldsymbol{f}$
- $\ldots$ **exercice** : compléter la définition pour chaque connecterur
- $[\forall v : A, \varphi]_\nu = \boldsymbol{v}$ si pour toute valeur $a \in I_A$, on a $[\varphi]_{\nu\{v:=a\}} = \boldsymbol{v}$
- $[\exists v : A, \varphi]_\nu = \boldsymbol{v}$ s'il existe une valeur $a \in I_A$, telle que $[\varphi]_{\nu\{v:=a\}} = \boldsymbol{v}$

Définition 12. *On dit qu'un séquent est valide (Notation $\Gamma \vDash A$) si pour toute valuation ν qui satisfait toutes les hypothèses de Γ, alors $[A]_\nu = \boldsymbol{v}$.*

5.6 Méta-théorèmes principaux

Nous donnons mes deux résultats suivants sans démonstration. Le lecteur est invité à consulter la nombreuse bibliographie sur ce sujet.

Méta-théorème 20 (Cohérence)**.** *Si $\Gamma \vdash_J A$, alors $\Gamma \vDash A$*

A fortiori, si $\Gamma \vdash_K A$, alors $\Gamma \vDash A$

Méta-théorème 21 (Complétude de la logique classique)**.** *Si $\Gamma \vDash A$, alors $\Gamma \vdash_K A$*

Exercice 40. *Montrer que le séquent $\forall x : A, P(x) \vdash \exists x : A, P(x)$ n'est pas prouvable en logique classique, ni en logique intuitionniste.* ***Aide :*** *Revoir l'exercice 39 page précédente.*

Exercice 41. *On considère le contexte* Γ *formé des axiomes suivants :*

$$\forall\ x\ : A,\ \sim x\ <\ x$$
$$\forall\ x\,y\,z\ : A,\ x\ <\ y \to y\ <\ z \to x\ <\ z$$
$$\forall\ x\,y\ : A,\ x\ \leq\ y \leftrightarrow x\ <\ y\ \vee\ x\ =\ y$$

Montrer que de Γ *on peut déduire les propositions suivantes :*

$$\forall\ x\ : A,\ x\ \leq\ x$$
$$\forall\ x\,y\ : A,\ x\ \leq\ y\ \to\ y\ \leq\ x\ \to\ x\ =\ y$$
$$\forall\ x\,y\,z\ : A,\ x\ \leq\ y \to y\ \leq\ z \to x\ \leq\ z$$

Deuxième partie

Spécifier et prouver des programmes fonctionnels

Dans cette partie, nous montrons quelques techniques de base pour spécifier et prouver des programmes fonctionnels très simples. Pour des preuves de programmes plus réalistes, voir par exemple [3].

Nous nous limitons à quelques types de données inductifs simples : booléens, entiers naturels, listes et arbres (monomorphes). Pour chaque type, nous énumérerons ses constructeurs, des axiomes ou schémas d'axiomes spécifiques à ce type, et présenterons des techniques de preuves de fonctions opérant sur ce type.

Chapitre 6

Quelques types inductifs

6.1 Les valeurs booléennes

Le type `bool`, présent en *OCaML* et en *Coq* ne contient que deux valeurs appelées *constructeurs* : `true` et `false`.

6.1.1 Règles spécifiques au type `bool`

Une première règle stipule que ces deux constructeurs sont différents.

$$\frac{}{\texttt{true} \neq \texttt{false}}\ \text{bool-diff}$$

Comment exprimer que `bool` ne contient *que* `true` et `false` ? La solution la plus commmode est d'énoncer le *schéma de règle* suivant :

Soit P un prédicat défini sur `bool`.

$$\frac{P(\texttt{true}) \qquad P(\texttt{false})}{\forall\, b : \texttt{bool},\ P(b)}\ \text{bool-cases}$$

En combinant cette règle avec une élimination du quantificateur universel, nous obtenons la variante suivante.

$$\frac{P(\texttt{true}) \qquad P(\texttt{false})}{b : \texttt{bool} \vdash P(b)}\ \text{bool-cases}$$

Du coup, nous pouvons prouver que tout booléen est, soit `true`, soit `false`.

$$\dfrac{\dfrac{\dfrac{}{\texttt{true} = \texttt{true}}\ =_i}{\texttt{true} = \texttt{true} \vee \texttt{true} = \texttt{false}}\ \vee_{i,1} \qquad \dfrac{\dfrac{}{\texttt{false} = \texttt{false}}\ =_i}{\texttt{false} = \texttt{true} \vee \texttt{false} = \texttt{false}}\ \vee_{i,2}}{\dfrac{b : \texttt{bool} \vdash b = \texttt{true} \vee b = \texttt{false}}{\forall b : \texttt{bool},\ b = \texttt{true} \vee b = \texttt{false}}\ \forall_i}\ \text{bool-cases}$$

6.1.2 Définitions de fonctions par cas

L'utilisation du type `bool` dans des programmes se fait simplement, soit en plaçant un des deux constructeurs, `true` ou `false` dans une expression du langage, soit en utilisant la construction `if` b `then` e_1 `else` e_2, où b est une expression de type `bool`, et e_1 et e_2 deux expressions *d'un même type* A. Alors, l'expression conditionnelle a aussi le type A.

Par exemple, en *OCaML*, voici les définitions de la négation, la conjonction et la disjonction booléennes.

```
let notb b = if b then false else true

let andb b b' = if b then b' else false

let orb b b' = if b then true else b'
```

Pour raisonner sur ces programmes, on utilise les règles de calcul suivante

> Toute expression de la forme `if true then` e_1 `else` e_2 se simplifie en e_1
>
> Toute expression de la forme `if false then` e_1 `else` e_2 se simplifie en e_2

Par exemple, l'expression `notb false`, équivalente à `if false then false else true`, se simplifie en `true`.

L'expression `notb (not b false)`, équivalente à `if (notb false) then false else true` va se simplifier en `if true then false else true`, puis en `false`.

Exercice 42. *Démontrer le théorème* $\forall b : \texttt{bool}, \texttt{notb (notb } b\texttt{)} = b$.

Exercice 43. *Écrire les règles de simplifications associées aux fonctions* `orb` *et* `andb`.

Montrer l'égalité $\forall b : \texttt{bool}, \texttt{orb } b \texttt{ (notb } b\texttt{)= } b$.

Expressions conditionnelles

Syntaxe, règles de simplification.

6.1.3 Exemples

négation, conjonction et disjonction booléennes.

6.2 Entiers naturels

Même plan avec la récursion en plus.

Troisième partie

Annexes

Chapitre 7

TO DO

Les trucs à rédiger et placer correctement.

1. Axiomes vs hypothèses

Bibliographie

[1] apprendre-en-ligne.net. Site Web sur l'informatique et les maths au lycée.

[2] J.P. Belna. *Histoire de la logique*. L'esprit des sciences. Ellipses, 2005.

[3] Yves Bertot and Pierre Castéran. *Le Coq'Art*. Version française : http ://www.labri.fr/∼casteran/CoqArt/coqartF.pdf.

[4] www.cgsecurity.org/wiki/Articles.

[5] G. Dowek. *La logique*. Flammarion, coll. "Dominos", 1995.

[6] Christine Paulin-Mohring. Mathématiques pour l'informatique 2. www.lri.fr/∼paulin/MathInfo2/.

[7] Arthur Shcopenhauer. *l'Art d'avoir toujours raison*. Éditions Mille et une Nuits. Voir aussi https ://fr.wikisource.org/wiki/L'Art d'avoir toujours raison.

[8] Coq Development Team. The Coq Proof Assistant. https ://coq.inria.fr/.

[9] Wikipedia. Crise des fondements. https ://fr.wikipedia.org/wiki/Crise_des_fondements.

Index

C

Conjonction (∧), 44
Contradiction (⊥), 38

D

Disjonction (∨), 45

E

Ex falso quodlibet sequitur, *voir* Elimination de la contradiction

M

Méta-théorèmes
- Complétude de la logique propositionnelle classique, 52
- Correction, 30, 49, 51
- Incomplétude de la logique minimale, 31
- Non-contradiction, 49, 51
- Remplacement, 50
- Substitution uniforme, 32

N

Négation ($\sim$), 40

P

Preuve par cas, *voir* Élimination de la disjonction
Principe d'explosion, *voir* Elimination de la contradiction

R

Règle d'hypothèse, 17
Règles d'élimination
- Élimination de l'implication (modus-ponens), 18
- Élimination de la conjonction ($\wedge_e$), 44
- Élimination de la contradiction, 40
- Élimination de la disjonction ($\vee_e$), 46

Règles d'introduction
- Introduction de l'implication, 20
- Introduction de la conjonction ($\wedge_i$), 44
- Introduction de la disjonction ($\vee_{i,1}$ et $\vee i, 2$), 46
- Introduction de la négation, 41

Règles dérivées
- Commutativité de la conjonction, 45
- Règle d'affaiblissement, 28
- Règle de coupure, 26

S

Satisfaction, 16
Séquents, 9

T

Tautologie, 16

V

Validité, 16