Méthodes formelles
Support de cours

Licence d'Informatique de l'Université de Franche-Comté

Centre de Télé-enseignement Universitaire

Françoise Bellegarde Jean-François Couchot Alain Giorgetti
Christophe Guyeux

Version du 31 mars 2014

Table des matières

Première partie

Introduction

Avant-propos

L'annexe A et les chapitres 1 à 4 du cours ont été adaptés par Alain Giorgetti à partir d'un support de cours de l'I.U.T. de Belfort-Montbéliard co-écrit par Christophe Guyeux et Jean-François Couchot. Les chapitres 5 à 12 ont été adaptés par Alain Giorgetti à base de supports de cours écrits par Françoise Bellegarde. Enfin, le chapitre 10 a été écrit par Alain Giorgetti.

Deuxième partie

Cours

Chapitre 1

Logique des propositions

I Introduction

L'objet de la logique est d'analyser le raisonnement humain. On n'étudie pas ici la logique en tant que discipline philosophique, mais en tant que formalisation du raisonnement. Cette formalisation a de nombreux buts, mais on s'intéresse surtout à deux d'entre eux. Le premier but est de disposer d'un langage commun pour définir toutes les théories scientifiques. C'est le but de la *logique mathématique*, qui tente d'unifier les raisonnements utilisés dans les sciences, et plus particulièrement en mathématiques. Le second but, qui nous intéresse encore davantage, est d'essayer de produire automatiquement des résultats du raisonnement. Il est apparu plus récemment, grosso modo avec l'invention des machines de calcul.

I.1 Production automatique

Chacun sait que, si un ordinateur calcule vite et bien, il est incapable de produire de lui-même quelque raisonnement que ce soit. Mais si l'on parvient à programmer les raisonnements logiques, alors l'ordinateur peut devenir, de ce point de vue aussi, plus rapide et plus efficace que l'esprit humain. Nous poursuivons l'étude de ce projet et de ses limites.

I.2 Des problèmes de l'évidence

Avant de chercher à programmer un raisonnement, il convient de savoir précisément ce qu'est un raisonnement. Il faut apprendre à disséquer nos démarches intellectuelles, et, avant tout, ne pas se laisser paralyser par l'évidence : rien n'est évident pour un ordinateur. Le plus difficile est peut-être de pourchasser, pour les expliciter clairement, tous les sous-entendus qui nous permettent de produire des raisonnements intelligibles. Finalement, il convient toujours d'adapter son discours au niveau de son auditoire : le niveau de l'ordinateur, de ce point de vue, peut être considéré comme égal à zéro.

I.3 Contenu du chapitre

Ce chapitre définit la "logique des propositions", également appelée "calcul des propositions" ou "calcul propositionnel". C'est la logique la plus élémentaire. Elle sert de base pour

définir beaucoup d'autres logiques plus expressives. Elle est présentée sous sa forme "classique".

Il est essentiel d'assimiler toutes les notions et tout le vocabulaire définis ici.

II Les fondements de la logique des propositions

Dans "Les métamorphoses du calcul", G. Dowek écrit : *Qu'est-ce donc qu'un raisonnement ? Si l'on sait que tous les écureuils sont des rongeurs, que tous les rongeurs sont des mammifères, que tous les mammifères sont des vertébrés et que tous les vertébrés sont des animaux, on peut en déduire que tous les écureuils sont des animaux.[...].*

Ce raisonnement est simple à l'extrême, mais sa structure ne diffère pas fondamentalement de celle d'un raisonnement mathématique. Dans les deux cas, le raisonnement est formé d'une suite de propositions dans laquelle chacune découle logiquement des précédentes, [...]. Dans ce cas, on applique la même règle trois fois. Cette règle permet, si l'on sait déjà que tous les Y sont des X et que tous les Z sont des Y, de déduire que tous les Z sont des X.

Cette partie formalise la notion de proposition (partie II.1), montre comment les propositions peuvent être connectées entre elles (partie II.2) et comment elles sont représentées syntaxiquement (partie II.3).

II.1 Les propositions

L'homme exprime son raisonnement par un discours, et ce discours utilise une langue (une langue naturelle, français, anglais,...). D'une manière générale, ce discours est articulé en phrases, d'un niveau de complexité variable, et c'est l'étude de ces « énoncés » que se propose de faire la logique.

DÉFINITION 1.1 (PROPOSITION). *Parmi tous les énoncés possibles qui peuvent être formulés dans une langue, on distingue ceux auxquels il est possible d'attribuer une « valeur de vérité » : vrai ou faux. Ces énoncés porteront le nom de* propositions .

EXEMPLE 1.1. Ainsi, « Henri IV est mort assassiné en 1610 », « Napoléon Bonaparte a été guillotiné en 1852 » sont des propositions, puisqu'on peut leur attribuer une valeur de vérité (« vrai » pour la première, « faux » pour la seconde).

II.1.1 Des énoncés qui ne sont pas des propositions

Un énoncé « hypothétique » comme « Dans six mois, il y aura une exceptionnelle période de beau temps » est exclu du domaine des propositions et donc de notre étude. Il en est de même pour ce qu'on appelle les « paradoxes » de la logique comme, par exemple, celui du menteur. En effet, il est impossible d'attribuer une valeur de vérité à ces énoncés : nous les rejetons en dehors du cadre des propositions.

II.1.2 Principes fondateurs du calcul des propositions

Le calcul que l'on étudie considère toujours comme acquises les vérités suivantes, élevées au rang d'axiomes.

Principe de non-contradiction : Une proposition ne peut être simultanément vraie et fausse.

Principe du tiers-exclu : Une proposition est vraie ou fausse (il n'y a pas d'autre possibilité).

II.1.3 D'autres logiques

Il existe, bien entendu, d'autres logiques :
- fondées sur d'autres axiomes,
- qui admettent d'autres « valeurs de vérité » : le « possible », par exemple
- qui attribuent des « coefficients de vraisemblance » aux énoncés,
- etc.

Ces logiques sortent du cadre de ce chapitre.

II.2 Les connecteurs logiques

L'analyse logique d'une phrase (reconnue comme proposition) fait apparaître des sous-phrases qui sont elles-mêmes des propositions. Ces « membres de phrases » sont reliés entre eux par des « connecteurs logiques ».

Par exemple, considérons l'énoncé : « J'ai obtenu une mauvaise note à cet examen parce que je n'ai pas assez travaillé ou parce que le cours est trop difficile ». Puisqu'il est possible d'attribuer une valeur de vérité à cet énoncé « global », c'est une proposition.

On peut alors mener ce qu'en grammaire française on appelle l'analyse logique de cette phrase, de manière à en extraire les propositions (au sens grammatical du terme) : « J'ai obtenu une mauvaise note à cet examen », « je n'ai pas assez travaillé », « le cours est trop difficile », qui sont aussi des propositions au sens logique du terme. Ces propositions, au sens grammatical du terme, sont reliées entre elles par « parce que » et par « ou », qui sont – grammaticalement parlant – des conjonctions (respectivement de subordination et de coordination). La conjonction « parce que » introduit habituellement un lien de « cause à effet ». La conjonction « ou » se contente de juxtaposer les propositions (au même niveau).

II.2.1 Vers une formalisation

Globalement, cette proposition exprime que « ma mauvaise note » est conséquence de l'une (au moins) des deux causes suivantes :
- « mon manque de travail »,
- « un cours trop difficile ».

Autrement posé (il s'agit d'un début de formalisation) :

(« mon manque de travail » ou « cours trop difficile ») entraîne « ma mauvaise note »

REMARQUE 1.1. Il ne faut pas sous-estimer la difficulté de ce travail d'analyse :
- le langage courant est souvent imprécis ou ambigu,

— il faut souvent se livrer à une véritable interprétation pour parvenir à formaliser une phrase.

REMARQUE 1.2. L'analyse en logique des propositions s'arrête au niveau des connecteurs logiques (qui vont être présentés un par un dans la suite de ce chapitre). Elle ne permet pas de prendre en compte certaines nuances, la concordance des temps, ou d'autres liens qui peuvent exister entre des propositions.

D'une manière générale, le calcul propositionnel ne se préoccupe que des valeurs de vérité, et pas du tout des liens sémantiques qui peuvent exister entre des propositions. Ces dernières sont reliées entre elles syntaxiquement par des connecteurs comme « ou » ou « entraîne ».

Les connecteurs logiques sont donc des symboles qui permettent de produire des propositions « plus complexes » à partir d'autres propositions « plus simples ». Ils sont définis axiomatiquement par leur "table de vérité".

II.2.2 Disjonction logique

À partir de deux propositions P et Q, le connecteur $\vee$, qui se lit « ou », permet de construire la nouvelle proposition (P ou Q), notée $(P \vee Q)$.

La proposition $P \vee Q$ est fausse si et seulement si les deux propositions P et Q sont fausses. Autrement dit, la valeur de vérité de $P \vee Q$ est définie en fonction de celles de P et de Q par la table de vérité suivante, où "F" se lit "faux" et "V" se lit "vrai".

P	Q	$P \vee Q$
F	F	F
F	V	V
V	F	V
V	V	V

REMARQUE 1.3. Dans le langage courant, le mot « ou » est souvent employé de deux façons distinctes. Il est parfois utilisé avec le sens « les deux cas peuvent se produire » (comme ici) et parfois avec le sens « p ou q , mais pas les deux » (e.g. « il ira à Paris ou à Marseille »). Sauf indication contraire, le « ou » sera toujours employé avec cette première signification.

II.2.3 Conjonction logique

Connecteur « et », symbole $\wedge$.

À partir de deux propositions P et Q, le connecteur $\wedge$, qui se lit « et », permet de construire la nouvelle proposition (P et Q), notée $(P \wedge Q)$, dont la valeur de vérité est définie en fonction de celles de P et de Q par la table de vérité :

P	Q	$P \wedge Q$
F	F	F
F	V	F
V	F	F
V	V	V

Autrement dit, la proposition $P \wedge Q$ est vraie si et seulement si les deux propositions P et Q sont vraies.

II.2.4 Négation logique

À partir d'une proposition P, le connecteur $\neg$, qui se lit « non », permet de construire la nouvelle proposition (non P), notée $(\neg P)$, dont la valeur de vérité est définie en fonction de celle de P par la table de vérité suivante :

P	$\neg P$
F	V
V	F

II.2.5 Implication logique

À partir de deux propositions P et Q, le connecteur $\Rightarrow$, qui se lit « si... alors », permet de construire la proposition (Si P, alors Q), notée $(P \Rightarrow Q)$, dont la valeur de vérité est définie en fonction de celles de P et de Q par la table de vérité suivante :

P	Q	$P \Rightarrow Q$
F	F	V
F	V	V
V	F	F
V	V	V

Autrement dit, la proposition $P \Rightarrow Q$ est vraie si P est fausse ou si Q est vraie.

REMARQUE 1.4. Lorsque la proposition P est fausse, la proposition « Si P, alors Q » est vraie, quelle que soit la valeur de vérité de la proposition Q.

Par exemple, la proposition : « Si le pôle Nord est l'endroit le plus chaud de la planète, alors les poules ont des dents » doit être considérée comme ayant la valeur de vérité « vrai ». Le connecteur logique « $\Rightarrow$ » ne s'occupe nullement (on l'a dit) des liens sémantiques qui peuvent exister entre la température qui règne au pôle Nord et la dentition des poules.

Il s'agit simplement d'une proposition, qui forme un tout, et qui a la valeur de vérité « vrai » lorsque, notamment, P et Q ont toutes les deux la valeur de vérité « faux ».

La manière de mener un raisonnement à l'aide de propositions qui se présentent sous la forme d'implications logiques est l'objet de la théorie de la déduction qui sera étudiée plus loin.

II.2.6 Équivalence logique

À partir de deux propositions P et Q, le connecteur $\Leftrightarrow$, qui se lit « si et seulement si », permet de construire la nouvelle proposition (P si et seulement si Q), notée $(P \Leftrightarrow Q)$, dont la valeur de vérité est donnée par la table de vérité suivante :

P	Q	$P \Leftrightarrow Q$
F	F	V
F	V	F
V	F	F
V	V	V

REMARQUE 1.5. Même remarque que pour l'implication logique : l'équivalence logique de deux propositions fausses est une proposition vraie.

Exercice (corrigé) 1.2. *En notant M et C les affirmations suivantes :*
— *M = « Jean est fort en Maths »,*
— *C = « Jean est fort en Chimie »,*
représenter les affirmations qui suivent sous forme symbolique, à l'aide des lettres M et C et des connecteurs usuels.

1. *« Jean est fort en Maths mais faible en Chimie »*
2. *« Jean n'est fort ni en Maths ni en Chimie »*
3. *« Jean est fort en Maths ou il est à la fois fort en Chimie et faible en Maths »*
4. *« Jean est fort en Maths s'il est fort en Chimie »*
5. *« Jean est fort en Chimie et en Maths ou il est fort en Chimie et faible en Maths »*

Réponses :
1. $M \wedge (\neg C)$ *; 2.* $(\neg M) \wedge (\neg C)$ *; 3.* $M \vee (\neg M \wedge C)$ *; 4.* $C \Rightarrow M$ *; 5.* $(M \wedge C) \vee (\neg M \wedge C)$.

Exercice (corrigé) 1.3. *Énoncer la négation des affirmations suivantes en évitant d'employer l'expression : « il est faux que »*

1. *« S'il pleut ou s'il fait froid je ne sors pas. »*
2. *« Le nombre 522 n'est pas divisible par 3 mais il est divisible par 7. »*
3. *« Ce quadrilatère n'est ni un rectangle ni un losange. »*
4. *« Si Paul ne va pas travailler ce matin il va perdre son emploi. »*
5. *« Tout nombre entier impair peut être divisible par 3 ou par 5 mais jamais par 2. »*
6. *« Tout triangle équilatéral a ses trois angles égaux à 60°. »*

Réponses :

1. *Il pleut ou il fait froid et pourtant, je sors.*
2. *Le nombre 522 est divisible par 3 ou il n'est pas divisible par 7.*
3. *Ce quadrilatère est un rectangle ou un losange.*

4. *Paul n'ira pas travailler ce matin mais il ne perdra pas son emploi.*
5. *Il existe un nombre entier impair, qui n'est divisible ni par 3 ni par 5, ou qui est divisible par 2.*
6. *Il existe un triangle équilatéral dont (au moins) un angle n'est pas égal à 60°.*

II.3 Variables et formules propositionnelles

Comme le calcul propositionnel ne s'occupe que des valeurs de vérité, il est possible, dans une expression logique, de remplacer chaque proposition donnée par un symbole (en général, une lettre de l'alphabet), appelé *variable propositionnelle*, et d'étudier ensuite les valeurs de vérité de l'expression en fonction des valeurs de vérité de ces symboles.

II.3.1 Formalisation

DÉFINITION 1.2 (FORMULES PROPOSITIONNELLES). *L'expression obtenue en remplaçant dans une proposition les propositions les plus simples par des variables propositionnelles est appelée* formule propositionnelle .

REMARQUE 1.6. La valeur de vérité d'une formule propositionnelle est une fonction des valeurs de vérité des variables propositionnelles qui y interviennent.

PROPRIÉTÉ 1.1 : Les règles (de syntaxe) qui permettent de former des formules propositionnelles correctes sont les suivantes :
- Toute variable propositionnelle est une formule propositionnelle.
- Si F et G sont des formules propositionnelles, alors $(\neg F)$, $(F \vee G)$, $(F \wedge G)$, $(F \Rightarrow G)$ et $(F \Leftrightarrow G)$ sont des formules propositionnelles.

La propriété suivante permet d'éviter d'écrire certaines paires de parenthèses dans une formule.

PROPRIÉTÉ 1.2 (RÈGLES DE PRIORITÉ DES CONNECTEURS LOGIQUES) : Les conventions de priorité des connecteurs logiques sont les suivantes (par ordre de priorité décroissante) :
- la négation,
- la conjonction et la disjonction (au même niveau),
- l'implication et l'équivalence (au même niveau).

EXEMPLE 1.4. $\neg A \wedge B \Rightarrow C$ doit être interprété par $((\neg A) \wedge B) \Rightarrow C$. $A \vee B \wedge C$ n'a pas de sens, car les deux connecteurs ont le même niveau de priorité. Il faut ajouter des parenthèses.

PROPRIÉTÉ 1.3 (ASSOCIATIVITÉ DES OPÉRATEURS $\vee$ ET $\wedge$) : Les opérateurs $\vee$ et $\wedge$ sont associatifs :
— $(A \vee B) \vee C = A \vee (B \vee C) = A \vee B \vee C$,
— $(A \wedge B) \wedge C = A \wedge (B \wedge C) = A \wedge B \wedge C$.
Mais le parenthésage est obligatoire quand $\vee$ et $\wedge$ se trouvent dans la même proposition, puisqu'il n'y a pas de priorité entre $\vee$ et $\wedge$: $(A \vee B) \wedge C \neq A \vee (B \wedge C)$.

REMARQUE 1.7. L'implication n'est pas associative : $A \Rightarrow (B \Rightarrow C) \neq (A \Rightarrow B) \Rightarrow C$. On convient que $A \Rightarrow B \Rightarrow C$ signifie $A \Rightarrow (B \Rightarrow C)$ donc que les parenthèses sont obligatoires pour écrire $(A \Rightarrow B) \Rightarrow C$.

En cas de doute, ne pas hésiter à mettre plus de parenthèses ;

Exercice (corrigé) 1.5. *Quelles sont les façons de placer des parenthèses dans $\neg p \vee q \wedge \neg r$ afin d'obtenir l'expression correcte d'une formule propositionnelle ? Déterminer la table de vérité de chacune des formules obtenues.*

Réponses :
1) $\neg p \vee (q \wedge \neg r)$; 2) $(\neg p \vee q) \wedge \neg r$; 3) $(\neg(p \vee q)) \wedge \neg r$; 4) $\neg(p \vee (q \wedge \neg r))$; 5) $\neg((p \vee q) \wedge \neg r)$.
Tables de vérité :

p	q	r	1	2	3	4	5
V	V	V	F	F	F	F	V
V	V	F	V	V	F	F	F
V	F	V	F	F	F	F	V
V	F	F	F	F	F	F	F
F	V	V	V	F	F	V	V
F	V	F	V	V	F	F	F
F	F	V	V	F	F	V	V
F	F	F	V	V	V	V	V

Exercice (corrigé) 1.6. *Construire les tables de vérité des formules propositionnelles suivantes :*

1. $(\neg p) \wedge q$
2. $(\neg p) \Rightarrow (p \vee q)$
3. $\neg((\neg p) \wedge (\neg q))$
4. $(p \wedge q) \Rightarrow (\neg q)$
5. $(p \Rightarrow q) \vee (q \Rightarrow p)$
6. $(p \Rightarrow (\neg q)) \vee (q \Rightarrow (\neg p))$

7. $(p \vee (\neg q)) \wedge ((\neg p) \vee q)$
8. $p \Rightarrow ((\neg p) \Rightarrow p)$

Réponse :

p	q	1	2	3	4	5	6	7	8
V	V	F	V	V	F	V	F	V	V
V	F	F	V	V	V	V	V	F	V
F	V	V	V	V	V	V	V	F	V
F	F	F	F	F	V	V	V	V	V

III Interprétation booléenne de la logique des propositions

Cette partie donne un sens, une *sémantique*, à toute formule propositionnelle. Cette présentation est fondée sur la notion d'algèbre de Boole et sur ses notations. Le lecteur qui ne connaît pas cette notion et ces notations doit d'abord les assimiler par l'étude de l'annexe A.

Puisqu'une formule propositionnelle représente un énoncé qui est soit vrai, soit faux, la signification ultime d'une formule propositionnelle est l'une des deux *valeurs de vérité*, "vrai" ou "faux". On code respectivement ces deux valeurs par les constantes booléennes 1 et 0. Cependant, la valeur d'une formule propositionnelle dépend de celle de ses variables propositionnelles, elle varie *en fonction de* ces valeurs. Par conséquent, on définit sa sémantique en deux étapes. La première étape transforme la formule propositionnelle en une *expression booléenne*. La seconde étape considère cette expression booléenne comme une *fonction booléenne* des variables qu'elle contient. Pour l'instant, on ne formalise que la première étape, par la définition suivante.

DÉFINITION 1.3 (EXPRESSION BOOLÉENNE ASSOCIÉE À UNE FORMULE PROPOSITIONNELLE). *Soit F une formule propositionnelle quelconque. L'expression booléenne associée à F, notée expr-bool(F), est obtenue de la manière suivante :*

1. *Si P est une variable propositionnelle, alors expr-bool$(P) = P$.*
2. *Si G est une formule propositionnelle, alors*

$$\textit{expr-bool}(\neg G) = \overline{\textit{expr-bool}(G)}.$$

3. *Si F est de la forme $G \vee H$, où G et H sont des formules propositionnelles, alors*

$$\textit{expr-bool}(G \vee H) = \textit{expr-bool}(G) + \textit{expr-bool}(H).$$

4. *Si F est de la forme $G \wedge H$, où G et H sont des formules propositionnelles, alors*

$$\textit{expr-bool}(G \wedge H) = \textit{expr-bool}(G) \cdot \textit{expr-bool}(H).$$

5. *Si F est de la forme $G \Rightarrow H$, où G et H sont des formules propositionnelles, alors*

$$\textit{expr-bool}(G \Rightarrow H) = \overline{\textit{expr-bool}(G)} + \textit{expr-bool}(H).$$

6. *Si F est de la forme $G \Leftrightarrow H$, où G et H sont des formules propositionnelles, alors*

$$\textit{expr-bool}(G \Leftrightarrow H) = \overline{\textit{expr-bool}(G)} \cdot \overline{\textit{expr-bool}(H)} + \textit{expr-bool}(G) \cdot \textit{expr-bool}(H).$$

EXEMPLE 1.7. Soit $F = A \vee \neg B \Leftrightarrow (B \Rightarrow C)$. On a alors :

$$\begin{array}{ll} & \textit{expr-bool}(F) \\ = & \\ & \overline{A + \overline{B}} \cdot \overline{\overline{B} + C} + (A + \overline{B}) \cdot (\overline{B} + C) \\ = & \\ & \overline{A} \cdot B \cdot B \cdot \overline{C} + \overline{B} + A \cdot C \\ = & \text{par la règle d'itempotence, appliquée à } B \\ & \overline{A} \cdot B \cdot \overline{C} + \overline{B} + A \cdot C \\ = & \text{par la règle de double négation} \\ & \overline{A} \cdot \overline{\overline{B}} \cdot \overline{C} + \overline{B} + A \cdot C \\ = & \text{par la 3e règle de redondance } a + \overline{a} \cdot b = a + b \text{ en remplaçant } a \text{ par } \overline{B} \text{ et } b \text{ par } \overline{A} \cdot \overline{C} \\ & \overline{A} \cdot \overline{C} + \overline{B} + A \cdot C \\ = & \\ & \overline{B} + \overline{A} \cdot \overline{C} + A \cdot C. \end{array}$$

REMARQUE 1.8. Les connecteurs logiques de négation, de disjonction et de conjonction sont simplement respectivement transformés en négation booléenne, somme booléenne et produit booléen. En revanche, il n'y a pas d'opération dans les algèbres de Boole qui corresponde aux connecteurs d'implication et d'équivalence logique. C'est pourquoi ils sont "éliminés" par la fonction *expr-bool*, qui les définit à partir des trois opérations des algèbres de Boole.

III.1 Fonctions de vérité

Cette partie présente la seconde étape de la définition de la sémantique booléenne de la logique des propositions.

DÉFINITION 1.4 (FONCTION DE VÉRITÉ). *Soit F une formule propositionnelle, dont les variables propositionnelles sont P_1, P_2, P_3, …, P_n ($n \geq 0$). La fonction de vérité de F, notée Φ_F, est la fonction booléenne qui associe aux n variables propositionnelles P_1, P_2, P_3, …, P_n de F la valeur de l'expression booléenne* expr-bool(F).

On peut donner une par une toutes les valeurs d'une fonction de vérité en construisant sa "table de vérité". Des exemples de tables de vérité ont déjà été donnés dans les parties II.2 et II.3. Pour pouvoir les considérer comme des tables de valeurs d'une fonction de vérité, il suffit de remplacer tous les F par 0 et tous les V par 1.

III.2 Conséquences logiques

Soit $\mathcal{F} = \{F_1, \ldots, F_n\}$ un ensemble de formules propositionnelles.

DÉFINITION 1.5 (CONSÉQUENCE LOGIQUE). *On dit que la formule propositionnelle A est une* conséquence logique *des formules propositionnelles $F_1, \ldots, F_n$ de l'ensemble lorsque, chaque fois que les fonctions de vérité Φ_{F_1}, …, Φ_{F_n} prennent simultanément la valeur « vrai » (ou 1), il en est de même pour la fonction de vérité de la formule A.*

NOTATION : On note ce résultat : $\mathcal{F} \models A$ (se lit : A est conséquence logique de $\mathcal{F}$).

EXEMPLE 1.8. $\{P \Rightarrow Q, P\} \models Q$. En effet :

P	Q	$P \Rightarrow Q$
F	F	V
F	V	V
V	F	F
V	V	V

Il n'y a qu'un seul cas dans lequel $P \Rightarrow Q$ et P sont simultanément vrais. Dans ce cas, Q est vrai.

III.3 Tautologies

DÉFINITION 1.6 (TAUTOLOGIE). *Une* tautologie *est une formule propositionnelle dont la fonction de vérité est la fonction booléenne constante qui vaut 1 quelle que soit la valeur de vérité de ses paramètres.*

NOTATION : La notation utilisée pour marquer une tautologie F est $\models F$ (se lit : « F est une tautologie »).

EXEMPLE 1.9. Soit $F = A \Rightarrow A$. Comme *expr-bool*$(F) = \overline{A} + A = 1$, on a $\models F$.

EXEMPLE 1.10. Montrons que $F = (A \Rightarrow C) \Rightarrow ((B \Rightarrow C) \Rightarrow (A \vee B \Rightarrow C))$ est une tautologie.

$$
\begin{aligned}
& \textit{expr-bool}((A \Rightarrow C) \Rightarrow ((B \Rightarrow C) \Rightarrow (A \vee B \Rightarrow C))) \\
= \quad & \\
& \overline{\textit{expr-bool}(A \Rightarrow C)} + \overline{\textit{expr-bool}(B \Rightarrow C)} + \textit{expr-bool}(A \vee B \Rightarrow C) \\
= \quad & \\
& \overline{\overline{A} + C} + \overline{\overline{B} + C} + \overline{A + B} + C \\
= \quad & \\
& A\overline{C} + B\overline{C} + \overline{A + B} + C \\
= \quad & \\
& (A + B)\overline{C} + \overline{A + B} + C \\
= \quad & \\
& A + B + \overline{A + B} + C \\
= \quad & \\
& 1 + C = 1
\end{aligned}
$$

donc $\models F$.

Exercice (corrigé) 1.11. *Les formules propositionnelles suivantes sont-elles des tautologies ?*

1. $(p \wedge q) \Rightarrow p$
2. $(p \vee q) \Rightarrow (p \wedge q)$
3. $(p \wedge q) \Rightarrow (p \vee q)$
4. $p \Rightarrow (p \vee q)$
5. $p \Rightarrow ((\neg p) \Rightarrow p)$
6. $p \Rightarrow (p \Rightarrow q)$
7. $p \Rightarrow (p \Rightarrow p)$
8. $(p \Rightarrow q) \Rightarrow ((q \Rightarrow r) \Rightarrow (p \Rightarrow r))$

Réponses : 1., 3., 4., 5., 7. et 8. sont des tautologies. L'établir (en calculant l'expression booléenne associée) est laissé en exercice.

III.4 Théorème de substitution

PROPRIÉTÉ 1.4 (THÉORÈME DE SUBSTITUTION) : Soit F une formule propositionnelle dans laquelle interviennent les variables propositionnelles P_1, P_2, P_3,..., P_n. Supposons que l'on remplace ces variables par des formules propositionnelles G_1, G_2, G_3,..., G_n. La nouvelle formule propositionnelle obtenue est notée F^*.

Dans ces conditions, si $\models F$, alors $\models F^*$.

PREUVE F étant une tautologie, sa fonction de vérité ne dépend pas des valeurs de vérité des variables booléennes, qui peuvent donc être remplacées par n'importe quelle fonction booléenne. ∎

Attention, la réciproque n'est pas vraie ...

EXEMPLE 1.12. Soit $F = A \Rightarrow B$ et $F^* = P \wedge \neg P \Rightarrow Q$, obtenue à partir de F en remplaçant A par $P \wedge \neg P$ et B par Q. Comme *expr-bool*$(F^*) = \overline{P \cdot \overline{P}} + Q = \overline{0} + Q = 1 + Q = 1$, alors F^* est une tautologie. Cependant de *expr-bool*$(F) = \overline{A} + B$, on ne peut pas dire que F est une tautologie.

Exemple d'utilisation de ce résultat :

EXEMPLE 1.13. La formule propositionnelle

$$G = ((P \Rightarrow Q \wedge \neg R) \vee (\neg S \Leftrightarrow T)) \Rightarrow ((P \Rightarrow Q \wedge \neg R) \vee (\neg S \Leftrightarrow T))$$

est compliquée puisqu'elle contient 5 variables propositionnelles. Il y a donc 32 lignes à calculer dans sa table de vérité. Cependant, il suffit de remarquer que G est obtenue par substitution à partir de $F = A \Rightarrow A$, en remplaçant A par $((P \Rightarrow Q \wedge \neg R) \vee (\neg S \Leftrightarrow T))$. Puisque F est une tautologie, F^* en est une aussi.

Ce résultat peut évidemment être appliqué aussi à des parties de formules propositionnelles, pour accélérer le calcul de leurs fonctions de vérité : si une partie d'une formule propositionnelle constitue à elle seule une tautologie, la partie correspondante de la fonction de vérité peut être avantageusement remplacée par 1.

III.5 Conclusion

Le calcul sur les fonctions de vérité paraît satisfaisant et séduisant, lorsqu'il s'agit de calculer des valeurs de vérité ou d'examiner des conséquences logiques. Il est vrai qu'il est simple, qu'il nécessite un minimum de réflexion (très important dans le cas des ordinateurs !) et qu'il est très facile à programmer.

Mais, pour une formule propositionnelle qui comporte 10 variables propositionnelles (ce qui n'est pas beaucoup pour les problèmes que l'on cherche à programmer !), la table des valeurs de la fonction de vérité comporte $2^{10} = 1024$ lignes. Celui qui opère à la main a déjà démissionné. L'ordinateur démissionne un peu plus loin, certes, mais il finit aussi par avouer son incapacité :

— Sur les machines modernes, il n'est plus impossible d'envisager d'écrire et d'exécuter une « boucle vide » qui porte sur toutes les valeurs entières représentables sur 32 bits, donc de 0 à $2^{32} - 1$. Le temps d'exécution est récemment devenu raisonnable.

— Il ne faut cependant pas exiger que ce temps demeure raisonnable dès qu'il s'agit d'exécuter un algorithme un peu compliqué. Et 32 variables constituent un nombre ridiculement petit pour un système expert, dans lequel les expressions offrent souvent une complexité qui n'a aucune commune mesure avec ce que l'on peut imaginer de plus compliqué ...

Les « raccourcis » qui viennent d'être étudiés et qui permettent d'accélérer, voire de supprimer totalement, le calcul d'une fonction de vérité, sont plus utiles lorsque l'on opère « à la main » que pour la programmation d'algorithmes de logique.

Il faut donc garder en réserve la méthode des fonctions de vérité : celle-ci peut être très utile dans certains cas, essentiellement lorsque le problème peut être résolu « à la main », mais il faut aussi trouver une autre méthode pour songer à aborder des problèmes plus complexes. Une telle méthode sera présentée dans le chapitre 4.

Chapitre 2

Déductions et démonstrations

I Présentation de la théorie de la démonstration

Il s'agit ici d'explorer les mécanismes du raisonnement humain, c'est-à-dire les schémas de pensée qui nous permettent de décider d'agir d'une certaine manière, dans le but d'obtenir un certain résultat.

Nous disposons en fait d'une « base de connaissances » qui fait que nous savons que, dans telles ou telles circonstances, certaines causes produisent certains effets. Pour obtenir ces effets, nous essayons de nous replacer dans les conditions de leur réalisation.

I.1 Système formel et règles d'inférence

Un système formel qui est chargé de reproduire mécaniquement ce fonctionnement est constitué d'*axiomes logiques*, qui jouent le rôle de « connaissances de base » ; il s'agit de formules de logique qui servent de « points de départ » aux déductions ultérieures.

Puis, le système est muni de « règles d'inférence », qui sont chargées de simuler les divers modes de raisonnement que nous utilisons. Elles se présentent sous la forme de règles qui, à partir de formules d'une certaine forme, autorisent la production de nouvelles formules qu'on veut considérer comme vraies chaque fois que les formules dont elles sont issues le sont. Ces formules auront alors le statut de nouveaux résultats considérés comme établis, et pourront venir enrichir la « base de connaissances ». On les appellera des *théorèmes logiques*.

I.2 Statut des axiomes logiques

Il convient de distinguer les axiomes logiques des autres axiomes qui peuvent être posés dans divers domaines, par exemple, en géométrie, l'axiome d'Euclide ; ces axiomes n'appartiennent pas à la logique et autorisent la construction d'une théorie (la géométrie euclidienne en l'occurrence).

Il faut aussi les distinguer des axiomes posés au sujet de la logique elle-même, comme, par exemple, celui que nous avons appelé le principe de non-contradiction. Ces axiomes énoncés *au sujet de* la logique, ne sont pas non plus des axiomes logiques.

Ils jouent le même rôle, pour la logique, que l'axiome d'Euclide pour la géométrie : ils conduisent à la construction d'une certaine logique, étant bien entendu que d'autres axiomes

peuvent conduire à la construction d'autres logiques que celle qui est l'objet du présent chapitre.

Plus précisément, les définitions suivantes donnent un sens très précis aux notions de "théorème", de "démonstration" et d'"axiome". Il faut les lire plusieurs fois et s'y reporter dans la suite, pour bien les assimiler.

DÉFINITION 2.1 (THÉORÈME LOGIQUE). *Un résultat obtenu par une déduction correcte ou une séquence de déductions correctes (c'est-à-dire qui utilisent explicitement les règles d'inférence autorisées) à partir des axiomes logiques et, éventuellement, d'autres résultats du même type déjà établis par ailleurs s'appelle un* théorème logique.

DÉFINITION 2.2 (DÉMONSTRATION). *La séquence de déductions qui conduit à un théorème logique est appelée* démonstration *de ce résultat.*

DÉFINITION 2.3 (AXIOME LOGIQUE). *Cette séquence peut éventuellement ne comporter qu'un seul élément. Dans ce cas, le « résultat » est un* axiome logique.

Un axiome logique est donc un théorème logique, et il ne se démontre pas.

Pour abréger le discours, mais aussi pour apprendre à le formaliser, nous introduisons une notation fondamentale. Cette notation sera présente dans toutes les parties de ce cours. Elle est le principal point commun entre ses chapitres. Chaque fois que vous rencontrerez cette notation et que vous avez un doute sur sa signification, revenez lire cette partie du cours pour lever ce doute.

NOTATION : On exprime que la formule F est un théorème par la notation : $\vdash F$, qui se lit « F est un théorème ».

Il est possible d'utiliser des formules logiques supplémentaires (autres que des axiomes ou des théorèmes) et de mener un raisonnement correct à partir de ces formules (et des axiomes et des théorèmes déjà connus). On parle alors de **démonstration sous hypothèses**.

NOTATION : L'affirmation "la formule logique C est démontrée sous les hypothèses H_1, H_2, $\ldots$, H_n" est notée $\{H_1, H_2, \ldots, H_n\} \vdash C$.

Dans cette notation, les accolades $\{$ et $\}$ ne sont pas indispensables. Elles sont souvent omises, la signification étant la même.

II Axiomes et règle d'inférence du système formel « *LP* »

Il existe plusieurs systèmes d'axiomes qui permettent de définir la logique propositionnelle. Nous nous en tiendrons à l'ensemble d'axiomes suivant, qui n'est ni minimal, ni contradictoire.

II.1 Axiomes

Axiomes relatifs à l'implication

— Axiome 1 : $P \Rightarrow (Q \Rightarrow P)$
— Axiome 2 : $(P \Rightarrow Q) \Rightarrow ((P \Rightarrow (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R))$

Axiomes relatifs à la conjonction

— Axiome 3 : $P \Rightarrow (Q \Rightarrow P \wedge Q)$
— Axiome 4 : $P \wedge Q \Rightarrow P$
— Axiome 5 : $P \wedge Q \Rightarrow Q$

Axiomes relatifs à la disjonction

— Axiome 6 : $P \Rightarrow P \vee Q$
— Axiome 7 : $Q \Rightarrow P \vee Q$
— Axiome 8 : $(P \Rightarrow R) \Rightarrow ((Q \Rightarrow R) \Rightarrow (P \vee Q \Rightarrow R))$

Axiomes relatifs à la négation

— Axiome 9 : $\neg\neg P \Rightarrow P$
— Axiome 10 : $(P \Rightarrow Q) \Rightarrow ((P \Rightarrow \neg Q) \Rightarrow \neg P)$

Axiomes relatifs à l'équivalence

— Axiome 11 : $(P \Rightarrow Q) \Rightarrow ((Q \Rightarrow P) \Rightarrow (P \Leftrightarrow Q))$
— Axiome 12 : $(P \Leftrightarrow Q) \Rightarrow (P \Rightarrow Q)$
— Axiome 13 : $(P \Leftrightarrow Q) \Rightarrow (Q \Rightarrow P)$

II.2 Une seule règle d'inférence

Ce système formel n'utilise que la règle d'inférence suivante.

« modus ponendo ponens » (le mode « en posant, on pose ») :
$\{P, P \Rightarrow Q\} \vdash Q$. « Des formules P et $P \Rightarrow Q$, on peut déduire par modus ponens la formule Q ».

Cette règle est plus implement appelée « modus ponens ».

II.3 Définition

DÉFINITION 2.4 (SYSTÈME FORMEL « *LP* »). *Le système formel composé des 13 axiomes précédents et du modus ponens est nommé « LP ».*

Ce système formel est une variante du système axiomatique proposé par Hilbert et Ackermann en 1934, qui comportait 15 axiomes.

II.4 Remarque

On peut définir d'autres systèmes formels en ajoutant à *LP* les règles d'inférence suivantes :

« modus tollendo tollens » (le mode « en supprimant, on supprime ») :
$\{P \Rightarrow Q, \neg Q\} \vdash \neg P$.

« modus ponendo tollens » (le mode « en posant, on supprime ») :
$\{\neg(P \wedge Q), P\} \vdash \neg Q$.

« modus tollendo ponens » (le mode « en supprimant, on pose ») :
$\{P \vee Q, \neg P\} \vdash Q$.

REMARQUE 2.1. Les noms de ces règles sont des noms latins qui ont été proposés par les philosophes du XVI$^{\text{ème}}$ siècle.

III Démonstrations avec ou sans hypothèses

Un raisonnement logique peut être rédigé sous forme de démonstration, soit d'un théorème, soit d'une conséquence de certaines hypothèses.

III.1 Démonstration d'un théorème

La démonstration d'un théorème est constituée :

1. d'un en-tête, portant l'indication « Démonstration » (de manière à l'isoler totalement du contexte),
2. puis d'un certain nombre de lignes, numérotées (pour pouvoir être référencées dans la suite). Chacune d'entre elles doit comporter deux champs :
   — le premier indique pourquoi le résultat suivant peut être avancé (il est indispensable pour pouvoir juger de la correction de la démonstration) ;
   — le second consiste en une formule, qui est le « résultat » de la ligne courante.
3. La conclusion de la démonstration est la formule écrite sur la dernière ligne. C'est le théorème qu'il fallait démontrer.
4. Elle est répétée dans une dernière ligne, non numérotée, qui porte l'en-tête « Conclusion ».

Dans une ligne, on peut avancer :
— un axiome,
— un théorème considéré comme connu (dont la démonstration a été vue par ailleurs),
— le résultat de l'application d'une règle d'inférence sur des formules qui sont écrites dans les lignes précédentes.

III.2 Démonstration sous hypothèses

Une démonstration sous hypothèses ...

1. Commence par une première ligne qui comporte les mots « Démonstration sous les hypothèses ».
2. Cette première ligne est suivie de l'écriture de l'ensemble des hypothèses utilisées.
3. Puis, comme dans une démonstration de théorème, de lignes numérotées dans lesquelles peuvent figurer les mêmes éléments, auxquels il faut ajouter les hypothèses, dont on a le droit de se servir comme s'il s'agissait de résultats établis.
4. La ligne indiquant la conclusion doit rappeler les hypothèses.

III.3 Exemple 1

Voici, par exemple, comment on peut démontrer la règle du « modus (tollendo) tollens » à l'aide de la règle du « modus ponens » et des axiomes. Il s'agit de déduire $\neg P$ sous les hypothèses $P \Rightarrow Q$ et $\neg Q$.

EXEMPLE 2.1 (MODUS (TOLLENDO) TOLLENS).

Démonstration sous les hypothèses $\{P \Rightarrow Q\,,\, \neg Q\}$

1	Axiome 10	$(P \Rightarrow Q) \Rightarrow ((P \Rightarrow \neg Q) \Rightarrow \neg P)$
2	Hypothèse	$P \Rightarrow Q$
3	m.p. sur 1, 2	$(P \Rightarrow \neg Q) \Rightarrow \neg P$
4	Axiome 1	$\neg Q \Rightarrow (P \Rightarrow \neg Q)$
5	Hypothèse	$\neg Q$
6	m.p. sur 4, 5	$(P \Rightarrow \neg Q)$
7	m.p. sur 3, 6	$\neg P$

Conclusion : $\{P \Rightarrow Q\,,\, \neg Q\} \vdash \neg P$.

"m.p." est une abréviation pour "modus ponens". Observez attentivement cette démonstration et sa structuration en trois colonnes. Vérifiez que la deuxième colonne donne toujours une explication correcte de la formule déduite dans la troisième colonne.

Ensuite, vous pouvez vous demander, à juste titre, pourquoi la première ligne de cette preuve utilise l'axiome 10 et pas un autre. Il n'y a pas de méthode générale pour trouver les étapes d'une preuve, mais, dans cet exemple, on peut expliquer pourquoi l'auteur de cette preuve a choisi l'axiome 10, comme ceci :

On cherche à démontrer un théorème comportant des négations. Or, seuls les axiomes 9 et 10 contiennent des négations. L'axiome 9 ne fait qu'éliminer deux négations successives. Cela ne semble pas approprié pour prouver $\neg P$ à partir de $\neg Q$. Il ne reste que l'axiome 10. On constate aussi que l'axiome 10, tel que, contient les hypothèses $P \Rightarrow Q$ et $\neg Q$.

Puis les lignes 2 et 3 utilisent l'hypothèse $P \Rightarrow Q$ et la règle de modus ponens pour simplifier ce qui reste à prouver. On peut appliquer la règle de modus ponens $A, A \Rightarrow B \vdash B$ en posant $A = (P \Rightarrow Q)$ et $B = ((P \Rightarrow \neg Q) \Rightarrow \neg P)$.

De même, expliquez la suite de cette preuve.

III.4 Exemple 2

Voici un deuxième exemple. C'est la démonstration du théorème appelé *théorème de réflexivité de l'implication* $\vdash (P \Rightarrow P)$.

EXEMPLE 2.2 (THÉORÈME DE RÉFLEXIVITÉ DE L'IMPLICATION).

Démonstration :

1	Axiome 2	$(P \Rightarrow (Q \Rightarrow P)) \Rightarrow ((P \Rightarrow ((Q \Rightarrow P) \Rightarrow P)) \Rightarrow (P \Rightarrow P))$ (en remplaçant Q par $Q \Rightarrow P$ et R par P)
2	Axiome 1	$P \Rightarrow (Q \Rightarrow P)$
3	m.p. sur 1, 2	$(P \Rightarrow ((Q \Rightarrow P) \Rightarrow P)) \Rightarrow (P \Rightarrow P)$
4	Axiome 1	$P \Rightarrow ((Q \Rightarrow P) \Rightarrow P)$ (en remplaçant Q par $Q \Rightarrow P$)
5	m.p. sur 3, 4	$(P \Rightarrow P)$

Conclusion : $\vdash (P \Rightarrow P)$

III.5 Contraposée

DÉFINITION 2.5 (CONTRAPOSÉE). *L'implication* $\neg Q \Rightarrow \neg P$ *est appelée* contraposée *de l'implication* $P \Rightarrow Q$.

PROPRIÉTÉ 2.1 (THÉORÈME DE LA CONTRAPOSÉE) : Le théorème
$$\vdash (P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$$
peut être utilisé sous le nom de *théorème de la contraposée*.

IV Méta-théorème de la déduction

En logique, un *méta*-théorème est une affirmation *à propos* d'une logique formelle. Ce n'est pas un théorème de cette logique. Touts les systèmes formels déductifs que nous étudions vérifient le méta-théorème suivant. Il relie les formules dont le symbole principal est $\Rightarrow$ et les déductions dans ces systèmes formels.

PROPRIÉTÉ 2.2 (MÉTA-THÉORÈME DE LA DÉDUCTION) : Pour toutes les formules propositionnelles H et C, $H \vdash C$ est un théorème si et seulement si $H \Rightarrow C$ est un théorème.

Ce théorème se généralise à n hypothèses comme suit.

PROPRIÉTÉ 2.3 (MÉTA-THÉORÈME GÉNÉRAL DE LA DÉDUCTION) :

$$\{H_1, H_2, \ldots, H_n\} \vdash C \text{ si et seulement si } \{H_1, H_2, \ldots, H_{n-1}\} \vdash H_n \Rightarrow C$$

Ce théorème est admis sans démonstration. Sa preuve est possible, mais plus ou moins facile, dans chacun des systèmes déductifs connus pour les formules propositionnelles.

REMARQUE 2.2. L'utilisation principale du méta-théorème de la déduction consiste à remplacer la démonstration d'une implication par une démonstration sous hypothèses.

EXEMPLE 2.3. Soit à démontrer : $\vdash (A \Rightarrow (B \Rightarrow C)) \Rightarrow (B \Rightarrow (A \Rightarrow C))$.

La démonstration de ce théorème équivaut à la démonstration de

$$\{A \Rightarrow (B \Rightarrow C)\} \vdash (B \Rightarrow (A \Rightarrow C)).$$

Cette dernière est équivalente à la démonstration de

$$\{A \Rightarrow (B \Rightarrow C), B\} \vdash (A \Rightarrow C),$$

elle-même équivalente à la démonstration

$$\{A \Rightarrow (B \Rightarrow C), B, A\} \vdash C.$$

Etablissons ce résultat dans le système formel "*LP*" :

Démonstration sous les hypothèses $A \Rightarrow (B \Rightarrow C)$, B et A

1	Hypothèse 3	A
2	Hypothèse 1	$A \Rightarrow (B \Rightarrow C)$
3	m.p. sur 1, 2	$B \Rightarrow C$
4	Hypothèse 2	B
5	m.p. sur 3, 4	C

Conclusion $\{A \Rightarrow (B \Rightarrow C), B, A\} \vdash C$.

EXEMPLE 2.4. On a démontré le « modus (tollendo) tollens » :

$$\{P \Rightarrow Q, \neg Q\} \vdash \neg P.$$

D'après le méta-théorème de la déduction, ce dernier résultat est équivalent à :

$$\{P \Rightarrow Q\} \vdash (\neg Q \Rightarrow \neg P).$$

Une nouvelle application de ce même méta-théorème donne :

$$\vdash (P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P).$$

V Quelques théorèmes classiques

Nous avons déjà évoqué le théorème de réflexivité de l'implication ($\vdash \quad P \Rightarrow P$) et le théorème de la contraposée. Citons encore ...

V.1 Théorème de transitivité de l'implication

PROPRIÉTÉ 2.4 (THÉORÈME DE TRANSITIVITÉ DE L'IMPLICATION) : On a
$$\vdash (A \Rightarrow B) \Rightarrow ((B \Rightarrow C) \Rightarrow (A \Rightarrow C)).$$

V.2 Théorème de la contradiction

PROPRIÉTÉ 2.5 (THÉORÈME DE LA CONTRADICTION) : Soit A et B deux formules propositionnelles quelconques.
$$\vdash \neg A \Rightarrow (A \Rightarrow B)$$

Intuitivement, cela signifie que si $\neg A$ et A sont établies, alors on peut en déduire n'importe quoi (B).

VI Méthodes de démonstration

Il peut sembler *a priori* difficile de trouver les idées qui permettent de mener à bien une démonstration sous hypothèses.

On pourra s'inspirer des principes suivants qui sont souvent d'une aide précieuse bien qu'on ne puisse affirmer qu'ils fournissent dans tous les cas la solution la plus courte, ni même ne suffisent toujours pour trouver une solution.

La première chose à faire est de faire intervenir au maximum le théorème de la déduction. On conseille ensuite d'observer...

Les hypothèses. Une hypothèse (ou un résultat intermédiaire) qui se présente sous la forme d'une

conjonction : s'utilise par application des axiomes 4 et 5

disjonction : s'utilise par une disjonction des cas (axiome 8)

négation : s'utilise par application du théorème de la contraposée

implication logique : souvent la technique de l'hypothèse supplémentaire (utilisée aussi dans le cas précédent) permet de l'utiliser.

équivalence logique : s'utilise par application des axiomes 12 et 13

L'application de ces quelques principes permet en général de bien démarrer les démonstrations, par leurs premières lignes. On observe ensuite...

La conclusion. Si elle se présente sous la forme d'une

conjonction : elle peut s'obtenir par application de l'axiome 3.

disjonction : elle peut s'obtenir par application de l'axiome 6 ou de l'axiome 7.

négation : elle peut s'obtenir par une réduction à l'absurde.

implication : vous n'avez pas suivi les conseils d'utiliser le théorème de la déduction au maximum !

équivalence logique : elle peut s'obtenir par application de l'axiome 11 (concrètement, on effectue en général séparément les démonstrations des deux implications et on peut s'abstenir de « reconstruire » l'équivalence logique à partir des deux implications, de l'axiome 11 et de deux « modus ponens »).

L'application de ces quelques principes permet en général de découvrir un bon chemin vers le résultat, c'est-à-dire les dernières lignes de la démonstration.

Au total, sauf dans des cas très compliqués, on devrait avoir ainsi l'intégralité de la démonstration.

VII Théorème de complétude du calcul propositionnel

On a jusqu'à maintenant deux points de vue :

1. La théorie des valeurs de vérité, avec ses
   — tables de vérités,
   — fonctions de vérités,
   — tautologie, conséquence, hypothèse.
2. La théorie de la démonstration, avec ses
   — axiomes,
   — règles d'inférence,
   — démonstrations (ou démonstrations sous hypothèses).

On peut se demander si les résultats obtenus dans chacune des deux théories sont identiques, c'est-à-dire faire l'étude de la complétude du calcul propositionnel.

VII.1 Théorème de complétude

Le théorème de complétude du calcul propositionnel s'énonce ainsi :

PROPRIÉTÉ 2.6 (THÉORÈME DE COMPLÉTUDE) : Tout théorème est une tautologie et réciproquement, soit :
$\vdash F$ si et seulement si $\models F$.

Autrement dit, les deux points de vue (théorie des valeurs de vérité, théorie de la démonstration) sont équivalents pour les théorèmes et les tautologies, en logique des propositions.

VIII Conclusion

Quand on cherche une démonstration selon le système *LP*, on a parfois des difficultés pour choisir l'étape suivante. C'est tout à fait normal. Ce n'est pas un problème d'incompréhension. C'est la méthode qui n'est tout simplement pas complètement automatisable. Vous apprendrez une autre méthode, automatisable, dans le chapitre 4.

Dans les sujets d'examens, l'énoncé donne toutes les étapes difficiles des démonstrations à faire selon le système *LP*. Il s'agit donc de compléter une démonstration "à trous" par des étapes faciles qui manquent. Ainsi, l'examen n'évalue pas votre ingéniosité, mais seulement la compréhension et l'application correcte des principes de construction d'une démonstration formelle.

En contrepartie, pour exercer votre ingéniosité, certains exercices et certaines parties des devoirs demandent de faire des démonstrations moins faciles, pour vous faire progresser dans l'art délicat de la démonstration.

Chapitre 3

Calcul des prédicats

I Introduction

I.1 Insuffisances de la formalisation en calcul propositionnel

La logique des propositions ne s'occupe, on le sait, que des liens entre propositions réalisés à l'aide des connecteurs logiques.

Autrement dit, une fois qu'une proposition « complexe » a été analysée suivant ces connecteurs, il subsiste des « atomes » (non sécables en logique des propositions) et l'analyse ne peut pas être poussée plus profondément, alors qu'il existe cependant encore des « relations » entre ces atomes, relations que la logique des propositions ne permet pas de prendre en compte.

I.1.1 Introduction des « prédicats »

La proposition « Pierre est le père de Marc » ne comporte aucun connecteur logique, elle n'est plus analysable en logique des propositions, on ne peut la formaliser que par une variable propositionnelle, par exemple : A. Alors que la proposition « Jean est le père de Sylvie » entretient des rapports évidents avec la précédente, elle ne peut être formalisée en logique des propositions, elle aussi, que par une (autre) variable propositionnelle, par exemple : B.

Après la formalisation, donc, on se retrouve avec deux variables propositionnelles A et B, sans lien aucun entre elles, alors qu'il est évident que ces deux propositions évoquent un même lien de parenté (la paternité), simplement entre des individus différents.

Ce premier exemple suggère la nécessité, pour poursuivre l'analyse des propositions, de la création d'un langage qui permettrait de décrire des propriétés accordées (ou non) à des individus, ou les reliant. Cette préoccupation est aussi celle des mathématiciens, qui s'occupent, par exemple, de décrire les propriétés des entiers naturels.

Il faudra donc envisager des « variables » , dans un certain domaine, et, pour certaines « valeurs » de ces variables, certaines propriétés seront « vraies » ou « fausses ».

I.1.2 Introduction de l'« univers du discours »

Soit la proposition « 7 n'a pas de racine carrée ». En calcul propositionnel, elle ne peut être formalisée que par une variable propositionnelle. De plus, il semble même difficile de lui

accorder une « valeur de vérité » ; en effet, si l'on se place dans l'ensemble des entiers naturels, cette proposition est « vraie », alors que si l'on se place dans l'ensemble des nombres réels, elle est évidemment fausse.

Dans cet exemple, comme dans celui de la section précédente, la proposition évoque une propriété, accordée ou refusée, à un objet bien précis (7). Bien entendu, la valeur de vérité dépend de la valeur de cet objet, mais aussi de l'ensemble dans lequel ces valeurs peuvent être choisies.

On pourra envisager de formaliser (partiellement) une telle proposition par une expression du type « x a une racine carrée », et la valeur de vérité de la proposition obtenue en « donnant à x une valeur » dépend, non seulement de cette valeur, mais aussi de l'ensemble dans lequel cette valeur peut être prise.

D'une manière générale, et avant de donner une définition précise de ces concepts, on dira qu'une « propriété » possédée (ou non) par un objet est un prédicat. La valeur de vérité de la proposition obtenue en appliquant ce prédicat à un (ou plusieurs) objets dépend de l'univers du discours, c'est à dire de l'ensemble des valeurs reconnues comme possibles (par exemple, dans l'exemple précédent, $\mathbb{N}$ ou $\mathbb{R}$), et aussi, bien entendu, de cette valeur elle-même.

I.1.3 Introduction de la « quantification »

Considérons les propositions « Tous les étudiants sont sérieux » et « certains étudiants sont sérieux ». La logique des propositions est incapable de faire apparaître le lien manifeste qui existe entre elles. Elle ne pourra les formaliser que par A et B, parce qu'elles sont différentes et que l'une n'est pas la négation de l'autre.

La généralisation du calcul propositionnel suggérée ici est encore plus grande que dans les exemples précédents, puisque la propriété évoquée (« être sérieux ») est accordée, par ces propositions, non pas seulement à un individu bien précis, mais à certains individus, considérés dans leur ensemble, ou même à toute une catégorie d'individus, sans qu'aucun ne soit nommément désigné.

Ce troisième exemple suggère la notion de quantification d'une variable (tous les…, certains…).

L'objet du calcul des prédicats est d'étudier et de formaliser les notions qui viennent d'être brièvement évoquées, et que le calcul propositionnel ne permet pas de faire apparaître.

I.2 Sujets et individus

Le calcul des prédicats fait intervenir des variables, qui prennent des valeurs dans un certain ensemble appelé, on l'a dit, univers du discours.

Les éléments de cet univers sont appelés *sujets*. Un sujet distingué ou *individu* est une instance de l'univers du discours qui peut prendre la place du sujet.

EXEMPLE 3.1. Le symbole 7 est un individu, ou un sujet distingué, dans l'univers du discours des entiers naturels.

I.3 Groupes opératoires et termes

Il existe deux manières de distinguer un sujet de l'univers du discours.

— le nommer (donner explicitement son nom, ou le symbole qui le représente), comme dans l'exemple précédent, pour l'entier 7.
— le désigner indirectement par une « propriété » caractéristique (qu'il doit donc être le seul à posséder, de manière à ce qu'il soit identifié sans aucune ambiguïté). Par exemple, si *Pierre* est le père de *Jean*, il est possible de distinguer *Pierre* sans le nommer explicitement par la locution « *le père de Jean* ».

De manière tout à fait précise :

DÉFINITION 3.1 (OPÉRATION n-AIRE). *Soit $\mathcal{U}$ l'univers du discours. On appelle* opération n-aire *définie sur $\mathcal{U}$ toute application de $\mathcal{U}^n$ dans $\mathcal{U}$.*

Par la suite, ou dira avec le même sens que f est d'arité n. Soit f une telle opération n-aire.

DÉFINITION 3.2 (GROUPE OPÉRATOIRE). *L'expression $f(x_1, x_2, \ldots, x_n)$ est appelée* groupe opératoire *à n places. Dans le calcul des prédicats, le symbole de l'application f est plutôt appelé symbole fonctionnel.*

Comme toute application donne une image et une seule de tout élément de son domaine, un groupe opératoire à n places représente bien, sans ambiguïté, un individu de l'univers du discours.

EXEMPLE 3.2. L'addition ordinaire des entiers est une opération binaire sur $\mathbb{N}$ et $+(x, y)$.

Le groupe opératoire correspondant désigne, de manière unique, un élément de cet ensemble, dès que les variables x et y ont été remplacées dans son expression par des sujets distingués (par une méthode ou une autre).

Ainsi, $+(5, 2)$ est l'individu qui peut aussi être représenté par 7.

EXEMPLE 3.3. Le groupe opératoire à 1 place *pere*(x) désigne sans ambiguïté un et un seul individu dans l'univers des êtres humains, dès que la variable x a été remplacée par un sujet distingué.

Si *Jean* est bien un nom d'individu (c'est à dire, désigne bien un individu unique), s'il en est de même de *Pierre*, et si *Pierre* est le père de *Jean*, alors *pere*(*Jean*) distingue l'individu *Pierre*.

Lorsque l'expression $f(x_1, \ldots, x_n)$ ne dépend pas de $x_1, \ldots, x_n$, on peut la remplacer par une constante c de l'univers. On remarque qu'une constante est un groupe opératoire zéro-aire.

DÉFINITION 3.3 (TERME). *En calcul des prédicats, on fait intervenir des groupes opératoires à n places. Le résultat est un* terme *du calcul des prédicats qui est défini de manière récursive par :*

— *une variable,*
— *une constante (un groupe opératoire zéro-aire),*
— *un groupe opératoire n-aire $(n \geqslant 1)$ $f(t_1, t_2, \ldots, t_n)$, dans lequel $t_1, t_2, \ldots, t_n$ sont des termes et f un symbole fonctionnel.*

I.4 Prédicats et atomes

Il s'agit ici de faire intervenir les relations qui peuvent lier des individus de l'univers du discours. Par exemple, « 22 est le double de 11 », « 46 est le double de 45 » sont des propositions (vraies ou fausses !) qui évoquent la relation « être le double de » .

DÉFINITION 3.4 (RELATION n-AIRE). *On appelle* relation n-aire *définie sur l'univers du discours $\mathcal{U}$ toute application de $\mathcal{U}^n$ dans $\{0,1\}$ (ou $\{$faux,vrai$\}$).*

DÉFINITION 3.5 (PRÉDICAT). *On appelle* prédicat à n places, *ou* prédicat de poids n, *ou encore* prédicat d'arité n, *tout symbole r tel que l'expression $r(t_1, t_2, \ldots, t_n)$, dans laquelle $t_1, t_2, \ldots, t_n$ sont des termes, s'interprète comme une relation n-aire entre les interprétations des termes $t_1, t_2, \ldots, t_n$.*

Par extension et par convention, un prédicat zéro-aire (sans « variables ») est une proposition (qui a donc une valeur de vérité).

DÉFINITION 3.6 (ATOME). *On appelle* formule atomique, *ou* atome, *toute formule de la forme $P(t_1, \ldots, t_n)$ où P est un prédicat et $t_1, \ldots, t_n$ sont des termes.*

Par exemple $\mathit{mange}(\mathit{chat}, \mathit{souris})$ est un atome, tandis que $\neg\, \mathit{mange}(\mathit{grenouille}, \mathit{éléphant})$ n'est pas un atome.

I.5 Quantificateur universel

Considérons la proposition « Tous les étudiants travaillent les mathématiques » pour l'analyser du point de vue du calcul des prédicats. Aucun connecteur logique n'apparaît. Par contre, on peut remarquer l'intervention d'un prédicat (de poids deux) qui peut être formalisé par $\mathit{travaille}(x, y)$, et qui signifie qu'un individu x travaille une certaine matière y.

Il faut préciser l'univers du discours, il s'agit ici d'un ensemble produit cartésien de deux ensembles :

— le premier est l'ensemble des étudiants,
— le second un ensemble de noms de matières (celles qui sont susceptibles d'être enseignées).

Il est bien clair qu'il s'agit d'une proposition. Or, si la seconde variable (y) est bien remplacée par un symbole d'individu ($maths$), aucun nom, explicite ou implicite, n'est donné pour désigner un étudiant particulier : on a « $\mathit{travaille}(x, maths)$ » sans que l'on sache « qui est x » .

Il s'agit d'une construction particulière, qui permet d'obtenir des propositions à partir de prédicats sans qu'une variable soit distinguée, et qui porte le nom de quantification : le sujet x n'est pas distingué, il est *quantifié*.

La notation utilisée pour cette quantification est « $\forall$ » qui représente un A à l'envers (pour *All*). Ainsi on représente la proposition ci dessus par

$$\forall x \,.\, \mathit{travaille}(x, \mathit{maths}).$$

DÉFINITION 3.7 (QUANTIFICATEUR UNIVERSEL). *$\forall$ est un symbole de quantificateur, appelé le* quantificateur universel. *On dit alors que $\forall x$ est le quantificateur universel de la variable x.*

REMARQUE 3.1. La présence du symbole de quantification est indispensable pour donner du sens aux formules avec variables.

— $\forall x \,.\, \textit{travaille}(x, \textit{maths})$ est une proposition
— $\textit{travaille}(x, \textit{maths})$ n'est pas une proposition : on ne peut pas lui attribuer une valeur de vérité. Il est nécessaire de substituer x par un terme distinguant un individu pour que cela le devienne (i.e. $\textit{travaille}(Jean, \textit{maths})$ ou bien $\textit{travaille}(\textit{fils}(\textit{Pierre}), \textit{maths})$). Dans cette expression, la variable x est dite *libre*.

I.6 Quantificateur existentiel

Dans la proposition « Tous les étudiants travaillent au moins une matière », intervient le même prédicat à deux places $\textit{travaille}(x, y)$; on remarque qu'aucun étudiant n'est explicitement nommé, pas plus que la matière qu'il travaille. Si le « Tous » induit une quantification universelle, le « au moins une matière » signifie qu'il existe une matière travaillée par cet étudiant. Il suffit alors d'introduire le quantificateur existentiel, représenté par un E (première lettre de Exists) retourné ($\exists$).

Cet exemple se traduit alors par

$$\forall x \,.\, \exists y \,.\; \textit{travaille}(x, y)$$

où la variable x est liée par le quantificateur universel $\forall x$ et la variable y est liée par le quantificateur existentiel $\exists y$

I.7 Alternance de quantificateurs

Attention : *Il est interdit d'intervertir des quantificateurs de symboles différents* (si l'on désire que la formule conserve le même sens, bien entendu).

EXEMPLE 3.4. La formule $\forall x \,.\, \exists y \,.\, \textit{travaille}(x, y)$ signifie que tout étudiant travaille une matière (au moins). Autrement dit, la matière travaillée dépend de cet étudiant, elle n'est éventuellement pas la même pour tous les étudiants.

La formule $\exists y \,.\, \forall x \,.\, \textit{travaille}(x, y)$ signifie qu'il y a une matière que tous les étudiants travaillent. La différence fondamentale avec le cas précédent est que, dans cette dernière affirmation, on affirme que tous les étudiants travaillent la même matière.

EXEMPLE 3.5. En supposant que (a, b, c) prenne ses valeurs dans un univers du discours qui est une partie de $\mathbb{R}^3$ telle que $b^2 - 4ac > 0$, la formule :

— $\forall a \,.\, \forall b \,.\, \forall c \,.\, \exists x \,.\, ax^2 + bx + c = 0$ est une formule vraie (une équation du second degré dont le discriminant est positif admet des racines réelles, et tout le monde sait que les valeurs de ces racines dépendent de celles de a, de b et de c).
— $\exists x \,.\, \forall a \,.\, \forall b \,.\, \forall c \,.\, ax^2 + bx + c = 0$ est une formule fausse : il n'y a pas de réel qui soit solution de n'importe quelle équation du second degré !

I.8 Formules du calcul des prédicats

DÉFINITION 3.8 (FORMULE). *La définition d'une* formule *est :*

- *Un atome est une formule,*
- *si P est une formule, si Q est un symbole de quantificateur ($\exists$ ou $\forall$) et si x est un symbole de variable, alors $Qx\,.\,P$ est une formule,*
- *si P est une formule, alors $(\neg P)$ est une formule,*
- *si P et Q sont des formules, alors $(P \vee Q)$, $(P \wedge Q)$, $(P \Rightarrow Q)$, $(P \Leftrightarrow Q)$ sont des formules.*
- *Il n'existe pas d'autres manières de construire une formule qu'en appliquant les règles précédentes un nombre fini de fois.*

I.9 Portée d'un quantificateur

DÉFINITION 3.9 (PORTÉE D'UN QUANTIFICATEUR). *La* portée d'un quantificateur *dans une formule du calcul des prédicats est la partie de cette formule couverte par ce quantificateur.*

Par convention, dans l'écriture d'une formule, un quantificateur est moins prioritaire que tout connecteur logique. Ceci signifie que la portée d'un quantificateur va toujours le plus à droite possible, d'après les parenthèses. En cas de doute sur la portée d'un quantificateur, on recommande d'ajouter une paire de parenthèses pour la délimiter.

Par exemple, $\forall x\,.\,P(x) \Rightarrow Q(x)$ correspond à $(\forall x\,.\,P(x) \Rightarrow Q(x))$ et non pas à $(\forall x\,.\,P(x)) \Rightarrow Q(x)$.

II Sémantique

II.1 Valeurs de vérité

Le calcul des prédicats utilise, comme le calcul propositionnel, les connecteurs logiques, et produit des propositions. Il est possible d'étendre la notion de « valeur de vérité » au calcul des prédicats. Mais l'étude de la valeur de vérité d'une formule du calcul des prédicats est beaucoup plus compliquée.

1. Une expression typique (une forme propositionnelle) du calcul propositionnel est $P \Rightarrow Q$. Les « atomes » sont ici des variables propositionnelles qui peuvent être remplacées par n'importe quelle proposition. Quelle que soit cette proposition, sa valeur de vérité ne peut être que « *vrai* » ou « *faux* ». Cela permet de lui associer une simple variable booléenne, sa valeur de vérité, pour obtenir simplement la fonction de vérité $\overline{p} + q$.
2. Une expression analogue (une formule) du calcul des prédicats pourrait être $\forall x\,.\,p(x, y) \Rightarrow q(x, z)$. Les atomes sont ici des prédicats d'arité 2 qui, eux aussi, ne peuvent prendre que les valeurs « *vrai* » ou « *faux* », mais pas indépendamment des individus considérés.

Il n'est donc pas possible de remplacer un atome par une simple variable booléenne. Seule une fonction booléenne (de variables non booléennes) peut convenir, pour faire intervenir les valeurs des variables qui sont les arguments du prédicat.

— si $f(x, y)$ est la fonction booléenne associée au prédicat $p(x, y)$
— si $g(x, z)$ est celle qui est associée à $q(x, z)$,

alors la fonction de vérité de la formule est : $\overline{f(x, y)} + g(x, z)$.

Attention, x, y et z ne sont pas ici des variables booléennes, mais elles prennent leurs valeurs dans l'univers du discours. Il n'est donc pas question de calculer avec x, y et z comme en algèbre de Boole. Le seul moyen, en général, pour étudier une telle fonction de vérité est de construire le tableau de ses valeurs, en donnant à x, y et z successivement toutes les valeurs possibles dans l'univers du discours (si celui-ci est infini, on imagine aisément les problèmes qui vont se poser...).

Les définitions de tautologie et de conséquence logique s'adaptent comme suit.

DÉFINITION 3.10 (TAUTOLOGIE, CONSÉQUENCE LOGIQUE, ÉQUIVALENCE). *si P et Q sont des formules du calcul des prédicats*

— $\models P$ *[P est une tautologie] si et seulement si, pour tous les univers du discours possibles, pour tous les prédicats qui peuvent intervenir dans P, et pour toutes les valeurs des variables dans chacun des univers, la valeur de vérité de P est « vrai ».*
— $\{P\} \models Q$ *[Q est conséquence logique de P] si et seulement si, dans les mêmes conditions que ci-dessus, chaque fois que P est vraie, Q l'est aussi.*
— $P \approx Q$ *[les formules P et Q sont équivalentes] si et seulement si $\{P\} \models Q$ et $\{Q\} \models P$.*

Il reste à donner la définition de la fonction de vérité pour les nouveaux symboles introduits (les quantificateurs).

— La valeur de vérité de $\forall x \,.\, p(x)$ est obtenue en faisant la liste des valeurs de vérité de $p(x)$ pour toutes les valeurs possibles de x dans l'univers du discours (liste effective lorsque c'est possible, ou démonstration). Si, pour toute valeur de x, la valeur de vérité de $p(x)$ est *vrai*, alors la valeur de vérité de $\forall x \,.\, p(x)$ est *vrai*. S'il y a une seule valeur de x pour laquelle la valeur de vérité de $p(x)$ est *faux*, alors la valeur de vérité de $\forall x \,.\, p(x)$ est *faux*.
— La valeur de vérité de $\exists x \,.\, p(x)$ est obtenue en faisant la liste des valeurs de vérité de $p(x)$ jusqu'à ce qu'on trouve *vrai*. Si on trouve *vrai*, la valeur de vérité de $\exists x \,.\, p(x)$ est *vrai*. Si, pour tout élément x de l'univers du discours, la valeur de vérité de $p(x)$ est *faux* (liste effective ou démonstration), alors celle de $\exists x \,.\, p(x)$ est *faux*.

Bien entendu, dans certains cas, il n'est pas nécessaire d'établir effectivement la table de vérité d'une formule du calcul des prédicats pour prouver qu'il s'agit d'une tautologie. Par exemple, il est bien clair que, pour un atome $p(x, y, z)$, on a : $\models (\forall x, y, z \,.\, p(x, y, z)) \Rightarrow (\forall x, y, z \,.\, p(x, y, z))$.

Donnons enfin deux exemples pour lesquels la construction d'une table de vérité n'est pas nécessaire :

EXEMPLE 3.6. Montrer que $\models (\forall x \,.\, p(x, x)) \Rightarrow (\forall x \,.\, (\exists y \,.\, p(x, y)))$.

Soit $\mathcal{U}$ un univers du discours, p une relation binaire sur $\mathcal{U}$.

— Premier cas : dans $\mathcal{U}$, $\forall x \,.\, p(x, x)$ est *faux* ; alors l'implication est vraie.
— Deuxième cas : dans $\mathcal{U}$, $\forall x \,.\, p(x, x)$ est *vrai* ; donc, pour chaque valeur de x dans $\mathcal{U}$, $p(x, x)$ est vrai. Alors $\forall x \,.\, \exists y \,.\, p(x, y)$ est *vrai* car, pour toute valeur de x, il suffit de prendre pour y la même valeur que celle de x pour que $p(x, y)$ soit *vrai*.

EXEMPLE 3.7. Montrer que $(\forall x\,.\,(s(x) \Rightarrow r(x))) \wedge (\exists x\,.\,(\neg r(x) \wedge s(x))$ n'est pas une tautologie.

Pour montrer que cette formule n'est pas une tautologie, il suffit d'exhiber un exemple dans lequel elle est fausse.

$\mathcal{U}$ est un ensemble E, dans lequel on a défini deux parties R et S, telles que $S \subset R$; $r(x)$ est le prédicat « $x \in R$ » et $s(x)$ est le prédicat « $x \in S$ » . Comme $S \subset R$, tout élément extérieur à R ne peut pas être dans S, donc $\exists x\,.\,\neg r(x) \wedge s(x)$ est *faux*, et donc la formule est fausse.

II.2 Simplification de formules quantifiées

De la définition de la valeur de vérité d'une formule quantifiée, on peut déduire

$$\neg(\exists x\,.\,p(x)) \approx (\forall x\,.\,\neg p(x))$$

et

$$\neg(\forall x\,.\,p(x)) \approx (\exists x\,.\,\neg p(x))$$

Pour illustrer cette propriété, voici un exemple : Considérons la proposition (fausse) « l'ensemble des entiers naturels admet un élément maximum ». Pour formaliser cette proposition en calcul des prédicats, il faut introduire le prédicat classique d'inégalité qu'il faudrait, en toute rigueur, noter $\leq (x, y)$, qui signifie que x est inférieur ou égal à y, mais pour lequel nous conserverons la notation usuelle $x \leqslant y$. L'univers du discours est évidemment $\mathbb{N}$, et, pour exprimer que $\mathbb{N}$ admet un élément maximum, on exprime que tout élément de $\mathbb{N}$ est inférieur ou égal à cet élément maximum, soit :

$$\exists M\,.\,\forall n\,.\,n \leqslant M.$$

Pour montrer que cette proposition est fausse, il suffit de démontrer que sa négation est vraie, c'est-à-dire

$$\neg(\exists M\,.\,\forall n\,.\,n \leqslant M).$$

D'après ce que nous venons de voir, cette négation a la même valeur de vérité que

$$\forall M\,.\,\exists n\,.\,\neg(n \leqslant M),$$

ou encore $\forall M\,.\,\exists n\,.\,M < n$. Ceci signifie que, pour tout entier M, on peut trouver un entier n qui est plus grand (il suffit de prendre n égal à $M + 1$).

Voici d'autres résultats d'équivalences entre formules permettant de réduire la portée de quantificateurs.

PROPRIÉTÉ 3.1 (RÉDUCTION DE PORTÉE DE QUANTIFICATEURS) : Soit p et q des prédicats unaires. Alors on a les deux équivalences suivantes :

$$(\forall x\,.\,p(x) \wedge q(x)) \approx \Big((\forall x\,.\,p(x)) \wedge (\forall x\,.\,q(x))\Big) \tag{3.1}$$

$$(\exists x\,.\,p(x) \vee q(x)) \approx \Big((\exists x\,.\,p(x)) \vee (\exists x\,.\,q(x))\Big) \tag{3.2}$$

II.3 Substitutions

Si t est un terme et φ est une formule pouvant contenir la variable x, alors on désigne par $\varphi(t/x)$ le résultat du remplacement de toutes les occurrences libres de x par t dans φ. Le résultat de ce remplacement est une formule qui est une conséquence logique de la formule originale φ si aucune des variables libres de t ne devient liée suite à ce remplacement. On dit alors que t "est substituable à" x dans φ. Pour rendre t substituable à x dans φ, c'est-à-dire pour éviter que des variables libres de t deviennent liées, il suffit de remplacer les noms des variables liées de φ par des noms "frais", c'est-à-dire qui n'apparaissent pas dans les variables libres de t. L'oubli de cette condition est une cause fréquente d'erreurs de raisonnement.

A titre d'exemple, considérez la formule φ définie par $\forall y \,.\, y \leq x$ sur l'univers $\mathcal{U}$.

— Si t est un terme sans la variable libre y, alors $\varphi(t/x)$ signifie juste que t est l'élément maximal.

— A l'opposé, si t est y, la formule $\varphi(y/x)$ est $\forall y \,.\, y \leq y$ qui ne dit plus que y est maximal, mais que la relation $\leq$ est réflexive.

III Formes normales

Cette partie présente présente deux étapes dites de "réduction" de formules, car elles transforment toutes les formules d'un langage du premier ordre en des formules de structure plus simple.

III.1 Forme prénexe

DÉFINITION 3.11 (FORME PRÉNEXE). *Une formule G est en forme prénexe si tous ses quantificateurs ($\forall$ et $\exists$) apparaissent en tête, à gauche, dans cette formule. Elle est ainsi de la forme $Q_1 x_1 \,.\, Q_2 x_2 \,.\, \ldots Q_n x_n \,.\, B$, où Q est $\forall$ ou $\exists$ et B est une formule sans quantificateurs.*

III.1.1 Méthode

La méthode à suivre pour mettre une formule sous forme prénexe est la suivante :

— Réduire les connecteurs : on ne conserve que $\wedge$, $\vee$ et $\neg$. Cette réduction est obtenue en appliquant des équivalences classiques entre formules (établies en logique des propositions, généralisées à toute formule). Par exemple

$$A \Rightarrow B \approx \neg A \vee B$$

et

$$A \Leftrightarrow B \approx (\neg A \wedge \neg B) \vee (A \wedge B)$$

— Renommer les variables liées, de manière à ce que toute variable liée ne le soit qu'une seule fois, et qu'aucune variable liée ne présente d'occurrence libre, d'après les égalités suivantes :

$$\forall x \,.\, A = \forall y \,.\, A(y/x)$$
$$\exists x \,.\, A = \exists y \,.\, A(y/x)$$

On dit qu'on a "polit" la formule, que la formule obtenue est "polie".

— Faire « remonter » les quantificateurs en tête, par les équivalences suivantes :

$$\neg\neg A \approx A$$

$$\neg(\forall x\,.\,A) \approx \exists x\,.\,\neg A$$

$$\neg(\exists x\,.\,A) \approx \forall x\,.\,\neg A$$

et, si x n'est pas variable libre de C,

$$C \vee \forall x\,.\,A \approx \forall x\,.\,(C \vee A)$$

$$C \vee \exists x\,.\,A \approx \exists x\,.\,(C \vee A)$$

$$C \wedge \forall x\,.\,A \approx \forall x\,.\,(C \wedge A)$$

$$C \wedge \exists x\,.\,A \approx \exists x\,.\,(C \wedge A)$$

EXEMPLE 3.8. On considère les propositions suivantes :

P_1 Tout crime a un auteur
P_2 Seuls les gens malhonnêtes commettent des crimes
P_3 On n'arrête que les gens malhonnêtes
P_4 Les gens malhonnêtes arrêtés ne commettent pas de crimes
P_5 Des crimes se produisent

On voudrait en déduire :

Q Il y a des gens malhonnêtes en liberté

Sans anticiper sur les méthodes de résolution, on va ajouter aux propositions P_1 à P_5 la proposition $P_6 \approx \neg Q$ (si on montre que cet ensemble de propositions est « insatisfaisable » , on aura montré que Q est conséquence de P_1 à P_5).

Pour cela, on va considérer les prédicats

— $\mathrm{ar}(x)$: la personne x est arrêtée
— $\mathrm{mal}(x)$: la personne x est malhonnête
— $\mathrm{co}(x, y)$: la personne x commet l'action y
— $\mathrm{cr}(y)$: l'action y est un crime

On obtient

F_1 $\forall y\,.\,(\mathrm{cr}(y) \Rightarrow \exists x\,.\,\mathrm{co}(x, y))$
F_2 $\forall x, y\,.\,(\mathrm{cr}(y) \wedge \mathrm{co}(x, y)) \Rightarrow \mathrm{mal}(x)$
F_3 $\forall x\,.\,\mathrm{ar}(x) \Rightarrow \mathrm{mal}(x)$
F_4 $\forall x\,.\,(\mathrm{mal}(x) \wedge \mathrm{ar}(x)) \Rightarrow (\forall y\,.\,\mathrm{cr}(y) \Rightarrow \neg\mathrm{co}(x, y))$
F_5 $\exists x\,.\,\mathrm{cr}(x)$
F_6 $\neg(\exists x\,.\,\mathrm{mal}(x) \wedge \neg\mathrm{ar}(x))$

qui deviennent, sous forme prénexe,

F_1 $\forall y\,.\,\exists x\,.\,(\neg\mathrm{cr}(y) \vee \mathrm{co}(x, y))$
F_2 $\forall x, y\,.\,(\neg\mathrm{cr}(y) \vee \neg\mathrm{co}(x, y) \vee \mathrm{mal}(x))$
F_3' $\forall x\,.\,(\neg\mathrm{ar}(x) \vee \mathrm{mal}(x))$
F_4 $\forall x, y\,.\,(\neg\mathrm{mal}(x) \vee \neg\mathrm{ar}(x) \vee \neg\mathrm{cr}(y) \vee \neg\mathrm{co}(x, y))$
F_5 $\exists x\,.\,\mathrm{cr}(x)$
F_6 $\forall x\,.\,(\neg\mathrm{mal}(x) \vee \mathrm{ar}(x))$

III.2 Forme de Skolem

DÉFINITION 3.12 (FORME DE SKOLEM). *Une formule est sous forme normale de Skolem si sa forme prénexe contient uniquement des quantificateurs universels.*

La démarche qui consiste a supprimer les quantifications existentielles de la forme prénexe est appelée *skolémisation* et est définie comme suit :

— Soit $\exists x_j$ un quantificateur existentiel qui figure après les n quantificateurs universels $\forall x_{j_1} \forall x_{j_2} \ldots \forall x_{j_n}$. La quantification $\exists x_j$ est supprimée et la variable x_j est remplacé par le terme $f(x_{j_1}, x_{j_2} \ldots, x_{j_n})$ où f est un symbole fonctionnel n-aire qui n'apparaît pas ailleurs.

— Soit $\exists x_j$ un quantificateur existentiel qui n'est précédé par aucun quantificateur universel : on peut le supprimer et remplacer x_j par un symbole de constante frais (qui n'est autre qu'un symbole fonctionnel zéro-aire)

Cette transformation est autorisée par le théorème suivant :

PROPRIÉTÉ 3.2 (THÉORÈME DE SKOLEM) : Soit $\mathcal{A}$ un ensemble fini de formules du premier ordre et $\mathcal{A}_S$ l'ensemble des formes de Skolem de ces formules. Alors, $\mathcal{A}$ admet un modèle si et seulement si $\mathcal{A}_S$ admet un modèle.

On va vérifier ce théorème pour un ensemble réduit à une seule formule du type

$$A = \forall x \,.\, \exists y \,.\, p(x, y)$$

Sa forme de Skolem est

$$A_S = \forall x \,.\, p(x, f(x))$$

— Soit E un modèle de A. Ce modèle admet une relation binaire p. E étant un modèle de A, par définition, A y est satisfaite. Donc la formule $\exists y \,.\, p(x, y)$ est valide dans E. Ceci signifie que, pour chaque valeur α de x, il existe (au moins) une valeur β de y telle que $p(\alpha, \beta)$ a la valeur de vérité « *vrai* ». On peut définir une fonction f de E dans lui-même, qui associe à chaque valeur de α un des β choisis précédemment ($f(\alpha) = \beta$). On a donc un modèle pour la formule $\forall x \,.\, p(x, f(x))$, c'est à dire pour A_S.

— Réciproquement, si E est un modèle de A_S, alors, évidemment, $\exists y \,.\, p(x, y)$ est valide dans E, puisqu'il suffit d'exhiber $y = f(x)$ pour le montrer, et donc, E est un modèle de A.

Les nouveaux symboles fonctionnels ainsi définis sont appelés "symboles de Skolem", ou (par abus) "fonctions de Skolem".

Attention : Les symboles de Skolem s'ajoutent à la signature fonctionnelle du langage. On ne peut pas dire qu'une formule et sa forme skolémisée sont équivalentes, puisqu'elles n'admettent pas les mêmes modèles. On dit parfois qu'elles sont "équisatisfaisables".

EXEMPLE 3.9 (SUITE DE L'EXEMPLE 3.8). Des formes de Skolem des formules énoncées à l'exemple 3.8 sont :

F''_1 $\forall y \,.\, \neg\mathrm{cr}(y) \vee \mathrm{co}(f(y), y)$
F''_2 $\forall x, y \,.\, \neg\mathrm{cr}(y) \vee \neg\mathrm{co}(x, y) \vee \mathrm{mal}(x)$
F''_3 $\forall x \,.\, \neg\mathrm{ar}(x) \vee \mathrm{mal}(x)$
F''_4 $\forall x, y \,.\, \neg\mathrm{mal}(x) \vee \neg\mathrm{ar}(x) \vee \neg\mathrm{cr}(y) \vee \neg\mathrm{co}(x, y)$
F''_5 $\mathrm{cr}(a)$
F''_6 $\forall x \,.\, \neg\mathrm{mal}(x) \vee \mathrm{ar}(x)$

où f est un symbole fonctionnel frais et a est une constante fraîche.

IV Théorie de la démonstration en calcul des prédicats

Pour faire des démonstrations formelles en calcul des prédicats, on définit un système formel, nommé "PR", qui ajoute des axiomes et des règles d'inférence au système "LP" étudié dans le cours sur la logique des propositions. Ces nouveaux axiomes et ces nouvelles règles d'inférence permettent de traiter les quantificateurs, qui sont les nouveaux symboles introduits.

IV.1 Le système formel « *PR* »

Le système formel « *PR* » permet de démontrer qu'une formule en logique des prédicats est valide, par déductions successives. En calcul des prédicats, une formule φ est dite *valide* si toutes ses interprétations sont vraies, pour tout univers $\mathcal{U}$ et pour toutes les valeurs possibles, choisies dans $\mathcal{U}$, des variables libres de φ.

IV.1.1 Axiomes

Les deux axiomes suivants s'ajoutent aux 13 axiomes du système "LP".

— Axiome 14 : $(\forall x \,.\, \varphi) \Rightarrow \varphi(t/x)$ si aucune variable libre de t n'est liée dans φ.
— Axiome 15 : $\varphi(t/x) \Rightarrow (\exists x \,.\, \varphi)$ si aucune variable libre de t n'est liée dans φ.

Ces axiomes s'appliquent dans les conditions indiquées dans la partie II.3 : t doit être substituable à x dans φ.

Le premier axiome est le « principe de particularisation » : si, pour toute valeur de x, φ est vrai, alors $\varphi(t/x)$ est vrai, puisque t est une « valeur particulière » de x. Le second axiome est le principe d'« exhibition d'une valeur particulière » : si on peut trouver une valeur t pour laquelle $\varphi(t/x)$ est vraie alors on peut affirmer qu'il existe une valeur de x pour laquelle φ est vraie (c'est t !).

IV.1.1.1 Exemples (mathématiques)

— Si tout nombre premier n'est divisible que par 1 et par lui-même, alors 17 (qui est un nombre premier) n'est divisible que par 1 et par 17.
— Puisque 2 divise 6, il existe des nombres qui sont divisibles par 2.

L'extrême simplicité de ces exemples ne doit pas faire oublier que le terme t peut être bien autre chose qu'une constante, en fait, tout terme substituable à x dans φ.

IV.1.2 Règles d'inférence

Les deux règles d'inférence suivantes s'ajoutent à la règle de *modus ponens* du système "LP".

Ici C est une formule sans occurrences libres de x.

— $C \Rightarrow \varphi \vdash C \Rightarrow \forall x \,.\, \varphi$ [règle d'introduction de $\forall$, notée $\forall_I$]

— $\exists x \,.\, \varphi, \varphi \Rightarrow C \vdash C$ [règle d'élimination de $\exists$, notée $\exists_E$].

La première règle d'inférence est le « principe de généralisation ». Il est très souvent utilisé en mathématiques : Si l'on conduit une démonstration qui a pour conclusion un résultat dont l'expression fait intervenir une variable x, et si cette démonstration n'a fait intervenir aucune hypothèse sur cette variable, alors le résultat est vrai pour toute valeur de x. En théorie de la démonstration, cette règle d'inférence s'appelle la règle d'introduction du quantificateur universel.

La seconde règle d'inférence montre comment utiliser une hypothèse quantifiée existentiellement.

IV.2 Validité des résultats établis en calcul propositionnel

Le calcul propositionnel étant un sous-ensemble du calcul des prédicats, tous les théorèmes établis en calcul propositionnel demeurent établis en calcul des prédicats. Bref, tant que l'on n'utilise pas les symboles (quantificateurs), les nouveaux axiomes (14 et 15) et les nouvelles règles d'inférence propres au calcul des prédicats, les démonstrations du calcul des propositions s'étendent sans difficultés au calcul des prédicats.

Il ne faudrait pas en conclure hâtivement que les règles d'inférence vraies en calcul propositionnel s'appliquent aussi en calcul des prédicats. A titre d'exemple, on va étudier le cas du méta-théorème de la déduction.

PROPRIÉTÉ 3.3 (MÉTA-THÉORÈME DE LA DÉDUCTION) : En calcul des prédicats, on a :

$$\text{Si } \{H_1, H_2, \ldots, H_{n-1}\} \vdash H_n \Rightarrow C, \text{ alors } \{H_1, H_2, \ldots, H_n\} \vdash C$$

PREUVE De $\{H_1, H_2, \ldots, H_{n-1}\} \vdash H_n \Rightarrow C$ et $\{H_1, H_2, \ldots, H_{n-1}, H_n\}$ par modus ponens on déduit C. ∎

En général, la réciproque n'est pas vraie. Cependant elle reste établie dans les cas suivants :

— lorsque la déduction de C est faite « à variables constantes » (cette expression malheureuse, mais consacrée, signifie simplement que la déduction n'utilise aucune des règles d'introduction des quantificateurs, donc est faite comme en calcul propositionnel).

— lorsque les hypothèses de la déduction sont toutes des formules closes (c'est à dire sans variables libres).

IV.3 Interprétations de « *PR* »

Soit E l'interprétation de « *PR* » .

— une formule de « *PR* » est dite ***satisfaite* dans** E chaque fois qu'il existe un ensemble de valeurs, choisies dans E, pour les variables libres qui interviennent dans cette formule tel que la proposition qui en résulte est « vraie » .

— une formule de « *PR* » est dite ***valide* dans** E si, pour tout ensemble de valeurs, choisies dans E, pour les variables libres qui interviennent dans E, la proposition qui en résulte est vraie.

— une formule de « *PR* » est *universellement valide* (ou tout simplement *valide*) lorsqu'elle est valide dans toute interprétation.

IV.4 Théorème de complétude de Gödel

La démonstration est longue et repose sur un raisonnement par récurrence sur la complexité de la formule (en gros, le nombre de connecteurs ou quantificateurs qui interviennent dans celle-ci), après avoir démontré la propriété individuellement pour chaque formule de complexité 1 (un seul connecteur ou quantificateur). Toute la difficulté provient de la cardinalité (du « nombre d'éléments ») de l'interprétation. Le résultat est connu sous le nom de théorème de complétude de Gödel (1930).

PROPRIÉTÉ 3.4 (THÉORÈMES DE COMPLÉTUDE DE GÖDEL) :

— le système formel « *PR* » est complet :

$$\models B \text{ si et seulement si } \vdash B$$

— sa généralisation aussi :

$$F \models B \text{ si et seulement si } F \vdash B$$

IV.5 Satisfaisabilité et insatisfaisabilité

Soit F un ensemble de formules de « *PR* » ; F est dit *satisfaisable* s'il est possible de trouver une interprétation dans laquelle les formules de F sont simultanément satisfaites. Une telle interprétation est appelée modèle de F. S'il n'est pas possible de trouver un modèle pour l'ensemble de formules F, celui-ci est dit *insatisfaisable*, *incohérent*, *contradictoire*.

Ce sont ces notions d'insatisfaisabilité qui sont à l'origine des méthodes de démonstration dites de réfutation. Elles sont basées sur le théorème suivant.

PROPRIÉTÉ 3.5 (THÉORÈME DE CONTRADICTION) : Soit F un ensemble de formules closes et B une formule close de « *PR* ». Alors, $F \vdash B$ si et seulement si $F \cup \{\neg B\}$ est insatisfaisable

Ce théorème peut être démontré à l'aide du théorème de complétude.

Chapitre 4

Méthode de résolution

Une *tautologie* est une formule propositionnelle vraie pour toutes les valeurs des propositions atomiques qui la composent. Comment détecter automatiquement qu'une formule est une tautologie ? On peut construire sa table de vérité, mais, s'il y a n propositions atomiques différentes dans la formule, sa table de vérité compte 2^n lignes, donc son calcul peut être trop long. On cherche des méthodes plus efficaces.

On étudie ici l'une de ces méthodes, appelée "résolution", d'abord dans le cadre de la logique des propositions, puis en logique du premier ordre.

La résolution propositionnelle est une procédure de décision de la validité des formules propositionnelles, qui procède par *réfutation*. Ceci est expliqué dans la partie I. Ensuite, la méthode procède en deux étapes. La première étape, détaillée dans la partie II, réduit toute formule en une conjonction de clauses. La seconde étape, détaillée dans la partie IV, sature cet ensemble de clauses en n'appliquant qu'une seule règle de déduction, appelée "résolution", présentée dans la partie III. La partie V généralise la méthode de résolution à la logique du premier ordre.

I Réfutation

Pour prouver que la formule F est une tautologie, la méthode cherche à montrer que $\neg F$ n'est pas satisfaisable, c'est-à-dire fausse pour toute valuation [1]. On dit aussi que $\neg F$ est "contradictoire". Nous verrons que la méthode décompose $\neg F$ en un ensemble de formules plus simples, dans lequel on cherche une contradiction interne. Ce "raisonnement par contradiction" est appelé *réfutation*.

Remarque : Si la question initiale n'est pas celle de la *validité* (tautologie) de F, mais celle de la *satisfaisabilité* de F, alors cette première étape de négation est inutile. On cherche directement une contradiction dans F, et on échange les conclusions. Autrement dit, on poursuit la méthode avec F au lieu de $\neg F$.

1. Une valuation de la formule propositionnelle F est une valeur booléenne associée à chaque variable propositionnelle de F. Ainsi, une valuation correspond à une ligne de la table de vérité de F.

II Mise en forme clausale

Afin d'écrire des algorithmes plus simples de raisonnement avec des formules propositionnelles, on transforme souvent ces formules en formules équivalentes, mais plus structurées, appelées *formes normales*. Cette partie définit la forme normale conjonctive (CNF, pour *Conjunctive Normal Form*), également appelée *forme clausale*.

II.1 Définitions

Un *littéral* est soit une proposition atomique, soit la négation d'une proposition atomique. Une *clause* est une disjonction $L_1 \vee \ldots \vee L_n$ ($n \geq 0$) de littéraux. La clause vide ($n = 0$) est la constante *faux*. Elle est notée $\bot$. Une formule est en CNF si c'est une conjonction $C_1 \wedge \ldots \wedge C_k$ ($k \geq 0$) de clauses. Si $k = 0$, c'est la constante *vrai*, notée $\top$.

L'algorithme de mise en forme normale consiste à appliquer répétitivement les règles de la figure 4.1, jusqu'à ce qu'aucune règle ne soit plus applicable. C'est une technique dite de *réécriture*. Ce système de réécriture a deux bonnes propriétés : il termine toujours, et la formule qui en résulte est en CNF.

$$\begin{array}{rcl}
F \Rightarrow G & \rightarrow & (\neg F) \vee G \\
\neg(\neg F) & \rightarrow & F \\
\neg(F \wedge G) & \rightarrow & (\neg F) \vee (\neg G) \\
\neg(F \vee G) & \rightarrow & (\neg F) \wedge (\neg G) \\
F \vee (G \wedge H) & \rightarrow & (F \vee G) \wedge (F \vee H) \\
(F \wedge G) \vee H & \rightarrow & (F \vee H) \wedge (G \vee H) \\
F \vee F & \rightarrow & F
\end{array}$$

FIGURE 4.1 – Règles de mise en forme clausale.

Pour toute formule propositionnelle F, on note *cnf*(F) le résultat de cet algorithme appliqué à cette formule. L'intérêt de la décomposition d'une formule en CNF est donné par le théorème suivant, où $\vdash$ est la règle de résolution, définie dans la partie suivante.
Théorème 1 : F est non satisfaisable si et seulement si *cnf*$(F) \vdash \bot$.

III Règle de résolution propositionnelle

La notation $\vdash$ désigne ici l'application répétitive de la règle suivante, dite de résolution propositionnelle :

$$\frac{\neg P \vee C, \qquad P \vee D}{C \vee D} \tag{4.1}$$

où P est un atome (proposition atomique) et C et D sont des clauses.

Cette règle déduit une nouvelle clause de deux clauses connues. Elle déduit la clause vide si C et D sont vides. Si C et D ont des littéraux communs,

— soit on les élimine par la règle de factorisation

$$\frac{L \vee L \vee C}{L \vee C}$$

où L est un littéral et C est une clause,

— soit on considère les clauses comme des ensembles de littéraux, donc sans doublons.

Soit E un ensemble de clauses et C une clause. On note $E \vdash C$ si des applications répétitives de la règle de résolution permettent de déduire la clause C à partir de l'ensemble de clauses E.

Par exemple, $\{p_1 \vee p_2, \neg p_2 \vee \neg p_3, p_3\} \vdash p_1$ en deux étapes.

La partie suivante introduit du vocabulaire pour décrire plus précisément une étape d'application de la règle de résolution propositionnelle.

III.1 Résolvante d'une paire de clauses

Deux clauses C_1 et C_2 forment une *paire résoluble* s'il existe une et une seule proposition atomique P tel que $\neg P$ appartient à l'une des clauses et P appartient à l'autre clause. Dans ce cas, on appelle *résolvante* de C_1 et C_2 la clause, notée $res(C_1, C_2)$, obtenue en prenant la réunion des littéraux de C_1 et C_2 moins cette paire opposée.

Exercice 4.1. *Questions de compréhension.*

1. *Quelle est la différence essentielle entre cette description d'une étape de résolution et la règle (4.1) ?*
2. *Expliquer pourquoi cette différence ne change pas le résultat de la méthode de résolution.*

IV La méthode de résolution propositionnelle

Pour toute formule propositionnelle F dont on cherche à établir la validité,

1. Calculer $cnf(\neg F)$, pour mettre la négation de F sous forme clausale.
2. Appliquer répétitivement la règle de résolution au résultat de $cnf(\neg F)$.
3. Si la clause vide $\bot$ est produite, $\neg F$ est non satisfaisable, donc F est valide.
4. Sinon, si aucune déduction n'est plus possible, $\neg F$ est satisfaisable, donc F n'est pas valide.

Lorsqu'aucune déduction n'est plus possible, on dit qu'on a "saturé" l'ensemble de clause initial.

V Résolution en logique des prédicats

La méthode de résolution étendue au calcul des prédicats est plus complexe que sa version en calcul propositionnel puisqu'elle doit prendre en compte les variables quantifiées. Elle sert de base au langage Prolog.

V.1 Résolvante d'une paire de clauses

Comme précédemment, on ne considère que des formules mises sous forme clausale.

Deux clauses C_1 et C_2 forment une paire résoluble si et seulement si elles contiennent au moins une paire de littéraux $P(t_1, \ldots, t_n)$ et $\neg P(t'_1, \ldots, t'_n)$, où P est un symbole relationnel, telle qu'il existe une substitution σ telle que pour tout i, $1 \leq i \leq n$, $t_i\sigma = t'_i\sigma$.

Toute substitution telle que $t_i\sigma = t'_i\sigma$ pour tout $1 \leq i \leq n$ est appelée *unificateur* de l'ensemble de paires de termes $\{(t_1, t'_1), \ldots, (t_n, t'_n)\}$. Il s'agit de trouver l'unificateur *le plus général* : c'est celui tel que toute autre substitution se déduit de cet unificateur par composition à partir d'une autre substitution. En pratique, cela revient à chercher la substitution la moins particulière possible.

On appelle résolvante de C_1 et C_2 la clause, notée $\mathit{Res}(C_1, C_2, \sigma)$, obtenue en appliquant σ à la réunion de C_1 et de C_2 privée des littéraux opposés $P(t_1, \ldots, t_n)\sigma$ et $\neg P(t'_1, \ldots, t'_n)\sigma$.

EXEMPLE 4.2. Soit $C_1 = P(x) \vee Q(g(x))$ et $C_2 = \neg P(f(y))$ deux clauses. La paire (C_1, C_2) est résoluble en prenant comme substitution $(f(y)/x)$. Sa résolvante est $\mathit{Res}(C_1, C_2, (f(y)/x)) = Q(g(f(y)))$.

Exercice 4.3. *Peut-on unifier les deux formules atomiques*

$$P(f(X, g(Z)), X, f(Y, g(b)))$$

et

$$P(f(U, g(f(a, b))), X, U)?$$

V.2 Résolution d'un ensemble de clauses

La démarche est exactement la même que dans la partie IV, modulo le fait qu'on considère ici la règle de résolution avec unificateur.

Il faut néanmoins prendre une précaution avec les variables. En effet, la clause avec variables $P(X, Y) \vee \neg Q(Y, Z)$, par exemple, abrège la formule close $\forall X.\forall Y.\forall Z.P(X, Y) \vee \neg Q(Y, Z)$. Si une autre clause contient X, par exemple, ce n'est pas la même variable X. Pour éviter toute erreur, on renomme les variables de chaque clause, pour que des clauses différentes aient des variables différentes.

EXEMPLE 4.4. Démontrons que l'ensemble de clauses

$$\begin{array}{ll} \{ & P(a, X) \vee Q(Y, X), \\ & \neg Q(X, Z) \vee R(a, X), \\ & \neg P(a, b), \\ & \neg R(U, V) \qquad\qquad \} \end{array}$$

est contradictoire. Les variables sont en majuscules et les constantes sont en minuscules.

La variable X apparaît dans deux clauses. On commence par indexer chaque variable par le numéro de la clause où elle apparaît, pour obtenir les clauses

$$C_1 = P(a, X_1) \vee Q(Y_1, X_1),$$
$$C_2 = \neg Q(X_2, Z_2) \vee R(a, X_2),$$
$$C_3 = \neg P(a, b) \text{ et}$$
$$C_4 = \neg R(U_4, V_4).$$

Puis on calcule des résolvantes entre clauses. On numérote aussi les clauses obtenues et leurs variables.

— C_1 et C_3 sont résolubles avec la substitution (b/X_1). On a $\mathit{Res}(C_1, C_3, (b/X_1)) = Q(Y_1, b)$. On renomme les variables dans ce résultat pour obtenir $C_5 = Q(Y_5, b)$.
— C_2 et C_4 sont résolubles avec la substitution $(a/U_4, X_2/V_4)$. On a $\mathit{Res}(C_2, C_4, (a/U_4, X_2/V_4)) = \neg Q(X_2, Z_2)$. On pose $C_6 = \neg Q(X_6, Z_6)$.
— C_5 et C_6 sont résolubles avec la substitution $(Y_5/X_6, b/Z_6)$. On a $\mathit{Res}(C_5, C_6, (Y_5/X_6, b/Z_6)) = \bot$, ce qui termine la démonstration.

V.3 Mise en œuvre de la résolution

La mise en œuvre de la résolution est moins immédiate que dans le cas propositionnel, en raison de la présence de variables quantifiées :

— La mise en forme clausale est plus complexe car elle nécessite deux étapes supplémentaires (mise en forme prénexe et skolémisation).
— La résolution est aussi plus complexe car elle nécessite d'exhiber l'unificateur le plus général.
— La méthode ne termine pas toujours. On n'est certain qu'elle termine que s'il existe une contradiction dans l'ensemble de clauses initial.

EXEMPLE 4.5 (SUITE DE L'EXEMPLE 3.9). Application la résolution du premier ordre aux clauses

$F_1'' : \quad \forall y \,.\, \neg \mathrm{cr}(y) \vee \mathrm{co}(f(y), y)$
$F_2'' : \quad \forall x, y \,.\, \neg \mathrm{cr}(y) \vee \neg \mathrm{co}(x, y) \vee \mathrm{mal}(x)$
$F_3'' : \quad \forall x \,.\, \neg \mathrm{ar}(x) \vee \mathrm{mal}(x)$
$F_4'' : \quad \forall x, y \,.\, \neg \mathrm{mal}(x) \vee \neg \mathrm{ar}(x) \vee \neg \mathrm{cr}(y) \vee \neg \mathrm{co}(x, y)$
$F_5'' : \quad \mathrm{cr}(a)$
$F_6'' : \quad \forall x \,.\, \neg \mathrm{mal}(x) \vee \mathrm{ar}(x)$

de l'exemple 3.9.

Dans ces clauses, pour éviter tout risque de confusion, on renomme les variables, pour que des clauses différentes contiennent des variables différentes. On obtient les six clauses suivantes :

$C_1 : \neg \mathrm{cr}(Y_1) \vee \mathrm{co}(f(Y_1), Y_1)$
$C_2 : \neg \mathrm{cr}(Y_2) \vee \neg \mathrm{co}(X_2, Y_2) \vee \mathrm{mal}(X_2)$
$C_3 : \neg \mathrm{ar}(X_3) \vee \mathrm{mal}(X_3)$
$C_4 : \neg \mathrm{mal}(X_4) \vee \neg \mathrm{ar}(X_4) \vee \neg \mathrm{cr}(Y_4) \vee \neg \mathrm{co}(X_4, Y_4)$
$C_5 : \mathrm{cr}(a)$
$C_6 : \neg \mathrm{mal}(X_5) \vee \mathrm{ar}(X_5)$

On obtient de nouvelles clauses par les résolutions suivantes :

$C_7 = Res(C_5, C_1, (a/Y_1)) = \mathrm{co}(f(a), a)$
$C_8 = Res(C_2, C_7, (f(a)/X_2, a/Y_2)) = \neg\mathrm{cr}(a) \vee \mathrm{mal}(f(a))$
$C_9 = Res(C_5, C_8) = \mathrm{mal}(f(a))$
$C_{10} = Res(C_9, C_6, (f(a)/X_5)) = \mathrm{ar}(f(a))$
$C_{11} = Res(C_4, C_9, (f(a)/X_4, a/Y_4)) = \neg\mathrm{ar}(f(a)) \vee \neg\mathrm{cr}(a) \vee \neg\mathrm{co}(f(a), a)$
$C_{12} = Res(C_{11}, C_{10}) = \neg\mathrm{cr}(a) \vee \neg\mathrm{co}(f(a), a)$
$C_{13} = Res(C_{12}, C_7) = \neg\mathrm{cr}(a)$
$Res(C_{13}, C_5) = \bot$

ce qui termine la démonstration. Par rapport à l'énoncé de l'exercice 3.8, ce résultat s'interprète comme suit : Q est une conséquence logique de P_1, P_2, P_3, P_4 et P_5.

Remarque : Puisqu'aucune résolution n'utilise C_3, on peut aussi conclure Q sans utiliser l'hypothèse P_3 dont cette clause est issue.

VI Conclusion du chapitre

Que ce soit dans le cas propositionnel ou dans le cas du calcul des prédicats la méthode de *résolution de Robinson* est un algorithme simple de démonstration automatique.

Sa principale différence avec les systèmes déductifs se situe dans la direction de la preuve : Tandis que les systèmes déductifs effectuent une preuve en arrière (à partir d'une conclusion souhaitée), la résolution fonctionne en avant, en produisant de nouvelles conséquences jusqu'à saturation.

Pour pouvoir appliquer la méthode de résolution il est nécessaire d'exprimer la ou les formules sous forme clausale. Le mécanisme de résolution consiste alors à *résoudre* certaines clauses pour engendrer de nouvelles clauses. Ces dernières sont ajoutées à l'ensemble de clauses jusqu'à obtenir la *clause vide*. Ce mécanisme n'est pas déterministe, dans la mesure où, à une étape donnée, il peut exister plusieurs possibilités de combiner des clauses.

Chapitre 5

Les lieurs

Il est important de bien choisir les notations quand on formalise. Les problèmes qui peuvent surgir sont nombreux. Citons en seulement quelques uns.

— ambiguité syntaxique due à des surcharges de notation : le même symbole utilisé dans des contextes différents,
— ambiguité sémantique due à des rapprochements divers dans l'esprit du lecteur ou des logiciels : doubles déclarations,
— confusion des rôles : constante/variable, valeur/symbole,
— mauvaise gestion de l'espace des identificateurs (variables libres ou liées),
— mauvais repérage des regroupements (parenthésages et priorités)
— confusion dues aux abréviations et aux sous-entendus (notations incomplètes).

Nous allons répertorier quelques problèmes de ce genre dans les notations des opérateurs *lieurs* et nous en profiterons pour préciser nos notations préférées pour cet enseignement.

Ce chapitre propose de la syntaxe, comme celle que l'on apprend habituellement quand on découvre un nouveau langage de programmation !

I Les lieurs de variables

Dans le calcul des prédicats (logique du premier ordre), certaines formules contiennent des quantificateurs $\forall$ et $\exists$ qui sont des *lieurs de variables*.

Par exemple, que signifie, pour vous, l'expression suivante ?

$$(\forall x, (\exists k, x = 2 * k + 6))$$

La variable x est la variable *liée* par le quantificateur $\forall$. L'expression $(\exists k, x = 2 * k + 6)$ est l'expression *quantifiée* par le quantificateur $\forall$.

Dans l'expression $\exists k, x = 2{*}k{+}6$, la variable k est liée par le quantificateur $\exists$ et $x = 2{*}k{+}6$ est l'expression quantifiée.

Il y a plusieurs problèmes posés par l'expression ci-dessus.

I.1 Les séparateurs

Le symbole qui sépare la variable liée et l'expression quantifiée est, selon les habitudes, tantôt une virgule, un point ou deux points :

$$(\forall x \,.\, (\exists k \,.\, x = 2 * k + 6))$$

ou

$$(\forall x : (\exists k : x = 2 * k + 6))$$

Ceci n'est pas très troublant mais malgré tout, il vaut mieux toujours utiliser la même notation.

`Nous utiliserons le point.`

Notez que ce symbole se lit de manière différente selon le quantificateur : Pour le $\forall$, le point est lu *on a*, pour le $\exists$ il est lu *tel que*.

I.2 Les parenthèses

Les parenthèses ont tout à la fois un rôle de groupement et de séparateur. Par exemple, vous avez peut-être l'habitude d'écrire l'expression précédente sous les formes :

$$(\forall x)((\exists k)x = 2 * k + y))$$

ou

$$(\forall (x)(\exists (k)x = 2 * k + y)$$

ou

$$\forall x \,.\, \exists k \,.\, x = 2 * k + y$$

Cette dernière formulation n'est pas ambiguë si l'on utilise une convention. Par exemple (classique), on suppose que la portée d'un quantificateur va toujours le plus à droite possible.

Nous suivrons cette convention. Néanmoins, pour plus de clarté, nous nous efforcerons le plus souvent d'écrire toutes les paires de parenthèses précisant les portées.

I.3 Le domaine des variables liées

La notation $(\forall x \,.\, (\exists k \,.\, x = 2 * k + y))$ est pleine de sous-entendus.

Pour pouvoir *interpréter* une telle expression, il faut connaître le domaine des valeurs des variables, c'est-à-dire leur *type*.

On pourrait préciser ce type pour chaque variable liée. Par exemple, on écrirait :

$$(\forall x : x \in \mathbb{N} \,.\, (\exists k : k \in \mathbb{N} \,.\, x = 2 * k + y))$$

ou

$$(\forall x : pair(x) \wedge x \in \mathbb{N} \,.\, (\exists k : k \in \mathbb{N} \,.\, x = 2 * k + y)).$$

Quand on fait des sous-entendus, il faut être sûr(e) d'être bien compris(e). Dans une spécification formelle, mieux vaut ne pas faire de sous-entendus. Dans un programme, on doit expliciter de tels sous-entendus à l'aide de commentaires.

I.4 Variables libres et variables liées

Dans l'expression $(\forall x \,.\, (\exists k \,.\, x = 2 * k + y))$, y est une *variable libre* (non liée). Dans l'expression $(\exists k \,.\, x = 2 * k + y)$, k est une variable liée tandis que x et y sont des variables libres. Dans l'expression $x = 2 * k + y$, toutes les variables sont libres.

En général, on permet qu'un seul *lieur* lie plusieurs variables, comme dans l'expression

$$(\exists x, y, z : x \in \mathbb{N} \land y \in \mathbb{N} \land z \in \mathbb{N} \,.\, x = y^2 + z^2),$$

qui est équivalente à

$$(\exists x : x \in \mathbb{N} \,.\, (\exists y : y \in \mathbb{N} \,.\, (\exists z : z \in \mathbb{N} \,.\, x = y^2 + z^2))).$$

Proposition : Les variables liées peuvent être renommées mais pas les variables libres.

Par exemple, $(\forall x \,.\, (\exists k \,.\, x = 2 * k + y))$ et $(\forall z \,.\, (\exists x \,.\, z = 2 * x + y))$ sont deux expressions équivalentes.

I.5 Autres lieurs

Les mathématiques ont mis au point de nombreux lieurs, tels que $\sum$, $\prod$, $\bigcup$, $\bigcap$, ...

$\sum$ généralise le symbole $+$, $\prod$ généralise le symbole $*$, $\bigcup$ généralise le symbole $\cup$ d'union de deux ensembles, $\bigcap$ généralise le symbole $\cap$ d'intersection de deux ensembles.

Question : Quels symboles sont généralisés par $\forall$ et $\exists$?

Exercice : Fabriquez des expressions avec ces lieurs. Utilisez d'abord la notation que vous avez apprise, puis la nôtre.

II Le constructeur d'ensembles

La notation d'un ensemble fini avec un petit nombre d'éléments se fait habituellement par une notation dite *en extension*. Par exemple :

$$E =_{\text{def}} \{2, 8, 6, 4\}$$

Les autres ensembles sont définis par une notation dite *en compréhension*. Par exemple :

$$A =_{\text{def}} \{x \mid x \in 1..5 \land (\exists k : k \in \mathbb{N} \,.\, x = 2 * k)\}$$

Cette notation utilise le *constructeur d'ensembles* noté $\{\}$, qui lie la variable x.

Remarquez que l'expression $x \in 1..5 \land (\exists k : k \in \mathbb{N} \,.\, x = 2 * k)$ décrit un domaine (un type), qui devient l'ensemble des valeurs que peut prendre la variable x, lorsqu'elle est liée par le constructeur $\{\}$.

Le symbole $\mid$ souvent utilisé ne nous est pas utile, puisque nous avons défini une notation commune à tous les lieurs. Avec cette notation, la définition de l'ensemble A devient

$$A =_{\text{def}} \{x : x \in 1..5 \land (\exists k : k \in \mathbb{N} \,.\, x = 2 * k) \,.\, x\}$$

Plus généralement, nous définirons formellement chaque ensemble en utilisant la notation suivante :

$$\{x : D \,.\, E\},$$

dans laquelle

— x est une variable liée,
— D est le domaine des valeurs prises par x,
— E est une expression dans laquelle x apparaît (éventuellement).

Ceci revient à considérer $\{\ \}$ comme un lieur de la variable x, qui définit l'ensemble des valeurs de l'expression E lorsque x parcourt le domaine D.

Enfin, nous considérons la notation usuelle $\{x \mid D\}$ seulement comme une abréviation pratique de la notation complète $\{x : D\ .\ x\}$.

II.1 Avantages de la notation à lieur

Nous venons de voir que le premier avantage de cette notation est d'inscrire les définitions d'ensembles comme un cas particulier de notation à lieur.

Voyons à présent le principal avantage de cette notation, comparée à certains usages confus du symbole |, comme dans l'expression suivante :

$$B =_{\text{def}} \{x^2 * y \mid (\exists k : k \in \mathbb{N}\ .\ x = 2 * k)\}$$

Cette notation n'est pas claire : elle n'indique pas si x et y sont liées ou non par le constructeur d'ensemble. Cette indication est fondamentale : selon le cas, l'ensemble B prend une toute autre signification.

Exercice : Citez des éléments de B en considérant que $y \in \mathbb{N}$ est liée, puis que y est libre.

Nous n'autoriserons pas cette notation et utiliserons notre notation commune à tous les lieurs. Dans le cas où y est libre,

$$B_1 =_{\text{def}} \{x : (\exists k : k \in \mathbb{N}\ .\ x = 2 * k)\ .\ x^2 * y\}$$

et

$$B_2 =_{\text{def}} \{x, y : (\exists k : k \in \mathbb{N}\ .\ x = 2 * k)\ .\ x^2 * y\}$$

dans le cas où y est liée.

Exercice : Notez que les notations diverses de ce paragraphe sous entendent une partie des domaines. Rétablissez les notations complètes au sujet des domaines.

Exercice : L'expression suivante obéit-elle à nos conventions ?

$$B_3 =_{\text{def}} \{(x, y) : (\exists k : k \in \mathbb{N}\ .\ x = 2 * k)\}$$

Qu'est-elle supposée définir ? Écrire l'expression équivalente qui correspond à nos conventions.

Exercice : Les deux expressions suivantes définissent-elles le même ensemble ?

$$\{x : x \in \mathbb{N}\ .\ x^2\}$$

$$\{z \mid z \in \mathbb{N} \wedge z = x^2\}$$

Une dernière question : Par quel genre de valeurs doit-on interpréter D et E dans les constructions d'ensemble

$$\{x : D\ .\ E\}$$

et

$$\{x \mid D\}\ ?$$

III Dépliement des formules liées à un domaine fini

On a, par exemple :

$$
\begin{array}{rcl}
(\forall x : x \in \{1,5,7\} \,.\, P(x)) & = & P(1) \wedge P(5) \wedge P(7) \\
(\exists x : x \in \{1,5,7\} \,.\, P(x)) & = & P(1) \vee P(5) \vee P(7) \\
(\Sigma x : x \in \{1,5,7\} \,.\, f(x)) & = & f(1) + f(5) + f(7) \\
(\prod x : x \in \{1,5,7\} \,.\, f(x)) & = & f(1) * f(5) * f(7) \\
(\bigcup i : i \in \{1,5,7\} \,.\, A_i) & = & A_1 \cup A_5 \cup A_7 \\
\{x : x \in \{1,5,7\} \,.\, f(x)\} & = & \{f(1), f(5), f(7)\}
\end{array}
$$

Chapitre 6

Preuve de propriété par récurrence

L'ensemble $\mathbb{N}$ des nombres naturels est infini. Prouver des propriétés sur un tel ensemble infini exige une technique de preuve qui est fondamentale en informatique : la preuve par récurrence.

I Principe de récurrence

Considérons une expression $P(n)$ où P est de profil $\mathbb{N} \to \{vrai, faux\}$. Par exemple

$$P(n) = (\Sigma i \,:\, 1 \leq i \leq n \,.\, 2 * i - 1) = n^2$$

$P(2)$ affirme que $1 + 3 = 2^2$ et $P(3)$ affirme que $1 + 3 + 5 = 3^2$.
Pour prouver que

$$(\forall n \,:\, n \in \mathbb{N} \,.\, P(n))$$

on propose de procéder de la manière suivante :

— Prouver que l'on a $P(0)$.
— Prouver que, pour un $n \geq 1$ quelconque, si l'on suppose que $P(0), \cdots, P(n-1)$ sont vrais, alors $P(n)$ est vrai.

On admet que ceci permet de conclure que $P(n)$ est vrai pour tout $n \in \mathbb{N}$. Intuitivement on peut prouver $P(1)$ puisque $P(0)$, puis $P(2)$ puisque $P(0)$ et $P(1)$, et ainsi de suite pour tout n. Cette manière de procéder est formalisée dans le principe suivant.

Principe de récurrence

$$(\forall n \,:\, n \in \mathbb{N} \,.\, (\forall i \,:\, 0 \leq i < n \,.\, P(i)) \Rightarrow P(n)) \Rightarrow (\forall n \,:\, n \in \mathbb{N} \,.\, P(n))$$

Le cas $P(0)$ peut-être mis en évidence séparément, dans l'énoncé équivalent suivant :

$$P(0) \wedge (\forall n \,:\, n \in \mathbb{N} \,.\, (\forall i \,:\, 0 \leq i \leq n \,.\, P(i)) \Rightarrow P(n+1)) \Rightarrow (\forall n \,:\, n \in \mathbb{N} \,.\, P(n))$$

I.1 Définitions

La preuve de $P(0)$ s'appelle *le cas de base* de la récurrence.

La preuve de $(\forall n \,:\, n \in \mathbb{N} \,.\, (\forall i \,:\, 0 \leq i \leq n \,.\, P(i)) \Rightarrow P(n+1))$ s'appelle *le pas de la récurrence*.

La partie $(\forall i \,:\, 0 \leq i \leq n \,.\, P(i))$ de ce pas de récurrence est appelée *hypothèse de récurrence*.

I.2 Comment présenter une preuve par récurrence ?

Quand on prouve $(\forall n \,:\, n \in \mathbb{N} \,.\, P(n))$ par récurrence, on prouve le cas de base et le pas de la récurrence séparément.

La preuve du pas de de récurrence est faite en prouvant $P(n+1)$ pour un $n \geq 0$ arbitraire et en supposant vraie l'hypothèse $(\forall i \,:\, 0 \leq i \leq n \,.\, P(i))$.

Par exemple, prouvons

$$P(n) = (\Sigma i \,:\, 1 \leq i \leq n \,.\, 2 * i - 1) = n^2$$

pour tout entier naturel n.

Preuve. La preuve est par récurrence sur n.

— **Cas de base** $P(0)$:

$$(\Sigma i \,:\, 1 \leq i \leq 0 \,.\, 2 * i - 1) = 0$$

Puisque le domaine de la somme est vide.

$$= 0^2$$

— **Pas de récurrence** :

Soit un $n \geq 0$ quelconque, on prouve $P(n+1)$ en utilisant l'hypothèse de récurrence $P(0) \wedge \cdots \wedge P(n)$.

Pour prouver $P(n+1)$, on transforme sa partie gauche

$$(\Sigma i \,:\, 1 \leq i \leq n+1 \,.\, 2 * i - 1)$$

en sa partie droite $(n+1)^2$.

$$\begin{array}{ll} & (\Sigma i \,:\, 1 \leq i \leq n+1 \,.\, 2 * i - 1) \\ = & \quad < \text{ en séparant le dernier terme } > \\ & (\Sigma i \,:\, 1 \leq i \leq n \,.\, 2 * i - 1) + (2 * (n+1) - 1) \\ = & \quad < \text{ par l'hypothèse de récurrence } > \\ & n^2 + (2 * (n+1) - 1) \\ = & \quad < \text{ par calcul arithmétique } > \\ & (n+1)^2 \end{array}$$

□

Dans la preuve ci-dessus, on a utilisé uniquement $P(n)$ comme hypothèse de récurrence. Dans ce cas on dit que la preuve est par *récurrence faible*. Sinon, on emploie le terme *récurrence forte*.

Une technique assez générale pour la preuve par *récurrence faible* est de manipuler $P(n+1)$ par logique équationnelle de façon à mettre en évidence $P(n)$, ceci afin de pouvoir utiliser l'hypothèse de récurrence.

II Déplacement du cas de base

La récurrence peut se faire pour un sous-ensemble infini quelconque $A = \{n \,:\, n \in \mathbb{N} \,.\, n \geq n_0\}$ de $\mathbb{N}$. Mais alors, le point de départ est n_0 et donc le cas de base est la preuve de la satisfaction de $P(n_0)$.

Principe de récurrence sur $A = \{n \,:\, n \in \mathbb{N} \,.\, n \geq n_0\}$

$$P(n_0) \wedge (\forall n \,:\, n \in A \,.\, (\forall i \,:\, n_0 \leq i \leq n \,.\, P(i)) \Rightarrow P(n+1)) \Rightarrow (\forall n \,:\, n \in A \,.\, P(n))$$

II.1 Exemple

Prouvons que $2 * n + 1 < 2^n$, pour $n \geq 3$.

On prouve que $(\forall n \,:\, 3 \leq n \,.\, P(n))$**, avec** $P(n) = (2 * n + 1 < 2^n)$**.**

Preuve. La preuve est par récurrence sur n.

— **Cas de base** $P(3)$:

$$P(3) = (2 * 3 + 1 < 2^3) = (7 < 8) = true$$

— **Pas de récurrence** :
Pour $n \geq 3$ quelconque, on prouve $P(n+1)$ en utilisant l'hypothèse de récurrence, $P(3) \wedge \cdots \wedge P(n)$.

$$P(n+1) = (2 * (n+1) + 1 < 2^{n+1})$$

$$\begin{array}{lll} & 2^{n+1} & \\ = & & \\ & 2 * 2^n & \\ > & & < \text{ par l'hypothèse de récurrence } > \\ & 2 * (2 * n + 1) & \\ = & & \\ & 4 * n + 2 & \\ = & & \\ & (2 * (n+1) + 1) + 2 * n - 1 & \\ > & & < \text{ puisque } 2 * n - 1 > 0 \text{ pour } n \geq 3 \ > \\ & 2 * (n+1) + 1 & \end{array}$$

□

II.2 Un autre exemple

Supposons une monnaie qui est composée uniquement de pièces de 2 centimes et de 5 centimes. Montrer alors que tout total au dessus de 3 centimes peut être formé avec ces pièces.

Preuve. L'énoncé de $P(n)$ est le suivant : "Il existe un multi-ensemble composé de pièces de 2 centimes et de 5 centimes ayant pour somme n.". Il s'agit alors de prouver que $(\forall n \,:\, 4 \leq n \,.\, P(n))$.

— **Cas de base** :
Pour obtenir 4 on prend 2 pièces de 2 centimes.
— **Pas de récurrence** :
L'hypothèse de récurrence $P(n)$ assure qu'il existe un multi-ensemble dont la somme est n.
Il y a deux cas :

1. Le multi-ensemble contient au moins une pièces de 5 centimes. Alors, on peut remplacer une des pièces de 5 centimes par 3 pièces de 2 centimes pour obtenir un multi-ensemble dont la somme est $n + 1$.
2. Le multi-ensemble contient seulement des pièces de 2 centimes. Dans ce cas, il y a au moins deux pièces de 2 centimes puisque $4 \leq n$. On remplace alors ces deux pièces de 2 centimes par une de 5 centimes et le tour est joué.

□

Pour formaliser la preuve, il nous faudrait avoir connaissance de l'axiomatique des multi-ensembles.

II.3 Une preuve par récurrence forte

Un nombre premier est un entier naturel supérieur ou égal à 2 qui n'a pas d'autres diviseurs que 1 et lui-même. Prouvons que tout $n \geq 2$ est un produit de nombres premiers. Pour cela on suppose avoir un prédicat $prime? : \mathbb{N} \to \{vrai, faux\}$ qui vaut vrai si n est premier.

L'énoncé de $P(n)$ est "n est un produit de nombres premiers".

Preuve. Le cas de base est $P(2)$, qui est vrai car 2 est premier. Ensuite, pour un $n > 2$ quelconque, on prouve $P(n)$ en utilisant les hypothèses de récurrence $P(2), \ldots, P(n-1)$.

Il y a deux cas :

1. $prime?(n)$ est vrai. Alors $n = n$ est la décomposition de n comme produit de nombres premiers, donc $P(n)$ est satisfait.
2. $prime?(n)$ est faux. Alors il existe p et q supérieurs ou égaux à 2 tels que $n = p * q$. Il en résulte que $p < n$ et $q < n$, ce qui permet d'utiliser les hypothèses de récurrence $P(p)$ et $P(q)$. Par hypothèse de récurrence, on sait que "p est un produit de nombres premiers" et que "q est un produit de nombres premiers". Donc $n = p * q$ est aussi un produit de nombres premiers.

□

Bien évidemment, cette preuve peut s'exprimer formellement.

III Spécifications récursives

Il y a souvent deux manières de spécifier des fonctions :
— soit directement par une expression dans un calcul (des propositions, des prédicats ou arithmétique). Par exemple

$$x^n = (\Pi i \,:\, 1 \leq i \leq n \,.\, x)$$

— soit par une définition récursive. Par exemple

$$x^0 = 1$$
$$x^{n+1} = x * x^n \textit{ pour } n \geq 0$$

Ici aussi, nous avons un cas de base et au moins un cas récursif.

Remarquons que l'on peut aussi écrire de manière équivalente :

$$x^0 = 1$$
$$x^n = x * x^{n-1} \textit{ pour } n \geq 1$$

Une définition récursive induit des preuves par récurrence.

III.1 Exemple

Preuve de $x^{m+n} = x^m * x^n$ pour des entiers naturels m et n quelconques.
Preuve. On veut prouver

$$P = (\forall m, n \,:\, m \in \mathbb{N} \wedge n \in \mathbb{N} \,.\, x^{m+n} = x^m * x^n).$$

Or, on peut écrire

$$P = (\forall n \,:\, n \in \mathbb{N} \,.\, (\forall m \,:\, m \in \mathbb{N} \,.\, x^{m+n} = x^m * x^n))$$

et utiliser une preuve par récurrence pour prouver que $(\forall n \,:\, n \in \mathbb{N} \,.\, P(n))$, avec

$$P(n) = (\forall m \,:\, m \in \mathbb{N} \,.\, x^{m+n} = x^m * x^n)$$

— **Cas de base**. On prouve que $P(0)$ est satisfait. Pour un m quelconque, on a :

$$\begin{array}{ll} & x^{m+0} = x^m * x^0 \\ = & x^m = x^m * 1 \\ = & x^m = x^m \\ = & \textit{true} \end{array}$$

— **Pas de récurrence**. On prouve $P(n+1)$, à savoir

$$(\forall m \,:\, m \in \mathbb{N} \,.\, x^{m+(n+1)} = x^m * x^{n+1})$$

en utilisant l'hypothèse de récurrence :

$$(\forall m \,:\, m \in \mathbb{N} \,.\, x^{m+n} = x^m * x^n)$$

Pour un m arbitraire, on a :

$$\begin{array}{ll} & x^{m+(n+1)} \\ = & x^{(m+1)+n} \\ = & \quad < \text{ par hypothèse de récurrence avec la substitution } [m := m+1] > \\ & x^{m+1} * x^n \\ = & \quad < \text{ par définition } > \\ & (x * x^m) * x^n \\ = & \quad < \text{ par calcul arithmétique } > \\ & x^m * (x * x^n) \\ = & \quad < \text{ par définition } > \\ & x^m * x^{n+1} \end{array}$$

Notons que la substitution dans l'hypothèse de récurrence est valide car le raisonnement est fait pour un m quelconque.

□

IV Exemple amusant

Pour terminer ce chapitre, exposons un exemple amusant et informel de raisonnement par récurrence.

Les *n* enfants qui jouent dehors sont supposés ne pas se salir. Cependant, au cours de leurs jeux, *k* d'entre eux se mettent de la boue sur le front. Chacun des enfants peut voir la boue qui marque le front des autres mais pas la sienne. Le père arrive et dit : "Je vois qu'au moins l'un d'entre vous a mis de la boue sur son front", ce qui exprime bien ce que chacun des enfants sait dès que $k > 1$. Le père va alors répéter la question suivante : "Est-ce que vous savez si vous avez de la boue sur votre propre front ?", question à laquelle les enfants, supposés francs et intelligents, vont répondre tous ensemble simultanément.

On peut prouver, par récurrence sur k, que les k enfants qui ont un front boueux vont répondre "non" les $k-1$ premières fois que la question est posée et "oui" la $k^{ième}$ fois.

Preuve.

— **Cas de base** : $k := 1$
Il est évident que l'unique enfant qui a de la boue sur le front répondra "oui" dès la première question, puisqu'il voit qu'aucun de ses camarades n'en a.

— **Pas de récurrence** : de k à $k+1$
A la $(k+1)^{ième}$ question, par hypothèse de récurrence, tout enfant qui a un front boueux peut raisonner ainsi : "Si je n'avais pas de boue sur mon front, alors les k enfants ayant un front boueux auraient répondu "oui" au $k^{ième}$ tour. Puisqu'ils ne l'ont pas fait, c'est que j'ai de la boue sur mon front."

□

Un peu de réflexion sur ce problème

Appelons *p* l'affirmation de départ du père. Dès que $k > 1$, tout enfant connaît p et voit aussi au moins un enfant ayant un front marqué de boue. Donc tous les enfants connaissent p au départ. Il semble que communiquer p aux enfants n'ajoute rien à ce que connaît chaque enfant. Mais, aussi étrange que cela soit à première vue, la communication publique de p par le père est absolument nécessaire.

Voici la réponse : Avant que le père ne communique publiquement p, tous les enfants ne savent pas que tous les enfants connaissent p. Mais, après cette communication, chacun des enfants sait que tous les enfants connaissent p et que tous les enfants savent que tous les autres enfants connaissent p, et ainsi de suite.

Exercice : Montrer par récurrence que si le père ne communique pas publiquement p, les enfants répondront toujours "non" à la question posée.

Exercice : Développer l'argumentation dans les cas particuliers $k = 2$, $k = 3$ et $k = 4$ pour voir exactement où intervient le fait que chacun des enfants sait que tous les enfants connaissent p.

Chapitre 7

Types inductifs, calculs et preuves

Un problème fondamental en informatique est de stocker et de manipuler des collections de données de même nature. Une *structure de données* est une manière de représenter de telles collections dans la mémoire d'un ordinateur, pour rendre leur manipulation possible et efficace. Etudier ces structures, notamment celles à base de pointeurs, fait partie d'une branche de l'informatique appelée "algorithmique". Raisonner sur des structures à base de pointeurs est difficile, en particulier parce que plusieurs pointeurs peuvent faire référence à la même partie de la mémoire. Mais on peut raisonner sur les collections indépendamment de la manière dont elles sont représentées, grâce aux *types de données abstraits* (*abstract datatypes* en anglais).

Vous connaissez déjà les types déclarés dans les langages de programmation. Nous parlerons ici du type *Liste*, du type *Arbre*, etc. Un type rassemble des éléments (valeurs, data, objets) qui sont construits de la même manière. Un *type* pourra donc être considéré comme un ensemble de valeurs (ce serait inexact en toute généralité, mais c'est correct dans le cadre limité de ce cours).

Ce chapitre définit certains types de données abstraits, dits "inductifs". Nous allons développer une méthode formelle pour spécifier des calculs et raisonner sur des éléments de ces types. Ce mode de raisonnement, qui généralise le raisonnement par récurrence, sera présenté dans le chapitre 10.

I Définir un type inductif

I.1 Des constructeurs

Nous avons besoin de symboles qui permettent le regroupement des informations dans la structure. On les appelle les *constructeurs* des éléments du type.

I.1.1 Exemple des listes

On introduit ici le type de données abstrait le plus simple et le plus fréquemment utilisé pour décrire des collections de données, celui des *listes (homogènes)*. Une liste homogène est une collection ordonnée (une séquence) finie de données de même nature.

Pour définir formellement les listes homogènes, on utilise une notion de type qui sera généralisée dans le chapitre 12. Pour l'instant, il suffit de comprendre que toute donnée a un type.

Par exemple, on écrit $i : \mathbb{N}$ et $x : \mathbb{R}$ pour signifier respectivement que la donnée i est un entier naturel et que la donnée x est un nombre réel. Le type des listes homogènes dont tous les éléments sont des entiers naturels est noté *Liste*$(\mathbb{N})$. Le type des listes homogènes dont tous les éléments sont des nombre réels est noté *Liste*$(\mathbb{R})$. Plus généralement, quel que soit le type α, le type des listes homogènes dont tous les éléments sont de type α est noté *Liste*(α). Ce type est dit "générique" ou "polymorphe" car on peut remplacer α par $\mathbb{N}$, $\mathbb{R}$, *Liste*$(\mathbb{N})$, *Liste*(*Liste*$(\mathbb{R})$), ...

Pour définir le type *Liste*(α), on a besoin de deux constructeurs :

— *Cons* qui réclame deux arguments : un élément a de type α et une liste l de type *Liste*(α). *Cons*(a, l) ajoute l'élément a en tête (au début) de la liste l.

— *Nil* qui désigne la liste vide (sans élément) et permet d'initialiser la construction.

Ainsi l'expression *Cons*$(2,$ *Cons*$(4,$ *Cons*$(1,$ *Nil*$)))$ est la liste des trois entiers $(2, 4, 1)$ dans cet ordre.

I.1.2 Exemple des arbres binaires

Dans un type spécifiant des arbres binaires, on a besoin d'un constructeur que nous noterons avec un point (.) qui permet de regrouper deux arbres binaires en un arbre binaire plus gros.

Ainsi l'expression $.(3, .(.(4, 5), .(7, 8)))$ spécifie un arbre binaire à feuilles de type entier naturel.

Les éléments d'un type sont donc constitués à partir de constructeurs et de types (ensembles de base), comme par exemple les entiers naturels $\mathbb{N}$.

Nous allons considérer des constructions *homogènes*. Par exemple des listes d'entiers, ou des listes d'arbres binaires ou des listes de listes d'entiers, ou des arbres de listes d'entiers.
Exercice : Donner un exemple d'élément appartenant à chacun de ces types.

Il est plus clair de préciser dans le nom du type, par un ou plusieurs paramètres, quelle est la nature de ces *briques de base*. Nous écrirons donc *Liste*$(\mathbb{N})$, *Arbre*$(\mathbb{N})$ ou encore *Liste*(*Liste*$(\mathbb{N})$).
Question : Est-ce que cela a un sens de parler d'un type *Arbre*(*Arbre*$(\mathbb{N})$) ?

Nous pouvons donc parler très généralement d'un type *Arbre*(α) ou d'un type *Liste*(α), où α est une variable qui est substituable par une expression de type.

I.2 Règles d'inférence

Les éléments d'un type, ou plutôt les expressions dénotant syntaxiquement les éléments de ce type, seront déduites par des règles d'inférence.

On rappelle qu'une règle d'inférence permet de déduire des théorèmes (des faits) à partir d'autres théorèmes et axiomes (d'autres faits). Une règle d'inférence prend la forme

$$\frac{P_1, \cdots, P_n}{C}$$

d'une fraction, avec des expressions prémisses au dessus de la barre de fraction et une expression conclusion en dessous de la barre. Une règle d'inférence permet de raisonner. On l'instancie en donnant des valeurs aux variables présentes dans ses prémisses et sa conclusion. On peut alors utiliser cette instance pour déduire le fait C instancié des prémisses $P_1, \cdots, P_n$ instanciées de la même manière.

Dans le cas de la définition d'un type, les prémisses et la conclusion seront de la forme $E : t$, où E est une expression à base de constructeurs et t est un type.

I.2.1 Exemple des listes

On peut affirmer que la liste vide d'expression *Nil* appartient au type *Liste*(α), soit en déclarant l'axiome

$$\textit{Nil} : \textit{Liste}(\alpha)$$

soit en donnant une règle d'inférence

$$\frac{}{\textit{Nil} : \textit{Liste}(\alpha)}$$

qui déclare que l'on peut déduire que *Nil* est une liste à partir d'aucune prémisse.

Par la règle

$$\frac{a : \alpha, l : \textit{Liste}(\alpha)}{\textit{Cons}(a, l) : \textit{Liste}(\alpha)}$$

on déclare qu'on peut fabriquer une expression de type *Liste*(α) à partir d'une expression a de type α et d'une expression l de liste, en mettant devant l l'élément a, par le constructeur *Cons*.

Notez que a est une variable substituable par des expressions de type α et que l est substituable par des expressions de type *Liste*(α).

Exercice : Utiliser ces deux règles d'inférence pour prouver que l'expression *Cons*$(2, \textit{Cons}(4, \textit{Cons}(1, \textit{Nil})))$ est de type *Liste*$(\mathbb{N})$.

I.2.2 Exemple des arbres binaires

On définit l'ensemble des expressions dénotant des arbres binaires à feuilles de type α, c'est-à-dire le type *Arbre*(α), par les deux règles suivantes :

$$\frac{a : \alpha}{a : \textit{Arbre}(\alpha)}$$

$$\frac{t_1 : \textit{Arbre}(\alpha), t_2 : \textit{Arbre}(\alpha)}{.(t_1, t_2) : \textit{Arbre}(\alpha)}$$

Exercice : Commenter la signification des deux règles d'inférence et montrer que l'expression $.(3, .(.(4, 5), .(7, 8)))$ spécifie un arbre binaire de type *Arbre*$(\mathbb{N})$.

Plus précisément, les règles d'inférence particulières à la construction des expressions des éléments d'un type *inductif* sont de la forme :

$$\frac{e_1 : T_1, \cdots, e_n : T_n}{e : T}$$

où e est une expression du type T que l'on déclare et les e_i sont des expressions respectives des types T_i. De plus :

- L'ensemble des règles d'inférence doit être fini.
- Le nombre des prémisses de chaque règle doit être fini.
- L'expression conclusion e est formée des e_i en prémisses arguments d'un *constructeur* du type T.

II Calculer sur des types inductifs

On veut pouvoir extraire une information de l'ensemble des objets (data) éléments d'un type inductif. On peut aussi voir cela comme un codage des objets du type où comme une interprétation dans un certain modèle de ces mêmes objets (i.e. on leur attache un certain sens dans un modèle).

II.1 Exemple sur les arbres binaires : calcul du nombre des feuilles

Il s'agit de définir une fonction *nb-feuilles* de profil (autrement dit "de type") $\mathit{Arbre}(\alpha) \to \mathbb{N}$.

Le calcul s'appuie sur la construction inductive des expressions des objets du type. On se pose la question *QUOI* pour le cas de la règle :

$$\frac{a : \alpha}{a : \mathit{Arbre}(\alpha)}$$

i.e. pour $a : \mathit{Arbre}(\alpha)$ venant de la prémisse $a : \alpha$

$$\mathit{nb\text{-}feuilles}(a) = QUOI?$$

Réponse :

$$\mathit{nb\text{-}feuilles}(a) = 1.$$

Notez que la réponse est bien une expression de type $\mathbb{N}$.

Ensuite, on se pose la question *QUOI* pour la règle :

$$\frac{t_1 : \mathit{Arbre}(\alpha), t_2 : \mathit{Arbre}(\alpha)}{.(t_1, t_2) : \mathit{Arbre}(\alpha)}$$

i.e. pour $.(t_1, t_2) : \mathit{Arbre}(\alpha)$ venant des deux prémisses $t_1 : \mathit{Arbre}(\alpha)$ et $t_2 : \mathit{Arbre}(\alpha)$

$$\mathit{nb\text{-}feuilles}(.(t_1, t_2)) = QUOI?$$

Réponse :

$$\mathit{nb\text{-}feuilles}(.(t_1, t_2)) = \mathit{nb\text{-}feuilles}(t_1) + \mathit{nb\text{-}feuilles}(t_2).$$

Notez à droite du signe $=$ la réutilisation de la fonction, pour des arguments de type $\mathit{Arbre}(\alpha)$. Le type des arguments donne une information sur la place possible de ces appels.

On dit d'un calcul défini à partir de lui-même qu'il est *inductif*. La structure des règles d'inférence qui définissent les types inductifs garantit la terminaison de ces calculs.

II.2 Qu'est-ce qu'un calcul sur un type inductif ?

On peut dire (*rapidement*) qu'un calcul sur un TYPE inductif T est une fonction Φ de profil $T \to T'$ où T' est un type quelconque (de base ou inductif), ce qu'on écrit $\Phi : T \to T'$.

Cette fonction de calcul Φ est définie sur chaque règle d'inférence $e_1 : T, \ldots, e_n : T \vdash e : T$ de T par une équation de la forme

$$\Phi(e) = \phi(\Phi_1(e_1), \cdots, \Phi_n(e_n))$$

où ϕ est une fonction (constructeur ou fonction de calcul) et où les Φ_i ($1 \leq i \leq n$) peuvent être l'identité ou la fonction Φ appliquée aux expressions e_i des prémisses $e_i : T$.

Remarque : La définition d'un calcul peut être plus complexe. En particulier, Φ peut être une expression de calcul complexe, mais cette notion de calcul est suffisante dans beaucoup de cas qui nous intéressent.

II.2.1 Exemples de calcul sur les listes

— Calcul du nombre d'éléments d'une liste :
Fonction *length : Liste*$(\alpha) \to \mathbb{N}$

$$\begin{aligned} &\textit{length}(\textit{Nil}) = 0 \\ &\textit{length}(\textit{Cons}(a, l)) = 1 + \textit{length}(l) \end{aligned}$$

— Juxtaposition de deux listes :
Fonction *append : Liste*$(\alpha) \times$ *Liste*$(\alpha) \to$ *Liste*(α).
On peut choisir de guider le calcul uniquement sur la construction inductive de la liste gauche.

$$\begin{aligned} &\textit{append}(\textit{Nil}, y) = y \\ &\textit{append}((\textit{Cons}(a, l), y) = \textit{Cons}(a, \textit{append}(l, y)) \end{aligned}$$

II.3 Composition des calculs

Il est très intéressant d'avoir des calculs car ce sont des briques de base qu'on peut ensuite composer avec d'autres calculs ou des constructeurs. Par exemple, on aurait pu spécifier la fonction *nb-feuilles* comme étant le résultat d'une composition :

$$\textit{nb-feuilles} =_{\mathsf{def}} \textit{length} \circ \textit{liste-des-feuilles}$$

ce qu'on peut écrire

$$\textit{nb-feuilles} =_{\mathsf{def}} \textit{liste-des-feuilles}; \textit{length}$$

ou encore

$$\textit{nb-feuilles}(t) =_{\mathsf{def}} \textit{length}(\textit{liste-des-feuilles}(t))$$

Exercice : Décrire la fonction *liste-des-feuilles*.
Réponse :

$$\begin{aligned} \textit{liste-des-feuilles}(a) &= \textit{Cons}(a, \textit{Nil}) \qquad \text{si } a : \alpha \\ \textit{liste-des-feuilles}(.(t_1, t_2)) &= \textit{append}(\textit{liste-des-feuilles}(t_1), \textit{liste-des-feuilles}(t_2)) \end{aligned}$$

Évidemment il faut que la composition soit possible, ce qui peut se voir par le profil des fonctions que l'on compose.

Dans l'exemple, la fonction *length* de profil *Liste*$(\alpha) \to \mathbb{N}$ se compose avec la fonction *liste-des-feuilles* de profil *Arbre*$(\alpha) \to$ *Liste*(α) pour donner une fonction de profil *Arbre*$(\alpha) \to \mathbb{N}$.

III Raisonner sur des types inductifs

L'intérêt d'avoir une spécification formelle est de pouvoir raisonner à partir de cette spécification et d'en prouver certaines propriétés importantes.

Une propriété inductive est une propriété qui est vraie pour tous les objets d'un type. Une propriété peut s'exprimer comme un prédicat (fonction à valeur dans $\{vrai, faux\}$) sur les calculs.

Par exemple, on veut pouvoir prouver que la composition ci-dessus est correcte, i.e. que l'on a bien :

$$Prop(t) =_{\text{def}} \textit{nb-feuilles}(t) = \textit{length}(\textit{liste-des-feuilles}(t))$$

La preuve est inductive sur la structure de l'arbre t.

On doit vérifier la propriété sur chaque règle d'inférence construisant t. Pour une règle d'inférence à prémisses, on peut supposer que la propriété est vraie sur les objets du type qui sont les briques de la construction. Cela marche selon un principe d'induction analogue à celui du raisonnement par récurrence sur les entiers naturels. i.e., si c'est vrai pour les briques, alors prouvons que cela reste vrai pour l'édifice.

Plus précisément, dans le cas des arbres binaires, on peut dire que si l'on prend pour hypothèses que la propriété P est vraie pour les composants t_1 et t_2 , i.e. les deux hypothèses d'induction $P(t_1)$ et $P(t_2)$, et que l'on prouve qu'elle est vraie pour l'arbre $.(t_1, t_2)$, alors on sait (principe d'induction) que P est vraie pour tous les arbres construits par cette règle d'inférence.

Une telle preuve peut se rédiger comme suit :

— Premier cas (cas de base) : Soit t une feuille $a : \alpha$.

$$Prop(a) =_{\text{def}} \textit{nb-feuilles}(a) = \textit{length}(\textit{liste-des-feuilles}(a))$$

Il suffit ici de déplier les définitions des fonctions par simple remplacement d'un égal par un égal :

$$Prop(a) =_{\text{def}} 1 = \textit{length}(\textit{Cons}(a, \textit{Nil}))$$

et encore une fois :

$$Prop(a) =_{\text{def}} 1 = 1$$

ce qui se réduit en

$$Prop(a) =_{\text{def}} \textit{vrai}$$

donc la propriété est vraie pour tous les arbres réduit à une feuille.

— Deuxième cas (deuxième règle d'inférence). Soit t obtenu par la règle de conclusion $.(t_1, t_2)$. On doit prouver

$$Prop(.(t_1, t_2)) =_{\text{def}} \textit{nb-feuilles}(.(t_1, t_2)) = \textit{length}(\textit{liste-des-feuilles}(.(t_1, t_2)))$$

sous les deux hypothèses d'induction venant des prémisses de la règle d'inférence :

$$Prop(t_1) =_{\text{def}} \textit{nb-feuilles}(t_1) = \textit{length}(\textit{liste-des-feuilles}(t_1))$$

et

$$Prop(t_2) =_{\text{def}} \textit{nb-feuilles}(t_2) = \textit{length}(\textit{liste-des-feuilles}(t_2))$$

On déplie les calculs :

$$Prop(.(t_1, t_2)) =_{\text{def}}$$
$$\textit{nb-feuilles}(t_1) + \textit{nb-feuilles}(t_2) = \textit{length}(\textit{append}(\textit{liste-des-feuilles}(t_1), \textit{liste-des-feuilles}(t_2)))$$

Ici, on voudrait pouvoir appliquer les hypothèses d'induction pour conclure, mais on ne le peut qu'en utilisant un lemme (qu'il faudrait aussi prouver). Ce lemme doit nous permettre de composer *length* avec *append*. Il faut l'inventer :

$$\textit{length}(\textit{append}(l_1, l_2)) = \textit{length}(l_1) + \textit{length}(l_2)$$

ce qui nous permet d'obtenir

$$Prop(.(t_1, t_2)) =_{\text{def}}$$
$$\textit{nb-feuilles}(t_1) + \textit{nb-feuilles}(t_2) = \textit{length}(\textit{liste-des-feuilles}(t_1)) + \textit{length}(\textit{liste-des-feuilles}(t_2))$$

et de conclure par les hypothèses d'induction.

Le principe d'induction appliqué dans cette démonstration sera présenté de manière générale dans le chapitre 10.

Chapitre 8

Calculs sur les listes

Ce chapitre approfondit la spécification de fonctions qui ont pour arguments une, deux ou plusieurs listes, de manière récursive, par des égalités qui distinguent les deux cas de définition de la liste.

Rappelons que

$$\mathit{Nil} : \mathit{Liste}(\alpha)$$

est le constructeur de la liste vide et que

$$\mathit{Cons} : \alpha \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha)$$

est l'autre constructeur, qui ajoute un élément devant ceux d'une autre liste. Rappelons aussi qu'il n'y a qu'une seule façon de construire une liste avec ces deux constructeurs.

Nous allons donner des exemples variés de définitions de calculs sur les listes qui s'appuient sur les deux cas de construction d'une liste.

I Les calculs simples sur une liste

I.1 Somme des éléments d'une liste d'entiers

On peut spécifier les calculs d'une fonction qui fait la somme des éléments d'une liste d'entiers.

$$somme(\mathit{Nil}) = 0 \tag{8.1}$$

$$somme(\mathit{Cons}(a, l)) = a + somme(l) \tag{8.2}$$

Cette spécification doit être complétée par le profil de la fonction *somme* : $\mathit{Liste}(\mathbb{N}) \to \mathbb{N}$.

Remarquez bien où se place l'appel récursif : Si l'on veut respecter le typage, c'est la seule place possible !

Le calcul d'une fonction *récursive* prend une forme égalitaire. Ainsi, on déplie les calculs de

$$somme(\mathit{Cons}(2, \mathit{Cons}(7, \mathit{Cons}(3, \mathit{Nil}))))$$

en écrivant

$$
\begin{array}{ll}
& somme(\mathit{Cons}(2, \mathit{Cons}(7, \mathit{Cons}(3, \mathit{Nil})))) \\
= & \qquad\qquad < \text{par (8.2)} > \\
& 2 + somme(\mathit{Cons}(7, \mathit{Cons}(3, \mathit{Nil}))) \\
= & \qquad\qquad < \text{par (8.2)} > \\
& 2 + (7 + somme(\mathit{Cons}(3, \mathit{Nil}))) \\
= & \qquad\qquad < \text{par (8.2)} > \\
& 2 + (7 + (3 + somme(\mathit{Nil}))) \\
= & \qquad\qquad < \text{par (8.1)} > \\
& 2 + (7 + (3 + 0)) \\
= & \qquad\qquad < \text{par calcul arithmétique} > \\
& 12
\end{array}
$$

On peut aussi voir ce calcul comme une preuve équationnelle du théorème

$$somme(\mathit{Cons}(2, \mathit{Cons}(7, \mathit{Cons}(3, \mathit{Nil})))) = 12$$

Il n'y a dans cette preuve qu'une possibilité à chaque pas, parce que la preuve est entièrement guidée par la structure de la liste, décrite par les constructeurs *Cons* et *Nil*, et que cette structure est unique.

En conséquence, on a bien un *calcul* unique (on dit aussi déterministe). C'est pourquoi cette description est un *programme* (on dit aussi une *implantation*).
Exercice : Spécifier les calculs du produit des éléments d'une liste d'entiers.

I.2 Exercice

1. Ecrire une fonction qui fait la somme des éléments d'un ensemble, sans récursion cette fois, mais en utilisant un lieur. Il s'agit de compléter la partie droite de l'égalité

   $$somme(A) = \ldots$$

   pour tout ensemble d'entiers A.
2. Déplier le lieur.
3. Y-a-t-il plusieurs manières de déplier le lieur ? Pourquoi ?

I.3 Exercice

Montrer que le typage des égalités (8.1) et (8.2) est correct en prenant pour hypothèses le type de *somme*, le type de $+ : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ et celui de l'égalité des entiers $= : \mathbb{N} \times \mathbb{N} \to \{\mathit{vrai}, \mathit{faux}\}$.

I.4 Un argument supplémentaire influence le calcul

Le profil d'une fonction *compte* qui doit compter le nombre d'occurrences d'un élément donné dans une liste est $\mathit{compte} : \alpha \times \mathit{Liste}(\alpha) \to \mathbb{N}$.

Ici il y a deux arguments, une liste et une valeur de type α. Il n'y a pas de question à se poser, on va s'appuyer sur la liste en argument pour spécifier le calcul.

$$compte(a, \mathit{Nil}) = 0 \tag{8.3}$$
$$compte(a, \mathit{Cons}(a, l)) = 1 + compte(a, l) \tag{8.4}$$
$$compte(a, \mathit{Cons}(b, l)) = compte(a, l) \quad \text{pour } a \neq b \tag{8.5}$$

On a écrit trois égalités au lieu de deux pour distinguer le cas ou l'élément de la liste correspond à l'élément compté.
Exercice : Déplier le calcul de $compte(4, \mathit{Cons}(5, \mathit{Cons}(4, \mathit{Cons}(7, \mathit{Nil}))))$.

I.5 Le résultat est de type liste

Le profil d'une fonction *filtre-pair* qui doit retenir dans le résultat la liste de tous les éléments pairs d'une liste donnée est *filtre-pair* : $\mathit{Liste}(\mathbb{N}) \to \mathit{Liste}(\mathbb{N})$.

Le résultat est une liste qu'il va falloir construire.

$$\mathit{filtre\text{-}pair}(\mathit{Nil}) = \mathit{Nil} \tag{8.6}$$
$$\mathit{filtre\text{-}pair}(\mathit{Cons}(x, l)) = \mathit{Cons}(x, \mathit{filtre\text{-}pair}(l)) \quad \text{si } x \text{ est pair} \tag{8.7}$$
$$\mathit{filtre\text{-}pair}(\mathit{Cons}(x, l)) = \mathit{filtre\text{-}pair}(l) \quad \text{si } x \text{ n'est pas pair} \tag{8.8}$$

I.6 Choix d'une liste pour guider le calcul

Le profil d'une fonction *append* qui doit juxtaposer deux listes est

$$\mathit{append} : \mathit{Liste}(\alpha) \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha).$$

La question se pose : quelle liste choisir pour guider le calcul ? la première, la seconde, ou les deux ? Y-a-t-il un choix possible ? Si oui, quel est le meilleur choix ?

Il n'y a pas de réponse générale à ces questions. Ici, c'est la première liste qui va servir de guide.

$$\mathit{append}(\mathit{Nil}, l_2) = l_2 \tag{8.9}$$
$$\mathit{append}(\mathit{Cons}(x, l_1), l_2) = \mathit{Cons}(x, append(l_1, l_2)) \tag{8.10}$$

Exercice : Arrivez-vous à utiliser la deuxième liste pour guider le calcul de la juxtaposition ?

En général, le choix de la liste (ou des listes) guidant le mieux la spécification des calculs est affaire d'intuition (on devine pourquoi un choix est judicieux), d'expérience (on a déjà spécifié beaucoup de fonctions ressemblantes) et d'expérimentation (sans intuition et sans expérience, on peut toujours tester tous les guides l'un après l'autre).

I.7 Construction plus complexe d'une liste résultat d'un calcul

La construction d'une liste peut ne pas suivre l'ordre donné par la liste en argument. Considérons par exemple le calcul de la liste miroir d'une liste donnée par une fonction

$$\textit{miroir} \ : \ \textit{Liste}(\alpha) \ \rightarrow \ \textit{Liste}(\alpha).$$

$$\textit{miroir}(\textit{Nil}) \ = \ \textit{Nil} \tag{8.11}$$

$$\textit{miroir}(\textit{Cons}(x, l)) \ = \ \textit{append}(\textit{miroir}(l), \textit{Cons}(x, \textit{Nil})) \tag{8.12}$$

On ne peut pas se passer d'une fonction auxiliaire, telle que *append* ici.

II Suivre deux listes en même temps

Un calcul peut très bien suivre la structure de deux listes à la fois. Par exemple le calcul de la liste des couples d'éléments de même rang de deux listes données

$$\textit{mise-en-couples} : \textit{Liste}(\alpha) \times \textit{Liste}(\beta) \rightarrow \textit{Liste}(\alpha \times \beta)$$

est spécifié par

$$\textit{mise-en-couples}(\textit{Nil}, \textit{Nil}) \ = \ \textit{Nil} \tag{8.13}$$

$$\textit{mise-en-couples}(\textit{Cons}(x, l_1), \textit{Cons}(y, l_2)) \ = \ \textit{Cons}((x, y), \textit{mise-en-couples}(l_1, l_2)) \tag{8.14}$$

si les deux listes sont supposées de même longueur.

III Préconditions d'un calcul

Le calcul de *mise-en-couples* n'est correct que si les deux listes sont de même longueur. Aussi, on doit associer une précondition :
$Precondition(\textit{mise-en-couples}(l_1, l_2)) \equiv (longueur(l_1) = longueur(l_2))$.
Exercice :

1. Spécifier la fonction *longueur* comme un calcul.
2. Spécifier comme un calcul une fonction *zip* qui prendrait deux listes de longueurs quelconques et ne mettrait en couples que les deux débuts de même longueur des deux listes.

Une précondition est une hypothèse du calcul, elle est donc supposée être vérifiée par les arguments de la fonction.

Par exemple la fonction *car* qui prend l'élément de tête d'une liste ne s'applique que sur des listes non vides. Sa précondition est $Precondition(\textit{car}(l)) \equiv (l \neq \textit{Nil})$.

IV Ajouter des arguments accumulateurs du résultat

Décrivons le calcul *cumuler* : *Liste*$(\mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ qui ajoute à un entier la somme des éléments d'une liste.

$$\textit{cumuler}(\textit{Nil}, s) = s \tag{8.15}$$

$$\textit{cumuler}(\textit{Cons}(x, l), s) = cumuler(l, x + s) \tag{8.16}$$

Exercice : Déplier le calcul *cumuler*$(\textit{Cons}(5, \textit{Cons}(4, \textit{Nil})), 3)$. Remarquer que le calcul se porte sur l'argument que l'on appelle accumulateur du résultat.
Exercice : Définir la fonction *somme* en utilisant la fonction *cumuler*.

V Divers niveaux de spécifications

On a maintenant plusieurs manières de s'exprimer pour décrire un calcul.

1. Expressions logico-ensemblistes :

   $$m = max(A) \Leftrightarrow (m \in A \wedge (\forall x \ : \ x \in A \,.\, x \leq m))$$

   Il reste à ajouter les domaines des variables. Quelles sont les variables libres ? Quelles sont les variables liées ?
   Ce mode d'expression n'indique pas comment calculer m.

2. Expression itérative utilisant un lieur :

   $$\textit{somme-des-carres}(A) = (\Sigma x \ : \ x \in A \,.\, x^2)$$

   Il reste à ajouter les domaines des variables et le type de la fonction.
   Ce mode d'expression indique comment on calcule le résultat quand on déplie le lieur mais le dépliage peut se faire de plusieurs manières, ce qui n'affecte pas le résultat du calcul. Pourquoi ?

3. Expression récursive s'appuyant sur la structure des listes :

   $$\textit{somme}(\textit{Nil}) = 0 \tag{8.17}$$

   $$\textit{somme}(\textit{Cons}(x, l)) = x + \textit{somme}(l) \tag{8.18}$$

   Ce mode d'expression indique comment on calcule le résultat de manière déterministe, par un dépliage de la définition récursive.

4. On peut aussi donner des règles permettant une déduction du résultat, par un petit système formel.

   $$\frac{s = somme(\emptyset)}{s = 0}$$

   $$\frac{s = somme(A \cup B)}{s = somme(A) + somme(B)} \text{ si } A \neq \emptyset \wedge B \neq \emptyset \wedge A \cap B = \emptyset$$

   Ce mode d'expression indique comment on calcule le résultat quand on déroule une preuve sur des données closes, mais il y a souvent une multitude de preuves possibles et d'impasses.

Chapitre 9

Termes

Nous avons vu avec l'exemple des listes comment un type inductif se définit en fonction de lui-même par des règles de construction. Ces règles utilisent des *constructeurs*, qui sont des symboles spéciaux permettant d'écrire de manière unique chaque élément du type. Par exemple, les constructeurs du type *Liste*(α) sont *Cons* et *Nil*. Chaque constructeur a un nombre de paramètres fixé.

Ainsi, on définit un type inductif particulier chaque fois qu'on choisit un ensemble quelconque (fini) de symboles de construction. Dans ce chapitre, nous allons traiter ensemble tous ces cas, en désignant par *terme* tout élément d'un type inductif et en donnant une définition commune à tous les types inductifs, avec ou sans variables.

Le type des termes est très général. Il permet de représenter des expressions de n'importe quelle nature et la plupart des syntaxes peuvent être définies dans le cadre de ce type. Toute expression d'un calcul peut se représenter par un terme.

I Les symboles fonctionnels

Puisqu'un terme représente une expression, ses constructeurs sont souvent des symboles d'opérations. On les appelle *symboles fonctionnels* et on les écrit comme des fonctions, c'est-à-dire avant leurs paramètres, qui sont encadrés de parenthèses et séparés par des virgules.

On désigne par F un ensemble fini de *symboles fonctionnels* (ou opérateurs).

On associe à chaque symbole fonctionnel le nombre de ses opérandes. Ce nombre est appelé *arité* du symbole fonctionnel. Formellement, l'arité est une application a de F dans l'ensemble $\mathbb{N}$ des entiers naturels, ce que l'on note :

$$a : F \to \mathbb{N}$$

Pour tout entier naturel n, on désigne par F_n l'ensemble des symboles fonctionnels de F d'arité n. Formellement, on a :

$$F_n = \{f \in F,\, a(f) = n\}$$

Les symboles d'arité 0, éléments de F_0, sont appelés *constantes*.

I.1 Exemple

On peut avoir

$$F =_{\text{def}} \{+, -, *, /, \mathit{Moins}, \mathit{Abs}, 2, 3, 4\}$$

avec $F_0 =_{\text{def}} \{2, 3, 4\}$, $F_1 =_{\text{def}} \{\mathit{Abs}, \mathit{Moins}\}$, $F_2 =_{\text{def}} \{+, -, *, /\}$ et $F_i = \emptyset$ pour $i \geq 3$.

I.2 Convention de nommage

Les symboles fonctionnels qui ne sont pas des constantes sont souvent notés

$$f, g, h, k, f_1, g_1, h_1, k_1, \ldots$$

et les constantes

$$a, b, c, d, a_1, a_2, d_1, \ldots.$$

I.3 Exercice

Décrire l'ensemble des symboles fonctionnels employés dans chacune des expressions suivantes :

1. $*(*(+(2,3),5),+(*(4,\mathit{Moins}(3)),2))$
2. $f(g(a,h(c)),g(f(a,b,c),h(a)),a)$

II Les termes clos

L'ensemble T_F des *termes clos* est le type inductif (voir cours correspondant) défini par les règles d'inférence suivantes : Pour tout entier naturel n et pour tout symbole fonctionnel f de F_n, on a la règle :

$$\frac{t_1, \ldots, t_n \in T_F}{f(t_1, \ldots, t_n) \in T_F}$$

Remarque : pour n nul, $f()$ est plus simplement noté f.

II.1 Exercice

Montrer (prouver) que les expressions suivantes appartiennent à T_F, où F est un des ensembles de symboles fonctionnels trouvés dans l'exercice précédent.

1. $*(*(+(2,3),5),+(*(4,\mathit{Moins}(3)),2))$
2. $f(g(a,h(c)),g(f(a,b,c),h(a)),a)$

III Représentation arborescente des termes

Il est souvent utile de représenter graphiquement un terme par un arbre, en appliquant la règle suivante :

Un terme de la forme $f(t_1, \ldots, t_n)$ est représenté par un arbre dont la racine est étiquetée par f et qui a n fils, qui sont les représentations arborescentes de $t_1, \ldots, t_n$ dans cet ordre. Si n est nul, alors la racine n'a pas de fils. Cette règle est illustrée dans la figure 9.1 par la représentation arborescente du terme $f(a, a, g(f(a, b, c), h(a)))$.

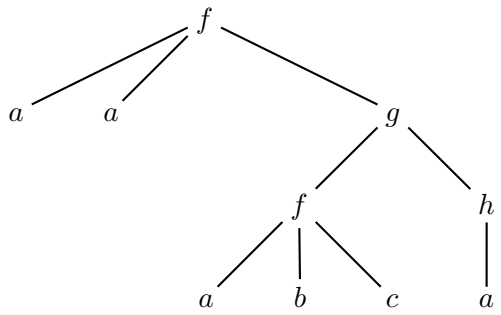

FIGURE 9.1 – Représentation arborescente d'un terme.

Dans l'arbre obtenu par application systématique de la règle précédente, tout nœud sans fils est appelé *feuille*. Les autres nœuds sont appelés *nœuds internes*.

III.1 Exercice

Dessiner sous forme arborescente les deux termes de l'exercice précédent.

III.2 Questions

Répondre aux questions suivantes, en justifiant la réponse :

1. Tout terme a-t-il une représentation arborescente ?
2. Toute arborescence est-elle un terme ?

IV Les termes avec variables

Dans une expression, une variable peut être remplacée (on dit *substituée*) par une expression quelconque. Par exemple, dans l'expression

$$e =_{\mathsf{def}} (2 + x) * (y * ((4 * \mathit{Moins}(x)) + 2))$$

on peut subtituer la variable x par $2 * x + 7$ et la variable y par $y + 2$, pour obtenir

$$e[x := 2 * x + 7, y := y + 2] =_{\mathsf{def}} (2 + (2 * x + 7)) * ((y + 2) * ((4 * (\mathit{Moins}(2 * x + 7)) + 2)).$$

Les variables seront souvent désignées par $x, y, z, x_1, y_1, z_1, \ldots$, sur la base des lettres de la fin de l'alphabet, à parir de x.

On désigne par X un ensemble quelconque (c'est-à-dire non nécessairement fini) de variables. L'ensemble $T_F(X)$ des *termes avec variables* dans X est le type inductif défini par les deux règles d'inférence suivantes :

—

$$\frac{x \in X}{x \in T_F(X)}$$

— Pour tout entier naturel n et pour tout symbole fonctionnel f de F_n,

$$\frac{t_1, \ldots, t_n \in T_F(X)}{f(t_1, \ldots, t_n) \in T_F(X)}$$

Les termes avec variables sont aussi appelés *termes ouverts*.

IV.1 Exercice

Avec $X = \{x, y, z\}$, décrire un ensemble des symboles fonctionnels F tel que les expressions suivantes soient dans $T_F(X)$:

1. $*(+(x, x), *(x, +(*(4, \mathit{Moins}(x)), 2)))$
2. $f(g(a, h(x)), y, g(f(a, z, z), h(y)))$

Syntaxiquement, les symboles fonctionnels sont des briques de base - des constructeurs de données - qu'on relie entre elles pour construire les termes. Les constantes sont situées aux feuilles de la représentation arborescente tandis que les symboles fonctionnels d'arité non nulle en sont les nœuds internes.

IV.2 Question

Peut-on représenter un terme ouvert de façon arborescente ? Quelle est la position des variables ?

V Le triangle d'un terme

On peut abstraire la représentation arborescente d'un terme sous la forme d'un triangle isocèle dont la base est en bas, comme le montre l'exemple de la figure 9.2.

Le sommet supérieur du triangle symbolise la racine et sa base symbolise toutes les feuilles. Sur cette base, les variables sont des feuilles prêtes à recevoir un (sous)-terme par l'application d'une substitution.

Ce genre de représentation triangulaire est utile au raisonnement et à la conception correcte de calculs sur les termes. Il est vivement conseillé de l'utiliser fréquemment.

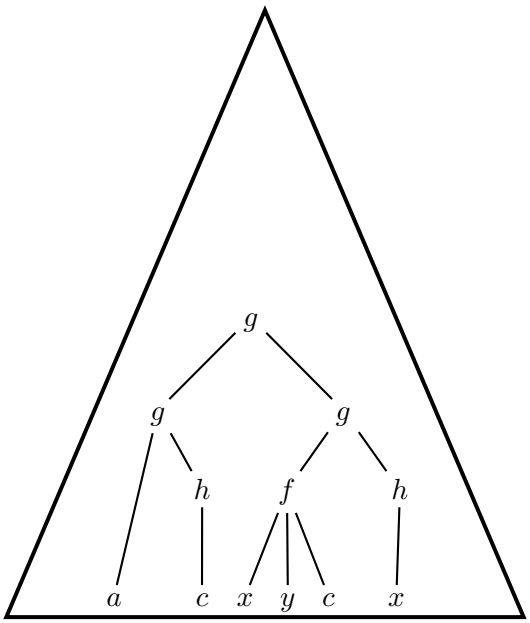

FIGURE 9.2 – Représentation d'un terme par un triangle.

V.1 Positions dans un terme

Il peut être utile algorithmiquement d'indiquer des positions spécifiques à l'intérieur d'un terme ou d'une arborescence, pour en extraire une partie (appelée *sous-terme* ou *sous-arbre*), ou pour désigner le symbole situé sur un nœud particulier.

Pour observer ainsi de façon plus précise l'intérieur du triangle, nous introduisons dans cette partie la notion très utile de position d'un nœud ou d'un sous-terme. La position d'un nœud dans une arborescence permet de repérer ce nœud particulier, quelle que soit sa profondeur (la profondeur d'un nœud est la distance de ce nœud à la racine de l'arbre, comptée en nombre d'arêtes qui les relient).

Pour cela on va en quelque sorte instituer un aiguillage possible dans le parcours de l'arborescence par l'intermédiaire de la numérotation $1, 2, \ldots, n$ des sous-termes immédiats d'une construction inductive $f(t_1, \ldots, t_n)$.

Par exemple, on numérote :

$$*([1] + ([1]x, [2]x), [2] * ([1]x, [2] + ([1] * ([1]4, [2]\mathit{Moins}([1]x)), [2]2)))$$

Une position est alors un chemin depuis la racine de l'arborescence, codée sous la forme d'un **mot**, tel que 1, 2, 1.1 ou $1.2.2$, dont les lettres sont les entiers naturels non nuls. Le type *Pos* des positions est constitué de l'ensemble des mots sur $\mathbb{N}-\{0\}$. On le note [1] $\mathit{Pos} =_{\mathsf{def}} (\mathbb{N}-\{0\})^*$.

On désigne en général les positions par des variables $u, v, w, u_1, v_1, w_1, \ldots$ qui commencent par la lettre u, v ou w.

On peut étiqueter les nœuds de la représentation arborescente d'un terme par leur position. On attribue à la racine la position ϵ (notation habituelle du *mot vide*). Les fils du nœud de position u ont pour position $u.1, u.2, \ldots$ de gauche à droite.

1. Notation classique en Théorie des Langages.

Par exemple, l'arborescence représentant le terme $f(g(a,b),h(c,d,e))$ compte 8 nœuds, qui ont pour étiquette

$$f, g, a, b, h, c, d \text{ et } e$$

et pour position, dans le même ordre

$$\epsilon, 1, 1.1, 1.2, 2, 2.1, 2.2 \text{ et } 2.3.$$

La figure 9.3 illustre cette numérotation sur l'exemple du terme $f(a,a,g(f(a,b,c),h(a)))$.

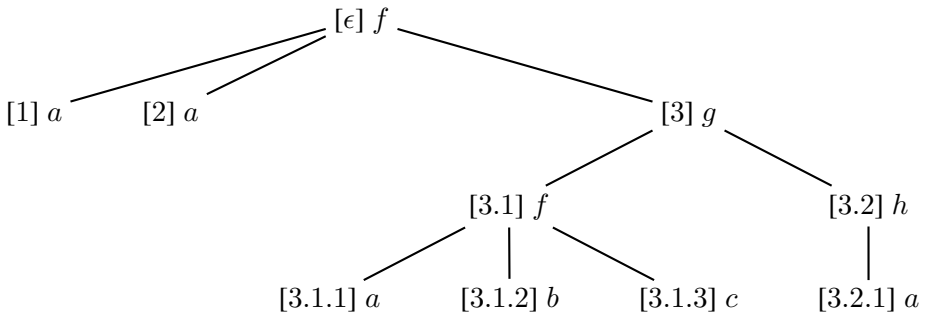

FIGURE 9.3 – Position des sous-termes d'un terme.

L'opération de base dans un ensemble des mots est la concaténation, notée par un point (.). On l'utilise ici entre positions.

C'est une opération associative et le mot vide est son élément neutre. Les axiomes de cette opération sont donc :

$$u.\epsilon = u \quad \text{(neutre à droite)} \tag{9.1}$$

$$\epsilon.u = u \quad \text{(neutre à gauche)} \tag{9.2}$$

$$(u.v).w = u.(v.w) \quad \text{(associativité)} \tag{9.3}$$

V.1.1 Question

Serait-il possible de définir le type *Pos* inductivement ?

V.2 Domaine

On peut désormais associer à tout terme l'ensemble des positions des nœuds de sa représentation arborescente. Cet ensemble est appelé *domaine* du terme. Cette association peut être définie comme un calcul *dom*, dont le profil est

$$dom : T_F(X) \to \{Pos\}$$

et dont la définition est

$$dom(x) = \{\epsilon\} \tag{9.4}$$

$$dom(f(t_1,\ldots,t_n)) = \{\epsilon\} \cup \bigcup_{i,1\leq i\leq n} \left(\bigcup_{u,u\in dom(t_i)} \{i.u\} \right) \tag{9.5}$$

L'équation (9.5) utilise le lieur $\bigcup$ de l'union généralisée. On peut aussi l'écrire (voir cours sur les notations)

$$\mathit{dom}(f(t_1,\ldots,t_n)) = \{\epsilon\} \cup \Big(\bigcup i : 1 \leq i \leq n \;.\; \Big(\bigcup u : u \in \mathit{dom}(t_i) \;.\; \{i.u\}\Big)\Big)$$

V.2.1 Exercice

Calculer $dom(f(g(a,b),h(c,d,e)))$.

V.3 Localisation d'un sous-terme

On sait désormais trouver un sous-terme s d'un terme t en donnant sa position u dans le terme ($u \in \mathit{dom}(t)$). Cette opération est notée *at*, c'est-à-dire que $s = \mathit{at}(t,u)$. Elle est définie par

$$\mathit{at}(t,\epsilon) = t \tag{9.6}$$

$$\mathit{at}(f(t_1,\ldots,t_n),i.u) = \mathit{at}(t_i,u) \text{ pour } 1 \leq i \leq n \tag{9.7}$$

Ainsi, l'ensemble des sous-termes d'un terme se définit par

$$\bigcup_{u,u\in\mathit{dom}(t)} \{\mathit{at}(t,u)\}$$

Chapitre 10

Principe d'induction structurelle

I Introduction

L'induction structurelle se fonde sur le concept de type inductif introduit dans le chapitre 7.

Le type inductif le plus simple est le type $\mathit{Liste}(\alpha)$ défini dans le chapitre 8. Le chapitre 8 présente des preuves par induction sur les listes, mais le principe d'induction utilisé n'est pas clairement énoncé. Ce document a pour objectif principal de combler cette lacune.

L'induction structurelle généralise le principe de récurrence sur les entiers naturels, étudié dans le chapitre 6 et formalisé dans la partie II. La partie III énonce le principe d'induction sur les listes sous la forme d'une règle d'inférence. La partie IV généralise le principe d'induction structurelle à tous les types inductifs.

II Le principe de récurrence

Ce principe sert à prouver des théorèmes sur les entiers naturels. Il est présenté dans le chapitre 6. Sous forme de règle d'inférence, il s'écrit :

$$\frac{P(0), \qquad \forall n : \mathbb{N} \,.\, P(n) \Rightarrow P(n+1)}{\forall n : \mathbb{N} \,.\, P(n)}$$

III Induction sur les listes

On peut énoncer explicitement le principe d'induction sur les listes sous forme d'une règle d'inférence :

$$\frac{P(\mathit{Nil}), \qquad \forall l : \mathit{Liste}(\alpha) \,.\, \forall x : \alpha \,.\, P(l) \Rightarrow P(\mathit{Cons}(x, l))}{\forall l : \mathit{Liste}(\alpha) \,.\, P(l)}$$

Cette règle signifie que, pour prouver une propriété $P(l)$ pour toute liste l, il suffit de prouver cette propriété pour la liste vide, et, séparément, de prouver, sous l'hypothèse que la propriété P est vraie pour une liste l, qu'elle est encore vraie pour toute liste de la forme $\mathit{Cons}(x, l)$.

La première condition est formalisée par la première prémisse $P(\mathit{Nil})$ et elle est appelée "cas de base". La seconde condition est formalisée par la seconde prémisse $P(l) \Rightarrow P(\mathit{Cons}(x, l)$ et elle est appelée "pas d'induction".

III.1 Exemple

Une preuve par induction sur les listes fonctionne comme une preuve par récurrence.

Par exemple, prouvons formellement, inductivement, que pour toute liste de listes d'entiers l, on a l'égalité :

$$\textit{somme-liste}(\textit{liste-sommes}(l)) = \textit{somme-liste}(\textit{aplatir}(l)).$$

Les fonctions *somme-liste*, *liste-sommes* et *aplatir* sont définies comme suit [1] :

$$\begin{aligned}
&(10.1) && \textit{somme-liste}(\textit{Nil}) &&= 0\\
&(10.2) && \textit{somme-liste}(\textit{Cons}(x,l)) &&= x + \textit{somme-liste}(l)\\
&(10.3) && \textit{liste-sommes}(\textit{Nil}) &&= \textit{Nil}\\
&(10.4) && \textit{liste-sommes}(\textit{Cons}(l,r)) &&= \textit{Cons}(\textit{somme-liste}(l), \textit{liste-sommes}(r))\\
&(10.5) && \textit{aplatir}(\textit{Nil}) &&= \textit{Nil}\\
&(10.6) && \textit{aplatir}(\textit{Cons}(l,r)) &&= \textit{append}(l, \textit{aplatir}(r))
\end{aligned}$$

III.1.1 Preuve formelle

La propriété $\textit{somme-liste}(\textit{liste-sommes}(l)) = \textit{somme-liste}(\textit{aplatir}(l))$ est notée $P(l)$. La preuve s'effectue par induction sur la liste l dont parle cette propriété $P(l)$.

Cas de base :

Le cas de base est $P(\textit{Nil})$. Il s'agit donc de prouver

$$\textit{somme-liste}(\textit{liste-sommes}(\textit{Nil})) = \textit{somme-liste}(\textit{aplatir}(\textit{Nil})).$$

On a d'une part

$\textit{somme-liste}(\textit{liste-sommes}(\textit{Nil}))$
$= \quad <$ règle (10.3), position 1, substitution $[\,]>$
$\textit{somme-liste}(\textit{Nil})$
$= \quad <$ règle (10.1), position ϵ, substitution $[\,]>$
0

et d'autre part

$\textit{somme-liste}(\textit{aplatir}(\textit{Nil}))$
$= \quad <$ règle (10.5), position 1, substitution $[\,]>$
$\textit{somme-liste}(\textit{Nil})$
$= \quad <$ règle (10.1), position ϵ, substitution $[\,]>$
0,

ce qui établit ce cas de base.

Pour le pas d'induction, on suppose que, pour une liste l quelconque fixée,

$$\textit{somme-liste}(\textit{liste-sommes}(l)) = \textit{somme-liste}(\textit{aplatir}(l)).$$

C'est l'affirmation $P(l)$ de la règle d'induction, appelée "hypothèse d'induction". On cherche à s'en servir pour prouver $P(\textit{Cons}(x,l))$, qui est ici

$$\textit{somme-liste}(\textit{liste-sommes}(\textit{Cons}(x,l))) = \textit{somme-liste}(\textit{aplatir}(\textit{Cons}(x,l))).$$

1. C'est l'exercice 1 de la feuille "Exercices sur les listes".

On simplifie au maximum le membre de gauche de cette égalité, étape par étape, comme suit :

$$\begin{array}{ll} & \textit{somme-liste}(\textit{liste-sommes}(\textit{Cons}(l,r))) \\ = & \quad < \text{ règle (10.4), position } 1, \text{ substitution } [\,] > \\ & \textit{somme-liste}(\textit{Cons}(\textit{somme-liste}(l), \textit{liste-sommes}(r))) \\ = & \quad < \text{ règle (10.2), position } \epsilon, \text{ substitution } [\,] > \\ & \textit{somme-liste}(l) + \textit{somme-liste}(\textit{liste-sommes}(r)) \\ = & \quad < \text{ hypothèse d'induction, position } 1, \text{ substitution } [l := r] > \\ & \textit{somme-liste}(l) + \textit{somme-liste}(\textit{aplatir}(r)) \end{array}$$

La troisième égalité applique l'hypothèse d'induction à la liste d'entiers r.

On simplifie au maximum le membre de droite de l'égalité $P(\textit{Cons}(x,l))$, étape par étape, comme suit :

$$\begin{array}{ll} & \textit{somme-liste}(\textit{aplatir}(\textit{Cons}(l,r))) \\ = & \quad < \text{ règle (10.6), position } 1, \text{ substitution } [\,] > \\ & \textit{somme-liste}(\textit{append}(l, \textit{aplatir}(r))) \\ = & \quad < \text{ lemme 1, position } \epsilon, \text{ substitution } [l_1 := l, l_2 := \textit{aplatir}(r)] > \\ & \textit{somme-liste}(\textit{append}(l, \textit{aplatir}(r))) \end{array}$$

La deuxième égalité utilise le lemme suivant, dont la preuve est laissée en exercice.

Lemme 1 : $\textit{somme-liste}(\textit{append}(l_1,l_2)) = \textit{somme-liste}(l_1) + \textit{somme-liste}(l_2)$.

Puisque les deux membres de l'égalité $P(\textit{Cons}(x,l))$ sont égaux au même terme, ils sont égaux entre eux, ce qui prouve l'égalité $P(\textit{Cons}(x,l))$.

IV Principe général

Il y a un principe d'induction par type inductif. On ne peut donc pas énoncer tous ces principes. Cependant, on peut voir les constructeurs d'un type comme un ensemble F de symboles fonctionnels. Le type inductif ainsi défini correspond à l'ensemble de termes clos $T(F)$ (voir chapitre sur les termes).

Le principe d'induction correspondant est

$$\frac{\bigwedge_{f\in F} \forall t_1,\ldots,t_n : T(F)\ .\ \bigwedge_{i=1}^{i=n_f} P(t_i) \Rightarrow P(f(t_1,\ldots,t_n))}{\forall t : T(F)\ .\ P(t)}$$

où n_f est l'arité de f.

V Raisonnement équationnel inductif

Dans le cadre des types et des calculs inductifs, la preuve inductive d'un théorème exprimé sous forme d'égalité peut souvent se faire par réduction de chaque membre de l'égalité et comparaison syntaxique des formes réduites.

Cette partie illustre cette stratégie par un exemple et formalise la présentation des déductions équationnelles, notamment afin de préparer l'étude de leur automatisation.

V.1 Exemple de preuve inductive

V.1.1 Théorème

Prouver le théorème

$$\textit{length}(\textit{append}(l, m)) = \textit{length}(l) + \textit{length}(m) \tag{10.7}$$

sur les listes (génériques) de type $Liste(\alpha)$.

L'induction peut porter sur l, sur m, ou sur ces deux variables. En général, mieux vaut ne mener qu'une seule induction à la fois. Si nécessaire, l'autre induction peut être ajoutée à l'intérieur de la première.

V.1.2 Axiomes

Chaque équation des définitions suivantes des calculs *length* et *append* est un axiome utilisable dans la preuve du théorème. Les variables de ces axiomes sont volontairement choisies distinctes entre axiomes, afin de mieux mettre en évidence le mécanisme de substitution.

$$\textit{append}(\textit{Nil}, l_1) = l_1 \tag{10.8}$$

$$\textit{append}((\textit{Cons}(x_1, l_2), l_3) = \textit{Cons}(x_1, \textit{append}(l_2, l_3)) \tag{10.9}$$

$$\textit{length}(\textit{Nil}) = 0 \tag{10.10}$$

$$\textit{length}(\textit{Cons}(x_2, l_4)) = 1 + \textit{length}(l_4) \tag{10.11}$$

V.1.3 Choix du schéma inductif

On choisit de prouver (10.7) par induction sur l, sa première variable. Autrement dit, on considère (10.7) comme une propriété $P(l)$ dont m est un paramètre. De manière équivalente, on peut considérer que m est universellement quantifié sur $Liste(\alpha)$ dans $P(l)$.

V.1.4 Cas de base

Il s'agit de prouver $P(\textit{Nil})$.

La réduction du membre gauche de cette égalité donne, étape par étape :

$\textit{length}(\textit{append}(\textit{Nil}, m))$

= < règle (10.8), position 1, substitution $[l_1 := m]$ >

$\textit{length}(m)$

La réduction du membre droit (où $+$ est un symbole fonctionnel d'arité 2 en notation infixée) donne :

$\textit{length}(\textit{Nil}) + \textit{length}(m)$

= < règle (10.10), position 1, substitution $[\,]$ >

$0 + \textit{length}(m)$

= < axiome $0 + x = x$ de l'arithmétique, position ϵ, substitution $[\,]$ >

length(m)

Les deux formes réduites étant syntaxiquement égales, $P(\textit{Nil})$ est prouvé.

V.1.5 Pas d'induction

Il s'agit de prouver $P(Cons(x, l))$ sous l'hypothèse $P(l)$, qui s'ajoute aux axiomes mais peut être utilisée dans les deux sens.

La réduction du membre gauche donne, étape par étape :

length(*append*(*Cons*$(x, l), m)$)
= < règle (10.9), position 1, substitution $[x_1 := x, l_2 := l, l_3 := m]$ >
length(*Cons*(x,*append*(l, m)))
= < règle (10.11), position ϵ, substitution $[x_2 := x, l_4 :=$ *append*$(l, m)]$ >
$1 +$ *length*(*append*(l, m))
= < hypothèse d'induction $P(l)$, position 2, substitution $[\,]$ >
$1 +$ *length*$(l) +$ *length*(m)

La réduction du membre droit (où $+$ est un symbole fonctionnel d'arité 2 en notation infixée) donne :

length$(Cons(x, l)) +$ *length*(m)
= < règle (10.11), position 1, substitution $[x_2 := x, l_4 := l]$ >
$1 +$ *length*$(l) +$ *length*(m)

V.2 Formalisation du raisonnement équationnel

Les raisonnements équationnels utilisés dans la partie précédente sont présentés étape par étape. Une étape consiste **uniquement** à appliquer une instance d'**une seule** règle de réécriture en **une seule** position d'un terme, pour obtenir un autre terme.

A chaque étape, on cite, dans cet ordre, la règle utilisée, la position où elle s'applique et la substitution qui produit l'instance requise.

Si la règle vient d'une égalité utilisable dans les deux sens, on doit préciser le sens d'utilisation de l'égalité, lorsqu'on la cite au lieu de la règle de réécriture.

Chapitre 11

Lambda-notation des fonctions

Pour définir une fonction de la variable x, le mathématicien écrit $f(x) = E$, où E est une expression dans laquelle x apparaît. Cette notation familière est pourtant source de confusion : elle ne montre pas le lien, qu'elle établit pourtant, entre la variable x dans $f(x)$ et dans E.

La lambda-notation des fonctions exposée dans ce chapitre évite cette confusion. Elle inscrit le concept de fonction comme un cas particulier d'expression liée.

Grâce à cette notation, les opérations usuelles sur les fonctions (application et composition) ont des équivalents syntaxiques, connus sous le nom de *lambda-calcul*. Ce système formel est décrit dans la partie II.

Le lambda-calcul est une théorie de fonctions construites par application et abstraction sur une variable. Sa description, bien que relativement simple, conduit à une théorie des modèles très complexe et très riche. Le langage du lambda-calcul est défini dans la partie I.

I Lambda-expressions

I.1 Exemple

La fonction *surface* qui calcule la surface d'un triangle de base et de hauteur données s'écrit ainsi :

$$surface =_{\mathsf{def}} (\lambda\, base, hauteur\, .\, (base * hauteur/2))$$

où $=_{\mathsf{def}}$ veut dire *est par définition*. Dans cette écriture, le symbole λ lie les variables $base$ et $hauteur$ avec l'expression qui suit le point (.).

I.2 Le lieur λ

Dans l'expression $\lambda\, x\, .\, E$, le symbole λ joue un rôle de lieur, selon des règles de syntaxe vues dans le chapitre "Notations à lieurs" à propos des quantificateurs $\forall$ et $\exists$.

Nous invitons donc le lecteur à consulter dans ce chapitre les conventions syntaxiques qui s'appliquent encore ici : parenthèses omises, variables libres et liées, . . .

Vous pouvez remarquer que l'on n'a pas précisé ici le domaine (le *type*) des variables liées. Ceci fera l'objet d'un prochain chapitre.

II Le lambda-calcul

Le lambda-calcul est un système formel dont les expressions comportent des lambda-notations et dont les règles formalisent les deux calculs fondamentaux sur les fonctions, qui sont l'application d'une fonction à une expression et la composition des fonctions.

Comme la lambda-notation représente une fonction par une expression, elle ramène la composition des fonctions au cas de l'application d'une fonction à une expression.

II.1 Expressions du lambda-calcul

Ces expressions sont appelées *termes* du lambda-calcul, car elles obéissent à une définition inductive qui est un cas particulier du type inductif des termes au programme d'un prochain chapitre.

Un terme du lambda-calcul est :

— soit une variable $x, y, \ldots$,

— soit de la forme $(\lambda\, x\, .\, E)$, où x est une variable et E est un terme,

— soit de la forme $(E\ F)$, avec E et F deux termes quelconques.

Un terme de la forme $(\lambda\, x\, .\, E)$ est appelé *abstraction* et représente la fonction qui associe l'expression E à la variable x. Notez qu'on ne précise pas si la variable x apparaît ou non dans E, car cela n'introduit aucun cas particulier.

Un terme de la forme $(E\ F)$ est appelé *application* du terme E au terme F. Le cas intéressant est celui où E représente une fonction (donc, est une abstraction). Dans ce cas, $(E\ F)$ représente l'application de cette fonction au terme F.

II.2 Notation simplifiée

L'écriture des termes du lambda-calcul se trouve simplifiée si l'on admet les deux conventions suivantes d'omission de parenthèses :

— $(E\ F\ G)$ est la notation simplifiée de $((E\ F)\ G)$

— $(\lambda\, x, y\, .\, E)$ est la notation simplifiée de $(\lambda\, x\, .\, (\lambda\, y\, .\, E))$

II.3 Traduire l'application d'une fonction

L'égalité suivante entre lambda-termes traduit l'application de la fonction $(\lambda\, x\, .\, E)$ au terme F :

$$(\lambda\, x\, .\, E)\ F = E\,(F/x) \tag{11.1}$$

Le second membre signifie qu'on remplace syntaxiquement dans E chaque occurrence de x par F. Cette opération, dite de *substitution*, a été introduite dans la partie II.3.

Il convient cependant d'ajouter une condition d'application, comme le montre l'exemple suivant : Si $E = (\lambda\, y\, .\, x)$ et $F = y$, alors le remplacement de x par y dans E donne $(\lambda\, y\, .\, y)$ tandis que le résultat attendu de l'application $(\lambda\ x\, .\, E)\ F$ est $(\lambda\, z\, .\, y)$. En effet, la variable y n'intervient pas dans l'expression de E après le ., donc il doit en être de même après substitution.

Dans le cas général, cette condition sur l'égalité (11.1) est

$$\textit{variables-liées}(E) \cap \textit{variables-libres}(F) = \emptyset.$$

Chapitre 12

Vérification et inférence de types

Ce chapitre montre comment le typage permet à l'informaticien d'éviter des erreurs de spécification, de programmation ou de logique. Il lui suffit de connaître deux algorithmes fondamentaux sur les types :

1. La détection des erreurs de typage : Dire si une expression e est correctement typée dans un type donné *T*. Ceci revient à prouver que $e \ : \ T$ est un théorème, dans un système formel explicitant les règles de typage.
2. L'inférence de type : Donner un type à *e* et à ses variables afin que e soit une expression correctement typée. Ce problème n'a pas toujours de solution algorithmique. Nous verrons ici qu'il en a une pour les types génériques. Il y a souvent plusieurs solutions, mais on peut trouver par déduction la plus générale des solutions.

I Introduction

I.1 Pourquoi des types ?

On comprend aisément que l'attribution d'un type particulier à chaque élément d'une expression en clarifie le sens.

Dans ce but, de nombreux langages de programmation disposent d'un système de types (dits *de base*, *élémentaires* ou *prédéfinis*) et de règles de construction de nouveaux types par l'utilisateur. Ces règles *de typage* permettent d'exprimer quels opérateurs et quelles fonctions sont applicables à quelles expressions, éliminant ainsi certaines erreurs de programmation.

I.2 Importance des types pour la consistance des logiques

L'importance des types a été soulignée par le paradoxe de Russel dans la théorie des ensembles. Ce paradoxe est le même que le paradoxe du barbier qui s'énonce ainsi : "Dans le village, le barbier rase tout homme qui ne se rase pas lui-même. Qui rase le barbier ?". On peut aussi bien répondre par "le barbier" que par "certainement pas le barbier", ce qui rend équivalent une expression "le barbier rase le barbier" avec sa négation "le barbier ne rase pas le barbier".

Cela crée donc un paradoxe, autrement dit une inconsistance.

Dans la théorie des ensembles, ce paradoxe s'énonce par : "Soit S l'ensemble des ensembles qui n'ont pas eux-mêmes pour élément. Est-ce que $S \in S$?". Donc, on a encore $S \in S \equiv S \notin S$.

Un moyen d'éviter le paradoxe est d'introduire la notion de type *ensemble* de valeurs d'un certain type, par exemple les ensembles d'entiers rassemblés dans le type *SET*$(\mathbb{N})$. Avec ce type *ensemble*, ce paradoxe ne peut pas exister car, pour avoir $S \in S$, il faudrait que S soit tout à la fois de type *SET*$(\mathbb{N})$ et de type *SET*(*SET*$(\mathbb{N})$), ce qui est considéré comme impossible. Donc la seule réponse *bien typée* est $S \notin S$.

I.3 Importance des types pour les spécifications

Connaître le type d'une expression, c'est connaître l'ensemble des valeurs qu'elle est susceptible de prendre quand elle est évaluée. Cela semble quand même la moindre des choses quand on est en train de spécifier une application.
Exercice : Quel est le type des expressions suivantes ?

1. $(x + 2) * y$
2. *append*(*Cons*(a, l),*feuilles*(t))
   Cons est un constructeur de listes vu dans un chapitre précédent. *append* est la fonction de concaténation de deux listes et *feuilles* est la fonction qui retourne la liste des feuilles d'un arbre.
3. $\lambda\, l$. *append*(*Cons*(a, l),*feuilles*(t))
4. $\lambda\, x, y$. $x \in f(f(y))$

C'est facile si vous connaissez le type des ingrédients : fonctions, variables et constructions utilisées. Avoir une discipline de type, c'est se forcer à plus de précision et de rigueur dans l'expression et donc éviter des erreurs.

I.4 Importance des types pour la programmation

Il y a toujours un typage implicite ou explicite dans les expressions et les constructions d'un programme : on manipule des ensembles de valeurs, on structure des données. Cependant les types contiennent des informations sur les valeurs qui ne sont pas toujours précisées ni exigées dans le programme.

A tout langage de programmation est attaché un certain nombre de regroupements des types de base que l'on peut utiliser.
Exercice : Quels sont les types possibles en C, en PASCAL ?

Le langage C qui demande de préciser le type des variables et vérifie un typage cohérent à la compilation du programme est dit à typage fort. Un langage qui n'exige rien de la sorte est dit à typage faible.

Il est important de se poser la question et d'écrire en commentaire quels sont les types des fonctions, des constantes et des domaines des variables utilisés, pour :

— une documentation précise des programmes,

— une première approche de la vérification de la correction de ces programmes.

Il a été dit que l'exécution des programmes bien typés "never goes wrong".

I.5 Origine des types génériques

Vous connaissez tous le type tableau de PASCAL : Il peut y avoir des tableaux d'entiers, des tableaux de réels, des tableaux de booléens, mais aussi des tableaux de tableaux d'entiers, etc.

Pour regrouper tous ces types sous le même concept de tableau, nous dirons que le type "tableau" est *générique* (on dit aussi *paramétrique* ou *polymorphe*).

Dans ce chapitre, nous allons apprendre comment noter les types génériques et comment relier ces types entre eux.

L'exemple du type des listes sera repris pour illustrer divers aspects de la généricité.

II Définitions et notations

La notation $e : T$ est usuellement utilisée pour exprimer que l'expression e est de type T. Même si cette notation ressemble à la syntaxe de la déclaration de variables dans votre langage de programmation favori, ce n'est pas une déclaration, mais une affirmation logique (vraie ou fausse).

On peut donner diverses interprétations pour $e : T$, mais la plus simple est largement suffisante pour tout ce module. Elle consiste à considérer un type comme un ensemble (parfois infini) de données de même nature. On peut interpréter $e : T$ par "e appartient à T", ce qui signifie que toutes les valeurs possibles de l'expression e appartiennent à l'ensemble T. Cependant, on distinguera soigneusement le symbole ":" du symbole "$\in$" qui exprime l'appartenance à un ensemble, ne serait-ce que pour pouvoir associer un type à ce symbole en écrivant $\in : T$.

Pour désigner un type générique, on va employer une variable substituable par un type quelconque : une *variable de type*.

Pour distinguer ces variables de type des variables substituables par des termes ou par des expressions, notées $x\,, y\,, z, \cdots$, nous les désignerons par des lettres grecques $\alpha\,, \beta\,, \gamma, \cdots$.

Ainsi on parlera du type générique *Liste*(α) pour signifier que la variable α peut être substituée par n'importe quel type (générique ou pas).

Par exemple, avec la substitution $[\alpha := \mathbb{N}]$, on obtient le type *Liste*$(\mathbb{N})$ des listes d'entiers naturels.

On définira et on utilisera aussi le type générique *Arbre*(α) des arbres binaires à feuilles de type quelconque.

II.1 Les constructeurs des types génériques

Un constructeur d'un type T est une fonction qui explique comment construire certains éléments du type T. Lorsque le type T est générique, ses constructeurs ont un profil qui dépend de la variable de type.

Par exemple, les constructeurs *Noeud* et *Feuille* du type *Arbre*(α) ont pour profil :

$$\textit{Feuille} : \alpha \to \textit{Arbre}(\alpha)$$

$$\textit{Noeud} : \textit{Arbre}(\alpha) \times \textit{Arbre}(\alpha) \to \textit{Arbre}(\alpha)$$

Il ne faut surtout pas confondre les constructeurs des éléments du type, comme *Noeud* et *Feuille*, avec le type générique *Arbre*.

III Le type des listes

Le type générique *Liste*(α) dispose des constructeurs suivants :
— *Nil* : *Liste*(α), qui construit une liste vide
— *Cons* : $\alpha \times$ *Liste*(α) $\rightarrow$ *Liste*(α), tel que *Cons*(x, l) désigne la liste obtenue en ajoutant x (de type α) au début de la liste l.

On donne à ces constructeurs une propriété de *liberté* qui inscrit le type *Liste* comme un cas particulier de type inductif. Les types inductifs ont étudiés dans le chapitre 7. Le type *Liste* a été étudié dans le chapitre 8. Il sert ici d'exemple et de base dans divers exercices.

La liberté des constructeurs de listes peut s'exprimer par les relations suivantes

$$\textit{Cons}(x_1, l_1) = \textit{Cons}(x_2, l_2) \Rightarrow (x_1 = x_2) \wedge (l_1 = l_2)$$

$$\textit{Cons}(x, l) \neq \textit{Nil}$$

qui signifient que toute liste admet une unique expression composée de *Nil* et de *Cons*.

Il en résulte que l'extraction du premier élément d'une liste est une fonction générique, qu'on pourra définir par équations en guise d'exercice.

IV Le type des ensembles

Si α est un type, le type des ensembles dont tous les éléments sont de type α est noté *SET*(α).

Remarque : Ce type est aussi parfois noté $\{\alpha\}$, mais cette seconde notation est en conflit avec le constructeur de singletons : Usuellement, $\{x\}$ désigne l'ensemble dont le seul élément est x.

IV.1 Exercice

On désigne par $\mathcal{P}(E)$ l'ensemble des parties (sous-ensembles) de l'ensemble E. Le symbole $\subseteq$ traduit l'inclusion large d'un ensemble dans un autre, qui inclut le cas où les deux ensembles sont égaux.

1. Proposer un profil (type) pour $\mathcal{P}$.
2. Expliquer pourquoi l'expression $x \subseteq \mathcal{P}(x)$, de type $\{\textit{vrai}, \textit{faux}\}$, est douteuse.
3. Expliquer pourquoi l'expression $x \in \mathcal{P}(x)$ peut être correcte.

V Les fonctions à profil générique

On a vu qu'une fonction peut avoir un profil générique, tel que $\alpha \rightarrow \beta$ ou $\alpha \times$ *Liste*(α) $\rightarrow \alpha$.

Quel est le type le plus général possible que l'on peut donner à une fonction donnée ? A priori, c'est α, mais ce n'est pas très informatif. En fait, le type *le plus général* que l'on cherche à écrire doit être *le plus informatif* possible, (donc le plus instancié possible) dans l'ensemble des plus généraux.

Par exemple, la fonction *miroir* d'une liste a le profil le plus général *Liste*(α) $\rightarrow$ *Liste*(α). C'est une instance de α, de $\alpha \rightarrow \beta$, de $\alpha \rightarrow \alpha$, de *Liste*($\alpha$) $\rightarrow \beta$, de $\beta \rightarrow$ *Liste*(α), de

Liste$(\alpha) \rightarrow$ *Liste*(β). Tous ces types seraient plus généraux que *Liste*$(\alpha) \rightarrow$ *Liste*(α). Mais c'est *Liste*$(\alpha) \rightarrow$ *Liste*(α) qui est la bonne réponse car si on l'instancie un peu plus, comme par exemple *Liste*$(\mathbb{N}) \rightarrow$ *Liste*$(\mathbb{N})$, on a trop perdu en généralité.

La fonction identité a son profil le plus général qui se note $\alpha \rightarrow \alpha$, car la fonction id existe dans n'importe quel type.

De même, on peut donner à l'égalité le profil le plus général suivant $\alpha \times \alpha \rightarrow \{vrai, faux\}$.

Quelqu'un (Ph. Wadler) a lancé le défi suivant : "Donnez moi le profil le plus général d'une fonction générique et je vous dirai ce qu'elle est !" Autrement dit, pas besoin de programme dans ces cas-là. Le type **est** le programme.

C'est le cas pour la fonction identité qui est la seule fonction qui peut avoir $\alpha \rightarrow \alpha$ comme profil le plus général. Une fonction d'un profil aussi général ne peut que ne rien faire.

V.1 Conclusion de cette partie

Sans aller jusqu'aux types qui sont des programmes, nous verrons plus loin comment un bon typage limite considérablement les erreurs de programmation.

VI Règles de typage

VI.1 Expressions de type

Les expressions de type sont formées avec :

— des types de base tels que $\mathbb{N}$, $\{vrai, faux\}$ (souvent noté $\mathbb{B}$), $\mathbb{Z}$, $\mathbb{R}$ ou d'autres symboles d'ensembles connus,

— des produits cartésiens ($\times$) d'ensembles, par exemple $A \times B$, ensemble des couples formés d'un élément de *A* et d'un élément de *B*.

— le constructeur *SET* tel que *SET*(A) désigne le type des ensembles dont tous les éléments sont de type *A*, c'est-à-dire l'ensemble des parties de *A*, que l'on note aussi $\mathcal{P}(A)$ ou 2^A, notation anglo-saxonne intéressante (pourquoi ?).

— le constructeur des ensembles de fonctions, par exemple $A \rightarrow B$ est l'ensemble des fonctions de *A* dans *B*,

— d'autres constructeurs de types, comme par exemple *Liste*$(\ldots)$.

Dans les expressions de type, les *variables de type* (substituables par des expressions de type) sont notées par des lettres grecques : $\alpha,\ \beta,\ \gamma, \ldots$

VI.2 Règles d'inférence

Les règles d'inférence de type sont :

— *application* (élimination d'une fonction) :

$$\frac{f\ :\ \alpha_1 \times \ldots \times \alpha_n \rightarrow \beta, e_1\ :\ \alpha_1, \ldots, e_n\ :\ \alpha_n}{f(e_1, \ldots, e_n)\ :\ \beta}$$

— *abstraction* (introduction d'une fonction) :

$$\frac{x_1\ :\ \alpha_1, \ldots, x_n\ :\ \alpha_n \vdash e\ :\ \beta}{(\lambda\, x_1, \ldots, x_n\, .\, e)\ :\ \alpha_1 \times \ldots \times \alpha_n \rightarrow \beta}$$

— *typage de la construction d'un ensemble* (introduction d'un ensemble) :

$$\frac{x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash e : \beta, \qquad D : \{\mathit{vrai}, \mathit{faux}\}}{\{x_1, \ldots, x_n : D \,.\, e\} : \mathit{SET}(\beta)}$$

Comme dans la déduction naturelle, le symbole $\vdash$ indique une inférence de type : dans l'expression

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash e : \beta, D : \{\mathit{vrai}, \mathit{faux}\}$$

le type β s'exprime en fonction des types $\alpha_1, \ldots, \alpha_n$ des variables $x_1, \ldots, x_n$ qui apparaissent dans e, et le type de D est aussi à déduire de ces types des variables $x_1, \ldots, x_n$.

Par ailleurs, en toute rigueur, la virgule à droite du $\vdash$ devrait être un $\wedge$. Il ne faut la voir que comme un raccourci pour exprimer qu'il y a deux sous-preuves à fournir, celle de

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash e : \beta$$

et celle de

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash D : \{\mathit{vrai}, \mathit{faux}\}.$$

VII Correction du type d'une expression

Nous devons montrer que l'expression e a le type voulu T. Il faut prouver que $e : T$.

VII.1 Exercice

Par exemple, montrez que

$$\begin{array}{l} \mathit{append} : \mathit{Liste}(\alpha) \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha), \\ \mathit{Cons} : \alpha \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha), \\ \mathit{feuilles} : \mathit{Arbre}(\alpha) \to \mathit{Liste}(\alpha), \\ a : A, \\ l : \mathit{Liste}(A), \\ t : \mathit{Arbre}(A), \\ \vdash \mathit{append}(\mathit{Cons}(a, l), \mathit{feuilles}(t)) : \mathit{Liste}(A) \end{array}$$

ce qui prouve que l'expression $\mathit{append}(\mathit{Cons}(a, l), \mathit{feuilles}(t))$ est correctement typée et a bien le type $\mathit{Liste}(A)$.

On peut déceler des erreurs de typage, si la preuve échoue.

VII.2 Exercice

Avec les hypothèses

$$\begin{array}{l} \mathit{foo} : \alpha \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha) \\ \mathit{Cons} : \alpha \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha), \\ \mathit{append} : \mathit{Liste}(\alpha) \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha), \\ a : \alpha, \\ x : \alpha, \\ l : \mathit{Liste}(\alpha) \end{array}$$

montrer que l'expression

$$\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l))$$

n'est pas correctement typée.

VIII Inférence d'un type

On veut trouver quel est le type le plus général d'une expression e qui type e correctement. On tente de faire une preuve sans échec de $e : \alpha$ avec α variable de type.

VIII.1 Exercice

Inférer le type de l'expression

$$\lambda x, y \,.\, g(x, f(f(y))).$$

VIII.2 Exercice

Inférer le type de l'expression

$$\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l))$$

avec les hypothèses

$$\begin{array}{rcl} \mathit{Cons} & : & \alpha \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha), \\ \mathit{append} & : & \mathit{Liste}(\alpha) \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha). \end{array}$$

Le processus peut bien sûr tomber en échec quand il n'y a pas de solution, c'est-à-dire quand l'expression n'est pas typable.

VIII.3 Exercice

En inférant le type de l'expression

$$\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l))$$

avec les hypothèses

$$\begin{array}{rcl} \mathit{foo} & : & \alpha \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha) \\ \mathit{Cons} & : & \alpha \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha), \\ \mathit{append} & : & \mathit{Liste}(\alpha) \times \mathit{Liste}(\alpha) \to \mathit{Liste}(\alpha), \end{array}$$

constater que l'expression n'est pas typable, donc est incorrecte.

Que peut-on conclure en regardant le résultat des exercices ci-dessus ? Plus on apporte de précisions, plus on précise les hypothèses, plus on a de chance de constater des erreurs !

VIII.4 Exercice

Pouvez-vous inférer le type de l'expression

$$\lambda x, y \,.\, g(x, f(f(y)))$$

avec les hypothèses

$$g \,:\, \alpha \times \mathit{SET}(\alpha) \to \{\mathit{vrai}, \mathit{faux}\}$$
$$f \,:\, \mathit{SET}(\alpha) \to \mathit{SET}(\mathit{SET}(\alpha)) \;?$$

Troisième partie

Solutions des exercices du cours

Chapitre 13

Méthode de résolution

I Réfutation

Pas d'exercice dans cette partie.

II Mise en forme clausale

Pas d'exercice dans cette partie.

III Règle de résolution propositionnelle

III.1 Résolvante d'une paire de clauses

Exercice 4.1 :

1. Quelle est la différence essentielle entre cette description d'une étape de résolution et la règle (4.1) ?

   Les hypothèses de la partie III.1 sont plus restrictives que celles de la règle de résolution (4.1) : Supposons, sans perte de généralité, que C_1 soit de la forme $\neg P \vee C$ et C_2 soit de la forme $P \vee D$. Alors la règle (4.1) s'applique à C_1 et C_2. Mais la partie III.1 exige en plus qu'il n'existe pas de proposition atomique Q telle que Q appartienne à C et $\neg Q$ appartienne à D, et pas de proposition atomique R telle que $\neg R$ appartienne à C et R appartienne à D, ce qui n'est pas demandé dans la règle de résolution (4.1).

2. Cette différence ne change pas le résultat de la méthode de résolution. En effet, si une telle proposition atomique Q, alors C_1 est de la forme $\neg P \vee Q \vee G$ et C_2 est de la forme $P \vee \neg Q \vee H$. D'après la règle de résolution (4.1),

   $$\frac{C_1, C_2}{Q \vee G \vee \neg Q \vee H}.$$

   Puisque la formule $Q \vee \neg Q$ est trivialement vraie, la formule $Q \vee G \vee \neg Q \vee H$ est vraie aussi. On a produit "vrai". On peut l'ajouter à l'ensemble des clauses, mais cela n'ajoute rien puisque "vrai" est le neutre de la conjonction ($\wedge$) entre les clauses de cet ensemble.

Donc on ne modifie pas cet ensemble, autrement dit cette résolution peut être faite, mais elle ne sert à rien.

IV La méthode de résolution propositionnelle

Pas d'exercice dans cette partie.

V Résolution en logique des prédicats

V.1 Résolvante d'une paire de clauses

Exercice 4.3 : La substitution $(f(Y, g(b))/X, f(a, b)/Z, f(Y, g(b))/U)$ unifie les formules atomiques $P(f(X, g(Z)), X, f(Y, g(b)))$ et $P(f(U, g(f(a, b))), X, U)$.

Chapitre 14

Lieurs

I Les lieurs de variables

L'expression

$$(\forall x, (\exists k, x = 2 * k + 6))$$

signifie que "pour toute variable x il existe une variable k telle que $x = 2 * k + 6$".

I.1 Les séparateurs

Pas d'exercice.

I.2 Les parenthèses

Pas d'exercice.

I.3 Le domaine des variables liées

Pas d'exercice.

I.4 Variables libres et variables liées

Pas d'exercice.

I.5 Autres quantificateurs lieurs

Question : Quels symboles sont généralisés par $\forall$ et $\exists$?

Le symbole $\forall$ généralise l'opérateur $\wedge$ du *et logique* et le symbole $\exists$ généralise l'opérateur $\vee$ du *ou logique*.

Exercice : Fabriquer des expressions avec ces quantificateurs. Utiliser d'abord la notation que vous avez apprise, puis celle de ce cours.

Notation apprise	Notation de ce cours
$\Sigma_{i=1}^{i=n} i^2$	$(\Sigma\, i : 1 \leq i \leq n \;.\; i^2)$
$\bigcap_{m\in\{12,15\}}(\bigcup_{k\in\mathbb{N}}\{m*k\})$	$(\bigcap m : m \in \{12,15\}.(\bigcup k : k \in \mathbb{N}.\{m*k\}))$

II Le constructeur d'ensembles

II.1 Avantages de la notation à lieur

Si la variable y est liée, alors 4 (pour $x = 2$ et $y = 1$) et 300 (pour $x = 10$ et $y = 3$) sont des éléments de B.

Si la variable y est libre, alors $4 * y$ et $100 * y$ sont des éléments de B.

Exercice : Rétablissez les notations complètes au sujet des domaines.

On ne traite que deux exemples :

$$B_1 =_{\text{def}} \{x : (x \in \mathbb{N}) \wedge (\exists k : k \in \mathbb{N} \;.\; x = 2 * k) \;.\; x^2 * y\}$$

et

$$B_2 =_{\text{def}} \{x, y : (x \in \mathbb{N}) \wedge (y \in \mathbb{N}) \wedge (\exists k : k \in \mathbb{N} \;.\; x = 2 * k) \;.\; x^2 * y\}$$

dans le cas où y est liée.

Exercice : L'expression

$$B_3 =_{\text{def}} \{(x, y) : (\exists k : k \in \mathbb{N} \;.\; x = 2 * k)\}$$

n'obéit pas à nos conventions. Elle est supposée définir un ensemble de couples. On aurait dû écrire

$$B_3 =_{\text{def}} \{x, y : x \in \mathbb{N} \wedge y \in \mathbb{N} \wedge (\exists k : k \in \mathbb{N} \;.\; x = 2 * k) \;.\; (x, y)\}$$

Exercice : Les deux expressions suivantes définissent-elles le même ensemble ?

$$\{x : x \in \mathbb{N} \;.\; x^2\}$$

$$\{z \mid z \in \mathbb{N} \wedge z = x^2\}$$

Non, il faudrait remplacer la seconde par

$$\{z \mid z \in \mathbb{N} \wedge (\exists x \in \mathbb{N} \;.\; z = x^2)\}$$

Une dernière question : Dans les constructions d'ensemble $\{x : D.E\}$ et $\{x|D\}$, D doit être interprété comme un ensemble d'environnements (voir cours 1) du domaine d'interprétation I. Chaque environnement donne une valeur aux k variables libres de D. C'est donc un k-uplet de I^k. L'expression E est une formule ouverte, qui s'interprète donc comme une fonction de I^n dans I, où n est le nombre de variables libres de E.

Chapitre 15

Types inductifs, calculs et preuves

I Définir un type inductif

I.1 Des constructeurs

I.1.1 Exemple des listes

Question : Quelle sont les différences majeures entre une liste et un ensemble ?

Dans une liste, les éléments sont ordonnés et on peut trouver plusieurs fois le même. Dans un ensemble, les éléments ne sont pas ordonnés et sont en un seul exemplaire.

I.1.2 Exemple des arbres binaires

Exercice : Donner un exemple d'élément appartenant à chacun de ces types.

$\mathit{Cons}(1, \mathit{Nil})$ est une liste d'entiers, $\mathit{Cons}(.(5, 2), \mathit{Cons}(.(1, 2), \mathit{Nil}))$ est une liste d'arbres binaires (à feuilles de type entier).

$\mathit{Cons}(\mathit{Cons}(5, \mathit{Nil}), \mathit{Nil})$ est une liste de listes d'entiers.

$.(\mathit{Cons}(1, \mathit{Nil}), \mathit{Cons}(5, \mathit{Cons}(2, \mathit{Nil})))$ est un arbre de listes d'entiers.

Question : Est-ce que cela a un sens de parler d'un type *Arbre*(*Arbre*($\mathbb{N}$)) ?

Le type *Arbre*(*Arbre*($\mathbb{N}$)) se confond avec le type *Arbre*($\mathbb{N}$), donc on n'utilisera que le second type.

I.2 Règles d'inférence

I.2.1 Exemple des listes

Exercice : Utiliser les règles d'inférence pour prouver que l'expression $\mathit{Cons}(2, \mathit{Cons}(4, \mathit{Cons}(1, \mathit{Nil})))$ est de type $\mathit{Liste}(\mathbb{N})$.

$$\dfrac{2 : \mathbb{N} \qquad \dfrac{4 : \mathbb{N} \qquad \dfrac{1 : \mathbb{N} \qquad \mathit{Nil} : \mathit{Liste}(\mathbb{N})}{\mathit{Cons}(1, \mathit{Nil}) : \mathit{Liste}(\mathbb{N})}}{\mathit{Cons}(4, \mathit{Cons}(1, \mathit{Nil})) : \mathit{Liste}(\mathbb{N})}}{\mathit{Cons}(2, \mathit{Cons}(4, \mathit{Cons}(1, \mathit{Nil}))) : \mathit{Liste}(\mathbb{N})}$$

I.2.2 Exemple des arbres binaires

On définit l'ensemble des expressions dénotant des arbres binaires à feuilles de type α, c'est-à-dire le type *Arbre*(α), par les deux règles suivantes :

$$\frac{a : \alpha}{a : \textit{Arbre}(\alpha)}$$

$$\frac{t_1 : \textit{Arbre}(\alpha), t_2 : \textit{Arbre}(\alpha)}{.(t_1, t_2) : \textit{Arbre}(\alpha)}$$

Exercice : Commenter la signification des deux règles d'inférence et montrer que l'expression $.(3, (.(4, 5), .(7, 8)))$ spécifie un arbre binaire de type *Arbre*$(\mathbb{N})$.

La première règle déclare que tout élément de type α peut être considéré comme de type *Arbre*(α). Par conséquent, il faudra préciser son type dans tout les contextes où il pourrait être ambigu. Une autre façon de faire serait de considérer un constructeur *feuille* et la règle

$$\frac{a : \alpha}{\textit{feuille}(a) : \textit{Arbre}(\alpha)}$$

qui ôte toute ambiguïté, mais c'est plus lourd. Nous préférerons en général la version du cours.

La seconde règle déclare qu'un arbre binaire peut aussi être formé à partir d'un nœud dont les deux fils sont eux-mêmes des arbres binaires.

Une preuve que l'expression $.(3, .(.(4, 5), .(7, 8)))$ spécifie un arbre binaire de type *Arbre*$(\mathbb{N})$ est

$$\dfrac{3 : \mathbb{N}}{3 : \textit{Arbre}(\mathbb{N})} \qquad \dfrac{\dfrac{\dfrac{4 : \mathbb{N}}{4 : \textit{Arbre}(\mathbb{N})} \quad \dfrac{5 : \mathbb{N}}{5 : \textit{Arbre}(\mathbb{N})}}{.(4, 5) : \textit{Arbre}(\mathbb{N})} \quad \dfrac{\dfrac{7 : \mathbb{N}}{7 : \textit{Arbre}(\mathbb{N})} \quad \dfrac{8 : \mathbb{N}}{8 : \textit{Arbre}(\mathbb{N})}}{.(7, 8) : \textit{Arbre}(\mathbb{N})}}{.(.(4, 5), .(7, 8)) : \textit{Arbre}(\mathbb{N})}$$

$$\dfrac{3 : \textit{Arbre}(\mathbb{N}) \qquad .(.(4, 5), .(7, 8)) : \textit{Arbre}(\mathbb{N})}{.(3, .(.(4, 5), .(7, 8))) : \textit{Arbre}(\mathbb{N})}$$

Chapitre 16

Calculs sur les listes

I Les calculs simples sur une liste

I.1 Somme des éléments d'une liste d'entiers

Exercice : Spécifier les calculs du produit des éléments d'une liste d'entiers.

$$\begin{aligned} produit(\textit{Nil}) &= 1 \\ produit(\textit{Cons}(a, l)) &= a * produit(l) \end{aligned}$$

I.2 Exercice

1. Ecrire une fonction qui fait la somme des éléments d'un ensemble, sans récursion cette fois, mais en utilisant un lieur.

   $$somme(A) = \left(\sum x : x \in A \,.\, x\right)$$

   ou

   $$somme(A) = \sum_{x \in A} x$$

2. On déplie le lieur sur l'exemple $A = \{2, 4, 5\}$.

   $$\begin{aligned} & somme(\{2,4,5\} \\ = & \\ & \textstyle\sum_{x \in \{2,4,5\}} x \\ = & \\ & 2 + \textstyle\sum_{x \in \{4,5\}} x \\ = & \\ & 2 + 4 + \textstyle\sum_{x \in \{5\}} x \\ = & \\ & 2 + 4 + 5 \\ = & \\ & 11 \end{aligned}$$

3. Il y a plusieurs manières de déplier le lieur, car il n'y a pas d'ordre dans un ensemble.

I.3 Exercice

Montrer que le typage des égalités (1) et (2) est correct en prenant pour hypothèses le type de *somme*, le type de $+$ qui est $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ et celui de l'égalité des entiers $= : \mathbb{N} \times \mathbb{N} \to \{\textit{vrai}, \textit{faux}\}$.

Le type de *somme* devrait être (H_{somme}) $\textit{Liste}(\mathbb{N}) \to \mathbb{N}$.

Pour (1), sachant que $0 : \mathbb{N}$, il faut que $somme(\textit{Nil})$ puisse avoir le type entier pour que l'égalité soit bien typée, ce qui est vérifié par la déduction suivante utilisant la règle de typage de l'application :

$$\frac{H_{somme},\ \textit{Nil} : \textit{Liste}(\alpha)[\alpha := \mathbb{N}]}{somme(\textit{Nil}) : \mathbb{N}}$$

Pour (2), il faut que $somme(\textit{Cons}(a, l))$ et $a + somme(l)$ puissent avoir le type entier pour que l'égalité soit bien typée.

Le type de $a + somme(l)$ est bien $\mathbb{N}$ vérifié par la déduction suivante utilisant la règle de typage de l'application :

$$\frac{H_{+},\ a : \mathbb{N},\ \frac{H_{somme},\ l:\textit{Liste}(\mathbb{N})}{somme(l):\mathbb{N}}}{(a + somme(l)) : \mathbb{N}}$$

Maintenant, on doit utiliser comme hypothèses les types trouvés pour les variables a et l, pour vérifier le bon typage par un entier de $somme(\textit{Cons}(a, l))$.

$$\frac{H_{somme},\ \frac{H_{\textit{Cons}},\ a:\mathbb{N},l:\textit{Liste}(\mathbb{N})}{\textit{Cons}(a,l):\textit{Liste}(\mathbb{N})}}{(somme(\textit{Cons}(a, l)) : \mathbb{N}}$$

I.4 Un argument supplémentaire influence le calcul

Exercice : On déplie le calcul de

$$\begin{aligned}
&compte(4, \textit{Cons}(5, \textit{Cons}(4, \textit{Cons}(7, \textit{Nil})))) = \\
&compte(4, \textit{Cons}(4, \textit{Cons}(7, \textit{Nil}))) = \\
&1 + compte(4, \textit{Cons}(7, \textit{Nil})) = \\
&1 + compte(4, \textit{Nil}) = \\
&1 + 0 = \\
&1
\end{aligned}$$

I.5 Le résultat est de type liste

Pas d'exercice dans cette partie du cours.

I.6 Choix d'une liste pour guider le calcul

Exercice : Arrivez-vous à utiliser la deuxième liste pour guider le calcul de la juxtaposition ?.

NON, du moins pas par un calcul direct. Ceci provient du sens du calcul qui examine une liste de droite à gauche.

II Suivre deux listes en même temps

Pas d'exercice dans cette partie du cours.

III Préconditions d'un calcul

Exercice :

1. Spécifier la fonction *longueur* comme un calcul.

$$\begin{aligned} \textit{longueur}(\textit{Nil}) &= 0 \\ \textit{longueur}(\textit{Cons}(a, l)) &= 1 + \textit{longueur}(l) \end{aligned}$$

2. Spécifier comme un calcul une fonction *zip* qui prendrait deux listes de longueurs quelconques et ne mettrait en couple que les deux débuts de même longueur des deux listes.

$$\begin{aligned} zip(\textit{Nil}, \textit{Nil}) &= \textit{Nil} \\ zip(\textit{Cons}(x, l), \textit{Nil}) &= \textit{Nil} \\ zip(\textit{Nil}, \textit{Cons}(y, l)) &= \textit{Nil} \\ zip(\textit{Cons}(x, l_1), \textit{Cons}(y, l_2)) &= \textit{Cons}((x, y), zip(l_1, l_2)) \end{aligned}$$

IV Ajouter des arguments accumulateurs du résultat

Décrivons le calcul *cumuler* : $\textit{Liste}(\mathbb{N}) \times \mathbb{N} \to \mathbb{N}$ qui ajoute à un entier la somme des éléments d'une liste.

$$\begin{aligned} \textit{cumuler}(\textit{Nil}, s) &= s \\ \textit{cumuler}(\textit{Cons}(x, l), s) &= cumuler(l, x + s) \end{aligned}$$

Exercice : Déplier le calcul

$$\begin{aligned} &\textit{cumuler}(\textit{Cons}(5, \textit{Cons}(4, \textit{Nil})), 3) = \\ &\textit{cumuler}(\textit{Cons}(4, \textit{Nil}), 5 + 3) = \\ &\textit{cumuler}(\textit{Nil}, 4 + (5 + 3)) = \\ &4 + (5 + 3) = \\ &12 \end{aligned}$$

Exercice : Définir la fonction *somme* en utilisant la fonction *cumuler*

$$somme =_{\text{def}} \lambda l.\textit{cumuler}(l, 0)$$

V Divers niveaux de spécifications

1. m et A sont les variables libres. x est la variable liée.
2. Le résultat du calcul ne dépend pas de l'ordre de dépliage car l'opérateur de somme (qui correspond à Σ) est associatif.

Chapitre 17

Termes

I Les symboles fonctionnels

I.1 Exemple

Pas d'exercice dans cette partie.

I.2 Convention de nommage

Pas d'exercice dans cette partie.

I.3 Exercice

Décrire l'ensemble des symboles fonctionnels employés dans chacune des expressions suivantes :

1. $*(*(+(2,3),5),+(*(4,Moins(3)),2))$
   On a
   $$F =_{\text{def}} \{*,+,2,3,4,5,Moins\}$$
   avec $F_0 =_{\text{def}} \{2,3,4,5\}$, $F_1 =_{\text{def}} \{Moins\}$, $F_2 =_{\text{def}} \{+,*\}$ et $F_i =_{\text{def}} \emptyset$ pour $i \geq 3$.
2. $f(g(a,h(c)),g(f(a,b,c),h(a)),a)$

   $$F =_{\text{def}} \{f,g,h,a,b,c\}$$
   avec $F_0 =_{\text{def}} \{a,b,c\}$, $F_1 =_{\text{def}} \{h\}$, $F_2 =_{\text{def}} \{g\}$, $F_3 =_{\text{def}} \{f\}$ et $F_i =_{\text{def}} \emptyset$ pour $i \geq 4$.

II Les termes clos

II.1 Exercice

Montrer (prouver) que les expressions suivantes appartiennent à T_F, où F est l'un des ensembles de symboles fonctionnels trouvés dans l'exercice précédent.

La règle d'inférence utilisée dans chacune des déductions suivantes n'est pas précisée, puisque c'est toujours l'unique règle de construction de l'ensemble T_F des termes clos dont les symboles fonctionnels sont dans F, c'est-à-dire :

Pour tout entier naturel n et pour tout symbole fonctionnel f de F_n,

$$\frac{t_1, \ldots, t_n \in T_F}{f(t_1, \ldots, t_n) \in T_F}$$

1. $*(*(+(2,3),5),+(*(4,\mathit{Moins}(3)),2))$
   (a) De $2 \in F$ on déduit que $2() : T_F$
   (b) De $3 \in F$ on déduit que $3() : T_F$
   (c) De (1a), (1b) et $+ \in F_2$ on déduit que $+(2,3) : T_F$
   (d) De $5 \in F$ on déduit que $5() : T_F$
   (e) De (1c), (1d) et $* \in F_2$ on déduit que $*(+(2,3),5) : T_F$
   (f) De $4 \in F$ on déduit que $4() : T_F$
   (g) De (1b) et $\mathit{Moins} \in F_1$ on déduit que $\mathit{Moins}(3) : T_F$
   (h) De (1f), (1g) et $* \in F_2$ on déduit que $*(4, \mathit{Moins}(3)) : T_F$
   (i) De (1h), (1a) et $+ \in F_2$ on déduit que $+(*(4, \mathit{Moins}(3)), 2) : T_F$
   (j) De (1e), (1i) et $* \in F_2$ on déduit que $*(*(+(2,3),5),+(*(4,\mathit{Moins}(3)),2)) : T_F$
2. $f(g(a,h(c)),g(f(a,b,c),h(a)),a)$
   (a) De $a \in F_0$ on déduit que $a : T_F$
   (b) De $b \in F_0$ on déduit que $b : T_F$
   (c) De $c \in F_0$ on déduit que $c : T_F$
   (d) De (2c) et $h \in F_1$ on déduit que $h(c) : T_F$
   (e) De (2a) et $h \in F_1$ on déduit que $h(a) : T_F$
   (f) De (2a), (2d) et $g \in F_2$ on déduit que $g(a,h(c)) : T_F$
   (g) De (2a), (2b), (2c) et $f \in F_3$ on déduit que $f(a,b,c) : T_F$
   (h) De (2g), (2e) et $g \in F_2$ on déduit que $g(f(a,b,c),h(a)) : T_F$
   (i) De (2f), (2h), (2a) et $f \in F_3$ on déduit que $f(g(a,h(c)),g(f(a,b,c),h(a)),a) : T_F$

III Représentation arborescente des termes

III.1 Exercice

Les figures 17.1 et 17.2 sont des représentations arborescentes des deux termes de l'exercice précédent.

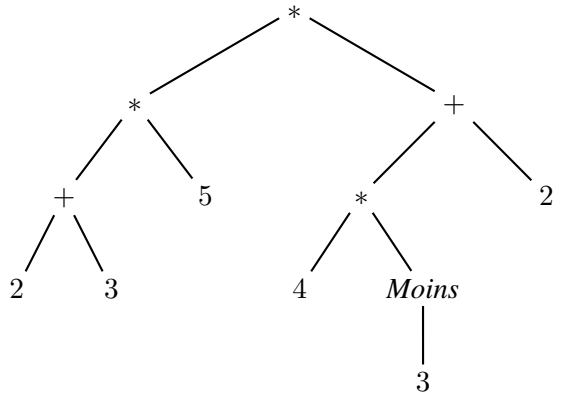

FIGURE 17.1 – Représentation arborescente du terme $*(*(+(2,3),5),+(*(4,\mathit{Moins}(3)),2))$.

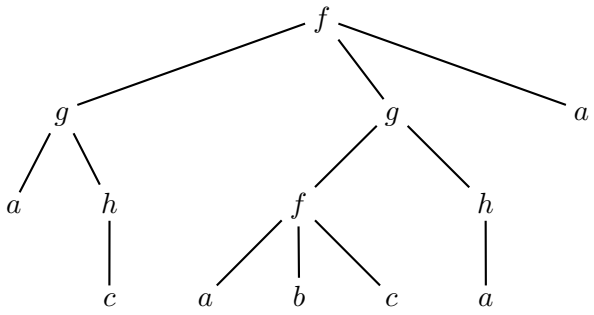

FIGURE 17.2 – Représentation arborescente du terme $f(g(a,h(c)),g(f(a,b,c),h(a)),a)$.

III.2 Questions

1. Tout terme a-t-il une représentation arborescente ?

   Oui, car en supposant que $t_1,\ldots,t_n$ ont une représentation arborescente, il suffit d'enraciner avec f pour obtenir une représentation arborescente de $f(t_1,\ldots,t_n)$. De plus, pour un terme ouvert, on peut dire qu'une variable est une arborescence.

2. Toute arborescence est-elle un terme ?

   Non. Par exemple $+(2,3,+(2,3))$ est une arborescence mais elle ne représente pas un terme car on ne peut pas attribuer d'arité à $+$.

IV Les termes avec variables

IV.1 Exercice

Avec $X = \{x, y, z\}$, décrire un ensemble des symboles fonctionnels F tel que les expressions suivantes soient dans $T_F(X)$:

1. $(x + x) * (x * (4 * (Moins(x)) + 2))$
   Avec $X = \{x, y, z\}$ (ou $X = \{x\}$), on a

   $$F =_{\mathsf{def}} \{+, *, Moins, 2, 4\}$$

   avec $F_0 =_{\mathsf{def}} \{2, 4\}$, $F_1 =_{\mathsf{def}} \{Moins\}$, $F_2 =_{\mathsf{def}} \{+, *\}$ et $F_i =_{\mathsf{def}} \emptyset$ pour $i \geq 3$.
2. $f(g(a, h(x)), y, g(f(a, z, z), h(y)))$
   Avec $X =_{\mathsf{def}} \{x, y, z\}$, on a

   $$F =_{\mathsf{def}} \{f, g, h, a\}$$

   avec $F_0 =_{\mathsf{def}} \{a\}$, $F_1 =_{\mathsf{def}} \{h\}$, $F_2 =_{\mathsf{def}} \{g\}$, $F_3 =_{\mathsf{def}} \{f\}$ et $F_i =_{\mathsf{def}} \emptyset$ pour $i \geq 4$.

IV.2 Question

— Peut-on représenter un terme ouvert de façon arborescente ?
Oui, selon la même règle que pour les termes clos, avec en plus la règle suivante.
— Quelle est la position des variables ?
Les variables sont des feuilles particulières.

V Le triangle d'un terme

V.1 Positions dans un terme

V.1.1 Question

Serait-il possible de définir le type *Pos* inductivement ?

Si l'on considère le . comme un constructeur de type inductif, on se prive de ses propriétés d'associativité et d'existence d'un élément neutre. En effet, les constructeurs sont *libres* par définition, c'est-à-dire qu'ils ne vérifient aucune propriété particulière. C'est une condition importante pour que la syntaxe garantisse seule qu'une définition inductive est complète.

En revanche, on peut profiter d'une partie des propriétés des positions en les considérant comme des listes particulières.

V.2 Domaine

V.2.1 Exercice

On a :

$$dom(x) = \{\epsilon\} \tag{17.1}$$

$$dom(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \bigcup_{i, 1 \leq i \leq n} \left(\bigcup_{u, u \in dom(t_i)} \{i.u\} \right) \tag{17.2}$$

Calcul de $dom(f(g(a,b), h(c,d,e)))$, sans détailler toutes les étapes intermédiaires :

$dom(f(g(a,b), h(c,d,e)))$

$=$

$\{\epsilon\} \cup (\bigcup u \ : \ u \in dom(g(a,b)) \ . \ \{1.u\}) \cup (\bigcup u \ : \ u \in dom(h(c,d,e)) \ . \ \{2.u\})$

$=$

$\{\epsilon\} \cup$
$(\bigcup u \ : \ u \in \{\epsilon\} \cup (\bigcup v \ : \ v \in dom(a) \ . \ \{1.v\}) \cup (\bigcup v \ : \ v \in dom(b) \ . \ \{2.v\}) \ . \ \{1.u\}) \cup$
$(\bigcup u \ : \ u \in$
$\{\epsilon\} \cup (\bigcup v \ : \ v \in dom(c) \ . \ \{1.v\}) \cup$
$(\bigcup v \ : \ v \in dom(d) \ . \ \{2.v\}) \cup (\bigcup v \ : \ v \in dom(e) \ . \ \{3.v\})$
$. \{2.u\})$

$=$

$\{\epsilon\} \cup$
$(\bigcup u \ : \ u \in \{\epsilon\} \cup \{1.\epsilon\} \cup \{2.\epsilon\} \ . \ \{1.u\}) \cup$
$(\bigcup u \ : \ u \in \{\epsilon\} \cup \{1.\epsilon\} \cup \{2.\epsilon\} \cup \{3.\epsilon\} \ . \ \{2.u\})$

$=$

$\{\epsilon\} \cup \{1.\epsilon\} \cup \{1.1.\epsilon\} \cup \{1.2.\epsilon\} \cup \{2.\epsilon\} \cup \{2.1.\epsilon\} \cup \{2.2.\epsilon\} \cup \{2.3.\epsilon\}$

$=$

$\{\epsilon, 1, 1.1, 1.2, 2, 2.1, 2.2, 2.3\}$

Chapitre 18

Vérification et inférence de types

I Introduction

I.1 Pourquoi des types ?

Pas d'exercice dans cette partie.

I.2 Importance des types pour la consistance des logiques

Pas d'exercice dans cette partie.

I.3 Importance des types pour les spécifications

Pas d'exercice dans cette partie.

Exercice :

1. Le type de l'expression $(x + 2) * y$ dépend de l'interprétation qu'on en fait : Si on interprète les variables dans un ensemble de nombres (les entiers par exemple) dans lequel le $+$ et le $*$ ont leur signification usuelle, alors cette expression est du même type que ses variables.
2. L'expression *append*(*Cons*(a, l),*feuilles*(t)) est une liste de même type que *Cons*(a, l) et que *feuilles*(t). Si a est de type α, alors *Cons*(a, l) est de type *Liste*(α). Donc il convient que t soit de type *Arbre*(α) et il en résulte que l et l'expression complète sont de type *Liste*(α).
3. Compte tenu de la réponse précédente, l'expression $\lambda\ l.$*append*(*Cons*(a, l),*feuilles*(t)) est de type *Liste*$(\alpha) \rightarrow$ *Liste*(α).
4. Ici, nous ne proposons qu'une solution correcte, sans chercher à savoir si c'est la plus générale (la question du type le plus général de l'expression $f(f(y))$ peut être abordée en exercice). Pour que l'expression $f(f(y))$ ait un sens, remarquons qu'il suffit que f soit de type $\alpha \rightarrow \alpha$ lorsque y est de type α. On peut écrire $x \in f(f(y))$ s'il existe un type β tel que $\alpha =$ *SET*(β). x est alors de type β. Par conséquent, un type possible pour $\lambda\ x, y.x \in f(f(y))$ est $\beta \times$ *SET*$(\beta) \rightarrow \{$*vrai*,*faux*$\}$.

I.4 Importance des types pour la programmation

Exercice : Les types possibles en C sont short, int, char, float, double, ... En PASCAL, il y a INTEGER, REAL, BOOLEAN, STRING, ...

II Définitions et notations

Pas d'exercice dans cette partie.

III Le type des listes

Pas d'exercice dans cette partie.

IV Le type des ensembles

IV.1 Exercice

1. La fonction $\mathcal{P}$ peut être définie pour tout ensemble x par

$$\mathcal{P}(x) = \{y \mid y \subseteq x\}.$$

   Ceci n'a de sens que si x est un ensemble, autrement dit est de type $\mathit{SET}(\alpha)$, avec α un type quelconque.
   Si x est de type $\mathit{SET}(\alpha)$ et si $y \in \mathcal{P}(x)$, alors y est de type $\mathit{SET}(\alpha)$. Le profil de la fonction $\mathcal{P}$ est donc

$$\mathcal{P} : \mathit{SET}(\alpha) \to \mathit{SET}(\mathit{SET}(\alpha))$$

2. L'opérateur d'inclusion $\subseteq$ a pour type $\mathit{SET}(\beta) \times \mathit{SET}(\beta) \to \{\mathit{vrai}, \mathit{faux}\}$, donc l'expression $x \subseteq \mathcal{P}(x)$ serait correcte si $\mathit{SET}(\alpha) = \mathit{SET}(\beta)$ et si $\mathit{SET}(\mathit{SET}(\alpha)) = \mathit{SET}(\beta)$, ce qui reviendrait à confondre α et $\mathit{SET}(\alpha)$. Cette expression n'est donc pas correcte.
3. En revanche, l'opérateur d'appartenance $\in$ a pour type $\beta \times \mathit{SET}(\beta) \to \{\mathit{vrai}, \mathit{faux}\}$ donc l'expression $x \in \mathcal{P}(x)$ est correcte en prenant $\beta = \mathit{SET}(\alpha)$.

V Les fonctions à profil générique

Pas d'exercice dans cette partie.

VI Règles de typage

Pas d'exercice dans cette partie.

VII Correction du type d'un expression

VII.1 Exercice

Démontrons formellement que

$$\begin{array}{l} \textit{append} \ : \ \textit{Liste}(\alpha) \times \textit{Liste}(\alpha) \to \textit{Liste}(\alpha), \\ \textit{Cons} \ : \ \alpha \times \textit{Liste}(\alpha) \to \textit{Liste}(\alpha), \\ \textit{feuilles} \ : \ \textit{Arbre}(\alpha) \to \textit{Liste}(\alpha), \\ a \ : \ A, \\ l \ : \ \textit{Liste}(A), \\ t \ : \ \textit{Arbre}(A), \\ \vdash \ \textit{append}(\textit{Cons}(a, l), \textit{feuilles}(t)) \ : \ \textit{Liste}(A) \end{array}$$

Toutes les déductions suivantes utilisent la règle (*application*).

1. $$\frac{\textit{feuilles} \ : \ \textit{Arbre}(A) \to \textit{Liste}(A), t \ : \ \textit{Arbre}(A)}{\textit{feuilles}(t) \ : \ \textit{Liste}(A)}$$

2. $$\frac{\textit{Cons} \ : \ A \times \textit{Liste}(A) \to \textit{Liste}(A), a \ : \ A, l \ : \ \textit{Liste}(A)}{\textit{Cons}(a, l) \ : \ \textit{Liste}(A)}$$

3. $$\frac{\begin{array}{l} \textit{append} \ : \ \textit{Liste}(A) \times \textit{Liste}(A) \to \textit{Liste}(A), \\ \textit{Cons}(a, l) \ : \ \textit{Liste}(A), \\ \textit{feuilles}(t) \ : \ \textit{Liste}(A) \end{array}}{\textit{append}(\textit{Cons}(a, l), \textit{feuilles}(t)) \ : \ \textit{Liste}(A)}$$

VII.2 Exercice

Avec les hypothèses

$$\begin{array}{ll} (H_1) & \textit{foo} \ : \ \alpha \times \textit{Liste}(\alpha) \to \textit{Liste}(\alpha), \\ (H_2) & \textit{Cons} \ : \ \alpha \times \textit{Liste}(\alpha) \to \textit{Liste}(\alpha), \\ (H_3) & \textit{append} \ : \ \textit{Liste}(\alpha) \times \textit{Liste}(\alpha) \to \textit{Liste}(\alpha), \\ (H_4) & \textit{Nil} \ : \ \textit{Liste}(\alpha), \\ (H_a) & a \ : \ \alpha, \\ (H_x) & x \ : \ \alpha, \\ (H_l) & l \ : \ \textit{Liste}(\alpha) \end{array}$$

on va montrer que l'expression

$$\textit{Cons}(\textit{append}(\textit{Cons}(a, \textit{Nil}), l), \textit{foo}(x, l))$$

n'est pas correctement typée.

On suppose que l'expression $\textit{Cons}(\textit{append}(\textit{Cons}(a, \textit{Nil}), l), \textit{foo}(x, l))$ est de type α_1 et on cherche une preuve de cette affirmation. L'expression ne sera pas correctement typée si les conditions nécessaires à son bon typage sont contradictoires.

La première condition est $\alpha_1 = \mathit{Liste}(\alpha_2)$. On a alors, selon la règle (*application*),

$$\frac{\begin{array}{l}(H_2)[\alpha := \alpha_2],\\ \mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l) : \alpha_2,\\ \mathit{foo}(x, l)) : \mathit{Liste}(\alpha_2)\end{array}}{\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l)) : \mathit{Liste}(\alpha_2)}$$

Une condition nécessaire pour $\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l) : \alpha_2$ est $\alpha_2 = \mathit{Liste}(\alpha_3)$. On a alors, selon la règle (*application*),

$$\frac{\begin{array}{l}(H_3)[\alpha := \alpha_3],\\ \mathit{Cons}(a, \mathit{Nil}) : \mathit{Liste}(\alpha_3),\\ l : \mathit{Liste}(\alpha_3)\end{array}}{\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l) : \mathit{Liste}(\alpha_3)}$$

On a aussi, toujours selon la règle (*application*),

$$\frac{\begin{array}{l}(H_1)[\alpha := \alpha_2],\\ x : \alpha_2,\\ l : \mathit{Liste}(\alpha_2)\end{array}}{\mathit{foo}(x, l) : \mathit{Liste}(\alpha_2)}$$

Si l est une variable, alors les deux conditions $l : \mathit{Liste}(\alpha_3)$ et $l : \mathit{Liste}(\alpha_2)$ impliquent $\alpha_3 = \alpha_2$, ce qui contredit $\alpha_2 = \mathit{Liste}(\alpha_3)$. L'expression n'est pas correctement typée.

En revanche, si l est une constante, alors les deux conditions $l : \mathit{Liste}(\alpha_3)$ et $l : \mathit{Liste}(\alpha_2)$ se généralisent en une seule, qui est (H_l). Cette hypothèse signifie que l est une liste quelconque (générique). En ajoutant la déduction

$$(\mathit{application})\frac{\mathit{Cons} \ : \ \alpha_3 \times \mathit{Liste}(\alpha_3) \rightarrow \mathit{Liste}(\alpha_3), \quad a \ : \ \alpha_3, \quad \mathit{Nil} \ : \ \mathit{Liste}(\alpha_3)}{\mathit{Cons}(a, \mathit{Nil}) : \mathit{Liste}(\alpha_3)}$$

on a une preuve complète que le type $\mathit{Liste}(\mathit{Liste}(\alpha_3))$ est correct pour l'expression

$$\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l)).$$

Conclusion : pour vérifier le bon typage d'une expression, on doit savoir quelles sont ses variables et ses constantes.

VIII Inférence d'un type

VIII.1 Exercice

Inférence du type de l'expression $\lambda x, y.g(x, f(f(y)))$.

La table 18.1 présente la recherche d'une preuve que $\lambda x, y.g(x, f(f(y)))$ est de type α_1, sous forme de tableau.

Numéro	Hypothèses	Déduction élémentaire	Règle ou numéro
		$(\lambda x, y\ .\ g(x, f(f(y)))) : \alpha_1$ si $x : \alpha_2, y : \alpha_3 \vdash g(x, f(f(y))) : \alpha_4$ et $\alpha_1 = \alpha_2 \times \alpha_3 \to \alpha_4$	(*abstraction*) (1) (a)
(1)	$x : \alpha_2$ $y : \alpha_3$	$g(x, f(f(y))) : \alpha_4$ si $x : \alpha_5$ et si $f(f(y)) : \alpha_6$ et si $g : \alpha_5 \times \alpha_6 \to \alpha_4$	(*application*) (2) (3) (α)
(2)	$x : \alpha_2$ $y : \alpha_3$	$x : \alpha_5$ si $\alpha_2 = \alpha_5$	 (b)
(3)	$x : \alpha_2$ $y : \alpha_3$	$f(f(y)) : \alpha_6$ si $f(y) : \alpha_7$ et si $f : \alpha_7 \to \alpha_6$	(*application*) (4) (β)
(4)	$x : \alpha_2$ $y : \alpha_3$	$f(y) : \alpha_7$ si $y : \alpha_8$ et si $f : \alpha_8 \to \alpha_7$	 (5) (γ)
(5)	$x : \alpha_2, y : \alpha_3$	$y : \alpha_8$ si $\alpha_3 = \alpha_8$	 (c)

TABLE 18.1 – Application des règles d'inférence des types génériques.

La ligne (1) est la prémisse de la règle (abstraction) quand sa conclusion est la première ligne. Les lignes numérotées avec des lettres latines, comme (a), sont des égalités entre expressions de type. Ce sont des contraintes nécessaires sur les $\alpha_{\ldots}$ pour qu'on puisse appliquer la règle. On les rassemblera en fin de recherche de preuve pour voir si elles ont une solution ou pas.

Les conditions (a), (b) et (c) sur les variables ne sont pas contradictoires. Une solution est

$$\alpha_1 = \alpha_5 \times \alpha_8 \to \alpha_4, \alpha_2 = \alpha_5, \alpha_3 = \alpha_8.$$

Les conditions (α), (β) et (γ) sur les symboles fonctionnels se généralisent en gardant, par exemple, les deux conditions (α) et (β). La condition (γ) peut être vue comme une instance (un cas particulier) de la condition (β).

Par conséquent, sous les hypothèses $g : \alpha_5 \times \alpha_6 \to \alpha_4$ et $f : \alpha_7 \to \alpha_6$, l'expression $(\lambda x, y\ .\ g(x, f(f(y))))$ a pour type $\alpha_5 \times \alpha_8 \to \alpha_4$.

On obtient une preuve de cette affirmation en transformant légèrement la table 18.1.

Prouver que c'est le type le plus général pour cette expression est un peu plus délicat, et n'est pas au programme de ce module. En revanche, on exige que l'application des règles pour le trouver ou pour prouver qu'un type est correct soit rigoureuse, détaillée, et entièrement formalisée.

VIII.2 Exercice

Inférence du type de l'expression

$$\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l))$$

avec les hypothèses

$$\begin{array}{l}\mathit{Cons} \ : \ \alpha \times \mathit{Liste}(\alpha) \rightarrow \mathit{Liste}(\alpha), \\ \mathit{append} \ : \ \mathit{Liste}(\alpha) \times \mathit{Liste}(\alpha) \rightarrow \mathit{Liste}(\alpha).\end{array}$$

La seule règle utilisée est l'application. La solution abrège certaines étapes qui ont été détaillées dans la correction précédente.

1.
$$\frac{\begin{array}{l}\mathit{Cons} \ : \ \alpha \times \mathit{Liste}(\alpha) \rightarrow \mathit{Liste}(\alpha), \\ \mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l) \ : \ \alpha, \\ \mathit{foo}(x, l) \ : \ \mathit{Liste}(\alpha)\end{array}}{\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l)) \ : \ \mathit{Liste}(\alpha)}$$

2.
$$\frac{\begin{array}{l}\mathit{append} \ : \ \mathit{Liste}(\beta) \times \mathit{Liste}(\beta) \rightarrow \mathit{Liste}(\beta), \\ \mathit{Cons}(a, \mathit{Nil}) \ : \ \mathit{Liste}(\beta), \\ l \ : \ \mathit{Liste}(\beta)\end{array}}{\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l) \ : \ \alpha}$$

si et seulement si $\alpha = \mathit{Liste}(\beta)$.

3.
$$\frac{\mathit{Cons} \ : \ \beta \times \mathit{Liste}(\beta) \rightarrow \mathit{Liste}(\beta), a \ : \ \beta, \mathit{Nil} \ : \ \mathit{Liste}(\beta)}{\mathit{Cons}(a, \mathit{Nil}) \ : \ \mathit{Liste}(\beta)}$$

4.
$$\frac{\mathit{foo} \ : \ \gamma \times \delta \rightarrow \mathit{Liste}(\alpha), x \ : \ \gamma, l \ : \ \delta}{\mathit{foo}(x, l) \ : \ \mathit{Liste}(\alpha)}$$

On admet que l est une variable. Les hypothèses sont compatibles entre elles si et seulement si $\delta = \mathit{Liste}(\beta)$ (pour l).

Il résulte de cette preuve que l'expression fournie est typable et que son type le plus général (obtenu en substituant toutes les contraintes) est

$$\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l)) \ : \ \mathit{Liste}(\mathit{Liste}(\beta))$$

lorsque $a \ : \ \beta$, $l \ : \ \mathit{Liste}(\beta)$, $x \ : \ \gamma$ et $\mathit{foo} \ : \ \gamma \times \mathit{Liste}(\beta) \rightarrow \mathit{Liste}(\mathit{Liste}(\beta))$.

VIII.3 Exercice

En inférant le type de l'expression

$$\mathit{Cons}(\mathit{append}(\mathit{Cons}(a, \mathit{Nil}), l), \mathit{foo}(x, l))$$

avec les hypothèses

$$\begin{array}{l} \textit{foo} \ : \ \alpha \times \textit{Liste}(\alpha) \rightarrow \textit{Liste}(\alpha) \\ \textit{Cons} \ : \ \alpha \times \textit{Liste}(\alpha) \rightarrow \textit{Liste}(\alpha), \\ \textit{append} \ : \ \textit{Liste}(\alpha) \times \textit{Liste}(\alpha) \rightarrow \textit{Liste}(\alpha), \end{array}$$

on constate que l'expression n'est pas typable, donc est incorrecte.

La seule règle utilisée est l'application. On reprend les 3 premières étapes de la preuve précédente. La quatrième étape devient

$$\frac{\textit{foo} \ : \ \alpha \times \textit{Liste}(\alpha) \rightarrow \textit{Liste}(\alpha), x \ : \ \alpha, l \ : \ \textit{Liste}(\alpha)}{\textit{foo}(x, l) \ : \ \textit{Liste}(\alpha)}$$

On rassemble toutes les contraintes de typage et les relations entre types :

$$l \ : \ \textit{Liste}(\beta), \alpha = \textit{Liste}(\beta), a \ : \ \beta, x \ : \ \alpha, l \ : \ \textit{Liste}(\alpha).$$

On exprime tout selon β :

$$l \ : \ \textit{Liste}(\beta), a \ : \ \beta, x \ : \ \textit{Liste}(\beta), l \ : \ \textit{Liste}(\textit{Liste}(\beta))$$

et on trouve une erreur. $l \ : \ \textit{Liste}(\beta)$ et $l \ : \ \textit{Liste}(\textit{Liste}(\beta))$ sont des hypothèses incompatibles.

VIII.4 Exercice

Voici comment inférer le type de l'expression

$$\lambda x, y.g(x, f(f(y)))$$

avec les hypothèses

$$\begin{array}{l} g \ : \ \alpha \times \textit{SET}(\alpha) \rightarrow \{\textit{vrai}, \textit{faux}\} \\ f \ : \ \textit{SET}(\alpha) \rightarrow \textit{SET}(\textit{SET}(\alpha)). \end{array}$$

L'arbre de déduction du type de l'expression $\lambda x, y.g(x, f(f(y)))$ est réduit à

$$\frac{x : \alpha_1, y : \alpha_2 \vdash g(x, f(f(y))) : \alpha_3}{\lambda x, y.g(x, f(f(y))) : \alpha_1 \times \alpha_2 \rightarrow \alpha_3} \textit{abstraction}$$

Il s'agit ensuite de trouver une preuve de

$$\frac{x : \alpha_1, y : \alpha_2}{g(x, f(f(y))) : \alpha_3}$$

1.

$$\frac{g \ : \ \alpha_1 \times \textit{SET}(\alpha_1) \rightarrow \{\textit{vrai}, \textit{faux}\}, \quad x \ : \ \alpha_1, \quad f(f(y)) \ : \ \textit{SET}(\alpha_1)}{g(x, f(f(y))) : \alpha_3} \textit{application}$$

si et seulement si $\alpha_3 = \{\textit{vrai}, \textit{faux}\}$.

2.
$$\frac{f \ : \ \mathit{SET}(\alpha_4) \rightarrow \mathit{SET}(\mathit{SET}(\alpha_4)), f(y) \ : \ \mathit{SET}(\alpha_4)}{f(f(y)) \ : \ \mathit{SET}(\alpha_1)}\mathit{application}$$

si et seulement si $\alpha_1 = \mathit{SET}(\alpha_4)$.

3.
$$\frac{f \ : \ \mathit{SET}(\alpha_5) \rightarrow \mathit{SET}(\mathit{SET}(\alpha_5)), y : \alpha_2}{f(y) \ : \ \mathit{SET}(\alpha_4)}\mathit{application}$$

si et seulement si $\alpha_2 = \mathit{SET}(\alpha_5)$ et $\alpha_4 = \mathit{SET}(\alpha_5)$.

Les relations entre types sont $\alpha_3 = \{\mathit{vrai}, \mathit{faux}\}$, $\alpha_1 = \mathit{SET}(\alpha_4)$, $\alpha_2 = \mathit{SET}(\alpha_5)$ et $\alpha_4 = \mathit{SET}(\alpha_5)$. Par conséquent, le type le plus général de l'expression $\lambda x, y.g(x, f(f(y)))$ est $\mathit{SET}(\mathit{SET}(\alpha_5)) \times \mathit{SET}(\alpha_5) \rightarrow \{\mathit{vrai}, \mathit{faux}\}$.

Quatrième partie

Annexes

Annexe A

Algèbre de Boole

I Propriétés générales

I.1 Définition

DÉFINITION A.1 (ALGÈBRE DE BOOLE). *On appelle* algèbre de Boole *la structure algébrique définie par un ensemble (non vide) $\mathcal{A}$ et trois opérations :*

— *la somme booléenne (binaire) :* $+$,
— *le produit booléen (binaire) :* $.$,
— *la négation booléenne (unaire) :* $\overline{\ }$ *(ex.* $\overline{a}$*).*

Ces opérations doivent de plus posséder les propriétés données dans le tableau A.1, pour que l'on puisse dire que le quadruplet $(\mathcal{A}, +, ., \overline{\ })$ *est une algèbre de Boole.*

Le tableau A.1 met en parallèle les mêmes propriétés, écrites en utilisant
— soit les notations générales d'une algèbre de Boole,
— soit les notations ensemblistes, définies lorsque $\mathcal{A}$ est l'ensemble $\mathcal{P}(E)$ des parties d'un ensemble E : c'est une algèbre de Boole particulière, bien connue, et qui possède des notations spécifiques.

REMARQUE A.1. Les signes opératoires utilisés sont les mêmes que ceux de l'addition et de la multiplication des réels. Cependant, ces opérations n'ont évidemment pas les mêmes propriétés, et ne portent pas sur les mêmes éléments.

I.2 Règles de calcul dans une algèbre de Boole

I.2.1 Particularités du calcul booléen

1. Les priorités habituelles sont respectées pour la somme et le produit booléen.
2. Les éléments neutres sont notés 0 et 1, par analogie avec les entiers de même symbole (ne pas oublier que ces calculs ne se déroulent pas dans $\mathbb{R}$...)
3. L'absence d'éléments symétriques pour la somme et pour le produit interdit les simplifications que l'on a l'habitude de pratiquer « sans y réfléchir » :
   — $a + b = a + c$ ne donne pas $b = c$,

Propriété	$\mathcal{P}(E)$	$\mathcal{A}$
idempotence	$A \cup A = A$ $A \cap A = A$	$a + a = a$ $a \cdot a = a$
commutativité	$A \cup B = B \cup A$ $A \cap B = B \cap A$	$a + b = b + a$ $a \cdot b = b \cdot a$
associativité	$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$	$a + (b + c) = (a + b) + c$ $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
éléments neutres	$A \cup \emptyset = A$ $A \cap E = A$	$a + 0 = a$ $a \cdot 1 = a$
absorption	$A \cup E = E$ $A \cap \emptyset = \emptyset$	$a + 1 = 1$ $a \cdot 0 = 0$
distributivités	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$a \cdot (b + c) = a \cdot b + a \cdot c$ $a + b \cdot c = (a + b) \cdot (a + c)$
involution	$E \setminus (E \setminus A) = A$	$\overline{\overline{a}} = a$
complémentation	$E \setminus \emptyset = E$ $E \setminus E = \emptyset$	$\overline{0} = 1$ $\overline{1} = 0$
partition	$A \cup (E \setminus A) = E$ $A \cap (E \setminus A) = \emptyset$	$a + \overline{a} = 1$ $a \cdot \overline{a} = 0$
« Lois de De Morgan »	$E \setminus (A \cup B) = (E \setminus A) \cap (E \setminus B)$ $E \setminus (A \cap B) = (E \setminus A) \cup (E \setminus B)$	$\overline{a + b} = \overline{a} \cdot \overline{b}$ $\overline{a \cdot b} = \overline{a} + \overline{b}$

TABLE A.1 – Propriétés d'une algèbre de Boole

— $ab = ac$ n'entraîne pas $b = c$.

En particulier, ne jamais perdre de vue que

— $a + b = 0$ n'est réalisable en algèbre de Boole que si $a = b = 0$

— $a.b = 1$ n'est réalisable en algèbre de Boole que si $a = b = 1$ ($A \cap B = E \Leftrightarrow A = E$ et $B = E$)

— $a.b = 0$ peut être réalisé avec $a \neq 0$ et $b \neq 0$ (par exemple, avec $b = \overline{a}$, mais ce n'est pas la seule solution...). On parle de « diviseurs de zéro ». (Ainsi, $A \cap B = \varnothing$ est possible sans avoir obligatoirement $A = \varnothing$ et $B = \varnothing$).

4. Il y a deux distributivités. Celle de la somme (booléenne) sur le produit (booléen) n'est pas habituelle. Par exemple, on a $(a + b)(a + c)(a + d)(a + e)(a + f) = a + bcdef$!

I.2.2 Règles de « redondance »

Dans une expression booléenne, une sous-expression est dite « redondante » lorsqu'on peut la supprimer sans changer la « valeur » de l'expression :

1. Dans une somme booléenne, tout terme absorbe ses multiples.
   Autrement dit : $a + a \cdot b = a$.

   PREUVE En effet, $a + a \cdot b = a \cdot (\overline{b} + b) + a \cdot b = a \cdot \overline{b} + a \cdot b + a \cdot b = a \cdot \overline{b} + a \cdot b$ (par idempotence) $= a \cdot (\overline{b} + b) = a$. ∎

2. Dans un produit booléen, tout facteur absorbe tout autre facteur qui le contient en tant que terme.
   Autrement dit : $a \cdot (a + b) = a$.

   PREUVE En effet, $a \cdot (a + b) = a \cdot a + a \cdot b = a + a \cdot b = a$. ∎

3. Enfin, la troisième règle de redondance s'exprime par :

$$a + \overline{a} \cdot b = a + b$$

   PREUVE $a + \overline{a} \cdot b = (a + \overline{a}) \cdot (a + b) = 1 \cdot (a + b) = a + b$. ∎

   EXEMPLE A.1. $ab + \overline{a}c + \overline{b}c = ab + (\overline{a} + \overline{b}) \cdot c = ab + \overline{ab} \cdot c = ab + c$

REMARQUE A.2. L'application de cette troisième règle peut être combinée avec celle des autres, comme par exemple dans le calcul suivant :

$$a \cdot b + \overline{a} \cdot c + b.c = a \cdot b + \overline{a} \cdot c$$

PREUVE $a \cdot b + \overline{a} \cdot c + b \cdot c = a \cdot b + \overline{a} \cdot c + (a + \overline{a}) \cdot b \cdot c = a \cdot b + \overline{a} \cdot c + a \cdot b \cdot c + \overline{a} \cdot b \cdot c$; $a \cdot b$ absorbe $a \cdot b \cdot c$ et $\overline{a} \cdot c$ absorbe $\overline{a} \cdot b \cdot c$, d'où le résultat. ∎

II Fonctions booléennes

II.1 Définitions

Soit $\mathcal{A}$ une algèbre de Boole.

DÉFINITION A.2 (FONCTION BOOLÉENNE). *On appelle* fonction booléenne de n variables *toute application de $\mathcal{A}^n$ dans $\mathcal{A}$ dont l'expression ne contient que :*

- *les symboles des opérations booléennes,*
- *des symboles de variables, de constantes,*
- *d'éventuelles parenthèses.*

EXEMPLE A.2. $f(a, b, c) = a \cdot \overline{b} + c$.

REMARQUE A.3. Si a est une variable booléenne, elle peut intervenir dans l'expression d'une fonction booléenne sous la forme a ou sous la forme $\overline{a}$, qui sont appelées les deux aspects de cette variable : affirmé et nié.

DÉFINITION A.3 (FONCTION BOOLÉENNE NULLE). *On appelle* fonction booléenne nulle *(à n variables) la fonction booléenne qui, à chaque valeur des variables, associe la valeur 0.*

Son expression est $f(x_1, x_2, \ldots, x_n) = 0$.

DÉFINITION A.4 (FONCTION RÉFÉRENTIEL). *On appelle* fonction référentiel *(à n variables) la fonction booléenne qui, à chaque valeur des variables, associe la valeur 1.*

Son expression est $f(x_1, x_2, \ldots, x_n) = 1$.