3 Théories axiomatiques

Nous avons jusqu'à présent étudié le *langage pur* du calcul des prédicats. Si nous avons dit comment, de façon générale, interpréter les symboles de fonction et de prédicat des langages, nous n'avons jamais examiné d'interprétation particulière. C'est ce que nous allons faire dans ce paragraphe et dans le suivant.

3.1 L'égalité

Soit donc, un symbole de relation binaire particulier que, pour ne pas confondre avec l'égalité intuitive nous noterons, dans un premier temps du moins, $\approx$[7]. Un langage L comprenant ce symbole sera dit *langage égalitaire*.
La sémantique nous permet de fixer le sens attendu du symbole d'égalité en donnant à tout langage égalitaire une *interprétation égalitaire*.

> Une L-structure de domaine D est une interprétation égalitaire de L si et seulement si $\| \approx \| = \{(a,a) \mid a \in D\}$[8]

On appelle l'ensemble $\{(a,a) \mid a \in D\}$ la *diagonale* du domaine D.

Il faut maintenant rendre compte de ce sens au niveau formel ce qui permettra d'exprimer et de décider l'égalité entre deux individus dans le cadre des seules preuves formelles. On définit pour cela une théorie particulière (*ie* : un ensemble de formules E) destinée à permettre l'interprétation du symbole $\approx$ comme la diagonale du domaine. C'est ce que l'on appelle une *théorie axiomatique* de l'égalité. Les axiomes de l'égalité sont les formules closes suivantes

Axiomes de l'égalité		
E1 :		$\forall x.x \approx x$
E2 :		$\forall x \forall y.(x \approx y) \Rightarrow (y \approx x)$
E3 :		$\forall x \forall y \forall z.((x \approx y) \wedge (y \approx z)) \Rightarrow (x \approx z)$
E4 :	pour tout $f \in \mathcal{F}_1$, on a	$\forall x_1..\forall x_n \forall y_1..\forall y_n.(x_1 \approx y_1 \wedge .. \wedge x_n \approx y_n) \Rightarrow f(x_1,..,x_n) \approx f(y_1,..,y_n)$
E5 :	pour tout $P \in \mathcal{P}_1$, on a	$\forall x_1..\forall x_n \forall y_1..\forall y_n.(P(x_1,..,x_n) \wedge x_1 \approx y_1 \wedge .. \wedge x_n \approx y_n) \Rightarrow P(y_1,..,y_n)$

Les trois premiers axiomes ($E1$, $E2$ et $E3$) définissent le symbole $\approx$ comme une *relation d'équivalence*; les deux derniers ($E4$ et $E5$) le définissent comme une *relation compatible* avec les autres symboles de relation et les symboles de fonction du langage.

A vec cette axiomatisation du symbole $\approx$, notre but est atteint en ce sens que

Théorème (7) : pour tout modèle $\mathcal{M}$ de E il existe un modèle $\mathcal{M}'$ de E qui soit égalitaire.

Preuve: Soit D le domaine de $\mathcal{M}$ et ε l'interprétation de $\approx$ dans $\mathcal{M}$. Par $E1$, $E2$ et $E3$, ε définie une relation d'équivalence sur D. On considère donc $D/_\varepsilon$ l'ensemble quotient de D (*ie* : l'ensemble ds classes d'équivalences de D selon ε) Si $a \in D$, notons $\overline{a}^\varepsilon$ sa classe d'équivalence, si $f \in \mathcal{F}_1$, notons $\|f|_\mathcal{M}$ son interprétation selon $\mathcal{M}$ et si $P \in \mathcal{P}_1$, notons $\|P|_\mathcal{M}$ son interprétation selon $\mathcal{M}$. On définit alors un nouveau modèle $\mathcal{M}'$ de E en prenant pour domaine $D/_\varepsilon$ et en fixant l'interprétation égalitaire :

$\| \approx \|_{\mathcal{M}'} = \{(\overline{a}^\varepsilon, \overline{a}^\varepsilon) \mid a \in D\}$;

pour tout $f \in \mathcal{F}_1$, on a pour tout $a_1,..,a_n \in D$ $\|f\|_{\mathcal{M}'}(\overline{a_1}^\varepsilon,..,\overline{a_n}^\varepsilon) = \overline{f(a_1,..,a_n)}^\varepsilon$;

pour tout $P \in |P_1$, on a pour tout $a_1,..,a_n \in D$ $\|P\|_{\mathcal{M}'} = \{(\overline{a_1}^\varepsilon,..,\overline{a_n}^\varepsilon) \mid (a_1,..,a_n) \in \|P\|_\mathcal{M}\}$

CQFD

Ce préliminaire étant donné, nous utiliserons désormais le symbole usuel $=$ pour dénoter la relation d'égalité.

[7] pour ne pas trop pousser le formalisme, nous conserverons une notation infixée

[8] Remarquons qu'il est fait ici usage des trois niveaux d'≪égalité≫: l'égalité formelle $\approx$; l'égalité *par définition* dénotée par le symbole (non formel) $=$; et enfin l'égalité de *désignation* lorsque nous assimilons deux élément du domaine en leur donnant le même nom a

Un manière équivalente d'introduire l'égalité est de paraphraser les axiomes de l'égalité sous forme de règles d'inférences. On rajoutera ainsi aux règles de la déduction naturelle :

RÉFLEXIVITÉ	SYMÉTRIE	TRANSIVITÉ
$\dfrac{}{\Gamma \vdash t = t} =_{refl}$	$\dfrac{\Gamma \vdash t = u}{\Gamma \vdash u = t} =_{sym}$	$\dfrac{\Gamma \vdash t = u \qquad \Gamma \vdash u = v}{\Gamma \vdash t = v} =_{trans}$

SUBSTITUTION
$\dfrac{\Gamma \vdash A[t/x] \qquad \Gamma \vdash t = u}{\Gamma \vdash A[u/x]} =_{subst}$

EXERCICE (4) : Montrer que $E, \Gamma \vdash A$ est dérivable dans le système de déduction naturelle donné en (2.4) si et seulement si $\Gamma \vdash A$ est dérivable dans le système augmenté des règles $=_{refl}$, $=_{sym}$, $=_{trans}$ et $=_{subst}$.

Nous noterons $t \neq u$ comme abréviation de $\neg(t = u)$.

3.2 L'arithmétique formelle

Nous allons, dans ce paragraphe formaliser une branche élémentaire (quoique non triviale) des mathématique : l'arithmétique. La théorie axiomatique que nous donnons est connue sous le nom d'*axiomes de Peano*. Comme nous l'avons fait pour , l'égalité, nous montrerons comment introduire les entiers dans le système formelle en se donnat de nouvelles règles de déductions.

Les axiomes de Peano On considère le langage égalitaire composé d'un sumbole de constante 0, d'un symbole de fonction unaire S et de deux symboles de fonctions binaires $+$ et $\times$. Plutôt que $S(t)$, nous noterons plus simplement St. La théorie P est définie comme l'ensemble des formules suivantes :

AXIOMES DE PEANO	
P1 :	$\forall n.Sn \neq 0$
P2 :	$\forall n \exists m.n \neq 0 \Rightarrow n = Sm$
P3 :	$\forall n \forall m.Sn = Sm \Rightarrow n = m$
P4 :	$\forall n.0 + n = n$
P5 :	$\forall n \forall m.Sn + m = S(n + m)$
P6 :	$\forall n.0 \times n = 0$
P7 :	$\forall n \forall m.Sn \times m = m + (n \times m)$
P8[9] :	pour toute formule A, on a $(A[0/n] \wedge \forall p.(A[p/n] \Rightarrow A[Sp/n])) \Rightarrow \forall n.A$

Les axiomes de Peano satisfont notre intention en ce sens que l'ensemble des entiers naturels $\mathbb{N}$ muni des opérations d'adition et de multiplication que nous manipulons depuis l'école primaire permettent de construire un modèle de P. On dit que la L-structure construite sur $\mathbb{N}$ est le *modèle standard* de P. Cependant, notre ensemble d'axiomes ne définit pas le seul ensemble $\mathbb{N}$: il est en effet possible de construire des modèles dits non standards de P.

Intuitivement : notons S^n0 le terme $S..S0$ où $n \in \mathbb{N}$ et S est répété n fois; on rajoute au langage un nouveau symbole de constatnte ∞ et on rajoute la théorie P l'ensemble de formules $\{\infty \neq S^n 0 \mid n \in \mathbb{N}\}$. Un théorème[10], que nous n'avons pas vu, de la théorie des modèles nous permet d'affirmer qu'il existe un modèle $\mathcal{M}$ de la théorie $P \cup \{\infty \neq S^n 0 \mid n \in \mathbb{N}\}$; $\mathcal{M}$ est en particulier un modèle de P, pourtant, son domaine contient ≪un élément de plus≫ que $\mathbb{N}$.

[10]Le *théorème de compacité* qui dit que une théorie (infinie) a un modèle si et seulement si toute partie finie de cette théorie a un modèle

EXERCICE (5) : Montrer (en n'utilisant que les axiomes de P) que $+$ et $\times$ sont associatives et commutatives, que la multiplication est distributive par rapport à l'addition, que l'addition est régulière (*ie*: $\forall n \forall m \forall p. n+p=m+p \Rightarrow n=m$) et que la multiplication est régulière pour les entiers non nuls (*ie*: $\forall n \forall m \forall p.(p \neq 0 \wedge p \times n = p \times m \Rightarrow n=m)$

L'ordre sur les entiers On peut retrouver la relation d'ordre usuelle sur les entiers en utilisant l'opération d'addition. Posons que l'expression $n \leq m$ est une *abréviation* pour la formule (à deux variables libres) $\exists p.n = m+p$. Nous utiliserons le symbole $::=$ lors de telles définitions. Ainsi, nous aurions écrit : $n \leq m \ ::= \exists p.n = m+p$.

EXERCICE (6) : Montrer que $n \leq m$ définit un ordre total sur les entiers compatible avec l'addition et la multiplication.

3.3 Des types

La théorie axiomatique de Peano permet, dans la langue formelle du calcul des prédicat, permet de parler des entiers. Mais elle ne permet de ne parler que des entiers alors que aimerions pouvoir parler aussi d'entiers relatifs, de rationels, de nombres réels (si nous sommes mathématiciens) ainsi que de booléens, de listes, d'arbres binaires, etc .. (si nous sommes informaticiens). En brefs, nous aimerions pouvoir *typer* les objets de notre discours car certaines propriétes ne seront vraies que des entiers, d'autres, que des listes; mais également, certaines fonctions ne devrons s'appliquer qu'à des types d'objets particuliers.
Un solution à ce problème est d'introduire dans le langage un symbole de prédicat unaire pour chaque type désiré et de forcer les arguments des diverses fonctions à satisfaire ces prédicats. Par exemple, pour les entiers, on introduit le symbole de prédicat unaire N et on reformule les axiomes de Peano

P1 $\forall n.\mathrm{N}n \Rightarrow Sn \neq 0$

P2 $\forall n.\mathrm{N}n \Rightarrow \exists m.\mathrm{N}m \wedge n \neq 0 \Rightarrow n = Sm$

P3 $\forall n.\mathrm{N}n \Rightarrow \forall m.\mathrm{N}m \Rightarrow Sn = Sm \Rightarrow n = m$

P4 $\forall n.\mathrm{N}n \Rightarrow 0+n=n$

P5 $\forall n.\mathrm{N}n \Rightarrow \forall m.\mathrm{N}m \Rightarrow Sn+m = S(n+m)$

P6 $\forall n.\mathrm{N}n \Rightarrow 0 \times n = 0$

P7 $\forall n.\mathrm{N}n \Rightarrow \forall m.\mathrm{N}m \Rightarrow Sn \times m = m + (n \times m)$

P8 $(A[0/n] \wedge \forall p.\mathrm{N}p \Rightarrow (A[p/n] \Rightarrow A[Sp/n])) \Rightarrow \forall n.(\mathrm{N}n \Rightarrow A)$

On rajoutera à ces axiomes quatre nouveaux que l'on peut lire comme les *règles de typage* des symboles de fonctions :

PT1 $N0$

PT2 $\forall n.\mathrm{N}n \Rightarrow \mathrm{N}Sn$

PT3 $\forall n.\mathrm{N}n \Rightarrow \forall m.\mathrm{N}m \Rightarrow \mathrm{N}(n+m)$

PT4 $\forall n.\mathrm{N}n \Rightarrow \forall m.\mathrm{N}m \Rightarrow \mathrm{N}(n \times m)$

Ces axiomes de typages expriment le fait que les fonctions désignées par le symboles sont *totales*. Nous verrons comment cette *discipline de types* pourra nous poser des problèmes.

Il est souvent d'usage d'abréger la formule $\forall n.\mathrm{N}n \Rightarrow A$ en $\forall n : N.A$ et la formule $\exists n.\mathrm{N}n \wedge A$ en $\exists n : N.A$
On peut d'ailleurs construire une théorie complète d'un calcul des prédicats *typé* qui n'utilise que de telles quantifications *restreintes*. On modifie alors un peu la construction des termes en associant à chaque symbole de fonction un *type* plutôt qu'une arité ce qui permet de ne construire que des termes *bien typés*. On procède de même pour les symboles de prédicats.

3.4 Des listes

Nous allons présenter dans ce paragraphe comment formaliser des objets que l'on considère comme appartenant au domaine de l'informatique plutôt qu'à celui des mathématiques : les listes. On peut, en effet, comme on l'a vu des entiers naturels, donner une théorie axiomatique des listes. Cette démarche n'est d'ailleurs pas complètement inconnue des informaticiens qui l'utilisent déjà pour définir ce que l'on appelle des *types abstraits*.

Soient le symbole de constante $[]$, les symboles de fonction binaire $::$ et @ que nous utiliserons en notation infixée, les symboles de fonction unaire hd et tl. Soit le symbole de prédicat unaire L. Nous proposons ci-dessous une première théorie axiomatique des listes dont nous examinerons les limites.

Théorie axiomatique des listes	
LT1 :	$\mathrm{L}[]$
LT2 :	$\forall a \forall x.\mathrm{L}x \Rightarrow \mathrm{L}(a :: x)$
LT3 :	$\forall x.\mathrm{L}x \Rightarrow \forall y.\mathrm{L}y \Rightarrow \mathrm{L}(x@y)$
L1 :	$\forall x.\mathrm{L}x \Rightarrow []@x = x$
L2 :	$\forall a \forall x.\mathrm{L}x \Rightarrow \forall y.\mathrm{L}y \Rightarrow (a :: x)@y = a :: (x@y)$
L3 :	$\forall a \forall x.\mathrm{L}x \Rightarrow hd(a :: x) = a$
L4 :	$\forall a \forall x.\mathrm{L}x \Rightarrow tl(a :: x) = x$
L5 :	pour toute formule A, on a $(A[[]/x] \wedge \forall a \forall y.(\mathrm{L}y \Rightarrow A[y/x] \Rightarrow A[(a :: y)/x])) \Rightarrow \forall x.(\mathrm{L}x \Rightarrow A)$

Exercice (7) : Construire un modèle de L (indication : le monoïde libre)

Les symboles $[]$ et $::$ jouent dans l'axiomatisation des listes un rôle particulier. Les seuls axiomes qui les décrivent sont LT1 et LT2 que nous avons appelé axiomes de typage. En ce sens, ils sont dits *libres* alors que le symbole @, en plus de son axiome de typage doit satisfaire les équation de L1 et L2. Les symboles $[]$ et $::$ sont appelés *constructeurs* du type L.

L'axiome LT2 n'impose aucune contrainte sur la variable a. On peut donc construire des listes de n'importe quoi : on a, par exemple, $\mathrm{L}(0 :: [] :: [])$. Les listes que nous avons axiomatisées ici ne sont pas des *listes paramétrées* comme le sont, par exemple les `'a list` du langage Caml pour lequel l'expression `0::[]::[]` n'est pas typable.
Du coups, nous sommes incapables de donner un axiome de typage pour le symbole hd : nous ne savons pas quel type assigner au terme $hd(x)$.

Il existe une seconde raison qui nous empêche de donner un type au symbole hd : elle n'est pas totalement définie; ou encore, il dénote une *fonction partielle* – quel pourrait bien être le sens de $hd([])$?
Cette remarque vaut également pour le symbole tl que nous n'avons pas défini pour le cas où son argument est $[]$.

Le schéma d'axiomes L5 nous donne un principe d'induction que nous qualifierons de *structurel* sur les listes. Il permet de démontrer des propriét'es universelles sur les listes comme le permet le principe d'induction sur les entiers.

Les listes paramétées Pour obtenir, dans un langage du premier ordre, des listes paramétrées, on peut avoir recours à un mécanisme analogue à celui des schémas d'axiome : on suppose disposer d'une infinité de symbole de prédicats unaires L_D, chacun devant servir à axiomatiser les listes d'objet de type D. Par exemple, en partant du seul type N des entiers, on se donnera la suite infinie de symboles $\mathrm{L}_{\mathrm{N}}, \mathrm{L}_{\mathrm{LN}}, \mathrm{L}_{\mathrm{LLN}}$, etc .. Pour parfaire notre formalisation, on restreindra D à n'être qu'un symbole de prédicats destiné à modéliser un type de données. Ceci nous conduit à distinguer de tels symboles de prédicats en nous donnant non plus un seul ensemble de symboles, mais deux : $\mathcal{P}_1$ les prédicats usuels et $\mathcal{D}_1$ l'ensemble (toujours infini) des symboles de prédicats unaires servant à typer les objets du discours. On appelera les éléments de $\mathcal{D}_1$ des *prédicats de type* ou, plus simplement des symboles de *type*
En appelant D-listes les listes paramétrées par un type $D \in \mathcal{D}_1$, on obtient ce que l'on pourrait qualifier de *schéma de théorie* axiomatique :

Théorie axiomatique des D-listes (i)	
LT1 :	$\mathrm{L}_D[]$
LT2 :	$\forall a.Da \Rightarrow \forall x.\mathrm{L}_D x \Rightarrow \mathrm{L}_D(a :: x)$
LT3 :	$\forall x.\mathrm{L}_D x \Rightarrow \forall y.\mathrm{L}_D y \Rightarrow \mathrm{L}_D(x@y)$
L1 :	$\forall x.\mathrm{L}_D x \Rightarrow []@x = x$
L2 :	$\forall a.Da \Rightarrow \forall x.\mathrm{L}_D x \Rightarrow \forall y.\mathrm{L}_D y \Rightarrow (a :: x)@y = a :: (x@y)$
L3 :	$\forall a.Da \Rightarrow \forall x.\mathrm{L}_D x \Rightarrow hd(a :: x) = a$
L4 :	$\forall a.Da \Rightarrow \forall x.\mathrm{L}_D x \Rightarrow tl(a :: x) = x$
L5 :	pour toute formule A, on a $(A[[]/x] \wedge \forall a.Da \Rightarrow \forall y.(\mathrm{L}_D y \Rightarrow A[y/x] \Rightarrow A[(a :: y)/x])) \Rightarrow \forall x.(\mathrm{L}_D x \Rightarrow A)$

Exercice (8) : Construire un modèle de D-listes (indication : le monoïde libre sur $I\!N \times \mathcal{A}$, où $\mathcal{A}$ est un alphabet infini)

Fonctions partielles Puisque nous avons su (*cf* exercice (8)) construire un modèle de la théorie des D-listes, nous avons donc, en particulier donné un sens au terme $hd([])$. Mieux, nous pouvons également construire un modéle de la théorie des D-listes à laquelle nous rajouterions l'axiome de typage

$$\forall x.\mathrm{L}_D x \Rightarrow D\, hd(x)$$

Il suffit pour cela de choisir[11] un élément quelconque $e \in \|D\|$ et de le fixer comme valeur de $\|hd([])\|$. Le mathématicien pourra se satisfaire de cette solution, mais l'informaticien, qui a pour projet de modéliser le comportement de programme, aura scrupule à autoriser l'expression $hd([])$ dont il sait que l'exécution provoquera une erreur. On peut, pour satisfaire l'informaticien envisager deux types de solutions au problème des fonctions partielles.

1. une première solution consiste à mettre en garde l'utilisateur de la fonction hd en précisant qu'il ne doit l'appliquer qu'aux listes non vides. On pose pour cela l'axiome de typage :

   $$\forall x.(\mathrm{L}_D x \wedge x \neq []) \Rightarrow D\, hd(x)$$

   On restreint ainsi le domaine attendu de hd au *sous-type* des listes non vides, charge à l'utilisateur de vérifier qu'il fait un emeploi licite des fonctions.
2. Une seconde solution est d'autoriser l'emploie sans restriction de la fonction hd, mais de signaler que le résultat peut n'être pas celui attendu (un élément de D). On pose alors

   $$\forall x.\mathrm{L}_D x \Rightarrow (D\, hd(x) \vee x = [])$$

   On peut lire cette seconde solution comme l'utilisation du mécanisme de levée d'exception.

Ces deux solutions permettent de ne pas avoir à compléter artificiellement le comportement des fonctions partielles. Elles ont pour prix d'alourdir le contrôle de type en le rendant indécidable.

Exercice (9) : Montrer que les deux solutions proposées sont logiquement équivalentes.

Des deux solutions proposées, nous retiendrons la première. La raison de ce choix est que, si nous savons prévenir, nous ne savons pas auusi bien guérir. Bien que logiquement équivalentes, nos deux solutions présupose des mécanismes de programmation dofférents. Les mécanismes d'exception ou de continuation existent bel et bien dans les langages de programmation, mais leur compréhension théorique n'est à ce jour pas suffisament aboutie pour que nous nous risquions à vous la présenter.
Nous adopterons donc la théorie L_D des D-listes suivante :

[11] On peut s'arranger pour toujours pouvoir le faire

Théorie axiomatique des D-listes (II)	
$\mathrm{L}_D\mathrm{T1}$:	$\mathrm{L}_D[\,]$
$\mathrm{L}_D\mathrm{T2}$:	$\forall a.Da \Rightarrow \forall x.\mathrm{L}_D x \Rightarrow \mathrm{L}_D(a :: x)$
$\mathrm{L}_D\mathrm{T3}$:	$\forall x.(\mathrm{L}_D x \wedge x \neq [\,]) \Rightarrow D\, hd(x)$
$\mathrm{L}_D\mathrm{T4}$:	$\forall x.(\mathrm{L}_D x \wedge x \neq [\,]) \Rightarrow \mathrm{L}_D\; tl(x)$
$\mathrm{L}_D\mathrm{T5}$:	$\forall x.\mathrm{L}_D x \Rightarrow \forall y.\mathrm{L}_D y \Rightarrow \mathrm{L}_D(x@y)$
L_D1 :	$\forall x.\mathrm{L}_D x \Rightarrow [\,]@x = x$
L_D2 :	$\forall a.Da \Rightarrow \forall x.\mathrm{L}_D x \Rightarrow \forall y.\mathrm{L}_D y \Rightarrow (a :: x)@y = a :: (x@y)$
L_D3 :	$\forall a.Da \Rightarrow \forall x.\mathrm{L}_D x \Rightarrow hd(a :: x) = a$
L_D4 :	$\forall a.Da \Rightarrow \forall x.\mathrm{L}_D x \Rightarrow tl(a :: x) = x$
L_D5 :	pour toute formule A, on a $(A[[\,]/x] \wedge \forall a.Da \Rightarrow \forall y.(\mathrm{L}_D y \Rightarrow A[y/x] \Rightarrow A[(a :: y)/x])) \Rightarrow \forall x.(\mathrm{L}_D x \Rightarrow A)$

Exercice (10) : Donner une interprétation des listes (de longueur n) comme fonctions de $[0..n-1]$ dans D.

Exercice (11) : Donner la théorie des suites (à préciser)