Cours de Lambda-calcul.*

Ch. Berline

CHAPITRE 5 : Le Système F.†‡
et le polymorphisme.

Table des matières

*DEA de Logique et Fondements de l'Informatique (2000-2002), UFR de Mathématiques, Université de Paris 7.

†Version (provisoire) du 25/02/01, révisée le 05/03/02 et 12/03/03.

‡*Remarque :* La présentation actuelle des Chapitres 4 (Système E) et 5 (Système F) les rend indépendants l'un de l'autre, ceci au prix de doublons (technique de réductibilité et certains commentaires). L'intérêt est de pouvoir en inverser l'ordre en cas de nécessité.

1 Introduction.

Le Système F, qui est un λ-calcul typé par la logique propositionnelle intuitioniste du second-ordre, est dû indépendamment à J.Y. Girard (1971) et J. Reynolds (1974). Nous présentons ici, suivant [K], la version "assignation de type" du système F (on parle aussi de *typage à la Curry*). Dans cette version les termes sont des termes du λ-calcul non typé auxquels on peut assigner une infinité de types. Dans la version forte (ou *typage à la Church*) l'ensemble des termes est plus riche et les termes portent des indications de type, et on peut leur associer un type unique. La version présentée ici admet des interprétations ensemblistes simples (par "candidats de réductibilité"), qui permettent de montrer les propriétés majeures du Système et sont bien adaptées à l'étude des rapports entre termes, vus comme programmes, et types, vus comme des spécifications de ces programmes. La version Church permet de plus des interprétations sémantiques fonctionnelles qui cernent de moins près le calcul mais ont d'autres avantages. Le Système F est une extension forte du Système S de Curry, et admet lui-même des extensions beaucoup plus forte. Il est aussi connu sous le nom de λ-calcul polymorphe et a un intérêt clair en informatique théorique (c'est d'ailleurs dans ce cadre que Reynolds l'a redécouvert). Nous expliquons pourquoi dans la dernière section.

2 Présentation du Système.

2.1 Formules, types et contextes.

L'ensemble $\mathcal{F}$ des *formules* du système F est l'ensemble des formules du calcul propositionnel du second ordre sur $\rightarrow$ et $\forall$. Ces formules sont donc construites inductivement à partir d'un ensemble dénombrable $VarProp$ de *variables propositionnelles* $X, Y, ...$ à l'aide du connecteur $\rightarrow$ et du quantificateur $\forall$. Ces formules sont donc des propositions (elles ne parlent pas d'individus). Deux formules sont dites *α-équivalentes* si elles sont identiques au renommage près de certaines variables liées.

Abbréviations usuelles. Celles qu'on a déja introduites dans le Chapitre 1 concernant les suites finies ; de plus :

$A, B \to C$ ou $A \to B \to C$ pour $A \to (B \to C)$ (contrairement à ce qui se passe pour l'application l'associativité est, en cas de défaut de parenthèses, une associativité à droite). Plus généralement :

$\bar{A} \to C$ abbrège $A_1 \to (A_2 \to (...(A_n \to C))..)$ si $l(\bar{A}) = n \neq 0$, et $\bar{A} \to C =_{def} C$ si $l(\bar{A}) = 0$.

Le système S de Curry, ou *Système simple,* est le système de typage qui correspond au calcul propositionnel du premier ordre sur $\to$. Chaque formule de S peut donc être vue comme une formule de F dans laquelle il n'y a pas de $\forall$.

Les variables propositionnelles sont aussi appelées *variables de type.*

Les *types du système* S sont tout simplement les formules de S.

Les *types* de F sont les classes de formules α-équivalentes, autrement dit les formules considérées à α-équivalence près : il est en effet facile de vérifier que $\to$ et $\forall$ opèrent en fait sur les classes. Chaque formule de S n'est α-équivalente qu'à elle-même ; on peut donc l'identifier avec sa classe, ce qui permet de considérer un type de S comme un type de F.

On définit le résultat $C[\bar{X} := \bar{A}]$ de la *substitution correcte de* $\bar{A}$ *à* $\bar{X}$ *dans* C, où C une formule de F, $\bar{A}$ une suite finie de formules et $\bar{X}$ une suite de variables distinctes de même longueur, de façon analogue à ce qui avait été fait au Chapitre 1 pour les termes du λ-calcul. Cette substitution correcte a les mêmes propriétés que celle que nous avons déja étudiée (permutation, substitutions successives, etc.). En particulier c'est une opération sur les types.

Un *contexte* est (dans ce cadre) un *ensemble fini* $\Gamma =_{def} \{x_1 : A_1, ..., x_n : A_n\}$ d'assignations de types à un ensemble fini de variables distinctes du λ-calcul pur. Cet ensemble peut être vide (cas $n = 0$). Nous emploierons la notation $\bar{x} : \bar{A}$, bien que nous ne nous intéressions pas ici à l'ordre dans lequel les variables sont énumérées. Nous désignerons par $VTlib(\Gamma)$ l'ensemble des variables propositionnelles libres dans les types de Γ. Ces variables sont dites "*déclarées dans* Γ", et x_i est déclarée de type A_i.

Γ, $x : A$ a le sens évident, et l'on suppose toujours dans ce cas que x ne figurait pas déja dans Γ; idem pour Γ, $\bar{y} : \bar{B}$.

Si $\Gamma =_{def} \{x_1 : A_1, ..., x_n : A_n\}$ et $\bar{X}$ est une suite de variables propositionnelles et $\bar{B}$ est une suite de formules, alors :

$\Gamma[\bar{X} := \bar{B}] =_{def} \{x_1 : A_1[\bar{X} := \bar{B}], ..., x_n : A_n[\bar{X} := \bar{B}]\}$

2.2 Les règles de typage.

Un séquent est une expression de la forme $\Gamma \vdash\ t : A$, où Γ est un contexte, A est un type, et t est un terme du λ-calcul non typé ; il se lit : "t est typable de type A dans le contexte Γ". Les règles de typages suivantes permettent de dériver les séquents de droites en supposant qu'on ait déja dérivé les séquents qui figurent à gauche ; il s'agit bien sûr de shémas de règles.

• axiome				$\Gamma, x : A \vdash_F x : A$
• $\rightarrow$ -intro	Si	$\Gamma, x : A \vdash_F t : B$	alors	$\Gamma \vdash_F \lambda x.t : A \rightarrow B$
• $\rightarrow$ -élim	Si	$\left.\begin{array}{c} \Gamma \vdash_F t : A \rightarrow B \\ \Gamma \vdash_F u : A \end{array}\right\}$	alors	$\Gamma \vdash_F (t)u : B$
• $\forall$-intro	Si	$\left\{\begin{array}{c} \Gamma \vdash_F t : A \\ X \notin VTlib(\Gamma) \end{array}\right\}$	alors	$\Gamma \vdash_F t : \forall X.A$
• $\forall$-élim	Si	$\Gamma \vdash_F t : \forall X.A$	alors	$\Gamma \vdash_F t : A[X := B]$

On appelle *dérivation du séquent* $\Gamma \vdash_F \ t : A$ toute suite finie d'instances de ces règles, dont tout séquent "hypothèse" figure comme conclusion d'un élément précédent de la suite. En particulier le premier élément d'une telle suite est nécessairement un axiome. Le dernier séquent "conclusion" est alors appellé *séquent conclusion de la dérivation.* Un séquent $\Gamma \vdash_F \ t : A$ est *dérivable* si il existe une dérivation dont il est la conclusion. Une description simplifiée et souvent suffisante consiste à définir une dérivation comme une suite de séquents telle que tout séquent qui n'est pas un axiome est obtenu à partir de certains des précédents en employant une instance d'une des règles ci-dessus (le nom de la règle étant précisé ; il s'agit donc en fait d'une suite de couples de la forme (nom de règle, séquent).

Un terme $t \in \Lambda$ est dit *typable dans* F si il figure dans un séquent dérivable. On emploiera aussi l'expression de *séquent dérivable dans F.* On utilisera les abbréviations $\bar{\Gamma} \vdash_F \ \bar{t} : \bar{A}$ où les trois suites finies sont de même longueur, pour signifier que $\Gamma_i \vdash_F \ t_i : A_i$ pour chaque i, et on utilisera $\Gamma \vdash_F \ \bar{t} : \bar{A}$ si tous les Γ_i sont égaux à Γ.

Les trois premières règles sont les règles de typage pour le système S ; on restreint alors évidemment ces schémas aux types et contextes qui ne contiennent pas de $\forall$, et on change le symbole $\vdash_F$ en un $\vdash_S$.

Exemple 1 $\perp =_{def} \forall X.X$, $ID =_{def} \forall X.(X \rightarrow X)$,
$B =_{def} \forall X.(X, X \rightarrow X)$ *et* $N =_{def} \forall X.((X \rightarrow X), X \rightarrow X)$
$\lambda x.x$ est typable dans F de type ID, les booléens sont typables de type B, et enfin les entiers de Church sont typables de type N.

Exemple 2 *Les termes suc $=_{def} \lambda\nu.\lambda f.\lambda x.\nu f(f)x$ et $\lambda\nu\lambda f\, \lambda x.f(\nu f)x$ sont typables dans F, de type $N \rightarrow N$. Les termes proposés au Chapitre 3 pour l'addition, la multiplication et l'exponentiation sont typables de type $N^2 \rightarrow N$, c'est-à-dire $N, N \rightarrow N$.*

Commentaires.

1. Ceux d'entre vous qui suivent le cours de théorie de la démonstration observeront qu'il ne s'agit pas ici d'un calcul des séquents à la Gentzen. Ce dont il s'agit ici c'est d'un codage des preuves de la logique propositionnelle intuitioniste du second ordre *écrites en déduction naturelle* par des termes du λ-calcul, avec la différence qu'au lieu de déduire la formule A on déduit le séquent

$\Gamma \vdash t : A$, le rôle de Γ étant de rappeler les hypothèses utilisées, et celui de t d'être le code de la démonstration de A considérée.

2. Il s'agit ici de "dérivations linéaires", qui correspond à une écriture linéaire des preuves, qui sont par essence de nature arborescente. Les dérivations arborescentes et les dérivations linéaires ne sont pas des objets de même nature : par exemple si l'on part d'une dérivation arborescente et si on veut la linéariser on est obligé d'introduire un certain arbitraire dans l'ordre d'écriture des règles, de plus on n'est pas obligé de répéter un même séquent, même si il se trouvait dérivé à plusieurs endroits de l'arbre. Les dérivations linéaires sont beaucoup plus simples à traiter typographiquement, et en général suffisantes pour permettre des preuves rigoureuses, dans le cadre du λ-calcul typé. Nous verrons toutefois une exception.

3. Il s'agit de plus ici d'un codage non injectif des preuves : les étapes de la démonstration qui concernent l'introduction ou l'élimination du $\forall$ ne sont tout simplement pas codées. On dit que cette version de F ne donne pas de contenu algorithmique au $\forall$.

4. Pour coder intégralement ces preuves il faudrait enrichir le λ-calcul de départ en ajoutant au niveau des termes les variables propositionnelles et deux nouveaux opérateurs d'application et d'abstraction (permettant respectivement d'abstraire sur les types et d'appliquer un type à un terme) ; on aurait alors des termes de la forme $\lambda X.t$ à droite de la règle $\forall$-intro, et $\{t\}\, B$ à droite de la règle $\forall$-élim.

Remarque 3 *1. Tout sous-terme d'un terme typable dans F l'est aussi.*

2. Tout terme typable dans F l'est d'une infinité de manière.

3. S est un sous-système de F : si A et tous les types de Γ sont dans S, alors $\Gamma \vdash_S t : A \Longrightarrow \Gamma \vdash_F t : A$.

Remarque 4 *xx est typable dans F mais pas dans S.*

Les deux lemmes suivants se prouvent avec une induction triviale sur le nombre de règles utilisées dans une dérivation de $\Gamma \vdash_F t : A$.

Lemme 5 *Si $\Gamma \vdash_F t : A$, alors toutes les variables libres de t sont déclarées dans Γ.*

Lemme 6 *$\Gamma \vdash_F t : A$ implique $\Gamma' \vdash_F t : A$,*
où Γ' est n'importequel contexte contenant la restriction de Γ à $Vlib(t)$.
C'est donc en particulier vrai pour toute extension de Γ.

Corollaire 7 *Tout terme clos typable est typable dans le contexte vide.*

La remarque suivante est laissée en exercice.

Remarque 8 *Un terme clos t est typable dans F ssi $\vdash_F 0t : X \to X$ ssi $\vdash_F 0t : \forall X.(X \to X)$, où 0 est le booléen $\lambda x.\lambda y.y$.*

Lemme 9 *Tout terme normal t est typable dans F dans le contexte $\bar{x} : \bar{\bot}$, où $\bar{x} = Vlib(t)$.*

Preuve. Facile par induction sur t. **A faire** (exercice).

Exercice 10 *Vérifier que le terme $Z =_{def} \lambda\nu.((\nu\, \lambda d.F)V)$ du Chapitre 3 (cf. définition par cas) est de type $N \to B$.*

3 Premières propriétés syntaxiques.

3.1 Les règles dérivées.

Lemme 11 $\Gamma \vdash_F t : A$ *implique* $\Gamma[\bar{X} := \bar{B}] \vdash_F t : A[\bar{X} := \bar{B}]$
De plus toute dérivation du premier séquent se mime en une dérivation de même longueur du second, où la substitution est opérée sur tous les séquents intermédiaires.

Preuve. Comme d'habitude induction sur la longueur des dérivations de séquent. On regarde la dernière règle utilisée pour prouver $\Gamma \vdash_F t : A$. Si c'est une des trois premières règles c'est trivial.

Si c'est la règle 4 on a $A = \forall Y.A'$ avec $Y \notin VTLib(\Gamma)$. On renomme éventuellement Y de manière que de plus $Y \notin VTLib(\bar{B})$, et on utilise les propriétés de la substitution correcte, en plus de l'hypothèse d'induction, pour conclure.

Si c'est la règle 5 on a $A = A'[Y := C]$. On renomme éventuellement Y pour que $Y \notin VTLib(\bar{B} \cup \Gamma)$, et on utilise les propriétés de la substitution correcte, en plus de l'hypothèse d'induction.

Lemme 12 $\Gamma \vdash_F t : \forall\bar{X}.A$ *implique* $\Gamma \vdash_F t : A[\bar{X} := \bar{B}]$

Preuve. Applications successives de la règle $\forall$-élim, en utilisant le lemme sur les substitutions correctes successives. Prendre le soin de renommer d'abord $\bar{X}$ à l'extérieur des variables libres de $\bar{B}$.

Lemme 13 *(adéquation syntaxique).*
Si $\Gamma, \bar{x} := \bar{A} \vdash_F t : B$ *et* $\Gamma \vdash_F \bar{u} : \bar{A}$ *alors* $\Gamma \vdash_F t[\bar{x} := \bar{u}] : B$

Preuve. **A Faire.**
Induction facile sur la longueur d'une dérivation de $\Gamma, \bar{x} := \bar{A} \vdash_F t : B$.

3.2 Typabilité.

La proposition suivante a un intérêt pratique certain.

Proposition 14 *Si $\Gamma \vdash_F t : A$, alors il existe une dérivation de ce séquent qui est $\forall$-normale, au sens suivant : une règle $\forall$-intro n'y est jamais suivie d'une règle $\forall$-élim.*

Preuve. L'idée est la suivante : on considère une dérivation de longueur minimale du séquent, soit d. Si d n'est pas $\forall$-normale elle contient une configuration :

(k) $\Gamma_1 \vdash_F t_1 : A_1$

(k+1) $\Gamma_1 \vdash_F t_1 : \forall X.A_1$ avec $X \notin VTlib(\Gamma_1)$

(k+2) $\Gamma_1 \vdash_F t_1 : A_1[X := B]$

où les nombres de gauche sont des nombres d'étapes. Il suit clairement du Lemme 11, et du fait que $X \notin \Gamma_1$, qu'on pouvait obtenir $\Gamma_1 \vdash_F t_1 : A_1[X := B]$ en k étapes au lieu de k+2.

Sous cette forme la démonstration est insuffisante car il n'est pas assuré que l'objet résultant de la substitution est encore une démonstration de $\Gamma \vdash_F t : A$ (des séquents intermédiaires et maintenant modifiés ou supprimés étaient peut-être utilisés par la suite). La présentation correcte consisterait à travailler avec les dérivations arborescentes et de faire la substitution ci-dessus uniquement dans le sous-arbre intéressé ; on obtient bien alors une dérivation de $\Gamma \vdash_F t : A$ et on a bien fait décroître le nombre de règles utilisées (dans l'arbre complet de dérivation).

Théorème 15 *Si* $\Gamma \vdash_F t : A$ *et si* $t \to_\beta t'$ *alors* $\Gamma \vdash_F t' : A$.

Preuve. Il suffit de travailler sur les couples t, t' tels que $t \to_{\beta 0} t'$. La preuve est alors par induction sur la dérivation du séquent, en regardant quelle est la dernière règle utilisée.

Si c'est un axiome c'est évident car il n'y a pas de t'.

Si c'est une règle $\forall$ on applique trivialement l'hypothèse d'induction.

Si c'est la règle $\to$-intro on s'y ramène facilement..

Le seul cas non trivial est celui de la règle $\to$-élim. Ce cas, où t est de la forme $t = uv$, se subdivise en deux cas. Si t n'est pas un redex ou si c'est un redex mais qui n'est pas réduit dans la β_0-réduction, alors une utilisation triviale de l'hypothèse d'induction suffit.

Le pas délicat est celui où $t = (\lambda x.u')v$ et $t' = u'[x := v]$.

A l'étape précédente on avait $\Gamma \vdash_F v : B$ et $\Gamma \vdash_F \lambda x.u' : B \to A$.

Si on pouvait en déduire $\Gamma, x : B \vdash_F u' : A$, on concluerait directement avec le Lemme 13 (adéquation syntaxique) ; le problème c'est que la dernière règle utilisée pour obtenir $\Gamma \vdash_F \lambda x.u' : B \to A$ pouvait être aussi bien une instance de $\forall$-élim que de $\to$-intro (les autres cas sont éliminés par la forme du terme ou du type).

On regarde la première fois où, dans le dérivation, on a obtenu

$\Gamma \vdash_F \lambda x.u' : \forall \bar{Y}.C$ avec :

C ne commence pas par $\forall$, les composantes de $\bar{Y}$ ne sont pas libres dans Γ, $l(\bar{Y}) \geqq 0$, et enfin $B \to A \equiv C[\bar{Y} := \bar{U}]$ pour un certain $\bar{U}$. Cette fois il est facile de voir, en éliminant les autres cas, que ce séquent-là a été obtenu en conclusion d'une règle $\to$-intro. Donc nécessairement $l(\bar{Y}) = 0$, la formule C est de la forme $B' \to A'$ avec $A \equiv A'[\bar{Y} := \bar{U}]$ et $B \equiv B'[\bar{Y} := \bar{U}]$. On déduit du Lemme 11 que $\Gamma'[\bar{Y} := \bar{U}], x : B'[\bar{Y} := \bar{U}] \vdash_F u' : A'\ [\bar{Y} := \bar{U}]$, et, puisque $\bar{Y}$ n'est pas libre dans Γ, que $\Gamma, x : B \vdash_F u' : A$. On peut alors conclure avec le lemme d'adéquation syntaxique que $\Gamma \vdash_F u'[x := v]$.

Remarque 16 *Le théorème précédent est faux pour la β-équivalence. Nous verrons en effet que tout terme typable dans le système F est fortement normalisable ; un terme normalisable non fortement n'est donc pas typable dans F, or sa forme normale l'est (Lemme 9). L'exemple suivant est plus raffiné puisqu'il montre un terme fortement normalisable, typable dans F, mais qui a strictement moins de types que sa forme normale.*

Exemple : $t =_{def} \lambda x.x$ et $t' =_{def} \lambda x.(\lambda d.x)(x)x$. On a $\vdash_F t : X \to X$ mais le séquent analogue n'est pas dérivable pour t'.

4 Réductibilité.

Il s'agit, ici, d'interprétation des formules de F par des sous-ensembles de Λ, de telle manière qu'un terme (disons clos) de type A soit dans l'interprétation de A (les termes du λ-calcul seront donc interprétés par eux-même). Le premier type d'applications de telles interprétations est la preuve des divers théorèmes de normalisation, mais nous en verrons d'autres. La méthode est connue sous le nom de *réductibilité* et a été en premier utilisée par Tait, et est un avatar simple des méthodes de *réalisabilité* (qui remontent à Kleene). Son adaptation au Système F (Girard 1971) passe par le choix adéquat d'un sous-ensemble $\mathcal{S}$ de $P(\Lambda)$ comme domaine du quantificateur $\forall$ (choix qui va dépendre de la question à résoudre) ; cette façon de faire permet de contourner les problèmes liés à l'imprédicativité de F (i.e. au fait que toute formule $\forall X.A$ "parle" d'elle-même).

$\mathcal{S}$ peut aussi être vu comme l'ensemble des valeurs de vérité possibles pour les variables propositionnelles. Le cas $\mathcal{S} = \{\emptyset, \Lambda\}$ correspondant à une logique 2-valuée (vrai, faux). Nous ferons apparaître ceci explicitement quand nous traiterons le Système AF_2.

La méthode consiste à interpréter les types par des parties de Λ de telle façon qu'un terme soit dans l'interprétation de son type, ce qui va nous forcer à ne considérer que des parties saturées, en un sens à préciser. L'interprétation se fait par induction sur la longueur des formules.

Dans tout le reste du chapitre les lettres U, W, N, S désignent des parties quelconques de Λ, et V dénote toujours l'ensemble des variables de Λ.

Les Sections 4.1 et 4.2 sont communes avec le Chapitre 4.

4.1 Interprétation de l'implication.

On commence par définir l'opérateur binaire sur $P(\Lambda)$ qui interprétera la flèche ; on le note par $\Rightarrow$.

Définition 17 $U \Rightarrow W \;=_{def} \{t \in \Lambda \;/\; \forall u \in U \; (t)u \in W\}$. *Cette flèche est clairement croissante en W et décroissante en U (pour l'inclusion).*

Pour tous U, W on a : $\emptyset \Rightarrow W = \Lambda$ et $U \Rightarrow \Lambda = \Lambda$. Par ailleurs on a $U \Rightarrow \emptyset = \emptyset$ pour tout $U \neq \emptyset$ (et $\emptyset \Rightarrow \emptyset = \Lambda$).

On dit aussi que la flèche ensembliste est *contravariante* par rapport à son premier argument et *covariante* par rapport au second.[1]

4.2 Saturation et auto-saturation.

La notion suivante est cruciale.

Définition 18 *W est* S-saturée, *$S \subseteq \Lambda$ si*

$$\forall u, \bar{t} \in \Lambda^{<\omega}\ \forall v \in S\ \ (u[v/x]\bar{t} \in W \Rightarrow (\lambda x.u)v\bar{t} \in W)$$

On dit que W est saturée *si elle est Λ-saturée.*
On dira que W est auto-saturée *si elle est W-saturée.*

L'ensemble des parties S-saturées est clairement clos par intersection quelconque (y compris l'intersection vide) et il est clos par $\Rightarrow$, en fait $U \Rightarrow W$ est S-saturé dès que W l'est. Par ailleurs si W est S-saturée et si $S' \subseteq S$ alors W est clairement S'-saturée, en particulier tout ensemble saturé est auto-saturé.

Exemple 19 *Tout ensemble clos par β-expansion de tête[2], et a fortiori tout ensemble clos par β-expansion ou par β-équivalence, est saturé ; en particulier les ensembles $\emptyset, \Lambda, N_g$, et N_h sont saturés, où N_g et N_h sont respectivement l'ensemble des termes normalisables à gauche et de tête. Ils sont donc aussi auto-saturés.*

Lemme 20 *L'ensemble N_f des termes fortement normalisables n'est pas saturé mais il est auto-saturé.*

Preuve. Supposons que $T \equiv u[x := v]\bar{t}$ est fortement normalisable ; ceci implique que u et tous les t_i le sont. Supposons de plus que v est fortement normalisable et qu'il y a une réduction infinie σ de $(\lambda x.u)v\bar{t}$; puisque u, v et $\bar{t}$ sont fortement normalisables σ est forcée de réduire un jour le rédex de tête : il existe des C-réduits $u', v', \bar{t}'$ de $u, v, \bar{t}$ tels que σ se factorise en : $(\lambda x.u)v\bar{t} \to_\beta (\lambda x.u')v'\bar{t}' \to_{\beta_0} u'[x := v']\bar{t}' \to_{\beta_0} \ldots$. Or une suite de réductions partant de $u'[x := v']\bar{t}'$ ne peut pas être infinie, sinon, comme $u[x := v]\bar{t} \to_\beta u'[x := v']\bar{t}'$, il y aurait une suite infinie de réductions partant de T ; contradiction.

Définition 21 *On dira que $\mathcal{S} \subseteq P(\Lambda)$ est* auto-saturé [3] *si pour toutes $S, S' \in \mathcal{S}$ on a que S est S'-saturé.*

[1] Cette interprétation correspond à l'interprétation (informelle) intuitioniste de l'implication par Heyting, à savoir : une preuve de $A \to B$ est une "méthode" (disons, nous, un algorithme) permettant de construire une preuve de B à partir de toute preuve de A.

[2] $U \subseteq \Lambda$ est *clos par β-expansion (resp. β-expansion de tête)* si $t \in U$ et $t' \to_\beta t$ (resp. $t' \to_h t$) impliquent $t' \in U$.

[3] L'introduction de cette notion est intéressante car elle permet de montrer un théorème d'adéquation très général, dont toutes les variantes qui sont nécessaires dans les preuves de normalisation, que ce soit en λ-calcul pur ou dans les systèmes typés.

Il est clair que tout sous-ensemble d'un ensemble auto-saturé est auto-saturé, que si $\mathcal{S}$ possède un élément maximum $S_{\max}$ il suffit de demander que tout S de $\mathcal{S}$ soit $S_{\max}$-saturé. On peut noter qu'un ensemble $\mathcal{S}$ auto-saturé ne contient que des parties auto-saturées de Λ, mais qu'un ensemble de parties auto-saturées n'est évidemment pas forcément auto-saturé. On remarquera enfin que $\mathcal{S} =_{def} \{N\}$ est autosaturé si et seulement N est auto-saturé.

Exemple 22 *Les ensembles $\mathcal{S}$ suivants sont auto-saturés et tels que $\Lambda, \emptyset \in \mathcal{S}$.*
$\{\emptyset, \Lambda\}$
$P^{sat}(\Lambda)$, *l'ensemble de toutes les parties saturées de Λ,*
et tous ses sous-ensembles ; en particulier :
$P^{\beta}(\Lambda)$, *l'ensemble des parties de Λ closes par β-équivalence*
$P^{\exp\beta}(\Lambda)$, *l'ensemble des parties de Λ closes par β-expansion.*
$P^{\beta\eta}(\Lambda)$, *l'ensemble des parties de Λ closes par $\beta\eta$-équivalence.*

Notation 23 *$P(N/U)$ dénote l'ensemble des parties de N qui contiennent U, pour tous $N, U \subseteq \Lambda$.*

Exemple 24 *Les ensembles $\mathcal{S}$ suivants sont auto-saturés et tels que $N \in \mathcal{S}$.*
1. Pour tout $N \subseteq \Lambda$ saturé, et pour tout $U \subseteq \Lambda$:
$P^{sat}(N/U) =_{def} P(N/U) \cap P^{sat}(\Lambda)$ *(ex : $N = N_g$ ou N_h, $U = \emptyset$).*
2. Pour tout $N \subseteq \Lambda$ qui est auto-saturé, l'ensemble
$P^{N sat}(N/U)$ *des parties N-saturées de N qui contiennent U (ex : $N = N_f$).*

4.3 Des environnements aux interprétations.

Définition 25 *Soit $\mathcal{S} \subseteq P(\Lambda)$. Une* interprétation I des variables propositionnelles de F dans $\mathcal{S}$ *(on parle aussi d'*environnement à valeur dans $\mathcal{S}$) *est une application I de $VarProp$ dans $\mathcal{S}$.*

Définition 26 *Etant données $S \in \mathcal{S}$ et $X \in VarProp$ on définit : $I[X := S]$ comme étant l'environnement I' qui prend la même valeur que I sur $VarProp$ sauf sur X, où $I'(X) = S$. On définit de manière analogue $I[\bar{X} := \bar{S}]$.*

Noter que pour tous $X \neq Y$ et $S, S' \in \mathcal{S}$ on a $I[X := S][Y := S'] = I[X := S, Y := S']$ (il ne peut pas y avoir de capture de variable dans un tel cadre sémantique).

Etant donné une interprétation I des variables propositionelles on souhaite évidemment l'étendre en une interprétation de toutes les formules du système, et ce par induction sur la complexité (ou longueur) des formules. Ceci est directement possible pour le système S et pour les systèmes d'intersection de type (cf. Chapitre Système E), mais pas pour F à cause de l'imprédicativité. En effet, dans notre contexte on ne pourrait guère imaginer que de définir $I(\forall X.A)$ comme $\cap \{ I(A[X := B]) \ / \ B \in \mathcal{F} \}$ ou $\cap \{ I[X := I(B)](A) \ / \ B \in \mathcal{F} \}$, or les deux sont incorrects en tant que définition par induction (c'est évident puisque B est de complexité arbitraire, mais il est intéressant de regarder le cas où $A \equiv X$ et $B \equiv \forall X.X$). La solution c'est de conduire la définition simultanément pour toutes les interprétations I qui sont à valeur dans S.

Définition 27 *$\mathcal{S}$ étant fixé, on définit simultanément la valeur $I(A)$ de A pour tous les environnements I à valeurs dans $\mathcal{S}$. Le cas où A est une variable propositionnelle est déja réglé. On pose :*
$I(A \to B) =_{def} I(A) \Rightarrow I(B)$
$I(\forall X.A) =_{def} \cap\{\, I[X := S](A) \;/\; S \in \mathcal{S}\}$.

Remarque 28 *Si $\mathcal{S}$ est clos par $\Rightarrow$ et par intersection quelconque (finie ou infinie), alors pour toute formule A et pour toute I à valeur dans $\mathcal{S}$ on a $I(A) \in \mathcal{S}$.*

Remarque 29 *$I(A)$ ne dépend que de A, de $\mathcal{S}$, et de la valeur de I sur les variables libres de A ; en particulier, si A est close son interprétation est indépendante du choix de I, et ne dépend que de A et $\mathcal{S}$.*

Lemme 30 *Pour tous I, A, B, X, on a : $I\,(A[X := B]) = I[X := I(B)]\,(A)$.*

Preuve. Induction sur la complexité de A.

Le lemme suivant permet de construire des $\mathcal{S} \subseteq P(N/V)$ qui sont clos par intersection quelconque et par $\Rightarrow$.

Définition 31 *Une partie N_0 de Λ est* bien adaptée *à $N \in P(\Lambda)$ si :*

$$\begin{array}{ll} 1) & V \subseteq N^0 \subseteq N \\ 2) & (N^0 \Rightarrow N) \subseteq N \\ 3) & N^0 \subseteq (N \Rightarrow N^0) \end{array}$$

on dira aussi que le couple (N, N^0) est adapté.

Lemme 32 *Si N^0 est bien adapté à N, alors l'ensemble $P(N/N^0)$ est clos par $\Rightarrow$ et par intersection quelconque ; de plus $P(N/N^0) \subseteq P(N/V)$.*

Preuve. La clôture par $\Rightarrow$ suit trivialement de l'adaptation de N^0 et des propriétés de variance et covariance de $\Rightarrow$, le reste est trivial.

4.4 Sémantiques ensemblistes adéquates.

Définition 33 *J'appellerai* ensemble adéquat *(sous-entendu :* de candidats de réductibilité) pour *F tout $\mathcal{S} \subseteq P(\Lambda)$ qui est clos par $\Rightarrow$ et par intersection quelconque (finie ou infinie) et qui est auto-saturé.*

Lemme 34 *(adéquation). Soit $\mathcal{S} \subseteq P(\Lambda)$ un ensemble adéquat pour F et I une interprétation à valeurs dans $\mathcal{S}$. Alors :*
1. Pour tout type A de F on a $I(A) \in \mathcal{S}$.
2. Pour tout t, et tous $\bar{x}, \bar{u}, \bar{A}, B$ on a : $\left.\begin{array}{c} \bar{x} : \bar{A} \vdash t : B \\ \bar{u} \in I(\bar{A}) \end{array}\right\} \Longrightarrow t[\bar{x} := \bar{u}] \in I(B)$
3. Si t est clos ou si $\forall i\; x_i \in I(A_i)$ on a $\bar{x} : \bar{A} \vdash t : B \Longrightarrow t \in I(B)$

Preuve.

1. Suit des propriétés de clôture de $\mathcal{S}$ et du fait que I est à valeurs dans $\mathcal{S}$.

3. Suit immédiatement de 2.

2. On raisonne par induction sur la longueur de la dérivation du séquent, et, à chaque étape, simultanément pour toutes les I à valeurs dans $\mathcal{S}$ et $\bar{u} \in I(\bar{B})$. On regarde la dernière règle utilisée.

Si c'est un axiome, alors $t \equiv x_i \in \bar{x}$ et $B = A_i$, et on a $t[\bar{x} := \bar{u}] = u_i \in I(B)$.

Si c'est $\rightarrow$-élim, alors $t \equiv vw$ et il existe C tel que $\bar{x} : \bar{A} \vdash_I v : C \rightarrow B$ et $\bar{x} : \bar{A} \vdash_I w : C$. Par hypothèse d'induction $v[\bar{x} := \bar{u}] \in I(C \rightarrow B) = I(C) \Rightarrow I(B)$ et $w[\bar{x} := \bar{u}] \in I(C)$. Il en résulte que $(vw)[\bar{x} := \bar{u}] \equiv v[\bar{x} := \bar{u}]w[\bar{x} := \bar{u}] \in I(B)$.

Si c'est $\rightarrow$-intro, alors $t \equiv \lambda y.v$ avec $y \notin \bar{x}$ et B est de la forme $B' \rightarrow C$, et enfin $\bar{x} : \bar{A}, y : B' \vdash_I v : C$, donc, par hypothèse d'induction, pour tout $w \in I(B')$ on a $v[\bar{x} := \bar{u}, y := w] \in I(C)$. Maintenant nous supposons de plus que y n'est pas libre dans $\bar{u}$; alors $v[\bar{x} := \bar{u}, y := w] = v[\bar{x} := \bar{u}][y := w]$. Comme $I(C)$ est $I(B')$-saturée (auto-saturation de $\mathcal{S}$) on a donc $(\lambda y.v[\bar{x} := \bar{u}])w \in I(C)$. Comme ceci a lieu pour tout $w \in I(B')$, on a bien $t \in (I(B') \Rightarrow I(C)) = I(B' \rightarrow C) \equiv I(B)$.

Si c'est $\forall$-intro. Alors $B \equiv \forall Y.B'$, avec Y non libre dans $\bar{A}$. On veut montrer que $t[\bar{x} := \bar{u}] \in I(B) = \cap\{\, I[Y := S](B') \;/\; S \in \mathcal{S}\}$. Il suffit d'appliquer l'hypothèse d'induction à toutes les interprétations de la forme $I[Y := S]$, où $S \in \mathcal{S}$, ce qui est possible car, Y n'étant pas libre dans $\bar{A}$, on a que $\bar{u} \in I[Y := S](\bar{A}) = I(\bar{A})$.

Si c'est $\forall$-élim. Alors $B \equiv B'[Y := C]$ et à l'étape précédente on avait $\bar{x} : \bar{A} \vdash t : \forall Y.B'$ avec Y non libre dans $\bar{A}$. L'hypothèse d'induction, appliquée cette fois avec la même I, donne $t[\bar{x} := \bar{u}] \in I(\forall Y.B')$. Comme $I(\forall Y.B') \subseteq I[Y := I(C)](B') = I(B'[Y := C])$ (Lemme 30), on a bien ce qu'on voulait.

Remarque 35 *Ce lemme concerne les systèmes S et F. La partie de ce lemme qui ne concerne que S ne nécessite pas la clôture par intersection. Par ailleurs ce lemme est vrai pour beaucoup de systèmes et sa démonstration en est très modulaire. Par exemple pour le Système E, qui est l'extension du système S la plus simple possible qui puisse nous aider à montrer le théorème de normalisation de tête du λ-calcul pur, il suffit de varier légèrement la définition de l'adéquation.*

Tous les exemples d'ensembles auto-saturés que nous avons donnés dans la Section 4.2 sont trivialement clos par intersection quelconque. Ceux qui sont adéquats pour F sont donc exactement ceux qui sont en plus clos par $\Rightarrow$.

Exemple 36 (ensembles $\mathcal{S}$ adéquats).
1. Tous les $\mathcal{S}$ de l'Example 22 sont clairement adéquats, et contiennent Λ et $\emptyset$.
2. Pour $U = \emptyset$ les $\mathcal{S}$ de l'Example 24 ne sont adéquats que si $N = \Lambda$, en effet dans ce cas $\emptyset \in \mathcal{S}$; or $(\emptyset \Rightarrow N) = \Lambda$.
3. On peut noter enfin que les $\mathcal{S}$ de la forme $\{N\}$ ne sont en général pas clos pour $\Rightarrow$: par exemple pour $N =_{def} N_h, N_g$ ou N_f on a $\delta \equiv \lambda x.xx \in N$ (car δ est normal), mais $\delta\delta \notin N$; donc $\delta \notin N \Rightarrow N$.

Ces exemples ne permettent donc pas d'avoir de $\mathcal{S}$ adéquat tel que $\mathcal{S} \subseteq P(N)$, pour $N \neq \Lambda$. Pour réaliser ceci on fera appel aux couples adaptés. En effet :

Lemme 37 *Si N est auto-saturé et si N^0 est bien adapté à N, alors $P^{Nsat}(N/N^0) =_{def} P(N/N^0) \cap P^{Nsat}(\Lambda)$ est adéquat pour F.*

Preuve. C'est une conséquence immédiate du Lemme 32 et du fait que $P^{Nsat}(\Lambda)$ est auto-saturé.

5 Applications de la sémantique.

5.1 La logique sous-jacente à F.

La réductibilité permet de montrer de façon très simple et élégante des propriétés de la logique sous-jacente à F.

Le premier lemme exprime la consistance logique de F.

Lemme 38 *Il n'existe pas de terme clos t tel que $\vdash_F t : \bot$, où $\bot =_{def} \forall X.X$*

Preuve. Soit I une interprétation quelconque à valeurs dans un $\mathcal{S}$ adéquat tel que $\emptyset \in \mathcal{S}$, par exemple $\mathcal{S} \equiv \{\emptyset, \Lambda\}$. Si il existait un tel t il serait nécessairement clos et, compte tenu du Lemme 34 on aurait $t \in I(\forall X.X)$, or $I(\forall X.X) = \cap\{\, S \in \mathcal{S}\} = \emptyset$.

La logique sous-jacente au système F est une logique intuitioniste. En particulier on ne peut y dériver ni la règle de raisonnement par l'absurde ni celle correspondant au tiers exclu (il en existe plusieurs variantes). L'exercice suivant permet d'en donner une preuve sémantique simple, sachant qu'en logique intuitioniste propositionnelle du second ordre on peut définir tous les connecteurs à partir de $\rightarrow$ et $\forall$, et en particulier : $non(X)$ par $X \rightarrow \bot$, et $A \vee B$ par $\forall X((A \rightarrow X),(B \rightarrow X) \rightarrow X)$ (cf. Définition 47, et les justifications qui la suivent).

Exercice 39 *Montrer (par réductibilité) qu'il n'existe pas de terme clos de type $A =_{def} \forall X\,(((X \rightarrow \bot) \rightarrow \bot) \rightarrow X)$, ni de type :*
$A' =_{def} \forall X \forall Y(((X \rightarrow Y) \rightarrow Y) \rightarrow X)$ ou
$T =_{def} \forall X \forall Y((X \rightarrow Y),((X \rightarrow \bot) \rightarrow Y) \rightarrow Y)$.
(A pour "absurde" et T pour "tiers-exclus").

5.2 Le Théorème de normalisation forte.

Compte tenu du Lemme 34, il suffit, pour démontrer le Théorème de normalisation forte de mettre au point une interpétation telle que pour toute A on ait $V \subseteq I(A) \subseteq N_f$.

Lemme 40 *L'ensemble $N_f^0 =_{def} \{x\bar{t} \;/\; x \in V,\ \bar{t} \in N_f^{<\omega}\}$ est bien adapté à N_f.*

Démonstration. Tout est trivial ; pour voir que $N_f^0 \to N_f \subseteq N_f$ il suffit de remarquer que tout terme t tel que tx est fortement normalisable, pour une variable x quelconque, est lui même fortement normalisable, ce qui est évident.

Corollaire 41 *L'ensemble $\mathcal{S}_f =_{def} P^{N_f Sat}(N_f/N_f^0)$ est adéquat.*

Théorème 42 *Tout terme typable dans F est fortement normalisable.*

Démonstration. On fixe une interprétation I à valeurs dans $\mathcal{S}_f$ (par exemple en décidant que pour toute variable propositionnelle X on a $I(X) =_{def} N_f$). Par les lemmes 40 et 32 on aura $I(A) \in \mathcal{S}_f$, donc $V \subseteq I(A) \subseteq N_f$, et on conclut avec le lemme d'adéquation (Lemme 34).

5.3 La correction des types de données.

Nous montrons ici que les types courants que nous avons introduits (N, B, ID), ainsi qu'un certain nombre d'autres, ne contiennent pas de termes indésirables. Nous commenterons ces résultats dans la dernière section de ce chapitre. Ici il n'est nul besoin de couples adaptés.

5.3.1 Les types de données simples.

On rappelle que $ID =_{def} \forall X(X \to X)$, $B =_{def} \forall X.(X, X \to X)$, et $N =_{def} \forall X.((X \to X), X \to X)$.

Proposition 43 *Soit $t \in \Lambda$ un terme clos, alors.*
1. $\vdash_F t : ID$ implique $t \to_\beta \lambda x.x$
2. $\vdash_F t : B$ implique $t \to_\beta \lambda x \lambda y.x$ ou $t \to_\beta \lambda x.\lambda y.y$
3. $\vdash_F t : N$ implique qu'il existe $n \in N$ tel que $t \to_\beta \hat{n}$ ou $t \to_\beta \lambda f.f$.

Preuve. Dans les trois cas on va utiliser des interprétations à valeurs dans l'ensemble adéquat $\mathcal{S} =_{def} P^{sat}(\Lambda)$, puis appliquer le lemme d'adéquation. En fait il suffira que $\mathcal{S}$ soit adéquat et contienne les ensembles S_0 définis ci-dessous, en particulier $P^{\beta}(\Lambda)$ et $P^{\exp\beta}(\Lambda)$ fonctionneraient tout aussi bien.

1. Soit $S_0 =_{def} \{ t \in \Lambda \,/\, t \to_\beta x \}$, où $x \in V$ est une variable fixée.

Si $\vdash_F t : ID$ alors $t \in I(ID)$ pour toute $\mathcal{S}$-interprétation I. En particulier $t \in S_0 \Rightarrow S_0$. Comme $x \in S_0$ on a nécessairement $tx \in S_0$, donc $tx \to_\beta x$. Nécessairement il existe u tel que $t \to_\beta \lambda x.u$ et $u \to_\beta x$; on en déduit que $t \to_\beta \lambda x.x$.

2. Idem avec $S_0 =_{def} \{ t \in \Lambda \,/\, t \to_\beta x \text{ ou } t \to_\beta y \}$, où x et y sont des variables distinctes fixées.

Si $\vdash_F t : B$ alors $t \in I(B)$ pour toute $\mathcal{S}$-interprétation I. En particulier $t \in S_0 \Rightarrow (S_0 \Rightarrow S_0)$. Comme $x, y \in S_0$ on a nécessairement $txy \in S_0$, donc $txy \to_\beta x$ ou y. Donc $t \to_\beta \lambda x.u$ avec $uy \to_\beta x$ ou y ; on en déduit que $u \to_\beta \lambda y.v$ avec $v \to_\beta x$ ou y, donc $t \to_\beta \lambda x \lambda y.x$ ou $\lambda x.\lambda y.y$. Cqfd.

3. Idem avec $S_0 =_{def} \{ t \in \Lambda \,/\, \exists n \geq 0 \; t \to_\beta f^n x\}$ où f et x sont des variables distinctes fixées.

Si $\vdash_F t : N$ alors $t \in I(N)$ pour toute $\mathcal{S}$-interprétation I. En particulier $t \in (S_0 \Rightarrow S_0) \Rightarrow (S_0 \Rightarrow S_0)$. Clairement $x \in S_0$ et $f \in (S_0 \Rightarrow S_0)$, donc $tfx \in S_0$, soit $tfx \rightarrow_\beta f^n x$ pour un n. Comme le second membre est soit x soit de la forme fw, w normal, ceci n'est possible que si il existe u tel que $t \rightarrow_\beta \lambda f.u$ et $ux \rightarrow_\beta f^n x$. A ce stade il faut distinguer deux cas : ou il existe v tel que $u \rightarrow_\beta \lambda x.v$ et $v \rightarrow f^n x$, ou $u \rightarrow_\beta f$ et $n = 1$. Dans le premier cas $t \rightarrow_\beta \lambda f.\lambda x.f^n x$, et dans le second $t \rightarrow_\beta \lambda f.f$.

5.3.2 Les types de données composés.

Pour pouvoir parler avec suffisamment de généralités de composition de types de données il vaut mieux disposer d'une définition abstraite de ces types. La définition suivante se prête bien à l'usage que nous allons en faire ; il s'agit d'une définition sémantique, et relative à un ensemble adéquat $\mathcal{S}$. Les $\mathcal{S}$ qui sont les plus pertinents pour ce genre de problèmes sont ceux de l'Example 22 : $P^{sat}(\Lambda)$, $P^{\beta}(\Lambda)$ et $P^{\exp\beta}(\Lambda)$ etc., mais d'autres fonctionneraient tout aussi bien.

Définition 44 [4] *Etant donné un ensemble adéquat $\mathcal{S}$, on dit qu'un type clos A est* un type de données relativement à $\mathcal{S}$ *si pour toute interprétation I à valeurs dans $\mathcal{S}$ on a que $I(A)$ n'est pas vide et que si $t \in I(A)$ alors il existe u (normal clos) tel que $t \rightarrow_\beta u$ et $\vdash_F u : A$.*

Rappelons que l'interprétation d'un type clos A ne dépend que de $\mathcal{S}$ et de A, et pas de l'interprétation spécifique I à valeurs dans $\mathcal{S}$.

Notons que si il existe $u \in \Lambda$ tel que $t \rightarrow_\beta u$ et $\vdash_F u : A$ alors u est clos et de plus sa forme normale u', qui existe puisque tout terme typable dans F est normalisable, satisfait encore $t \rightarrow_\beta u'$ et $\vdash_F u' : A$; il n'est donc pas nécessaire de préciser "normal clos". Mais il est bon de l'avoir en tête.

Exemple 45 *Les types ID, B, N sont des types de données relativement à tout $\mathcal{S}$ qui convient à la preuve de la Proposition 43, puisque cette preuve vaut non seulement pour les termes du bon type, mais plus généralement pour tous les termes situés dans l'interprétation du type.*

Exemple 46 *Le type $B \to B$ n'est pas un type de données. En effet le terme $t = \lambda b.((b\,\lambda x.0)\lambda x.0)x$, où 0 est le booléen "Faux", est tel que $t \in I(B) \Rightarrow I(B) = I(B \to B)$, pour toutes les interprétations I pertinentes ; toutefois t ne se réduit pas à un terme clos.*

Définition 47 *Soient A et B deux types on définit :*
$A \wedge B =_{def} \forall X((A, B \to X) \to X)$
$A \vee B =_{def} \forall X((A \to X), (B \to X) \to X)$
$L[A] =_{def} \forall X((A, X \to X), X \to X)$

[4] Cette définition est voisine de celle du livre de Krivine, mais elle est a priori plus restrictive (cerne de plus près la syntaxe), et permet d'accélerer un petit peu les preuves.

Lemme 48 *1. Si* $\Gamma \vdash_F a : A$ *et* $b : B$*, alors* $\Gamma \vdash_F \lambda f.fab : A \wedge B$
2. Si $\Gamma \vdash_F a : A$ *alors* $\Gamma \vdash_F \lambda f.\lambda g.fa : A \vee B$ *et*
si $\Gamma \vdash_F b : B$*, alors* $\Gamma \vdash_F \lambda f.\lambda g.gb : A \vee B$.
3. Si $\Gamma \vdash_F a_1 : A ... \Gamma \vdash_F a_n : A$*, alors* $\Gamma \vdash \lambda f.\lambda x.(fa_1)...(fa_n)x : L[A]$,
et $\Gamma \vdash \lambda f.fa_1 : L[A]$

Nous nous intéressons dans cette section aux deux premières formules en tant que types de données. Toutefois ce sont aussi les interprétations respectives de la conjonction et de la disjonction à partir de $\to$ et $\forall$, en logique propositionnelle du second ordre. On remarquera en effet que :
1. Le terme $\lambda a.\lambda b.\lambda f.fab : A, B \to A \wedge B$ permet d'associer aux (codes de deux) preuves de A et B un (code de) preuve de $A \wedge B$. Les termes $\lambda F.F1$ et $\lambda F.F0$ associent à toute preuve de $A \wedge B$ respectivement une preuve de A et une preuve de B.
2. Le terme $\lambda a.\lambda f.\lambda g.fa : A \to A \vee B$ permet d'associer à toute preuve de A une preuve de $A \vee B$, et de même pour B. Par ailleurs le terme $\lambda f.\lambda g.\lambda F.Ffg$: $\forall X\, ((A \to X), (B \to X), (A \vee B) \to X)$ permet d'associer à toute paire de preuves de X sous hypothèse A et sous hypothèse B respectivement, une preuve de X sous hypothèse $A \vee B$.

Du point de vue des types de données elles représentent respectivement le produit et la somme disjointe de deux types de données A et B. Le troisième type est, lui, le type des listes finies d'éléments de type A. Ceci est essentiellement justifié par la proposition suivante

Proposition 49 *Si A et B sont deux types de données relativement à un ensemble adéquat pertinent $\mathcal{S}$, alors $A \wedge B$, $A \vee B$, $L[A]$ sont des types de données relativement à $\mathcal{S}$. Plus précisément : si I est une interprétation quelconque à valeurs dans $\mathcal{S}$ on a :*
1. Si $t \in I(A \wedge B)$ alors il existe a, b clos normaux tels que $t \to_\beta \lambda f.fab$ et $\vdash_F a : A$ et $\vdash_F b : B$.
2. Si $t \in I(A \vee B)$ alors : soit il existe a clos normal tel que $t \to_\beta \lambda f.\lambda g.fa$ et $\vdash_F a : A$, soit il existe b clos normal tel que $t \to_h \lambda f.\lambda g.gb$ et $\vdash_F b : B$.
3. Si $t \in I(L[A])$ alors il existe $n \geqslant 0$ et $a_1, ..., a_n$ clos normaux tels que $t \to_\beta \lambda f.\lambda x.(fa_1)...(fa_n)x$ ou $t \to_\beta \lambda f.fa_1$ et $\vdash_F a_1 : A ...\ \vdash_F a_n : A$.

Preuve. Dans les trois cas on travaille avec une interprétation I à valeurs dans $\mathcal{S}$ et on applique le lemme d'adéquation.

1. On fixe $f \in V$ et on pose :

$S_0 =_{def} \{\, t \in \Lambda \,/\, \exists a, b$ normaux tels que $t \to_\beta fab$ et $\vdash_F a : A$ et $\vdash_F b : B.\}$.

On notera que a et b sont nécessairement clos, puique typables dans le contexte vide.

Clairement :

$I(A \wedge B) \subseteq J((A, B \to X) \to X)$ pour $J =_{def} I[X := S_0]$, donc :

$I(A \wedge B) \subseteq (J(A), J(B) \Rightarrow S_0) \Rightarrow S_0$. Montrons que $f \in J(A), J(B) \Rightarrow S_0$. Comme les éléments de $J(A)$, $J(B)$ se β-réduisent à des termes clos normaux de type A et B respectivement, et comme S_0 est clos par β-expansion, il suffit de vérifier que pour tous les termes clos normaux a de type A et b de type B on a $fab \in S_0$, ce qui est évident. Il en résulte que $tf \in S_0$. Donc il existe $a : A$ et $b : B$ tels que $tf \to_\beta fab$. Comme b, qui est clos, ne peut pas être la variable f, on conclut qu'il existe u tel que $t \to_\beta \lambda f.u$ et $u \to_\beta fab$, donc $t \to_\beta \lambda f.fab$.

2. On fixe $f, g \in V$ et on pose :

$S_0 =_{def} \{\, t \in \Lambda \,/\, \exists a$ normal $(\, t \to_\beta fa$ et $\vdash_F a : A\,)$ ou $\exists b$ normal $(t \to_\beta gb$ et $\vdash_F b : B.)\,\}$.

3. On fixe $f, x \in V$ et on pose :

$S_0 =_{def} \{\, t \in \Lambda \,/\, \exists n \geqslant 0\, \exists a_1, ..., a_n$ normaux $t \to_h (fa_1)...(fa_n)x$ et $\vdash_F a_1 : A, ..., \vdash_F a_n : A)\,\}$.

Puis on raisonne comme précédemment.

En plus d'être adéquat, tout ce qu'on demande à $\mathcal{S}$ pour être pertinent c'est de contenir les ensembles S_0 qui interviennent dans la preuve (et ceci héréditairement) ; là encore $P^{sat}(\Lambda)$, $P^{\beta}(\Lambda)$ et $P^{\exp\beta}(\Lambda)$ conviennent.

On pourra noter que $L[Y]$ n'est pas un type de données (il n'est pas clos).

6 Intérêt et limites du Système F.

6.1 Le paradigme de la programmation par preuves.

A ECRIRE (en attendant voir par exemple mes notes sur la Correspondance de Curry-Howard, et/ou certaines des références qui sont dedans).

6.2 Expressivité logique de F et intérêt pour l'informatique.

6.2.1 Expression des autres connecteurs et quantificateurs.

Nous avons déja rencontré $\bot, \wedge, \vee$ et *non*.

On définit $\exists Y.A$ comme étant $\forall X\, [\forall Y (A \to X) \to X]$.

6.2.2 Expression et correction globale des données.

Les possibilités d'expression logique de F (c'est à dire au niveau des types) sont assez importantes. Il est en effet possible de définir dans la logique de F (et non en rajoutant à chaque fois des constantes et des règles ad-hoc) tous les types de données courant (nous avons vu déja le cas des booléens et des entiers), et même de plusieurs façons différentes (par exemple correspondant à des définitions inductives différentes de l'ensemble des entiers comme algèbre). Ces types seront "corrects" au sens suivant : $\vdash_F t : N$ implique que le calcul complet de t donne nécessairement (le représentant d') un entier, et l'analogue est vrai pour tous les autres types de données.

6.2.3 Le polymorphisme.

Au niveau programmation l'intérêt de F c'est le "polymorphisme", c'est à dire le fait de pouvoir représenter des types "génériques" tels le type $L[-]$, qui a pour instance $L[B]$, $L[N]$, etc..qui représentent les types de listes finies de booléens ou d'entiers, et de pouvoir typer les actions "génériques". Nous verrons par exemple qu'il existe un terme clos *inverse* du λ-calcul pur, qui est typable dans F de type $\forall X.(L[X] \to L[X])$ et qui renverse toutes les listes finies, qu'elles soient listes de booléens, listes d'entier, ou listes de "données" plus compliquées. Il existe aussi un autre terme, appelons le *map*, qui est typable de type $\forall X.((X \to X) \to (L[X] \to L[X]))$ tel que pour tout terme ϕ tel que $\vdash_F \phi : N \to N$ et pour tout terme l tel que $\vdash_F l : L[N]$ (ce qui force l à se β-réduire à une liste $< \bar{n}_1, ..., \bar{n}_n >$, par le théorème de correction des données), on a que $(map)\phi l$ se β-réduit à $< \phi\bar{n}_1, ..., \phi\bar{n}_n >$. De plus le même terme *map* va convenir pour les listes de booléens et plus généralement pour toutes les listes correspondant aux autres types de données.

En résumé : possibilité de créer des types de données composés, de manière uniforme, et de travailler avec de manière aussi uniforme que possible.

6.2.4 L'imprédicativité.

C'est l'inconvénient conceptuel lié à une utilisation pleine du $\forall$. L'imprédicativité rend nettement plus compliquées les modélisations du système (surtout fonctionnelles) [par rapport aux systèmes prédicatifs] mais n'empêche pas de travailler avec le système.

6.3 Calculabilité.

6.3.1 Terminaison des calculs.

On est assuré que tous les calculs faits dans F terminent (Théorème de forte normalisation).

6.3.2 Fonctions et algorithmes représentables dans le système F.

Puisque le système est fortement normalisant toute fonction représentable dans F est une fonction totale, et donc récursive totale.

Les fonctions représentables dans le système F, par des termes de type $N^n \to N$, pour un certain n, sont les fonctions récursives n-aires *prouvablement totales dans l'arithmétique du second ordre* (Girard 1971), comme nous le montrerons dans le Chapitre 5 (AF2). Cette classe contient normalement toutes les fonctions récursives totales qu'on peut jamais avoir envie de programmer. Ce résultat est à comparer avec celui de Schwichtenberg et Statman (1975) qui montre que la classe des fonctions représentables dans le système S, par un terme de type $((X \to X), X \to X)^n \to ((X \to X), X \to X)$, est celle des "polynômes étendus". Cette classe de fonctions est définie comme la plus petite classe qui contient les projections, les fonctions constantes, la fonction sg définie par $sg(0) = 0$ et $sg(n+1) = 1$, et qui est close par $+$ et $\times$.

Le terme de Maurey, qui calcule l'inf de deux entiers m et n en un temps $O(\inf(n, m))$ dans la représentation de Church, est normal, et donc typable dans F. Mais on peut montrer qu'il n'est pas typable de type $N^2 \to N$ dans

F [Krivine]. En conséquence, bien que la fonction inf soit primitive récursive et prouvablement totale, et donc représentable dans F par un terme de type $N^2 \to N$, elle ne sera pas représentée par le terme de Maurey. Plus généralement, si il est vrai qu'on a finalement pû trouver dans F un terme qui calculait l'inf de m, n en $O(\inf(m,n)\log(\inf(m,n)))$ (R. David), la question de savoir si il en existe un qui calcule en un temps $O(\inf(n,m))$ est encore ouverte. Il est normal d'avoir moins d'algorithmes représentés dans F que dans le λ-calcul pur (à représentation fixée des entiers), mais du point de vue de la programmation c'est un inconvénient de ne pas pouvoir calculer des fonctions de base avec des algorithmes efficaces.

6.4 L'indécidabilité de F.

Le second revers d'un usage plein du $\forall$ c'est l'indécidabilité. En effet, les questions naturelles suivantes sont indécidables (alors qu'elles sont décidables pour S) ; il s'agit évidemment de trouver un algorithme uniforme qui donne la réponse correcte pour toutes les données).

1. "Le problème de la vérification de type" (en anglais : "type-checking") :
"Etant donnés Γ, A, t le séquent $\Gamma \vdash_F t : A$" est-il dérivable ?"
2. Une instance particulière :
"Etant donné t clos le séquent $\vdash_F t : ID$ est-il dérivable ?"
(où $ID =_{def} \forall X\,(X \to X)$ est le plus simple des types "habités").
3. "Le problème de la typabilité" :
"le terme t est-il typable dans F?" même question avec t clos.

Ces questions sont évidement semi-décidables, puisqu'on peut énumérer les séquents dérivables, les termes correspondants, etc. Nous avons vu dans la Remarque 8 que le problème de la typabilité se réduisait facilement à celui de la vérification de type. En fait les deux problèmes sont indécidables (J.B. Wells 1996-98)[Wel], mais le problème est resté ouvert très longtemps et la preuve est difficile et très technique. Le système S, lui, est décidable, ainsi qu'un certain nombre de systèmes intermédiaires.

L'indécidabilité est une raison pour laquelle, en pratique, on a beaucoup plus utilisé des langages typés où l'usage du $\forall$ est très restreint mais où les problèmes ci-dessus sont décidables (ML, $CAML$, et variations).

6.5 Vers un plus grand pouvoir d'expression.

D'un autre côté il est intéressant pour beaucoup de raisons (expressivité au niveau logique, correction mathématique des programmes, possibilité d'avoir des algorithmes plus efficients, etc.) de s'intéresser à des systèmes encore plus riches que F. Nous étudierons bientôt le système $AF2$, dont la logique est la calcul des prédicats du second ordre. Il existe des extensions de $AF2$ avec des possibilités de programmer par point fixe, ce qui permet de récupérer des algorithmes plus efficaces, et aussi de travailler avec des types de données "infinitaires" tels les streams (listes infinies d'entiers, de caractères, etc.). Enfin il existe des extensions de F ou $AF2$ basées sur des logiques d'ordre bien supérieur (cf. [Barendregt]). Ces systèmes, tels le Calcul des Constructions de Coquand et Huet, sont principalement utilisés dans le développement de logiciels de démonstration assistée (ex. COQ et variations), qui est un sujet de recherche actuellement très actif. Tous ces systèmes sont intuitionistes.

En ce qui concerne la logique classique on sait, depuis moins de 10 ans (et c'est toujours en progression), qu'on peut enrichir le λ-calcul et les règles des systèmes de manière à étendre la Correspondance de Curry-Howard à la logique classique. Du point de vue de la programmation cela revient à introduire des traits de programmation impérative là où on avait de la programmation essentiellement fonctionnelle (instructions dites “de contrôle”, dont la plus simple est “exit”).

Quelques (propositions de) solutions d'exercices.

Exercice 39.

Soit $\mathcal{S} \subseteq \Lambda$ adéquat pour F, et tel que $\mathcal{S}$ ait (au moins) pour élément $\emptyset$ et deux ensembles disjoints non vides S_0 et S_1 (par exemple : $\mathcal{S} =_{def} P^{\beta}(\Lambda)$). Soit I une interprétation à valeurs dans $\mathcal{S}$. Puisque $\cap_{S\in\mathcal{S}} S = \emptyset$on a $I(\perp) = \emptyset$. Par ailleurs, pour tout $S \in \mathcal{S}$, $S \neq \emptyset$ on a :

$I(A) \subseteq ((S \Rightarrow \emptyset) \Rightarrow \emptyset) \Rightarrow S \ = \ \Lambda \Rightarrow S$. Il en résulte que :

$I(A) \subseteq (\Lambda \Rightarrow S_0) \cap (\Lambda \Rightarrow S_1) = \Lambda \Rightarrow (S_0 \cap S_1) = \Lambda \Rightarrow \emptyset = \emptyset$.

L'existence d'un t tel que $\vdash t : A$ contredirait donc le Lemme d'adéquation.

Par ailleurs on a aussi, pour tout $S \in \mathcal{S}$, $S \neq \emptyset$:

$I(T) \subseteq ((S \to S), ((S \to \emptyset) \to S) \to S) \subseteq ((S \to S), \Lambda \to S)$.

Donc, si $t \in I(T)$ (ce qui aurait lieu si on avait $\vdash t : T$) on aura $t\, id\, id \in S$. Donc $t\, id\, id \in S_0 \cap S_1 = \emptyset$, contradiction.

Notes bibliographiques.

J.L. Krivine, Lambda-calcul, types et modèles, Masson 1990, Ellis et Horwood 1993 pour la version anglaise.

J.L. Krivine et M. Parigot, Programming with Proofs, J. Inf. Cybern. EIK 26 n°3, p.149-167, 1990. [AF2 et extensions]

J.L.Krivine, Un algorithme non typable dans le système F, C.R. Acad. Sc. Paris, t.304, série I, n°5, 1987.

J.Y. Girard, Y. Lafont, P. Taylor, Proofs and types, Cambridge Tracts in Computer Science 7, Cambridge University Press, 1989.

J.Y. Girard, Thèse de doctorat d'état, en mathématiques, Université Paris 7, 1972.

D. Leivant, Reasoning about functional programs and complexity classes associated with type disciplines, in FOCS, 1983, pp.460-469. [AF2]

H. Barendregt, Lambda-calculi with types, in Handbook of Logic in Computer Science II, p.117-309, Oxford Science publications, 1992.

J.B. Wells, Typability and type checking in System F are equivalent and undecidable, Annals of Pure and Applied Logic 98, p. 111-156, 1999.

S. Ronchi della Rocca, pour des versions simplifiées du Système F et une version sans la α-équivalence (à trouver sur sa page web).

A AJOUTER : biblio sur la Correspondance de Curry-Howard (voir mes notes).