

Projet fonctionnel

*Basé sur le cours de Daniel HIRSCHKOFF
Notes prises par Hugo SALOU*



3 avril 2025

Table des matières

1	Le λ-calcul pur.	5
1.1	Liaison et α -conversion.	5
1.2	La β -réduction.	6
1.3	Substitutions.	7
1.4	Comparaison λ -calcul et FUN.	8
1.5	Exercice : les booléens.	9
1.6	Confluence de la β -réduction.	9
2	Le λ-calcul simplement typé.	14
2.1	Définition du système de types.	14
2.2	Propriétés de la relation de typage.	15
2.3	Normalisation forte.	16
2.4	Extension : le λ -calcul typé avec \times et 1	20
3	Introduction à la théorie de la démonstration.	22
3.1	Formules et preuves.	22
3.2	Et en Rocq ?	24
3.3	Liens avec le λ -calcul simplement typé : <i>correspondance de Curry-Howard</i>	25
3.4	Curry-Howard du côté calcul : les coupures.	26
3.5	Faux, négation, consistance.	27
3.6	Et en Rocq ? (partie 2)	29
3.7	Logique intuitionniste <i>vs</i> logique classique.	29
3.8	Logique classique et Curry-Howard : intuition opérationnelle.	31
4	Le λ-calcul polymorphe.	33
4.1	Typage, version implicite.	33
4.2	Typage, version explicite.	34

4.3	Point de vue logique sur le polymorphisme, système F.	36
4.4	Représentation des connecteurs logiques.	37
4.5	Le λ -calcul polymorphe, côté programmation.	38
4.6	Polymorphisme et inférence de types.	39

5 Types linéaires. 41

Introduction.

Le cours se décompose en deux parties :

- ▷ une partie *pratique* ($\sim 2/3$ du cours) avec de la programmation OCaml en binôme ([fouine](#), puis [pieuvre](#)) ;
- ▷ une partie *théorique* : avec du λ -calcul, typage et lien avec la logique.

Ce document contiendra les notes de cours pour la partie théorique. D'autres documents pour la partie projet sont en ligne sur mon site (transformation CPS, et optimisations associées à la β -réduction).

Parfois, des références aux chapitres du cours de théorie de la programmation seront fait, ce sont des liens cliquables qui mènent vers le PDF du chapitre en question, par exemple :

[Théorie de la Programmation \[Chapitre 0\]](#).

1 Le λ -calcul pur.

Le λ -calcul a trois domaines proches :

- ▷ la *calculabilité*, avec l'équivalence entre machines de Turing et λ -expression (vue en FDI) ;
- ▷ la *programmation fonctionnelle* (vue en **Théorie de la Programmation** [Chapitre 6] avec le petit langage FUN) ;
- ▷ la *théorie de la démonstration* (vue dans la suite de ce cours).

On se donne un ensemble infini \mathcal{V} de variables notées x, y, z, \dots . Les *termes* (du λ -calcul) ou λ -termes sont définis par la grammaire

$$M, N, \dots ::= \lambda x. M \mid M N \mid x.$$

La construction $\lambda x. M$ s'appelle l' *abstraction* ou λ -*abstraction*. Elle était notée **fun** $x \rightarrow M$ en cours de théorie de la programmation.

Notation. ▷ On notera $M N P$ pour $(M N) P$.

- ▷ On notera $\lambda xyz. M$ pour $\lambda x. \lambda y. \lambda z. M$ (il n'y a pas lieu de mettre des parenthèses ici, vu qu'il n'y a pas d'ambiguïtés).
- ▷ On notera $\lambda x. M N$ pour $\lambda x. (M N)$. **Attention**, c'est différent de $(\lambda x. M) N$.

1.1 Liaison et α -conversion.

Remarque 1.1 (Liaison). Le « λ » est un lieu. Dans $\lambda y. x y$, la variable y est *liée* mais pas x (la variable x libre). On note $\mathcal{V}\ell(M)$ l'ensemble des variables libres de M , définie par induction sur M (il y a 3 cas).

Remarque 1.2 (α -conversion).

On note $=_\alpha$ la relation d' α -conversion. C'est une relation binaire sur les λ -termes fondée sur l'idée de renommage des abstractions *en évitant la capture de variables libres* :

$$\lambda x. x y =_\alpha \lambda t. x t \neq_\alpha \lambda x. x x.$$

Ainsi $\lambda x. M =_\alpha \lambda z. M'$ où M' est obtenu en remplaçant x par z *là où il apparaît libre* et *à condition que* $z \notin \mathcal{V}\ell(M)$. Ceci, on peut le faire partout.

Lemme 1.1. La relation $=_\alpha$ est une relation d'équivalence. Si $M =_\alpha N$ alors $\mathcal{V}\ell(M) = \mathcal{V}\ell(N)$.

Par convention, on peut identifier les termes modulo $=_\alpha$. On pourra donc toujours dire

« considérons $\lambda x. M$ où $x \notin E [\dots]$ »

avec E un ensemble *fini* de variables.

Ceci veut dire qu'on notera

$$M = N \text{ pour signifier que } M =_\alpha N.$$

1.2 La β -réduction.

Définition 1.1 (β -réduction). On définit la relation de β -réduction sur les λ -termes, notée \rightarrow_β ou \rightarrow , définie par les règles d'inférences :

$$\frac{}{(\lambda x. M) N \rightarrow_\beta M[N/x]} \quad \frac{M \rightarrow_\beta M' \quad N \rightarrow_\beta N'}{M N \rightarrow_\beta M' N'} \quad \frac{M \rightarrow_\beta M' \quad N \rightarrow_\beta N'}{M N \rightarrow_\beta M N'}$$

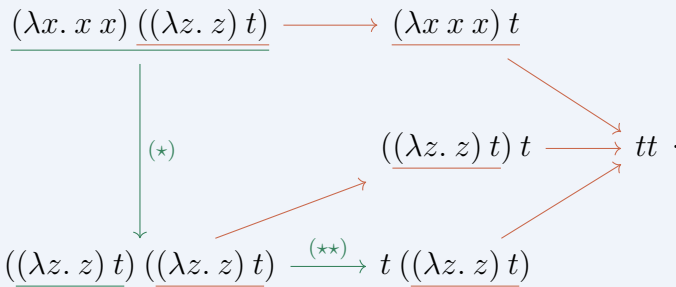
où $M[N/x]$ est la substitution de x par N dans M (on le définit ci-après).

Définition 1.2. Un terme de la forme $(\lambda x. M) N$ est appelé un *redex* (pour *reducible expression*) ou β -redex. Un terme M est une *forme normale* s'il n'existe pas de N tel que $M \rightarrow_\beta N$.

Remarque 1.3. La relation \rightarrow_β n'est pas terminante :

$$\Omega := (\lambda x. x x) (\lambda y. y y) \rightarrow_\beta (\lambda y. y y) (\lambda y. y y) =_\alpha \Omega.$$

Exemple 1.1.



Un pas de β -réduction peut :

- ▷ dupliquer un terme (c.f. (\star)) ;
- ▷ laisser un redex inchangé (c.f. $(\star\star)$) ;
- ▷ faire disparaître un redex (qui n'est pas celui que l'on contracte) :

$$(\lambda x. u)((\lambda z. z) t) \rightarrow_\beta u ;$$

- ▷ créer de nouveaux redex :

$$(\lambda x. x y) (\lambda z. z) \rightarrow_\beta (\lambda z. z) y.$$

1.3 Substitutions.

Exemple 1.2. Le terme $\lambda xy.x$ c'est une « fonction fabriquant des fonctions constantes » au sens où

$$(\lambda xy.x)M \rightarrow_\beta \lambda y. M,$$

à condition que $y \notin \mathcal{V}\ell(M)$. On doit cependant α -renommer pour éviter la capture :

$$\begin{array}{c} (\lambda xy.x) (\lambda t. y) \not\rightarrow_\beta \lambda y. (\lambda t. y) \\ \parallel \\ (\lambda xy'.x) (\lambda t. y) \rightarrow_\beta \lambda y'. (\lambda t. y). \end{array}$$

Définition 1.3. On procède par induction, il y a trois cas :

- ▷ $y[N/x] := \begin{cases} N & \text{si } y = x \\ y & \text{si } y \neq x \end{cases}$
- ▷ $(M_1 M_2)[N/x] := (M_1[N/x]) (M_2[N/x])$
- ▷ $(\lambda y. M)[N/x] := \lambda y. (M[N/x])$ **à condition que** $y \notin \mathcal{V}\ell(N)$ **et** $y \neq x$.

Lemme 1.2 (Gymnastique des substitutions). Pour $y \notin \mathcal{V}\ell(R)$,

$$(P[Q/y])[R/x] = (P[R/x])[Q[R/x]/y].$$

Lemme 1.3. Si $M \rightarrow_\beta M'$ alors $\mathcal{V}\ell(M') \subseteq \mathcal{V}\ell(M)$.

1.4 Comparaison λ -calcul et FUN.

En λ -calcul, on a une règle

$$\frac{M \rightarrow_\beta M'}{\lambda x. M \rightarrow_\beta \lambda x. M'}.$$

Cette règle n'existe pas en FUN (ni en **fouine**) car on traite les fonctions comme des valeurs. Et, en FUN, les trois règles suivantes sont

mutuellement exclusives :

$$\frac{}{(\lambda x. M) N \rightarrow_{\beta} M[N/x]} \quad \frac{M \rightarrow_{\beta} M'}{M N \rightarrow_{\beta} M' N} \quad \frac{N \rightarrow_{\beta} N'}{M N \rightarrow_{\beta} M N'}$$

car on attend que N soit une **valeur** avant de substituer.

En FUN (comme en **fouine**), pour l'exemple 1.1, on se limite à n'utiliser que les flèches rouges.

La relation \rightarrow_{β} est donc « plus riche » que \rightarrow_{FUN} . En FUN, on a une *stratégie de réduction* : on a au plus un redex qui peut être contracté. On n'a pas de notion de valeur en λ -calcul pur. Le « *résultat d'un calcul* » est une forme normale.

1.5 Exercice : les booléens.

On définit

$$\mathbf{T} := \lambda xy. x \quad \mathbf{F} := \lambda xy. y.$$

Ainsi, pour tout M (si $y \notin \mathcal{V}\ell(M)$),

$$\mathbf{T} M \rightarrow \lambda y. M \quad \mathbf{F} M \rightarrow \lambda y. y =: \mathbf{I}.$$

La construction **if** b **then** M **else** N se traduit en $b M N$.

Le « non » booléen peut se définir par :

- ▷ **not** := $\lambda b. b \mathbf{F} \mathbf{T} = \lambda b. b (\lambda xy. y) (\lambda tu. t)$;
- ▷ **not'** := $\lambda b. \lambda xy. byx$.

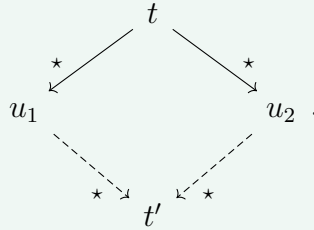
La première version est plus abstraite, la seconde est « plus électricien ». On a deux formes normales **différentes**.

De même, on peut définir le « et » booléen :

- ▷ **and** := $\lambda b_1. \lambda b_2. b_1 (b_2 \mathbf{T} \mathbf{F}) \mathbf{F}$;
- ▷ **and'** := $\lambda b_1. \lambda b_2. \lambda xy. b_1 (b_2 x y) y$.

1.6 Confluence de la β -réduction.

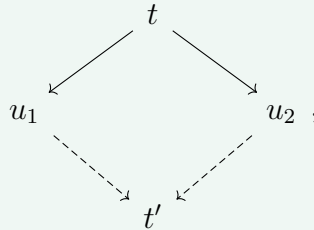
Définition 1.4 (Rappel, c.f. **Théorie de la Programmation** [Chapitre 10]). On dit que \rightarrow est *confluente* en $t \in A$ si, dès que $t \rightarrow^* u_1$ et $t \rightarrow^* u_2$ il existe t' tel que $u_1 \rightarrow^* t'$ et $u_2 \rightarrow^* t'$.



Les flèches en pointillés représentent l'existence.

On dit que \rightarrow est *confluente* si \rightarrow est confluente en tout $a \in A$.

La propriété du diamant correspond au diagramme ci-dessous :



c'est-à-dire si $t \rightarrow u_1$ et $t \rightarrow u_2$ alors il existe t' tel que $u_1 \rightarrow t'$ et $u_2 \rightarrow t'$.

La confluence pour \rightarrow , c'est la propriété du diamant pour \rightarrow^* . On sait déjà que la β -réduction n'a pas la propriété du diamant (certains chemins de l'exécution sont plus longs), mais on va montrer qu'elle est confluente.

Définition 1.5. On définit la relation de *réduction parallèle*, notée \Rightarrow , par les règles d'inférences suivantes :

$$\begin{array}{c}
\frac{}{x \Rightarrow x} \quad \frac{M \Rightarrow M'}{\lambda x. M \Rightarrow \lambda x. M'} \\
\frac{M \Rightarrow M' \quad N \Rightarrow N'}{M N \Rightarrow M' N'} \quad \frac{M \Rightarrow M' \quad N \Rightarrow N'}{(\lambda x. M) N \Rightarrow M'[N'/x]}
\end{array}$$

Lemme 1.4. La relation \Rightarrow est réflexive.

Lemme 1.5. Si $\mathcal{R} \subseteq \mathcal{S}$ alors $\mathcal{R}^* \subseteq \mathcal{S}^*$. De plus, $(\mathcal{R}^*)^* = \mathcal{R}^*$.

Lemme 1.6. Les relations \rightarrow^* et \Rightarrow^* coïncident.

Preuve. \triangleright On a $\rightarrow^* \subseteq \Rightarrow^*$ car cela découle de $\rightarrow \subseteq \Rightarrow$ par induction sur \rightarrow en utilisant la réflexivité de \Rightarrow .

- \triangleright On a $\Rightarrow^* \subseteq \rightarrow^*$ car cela découle de $\Rightarrow \subseteq \rightarrow^*$. En effet, on montre que pour tout M, M' si $M \Rightarrow M'$ alors $M \rightarrow^* M'$, par induction sur \Rightarrow . Il y a 4 cas.
- Pour $x \Rightarrow x$, c'est immédiat.
 - Pour l'abstraction, on suppose $M \Rightarrow M'$ alors par induction $M \rightarrow^* M'$, et donc $\lambda x. M \rightarrow^* \lambda x. M'$ par induction sur $M \rightarrow^* M'$.
 - Pour l'application, c'est plus simple que pour la précédente.
 - Pour la substitution, supposons $M \Rightarrow M'$ et $N \Rightarrow N'$. On déduit par hypothèse d'induction $M \rightarrow^* M'$ et $N \rightarrow^* N'$. Et, par induction sur $M \rightarrow^* M'$, on peut montrer que $(\lambda x. M) N \rightarrow^* (\lambda x. M') N$. Puis, par induction sur $N \rightarrow^* N'$, on montre $(\lambda x. M') N \rightarrow^* (\lambda x. M') N'$. Enfin, par la règle de β -réduction, on a $(\lambda x. M') N' \rightarrow M'[N'/x]$. On rassemble tout pour

obtenir :

$$(\lambda x. M) N \rightarrow^* M'[N'/x].$$

□

On est donc ramené à montrer que \Rightarrow^* a la propriété du diamant. Or \Rightarrow a la propriété du diamant, ce que l'on va montrer en TD.

Lemme 1.7. Si $M \Rightarrow M'$ alors $N \Rightarrow N'$ implique $M[N/x] \Rightarrow M'[N'/x]$.

Preuve. Par induction sur $M \Rightarrow M'$, il y a 4 cas. On ne traite que le cas de la 4ème règle. On suppose donc $M = (\lambda y. P) Q$ avec $y \notin \mathcal{V}\ell(N)$ et $y \neq x$. On suppose aussi $P \Rightarrow P'$, $Q \Rightarrow Q'$ et $M' = P'[Q'/y]$. On suppose de plus $N \Rightarrow N'$. Par hypothèse d'induction, on a $P[N/x] \Rightarrow P'[N'/x]$ et $Q[N/x] \Rightarrow Q'[N'/x]$. On applique la 4ème règle d'inférence définissant \Rightarrow pour déduire

$$\begin{aligned} & \underbrace{(\lambda y. (P[N/x]))}_{\parallel} (Q[N/x]) \Rightarrow (P'[N'/x])[Q'[N'/x]/y] = (P'[Q'/y])[N'/x] \\ & \quad \parallel \\ & (\lambda y. P)[N/x] \\ & \quad \text{car } x \neq y \end{aligned}$$

par le lemme de gymnastique des substitutions et car $y \notin \mathcal{V}\ell(N') \subseteq \mathcal{V}\ell(N)$ et car $N \rightarrow^* N'$. □

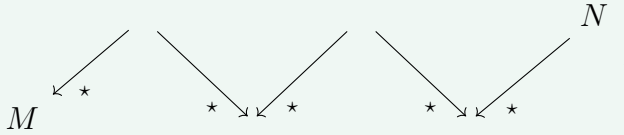
Proposition 1.1. La relation \Rightarrow a la propriété du diamant.

Preuve. Vu en TD. □

Corollaire 1.1. On a la confluence de \rightarrow_β .

Définition 1.6. La β -équivalence, ou β -convertibilité est la plus petite relation d'équivalence contenant \rightarrow_β . On la note $=_\beta$.

Si l'on a



alors $M =_{\beta} N$.

Proposition 1.2. Tout λ -terme est β -équivalent à au plus une forme normale.

Preuve. Si $M =_{\beta} N$ et M, N sont des formes normales, alors par confluence il existe P tel que $M \rightarrow^* P$ et $N \rightarrow^* P$. On a donc que $M = N = P$. \square

Remarque 1.4 (Conséquences). \triangleright Deux normales distinctes (au sens de $=_{\alpha}$) ne sont pas β -convertibles.

- \triangleright Si on a un λ -terme qui diverge et qui a une forme normale, par exemple $(\lambda x. y) \Omega$, alors on peut toujours « revenir » sur la forme normale.

2 Le λ -calcul simplement typé.

Dans ce chapitre, on va parler de *typage*. Ceci permet de « stratifier » les λ -termes. En effet, pour l’instant, tous les termes se ressemblent.

2.1 Définition du système de types.

Définition 2.1. On se donne un ensemble de *types de base*, notés X, Y, Z, \dots . Les types simples sont donnés par la grammaire suivante :

$$A, B, C ::= \mathbf{X} \mid A \rightarrow B.$$

Il n’y a donc que deux « types » de types : les types de base, et les types fonctions. Il n’y a donc pas de type `unit`, `bool`, `...`. En effet, ceci demanderait d’ajouter des constantes `()`, `true`, `false`, *etc* dans la grammaire du λ -calcul (et ceci demanderait ensuite d’ajouter des règles de typage supplémentaire). On verra en TD comment typer **T** et **F** comme défini au chapitre précédent.

Par convention, on notera $A \rightarrow B \rightarrow C$ pour $A \rightarrow (B \rightarrow C)$.

Définition 2.2. On définit une *hypothèse de typage* comme un couple variable-type (x, A) noté $x : A$.

Définition 2.3. Un *environnement de typage*, noté $\Gamma, \Gamma', \text{etc}$ est un dictionnaire sur $(\mathcal{V}, \text{Types})$, *c.f.* cours de Théorie de la Programmation. On notera $\Gamma(x) = A$ lorsque Γ associe x à A . On définit

le *domaine* de Γ comme

$$\text{dom}(\Gamma) := \{x \mid \exists A, \Gamma(x) = A\}.$$

On note aussi $\Gamma, x : A$ l'extension de Γ avec $x : A$ si $x \notin \text{dom}(\Gamma)$.

Définition 2.4. On définit la *relation de typage*, notée $\Gamma \vdash M : A$ (« sous les hypothèses Γ , le λ -terme M a le type A ») par les règles d'inférences suivantes :

$$\begin{array}{c} \Gamma(x) = A \quad \frac{}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \\[2ex] \frac{x \notin \text{dom}(\Gamma) \quad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \end{array}$$

Dans cette dernière règle, on peut toujours l'appliquer modulo α -conversion (il suffit d' α -renommer x dans $\lambda x. M$).

Exemple 2.1. On peut omettre le « \emptyset » avant « \vdash ».

$$\frac{\frac{\frac{}{f : \mathbf{X} \rightarrow \mathbf{X}, z : \mathbf{X} \vdash f : \mathbf{X} \rightarrow \mathbf{X}}{f : \mathbf{X} \rightarrow \mathbf{X}, z : \mathbf{X} \vdash f : \mathbf{X} \rightarrow \mathbf{X}} \quad \frac{\frac{f : \mathbf{X} \rightarrow \mathbf{X}, z : \mathbf{X} \vdash f : \mathbf{X} \rightarrow \mathbf{X}}{f : \mathbf{X} \rightarrow \mathbf{X}, z : \mathbf{X} \vdash f z : \mathbf{X}} \quad \frac{}{f : \mathbf{X} \rightarrow \mathbf{X}, z : \mathbf{X} \vdash z : \mathbf{X}}}{\frac{f : \mathbf{X} \rightarrow \mathbf{X}, z : \mathbf{X} \vdash f(fz) : \mathbf{X}}{f : \mathbf{X} \rightarrow \mathbf{X} \vdash \lambda z. f(fz) : \mathbf{X} \rightarrow \mathbf{X}}} \vdash \lambda f. \lambda z. f(fz) : (\mathbf{X} \rightarrow \mathbf{X}) \rightarrow \mathbf{X} \rightarrow \mathbf{X}$$

Exemple 2.2.

$$\frac{\frac{\frac{}{a, X, t : X \rightarrow Y \vdash t : X \rightarrow Y} \quad \frac{}{a, X, t : X \rightarrow Y \vdash a : X}}{a, X, t : X \rightarrow Y \vdash t a : Y}}{a : X \vdash \lambda t. t a : (X \rightarrow Y) \rightarrow Y} .$$

2.2 Propriétés de la relation de typage.

Lemme 2.1 (Lemme administratif).

- ▷ Si $\Gamma \vdash M : A$ alors $\mathcal{V}\ell(M) \subseteq \text{dom}(\Gamma)$.
- ▷ *Renforcement* : si $\Gamma, x : B \vdash M : A$ et $x \notin \mathcal{V}\ell(M)$ alors $\Gamma \vdash M : A$.
- ▷ *Affaiblissement* : si $\Gamma \vdash M : A$ alors, pour tout B et tout $x \notin \text{dom}(\Gamma)$ alors $\Gamma, x : B \vdash A$.
- ▷ *Contraction* : si $\Gamma, x : B, y : B \vdash M : A$ alors $\Gamma, x : B \vdash M[x/y] : A$ □

Proposition 2.1 (Pr  servation du typage). Si $\Gamma \vdash M : A$ et $M \rightarrow_\beta M'$ alors $\Gamma \vdash M' : A$.

Preuve. On proc  de comme en [Th  orie de la Programmation \[Chapitre 7\]](#) avec le lemme suivant.

Lemme 2.2. Si $\Gamma, x : A \vdash M : B$ et $\Gamma \vdash N : A$ alors $\Gamma \vdash M[N/x] : B$.

□

2.3 Normalisation forte.

D  finition 2.5. Un λ -terme M est dit *fortement normalisant* ou *terminant* si toute suite de β -r  ductions issue de M conduit    une forme normale. Autrement dit, il n'y a pas de divergence issue de M .

Th  or  me 2.1. Si M est typage (il existe Γ, A tels que $\Gamma \vdash M : A$) alors M est fortement normalisant.

Remarque 2.1 (Quelques tentatives de preuves rat  es.). ▷ Par induction sur M ? Non.

- ▷ Par induction sur la relation de typage $\Gamma \vdash M : A$? Non (le cas de l'application pose problème car deux cas de β -réductions).

Pour démontrer cela, on utilise une méthode historique : les *candidats de réductibilité*.

Définition 2.6 (Candidat de réductibilité). Soit A un type simple. On associe à A un ensemble de λ -termes, noté \mathcal{R}_A appelé *candidats de réductibilité* (ou simplement *candidats*) associé à A , défini par induction sur A de la manière suivante :

- ▷ $\mathcal{R}_X := \{M \mid M \text{ est fortement normalisant}\}$;
- ▷ $\mathcal{R}_{A \rightarrow B} := \{M \mid \forall N \in \mathcal{R}_A, M N \in \mathcal{R}_B\}$.

L'idée est la suivante :

$$\begin{array}{c} M \text{ typable} \\ \Gamma \vdash M : A \end{array} \quad \rightsquigarrow \quad M \in \mathcal{R}_A \quad \rightsquigarrow \quad M \text{ fortement normalisant.}$$

Remarque 2.2 (Rappel sur le PIBF, c.f. Théorie de la Programmation [Chapitre 10]). Le principe d'induction bien fondé nous dit qu'une relation \mathcal{R} est terminante ssi pour tout prédicat \mathcal{P} sur E vérifie que si

$$\forall x \in E \left((\forall y, x \mathcal{R} y \implies \mathcal{P}(y)) \implies \mathcal{P}(x) \right)$$

alors $\forall x \in E, \mathcal{P}(x)$.

Proposition 2.2. Soit A un type simple. On a :

CR 1. Pour tout $M \in \mathcal{R}_A$, M est fortement normalisant.

CR 2. Pour tout $M \in \mathcal{R}_A$, si $M \rightarrow_\beta M'$ alors $M' \in \mathcal{R}_A$.

CR 3. Pour tout M neutre (c-à-d, M n'est pas une λ -abstraction), si $\forall M', M \rightarrow_\beta M' \implies M' \in \mathcal{R}_A$ alors $M \in \mathcal{R}_A$.

Preuve. On montre la conjonction de **CR 1**, **CR 2** et **CR 3** par induction sur A . Il y a deux cas.

▷ Cas X un type simple.

CR 1. C'est vrai par définition.

CR 2. Si M est fortement normalisant, et $M \rightarrow_\beta M'$ alors $M' \in \mathcal{R}_X$.

CR 3. Si M est neutre et si on a que « pour tout M' tel que $M \rightarrow_\beta M'$ alors $M' \in \mathcal{R}_X$ » alors c'est l'induction bien fondée pour \rightarrow_β sur \mathcal{R}_X .

▷ Cas $A \rightarrow B$ un type flèche.

CR 1. Soit $M \in \mathcal{R}_{A \rightarrow B}$. Supposons que M diverge :

$$M \rightarrow_\beta M_1 \rightarrow_\beta M_2 \rightarrow_\beta \cdots$$

On a observé que $x \in \mathcal{R}_A$ pour une variable x arbitraire (conséquence de **CR 3** pour A). Par définition de $\mathcal{R}_{A \rightarrow B}$, $M x \in \mathcal{R}_B$. Par **CR 1** pour B , on a que $M x$ est fortement normalisant. Or, $M x \rightarrow_\beta M_1 x$ car $M \rightarrow_\beta M_1$. On construit ainsi une divergence dans \mathcal{R}_B à partir de $M x$:

$$M x \rightarrow_\beta M_1 x \rightarrow_\beta M_2 x \rightarrow_\beta \cdots$$

C'est absurde car cela contredit que $M x$ fortement normalisant.

CR 2. Soit $M \in \mathcal{R}_{A \rightarrow B}$ et $M \rightarrow M'$. Montrons que $M' \in \mathcal{R}_{A \rightarrow B}$, *i.e.* pour tout $N \in \mathcal{R}_A$ alors $M' N \in \mathcal{R}_B$. Soit donc $N \in \mathcal{R}_A$. On sait que $M N \in \mathcal{R}_B$ (car $M \in \mathcal{R}_{A \rightarrow B}$). Et comme $M \rightarrow_\beta M'$ alors $M N \rightarrow_\beta M' N$ et, par **CR 2** pour B , on a $M' N \in \mathcal{R}_B$. On a donc montré $\forall N \in \mathcal{R}_{A \rightarrow B}, M' N \in \mathcal{R}_B$ autrement dit, $M' \in \mathcal{R}_{A \rightarrow B}$.

CR 3. Soit M neutre tel que $\forall M', M \rightarrow_\beta M' \implies M' \in \mathcal{R}_{A \rightarrow B}$. Montrons que $M \in \mathcal{R}_{A \rightarrow B}$. On sait que \rightarrow_β est

terminante sur \mathcal{R}_A (par **CR 1** pour A). On peut donc montrer que $\forall N \in \mathcal{R}_A, M N \in \mathcal{R}_B$ par induction bien fondée sur \rightarrow_β . On a les hypothèses suivantes :

- hypothèse 1 : pour tout M' tel que $M \rightarrow_\beta M'$ alors $M' \in \mathcal{R}_{A \rightarrow B}$;
- hypothèse d'induction bien fondée : pour tout N' tel que $N \rightarrow_\beta N'$ que $M N' \in \mathcal{R}_B$.

On veut montrer $M N \in \mathcal{R}_B$. On s'appuie sur **CR 3** pour B et cela nous ramène à montrer que, pour tout P tel que $M N \rightarrow_\beta P$ est $P \in \mathcal{R}_B$. On a trois cas possibles pour $M N \rightarrow_\beta P$.

- Si $M = \lambda x. M_0$ et $P = M_0[N/x]$ qui est exclu car M est neutre.
- Si $P = M' N$ alors par hypothèse 1 $M' \in \mathcal{R}_{A \rightarrow B}$ et donc $M' N \in \mathcal{R}_B$.
- Si $P = M N'$ alors, par par hypothèse d'induction bien fondée, $M N' \in \mathcal{R}_B$.

□

Lemme 2.3. Soit M tel que $\forall N \in \mathcal{R}_A, M[N/x] \in \mathcal{R}_B$. Alors $\lambda x. M \in \mathcal{R}_{A \rightarrow B}$.

Preuve. On procède comme pour **CR 3** pour $A \rightarrow B$. □

Lemme 2.4. Supposons $x_1 : A_1, \dots, x_k : A_k \vdash M : A$. Alors, pour tout N_1, \dots, N_k tel que $N_i \in \mathcal{R}_{A_i}$, on a

$$M[N_1 \dots N_k / x_1 \dots x_k] \in \mathcal{R}_A.$$

On note ici la *substitution simultanée* des x_i par des N_i dans M . C'est **n'est pas** la composition des substitutions.

Preuve. Par induction sur la relation de typage, il y a trois cas.

- ▷ Si on a utilisé la règle de l'axiome, c'est que M est une variable : $M = x_i$ et $A = A_i$. Soit $N_i \in \mathcal{R}_{A_i}$ alors $M[N_1 \cdots N_k/x_1 \cdots x_k] = N_i \in \mathcal{R}_A$.
- ▷ Si on a utilisé la règle de l'application, c'est que M est une application : $M = M_1 M_2$ et $M_1 : B \rightarrow A$ et $M_2 : B$. On a :

$$M[N_1 \cdots N_k/x_1 \cdots x_k] = M_1[N_1 \cdots N_k/x_1 \cdots x_k] M_2[N_1 \cdots N_k/x_1 \cdots x_k].$$

On conclut en appliquant les hypothèses d'inductions : $M_1[N_1 \cdots N_k/x_1 \cdots x_k] \in \mathcal{R}_{B \rightarrow A}$ et $M_2[N_1 \cdots N_k/x_1 \cdots x_k] \in \mathcal{R}_B$.

- ▷ Si on a utilisé la règle de l'abstraction, c'est que $M = \lambda y. M_0$ avec $y \notin \{x_1, \dots, x_k\} \cup \mathcal{V}\ell(N_1) \cup \dots \cup \mathcal{V}\ell(N_k)$. Supposons que $x_1 : A_1, \dots, x_k : A_k \vdash \lambda y. M_0 : A \rightarrow B$. Alors nécessairement $x_1 : A_1, \dots, x_k : A_k, y : A \vdash M_0 : B$. Par hypothèse d'induction, on a que pour tout $N_i \in \mathcal{R}_{A_i}$ on a

$$M_0[N_1 \cdots N_k/x_1 \cdots x_k][N/y] = M_0[N_1 \cdots N_k N/x_1 \cdots x_k y] \in \mathcal{R}_B.$$

Par le lemme précédent, on déduit que

$$(\lambda y. M_0)[N_1 \cdots N_k/x_1 \cdots x_k] = \lambda y. (M_0[N_1 \cdots N_k/x_1 \cdots x_k]) \in \mathcal{R}_{A \rightarrow B}.$$

□

Corollaire 2.1. Si $\Gamma \vdash M : A$ alors $M \in \mathcal{R}_A$.

2.4 Extension : le λ -calcul typé avec \times et 1.

En ajoutant les couples et *unit*, il faut modifier quatre points.

Syntaxe. $M, N ::= \lambda x. M \mid M N \mid x \mid (M, N) \mid \star \mid \pi_1 M \mid \pi_2 M$

β -réduction.

$$\frac{M \rightarrow_\beta M'}{(M, N) \rightarrow_\beta (M', N)} \quad \frac{N \rightarrow_\beta N'}{(M, N) \rightarrow_\beta (M, N')}$$

$$\overline{\pi_1 (M, N) \rightarrow_\beta M} \quad \overline{\pi_2 (M, N) \rightarrow_\beta N}.$$

Types.

$$A, B ::= X \mid A \rightarrow B \mid A \times B \mid \mathbf{1}$$

Typage.

$$\frac{}{\star : \mathbf{1}} \quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B}$$

$$\frac{\Gamma \vdash P : M \times N}{\Gamma \vdash \pi_1 P : M} \quad \frac{\Gamma \vdash P : M \times N}{\Gamma \vdash \pi_2 P : N}.$$

3 Introduction à la théorie de la démonstration.

3.1 Formules et preuves.

Définition 3.1. On se donne un ensemble de *variables propositionnelles*, qui seront notées X, Y, Z , etc. L'ensemble des *formules* est défini par la grammaire :

$$A, B ::= X \mid A \Rightarrow B.$$

Cet ensemble de formules s'appelle le « *fragment implicatif de la logique propositionnelle intuitionniste* ».

Cela peut sembler inhabituel car, généralement, on commence par introduire \neg , \vee et \wedge , car on a en tête les booléens.

Définition 3.2. Les *séquents*, notés $\Gamma \vdash A$, un couple formé de Γ une **liste** de formules, et A une formule. La liste Γ est une *liste d'hypothèses*. On notera Γ, A la notation pour l'extension de la liste.

Définition 3.3. On *prouve* (*dérive*) les séquents à l'aides des *règles de déduction* (*d'inférence*) :

$$A \in \Gamma \quad \frac{}{\Gamma \vdash A} \text{Ax} \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow_I \quad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow_E.$$

On les appelle, dans l'ordre, règle de l'*axiome*, règle de l'*introduction de \Rightarrow* et règle de l'*élimination de \Rightarrow* . Il s'agit des *règles de déduction naturelle pour le fragment implicatif de la logique propositionnelle intuitionniste*.

Définition 3.4. Le séquent $\Gamma \vdash A$ est *prouvable* s'il existe une *preuve (dérivation)* ayant $\Gamma \vdash A$ à la racine et des axiomes aux feuilles. La formule A est *prouvable* si $\vdash A$ l'est.

Exemple 3.1.

$$\frac{\frac{\frac{\frac{\frac{}{X \Rightarrow Y, X \vdash X \Rightarrow Y} \text{Ax}}{X \Rightarrow Y, X \vdash X} \Rightarrow_E}{X \Rightarrow Y, X \vdash Y} \Rightarrow_I}{X \Rightarrow Y \vdash X \Rightarrow Y} \Rightarrow_I}{\vdash (X \Rightarrow Y) \Rightarrow (X \Rightarrow Y)} \Rightarrow_I .$$

On écrit généralement des « preuves génériques », en utilisant A, B plutôt que X, Y .

Exemple 3.2.

$$\frac{\frac{\frac{\frac{\frac{}{(A \Rightarrow A) \Rightarrow B \vdash (A \Rightarrow A) \Rightarrow B} \text{Ax}}{(A \Rightarrow A) \Rightarrow B \vdash A \Rightarrow A} \Rightarrow_I}{(A \Rightarrow A) \Rightarrow B \vdash B} \Rightarrow_I}{\vdash ((A \Rightarrow A) \Rightarrow B) \Rightarrow B} \Rightarrow_I .$$

3.2 Et en Rocq ?

En Rocq, un objectif de preuve

$$\left. \begin{array}{l} H_1 : A_1 \\ H_2 : A_2 \\ H_3 : A_3 \\ \vdots \end{array} \right\} \Gamma$$

$$A$$

correspond au séquent

$$\Gamma \vdash A.$$

Chaque tactique correspond à des opérations sur l'arbre de preuve. On construit « au fur et à mesure » l'arbre de preuve montrant $\Gamma \vdash A$. Voici ce que quelques tactiques Rocq font.

$$\begin{array}{c} \frac{??}{\Gamma, A, B, A \vdash A} \xrightarrow{\text{assumption}} \frac{??}{\Gamma, A, B, A \vdash A} \text{Ax} \\[2ex] \frac{??}{\Gamma \vdash C} \xrightarrow{\text{assert } A} \frac{\frac{??}{\Gamma, A \vdash B} \quad \frac{??}{\Gamma \vdash A} \Rightarrow_E}{\Gamma \vdash B} \\[2ex] \frac{??}{\Gamma \vdash C} \xrightarrow{\text{cut } A} \frac{\frac{??}{\Gamma \vdash A \Rightarrow B} \quad \frac{??}{\Gamma \vdash A} \Rightarrow_E}{\Gamma \vdash B} \\[2ex] \frac{??}{\Gamma \vdash C} \xrightarrow{\text{apply } H} \frac{\frac{??}{\Gamma, A \Rightarrow B \vdash A \Rightarrow B} \text{Ax} \quad \frac{??}{\Gamma, A \Rightarrow B \vdash A} \Rightarrow_E}{\Gamma, \underbrace{A \Rightarrow B}_H \vdash B} \\[2ex] \frac{??}{\Gamma \vdash B \Rightarrow C} \xrightarrow{\text{intro}} \frac{\frac{??}{\Gamma, B \vdash C}}{\Gamma \vdash B \Rightarrow C} \Rightarrow_I \end{array}$$

3.3 Liens avec le λ -calcul simplement typé : *correspondance de Curry-Howard*.

Les règles de typage du λ -calcul correspondent aux règles d'inférences du fragment implicatif :

$$\begin{array}{ccc}
 \frac{x : A \in \Gamma \quad \overline{\Gamma \vdash x : A}}{\Gamma, x : A \vdash M : B} & \longleftrightarrow & \frac{A \in \Gamma \quad \overline{\Gamma \vdash A}}{\Gamma, A \vdash B} \text{Ax} \\
 \frac{\Gamma \vdash \lambda x. M : A \rightarrow B}{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A} & \longleftrightarrow & \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow_I \\
 \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} & \longleftrightarrow & \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow_I
 \end{array}$$

En retirant les λ -termes en bleu (incluant les « : »), et en changeant \rightarrow en \Rightarrow , on obtient exactement les mêmes règles.

Si on sait que $\Gamma \vdash x : A$ alors, en effaçant les parties en bleu, on obtient une preuve de $\tilde{\Gamma} \vdash A$.

Inversement, on se donne une preuve de $\Gamma \vdash A$. On se donne des variables x_i pour transformer $\Gamma = A_1, \dots, A_k$ en $\hat{\Gamma} = x_1 : A_1, \dots, x_k : A_k$. Par induction sur $\Gamma \vdash A$, on montre qu'il existe un λ -terme tel que $\hat{\Gamma} \vdash M : A$. On a trois cas.

- ▷ Pour \Rightarrow_I , par induction, si $\hat{\Gamma}, x = A \vdash M : B$, on déduit $\hat{\Gamma} \vdash \lambda x. M : A \rightarrow B$.
- ▷ Pour \Rightarrow_{I_1} , par induction, si $\hat{\Gamma} \vdash M : A \rightarrow B$ et $\hat{\Gamma} \vdash N : A$, on déduit $\hat{\Gamma} \vdash M N : B$.
- ▷ Pour **Ax**, on sait $A \in \Gamma$ donc il existe x tel que $x : A \in \hat{\Gamma}$, et on conclut $\hat{\Gamma} \vdash x : A$.

On a les propriétés suivantes pour la relation de déduction :

- ▷ *affaiblissement* : si $\Gamma \vdash B$ (implicitement « est prouvable ») alors $\Gamma, A \vdash B$;
- ▷ *contraction* : si $\Gamma, A, A \vdash B$ alors $\Gamma, A \vdash B$;
- ▷ *renforcement* si $\Gamma, A \vdash B$ alors $\Gamma \vdash B$ à condition qu'on n'utilise pas l'axiome avec l'hypothèse **A** (celle là uniquement, les A intermédiaires ne posent pas de problèmes) pour déduire B .

▷ *échange* ; si $\Gamma, A, B, \Gamma' \vdash C$ alors $\Gamma, B, A, \Gamma' \vdash C$.

C'est analogue aux propriétés du typage en λ -calcul.

En effet, la propriété de renforcement, très imprécise dans sa formulation logique, est simplement : si $\hat{\Gamma}, x : A \vdash M : B$ alors $\hat{\Gamma} \vdash M : B$ à condition que $x \notin \mathcal{V}\ell(M)$.

Si on veut prouver ces propriétés (au lieu d'utiliser la correspondance de Curry-Howard), on ferait une induction sur la preuve du séquent qui est donné.

La règle

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

est *admissible*. En effet, si on sait prouver les prémisses (ici, $\Gamma \vdash B$) alors on sait prouver la conclusion (ici, $\Gamma, A \vdash B$). Ceci dépend fortement de la logique que l'on utilise.

3.4 Curry-Howard du côté calcul : les coupures.

Typons un redex :

$$\frac{\frac{\Gamma, x : A \vdash M : B}{\lambda x. M : A \rightarrow B} \Rightarrow_I \quad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x. M) N : M} \Rightarrow_E.$$

Oui, c'est exactement la même chose que la tactique `assert` en Rocq.

Définition 3.5. Une *coupure* est un endroit dans la preuve où il y a un usage d'une règle d'élimination (\Rightarrow_E) dont la prémisses principale est déduite à l'aide d'une règle d'introduction (\Rightarrow_I) pour le même connecteur logique.

Remarque 3.1. Ici, on n'a qu'un seul connecteur logique, \Rightarrow , mais cela s'étend aux autres connecteurs que l'on pourrait ajouter. La *prémisse principale* est, par convention, la première.

On peut *éliminer une coupure* pour \Rightarrow , c'est-à-dire transformer une preuve (c.f. contracter un β -redex) en passant de

$$\frac{\frac{\frac{\delta}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \Rightarrow_I \quad \frac{\delta'}{\Gamma \vdash A} \Rightarrow_E}{\Gamma \vdash B}$$

à

$$\frac{\delta[\delta'/A]}{\Gamma \vdash B}$$

où l'on note $\delta[\delta'/A]$ la preuve obtenue en remplaçant dans δ chaque usage de l'axiome avec A par δ' .

On a le même séquent en conclusion (c.f. préservation du typage en λ -calcul simplement typé).

La correspondance de Curry-Howard c'est donc :

$$\begin{array}{ll} \text{Types} & \longleftrightarrow \text{Formules} \\ \text{Programmes} & \longleftrightarrow \text{Preuves} \\ \beta\text{-réduction} & \longleftrightarrow \text{Élimination d'une coupure} \\ \textbf{Programmation} & \longleftrightarrow \textbf{Logique} \end{array}$$

3.5 Faux, négation, consistance.

On modifie nos formules :

$$A, B ::= X \mid A \Rightarrow B \mid \perp$$

et on ajoute la règle d'élimination du \perp (il n'y a pas de règle d'introduction) :

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp_E.$$

La négation $\neg A$ est une notation pour $A \Rightarrow \perp$. On peut donc prouver le séquent $\vdash A \Rightarrow \neg\neg A$:

$$\frac{\frac{\frac{\overline{A, \neg A \vdash \neg A} \text{ Ax}}{A, \neg A \vdash \perp} \Rightarrow_E}{A \vdash \neg\neg A} \Rightarrow_I}{\vdash A \Rightarrow \neg\neg A} \Rightarrow_I .$$

Théorème 3.1 (Élimination des coupures). Si $\Gamma \vdash A$ (est prouvable) alors il existe une preuve *sans coupure* de $\Gamma \vdash A$.

Preuve. *c.f.* TD. □

Remarque 3.2 (Lien avec normalisation forte en λ -calcul simplement typé). Ici, on veut la normalisation faible (« il existe une forme normale ... »). On ne peut pas appliquer *stricto sensu* la normalisation forte pour le λ -calcul simplement typé car le système de type contient \perp .

Lemme 3.1. Une preuve sans coupure de $\vdash A$ en logique intuitionniste se termine (à la racine) nécessairement par une règle d'introduction.

Preuve. Par induction sur $\vdash A$. Il y a 4 cas.

- ▷ **Ax** : Absurde car $\Gamma = \emptyset$.
- ▷ \Rightarrow_I : OK
- ▷ \Rightarrow_E : On récupère une preuve de $\vdash B \Rightarrow A$ qui termine (par induction) par une introduction \Rightarrow_I . Absurde car c'est une coupure.
- ▷ \perp_E : On récupère une preuve de \perp qui termine par une règle d'induction : impossible.

□

Corollaire 3.1 (Consistance de la logique). Il n’y a pas de preuve de \vdash en logique propositionnelle intuitionniste dans le fragment avec \Rightarrow et \perp .

Preuve. S’il y en avait une, il y en aurait une sans coupure, qui se termine par une règle d’introduction, impossible. \square

3.6 Et en Rocq ? (partie 2)

On étend les formules avec $\forall, \exists, \neg, \vee, \wedge$, *etc.* Les preuves sont des λ -termes. En effet, dans une preuve de $\vdash X \rightarrow X \rightarrow X$ on peut écrire

`exact (fun x y → x),`

pour démontrer le séquent.

Le mot clé **Qed** prend le λ -terme construit par la preuve et calcule M' sous forme normale tel que $M \rightarrow_{\beta}^* M'$. La logique de Rocq est *constructive*. C’est-à-dire qu’une preuve de $A \Rightarrow B$ c’est une fonction qui transforme une preuve de A en une preuve de B . Après avoir appelé **Qed**, il est possible d’extraire le λ -terme construit en un programme OCaml, Haskell, *etc.*

3.7 Logique intuitionniste vs logique classique.

Dans la logique que l’on a considérée (TD), on a deux règles d’introduction pour \vee :

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_1^g \quad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_1^g.$$

Lorsqu’on a une preuve de $A \vee B$, on a, soit une preuve de A , soit une preuve de B . Ce n’est pas une preuve « il est impossible de ne pas avoir A et B ». La logique est *constructive*.

On rappelle que $\neg A := A \rightarrow \perp$, et que l’on se donne la règle d’élimination de \perp :

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp_E.$$

La *logique classique* est la logique obtenue à l'aide de l'ajout d'une des règles suivantes :

Tiers exclu. $\frac{}{\Gamma \vdash A \vee \neg A}$ tiers exclu

Ce n'est pas constructif : on ne sait pas si l'on a une preuve de $\Gamma \vdash A$ ou de $\Gamma \vdash \neg A$.

Absurde. $\frac{}{\Gamma \vdash (\neg \neg A) \Rightarrow A}$ absurde

C'est mieux : ici, on n'a pas de \vee .

Loi de Peirce. $\frac{}{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$ peirce

C'est encore mieux : ici, on n'a pas de \vee , ni de \perp , mais c'est plus subtil.

En choisissant un de ces axiomes, on a la même notion de prouvabilité.

Exercice 3.1. Montrons que **absurde** implique **tiers exclu** (au sens de « on peut dériver l'un dans le système incluant l'autre »).

$$\begin{array}{c}
 \frac{\frac{\frac{\frac{\frac{\frac{\Gamma, \neg(A \vee \neg A), A \vdash \neg(A \vee \neg A)}{\Gamma, \neg(A \vee \neg A), A \vdash A} \text{ ax}}{\Gamma, \neg(A \vee \neg A), A \vdash A \vee \neg A} \vee^g}{\Gamma, \neg(A \vee \neg A), A \vdash \perp} \Rightarrow_1}{\Gamma, \neg(A \vee \neg A) \vdash \neg A} \vee^d}{\Gamma, \neg(A \vee \neg A) \vdash A \vee \neg A} \Rightarrow_E}{\Gamma, \neg(A \vee \neg A) \vdash \neg(A \vee \neg A)} \text{ ax}}{\Gamma, \neg(A \vee \neg A) \vdash A \vee \neg A} \Rightarrow_1 \\
 \frac{\Gamma \vdash (\neg \neg(A \Rightarrow \neg A)) \Rightarrow (A \vee \neg A) \text{ absurde}}{\Gamma \vdash A \vee \neg A} \Rightarrow_E
 \end{array}$$

Théorème 3.2 (Glivenko). Une formule A est prouvable en logique classique si et seulement si $\neg \neg A$ est prouvable en logique intuitionniste.

Preuve. Ressemble un peu à la traduction par continuation des programmes [fouine](#). \square

Corollaire 3.2. La logique classique est consistante ssi la logique intuitionniste est consistante.

Preuve. Si $\vdash \perp$ en logique intuitionniste alors $\vdash \perp$ en logique classique. Si $\vdash \perp$ en logique classique, alors $\neg \neg \perp$ en intuitionniste,

et on peut en déduire une preuve de $\vdash \perp$ en intuitionniste :

$$\frac{\frac{\text{par hyp.}}{\vdash (\perp \rightarrow \perp) \rightarrow \perp} \quad \frac{\frac{\perp \vdash \perp}{\vdash \perp \rightarrow \perp} \text{ ax}}{\vdash \perp} \Rightarrow \text{E}.$$

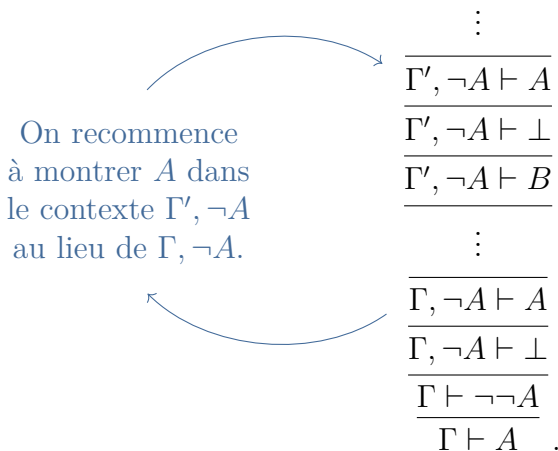
□

3.8 Logique classique et Curry-Howard : intuition opérationnelle.

On cherche à compléter la correspondance de Curry-Howard :

Types	\longleftrightarrow	Formules
Programmes	\longleftrightarrow	Preuves
β -réduction	\longleftrightarrow	Élimination d'une coupure
Principes classiques	\longleftrightarrow	???
Programmation	\longleftrightarrow	Logique

Avec la preuve par l'absurde, on peut « recommencer dans un contexte différent ».



S'autoriser les principes classiques, c'est savoir utiliser les exceptions : si ça explose, je peux le rattraper. En effet, l'élimination du \perp fait penser à un opérateur comme **raise** : **exn** \rightarrow 'a, et la construction **try...with...** pour pouvoir « sauter » à des endroits du programme, et dévier le flot du programme.

4 Le λ -calcul polymorphe.

On étend la grammaire des types :

$$A, B ::= X \mid A \rightarrow B \mid \forall X A.$$

Ici, les X ne sont plus les types de base (noté \mathbf{X} précédemment), mais ce sont des *variables de types*.

Pour mieux refléter les notations de la littérature, on aura plutôt :

$$A, B ::= \alpha \mid A \rightarrow B \mid \forall \alpha A.$$

Le « $\forall \alpha A$ » est une structure qui lie, \forall est un lieur. Il y a donc une notion de variables libres d'un type $\mathcal{V}\ell(A)$, d' α -conversion, et de substitution. Par exemple,

$$\mathcal{V}\ell(\forall \alpha \forall \beta (\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \alpha)) = \{\gamma\}$$

et

$$\forall \alpha \alpha \rightarrow \alpha =_{\alpha} \forall \beta \beta \rightarrow \beta.$$

4.1 Typage, version implicite.

Aux trois règles des types simples, on ajoute les deux règles ci-dessous :

$$\alpha \notin \mathcal{V}\ell(\Gamma) \quad \frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall \alpha A} \mathcal{T}_g \quad \frac{\Gamma \vdash M : \forall \alpha A}{\Gamma \vdash M : A[B/\alpha]} \mathcal{T}_i.$$

Les règles correspondent à la *généralisation* et à l'*instantiation*.

Et, *on ne change pas les λ -termes*.

4.2 Typage, version explicite.

On change les λ -termes :

$$M, N ::= x \mid M N \mid \lambda x : A. M \mid \Lambda \alpha. M \mid M A.$$

La construction $\Lambda \alpha. M$ est une *abstraction de types*, et la construction $M A$ est l'*application d'un terme à un type*. On note **explicitement** le type « d'entrée » de l'abstraction.

On change les règles de β -réduction :

$$\frac{}{(\Lambda \alpha. M) A \rightarrow_{\beta} M[A/\alpha]} \quad \frac{M \rightarrow_{\beta} M'}{\Lambda \alpha. M \rightarrow_{\beta} \Lambda \alpha. M'}$$

$$\frac{M \rightarrow_{\beta} M'}{M A \rightarrow_{\beta} M' A}.$$

On change également les règles de typage :

$$\Gamma(x) = A \quad \frac{}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A. M : A \rightarrow B} \quad x \notin \text{dom}(\Gamma)$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M : \forall \alpha A}{\Gamma \vdash \Lambda \alpha. M : \forall \alpha A \quad \Gamma \vdash M B : A[B/\alpha]} \quad \alpha \notin \mathcal{V}(\Gamma)$$

Lemme 4.1. On a $\Gamma \vdash M : A$ dans le système explicite ssi il existe M' dans le système explicite avec $\Gamma \vdash M' : A$ (explicite) et M est « l'effacement » de M' .

- ▷ La représentation implicite est plus utilisée dans les langages comme OCaml, où l'on doit inférer les types.
- ▷ La représentation explicite correspond plus aux langages comme Rocq, avec un lien plus naturel avec la logique.

Exemple 4.1. Soit le λ -terme non typé $\underline{2} := \lambda f. \lambda z. f(fz)$. Comment le typer en version explicite ?

1.

$$\frac{\frac{\frac{\vdots}{f : \alpha \rightarrow \alpha, z : \alpha \vdash f(fz) : \alpha}}{f : \alpha \rightarrow \alpha \vdash \lambda z : \alpha. f(fz) : \alpha \rightarrow \alpha}}{\emptyset \vdash \lambda f : \alpha \rightarrow \alpha. \lambda z : \alpha. f(fz) : (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha}}{\emptyset \vdash \Lambda \alpha. \lambda f : \alpha \rightarrow \alpha. \lambda z : \alpha. f(fz) : \forall \alpha (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha.}$$

2.

$$\frac{\frac{\frac{\vdots}{f : \forall \alpha \alpha \rightarrow \alpha, z : \beta \vdash f \beta (f \beta z) : \beta}}{f : \forall \alpha \alpha \rightarrow \alpha \vdash \lambda z : \beta. f \beta (f \beta z) : \beta \rightarrow \beta}}{\emptyset \vdash \lambda f : \forall \alpha \alpha \rightarrow \alpha. \lambda z : \beta. f \beta (f \beta z) : (\forall \alpha \alpha \rightarrow \alpha) \rightarrow \beta \rightarrow \beta.}$$

Exemple 4.2. On suppose que l'on a les couples. Comment compléter le séquent

$$y : \beta, z : \gamma \vdash \lambda f : \boxed{?}. (f y, f z) : \boxed{?}.$$

- ▷ Pour les types simples, on ne peut pas si $\beta \neq \gamma$.
- ▷ Mais, avec polymorphisme, on a :

$$y : \beta, z : \gamma \vdash \lambda f : \forall \alpha \alpha \rightarrow \delta. (f y, f z) : (\forall \alpha \alpha \rightarrow \delta) \rightarrow \delta \times \delta.$$

Exemple 4.3.

$$\frac{\frac{\frac{}{x : \alpha \vdash x : \alpha}}{\emptyset \vdash \lambda x. x : \alpha \rightarrow \alpha}}{\emptyset \vdash \lambda x. x : \forall \alpha \alpha \rightarrow \alpha}}{\emptyset \vdash \lambda x. x : (\forall \beta \beta) \rightarrow (\forall \delta \delta)}.$$

En effet, $(\forall \beta \beta) \rightarrow (\forall \beta \beta) =_{\alpha} (\forall \beta \beta) \rightarrow (\forall \delta \delta)$.

Exemple 4.4 (Ouch!).

$$\text{NON! On a } \alpha \in \mathcal{V}\ell(x : \alpha)! \quad \frac{\frac{\frac{}{x : \alpha \vdash x : \alpha}}{x : \alpha \vdash x : \forall \alpha \alpha}}{x : \alpha \vdash x : \beta}}{\emptyset \vdash \lambda x. x : \alpha \rightarrow \beta}}{\emptyset \vdash \lambda x. x : \forall \alpha \forall \beta \alpha \rightarrow \beta}.$$

4.3 Point de vue logique sur le polymorphisme, système F.

On se place du point de vue Curry-Howard. On ajoute deux règles de déduction supplémentaire :

$$\alpha \notin \mathcal{V}\ell(\Gamma) \quad \frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha A} \forall_I \quad \frac{\Gamma \vdash \forall \alpha A}{\Gamma \vdash A[B/\alpha]} \forall_E.$$

On a, encore, un possibilité d'éliminer les coupures : si $\alpha \notin \mathcal{V}\ell(\Gamma)$,

$$\frac{\frac{\frac{\delta}{\Gamma \vdash A}}{\Gamma \vdash \forall \alpha A} \forall_I}{\Gamma \vdash A[B/\alpha]} \forall_E \quad \rightsquigarrow \quad \frac{\delta[B/\alpha]}{\Gamma \vdash A[B/\alpha]}$$

Ceci correspond, par correspondance de Curry-Howard, à une β -réduction :

$$(\Lambda\alpha M) \textcolor{brown}{B} \rightarrow_{\beta} M[\textcolor{brown}{B}/\alpha].$$

On a aussi un théorème de normalisation forte.

Théorème 4.1. Si $\Gamma \vdash M : A$ en λ -calcul polymorphe, alors M est fortement normalisant.

Pour démontrer ce théorème, on utilise encore les candidats de réductibilité (il ne faut définir que $\mathcal{R}_{\forall\alpha A}$), mais la preuve est bien plus complexe. En effet, les types polymorphes ont un grand pouvoir expressif (*c.f.* la section suivante).

Le système que l'on a s'appelle *système F*.¹ C'est de la logique du second ordre : on peut quantifier sur les variables propositionnelles. On quantifie donc sur des ensembles de valeurs au lieu de se limiter uniquement à quantifier sur les valeurs.

4.4 Représentation des connecteurs logiques.

On peut représenter les connecteurs logiques différemment, sans les ajouter explicitement, mais uniquement avec le fragment contenant \rightarrow et \forall .

- ▷ On peut représenter $\perp := \forall\alpha\alpha$. En effet, on a la correspondance suivante :

$$\frac{\Gamma \vdash \forall\alpha\alpha}{\Gamma \vdash A} \forall_I \quad \rightsquigarrow \quad \frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp_E.$$

- ▷ On peut représenter $\top := \forall\alpha\alpha \rightarrow \alpha$.
- ▷ On peut représenter $A \wedge B := \forall\alpha (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha$:

1. À ne pas confondre avec *Station F*.

$$\begin{array}{c}
\frac{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash c : A \rightarrow B \rightarrow \alpha}{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash c M : B \rightarrow \alpha} \quad \frac{\Gamma \vdash M : A}{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash M : A} \quad \frac{\Gamma \vdash N : B}{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash M : A} \\
\hline
\frac{\Gamma, c : A \rightarrow B \rightarrow \alpha \vdash c M N \alpha}{\Gamma \vdash \lambda c. c M N : (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha} \\
\hline
\Gamma \vdash \lambda c. c M N : \forall \alpha (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha
\end{array}$$

d'où l'introduction du \wedge . Pour l'élimination, si

$$\Gamma \vdash M : \forall \alpha (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha,$$

alors $\Gamma \vdash M (\lambda x. \lambda y. x) : A$ et $\Gamma \vdash M (\lambda x. \lambda y. y) : B$.

4.5 Le λ -calcul polymorphe, côté programmation.

« **Généricité** ». On peut avoir plusieurs types en même temps. Par exemple, une fonction de tri a un type

$$\forall \alpha (\alpha \rightarrow \alpha \rightarrow \text{bool}) \rightarrow \text{list } \alpha \rightarrow \text{list } \alpha.$$

« **Paramétricité** ». Lorsque l'on a une fonction $f : (\forall \alpha : \alpha) \rightarrow A$, la fonction n'inspecte pas son argument. Elle ne fait que le dupliquer, le passer à d'autres fonctions, le rejeter, mais elle ne peut pas voir les données sous-jacentes. (On exclue ici les fonctions types `Obj.magic`).

Quelques conséquences sur les formes normales :

- ▷ Il n'y a qu'une seule forme normale ayant un type $\forall \alpha \alpha \rightarrow \alpha$.
- ▷ Il n'y a que deux formes normales ayant un type $\forall \alpha \alpha \rightarrow \alpha \rightarrow \alpha$: $\lambda u. \lambda v. u$ et $\lambda u. \lambda v. v$.

On a aussi quelques « théorèmes gratuits », comme par exemple : si $f : \alpha \rightarrow \beta$ est croissante vis à vis de $\text{ordre}_\alpha : \alpha \rightarrow \alpha \rightarrow \text{bool}$ et $\text{ordre}_\beta : \beta \rightarrow \beta \rightarrow \text{bool}$, alors

$$\text{map } f (\text{sort } \text{ordre}_\alpha \text{ } l) = \text{sort } \text{ordre}_\beta (\text{map } f \text{ } l).$$

4.6 Polymorphisme et inférence de types.

Pour l'inférence de types, on se place dans la représentation implicite du typage polymorphique. On s'intéresse à la question suivante : pour M un λ -terme donné, existe-t-il Γ et A tels que $\Gamma \vdash M : A$?

Cependant, il y a un problème :

Théorème 4.2 (1992). L'inférence de types pour le typage polymorphique implicite est *indécidable*.

Ceci conclut l'étude du Système F. On va maintenant s'intéresser à une variante de système F développée en 1978, le *λ -calcul avec schéma de types*. On note ρ le schéma de types définis par :

$$A ::= \alpha \mid A \rightarrow B \qquad \rho ::= A \mid \forall \alpha \rho.$$

Remarque 4.1. Il n'y a pas de types de la forme $(\forall \alpha \alpha \rightarrow \alpha) \rightarrow (\forall \beta A)$. La quantification est *préfixe*.

Les termes sont donnés par la grammaire :

$$M ::= x \mid \lambda x. M \mid M N \mid \text{let } x = M \text{ in } N.$$

Pour le système de types, on considère Γ un dictionnaire sur (variables du λ -calcul x , schémas de types ρ). La relation de typage est notée $\Gamma \vdash M : A$ où A est un type *simple*. On la définit par induction à l'aide des règles :

$$\begin{array}{c} \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \\[10pt] \frac{\Gamma \vdash M : A \quad \Gamma, x : \forall \vec{\alpha} A \vdash N : B}{\Gamma \vdash \text{let } x = M \text{ in } N : B} \qquad \frac{\Gamma(x) = \forall \vec{\alpha} A}{\Gamma \vdash x : A[\vec{B}/\vec{\alpha}]} \end{array}$$

Ici, on note $\forall \vec{\alpha} A$ pour $\forall \alpha_1 \cdots \forall \alpha_n A$ avec possiblement aucun \forall . De même, la substitution $A[\vec{B}/\vec{\alpha}]$ se fait simultanément.

La règle d'**instantiation** de système F se retrouve partiellement dans la règle de l'axiome. La règle d'**généralisation** de système F se retrouve uniquement dans la règle du `let... = ... in ...`, et uniquement après le `in`.

Exemple 4.5. Le terme

$$\text{let } f = \lambda x. x \text{ in } f$$

a pour type $\beta \rightarrow \beta$, sans généralisation !

$$\frac{\frac{x : \alpha \vdash x : \alpha}{\emptyset \vdash \lambda x. x : \alpha \rightarrow \alpha} \quad \frac{f : \forall \alpha \alpha \rightarrow \alpha \vdash f : \beta \rightarrow \beta}{\emptyset \vdash \text{let } f = \lambda x. x \text{ in } f : \beta \rightarrow \beta}.$$

En OCaml, lorsqu'on essaie de typer

```
(fun z -> z) (fun x -> x)
```

on se rend compte que l'on n'a pas un type `'a -> 'a`, mais

```
'_weak1 -> '_weak1.
```

Théorème 4.3. Il existe un algorithme correct et complet pour l'inférence de types pour les schémas de types.

Pour ce faire, on fait comme en **Théorie de la Programmation** [Chapitre 7], où l'on part de M , on génère des contraintes, et à l'aide de l'algorithme d'unification, on trouve le type « le plus général ».

5 Types linéaires.

Le but des types linéaires est de *contrôler l'usage des ressources*. Avec un tel système de typage, on pourrait :

- ▷ désallouer une zone mémoire, fermer un canal/un fichier ; on se limite à un seul usage : les deux termes suivants ne sont pas typables
 - $(\text{free } x, \text{free } x)$
 - $(\lambda y. \lambda z. (\text{free } y, \text{free } z)) x x$
- ▷ l'initialisation d'une zone mémoire ; au moins un usage
- ▷ si on sait au plus un usage, alors on peut désallouer après un usage.

On se donne la syntaxe suivante.

- ▷ Les booléens sont :

$$b ::= \text{true} \mid \text{false}.$$

- ▷ Les termes sont :

$$\begin{aligned} M, N ::= & \eta b \mid \eta(M, N) \mid \eta \lambda x. M \\ & \mid \text{if } b \text{ then } M \text{ else } N \mid \text{let}(x, y) = M \text{ in } N \mid M N \\ & \mid x \mid \text{let } x = M \text{ in } N. \end{aligned}$$

- ▷ Les usages sont :

$$\eta ::= \text{lin} \mid \text{un}.$$

Les *usages* représentent une utilisation *linéaire* (lin) ou non restrictive (*unrestricted*). Un usage lin est utilisé exactement *une* fois.

On s'autorise le filtrage sur les couples au lieu de **fst** M et **snd** M car, dans le cas linéaire, on ne pourrait utiliser qu'une des deux composantes... zut !

On accepte, par exemple, $\text{lin}(\text{lintrue}, \text{linfalse})$ ou $\text{un}(\text{lintrue}, \text{unfalse})$.

Les types sont définis par la grammaire

$$T, U ::= \text{bool} \mid A \rightarrow B \mid A * B \quad A, B ::= \eta T.$$

Les T, U sont des *prétypes*, les A, B sont les *types*.

La séparation dans la grammaire permet de rejeter une expression comme $\text{un}(\text{lin bool} * \text{un bool})$.

On se donne les règles de typages pour cette version linéaire du λ -calcul.

Première tentative.

On propose la règle pour les couples linéaires :

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : \text{lin}(A * B)}.$$

Cependant, on ne mets pas de restrictions sur A : on peut avoir $A = \text{lin}T$ ou $A = \text{un}T$... Zut !

Seconde tentative.

On propose une règle pour les couples

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B \quad \eta(A) \quad \eta(B)}{\Gamma \vdash (M, N) : \eta(A * B)},$$

où l'on définit le prédicat $\eta(A)$ par :

- ▷ si $\eta = \text{un}$ alors A ne peut pas être $\text{lin}T$;
- ▷ si $\eta = \text{lin}$ alors A peut être quelconque.

Puis, on se donne une règle pour le filtrage

$$\frac{\Gamma \vdash M : \eta(A_1 * A_2) \quad \Gamma, x : A_1, y : A_2 \vdash N : B}{\Gamma \vdash \text{let}(x, y) = M \text{ in } N : B}.$$

Ici, η n'a pas d'importance. On peut donc typer

$$\text{let}(x, y) = \text{un}(\text{lintrue}, \text{unfalse}) \text{ in } \text{lin}(x, x),$$

mais on ne veut pas pouvoir le faire!

Tentative finale.

Définition 5.1. On définit $\Gamma_1 \circ \Gamma_2$ est défini comme « $\Gamma_1 \cup \Gamma_2$ » à ceci près que :

- ▷ si $\Gamma_1(x) = \text{lin}T$ alors $x \notin \text{dom}(\Gamma_2)$;
- ▷ si $\Gamma_2(x) = \text{lin}T$ alors $x \notin \text{dom}(\Gamma_1)$;
- ▷ si $x \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)$ alors il existe T tel que $\Gamma_1(x) = \Gamma_2(x) = \text{un}T$.

Les vraies règles de typage sont :

$$\frac{\Gamma_1 \vdash M : A \quad \Gamma_2 \vdash N : B \quad \eta(A) \quad \eta(B)}{\Gamma_1 \circ \Gamma_2 \vdash (M, N) : \eta(A * B)}$$

$$\frac{\Gamma_1 \vdash M : \eta(A * B) \quad \Gamma_2, x : A, y : B \vdash N : C}{\Gamma_1 \circ \Gamma_2 \vdash \text{let}(x, y) = M \text{ in } N : C}$$

$$\frac{\text{un}(\Gamma)}{\Gamma, x : A \vdash x : A} \quad \frac{\text{un}(\Gamma)}{\Gamma \vdash \eta b : \eta \text{bool}}$$

$$\frac{\Gamma_1 \vdash M : \eta \text{bool} \quad \Gamma_2 \vdash N_1 : A \quad \Gamma_2 \vdash N_2 : A}{\Gamma_1 \circ \Gamma_2 \vdash \text{if } M \text{ then } N_1 \text{ else } N_2 : A}$$

$$\frac{\Gamma, x : A \vdash M : B \quad \eta(\Gamma)}{\Gamma \vdash \eta\lambda x. M : \eta(A \rightarrow B)}$$

$$\frac{\Gamma_1 \vdash M : \eta(A \rightarrow B) \quad \Gamma_2 \vdash N : A}{\Gamma_1 \circ \Gamma_2 \vdash M N : B}$$

On note ici $\text{un}(\Gamma)$ (et éventuellement $\eta(\Gamma)$) le prédicat disant que Γ le contient des types que de la forme $\text{un}T$.

Dans le cas de l'axiome, on assure que l'on n'a pas de variables inutilisées. Dans le cas de l'abstraction, on contrôle l'usage des variables qui se retrouvent dans la clôture associée à $\lambda x. M$.

Exercice 5.1. Peut-on trouver un terme ayant pour type

1. $\text{un}(\text{unbool} \rightarrow \text{linbool})$?
2. $\text{un}(\text{linbool} \rightarrow \text{unbool})$?

On ne peut pas proposer :

1. $\text{un}(\lambda x. x)$ car on n'a pas de lin
2. $\text{un}(\lambda x. \text{unfalse})$ car on n'utilise pas le x .

Oui, on peut :

1. $\text{un}(\lambda x. \text{linfalse})$
2. $\text{un}(\lambda x. \text{if } x \text{ then unfalse else untrue})$

Remarque 5.1 (Obligation de linéarité). On a des règles de « bonne formation » : $\eta(A)$ et $\eta(\Gamma)$. La décomposition $\Gamma_1 \circ \Gamma_2$ correspond à un aiguillage.

Les types linéaires correspondent à des *types substructurels*. Ceci est relié à la *logique linéaire*.

Lemme 5.1. \triangleright *Affaiblissement.* Si $\Gamma \vdash M : A$ alors pour $x \notin \text{dom}(\Gamma)$ on a $\Gamma, x : \text{un}T \vdash M : A$.

▷ *Contraction.* Si $\Gamma, x : \text{un}T, y : \text{un}T \vdash M : B$ alors on a $\Gamma, x : \text{un}T \vdash M[x/y] : B$. Ceci n'est pas vrai avec lin .