Logique et λ-calcul

Cours du M1 STIC ISC

Tom Hirschowitz

1 Bases mathématiques

Dans cette partie, on introduit certaines notions mathématiques de base nécessaires au développement du cours. Il est déconseillé de les lire en première lecture et conseillé de les consulter au besoin, au fil des références qui y sont faites plus bas.

1.1 Catégories

1.2 Systèmes de transitions

Dans cette partie, on introduit la notion de système de transitions étiquetées, ainsi que la notion de bisimulation. Les systèmes de transitions étiquetées sont une bonne notion abstraite de système dynamique, où on a un ensemble d'états, et, de chaque état, un ensemble de transitions possibles vers les autres états. La bisimulation est l'outil standard pour comparer le comportement dynamique de deux états dans un système de transitions. Elle peut être utilisée aussi pour comparer deux systèmes de transitions. Par exemple, on considère en général qu'une fonction $f : G \to H$ entre deux systèmes de transitions préserve la dynamique quand son graphe, i.e., la relation R définie par $x \mathrel{R} y$ ssi $y = f(x)$, est une bisimulation.

Avant de parler d'équivalence dynamique, commençons par la notion de système dynamique qui va nous occuper.

Définition 1. Un *graphe réflexif* est un diagramme

$$E \underset{t}{\overset{s}{\underset{\longleftarrow}{\overset{\longrightarrow}{\longleftarrow}}}} V$$

de fonctions, tel que $s \circ e = t \circ e = \mathrm{id}_V$.

E est un ensemble d'*arêtes* dont s et t donnent respectivement la *source* et le *but* parmi l'ensemble V des *sommets*. La fonction e désigne, pour chaque sommet $v \in V$, une arête $e(v) \in E$, de source et de but v, dite *arête identité* sur v.

Un morphisme $f : G \to G'$ entre deux graphes réflexifs G et G' est une paire de fonctions $f_E : G(E) \to G'(E)$ et $f_V : G(V) \to G'(V)$ (où $G(E)$ désigne l'ensemble des arêtes de G, et ainsi de suite), tel que les diagrammes obtenus à partir de

$$\begin{array}{ccc} G(E) & \overset{s}{\underset{t}{\overset{\longleftarrow e \longrightarrow}{}}} & G(V) \\ \downarrow{\scriptstyle f_E} & & \downarrow{\scriptstyle f_V} \\ G'(E) & \overset{s}{\underset{t}{\overset{\longleftarrow e \longrightarrow}{}}} & G'(V) \end{array}$$

en prenant respectivement i, s et t pour G et G' commutent.

Proposition 1. Les graphes réflexifs et les morphismes entre eux forment une catégorie RGph, au sens de la partie 1.1.

Définition 2. Un *système de transitions étiquetées* est un morphisme $G \to A$ de graphes réflexifs.

La catégorie des systèmes de transitions étiquetées sur A est la catégorie tranche RGph/A.

Le graphe A s'appelle l'*alphabet*. On omet souvent l'adjectif «étiqueté».

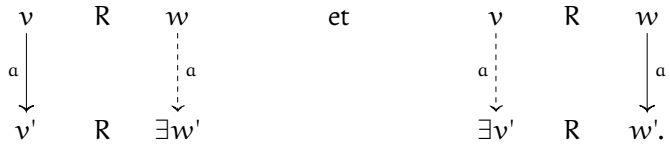
On peut maintenant aborder la notion de bisimulation.

Soit $p : G \to A$ un système de transitions.

Définition 3. Une *bisimulation* est une relation $R : G(V) \nrightarrow G(V)$ binaire entre les sommets de G, telle que

- pour tous $v, v' \in G$, $v \,R\, v'$ implique $p(v) = p(v')$;
- pour toute arête $a \in A(E)$, $(R^{op}; \xrightarrow{a}) \subseteq (\xrightarrow{a}; R^{op})$,
- pour toute arête $a \in A(E)$, $(R; \xrightarrow{a}) \subseteq (\xrightarrow{a}; R)$.

Pour comprendre la définition, il faut un peu d'explications. D'abord, on note les relations avec des flèches $\nrightarrow$, au lieu de $\to$. Ensuite, on écrit $v \xrightarrow{a} v'$ pour dire qu'il existe une arête $e : v \to v'$ de G, telle que $p(e) = a$. On dit que a est l'*étiquette* de e. Ensuite, les deuxième et troisième points postulent des inclusions de relations. On note R^{op} la relation *opposée* à la relation R, i.e., telle que $x \,R^{op}\, y$ ssi $y \,R\, x$. On note aussi $R;S$ la *composée* des relations R et S, définie par $x \,(R;S)\, z$ ssi il existe y tel que $x \,R\, y$ et $y \,S\, z$. Les relations, munies de cette notion de composition, forment une catégorie et même une *allégorie* (une catégorie avec plein de propriétés), au sens de Peter Freyd. Les points 2 et 3 se visualisent ainsi:



Par exemple, pour le diagramme de gauche: on a $w \, R^{op} \, v \xrightarrow{a} v'$; le point 2 demande l'existence d'un w' tel que $w \xrightarrow{a} w' \, R^{op} \, v'$.

On donne un exemple ci-dessous.

On peut étendre la notion de bisimulation aux relations $R : G(V) \nrightarrow G'(V)$ entre systèmes de transitions sur un alphabet A. Chaque relation R de ce type engendre une relation entre les sommets du graphe coproduit $G + G'$. Ce graphe a pour sommets l'union disjointe de ceux de G et G': $(G + G')(V) = G(V) + G'(V)$, où $X + Y = (\{0\} \times X) \cup (\{1\} \times Y)$, pour tous ensembles X et Y. De la même manière, $(G + G')(E) = G(E) + G'(E)$ et les fonctions s, t, e suivent naturellement. R induit une relation $R' : (G + G') \nrightarrow (G + G')$ définie par $x \, R' \, y$ ssi $x = (0, v)$, $y = (1, w)$ et $x \, R \, y$.

Définition 4. On dit que R est une bisimulation ssi R' en est une.

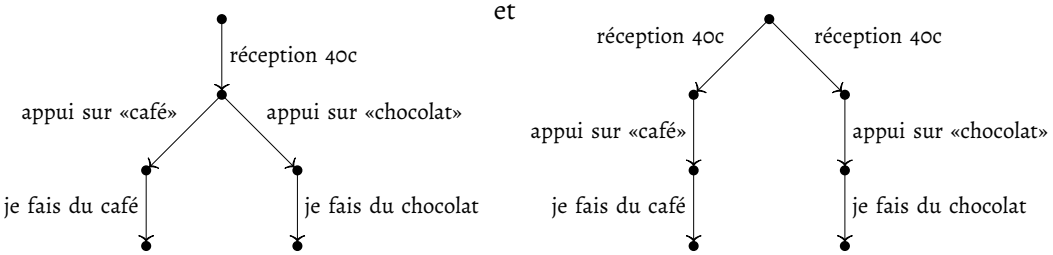
Proposition 2. Les bisimulations sont stables par unions arbitraires.

Preuve. Facile. □

Corollaire 1. Pour tout système de transitions étiquetées $p : G \to A$, l'union de toutes les bisimulations est encore une bisimulation, notée $\sim$ et appelée la *bisimilarité*.

On a donc que deux sommets de G sont *bisimilaires*, i.e., reliés par $\sim$, ssi il existe une bisimulation R les reliant.

Exemple 1. La bonne et la mauvaise machine à café.



Quel graphe vous paraît modéliser de plus près le comportement d'une machine à café?

Une *simulation*, c'est comme une simulation, sans le point 3 de la définition. Du coup, une bisimulation est une simulation dont l'opposée est une simulation. Montrer que la

bonne machine à café simule la mauvaise, mais pas l'inverse. Moralement, le problème est qu'il arrive à la mauvaise machine à café de ne pas accepter l'appui sur le bouton «chocolat», alors qu'elle a accepté 40c.

Cet exemple illustre, entre autres choses, pourquoi on ne choisit pas comme notion d'équivalence dynamique la notion suivante. Pour un sommet $x \in G(V)$, soit $\downarrow x$ l'ensemble des chemins ρ dans A tels qu'il existe un chemin μ dans G partant de x et tel que $p(\mu) = \rho$. Ces chemins sont appelés des *traces* de x.

Définition 5. Deux sommets x et y de G sont *trace-équivalents* ssi $\downarrow x = \downarrow y$.

On appelle ça l'*équivalence de trace* et on la note $x \sim_t y$. Si on prend pour x et y les deux machines à café candidates ci-dessus, on a bien $x \sim_t y$, alors que leurs comportements dynamiques sont très différents.

La question de savoir quelle est la meilleure notion d'équivalence dynamique dépend beaucoup des contextes et présente souvent des subtilités. Dans le domaine des *calculs de processus*, par exemple, qui sont une adaptation des idées du λ-calcul à un cadre parallèle, voire concurrent, cette question est encore aujourd'hui le sujet de nombreuses recherches.

On se contentera d'en mentionner encore une, parce qu'elle nous sert ci-dessous. Un peu de notation d'abord: pour $p : G \to A$, on note $x \overset{a}{\Longrightarrow} y$ pour l'existence d'un chemin $\rho : x \rightsquigarrow y$ dans G dont l'image par p comprend l'arête a, éventuellement entourée d'arêtes identité:

$$p(x) \xrightarrow{e(p(x))} \dots \xrightarrow{e(p(x))} p(x') \xrightarrow{a} p(y') \xrightarrow{e(p(y))} \dots \xrightarrow{e(p(y))} p(y)\,.$$

Ça impose en particulier que $p(x) = p(x')$ et $p(y') = p(y)$. De la même manière, on note $x \overset{\hat{a}}{\Longrightarrow} y$ pour l'existence d'un chemin comme ci-dessus, sauf que si a est une arête identité, alors le chemin peut-être vide (constitué d'un seul sommet).

Maintenant qu'on a vu dans l'exemple 1 la définition équivalente des bisimulations en termes de simulations, on ne s'en prive pas.

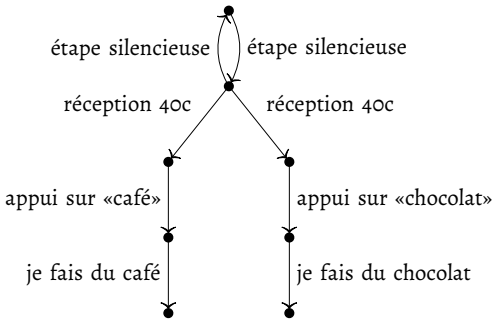
Définition 6. Une *simulation faible* est une relation $R : G(V) \nrightarrow G(V)$ binaire entre les sommets de G, telle que

- pour tous $v, v' \in G$, $v \,R\, v'$ implique $p(v) = p(v')$;
- pour toute arête $a \in A(E)$, $(R^{op}; \overset{a}{\Longrightarrow}) \subseteq (\overset{a}{\Longrightarrow}; R^{op})$.

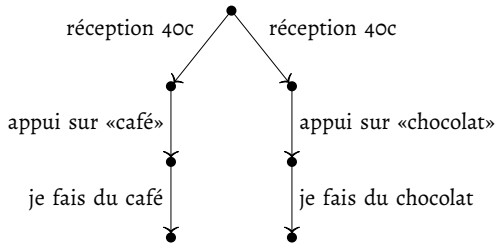
Une *bisimulation faible* est une simulation R dont l'opposée R^{op} est aussi une simulation.

L'idée de cette notion faible de bisimulation est qu'on veut ignorer certaines étapes «silencieuses» de calcul, tant qu'elles ne modifient pas le comportement «visible».

Exemple 2. Les deux comportements



et



sont faiblement bisimilaires, mais pas fortement.

2 Calcul

2.1 Premiers pas, entiers de Church, paires

On peut, en première approximation, faire semblant que le λ-calcul c'est trivial. Commençons par là. On fixe d'abord un ensemble χ de *variables*, infini. On définit l'ensemble Λ des λ-termes par

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

où $x \in \chi$.

Pour éviter un trop plein de parenthèses, on définit des priorités. L'application $M\,N$ est par convention associative à gauche, c'est-à-dire que, par exemple $M\,N\,P$ signifie en réalité $(M\,N)\,P$. De même, on considère que l'application a priorité sur l'abstraction λ: par exemple, $\lambda x\,.\,M\,N$ signifie $\lambda x\,.\,(M\,N)$. On n'appliquera pas souvent cette convention, qui est moins facile à se rappeler.

On définit ensuite la *relation de β-réduction* par la règle

$$(\lambda x\,.\,M)\,N \to_\beta M[x \mapsto N],$$

où $M[x \mapsto N]$ est défini comme M, avec x remplacé par N. On décrète qu'on peut appliquer cette règle n'importe où dans le terme. Formellement $\to_\beta$ est la plus petite relation contenant la règle ci-dessus et telle que

$$\frac{M \to_\beta M'}{M\,N \to_\beta M'\,N} \qquad \frac{N \to_\beta N'}{M\,N \to_\beta M\,N'} \qquad \frac{M \to_\beta M'}{\lambda x\,.\,M \to_\beta \lambda x\,.\,M'}.$$

C'est le sens de ce «remplacé» qui pose problème. Dans cette partie, on se contente d'apprendre à calculer avec ça. On formalisera un peu plus dans la partie suivante.

Le sens de la règle β est qu'appliquer une fonction $x \mapsto M$ à un argument N consiste à remplacer x par N dans M.

Pour comprendre pourquoi c'est pas forcément évident, que doit valoir à votre avis $(\lambda y\,.\,x)[x \mapsto y]$? Ici, on suppose que x et y sont deux variables distinctes. Le terme $\lambda y\,.\,x$ désigne la fonction constante égale à x. Remplacer x par y dedans devrait donner la fonction constante égale à y. Or, si on se contente de remplacer textuellement x par y, on obtient plutôt $\lambda y\,.\,y$, la fonction identité.

Comment faire alors? Il faut comprendre que la variable y dans $\lambda y\,.\,x$, est indifférente : on pourrait choisir n'importe quelle autre variable sans changer le sens de la fonction. Presque n'importe quelle autre variable en fait : si on choisit x on tombe sur l'identité.

De manière générale, on a envie de dire que $\lambda x\,.\,M$ est équivalent à $\lambda y\,.\,(M\,[x \mapsto y])$, pour n'importe quel y. On est bien embêtés, parce que, en essayant de définir la substitution $M\,[x \mapsto y]$, on en arrive à une équivalence qui dépend elle-même de la substitution.

On verra en partie suivante une solution pour éviter ce raisonnement cyclique. Ici, contentons-nous de donner une recette pour calculer la substitution. On commence par remarquer que, chaque terme étant fini, il ne fait intervenir qu'un nombre fini de variables. Or χ est infini. On a donc:

Lemme 1. Pour tout λ-terme M, il existe une variable x n'apparaissant pas dans M. On écrit $x \notin M$ par abus de notation. Le résultat marche évidemment aussi pour les ensembles finis de termes.

On a de plus que si M est un sous-terme de N et $x \notin N$, alors $x \notin M$.

La recette pour calculer $M[x \mapsto N]$ est alors la suivante, par induction sur M:

$$\begin{aligned}
x\,[x \mapsto N] &= N \\
y\,[x \mapsto N] &= y && \text{si } x \neq y \\
(M_1\,M_2)[x \mapsto N] &= M_1[x \mapsto N]\,M_2[x \mapsto N] \\
(\lambda y\,.\,M')[x \mapsto N] &= \lambda z\,.\,(M'[y \mapsto z][x \mapsto N]) && \text{pour } z \notin \{x, M', N\}.
\end{aligned}$$

Ainsi, par exemple, $(\lambda y\,.\,x)[x \mapsto y] = \lambda z\,.\,y$.

Exercice 1. Réduire autant que possible:

- $(\lambda x\,.\,x)\,\lambda y\,.\,y$,
- $(\lambda f.\,\lambda x\,.\,(f\,(f\,(f\,x))))\,(\lambda y\,.\,y)\,z$, qu'on pourrait noter selon nos conventions $(\lambda f.\,\lambda x\,.\,f\,(f\,(f\,x)))\,(\lambda y\,.\,y)\,z$,
- $(\lambda f.\,(f\,x\,y))\,(\lambda x\,.\,\lambda y\,.\,x\,y)\,(\lambda z\,.\,z)\,u$,
- $(\lambda x\,.\,x\,x)(\lambda x\,.\,x\,x)$.

Si vous avez réussi le dernier point de l'exercice précédent, vous avez dû faire une erreur. Le terme donné est censé se réduire indéfiniment sur lui-même.

Les entiers naturels admettent un codage standard dans Λ: on définit

$$\overline{n} = \lambda f.\, \lambda x.\, (f \ldots f\, x)$$

où f est appliquée n fois.

Par exemple, $\overline{3} = \lambda f.\, \lambda x.\, (f\,(f\,(f\,x)))$.

Exercice 2. Définir la somme, avec le cahier des charges suivant: c'est un terme S tel que pour tous entiers m et n, on ait $S\,\overline{m}\,\overline{n} \to_\beta^\star \overline{m+n}$, où $\to_\beta$ est la clôture réflexive transitive de $\to_\beta$. Vérifier (faire le calcul) sur $2+3$. Réponse: $\lambda m.\, \lambda n.\, \lambda f.\, \lambda x.\, (m\,f\,(n\,f\,x))$.

Exercice 3. Définir Paire, π, π', tels que pour tous M et N, $\pi\,(\text{Paire}\,M\,N) \to_\beta M$ et $\pi'\,(\text{Paire}\,M\,N) \to_\beta N$.

2.2 Types algébriques, premier passage

On peut généraliser les deux exercices précédents pour coder tous les *types de données algébriques*, ou *inductifs*, en λ-calcul. Qu'est-ce que c'est? On peut voir les entiers comme le type OCaml suivant:

```
type nat = Z | S of nat
```

dont les valeurs sont (presque) toutes de la forme `S (...(S Z)...)`. (En fait, en OCaml, on peut tricher et définir `let rec x = S x`.)

Un type de données algébrique, en première approximation, est un type OCaml de la forme

```
type t = A1 of T1 | ... | An of Tn
```

où chaque `Ti` est un produit de `t`. Par exemple, les entiers ci-dessus sont de cette forme.

Les paires en revanche, ne le sont pas tout-à-fait. Si `u` et `v` sont deux types, on peut voir les paires comme le type

```
type t = A of u * v
```

Mais pour entrer dans le moule, il faut que `u` et `v` soient des produits de `t`, ce qui n'est a priori pas le cas. Par exemple, si `u` et `v` sont le type `nat` ci-dessus, il n'est pas du tout évident que le type `nat * nat` des paires d'entiers est algébrique au sens ci-dessus. On pourrait essayer avec un constructeur `S` pour l'entier de gauche dans la paire, et un pour celui de droite. Ça donnerait

```
type paire_nat = ZZ | Sgauche of t | Sdroit of t
```

mais c'est un peu foireux: on a deux façons de coder la paire `(1,1)`, `Sgauche (Sdroit ZZ)` et `Sdroit (Sgauche ZZ)`.

On généralise donc un peu la notion de type algébrique. On dit maintenant que c'est l'un des types définis par une déclaration de types récursifs de la forme

```
type t1 = A1_1 of T1_1 | ... | A1_n of T1_n1
and
...
and
tp = Ap_1 of Tp_1 | ... | Ap_np of Tp_np
```

où chaque Ti_j est un produit de tk. Les paires d'entiers sont par exemple définissables par

```
type nat = Z | S of nat
and paire_nat = Paire of nat * nat
```

On peut définir de nombreux types utiles dans ce cadre. Par exemple, les arbres binaires à feuilles entières:

```
type nat = ... (* comme avant *)
and arbre = Feuille of nat | Noeud of arbre * arbre
```

Comme ci-dessus, en OCaml on peut toujours tricher et définir `let rec x = Noeud (x,x)`, mais dans l'idée on est plutôt en train de définir le type des arbres.

En λ-calcul, on sait coder tout ces types algébriques. Il faudra expliquer précisément ce que ça signifie, mais avant j'ai envie d'expliquer comment on fait.

Le point de départ est de se demander ce que peut être une fonction dont le domaine est l'un de ces types algébriques. Par exemple, pour les entiers, les fonctions qui nous intéressent sont celles de la forme

```
let rec f = function
| Z -> ...
| S x -> ... (f x) ...
```

Le premier «...» ici est n'importe quelle valeur du type d'arrivée. Le second est n'importe quelle valeur du type d'arrivée, pouvant dépendre du résultat de `f x`. Si le type d'arrivée est `T`, une telle fonction est donc entièrement déterminée par une valeur de type `T * (T -> T)`, i.e., une paire `(a,g)`.

Pour coder ça en λ-calcul, l'idée est d'inverser le rapport entre données et fonctions. Les entiers vont devenir des fonctions. Et pour appliquer la fonction `f` ci-dessus à un entier, disons `x`, on invoquera `x f`, autant dire le monde à l'envers. Comme `f` a pour type `T * (T -> T)`, `x` doit avoir pour type quelque chose d'équivalent à `T * (T -> T) -> T`. De plus, cette manipulation doit marcher pour tout `T`, donc `x` doit avoir pour type $\forall X.\ (X\times (X \to X) \to X)$, en un sens à préciser. Un type équivalent est le type $\forall X.\ (X \to X) \to X \to X$.

L'application de `f` à `Z` doit donner `a` et l'application à une valeur de la forme `S v` doit donner `g (f v)`. Zéro doit donc être la fonction `(a,g)` $\mapsto$ `a`, alors que `S v` devient la fonction `(a,g)` $\mapsto$ `g (v (a, g))`. Cette idée mène directement à représenter zéro par le terme $\lambda g.\ \lambda a.\ a$, i.e., $\lambda f.\ \lambda x.\ x$. Pour le successeur, c'est moins clair. L'application

de `f` à `S Z` doit donner `g a`. L'application à `S (S Z)` doit donner `g (f (S Z))`, i.e., `g (g a)`. Par induction, on arrive au fait que l'application au codage de n est l'application itérée n fois de `g` à `a`. On veut donc coder n par $\lambda g.\,\lambda a.\,(g\,\ldots\,(g\,a)\ldots)$, i.e., $\lambda f.\,\lambda x.\,(f\ldots(f\,x)\ldots)$.

Tentons maintenant d'appliquer cette idée pour coder les paires. On repart de l'idée de disséquer les fonctions sur les paires. Une fonction `T * T' -> U`, c'est la même chose qu'une fonction `T -> T' -> U`. Une paire, dans notre monde à l'envers, doit donc attendre sa fonction et l'appliquer aux deux composantes: la paire `(a,b)` devient la fonction `f  f a g`, i.e., en λ-calcul, la paire $\langle M, N\rangle$ devient $\lambda f.\,f\,M\,N$. Le pseudo-type décrivant ça est $\forall X.\,(T \to T' \to X) \to X$.

Essayons maintenant de combiner les deux. Par exemple, reprenons le type

```
type arbre = Feuille | Noeud of arbre * arbre
```

Une fonction de ce type vers un type `U` consiste en l'image de `Feuille`, plus, pour le cas `Noeud`, une fonction prenant en argument les deux sous-arbres. On obtient donc le pseudo-type $\forall X.\,(X \to (X \to X \to X) \to X)$ et, par exemple, le codage de `Noeud (Noeud (Feuille,Feuille), Feuille)` est $\lambda x.\,\lambda f.\,f\,(f\,x\,x)\,x$. Il peut être instructif de dessiner ce dernier terme comme un arbre.

On aimerait généraliser ça aux définitions de types mutuellement récursives, mais c'est significativement plus compliqué, donc on laisse tomber pour l'instant.

2.3 Confluence

Dans les parties précédentes, on a programmé un peu en λ-calcul, sans trop se poser de question. On va ici s'en poser une premère sérieuse: est-ce qu'un programme a toujours le même résultat?

Formellement, on pose:

Définition 7. Un λ-terme M est une *forme normale* ssi il ne se réduit pas, i.e., il n'existe pas de N tel que $M \to_\beta N$.

Une première question à se poser est: un M donné admet-il toujours au moins une forme normale?

Exercice 4. Montrer que non. Par exemple, montre qu'il existe un M tel que $M \to (M\,M)$. En fait, on peut même trouver un terme Y tel que pour tout T, $Y\,T$ est un point fixe de T, i.e., $(Y\,T) \to^\star T\,(Y\,T)$.

Par exemple, soit, pour toute terme T, $\delta_T = \lambda x.\,(T\,(x\,x))$ et $Y = \lambda f.\,(\delta_f\,\delta_f)$. On a $Y\,T \to \delta_T\,\delta_T \to T\,(\delta_T\,\delta_T)$. Bon, ici $Y\,T$ se réduit en une étape sur le point fixe $\delta_T\,\delta_T$. Mais il en existe des vrais de vrais.

Ces termes sans forme normale sont à l'origine des paradoxes en logique, comme on le verra plus loin.

Une second question qu'on se pose ici est: pour un M donné, peut-il exister deux formes normales différentes N_1 et N_2, telles que $N_1 {}^{\star}\!\leftarrow M \to^{\star} N_2$? On va répondre par la négative, mais ça va demander un peu de boulot. Le résultat, plus fort, qu'on va en fait démontrer est:

Théorème 1. Pour tous $N_1 {}^{\star}\!\leftarrow M \to^{\star} N_2$ (avec N_1 et N_2 pas forcément en forme normale) il existe N tel que $N_1 \to^{\star} N {}^{\star}\!\leftarrow N_2$, qui a l'autre pour corollaire.

On commence par montrer le cas à une étape, qui s'appelle *confluence locale*.

Lemme 2. Pour tous $N_1 \leftarrow M \to N_2$ il existe N tel que $N_1 \to^{\star} N {}^{\star}\!\leftarrow N_2$.

Preuve. On procède par induction sur M, puis par analyse de cas sur la réduction.

Si M est une variable, il n'y a pas de réduction possible, donc il n'y a rien à démontrer.

Si $M = \lambda x\,.\,M'$, alors les deux réductions vers N_1 et N_2 sont de la forme $\lambda x\,.\,N'_1 \leftarrow M \to \lambda x\,.\,N'_2$, avec $N_1 = \lambda x\,.\,N'_1$, $N_2 = \lambda x\,.\,N'_2$ et $N'_1 \leftarrow M' \to N'_2$. Par hypothèse d'induction, on obtient N' tel que $N'_1 \to^{\star} N' {}^{\star}\!\leftarrow N'_2$ et donc $N_1 \to^{\star} \lambda x\,.\,N' {}^{\star}\!\leftarrow N_2$. Donc $N = \lambda x\,.\,N'$ convient.

Si $M = M_1 M_2$, c'est plus compliqué. Il y a plusieurs cas faciles, qu'on traite d'abord.

Si $N_1 = M'_1 M_2$, avec $M_1 \to M'_1$, et $N_2 = M_1 M'_2$, avec $M_2 \to M'_2$, alors $N = M'_1 M'_2$ convient.

Le cas symétrique est tout aussi facile, ainsi que le cas où $N_1 = N_2$.

Enfin, si $N_1 = M'_1 M_2$ et $N_2 = M''_1 M_2$, avec $M'_1 \leftarrow M_1 \to M''_1$, on a par hypothèse d'induction l'existence d'un P_1 tel que $M'_1 \to^{\star} P_1 {}^{\star}\!\leftarrow M''_1$ et donc $N = P_1 M_2$ convient.

Le cas symétrique est tout aussi facile.

Reste le cas où l'une des deux réductions casse l'application $M_1 M_2$, c'est-à-dire qu'on a $M_1 = \lambda x\,.\,M'$ et, par exemple (le cas symétrique marche pareil), $N_1 = M'[x \mapsto M_2]$. On a alors deux cas.

- Si la réduction $M \to N_2$ a lieu dans M_2, i.e., $N_2 = M_1 M'_2$ avec $M_2 \to M'_2$, alors on a $M'[x \mapsto M_2] \to^{\star} M'[x \mapsto M'_2]$, comme le montre le lemme suivant. On prend donc $M'[x \mapsto M'_2]$ pour N, qui convient puisque $M_1 M'_2$ se réduit aussi dessus.
- Si la réduction $M \to N_2$ a lieu dans M_1, on a $N_2 = (\lambda x\,.\,M'')\,M_2$, avec $M' \to M''$, alors on pose $N = M''[x \mapsto M_2]$ et on a par le lemme suivant que $M'[x \mapsto M_2] \to N \leftarrow N_2$, comme souhaité. □

Lemme 3. Pour tous M, M', N, N' et x, si $M \to M'$ et $N \to N'$, on a

- $M[x \mapsto N] \to^\star M[x \mapsto N']$ et
- $M[x \mapsto N] \to M'[x \mapsto N]$.

Preuve. On commence par le second résultat, par induction sur M. Si M est une variable, c'est impossible. Si $M = \lambda y\,.\,M_1$ (on peut choisir $y \notin \{N, x\}$), alors $M' = \lambda y\,.\,M'_1$, avec $M_1 \to M'_1$. Par hypothèse d'induction, on a $M_1[x \mapsto N] \to M'_1[x \mapsto N]$. Or $(\lambda y\,.\,M_1)[x \mapsto N] = \lambda y\,.\,(M_1[x \mapsto N])$ (grâce au choix de y), ce qui permet de conclure. Si $M = M_1 M_2$, alors si la réduction a lieu dans M_1 ou M_2, on conclut pareil par induction. Si maintenant la réduction est en surface, i.e., $M = (\lambda y\,.\,M_3)M_2$ et $M' = M_3[y \mapsto M_2]$, on calcule un peu les substitutions, en choisissant $y \notin \{N, x\}$, pour s'apercevoir qu'on veut montrer

$$(\lambda y\,.\,(M_3[x \mapsto N]))(M_2[x \mapsto N]) \to (M_3[y \mapsto M_2])[x \mapsto N].$$

Le membre de gauche se réduit à $(M_3[x \mapsto N])[y \mapsto (M_2[x \mapsto N])]$, qui est égal au membre de droite par le lemme suivant.

On démontre maintenant le premier résultat, par induction sur M. Si $M = y \neq x$, alors le résultat revient à $y \to^\star y$, ce qui est ma foi vrai. Si $M = x$, ça revient à $N \to^\star N'$, ce qui est vrai par hypothèse.

Si $M = M_1 M_2$, on a par hypothèse d'induction que $M_i[x \mapsto N] \to^\star M_i[x \mapsto N']$, pour $i \in \{1, 2\}$, donc

$$(M_1 M_2)[x \mapsto N] = (M_1[x \mapsto N])(M_2[x \mapsto N]) \to^\star (M_1[x \mapsto N'])(M_2[x \mapsto N]) \to^\star (M_1[x \mapsto N'])(M_2[x \mapsto N']) = (M_1 M_2)[x \mapsto N'],$$

comme souhaité.

Enfin, si $M = \lambda y\,.\,M'$ (on peut choisir $y \notin \{x, N\}$), on a, en appliquant l'hypothèse d'induction à M':

$$(\lambda y\,.\,M')[x \mapsto N] = \lambda y\,.\,(M'[x \mapsto N]) \to^\star \lambda y\,.\,(M'[x \mapsto N']) = (\lambda y\,.\,M')[x \mapsto N'],$$

ce qui conclut la preuve. □

Lemme 4. Pour tous M, N, P, x et y, si $x \notin \{P, y\}$, alors

$$M[x \mapsto N][y \mapsto P] = M[y \mapsto P][x \mapsto (N[y \mapsto P])].$$

Preuve. On procède par induction sur M. Si $M = z \notin \{x, y\}$, les deux côtés donnent z, on gagne. Si $M = x$, les deux membres valent $N[y \mapsto P]$: on gagne encore (on a utilisé $y \neq x$). Si $M = y$, les deux membres valent P, encore gagné (attention, on a utilisé $y \neq x$ et $x \notin P$).

Si $M = \lambda z\,.\,M'$ (on peut choisir $z \notin \{x, y, N, P\}$), on veut montrer

$$\lambda z\,.\,(M'[x \mapsto N][y \mapsto P]) = \lambda z\,.\,(M'[y \mapsto P][x \mapsto (N[y \mapsto P])]),$$

ce qui est vrai par hypothèse d'induction sur M.

Si $M = M_1 M_2$ on veut montrer

$$(M_1[x \mapsto N][y \mapsto P])(M_2[x \mapsto N][y \mapsto P])$$
$$= (M_1[y \mapsto P][x \mapsto (N[y \mapsto P])])(M_2[y \mapsto P][x \mapsto (N[y \mapsto P])])$$

ce qui est vrai par hypothèse d'induction appliquée à M_1 puis M_2. □

Ceci achève la preuve du lemme 2.

En fait, démontrer le cas général demande un saut conceptuel important: les tentatives naïves par induction sur le nombre d'étapes à gauche et à droite ne passent pas.

Exercice 5. Essayer de démontrer le théorème 1.

Le saut conceptuel dont on a besoin est l'invention de la *réduction parallèle*, qui est définie par les règles suivantes:

$$\frac{M \Rightarrow M' \qquad N \Rightarrow N'}{(\lambda x\,.\,M)\,N \Rightarrow M'[x \mapsto N']} \qquad \frac{}{M \Rightarrow M}$$

$$\frac{M \Rightarrow M' \quad N \Rightarrow N'}{M\,N \Rightarrow M'\,N'} \qquad \frac{M \Rightarrow M'}{\lambda x\,.\,M \Rightarrow \lambda x\,.\,M'}.$$

La différence avec $\to_\beta$ est qu'on peut réduire à plusieurs endroits en une étape.

Exercice 6. Montrer que $((\lambda x\,.\,x)\,y)\,((\lambda x\,.\,x)\,z) \Rightarrow y\,z$ en une étape.

En revanche, ne rêvons pas, on n'a pas l'égalité $(\Rightarrow;\Rightarrow) = \Rightarrow$. La raison intuitive en est que la réduction, même parallèle, peut rendre possible de nouvelles réductions.

Exercice 7. Démontrer qu'on n'a pas

$$(\lambda x\,.\,(x\,y))\,(\lambda z\,.\,z) \Rightarrow y,$$

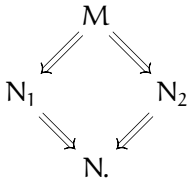
alors qu'on a bien

$$(\lambda x\,.\,(x\,y))\,(\lambda z\,.\,z) \Rightarrow (\lambda z\,.\,z)\,y \Rightarrow y.$$

L'intérêt de la réduction parallèle pour démontrer la confluence est qu'on a la confluence locale *en une étape*, qu'on appelle aussi la *propriété du diamant*:

Lemme 5. (Confluence locale) Pour tous $N_1 \Leftarrow M \Rightarrow N_2$ il existe N tel que $N_1 \Rightarrow N \Leftarrow N_2$.

Ça s'appelle diamant parce que ça se dessine:



Preuve. On procède par induction sur M.

Si M est une variable, c'est facile.

Si $M = \lambda x\,.\,M'$, alors les deux réductions vers N_1 et N_2 sont de la forme $\lambda x\,.\,N'_1 \Leftarrow M \Rightarrow \lambda x\,.\,N'_2$, avec $N_1 = \lambda x\,.\,N'_1$, $N_2 = \lambda x\,.\,N'_2$ et $N'_1 \Leftarrow M' \Rightarrow N'_2$. Par hypothèse d'induction, on obtient N' tel que $N'_1 \Rightarrow N' \Leftarrow N'_2$ et donc $N_1 \Rightarrow \lambda x\,.\,N' \Leftarrow N_2$. Donc $N = \lambda x\,.\,N'$ convient.

Si $M = M_1 M_2$, c'est plus compliqué. Comme plus haut, traitons d'abord le cas facile.

Si $N_1 = M'_1 M'_2$ et $N_2 = M''_1 M''_2$, avec $M'_i \Leftarrow M_i \Rightarrow M''_i$, pour $i \in \{1, 2\}$, alors par hypothèse d'induction, pour tout $i \in \{1, 2\}$, on obtient $M'_i \Rightarrow M'''_i \Leftarrow M''_i$ et $N = M'''_1 M'''_2$ convient.

Reste les cas où l'une des deux réductions casse l'application $M_1 M_2$, c'est-à-dire qu'on a $M_1 = \lambda x\,.\,M_3$ et, par exemple (le cas symétrique marche pareil), $N_1 = M'_3[x \mapsto M'_2]$, avec $M_2 \Rightarrow M'_2$ et $M_3 \Rightarrow M'_3$. On a alors trois cas, selon la réduction vers N_2.

- Soit $N_2 = M$ et on conclut facilement.
- Soit $N_2 = (\lambda x\,.\,M''_3)M''_2$ avec $M_i \Rightarrow M''_i$ pour $i \in \{2, 3\}$, auquel cas on a par hypothèse d'induction l'existence de M'''_3 et M'''_2 tels que $M'_i \Rightarrow M'''_i \Leftarrow M''_i$, pour $i \in \{2, 3\}$. On pose alors $N = M'''_3[x \mapsto M'''_2]$. On a directement $N_2 \Rightarrow N$ et, par le lemme suivant, $N_1 \Rightarrow N$.
- Soit $N_2 = M''_3[x \mapsto M''_2]$ avec $M_i \Rightarrow M''_i$ pour $i \in \{2, 3\}$, auquel cas on a par hypothèse d'induction l'existence de M'''_3 et M'''_2 tels que $M'_i \Rightarrow M'''_i \Leftarrow M''_i$, pour $i \in \{2, 3\}$. On pose alors $N = M'''_3[x \mapsto M'''_2]$. On a donc $N_2 \Rightarrow N$ et $N_1 \Rightarrow N$, toujours par le lemme suivant. □

Lemme 6. Si $M \Rightarrow M'$ et $N \Rightarrow N'$, alors, pour toute variable x, $M[x \mapsto N] \Rightarrow M'[x \mapsto N']$.

Preuve. On procède par induction sur M puis par cas sur la démonstration de $M \Rightarrow M'$. Si $M = y \neq x$, alors le résultat revient à $y \Rightarrow y$, ce qui est ben vrai. Si $M = x$, ça revient à $N \Rightarrow N'$, ce qui est vrai par hypothèse.

Si $M = M_1 M_2$ et la réduction vers M' se fait car $M_i \Rightarrow M'_i$, pour $i \in \{1, 2\}$ et $M' = M'_1 M'_2$, alors on a par hypothèse d'induction que $M_i[x \mapsto N] \Rightarrow M'_i[x \mapsto N']$ et donc $M[x \mapsto N] \Rightarrow M'[x \mapsto N']$.

Si $M = M_1 M_2$ et la réduction vers M' se fait car $M_1 = \lambda y\,.\,M_3$, $M_i \Rightarrow M'_i$, pour $i \in \{2, 3\}$ et $M' = M'_3[y \mapsto M'_2]$, alors, en choisissant $y \notin \{x, N\}$, on a $M[x \mapsto N] = (\lambda y\,.\,(M_3[x \mapsto N]))\,(M_2[x \mapsto N])$. En appliquant l'hypothèse d'induction à M_3 et M_2, on obtient

$$M[x \mapsto N] \Rightarrow (M'_3[x \mapsto N'])[y \mapsto (M'_2[x \mapsto N'])].$$

Or, par le lemme 4,

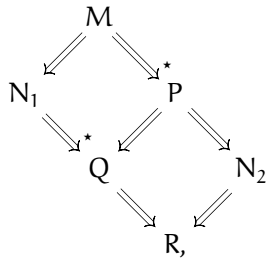
$$M'[x \mapsto N'] = M'_3[y \mapsto M'_2][x \mapsto N'] = (M'_3[x \mapsto N'])[y \mapsto M'_2[x \mapsto N']],$$

qui est exactement le membre droit ci-dessus. □

On peut maintenant démontrer le théorème 1, grâce aux lemmes suivants, des inductions faciles, qui généralisent progressivement la propriété du diamant.

Lemme 7. Pour tous $N_1 \Leftarrow M \Rightarrow^\star N_2$, il existe N tel que $N_1 \Rightarrow^\star N \Leftarrow N_2$.

Preuve. On procède par induction sur la longueur de $M \Rightarrow^\star N_2$. Si c'est 0, c'est trivial. Si c'est $n+1$, on a $M \Rightarrow^\star P \Rightarrow N_2$ et, par hypothèse d'induction, un Q tel que $N_1 \Rightarrow^\star Q \Leftarrow P$. Par la propriété du diamant, on trouve un R tel que $Q \Rightarrow R \Leftarrow N_2$, on a donc

ce qui conclut la démonstration.  □

Lemme 8. Pour tous $N_1 {}^\star\!\Leftarrow M \Rightarrow^\star N_2$, il existe N tel que $N_1 \Rightarrow^\star N {}^\star\!\Leftarrow N_2$.

Preuve. On procède par induction sur la paire n_1, n_2 des longueurs des réductions $N_1 {}^\star\!\Leftarrow M \Rightarrow^\star N_2$, dans l'ordre lexicographique. Si c'est $0, 0$, c'est trivial. Si c'est $n_1, n_2 = n'_1 + 1, n'_2$ et le résultat est vrai pour toutes les paires plus petites, alors on a $N_1 \Leftarrow N_3 {}^\star\!\Leftarrow M \Rightarrow^\star N_2$, donc par hypothèse d'induction il existe P tel que $N_3 \Rightarrow^\star P {}^\star\!\Leftarrow N_2$. □

On conclut la preuve du théorème 1 en constatant que $\rightarrow \subseteq \Rightarrow \subseteq \rightarrow^\star$. En effet, pour tous $N_1 {}^\star\!\leftarrow M \rightarrow^\star N_2$, on a $N_1 {}^\star\!\Leftarrow M \Rightarrow^\star N_2$, donc l'existence de N tel que $N_1 \Rightarrow^\star N {}^\star\!\Leftarrow N_2$

par le lemme précédent. Mais N est du coup tel que $N_1 (\rightarrow^\star)^\star N {}^\star(\!{}^\star\!\leftarrow) N_2$, d'où le résultat.

2.4 Stratégies d'évaluation

Dans la partie précédente, on a vu que toutes les séquences de réductions commencées à partir d'un même terme peuvent se rejoindre. On a appelé ça la confluence. Mais ça ne garantit pas complètement que toutes les stratégies d'évaluation sont équivalentes. D'abord, que peut signifier «équivalentes» ici? On sait par confluence que si un terme M converge, par deux suites de réductions différentes, vers des formes normales M_1 et M_2, on a $M_1 = M_2$. Mais la convergence vers une valeur n'est pas le seul comportement observable pour un λ-terme: il peut aussi *diverger*, c'est-à-dire se réduire indéfiniment sans jamais atteindre de forme normale, comme on l'a vu à l'exercice *1* avec $\Omega = (\lambda x.\, x\, x)(\lambda x.\, x\, x)$. La divergence ou non peut dépendre de la stratégie d'évaluation. Par exemple, $(\lambda x.\, y)\,\Omega$ se réduit en y, mais aussi en lui-même si on essaie de réduire l'argument avant de le passer à la fonction.

Si le but est de trouver une forme normale à coup sûr quand elle existe, on peut adopter la stratégie dite *standard*, qui consiste à réduire le plus à gauche et le plus à l'extérieur possible. Formellement, on définit une sous-relation $\to_{\text{std}} \subseteq \to_\beta$, comme suit. On définit d'abord l'ensemble Neutres des termes *neutres*: c'est les termes qui ne sont pas de la forme $\lambda x.\, M$. On définit aussi l'ensemble NF des formes normales pour $\to_\beta$.

$$\frac{}{(\lambda x.\, M)\,N \to_{\text{std}} M[x \mapsto N]} \qquad \frac{M \to_{\text{std}} M'}{M\,N \to_{\text{std}} M'\,N}\ (\text{si } M \in \text{Neutres}) \qquad \frac{N \to_{\text{std}} N'}{V\,N \to_{\text{std}} V\,N'}\ (\text{si } V \in \text{Neutres} \cap \text{NF}) \qquad \frac{M \to_{\text{std}} M'}{\lambda x.\, M \to_{\text{std}} \lambda x.\, M'}.$$

Exercice 8. Réduire $(\lambda x.\, y)\,\Omega$ selon la stratégie standard.

Proposition 3. La relation $\to_{\text{std}}$ est déterministe: si $M_1 \,{}_{\text{std}}\!\leftarrow M \to_{\text{std}} M_2$, alors $M_1 = M_2$.

Preuve. Par induction sur M. Si M est une variable, facile. Si $M = \lambda x.\, M'$, facile par hypothèse d'induction. Si $M = M'\,M''$, on a plusieurs cas. Si M' n'est pas neutre, on peut appliquer la règle β, mais aucune autre, donc on gagne encore par hypothèse d'induction. Si M' est neutre et se réduit par $\to_{\text{std}}$, alors on ne peut réduire que dans M' et on conclut par hypothèse d'induction. Si M' est neutre et en forme normale, alors, si M'' se réduit, on ne peut réduire que là et on conclut par hypothèse d'induction. Si enfin M'' est en forme normale on ne peut pas réduire et c'est fini. □

Théorème 2. Si M admet une forme normale pour $\to_\beta$, alors $\to_{\text{std}}$ la trouve.

La preuve de ce résultat est compliquée et dépasse le cadre du cours.

En OCaml, le programme

```
(fun x -> 0)((fun x -> Obj.magic x x)(fun x -> Obj.magic x x));;
```

boucle (on est obligé d'utiliser `Obj.magic` pour contourner le problème de typage). C'est donc qu'OCaml n'utilise pas la stratégie standard. En effet, OCaml utilise une stratégie

dite en «appel par valeur de droite à gauche»: on définit l'ensemble Val des *valeurs* qui est constitué des variables et des abstractions, puis la sous-relation $\to_V \subseteq \to_\beta$ définie par

$$\frac{}{(\lambda x\,.\,M)\,V \to_V M[x \mapsto N]}\ (\text{si } V \in \text{Val})$$

$$\frac{N \to_V N'}{M\,N \to_V M\,N'}$$

$$\frac{M \to_V M'}{M\,V \to_V M'\,V}\ (\text{si } V \in \text{Val})$$

On note en particulier qu'on ne réduit pas sous les λ. Par exemple, le programme OCaml

```
(fun y -> ((fun x -> Obj.magic x x)(fun x -> Obj.magic x x)));;
```

ne boucle pas. Cette stratégie est aussi déterministe, mais ne trouve pas forcément la forme normale d'un terme, par exemple $\lambda x\,.\,((\lambda y\,.\,y)\,z)$.

La réduction standard, à laquelle on enlève la règle pour réduire sous les λ, s'appelle l'«appel par nom». On peut définir des fonctions des λ-termes dans eux-mêmes qui traduisent l'appel par valeur en appel par nom, et inversement.

2.5 Un peu de calculabilité

Dans cette partie, on montre que le λ-calcul est Turing puissant, après avoir rappelé ce que ça signifie.

La notion de *calcul*, au sens mathématique, s'est formée progressivement au cours du 20ème siècle. On peut considérer que sa première définition date du début des années 1930 avec l'étude par Church et son étudiant Kleene des fonctions λ-définissables, qui sont en gros les fonctions sur les entiers définissables en λ-calcul. Mais on notera que le λ-calcul est loin de l'architecture actuelle des ordinateurs. C'est qu'il s'est passé bien des choses entre l'invention du λ-calcul, le calcul des fonctions, et celle des langages de programmation fonctionnels en 1958 (LISP). On peut peut-être considérer que, partant d'une notion intuitive de calcul (λ), les scientifiques ont mis du temps pour arriver à une notion applicable, au sens de la construction effective d'une machine: les machines de Turing. Une fois cette construction achevée, les premiers langages pour programmer la machine obtenue sont d'abord restés proches des machines de Turing, avant de revenir à des langages plus proches de λ comme LISP.

En fait, ça ne s'est pas si clairement passé comme ça. La notion de calculabilité a plutôt évolué à partir des questions de décidabilité en logique. En effet, en 1931, Gödel utilise comme outil technique dans sa preuve d'incomplétude de l'arithmétique la notion de *fonction récursive primitive*, qu'on va définir formellement plus bas. Il savait déjà que cette classe de fonctions ne couvrait pas toutes les fonctions «calculables» mécaniquement. Ackermann exhiba d'ailleurs une fonction clairement calculable mais non primitive récursive. Gödel, aidé par Herbrand, proposa donc en 1934 l'ensemble plus grand des *fonctions récursives partielles*, qui fut modifié plus tard par Kleene.

Ce n'est qu'en 1936 que Turing inventa la notion de *machine automatique* qui porte aujourd'hui son nom. Kleene et Turing montrèrent vite que les fonctions *partielles* calculables par une machine de Turing coïncident avec les fonctions récursives partielles de Gödel, ainsi qu'avec les fonctions λ-définissables.

Pour donner une petite idée de ces travaux, on va ici introduire les fonctions récursives partielles et les fonctions λ-définissables, puis énoncer le résultat. Montrer qu'elles coïncident dépasserait le cadre du cours, mais on montrera que toute fonction récursive partielle est λ-définissable.

D'abord les fonctions λ-définissables, c'est facile vu ce qu'on a déjà fait.

On rappelle que $\overline{n}$ désigne le λ-terme codant l'entier n (voir partie 2.1).

Définition 8. Une fonction partielle $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ est λ-définissable ssi il existe un λ-terme M tel que pour tout $(p_1, \ldots, p_n) \in \mathbb{N}^n$, et $k \in \mathbb{N}$,
- $f(p_1, \ldots, p_n) = k$ ssi $M\,\overline{p_1}\ldots\overline{p_n} \to_{std}^{\star} \overline{k}$;
- $f(p_1, \ldots, p_n)$ diverge ssi $M\,\overline{p_1}\ldots\overline{p_n}$ diverge.

Notons bien qu'on utilise ici la réduction standard, celle qui est sûre de trouver la forme normale si elle existe.

Les fonctions récursives ont une définition plus *ad hoc*, par induction. En gros, ce sont les (multi) fonctions contenant les projections, la fonction successeur, la fonction constante égale à 0, la fonction caractéristique de 0, et stables par composition, récursion et minimisation:

Définition 9. L'ensemble des *fonctions récursives* est le plus petit sous-ensemble $\mathcal{R}$ de $\sum_{n\in\mathbb{N}}(\mathbb{N}^n \rightharpoonup \mathbb{N})$ tel que
- pour tout $n \in \mathbb{N}$, et $i \in n$, la $i$ème projection $(x_1, \ldots, x_n) \mapsto x_i$ est dans $\mathcal{R}$;
- la fonction $S : \mathbb{N} \to \mathbb{N}$ définie par $S(n) = n+1$ est dans $\mathcal{R}$;
- la fonction $Z : 1 \to \mathbb{N}$ envoyant l'unique élément de 1 sur 0 est dans $\mathcal{R}$;
- pour tous $n, p \in \mathbb{N}$, $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ et $(f_1, \ldots, f_n) \in (\mathbb{N}^p \rightharpoonup \mathbb{N})^n$, si f et les f_i sont dans $\mathcal{R}$, alors $(x_1, \ldots, x_p) \mapsto f(f_1(x_1, \ldots, x_p), \ldots, f_n(x_1, \ldots, x_p))$ y est aussi;
- pour tout $n \in \mathbb{N}$, $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ et $g : \mathbb{N}^{n+2} \rightharpoonup \mathbb{N}$ dans $\mathcal{R}$, la fonction partielle $rec_{f,g} : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ définie comme suit est aussi dans $\mathcal{R}$:

$$\begin{aligned} rec_{f,g}(x_1, \ldots, x_n, 0) &= f(x_1, \ldots, x_n) \\ rec_{f,g}(x_1, \ldots, x_n, k+1) &= g(x_1, \ldots, x_n, k, rec_{f,g}(x_1, \ldots, x_n, k)). \end{aligned}$$

- pour tout $n \in \mathbb{N}$ et $f : \mathbb{N}^{n+1} \to \mathbb{N}$ (totale) dans $\mathcal{R}$, la fonction μ_f définie comme suit y est aussi:

$$\mu_f(x_1, \ldots, x_n) = \begin{cases} p & \text{si } p = \min\{p \in \mathbb{N} \mid f(x_1, \ldots, x_n, p) > 0\} \\ \bot & \text{si } f(x_1, \ldots, x_n, p) = 0 \text{ pour tout } p, \end{cases}$$

où $\bot$ indique que la fonction n'est pas définie.

Théorème 3. Toute fonction récursive partielle est λ-définissable.

Preuve. On procède par induction sur la démonstration que la fonction donnée est récursive. La $i$ème projection est définissable par $\lambda x_1 . \ldots \lambda x_n . x_i$. La fonction successeur, on l'a vu, est définissable par $\lambda n . \lambda x . \lambda f . f(n\,x\,f)$. La fonction constante égale à 0 est définissable par $\lambda n . \lambda x . \lambda f . x$. Étant donnés $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ et $(f_1, \ldots, f_n) \in (\mathbb{N}^p \rightharpoonup \mathbb{N})^n$ dans $\mathcal{R}$, la fonction $(x_1, \ldots, x_p) \mapsto f(f_1(x_1, \ldots, x_p), \ldots, f_n(x_1, \ldots, x_p))$ est définissable par $\lambda x_1 . \ldots \lambda x_p . (\overline{f}\,(\overline{f_1}\,x_1 \ldots x_p) \ldots (\overline{f_n}\,x_1 \ldots x_p))$.

Si $f : \mathbb{N}^p \rightharpoonup \mathbb{N}$ et $g : \mathbb{N}^{p+2} \rightharpoonup \mathbb{N}$ sont dans $\mathcal{R}$, $\mathrm{rec}_{f,g}$ est définissable par

$$\lambda x_1 . \ldots \lambda x_p . \lambda n . \left(\pi' \left(n \left(\mathrm{Paire}\,\overline{0}\left(\overline{f}\,x_1 \ldots x_p\right)\right)(\lambda x . \mathrm{Paire}\,(S\,(\pi\,x))\,(\overline{g}\,x_1 \ldots x_p\,(\pi\,x)\,(\pi'\,x)))\right)\right),$$

où S est la fonction successeur $\lambda n . \lambda x . \lambda f . (f\,(n\,x\,f))$ et $\mathrm{Paire}\,M\,N$, $\pi\,M$ et $\pi'\,M$ sont comme dans l'exercice 3. L'idée est d'utiliser n comme un itérateur sur des paires, en partant de $(0, f(x_1, \ldots, x_p))$ et en appliquant, à chaque itération, le successeur à gauche et la définition récursive à droite.

Prouvons que ce dernier terme définit bien $\mathrm{rec}_{f,g}$. Fixons $x_1 \ldots x_p$ et posons $x_0 = \mathrm{Paire}\,\overline{0}\,(\overline{f}\,x_1 \ldots x_p)$ et $h = \lambda x . \mathrm{Paire}\,(S\,(\pi\,x))\,(\overline{g}\,x_1 \ldots x_p\,(\pi\,x)\,(\pi'\,x))$. On prouve par induction sur n que $\overline{n}\,x_0\,h$ se réduit à $\mathrm{Paire}\,\overline{n}\,(\overline{\mathrm{rec}_{f,g}(x_1, \ldots, x_p, n)})$. Ça entraîne qu'en appliquant π' au résultat, on obtient bien $\mathrm{rec}_{f,g}(x_1, \ldots, x_p, n)$.

Si $n = 0$, on obtient directement

$$\overline{n}\,x_0\,h \to_\beta x_0 = \mathrm{Paire}\,\overline{0}\left(\overline{f}\,x_1 \ldots x_p\right) = \mathrm{Paire}\,\overline{n}\left(\overline{\mathrm{rec}_{f,g}(x_1, \ldots, x_p, n)}\right).$$

Si $n = n' + 1$, alors $\overline{n}$ est β-équivalent à $\lambda x . \lambda f . (f\,(\overline{n'}\,x\,f))$ et on obtient que $\overline{n}\,x_0\,h$ est β-équivalent à $h\,(\overline{n'}\,x_0\,h)$. Par hypothèse d'induction, ce dernier terme se réduit en $h\,(\mathrm{Paire}\,\overline{n'}\,(\overline{\mathrm{rec}_{f,g}(x_1, \ldots, x_p, n')}))$, puis en

$$\mathrm{Paire}\left(S\,\overline{n'}\right)\left(\overline{g}\,x_1 \ldots x_p\,\overline{n'}\left(\overline{\mathrm{rec}_{f,g}(x_1, \ldots, x_p, n')}\right)\right),$$

i.e., en

$$\mathrm{Paire}\left(S\,\overline{n'}\right)\left(\overline{\mathrm{rec}_{f,g}(x_1, \ldots, x_p, n)}\right),$$

puis en

$$\mathrm{Paire}\,(\overline{n})\left(\overline{\mathrm{rec}_{f,g}(x_1, \ldots, x_p, n)}\right),$$

comme souhaité.

Enfin, si $f : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ est totale et dans $\mathcal{R}$, montrons que μ_f est définissable. Soit Y un combinateur de point fixe, par exemple comme dans l'exercice 4. On pose de plus $\text{true} = \lambda x\,.\,\lambda y\,.\,x$ et $\text{false} = \lambda x\,.\,\lambda y\,.\,y$. Ça donne une représentation du type algébrique des booléens, qu'on peut définir par

```
type bool = True | False
```

en OCaml. Pour faire un branchement conditionnel, if C then P else Q, on peut juste écrire C P Q. Il y a une petite subtilité avec les comportements divergents. Par exemple, si P diverge mais pas Q et $C = \text{false}$, on aimerait que if C then P else Q se réduise en Q, sans possibilité de diverger. Or, $(\lambda x\,.\,\lambda y\,.\,y)\,P\,Q$ diverge, par exemple, en appel par valeur. Comme notre notion de λ-définissabilité utilise la réduction standard, on évite la divergence.

On peut maintenant représenter μ_f en posant $R = \lambda r\,.\,\lambda x\,.\,(f\,x_1 \ldots x_n\,x)\,(r\,(S\,x))\,(\lambda w\,.\,x)$, $M = Y\,R$ et $\overline{\mu_f} = \lambda x_1\,.\,\ldots\lambda x_n\,.\,(M\,\bar{0})$. Montrons que ça se passe bien. On a

$$M\,\bar{0} \to^{\star}_{\text{std}} \left(f\,x_1 \ldots x_n\,\bar{0}\right)\left(M\left(S\,\bar{0}\right)\right)(\lambda w\,.\,x),$$

car $M\,N \to_{\text{std}} R\,M\,N$. Si $f\,x_1 \ldots x_n\,\bar{0} \to^{\star}_{\text{std}} \bar{0}$, alors le tout se réduit en $M\,(S\,\bar{0})$. Si au contraire $f\,x_1 \ldots x_n\,\bar{0}$ se réduit à $\overline{p+1}$ pour un entier p, alors le tout se réduit à l'application $p+1$ fois itérée:

$$(\lambda w\,.\,x)\left(\ldots\left((\lambda w\,.\,x)\left(M\left(S\,\bar{0}\right)\right)\right)\right),$$

qui se réduit (en une étape!) à x. On peut en fait montrer

- $f\,x_1 \ldots x_n(S^p\,\bar{0})$ se réduit à $\bar{n}$ ssi $f\,x_1 \ldots x_n\,\bar{p}$ se réduit à $\bar{n}$;
- pour tout entier p, si $f\,x_1 \ldots x_n(S^p\,\bar{0})$ se réduit à $\bar{0}$, alors $M\,(S^p\,\bar{0})$ se réduit à $M\,(S^{p+1}\,\bar{0})$, alors que si $f\,x_1 \ldots x_n(S^p\,\bar{0})$ se réduit à $\bar{q}$ pour un entier q différent de 0, alors $M\,(S^p\bar{0})$ se réduit à $\bar{p}$.

On a déjà à peu près montré le second point. Le premier est une conséquence de la confluence et du théorème de standardisation 2.

On a donc bien par induction

$$\overline{\mu_f}\,x_1 \ldots x_n = M\,\bar{0} \to^{\star}_{\text{std}} M\left(S^p\,\bar{0}\right)$$

tant que tous les $f\,x_1 \ldots x_n(S^q\,\bar{0})$ se réduisent en $\bar{0}$ pour $q < p$, puis en $S^p\,\bar{0}$ si $f\,x_1 \ldots x_n(S^p\,\bar{0})$ se réduit sur un entier de Church strictement positif. Et donc sur $\bar{p}$. Si pour aucun p on n'a $f\,x_1 \ldots x_n(S^p\,\bar{0})$ différent de 0, alors $\mu_f\,x_1 \ldots x_n$ diverge. Cela coïncide bien avec la définition de μ_f. □

Pour conclure, donnons une idée de la réciproque. En fait, on a mentionné que les fonctions récursives partielles sont aussi celles qui sont *calculables*, i.e., calculables par une machine de Turing. Si on montre que toutes les fonctions λ-définissables sont calculables,

on saura donc qu'elles sont aussi récursives partielles. Or, montrer ça revient à interpréter le λ-calcul dans les machines de Turing, i.e., à écrire un compilateur.

2.6 Un poil de complexité

Dans la partie précédente, on a donné une idée de l'équivalence entre fonctions calculables (Turing), récursives partielles (Herbrand, Gödel, Kleene) et λ-définissables (Church, Kleene). Sans entrer dans le détail, ces résultats parlent de décidabilité, mais pas de complexité. En d'autres termes, la correspondance n'est pas du tout évidente entre

- d'une part, le nombre d'étapes de calcul effectuées par la représentation $\bar{f}\,\overline{n}$ de l'application d'une fonction f à un entier n en λ-calcul,
- d'autre part, le nombre d'étapes de calcul effectuées par la machine de Turing correspondante.

En fait, ça dépend de la compilation qu'on a choisie du λ-calcul vers les machines de Turing. Contentons nous de mentionner qu'on considère en général qu'on perd un facteur log en temps à la compilation.

3 Démonstrations

3.1 Rappels: formules du calcul des prédicats

En maths, on démontre des formules. La logique (mathématique) s'occupe de définir ce que ça signifie. La première étape de ce processus consiste à définir ce qu'est une formule.

Ça se fait généralement en suivant le «patron» suivant à trois étages:

- d'abord on construit les *termes* du langage; par exemple: les nombres, les variables, les opérations; un exemple de terme (en arithmétique) est $x+1$;
- ensuite, on décrit les *relations atomiques*; par exemple: l'égalité, l'appartenance;
- enfin, on donne des *connecteurs logiques*; par exemple, la disjonction $\vee$, la conjonction $\wedge$, ou l'implication $\Rightarrow$.

Une fois qu'on a défini les formules, on s'occupe de décider quand elles sont vraies. On verra ça un peu plus loin; commençons par décrire plus précisément les trois étages de construction des formules.

Le «patron» esquissé plus haut s'appelle le *calcul des prédicats* et se formalise comme suit (dans sa version monosortée).

On prend d'abord en paramètre une *signature*, i.e., une famille d'ensembles finis indexée par les entiers, $\Sigma = (\Sigma_n)_{n\in\mathbb{N}}$.

Exemple 3. L'*arithmétique de Peano* a la signature définie par:

- $\Sigma_0^{\text{Peano}} = \{0\}$,
- $\Sigma_1^{\text{Peano}} = \{S\}$,

- $\Sigma_2^{\text{Peano}} = \{+, \times\}$,
- $\Sigma_n^{\text{Peano}} = \emptyset$ pour $n > 2$.

Soyons précis: il faut lire, par exemple, que Σ_0^{Peano} est un ensemble à un élément, qu'on appelle 0.

On fixe un ensemble infini $\mathcal{V}$ de *variables*.

Définition 10. L'ensemble $\mathcal{T}(\Sigma)$ des *termes* sur une signature Σ est défini inductivement par:
- les éléments de Σ_0 sont des termes;
- les *variables* sont des termes;
- pour tout $n \in \mathbb{N}$ et $o \in \Sigma_n$, si $t_1, \ldots, t_n$ sont des termes, alors $o(t_1, \ldots, t_n)$ est un terme.

Exemple 4. Des exemples d'éléments de $\mathcal{T}(\Sigma^{\text{Peano}})$: 0, $S(0)$, $S(S(0))$, $S(S(S(0)))$ (qui représentent respectivement les nombres $0, 1, 2, 3$); $S(x)$, le successeur d'une variable x, $x + S(0)$, $x \times x$. Pour l'instant, rien n'exige, par exemple, que $S(x)$ égale $x + S(0)$. Ça va être imposé par les axiomes, un peu plus loin.

Remarque 1. Ici, on fait ça sans trop se poser la question de ce que signifie mathématiquement $o(t_1, \ldots, t_n)$. Pour certains auteurs, les termes sont certaines listes d'éléments de

$$\{\text{lp}, \text{rp}\} \cup \left(\sum_{i \in \mathbb{N}} \Sigma_i \right) \cup \mathcal{V}$$

où on utilise lp et rp pour dire «parenthèse à gauche» et «parenthèse à droite», respectivement.

Une autre manière de faire est de définir les termes comme l'union (dirigée) d'ensembles:
- $T_0 = \Sigma_0 \cup \mathcal{V}$,
- $T_{n+1} = T_n + \{(o, (t_1, \ldots, t_m)) \mid o \in \Sigma_m \wedge t_1, \ldots, t_m \in T_n\}$,

où la somme $X + Y$ de deux ensembles est définie par $(\{0\} \times X) \cup (\{1\} \times Y)$.

Il y a encore d'autres possibilités, l'essentiel étant d'avoir le principe d'induction ci-dessus.

Avant de parler d'axiomes, on mentionne le premier résultat important sur les termes, appelé *principe d'induction*:

Proposition 4. Soit P un ensemble de termes. Supposons
- $\Sigma_0 \subseteq P$,
- $\mathcal{V} \subseteq P$,
- pour tous $t_1, \ldots, t_n \in P$ et $o \in \Sigma_n$, $o(t_1, \ldots, t_n) \in P$.

Alors $P = \mathcal{T}(\Sigma)$.

Ça, c'était pour les termes. Les relations atomiques sont elles aussi spécifiées par une signature, disons R, donnant pour chaque entier n l'ensemble des relations d'arité n.

Exemple 5. Pour l'arithmétique de Peano, on prend $R_2 = \{=\}$ et tous les autres R_n vides: on n'a que l'égalité comme relation atomique. D'autres relations sont définissables à partir d'elle; par exemple $x \leqslant y$ est défini comme $\exists z\,.\,(x + z) = y$.

Enfin, on spécifie les connecteurs logiques par une autre forme de signature. Comme il n'y en a que quelques-uns on les définit tous pour commencer, puis, selon les cas, on fait semblant d'en oublier certains.

L'ensemble $\mathcal{F}(\Sigma, R)$ des formules est défini en pseudo-BNF par

$$F, G ::= r(t_1, \ldots, t_n) \mid \bot \mid \top \mid \neg F \mid \exists x\,.\,F \mid \forall x\,.\,F \mid F \vee G \mid F \wedge G \mid F \Rightarrow G$$

et, plus formellement, inductivement par

- pour tout $r \in R_n$, et $t_1, \ldots, t_n \in \mathcal{T}(\Sigma)$, $r(t_1, \ldots, t_n) \in \mathcal{F}(\Sigma, R)$;
- $\bot$ et $\top$ sont dans $\mathcal{F}(\Sigma, R)$;
- pour toute formule $F \in \mathcal{F}(\Sigma, R)$, $\neg F \in \mathcal{F}(\Sigma, R)$;
- pour toute formule $F \in \mathcal{F}(\Sigma, R)$ et toute variable $x \in \mathcal{V}$, $\exists x\,.\,F$ et $\forall x\,.\,F$ sont dans $\mathcal{F}(\Sigma, R)$;
- pour toutes formules $F, G \in \mathcal{F}(\Sigma, R)$, $F \vee G$, $F \wedge G$, $F \Rightarrow G$ sont dans $\mathcal{F}(\Sigma, R)$.

(Ça c'est pour la logique dite du premier ordre. Il y en a d'autres, qu'on verra peut-être plus tard.)

Exemple 6. La formule $\forall x\,.\,\exists y\,.\,(y = x \times x)$ postule l'existence du carré de chaque entier, sans pour l'instant en postuler l'unicité.

Définition 11. Une *théorie* sur une signature (Σ, R) est un ensemble de formules dans $\mathcal{F}(\Sigma, R)$.

On a défini la notion de formule, sans chercher à préciser quand les formules sont vraies ou fausses.

Il existe une multitude de façons de définir ça, qui ne sont pas toutes équivalentes (c'est-à-dire qu'elles ne définissent pas les mêmes formules vraies).

La manière la plus intuitive (pour la plupart des gens) de définir les formules vraies est de définir la notion de *modèle* et de dire qu'une formule est vraie quand elle l'est dans tous les modèles.

Un inconvénient qu'on peut trouver à ce genre de définition est qu'elles exigent la maîtrise de théories logiques subtiles, telles que la théorie des ensembles de Zermelo et Frankel, même pour des démonstrations simples d'arithmétique.

Un autre inconvénient est qu'elles dépendent un peu trop, et de manière mal contrôlée, des principes mathématiques qu'on s'autorise à utiliser — qu'on appelle en général la logique *ambiante*, ou *méta-logique*.

Exemple 7. On connait certaines formules qui deviennent fausses si on s'interdit l'*axiome du choix*, selon lequel toute fonction $f : X \to Y$ admet une *section*, c'est-à-dire une fonction $s : Y \to X$ telle que $f \circ s = \mathrm{id}$ (l'identité).

D'autres inconvénients?

On va présenter plusieurs notions de vérité, dont l'avantage théorique principal est qu'elles sont définissables en arithmétique, c'est-à-dire à peu près quelle que soit la logique ambiante, tant qu'elle n'est pas trop faible. Un avantage d'ordre plus pratique est que démontrer une formule ressemble à un jeu, où on a à chaque étape le choix entre quelques coups possibles. Ce genre de trucs s'enseigne mieux.

L'idée clef pour ça est de définir non seulement la notion de vérité mais la notion de *démonstration* (ou de *preuve*). C'est en examinant les propriétés de cette notion qu'on va (re)découvrir le lien avec le calcul.

3.2 Déduction naturelle

Répétons-nous, ou presque: la manière la plus intuitive (pour la plupart des gens) de définir la notion de preuve s'appelle la *déduction naturelle*.

On peut définir les preuves de déduction naturelle en arithmétique, mais c'est un peu se lier les mains derrière le dos, donc on va s'autoriser des constructions plus ensemblistes.

Les preuves vont être des sortes d'arbres, dont la forme est localement très contrainte. Chaque nœud représente l'application d'une règle logique. Naïvement, on aimerait étiqueter les nœuds par le «nom» de la règle correspondante et les arêtes par des formules. Par exemple, on aimerait représenter la conjonction par le nœud

$$\frac{A \quad B}{A \wedge B},$$

interprété comme suit: si on a déjà démontré A et B, on peut considérer qu'on a démontré $A \wedge B$; ou bien: une démonstration de $A \wedge B$ consiste en une paire d'une démonstration de A et d'une démonstration de B. Cette méthode marche pour beaucoup de connecteurs logiques, mais bute plusieurs fois, ce qui nous force à la généraliser.

Le premier point de blocage est le connecteur $\Rightarrow$. On aimerait dire qu'une démonstration de $A \Rightarrow B$ consiste en une démonstration de B obtenue en supposant A. D'ailleurs, comment faire des preuves en supposant vraie une théorie donnée?

La manière standard de corriger le problème conististe à étiqueter les arêtes de l'arbre avec non seulement la formule à démontrer, mais aussi avec une liste d'hypothèses

supposées vraies. On note les étiquettes $\Gamma \vdash A$, où Γ est une liste de formules et on les appelle des *séquents*.

On obtient une règle limpide (hopefully) pour l'implication

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

où Γ est une liste de formules. La règle pour la conjonction devient

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

Bien sûr, il faut pouvoir utiliser ses hypothèses, donc on ajoute la règle dite d'*axiome* $\frac{}{\Gamma \vdash A}$ si $A \in \Gamma$.
On note que si Γ contient plusieurs occurrences de A, on pourrait considérer que chacune donne une démonstration (un peu) différente.

Remarque 2. En passant des simples formules aux séquents, on répond du même coup à la question de ce qu'est une preuve dans une théorie donnée, en tout cas tant que la théorie en question est finie. (Rappelons qu'une *liste* est une séquence finie.) En effet, si T est une théorie finie, on choisit un ordre pour obtenir une liste Γ_T et on fait nos preuves en mettant Γ_T à gauche du symbole $\vdash$.

Jusqu'ici, on a vu comment construire des preuves de $A \wedge B$ ou $A \Rightarrow B$. Mais comment en utiliser comme hypothèse? Par exemple, on aimerait bien, sachant $A \wedge B$, en déduire A (ou B). On ajoute des règles exprès:

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad\qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}.$$

On appelle ça des règles d'élimination. La règle pour l'implication est plus bizarre.

Exercice 9. Essayer de deviner quand et comment on a envie d'éliminer $\Rightarrow$, c'est-à-dire comment on utilise une hypothèse de la forme $A \Rightarrow B$.

Réponse: quand on sait déjà A. Ça donne la règle

$$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B},$$

qui s'appelle le *modus ponens*.

Des connecteurs un peu bizarres sont $\top$ et $\bot$, respectivement le vrai et le faux. Le vrai est vrai quelles que soient les hypothèses, donc on pose la règle

$$\frac{}{\Gamma \vdash \top}.$$

C'est une règle d'introduction, et $\top$ n'a pas de règle d'élimination — il ne sert à rien comme hypothèse.

Le faux, a.k.a. la contradiction, entraîne n'importe quoi, on a donc la règle

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

pour n'importe quel A. Il n'y a pas de règle d'introduction: ça reviendrait à permettre directement la contradiction. On va passer pas mal de temps, plus loin, à se demander quand une théorie est *cohérente*, c'est-à-dire qu'elle ne permet pas de démontrer $\bot$.

Pour la disjonction, on peut tranquillement poser

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad\qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}.$$

Ce sont des règles d'introduction. La règle d'élimination est plus bizarre:

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}.$$

Elle dit que si on a démontré $A \vee B$ et qu'en sachant l'un ou l'autre on sait démontrer C, alors on a démontré C.

Enfin, on peut définir $\neg A$ comme $A \Rightarrow \bot$. Ça nous empêche de considérer des fragments sans $\bot$ ou $\Rightarrow$, mais bon, tant pis.

Les règles définies jusqu'ici forment ce qu'on appelle la *déduction naturelle intuitionniste*

Définition 12. Une *démonstration* est un arbre dont
- les arêtes et la racine sont étiquetées par des séquents,
- chaque nœud est étiqueté par une des règles ci-dessus.

Un séquent $\Gamma \vdash A$ est *dérivable* ssi c'est l'étiquette de la racine d'une démonstration.

Là encore, on est informels: qu'est-ce exactement qu'un arbre étiqueté? Qu'est-ce qu'on entend par «chaque nœud est étiqueté par une des règles ci-dessus»?

Exemple 8. Que signifie $\frac{\Gamma \vdash A \wedge A}{\Gamma \vdash A}$? En vrai, il faudrait nommer toutes les règles ci-dessus et étiqueter toutes nos applications de règles par ces noms. L'arbre précédent est ambigu: il peut soit désigner une règle de projection gauche, soit une règle de projection droite. En pratique, ça n'arrivera pas souvent, donc on omet les étiquettes pour plus de lisibilité. Quand on en a besoin pour éviter une ambiguïté, on écrit par exemple $\frac{\Gamma \vdash A \wedge A}{\Gamma \vdash A}\wedge$–gauche.

Un autre exemple d'ambiguïté est l'axiome. Par exemple, on écrira les deux sens possibles de $\frac{}{A, A \vdash A}$ comme

$$\frac{}{A, A \vdash A}\text{ax}_1 \quad \text{et} \quad \frac{}{A, A \vdash A}\text{ax}_2,$$

c'est-à-dire en donnant en étiquette l'indice de l'hypothèse utilisée.

Trois tonnes d'autres exemples.

En déduction naturelle intuitionniste, on ne sait pas forcément dériver $A \vee \neg A$ pour tout A. Si on ajoute de quoi le faire, on passe en déduction naturelle *classique*. Il y a plusieurs manières de le faire, qu'on verra en partie 3.9.

3.3 Elimination des coupures

Dans cette partie, on propose une manière de mettre les démonstrations en forme normale. Attention, c'est très fastidieux. C'est comme ça que les gens faisaient avant de comprendre la correspondance de Curry-Howard. On verra plus tard à quel point ça simplifie toute l'affaire.

Avec les règles proposées ci-dessus, on peut faire des démonstrations un peu débiles. Par exemple:

$$\dfrac{\dfrac{\dfrac{\dots}{\Gamma \vdash A} \quad \dfrac{\dots}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}}{\Gamma \vdash A}$$

où on fait une démonstration de B, pour la jeter ensuite.

Un autre exemple:

$$\dfrac{\dfrac{\dfrac{\dots}{\Gamma \vdash A}}{\Gamma \vdash A \vee B} \quad \dfrac{\dots}{\Gamma, A \vdash C} \quad \dfrac{\dots}{\Gamma, B \vdash C}}{\Gamma \vdash C}$$

Ici, on fait semblant de ne pas savoir si c'est A ou B qui est vrai et on se force à démontrer C dans les deux cas, alors qu'on sait démontrer A.

On peut aussi faire des démonstrations qui ont l'air débiles mais qui ne le sont pas. Par exemple:

$$\dfrac{\dfrac{\dfrac{\dots}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{\dots}{\Gamma \vdash A}}{\Gamma \vdash B}$$

Ici, on a l'air de supposer A alors qu'on sait le démontrer. En fait, si on utilise beaucoup A dans la démonstration de B, on préfère ne le démontrer qu'une fois (ça s'appelle un «lemme» ou un résultat intermédiaire en langage courant).

Dans ces trois exemples, on utilise d'abord (en haut) une règle d'introduction et ensuite une règle d'élimination.

Définition 13. On appelle ça des *coupures*.

Le processus de normalisation s'appelle donc l'*élimination des coupures*.

Il devrait être très naturel de définir une relation binaire entre démonstrations, consistant à *réduire* les coupures. On note cette relation par $\rightsquigarrow$. Par exemple, on a

$$\dfrac{\dfrac{\overset{\cdots}{\Gamma \vdash A} \quad \dfrac{\overset{\cdots}{\Gamma \vdash B}}{}}{\Gamma \vdash A \wedge B}}{\Gamma \vdash A} \rightsquigarrow \dfrac{\overset{\cdots}{\Gamma \vdash A}}{} \text{ et } \dfrac{\dfrac{\overset{\cdots}{\Gamma \vdash A} \quad \dfrac{\overset{\cdots}{\Gamma \vdash B}}{}}{\Gamma \vdash A \wedge B}}{\Gamma \vdash B} \rightsquigarrow \dfrac{\overset{\cdots}{\Gamma \vdash B}}{}$$

Les cas de $\vee$ et $\Rightarrow$ sont un peu plus compliqués. Par exemple, pour

$$\dfrac{\dfrac{\overset{\cdots}{\Gamma \vdash A}}{\Gamma \vdash A \vee B} \quad \dfrac{\overset{\cdots}{\Gamma, A \vdash C}}{} \quad \dfrac{\overset{\cdots}{\Gamma, B \vdash C}}{}}{\Gamma \vdash C},$$

on aimerait dire que ça se réduit à la preuve $\dfrac{\overset{\cdots}{\Gamma, A \vdash C}}{}$, dans laquelle on aurait remplacé chaque utilisation de l'hypothèse A par la preuve $\dfrac{\overset{\cdots}{\Gamma \vdash A}}{}$.

Pour faire ça proprement, on va définir l'opération de substitution sur les arbres de démonstration, avant de s'apercevoir que ça fait à peu près la même chose que la substitution du λ-calcul. On définit donc par induction une fonction

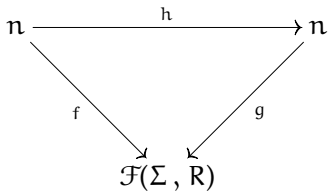
$$\text{subst} : \prod_{\Gamma, \Delta \in \mathcal{F}(\Sigma, R)^*} \prod_{A, B \in \mathcal{F}(\Sigma, R)} \langle \Gamma, A, \Delta \vdash B \rangle \to \langle \Gamma \vdash A \rangle \to \langle \Gamma, \Delta \vdash B \rangle,$$

où $\langle \Gamma \vdash A \rangle$ désigne l'ensemble des arbres de démonstration de $\Gamma \vdash A$. Pour $t \in \langle \Gamma, A, \Delta \vdash B \rangle$ et $u \in \langle \Gamma \vdash A \rangle$, on aura donc $(\text{subst}\, t\, u) \in \langle \Gamma, \Delta \vdash B \rangle$.

On commence par définir le renommage d'une démonstration. Voyons les listes de formules comme des paires (n, f) où n est un entier et $f : n \to \mathcal{F}(\Sigma, R)$, où n est vu comme l'ensemble $\{1, \ldots, n\}$.

Définition 14. Un *renommage* de (n, f) vers (m, g) est une fonction $h : n \to m$ telle que

commute.



On définit une fonction de *renommage*

$$\text{ren} : \prod_{\Gamma, \Delta \in \mathcal{F}(\Sigma, R)^*} \prod_{A \in \mathcal{F}(\Sigma, R)} \prod_{h : \Gamma \to \Delta} \langle \Gamma \vdash A \rangle \to \langle \Delta \vdash A \rangle$$

par induction sur l'argument t de type $\langle \Gamma \vdash A \rangle$:

- si t est un axiome, avec $\Gamma(i) = A$, alors par définition des renommages, $\Delta(h(i)) = A$ et on définit $\mathrm{ren}_{\Gamma,\Delta,A,h}t$ comme l'axiome $\Delta \vdash A$ correspondant;
- si t est de la forme $\dfrac{\dfrac{t_1}{\Gamma \vdash B \Rightarrow A} \quad \dfrac{t_2}{\Gamma \vdash B}}{\Gamma \vdash A}$, on définit récursivement $\mathrm{ren}_{\Gamma,\Delta,A,h}t$ par
$$\frac{\dfrac{\mathrm{ren}_{\Gamma,\Delta,B\Rightarrow A,h}t_1}{\Delta \vdash B \Rightarrow A} \quad \dfrac{\mathrm{ren}_{\Gamma,\Delta,B,h}t_2}{\Delta \vdash B}}{\Delta \vdash A};$$
- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma, B \vdash C}}{\Gamma \vdash B \Rightarrow C}$, avec $A = (B \Rightarrow C)$, on définit $\mathrm{ren}_{\Gamma,\Delta,A,h}t$ par $\dfrac{\dfrac{\mathrm{ren}_{(\Gamma,B),(\Delta,B),C,h'}t'}{\Delta, B \vdash C}}{\Delta \vdash B \Rightarrow C}$, où h' envoie les éléments de Γ sur Δ comme prescrit par h et B sur B;
- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma \vdash A \wedge B}}{\Gamma \vdash A}$, on pose $\mathrm{ren}_{\Gamma,\Delta,A,h}t$ égal à $\dfrac{\dfrac{\mathrm{ren}_{\Gamma,\Delta,A,h}t'}{\Delta \vdash A \wedge B}}{\Delta \vdash A}$;
- on procède similairement pour le cas où t est de la forme $\dfrac{\dfrac{t'}{\Gamma \vdash B \wedge A}}{\Gamma \vdash A}$;
- si t est de la forme $\dfrac{\dfrac{t_1}{\Gamma \vdash B} \quad \dfrac{t_2}{\Gamma \vdash C}}{\Gamma \vdash B \wedge C}$, avec $A = (B \wedge C)$, alors on pose $\mathrm{ren}_{\Gamma,\Delta,A,h}t$ égal à $\dfrac{\dfrac{\mathrm{ren}_{\Gamma,\Delta,A,h}t_1}{\Delta \vdash B} \quad \dfrac{\mathrm{ren}_{\Gamma,\Delta,A,h}t_2}{\Delta \vdash C}}{\Delta \vdash B \wedge C}$;
- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma \vdash B \vee C} \quad \dfrac{t_B}{\Gamma, B \vdash A} \quad \dfrac{t_C}{\Gamma, C \vdash A}}{\Gamma \vdash A}$, on pose $\mathrm{ren}_{\Gamma,\Delta,A,h}t$ égal à $\dfrac{\dfrac{\mathrm{ren}_{\Gamma,\Delta,B\vee C,h}t'}{\Delta \vdash B \vee C} \quad \dfrac{\mathrm{ren}_{(\Gamma,B),(\Delta,B),A,h_B}t_B}{\Delta, B \vdash A} \quad \dfrac{\mathrm{ren}_{(\Gamma,C),(\Delta,C),A,h_C}t_C}{\Delta, C \vdash A}}{\Delta \vdash A}$, où h_B envoie Γ sur Δ comme prescrit par h et B sur B, et similairement pour h_C;
- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma \vdash B}}{\Gamma \vdash B \vee C}$, avec $A = B \vee C$, on pose $\mathrm{ren}_{\Gamma,\Delta,A,h}t$ égal à $\dfrac{\dfrac{\mathrm{ren}_{\Gamma,\Delta,A,h}t'}{\Delta \vdash B}}{\Delta \vdash B \vee C}$;
- on procède similairement si t est de la forme $\dfrac{\dfrac{t'}{\Gamma \vdash C}}{\Gamma \vdash B \vee C}$;
- si t est de la forme $\dfrac{}{\Gamma \vdash \top}$, alors $\mathrm{ren}_{\Gamma,\Delta,A,h}t$ est $\dfrac{}{\Delta \vdash \top}$;

- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma \vdash \bot}}{\Gamma \vdash A}$, on pose $\mathrm{ren}_{\Gamma,\Delta,A,h} t$ égal à $\dfrac{\dfrac{\mathrm{ren}_{\Gamma,\Delta,A,h} t'}{\Delta \vdash \bot}}{\Delta \vdash A}$.

Cette définition a l'air de ne faire que des choses triviales, mais n'est en fait pas si innocente que ça.

Exercice 10. Soit t la démonstration $\dfrac{\dfrac{}{A, B \vdash A}}{A \vdash B \Rightarrow A}$

et soit h l'injection de A dans A, C. Calculer $\mathrm{ren}_{(A),(A,C),(B \Rightarrow A),h} t$.

On peut maintenant définir la substitution. Rappelons que pour $t \in \langle \Gamma, A, \Delta \vdash B \rangle$ et $u \in \langle \Gamma \vdash A \rangle$, on veut $(\mathrm{subst}\, t\, u) \in \langle \Gamma, \Delta \vdash B \rangle$. On procède par induction sur t:

- si t est la preuve axiome $\Gamma, A, \Delta \vdash A$ étiquetée par l'indice de A dans Γ, A, Δ, alors on définit $\mathrm{subst}\, t\, u = \mathrm{ren}_{\Gamma,(\Gamma,\Delta),A,h} u$, où h est l'injection $\Gamma \hookrightarrow (\Gamma, \Delta)$;
- si t est de la forme
$$\dfrac{\dfrac{t_1}{\Gamma, A, \Delta \vdash C \Rightarrow B} \quad \dfrac{t_2}{\Gamma, A, \Delta \vdash C}}{\Gamma, A, \Delta \vdash B},$$
alors $\mathrm{subst}_{\Gamma,\Delta,A,B} t\, u$ est la démonstration
$$\dfrac{\dfrac{\mathrm{subst}_{\Gamma,\Delta,A,B} t_1\, u}{\Gamma, \Delta \vdash C \Rightarrow B} \quad \dfrac{\mathrm{subst}_{\Gamma,\Delta,A,B} t_2\, u}{\Gamma, \Delta \vdash C}}{\Gamma, \Delta \vdash B};$$
- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma, A, \Delta, C \vdash D}}{\Gamma, A, \Delta \vdash C \Rightarrow D}$, avec $B = (C \Rightarrow D)$, on définit $\mathrm{subst}_{\Gamma,\Delta,A,B} t\, u$ comme étant la démonstration $\dfrac{\dfrac{\mathrm{subst}_{\Gamma,(\Delta,C),A,D} t'\, u}{\Gamma, \Delta, C \vdash D}}{\Gamma, \Delta \vdash C \Rightarrow D}$;
- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma, A, \Delta \vdash B \wedge C}}{\Gamma, A, \Delta \vdash B}$, on pose $\mathrm{subst}_{\Gamma,\Delta,A,B} t\, u$ égal à $\dfrac{\dfrac{\mathrm{subst}_{\Gamma,\Delta,A,(B \wedge C)} t'\, u}{\Gamma, \Delta \vdash B \wedge C}}{\Gamma, \Delta \vdash B}$;
- si t est de la forme $\dfrac{\dfrac{t_1}{\Gamma, A, \Delta \vdash C} \quad \dfrac{t_2}{\Gamma, A, \Delta \vdash D}}{\Gamma, A, \Delta \vdash C \wedge D}$, avec $B = (C \wedge D)$, on pose $\mathrm{subst}_{\Gamma,\Delta,A,B} t\, u$ égal à $\dfrac{\dfrac{\mathrm{subst}_{\Gamma,\Delta,A,C} t_1\, u}{\Gamma, \Delta \vdash C} \quad \dfrac{\mathrm{subst}_{\Gamma,\Delta,A,D} t_2\, u}{\Gamma, \Delta \vdash D}}{\Gamma, \Delta \vdash C \wedge D}$,

- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma, A, \Delta \vdash C \vee D} \quad \dfrac{t_C}{\Gamma, A, \Delta, C \vdash B} \quad \dfrac{t_D}{\Gamma, A, \Delta, D \vdash B}}{\Gamma, A, \Delta \vdash B}$, on pose $\mathrm{subst}_{\Gamma,\Delta,A,B}\, t\, u$ égal à $\dfrac{\dfrac{t'}{\Gamma, \Delta \vdash C \vee D} \quad \dfrac{\mathrm{subst}_{\Gamma,(\Delta,C),A,B}\, t_C\, u}{\Gamma, \Delta, C \vdash B} \quad \dfrac{\mathrm{subst}_{\Gamma,(\Delta,D),A,B}\, t_D\, u}{\Gamma, \Delta, D \vdash B}}{\Gamma, \Delta \vdash B}$;
- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma, A, \Delta \vdash C}}{\Gamma, A, \Delta \vdash C \vee D}$, avec $B = C \vee D$, on pose $\mathrm{subst}_{\Gamma,\Delta,A,B}\, t\, u$ égal à $\dfrac{\dfrac{\mathrm{subst}_{\Gamma,\Delta,A,C}\, t'\, u}{\Gamma, \Delta \vdash C}}{\Gamma, \Delta \vdash C \vee D}$;
- on procède similairement si t est de la forme $\dfrac{\dfrac{t'}{\Gamma, A, \Delta \vdash D}}{\Gamma, A, \Delta \vdash C \vee D}$, avec $B = C \vee D$;
- si t est de la forme $\dfrac{}{\Gamma, A, \Delta \vdash \top}$, avec $B = \top$, alors $\mathrm{subst}_{\Gamma,\Delta,A,B}\, t\, u$ est $\dfrac{}{\Gamma, \Delta \vdash \top}$;
- si t est de la forme $\dfrac{\dfrac{t'}{\Gamma, A, \Delta \vdash \bot}}{\Gamma, A, \Delta \vdash B}$, on pose $\mathrm{subst}_{\Gamma,\Delta,A,B}\, t\, u$ égal à $\dfrac{\dfrac{\mathrm{subst}_{\Gamma,\Delta,A,\bot}\, t'\, u}{\Gamma, \Delta \vdash \bot}}{\Gamma, \Delta \vdash B}$.

Grâce à cette définition de la substitution, on peut définir la relation $\rightsquigarrow$ comme la plus petite relation satisfaisant les règles suivantes. C'est une relation ne reliant que des démonstration du même séquent, ce qui justifie les dernières règles (sous le trait horizontal).

$$\frac{\dfrac{t_1}{\Gamma \vdash A} \quad \dfrac{t_2}{\Gamma \vdash B}}{\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}} \rightsquigarrow \frac{t_1}{\Gamma \vdash A}$$

$$\frac{\dfrac{t_1}{\Gamma \vdash A} \quad \dfrac{t_2}{\Gamma \vdash B}}{\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}} \rightsquigarrow \frac{t_2}{\Gamma \vdash B}$$

$$\frac{\dfrac{\dfrac{t}{\Gamma \vdash A}}{\Gamma \vdash A \vee B} \quad \dfrac{t_A}{\Gamma, A \vdash C} \quad \dfrac{t_B}{\Gamma, B \vdash C}}{\Gamma \vdash C} \rightsquigarrow \frac{\mathrm{subst}_{\Gamma,\varepsilon,A,C}\, t_A\, t}{\Gamma \vdash C}$$

$$\frac{\dfrac{\dfrac{t}{\Gamma \vdash B}}{\Gamma \vdash A \vee B} \quad \dfrac{t_A}{\Gamma, A \vdash C} \quad \dfrac{t_B}{\Gamma, B \vdash C}}{\Gamma \vdash C} \rightsquigarrow \frac{\mathrm{subst}_{\Gamma,\varepsilon,B,C}\, t_B\, t}{\Gamma \vdash C}$$

$$\frac{\dfrac{\dfrac{t_1}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{t_2}{\Gamma \vdash A}}{\Gamma \vdash B} \rightsquigarrow \frac{\mathrm{subst}_{\Gamma,\varepsilon,A,B}\, t_1\, t_2}{\Gamma \vdash B}$$

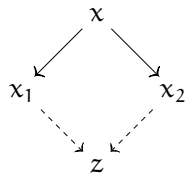
Note: on est un peu embêté avec la règle

$$\frac{\dfrac{t}{\Gamma \vdash \bot}}{\Gamma \vdash A}.$$

En effet, on aimerait que la relation $\rightsquigarrow$ soit *confluente*.

Définition 15. Une relation binaire R sur un ensemble X est confluente ssi $(R^{op};R) \subseteq (R;R^{op})$.

Concrètement, R est confluente ssi pour tout diagramme comme la partie solide du diagramme ci-dessous, il existe z tel qu'on ait la partie en pointillés:



Ce dont on a envie pour la règle $\bot$ est qu'une fois qu'on a une preuve t de $\bot$, toutes les preuves dans le même contexte Γ se réduisent à t. Or, si on considère une démonstration de la forme

$$\frac{t}{\Gamma \vdash A} \rightsquigarrow \frac{t'}{\Gamma \vdash A} \quad \text{si} \quad t \rightsquigarrow t'$$

$$\frac{t}{\Gamma \vdash A \vee B} \rightsquigarrow \frac{t'}{\Gamma \vdash A \vee B} \quad \text{si} \quad t \rightsquigarrow t'$$

$$\frac{t}{\Gamma \vdash B} \rightsquigarrow \frac{t'}{\Gamma \vdash B} \quad \text{si} \quad t \rightsquigarrow t'$$

$$\frac{\dfrac{t}{\Gamma \vdash A \vee B} \quad \dfrac{t_A}{\Gamma, A \vdash C} \quad \dfrac{t_B}{\Gamma, B \vdash C}}{\Gamma \vdash C} \rightsquigarrow \frac{\dfrac{t'}{\Gamma \vdash A \vee B} \quad \dfrac{t_A}{\Gamma, A \vdash C} \quad \dfrac{t_B}{\Gamma, B \vdash C}}{\Gamma \vdash C} \quad \text{si} \quad t \rightsquigarrow t'$$

$$\frac{\dfrac{t}{\Gamma \vdash A \vee B} \quad \dfrac{t_A}{\Gamma, A \vdash C} \quad \dfrac{t_B}{\Gamma, B \vdash C}}{\Gamma \vdash C} \rightsquigarrow \frac{\dfrac{t}{\Gamma \vdash A \vee B} \quad \dfrac{t'_A}{\Gamma, A \vdash C} \quad \dfrac{t_B}{\Gamma, B \vdash C}}{\Gamma \vdash C} \quad \text{si} \quad t_A \rightsquigarrow t'_A$$

$$\frac{\dfrac{t}{\Gamma \vdash A \vee B} \quad \dfrac{t_A}{\Gamma, A \vdash C} \quad \dfrac{t_B}{\Gamma, B \vdash C}}{\Gamma \vdash C} \rightsquigarrow \frac{\dfrac{t}{\Gamma \vdash A \vee B} \quad \dfrac{t_A}{\Gamma, A \vdash C} \quad \dfrac{t'_B}{\Gamma, B \vdash C}}{\Gamma \vdash C} \quad \text{si} \quad t_B \rightsquigarrow t'_B$$

$$\frac{\dfrac{t_1}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{t_2}{\Gamma \vdash A}}{\Gamma \vdash B} \rightsquigarrow \frac{\dfrac{t'_1}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{t_2}{\Gamma \vdash A}}{\Gamma \vdash B} \quad \text{si} \quad t_1 \rightsquigarrow t'_1$$

$$\frac{\dfrac{t_1}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{t_2}{\Gamma \vdash A}}{\Gamma \vdash B} \rightsquigarrow \frac{\dfrac{t_1}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{t'_2}{\Gamma \vdash A}}{\Gamma \vdash B} \quad \text{si} \quad t_2 \rightsquigarrow t'_2$$

$$\frac{\dfrac{t}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \rightsquigarrow \frac{\dfrac{t'}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \quad \text{si} \quad t \rightsquigarrow t'$$

Figure 1 – Règles de contexte

3.4 Représentation syntaxique des démonstrations

Restreignons nous pour l'instant au connecteur $\Rightarrow$. Au lieu de considérer les démonstrations comme des arbres, on peut les voir comme des «termes typés», inductivement engendrés par les conditions suivantes:

- si Γ comprend n formules, alors, pour tout $i \in \{1, \dots, n\}$, i est de type $\Gamma(i)$ dans le contexte Γ, ce qu'on note $\Gamma \vdash i : \Gamma(i)$;
- si $\Gamma \vdash t : A \Rightarrow B$ et $\Gamma \vdash u : A$, alors $(t\,u)$ est un terme de type B dans le contexte Γ,
- si $\Gamma, A \vdash t : B$, alors $\lambda . t$ est un terme de type $A \Rightarrow B$ dans le contexte Γ.

On présente généralement ces règles de construction comme des règles d'inférence logique:

$$\frac{}{\Gamma \vdash \Gamma(i)} i \in \{1, \dots, n\} \quad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B} \quad \frac{\Gamma, A \vdash t : B}{\Gamma \vdash \lambda . t : A \Rightarrow B}.$$

C'est presque des termes du λ-calcul. Il y a néanmoins deux problèmes. D'abord, il n'y a pas de variables mais des numéros de formules, qui doivent rester plus petits que la taille du contexte Γ. Ensuite, il peut y avoir plusieurs démonstrations s'envoyant sur le même terme. Par exemple, soient $\Gamma = (A \Rightarrow B, A)$ et $\Delta = (A' \Rightarrow B, A')$, pour des formules $A \neq A'$. Les deux démonstrations

$$\frac{\Gamma \vdash 1 : A \Rightarrow B \quad \Gamma \vdash 2 : A}{\Gamma \vdash 12 : B} \quad \text{et} \quad \frac{\Delta \vdash 1 : A' \Rightarrow B \quad \Delta \vdash 2 : A'}{\Delta \vdash 12 : B}$$

donnent le terme 12. Ça fait même un peu peur, parce que ces deux démonstrations démontrent la même formule B. Heureusement, elles n'utilisent pas le même contexte. Pour formaliser en quel sens ça nous rassure, on procède comme suit. Soit LC la famille des termes indexée par les séquents ($LC(\Gamma \vdash A)$ est l'ensemble des termes de type A dans le contexte Γ). Soit $\langle - \rangle$ la famille des démonstrations, i.e., $\langle \Gamma \vdash A \rangle$ est l'ensemble des démonstrations de $\Gamma \vdash A$. En oubliant les termes dans les règles ci-dessus, on obtient une fonction indexée (i.e., préservant les indices) $\varphi : LC \to \langle - \rangle$. On pourrait espérer que pour tous Γ, A, la fonction $\varphi_{\Gamma \vdash A} : LC(\Gamma \vdash A) \to \langle \Gamma \vdash A \rangle$ soit injective.

Voilà un contre-exemple. Dans le contexte vide, l'identité $\lambda . 1$ admet tous les types de la forme $A \Rightarrow A$. Or $\lambda . \lambda . 2$ (qui correspond à $\lambda x . \lambda y . y$) admet tous les types de la forme $B \Rightarrow C \Rightarrow C$. Donc, pour toute formule A, si on prend $B = (A \Rightarrow A)$, on a la démonstration t_A suivante

$$\frac{\dfrac{\dots}{\vdash \lambda . \lambda . 2 : (A \Rightarrow A) \Rightarrow (C \Rightarrow C)} \quad \dfrac{\dots}{\vdash \lambda . 1 : A \Rightarrow A}}{\vdash (\lambda . \lambda . 2)(\lambda . 1) : C \Rightarrow C}.$$

Le terme $(\lambda . \lambda . 2)(\lambda . 1)$ est en même temps l'image d'une infinité de démonstrations de $\vdash C \Rightarrow C$.

Pour remédier à ce problème, il suffit d'annoter tous les λ par le type (la formule) de l'argument. On aurait ici $(\lambda^{A \Rightarrow A} . \lambda^{C} . 2)(\lambda^{A} . 1)$. Ces termes annotés sont appelés termes *à la Church*, notre première tentative étant *à la Curry*. Le λ-calcul à la Curry n'est pas du tout

dénué d'intérêt: il permet de voir la logique comme le typage d'un langage non-typé. Pour l'instant, la correspondance étant plus forte avec le λ-calcul à la Church, on se focalise dessus. La définition est, présentée avec des règles d'inférence:

$$\frac{}{\Gamma \vdash \Gamma(i)}\, i \in \{1, \dots, n\} \quad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B} \quad \frac{\Gamma, A \vdash t : B}{\Gamma \vdash \lambda^A.\, t : A \Rightarrow B}.$$

On a toujours une fonction évidente vers les démonstrations, qu'on note encore φ parce qu'on oublie l'autre, et qui vérifie le résultat suivant:

Proposition 5. Pour tous Γ et A, la fonction $\varphi : \mathrm{LC}_{\mathrm{Church}}(\Gamma \vdash A) \to \langle \Gamma \vdash A \rangle$ est bijective.

Preuve. Pour l'injectivité, on procède par induction; c'est facile. Pour la surjectivité, il est aussi facile de pondre un λ-terme à la Church à partir d'une démonstration, par induction. □

Il devrait sauter aux yeux que les éléments de LC sont des sortes de λ-termes, représentés en se passant des variables. On va formaliser ça.

3.5 Lien syntaxique avec λ

Rendons ça un peu plus explicite. On ne va pas non plus être complètement formels, parce que c'est trop compliqué. On va dire ici que l'ensemble des variables χ de la partie 2 est l'ensemble $\mathbb{N}$ des entiers.

On définit une relation $\equiv$ entre λ-termes purs, qui dit quand deux termes ne diffèrent que par le nom des variables liées. On l'appelle *équivalence structurelle*, où α-équivalence. Pour ça, il faut d'abord définir les *variables libres* et le *renommage* de variables dans un terme. D'abord les variables libres. C'est une fonction de type $\Lambda \to \mathcal{P}(\chi)$. Voilà la définition, par induction:

$$\begin{aligned} \mathrm{fv}(x) &= \{x\} \\ \mathrm{fv}(t_1 t_2) &= \mathrm{fv}(t_1) \cup \mathrm{fv}(t_2) \\ \mathrm{fv}(\lambda x.\, t) &= \mathrm{fv}(t) \setminus \{x\}. \end{aligned}$$

C'est super facile, on est d'accord?

Le renommage est une fonction partielle à trois arguments, de type $(\Lambda \times \chi \times \chi) \rightharpoonup \Lambda$, qu'on note $(t, x, y) \mapsto t\{x \mapsto y\}$. Elle est partielle, mais on va savoir calculer son domaine de définition. Voilà sa définition, par induction sur t:

$$
\begin{array}{rcll}
z\{x \mapsto y\} & = & z & \text{si } z \neq x \\
x\{x \mapsto y\} & = & y & \\
(t_1 t_2)\{x \mapsto y\} & = & (t_1\{x \mapsto y\})\,(t_2\{x \mapsto y\}) & \\
(\lambda x\,.\,t)\{x \mapsto y\} & = & \lambda x\,.\,t & \\
(\lambda y\,.\,t)\{x \mapsto y\} & = & \lambda y\,.\,t & \text{si } x \notin \mathrm{fv}(t) \\
(\lambda y\,.\,t)\{x \mapsto y\} & = & \text{non défini} & \text{si } x \neq y \text{ et } x \in \mathrm{fv}(t)! \\
(\lambda z\,.\,t)\{x \mapsto y\} & = & \lambda z\,.\,(t\{x \mapsto y\}) & \text{si } z \notin \{x, y\}.
\end{array}
$$

Le seul cas de non-définition du renommage est quand la variable x à renommer est sous un λ liant la variable de remplacement y. Le cas le plus simple est $(\lambda y\,.\,x)\{x \mapsto y\}$. On dit que y *capture* x dans le terme et on note ça $\mathrm{capt}_x(\lambda y\,.\,x) = \{y\}$. On peut définir ça en général:

$$
\begin{array}{rcll}
\mathrm{capt}_x(z) & = & \emptyset & \\
\mathrm{capt}_x(x) & = & \emptyset & \\
\mathrm{capt}_x(t_1 t_2) & = & (\mathrm{capt}_x(t_1)) \cup (\mathrm{capt}_x(t_2)) & \\
\mathrm{capt}_x(\lambda x\,.\,t) & = & \emptyset & \\
\mathrm{capt}_x((\lambda y\,.\,t)) & = & \emptyset & \text{si } x \notin \mathrm{fv}(t) \\
\mathrm{capt}_x((\lambda y\,.\,t)) & = & \{y\} \cup \mathrm{capt}_x(t) & \text{si } x \neq y \text{ et } x \in \mathrm{fv}(t)
\end{array}
$$

On a:

Proposition 6. Pour tout $t \in \Lambda$, $t\{x \mapsto y\}$ est défini ssi $y \notin \mathrm{capt}_x(t)$.

On peut enfin définir l'équivalence structurelle $\equiv$. C'est, par définition, la plus petite relation d'équivalence telle que

$$
\frac{}{\lambda x\,.\,t \equiv \lambda y\,.\,(t\{x \mapsto y\})}\ (\text{pour } y \notin \mathrm{capt}_x(t)) \qquad \frac{t_1 \equiv t'_1 \quad t_2 \equiv t'_2}{(t_1 t_2) \equiv (t'_1 t'_2)} \qquad \frac{t \equiv t'}{\lambda x\,.\,t \equiv \lambda x\,.\,t'}.
$$

Quand on dit «relation d'équivalence», ça veut dire qu'on ajoute les règles

$$
\frac{t_1 \equiv t_2 \quad t_2 \equiv t_3}{t_1 \equiv t_3} \qquad \frac{}{t \equiv t} \qquad \frac{t_1 \equiv t_2}{t_2 \equiv t_1}.
$$

(Ici, en fait, la transitivité et la symétrie pourraient être omises: elles seraient des conséquences de la définition.)

Exercice 11. Quelles équivalences sont vraies? Ou plutôt, sous quelles conditions ces équivalences sont-elles vraies (la réponse pouvant être «jamais»)?

- $\lambda x\,.\,x \equiv \lambda x\,.\,x$.

- $\lambda x\,.\,x \equiv \lambda y\,.\,y$.
- $\lambda x\,.\,(x\,y) \equiv \lambda y\,.\,(y\,y)$.
- $\lambda x\,.\,y \equiv \lambda x\,.\,z$.

Exercice 12. Pourquoi la réflexivité ($\frac{}{t \equiv t}$) est-elle nécessaire? Imaginons qu'on ajoute la règle $\frac{}{x \equiv x}$ pour toute variable x. Pourquoi la réflexivité est-elle toujours nécessaire?

L'ensemble des λ-termes purs, considéré informellement en partie 2, est $\Lambda/\equiv$, i.e., l'ensemble des classes d'équivalence modulo $\equiv$.

On adapte maintenant ces techniques au cas typé. Soit Λ^{ST} l'ensemble engendré par la grammaire

$$t, u ::= x \mid t\,u \mid \lambda^A x\,.\,t$$

pour $A \in \mathcal{F}(\Sigma, R)$ et $x \in \chi$. On va maintenant définir un prédicat dit de *typage* pour ces termes.

Un *contexte* Θ est une liste finie de paires (x, A) d'une variable x et d'une formule A, telle que chaque variable apparaisse au plus une fois. On note $x : A$ pour la paire (x, A), $\theta(x) = A$ pour $(x, A) \in \theta$ et $\mathrm{dom}(\theta)$ pour l'ensemble des variables x telles que que $\theta(x)$ est défini.

On définit le typage comme une relation ternaire entre un contexte Θ, une formule $A \in \mathcal{F}(\Sigma, R)$ et un terme $t \in \Lambda^{ST}$. C'est la plus petite relation satisfaisant les règles d'inférence suivantes:

$$\frac{}{\Theta \vdash x : A}\theta(x) = A \qquad \frac{\Theta \vdash t : A \Rightarrow B \quad \Theta \vdash u : A}{\Theta \vdash t\,u : B} \qquad \frac{\Theta, y : A \vdash t\{x \mapsto y\} : B}{\Theta \vdash \lambda^A x\,.\,t : A \Rightarrow B}$$

$$\text{pour } y \notin \mathrm{dom}(\Theta) \cup \mathrm{capt}_x(t).$$

On obtient un ensemble de termes «crus» Λ^{ST} dont les λ sont annotés par des formules, plus un prédicat de typage. On notera $\Lambda^{ST}(\Theta \vdash A)$ l'ensemble des termes $t \in \Lambda^{ST}$ tels que $\Theta \vdash t : A$ est dérivable avec les règles ci-dessus.

On définit la relation d'équivalence structurelle sur Λ^{ST} exactement comme dans le cas non typé. On la notera aussi $\equiv$. On peut même le faire presque formellement en deux lignes comme suit. On a une fonction évidente $\epsilon : \Lambda^{ST} \to \Lambda$ consistant à oublier les annotations de type. On décrète que deux termes typés t et u sont équivalents quand $\epsilon(t) \equiv \epsilon(u)$ et t et u ont les mêmes annotations de types. C'est cette dernière partie qui est légèrement informelle, mais c'est pas la plus dure.

Comme on a décrété qu'on prenait comme ensemble de variables χ les entiers, on a une fonction $U : \mathrm{Contextes} \times \mathcal{F}(\Sigma, R) \to \mathrm{Sequents}$, qui envoie $\Theta \vdash A$ sur $\mathrm{list}(\Theta) \vdash A$, où $\mathrm{list}(\Theta)$ est la liste des formules apparaissant dans Θ, dans l'ordre d'apparition. Par exemple, $\mathrm{list}(3 : A, 2 : C) = (A, C)$. De même, on sait définir une fonction $\psi^{cru} : \Lambda^{ST}(\Theta \vdash A) \to \mathrm{LC}_{\mathrm{Church}}(\mathrm{list}(\Theta) \vdash A)$, pour tous Θ et A. En notant $\mathrm{indice}(\theta, x)$ la position de $(x, \theta(x))$ dans θ, on pose

$$\begin{aligned}
\psi^{\mathrm{cru}}_{\Theta\vdash A}(x) &= \mathrm{indice}(\theta\,,\,x)\\
\psi^{\mathrm{cru}}_{\Theta\vdash A}(t\,u) &= \psi^{\mathrm{cru}}_{\Theta\vdash A}(t)\,\psi^{\mathrm{cru}}_{\Theta\vdash A}(u)\\
\psi^{\mathrm{cru}}_{\Theta\vdash A}(\lambda^{B}x\,.\,t) &= \lambda^{B}.\,(\psi^{\mathrm{cru}}_{\Theta,y:B\vdash C}(t\{x\mapsto y\}))
\end{aligned}$$

avec, dans le dernier cas, $A = (B \Rightarrow C)$ et $y \notin \mathrm{dom}(\theta) \cup \mathrm{capt}_x(t)$.

Cette définition passe au quotient pour donner $\psi_{\Theta\vdash A} : \Lambda^{\mathrm{ST}}(\Theta \vdash A)/{\equiv} \to \mathrm{LC}_{\mathrm{Church}}(\mathrm{list}(\Theta) \vdash A)$. La fonction ψ ne peut pas vraiment être une bijection indexée par les paires $\theta \vdash A$, puisque son domaine est indexé par les paires $\Theta \vdash A$ d'un contexte et d'une formule, alors que son codomaine est indexé par les séquents $\Gamma \vdash A$.

On a en revanche une bijection fibre à fibre entre Λ^{ST} et $\mathrm{LC}_{\mathrm{Church}}$:

Proposition 7. Pour tous Θ et A, la fonction $\psi_{\Theta\vdash A}$ est bijective.

Preuve. L'injectivité découle du fait que la définition de ψ^{cru} ne dépend jamais des noms de variables. La surjectivité est conséquence du fait que $\mathbb{N}$ soit infini. □

Pour comprendre la différence avec ce que serait une bijection indexée, il faut voir cette bijection comme une bijection indexée $(\Lambda^{\mathrm{ST}}/{\equiv}) \to \mathrm{list}^{\star}(\mathrm{LC}_{\mathrm{Church}})$, où $\mathrm{list}^{\star}(\mathrm{LC}_{\mathrm{Church}})$ est obtenu par produit fibré:

$$\begin{array}{ccc}
\mathrm{list}^{\star}(\mathrm{LC}_{\mathrm{Church}}) & \longrightarrow & \mathrm{LC}_{\mathrm{Church}} \\
\downarrow & & \downarrow \\
\mathrm{Contextes}\times\mathcal{F}(\Sigma\,,\,R) & \xrightarrow[\mathrm{list}]{} & \mathrm{Sequents}
\end{array}$$

On a donc une bijection indexée *au sens des fibres de* Λ^{ST}.

Dans l'autre sens, on peut trouver une *section* $\overline{\mathrm{list}} : \mathrm{Sequents} \to (\mathrm{Contextes}\times\mathcal{F}(\Sigma\,,\,R))$, i.e., telle que pour tous Γ et A, on ait $\mathrm{list}(\overline{\mathrm{list}}(\Gamma \vdash A)) = (\Gamma \vdash A)$. (On choisit par exemple d'envoyer $A_1, \ldots, A_n$ sur $1 : A_1, \ldots, n : A_n$.)

3.6 Fonctions structurelles sans variables

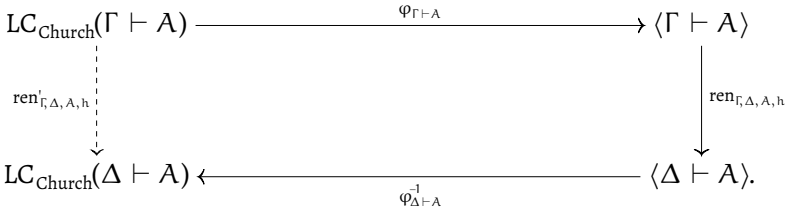
En partie précédente, on a exhibé une correspondance bijective, pour chaque séquent, entre les λ-termes à la Church typés et les démonstrations, via LC:

$$\langle \Gamma \vdash A\rangle \xrightarrow{\varphi_{\Gamma\vdash A}} \mathrm{LC}(\Gamma \vdash A) \xleftarrow{\psi_{\Theta\vdash A}} \left(\Lambda^{\mathrm{ST}}(\Theta \vdash A)/{\equiv}\right),$$

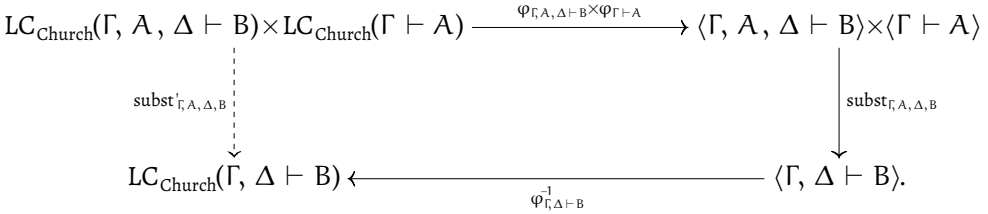
si $\mathrm{list}(\Theta) = \Gamma$ (et un tel Θ existe pour tout Γ).

Peut-on en plus établir une correspondance entre la β-réduction et l'élimination des coupures? Pour commencer, nous considérons dans cette partie le transport le long de φ des fonctions ren et subst, de renommage et de substitution.

Observons d'abord que la bijection indexée de la proposition 5 induit des fonctions



et



Dans la suite, on va nommer ces fonctions ren et subst aussi, le contexte indiquant s'il s'agit de fonctions sur les démonstrations ou sur les éléments de LC. On peut calculer ces fonctions sur les λ-termes directement et on s'aperçoit qu'elles ne dépendent pas des types:

Proposition 8. Si un λ-terme t est à la fois dans $LC_{Church}(\Gamma \vdash A)$ et $LC_{Church}(\Delta \vdash B)$ et si $|\Gamma| = |\Delta| = n$, alors pour tout $h : n \to m$ venant à la fois d'un renommage $\Gamma \to \Gamma'$ et d'un renommage $\Delta \to \Delta'$, alors $ren_{\Gamma,\Gamma',A,h}t = ren_{\Delta,\Delta',B,h}t$.

En fait, on observe mieux: la fonction ren ne dépend que de h (si on considère que h porte son domaine et son codomaine, ici n et m). On a:

$$\begin{aligned} ren_h i &= h(i) && \text{si } i \leqslant n \\ ren_h i &= i + (m - n) && \text{si } i > n \\ ren_h(t_1 t_2) &= (ren_h t_1)(ren_h t_2) \\ ren_h(\lambda.\, t) &= \lambda.\, (ren_{h+1} t), \end{aligned}$$

où $h+1$ désigne la fonction évidente $(n+1) \to (m+1)$.

Pour la substitution $LC_{Church}(\Gamma, A, \Delta \vdash B)\times LC_{Church}(\Gamma \vdash A) \to LC_{Church}(\Gamma, \Delta \vdash B)$, le calcul ne dépend que de la taille de Δ et de l'indice de A dans Γ, A, Δ. La taille de Δ augmente de 1 quand on passe sous un λ; l'indice de A ne change pas. On pourrait donc noter ça:

$$
\begin{aligned}
j[i \mapsto u]_n &= j && \text{si } j < i \\
i[i \mapsto u]_n &= \mathrm{ren}_{h_{i,n}}(u) \\
j[i \mapsto u]_n &= j-1 && \text{si } i < j \\
(t_1 t_2)[i \mapsto u]_n &= \left(t_1[i \mapsto u]_n\right)\left(t_2[i \mapsto u]_n\right) \\
(\lambda.\, t)[i \mapsto u]_n &= \lambda.\, \left(t[i \mapsto u]_{n+1}\right),
\end{aligned}
$$

où on définit, pour tous $1 \leqslant i \leqslant n$, $h_{i,n} : (i-1) \to (i-1+n)$ par $h(j) = j$ pour tout $j < i$.

Exercice 13. Calculer $(\lambda.\, 2)[1 \mapsto \lambda.\, 2]_1$. Réponse: $\lambda.\, (\lambda.\, 3)$.

3.7 Fonctions structurelles avec variables

Examinons maintenant le transport des fonctions structurelles ren et subst qu'on vient de calculer le long de ψ. Ça se passe un peu différemment puisqu'on peut définir ici la substitution directement sur Λ^{ST}, i.e., sur les termes annotés non forcément typés, sans information supplémentaire. On commence par la définir comme une fonction partielle sur les termes crus:

$$
\begin{aligned}
x[x \mapsto t] &= t \\
y[x \mapsto t] &= y && \text{si } x \neq y \\
(t_1 t_2)[x \mapsto t] &= (t_1[x \mapsto t])(t_2[x \mapsto t]) \\
(\lambda x.\, t')[x \mapsto t] &= \lambda x.\, t' \\
(\lambda y.\, t')[x \mapsto t] &= \lambda y.\, (t'[x \mapsto t]) && \text{si } x \notin \mathrm{fv}(t') \text{ ou } y \notin \mathrm{fv}(t) \\
(\lambda y.\, t')[x \mapsto t] &= \text{non défini} && \text{sinon.}
\end{aligned}
$$

Cette fonction partielle $\Lambda^{ST} \times \chi \times \Lambda^{ST} \rightharpoonup \Lambda^{ST}$ a pour domaine de définition les triplets (u, x, t) tels que $\mathrm{fv}(t) \cap \mathrm{capt}_x(u) = \emptyset$. Elle devient bien sûr totale quand on passe au quotient. Attention, d'habitude quand on passe au quotient, on vérifie que la fonction donne la même chose sur tous les représentants d'une classe d'équivalence. Ici, on vérifie quelque chose d'un peu plus faible, mais qui suffit à définir une fonction: on vérifie que toute classe d'équivalence admet un représentant sur lequel la fonction est définie et que dès qu'elle l'est sur deux représentants différents, leurs images sont équivalentes.

Proposition 9. On obtient une fonction subst : $(\Lambda^{ST}/\equiv) \times \chi \times (\Lambda^{ST}/\equiv) \to (\Lambda^{ST}/\equiv)$.

On note $t[x \mapsto u]$ pour $\mathrm{subst}\, t\, x\, u$.

Cette seule fonction préserve le type:

Proposition 10. Pour tous $t \in (\Lambda^{ST}/\equiv)(\Theta, x : A, \Theta' \vdash B)$ et $u \in (\Lambda^{ST}/\equiv)(\Theta \vdash A)$, on a $t[x \mapsto u] \in (\Lambda^{ST}/\equiv)(\Theta, \Theta' \vdash B)$.

On s'appuie sur un lemme dit d'*affaiblissement*:

Lemme 9. Pour tous Θ, Θ', A et $t \in (\Lambda^{ST}/\equiv)(\Theta \vdash A)$, si $\mathrm{dom}(\Theta) \cap \mathrm{dom}(\Theta') = \emptyset$, alors $t \in (\Lambda^{ST}/\equiv)(\Theta, \Theta' \vdash A)$.

Preuve. Par induction sur t, j'ai la flemme pour l'instant: exercice, ou au tableau. □

Ça donne une preuve de la proposition 10 par induction sur t. A chaque fois, il faut d'une part supposer que le résultat cru est bien défini et d'autre part rappeler (1) qu'il ne dépend que de la classe d'équivalence des arguments et (2) qu'il existe toujours des représentants pour lesquels c'est défini. Pareil, pour l'instant j'avance, ça viendra plus tard.

Enonçons déjà le résultat voulu: le diagramme

$$\begin{array}{ccc} (\Lambda^{ST}/\equiv)(\Theta, x : A, \Theta' \vdash B) \times (\Lambda^{ST}/\equiv)(\Theta \vdash A) & \xrightarrow{\text{subst}} & (\Lambda^{ST}/\equiv)(\Theta, \Theta' \vdash B) \\ \downarrow{\scriptstyle \psi_{\Theta, x:A, \Theta' \vdash B} \times \psi_{\Theta \vdash A}} & & \downarrow{\scriptstyle \psi_{\Theta \vdash B}} \\ \mathrm{LC}_{\mathrm{Church}}(\Gamma, A, \Delta \vdash B) \times \mathrm{LC}_{\mathrm{Church}}(\Gamma \vdash A) & \xrightarrow[\text{subst}]{} & \mathrm{LC}_{\mathrm{Church}}(\Gamma, \Delta \vdash B), \end{array}$$

commute, où $\Gamma = \mathrm{list}(\Theta)$ et $\Delta = \mathrm{list}(\Theta')$.

3.8 Lien calculatoire avec λ

Revenons un moment à la question de l'élimination des coupures. Bien sûr, les bijections $\varphi_{\Gamma \vdash A}$ induisent une relation $\rightsquigarrow$ sur les λ-termes. On va la calculer explicitement. Rappelons la règle pour éliminer une coupure sur $\Rightarrow$:

$$\dfrac{\dfrac{\dfrac{t_1}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{t_2}{\Gamma \vdash A}}{\Gamma \vdash B} \quad \rightsquigarrow \quad \dfrac{\mathrm{subst}_{\Gamma, \varepsilon, A, B}\, t_1\, t_2}{\Gamma \vdash B}.$$

Si on regarde ce qu'elle donne sur les termes, on obtient

$$\left(\lambda^A.\, t_1\right) t_2 \rightsquigarrow t_1[n+1 \mapsto t_2]_0,$$

où n est la longueur de Γ.

Il devrait maintenant sauter aux yeux qu'on est très proches de $\to_\beta$.

On a en fait un résultat qui dit ça formellement en termes de systèmes de transitions étiquetées (voir partie 1.2). On va maintenant considérer Λ^{ST}, $\Lambda^{ST}/\equiv$, LC_{Church} et $\langle - \rangle$ comme des systèmes de transitions étiquetées sur un certain alphabet, puis montrer que φ et ψ sont des bisimulations.

D'abord, notre alphabet sera le graphe $\mathfrak{A}$ ayant pour sommets l'ensemble des séquents et pour arêtes, en plus des arêtes identités, pour chaque séquent $\Gamma \vdash A$, une arête $\beta : (\Gamma \vdash A) \to (\Gamma \vdash A)$.

On a vu que $\langle - \rangle$ est équipé d'une relation de transition $\rightsquigarrow$, qui ne relie que des démonstrations du même séquent. Cette relation fournit un graphe réflexif, qui admet un morphisme évident vers $\mathfrak{A}$, envoyant chaque arête $P \rightsquigarrow Q$ entre preuves de $\Gamma \vdash A$ vers l'arête β correspondante.

Sur les termes de LC_{Church}, on a aussi une relation, également notée $\rightsquigarrow$, qui donne de la même manière un graphe réflexif au-dessus de $\mathfrak{A}$.

Enfin, considérons les termes de Λ^{ST}, pour lesquels c'est plus compliqué. On a en effet pour eux deux relations. D'abord, rappelons qu'on n'avait pas défini $\to_\beta$ très formellement en partie 2.1. Faisons-le ici, en décrétant que $(\lambda x\,.\,t)\,u \to_\beta t[x \mapsto u]$ quand les deux côtés sont définis, i.e., quand $fv(u) \cap capt_x(t) = \emptyset$. Telle quelle, cette relation ne correspond pas exactement à $\to_\beta$ puisque, par exemple $(\lambda x\,.\,\lambda y\,.\,x)\,y$ ne se réduit vers rien. Pour corriger ça, on utilise la relation $\equiv$, et on définit le graphe réflexif dont les sommets sont les termes de Λ^{ST} et les arêtes sont données par $\equiv + \to_\beta$, i.e., par les triplets $(0, t, u)$ telles que $t \equiv u$ et les triplets $(1, t, u)$ tels que $t \to_\beta u$. Comme on a toujours $t \equiv t$, on peut prendre ces arêtes comme arêtes identités. Ce graphe admet un morphisme de graphes réflexifs vers $\mathfrak{A}$, envoyant chaque t dans $\Lambda^{ST}(\theta \vdash A)$ sur $list(\theta) \vdash A$, chaque arête $(0, t, u)$ entre termes de type $\theta \vdash A$ sur $id_{list(\theta)\vdash A}$ et chaque arête $(1, t, u)$ sur l'arête β correspondante. On a alors bien

$$(\lambda x\,.\,\lambda y\,.\,x)\,y \equiv (\lambda x\,.\,\lambda z\,.\,x)\,y \to_\beta \lambda z\,.\,y,$$

ou bien, en termes de $\mathfrak{A}$-transitions,

$$(\lambda x\,.\,\lambda y\,.\,x)\,y \xrightarrow{id} (\lambda x\,.\,\lambda z\,.\,x)\,y \xrightarrow{\beta} \lambda z\,.\,y,$$

donc en particulier $(\lambda x\,.\,\lambda y\,.\,x)\,y \xRightarrow{\beta} \lambda z\,.\,y$.

En fermant le graphe Λ^{ST} par composition avec les arêtes $\equiv$, c'est-à-dire prenant plutôt comme arêtes $t \to_{\beta'} u$ les paires (t, u) telles que $t \equiv t' \to_\beta u' \equiv u$, on obtient un graphe réflexif au-dessus de $\mathfrak{A}$ qui s'étend au quotient, donc un graphe réflexif $\Lambda^{ST}/\equiv$ au-dessus de $\mathfrak{A}$.

On obtient donc graphes réflexifs au-dessus de $\mathfrak{A}$, $\langle - \rangle$, LC_{Church}, Λ^{ST} et $\Lambda^{ST}/\equiv$, avec des morphismes

$$\langle - \rangle \xrightarrow{\varphi} LC_{Church} \xleftarrow{\psi} \Lambda^{ST} \xrightarrow{quotient} \Lambda^{ST}/\equiv .$$

Théorème 4. On a: φ est une bisimulation forte et que ψ et le quotient sont des bisimulations faibles. Enfin, la fonction $\psi : \Lambda^{ST}/\equiv \to LC_{Church}$ étendue au quotient est une bisimulation forte.

La démonstration repose sur le fait que subst commute avec la bijection.

On a donc, pour le fragment propositionnel avec juste $\Rightarrow$, non seulement une bijection entre démonstrations en déduction naturelle et λ-termes simplement typés, mais également une correspondance parfaite entre l'élimination des coupures et la β-réduction.

On appelle ce résultat la correspondance de Curry-Howard, ou peut-être est-ce plutôt son corollaire:

Corollaire 2. Si une démonstration P et un terme t sont reliés par la bijection, alors
- soit P et t divergent tous les deux,
- soit P et t convergent vers des formes normales V et v, reliées par la bijection.

3.9 Déduction naturelle classique

La déduction naturelle classique est obtenue en ajoutant une manière de déduire que $\neg\neg A \Rightarrow A$, pour toute formule A. Toutes les manières de faire ça ne sont pas équivalentes, surtout quand il manque certains connecteurs logiques. On ne va pas entrer ici dans ces considérations, qui font l'objet de travaux de recherche. On se contentera de la formulation qui consiste en l'ajout d'un axiome appelé la loi de Peirce:

$$\frac{}{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

Cette loi permet de déduire $A \vee \neg A$ pour tout A.

4 Bibliography