Structures mathématiques du langage

Alain Lecomte
Université Paris 8 - Vincennes-Saint-Denis
Licence de Sciences du Langage

Table des matières

1 La Théorie Constructive des Types

1.1 De la logique intuitionniste à la théorie constructive des types

Il y a, en logique classique comme en logique intuitionniste un fait troublant : de $\forall x P(x)$ il est possible de déduire $\exists x P(x)$. Cela est troublant car a priori aucune *existence* ne semble impliquée

dans la première formule. Logiquement, *tous les entiers plus grands que 2 et plus petits que 1 sont des nombres premiers multiples de 2* est vraie simplement parce qu'il n'existe pas d'entier plus grand que 2 et plus petit que 1 ! Mais cela n'entraîne pas qu'il existe des nombres premiers multiples de 2 ! Autrement dit, supposons que l'univers soit vide... alors $\forall x P(x)$ est vraie, et $\exists x P(x)$ est nécessairement fausse ! Les logiciens résolvent cette question en posant par définition qu'un univers (ou *domaine*) est un ensemble **non vide**. A vrai dire, ceci est étranger à la logique pure : c'est un principe hétérogène par rapport aux "lois de la logique". Comme si on pouvait prouver l'existence de quelque chose à partir des pures lois de la raison[1] ! On pourrait très bien exiger que les lois de la logique soient vraies alors même que l'univers est vide, mais cela voudrait dire que certaines lois de la logique (comme celle signalée ci-dessus) ne sont pas des lois absolues, mais des lois soumises à certaines hypothèses, par exemple celle selon laquelle le domaine de quantification est non vide. Dans la *Théorie Intuitioniste des Types* de P. Martin-Löf, il en va ainsi. A côté de *jugements* dits *catégoriques*, figurent des *jugements hypothétiques* : ils sont faits sous réserve que certaines hypothèses soient valides.

Par ailleurs, on sait fort bien que, contrairement à ce qu'impose la conception frégéenne, pour laquelle il n'existe qu'un seul univers, et où, donc, le domaine de quantification est toujours cet univers (ce qui fait qu'une phrase comme *tous les chats sont gris* se traduit en $\forall x\ chat(x) \Rightarrow gris(x)$ c'est-à-dire quelquechose d'équivalent à "pour tout individu dans l'univers, si c'est un chat alors il est gris", ce qui est pour le moins peu naturel), nos phrases du langage ordinaire ont des domaines de quantification variables. Elles *dépendent* de ces domaines. Par exemple, la phrase *tous les chats sont gris* se traduirait plutôt par $(\forall x \in CHAT)\ gris(x)$ ou (comme ce sera le cas en théorie des types) $(\forall x\ :\ CHAT)\ gris(x)$. On dira en ce cas que le jugement $gris(x)$ est dépendant du domaine CHAT. Evidemment, à supposer que ce jugement soit vrai, si le domaine CHAT s'avère être vide, l'autre jugement que serait $(\exists x\ :\ CHAT)\ gris(x)$ serait faux : on n'arriverait pas à trouver d'individu a tel que le jugement $a : CHAT$ soit vrai. Peut-être serait-on tenté de dire qu'il n'est pas faux, mais simplement "non prouvé", pourtant en l'occurrence, cela reviendra au même puisque, en admettant la thèse très forte de la Théorie Intuitionniste des Types, selon laquelle

une proposition est identifiée à l'ensemble de ses preuves

dire qu'un tel ensemble de preuves est vide, c'est dire que la proposition est fausse. On notera alors, en passant, que nous identifions aussi la notion d'ensemble et la notion de proposition, dire que le domaine CHAT est vide, c'est comme dire que la proposition *x est un chat* est fausse : ici, une preuve de cette dernière proposition EST un élément de l'ensemble CHAT.

Pour la théorie intuitionniste, la preuve d'une proposition existentielle $(\exists x\ :\ A)B(x)$ consiste en un *couple* formé d'un élément i du domaine A **et** d'une preuve de $B(i)$. Ceci n'est pas très bien reflété dans les règles de déduction naturelle pour la LI. La Théorie des Types de P. Martin-Löf l'explicite.

Dans la notation de la TCT, $a\ :\ A$ signifie que "a est un élément de A", ou "$a \in A$". On peut avoir une suite de déclarations en cascade, par exemple, pour exprimer vraiment que $a\ :\ A$ signifie que a

[1] C'est ainsi d'ailleurs que Saint Anselme croyait pouvoir prouver l'existence de Dieu, au moyen de la pure logique.

est un élément de A, il faudrait spécifier que A est du type Ens, ce qu'on noterait aussi : $A\ :\ Ens$. Dans le cas d'une proposition, on écrit $A\ vraie$ simplement pour dire que A est reconnue comme vraie (parce qu'elle a une preuve, même si on ne sait pas laquelle). On a les règles suivantes :

$$\frac{\begin{array}{c}(x\ :\ A)\\ \vdots\\ B(x)\ vraie\end{array}}{(\forall x\ :\ A)B(x)\ vraie}\,[\forall I] \qquad \frac{a\ :\ A \qquad B(a)\ vraie}{(\exists x\ :\ A)B(x)\ vraie}\,[\exists I]$$

$$\frac{(\forall x\ :\ A)B(x)\ vraie \qquad a\ :\ A}{B(a)\ vraie}\,[\forall E] \qquad \frac{(\exists x\ :\ A)B(x)\ vraie \qquad \begin{array}{c}(x\ :\ A, B(x)\ vraie)\\ \vdots\\ C\ vraie\end{array}}{C\ vraie}\,[\exists E]$$

Dans $[\forall I]$ et $[\exists E]$, la variable x est soumise à la même contrainte de n'apparaître dans aucune autre hypothèse. Dans $[\exists I]$ et $[\forall E]$, a dénote un élément de A, autrement dit on a l'assurance que A est non vide.
On notera la différence de ces règles avec celles de la logique intuitionniste vues plus haut :

- chaque fois, l'assertion (le "jugement") selon laquelle on a pris un élément dans un ensemble (A) et que donc cet ensemble n'est pas vide, est explicite
- tous les jugements sont relatifs à des ensembles particuliers.

Il est intéressant de donner comme exemple la preuve que si pour tout $x \in A$, on a : si $B(x)$ alors $C(x)$ vraie, alors s'il existe un $x \in A$ tel que $B(x)$ soit vraie, il existe nécessairement aussi un $x \in A$ tel que $C(x)$ soit vraie. cf. fig 1. Noter, dans cette preuve, l'introduction explicite de la variable x, sous la forme $(x\ :\ A)$, qui n'est possible bien sûr *que si A est non vide* (si A était vide, on ne pourrait même pas écrire $(x\ :\ A)$ sans contradiction).

1.1.1 Règles de formation

Une autre particularité de la TCT est que les règles de formation des expressions figurent au même niveau que les règles logiques : ce sont donc elles-mêmes des règles logiques. Pour qu'une certaine expression P soit une proposition par exemple (appartienne au type $prop$), il faut que certaines conditions soient remplies. On peut par exemple penser au fait qu'une expression arithmétique x/y est *mal formée* si $y = 0$. Une expression nominale définie est mal formée également si, dans le contexte (cf. la définition de ce mot plus loin), il y a deux objets qui correspondent à la description. Voir plus loin quelques règles de formation.

1.1.2 Jugements

La TCT appelle *jugements* les déclarations (comme dans un programme) du genre :

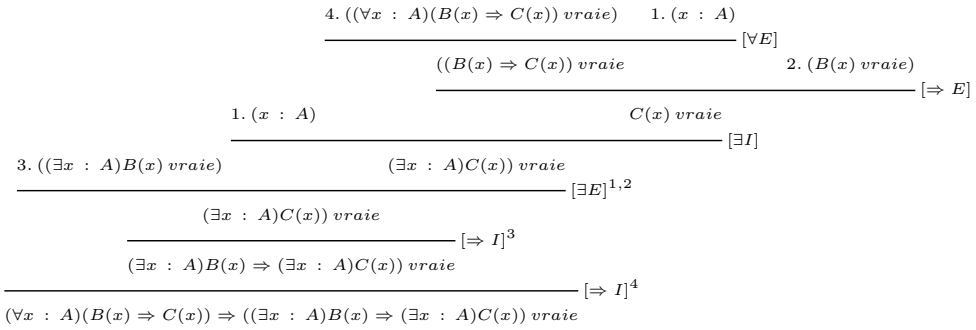

FIG. 1 – *Commentaire : x est introduit comme variable en 1. (qui est répété). L'antécédent de l'implication à démontrer est introduit comme hypothèse en 4. On en déduit par $[\forall E]$ que $(B(x) \Rightarrow C(x))$ est vraie pour cet x "quelconque" qui a été introduit. Avec l'hypothèse 2. selon laquelle on pose que $B(x)$ est vraie (pour le x introduit en 1.) on déduit $C(x)$. Réutilisant le fait que $x \in A$, on peut introduire l'existentielle et obtenir $(\exists x : A)C(x)$. Ainsi étant parti de l'hypothèse 2. valide quel que soit x, et ayant prouvé que sous cette hypothèse, $(\exists x : A)C(x)$ est vraie, on peut maintenantdécharger l'hypothèse 2. et déduire grâce à l'hypothèse 3. que $(\exists x : A)C(x)$ est vraie. Ensuite on peut décharger successivement les hypothèses 3. et 4.*

- $a : A$, pour dire que a est un élément de A, ou une preuve de A
- $A : Ens$, pour dire que A est du type Ens (est un ensemble)

Certains jugements peuvent aussi porter sur des égalités (on peut faire des *déclarations d'égalité*) :

- $a = b : A$, pour dire que a et b sont des éléments égaux de A
- $A = B : Ens$, pour dire que deux ensembles A et B sont égaux.

$A\ vraie$ est aussi un jugement, mais il dérive d'un $a : A$: c'est un jugement de la première espèce dont on a "oublié" l'objet preuve, lequel apportait la véracité de A.
Dans la règle $[\forall I]$, $B(x)$ est $vraie$ sous l'hypothèse que $x \in A$, autrement dit, nous avons là un jugement *hypothétique*. Un tel jugement peut s'écrire aussi :

$$B(x)\ vraie\ \ (x : A)$$

ceci est une manière de *linéariser* le lien entre hypothèse (à droite et entre parenthèses) et jugement. On peut avoir évidemment des jugements "en cascade" comme :

$$B(x)\ vraie\ \ (x : A)\ (A : Ens)$$

La donnée d'un ensemble (ou d'une proposition) peut dépendre d'une hypothèse explicite que l'on a faite précédemment. Par exemple, l'ensemble des jours d'un mois dépend de ce mois. Si $x = juin$, l'ensemble $Jours$ sera : $\{1, 2, ..., 30\}$, si $x = juillet$, $Jours = \{1, 2, ..., 30, 31\}$, il sera donc naturel de faire dépendre l'ensemble $Jours$ d'une variable, on écrira $Jours(x)$ et plus

précisément le jugement :

$$Jours(x) \ : \ Ens \ (x \ : \ mois)$$

pour dire que $Jours(x)$ est un ensemble dépendant (type dépendant). Il dépend d'une hypothèse x de type $mois$.
Une entité peut être définie sous plusieurs hypothèses. L'ensemble de ces hypothèses est alors appelé un *contexte*.
On peut par exemple avoir :

$$A(x_1, ..., x_n) \ : \ Ens \quad [x_1 \ : \ C_1, ..., x_n \in C_n(x_1, ..., x_{n-1})]$$

qui signifie que $A(x_1, ..., x_n)$ est un ensemble qui dépend des hypothèses $x_1, ..., x_n$ avec de plus chaque hypothèse dépendant des précédentes.

1.2 Théorie des types et théorie des ensembles

A première vue, on peut considérer les types comme des ensembles, et la théorie des types se rabattra sur la théorie des ensembles. Mais il y a de nombreuses raisons de ne pas le faire, qui tiennent aux différences entre types et ensembles. La première remarque est que la TCT ne nécessite pas une théorie des ensembles préalable, elle n'est pas fondée sur la théorie des ensembles. Cette dernière en effet possède une relation primitive, l'appartenance ($\in$) et un certain nombre d'axiomes (appelés axiomes de Zermelo Fraenkel) dont l'extensionnalité, et la compréhension. L'extensionnalité repose sur une autre relation primitive, qui est l'égalité entre éléments. La théorie classique des ensembles suppose en effet qu'on parte d'un socle constitué d'éléments de base (ou *urelements*) et d'une relation d'égalité qui consiste en fait sur une possibilité d'identifier ces éléments (on sait quand $x = y$: c'est quand x et y désignent le même élément !). L'axiome de compréhension, de son côté, indique que quelle que soit la propriété formulable en logique des prédicats, on peut toujours fabriquer un ensemble à partir d'un autre ensemble,E, en prenant tous ceux de E qui vérifient la propriété en question. D'où une prolifération inquiétante d'ensembles ! Une théorie des types peut très bien passer outre ce genre d'exigence. D'abord, on peut imaginer qu'il y aura une relation d'égalité spécifique pour chaque type, ensuite on peut imaginer que toute propriété ne donne pas forcément un nouveau type. Autre remarque : implicitement, la théorie des ensembles dépend de la logique des prédicats, alors que la théorie des types englobe une telle logique (les deux se développent en même temps). De fait, la TCT s'inscrit dans la continuation directe de la logique intuitionniste et peut ignorer au passage la théorie des ensembles. Evidemment, si les types ne sont plus des ensembles, on se prive de beaucoup de commodités associées avec la notion d'ensemble. Par exemple, il y a quantité de relations "naturelles" qui viennent avec les ensembles, comme les notions d'union, d'intersection et d'inclusion. Développer un cadre hors de la TE implique que l'on se passe de tout cela ou bien qu'on le remplace par autre chose. En particulier, la relation d'inclusion est fondamentale si on veut parler de sous-types et dire par exemple que le type $[human]$ est un sous-type du type $[animate]$. D'où les recherches qui existent sur la notion de sous-typage et dont nous dirons un mot plus loin.

La notion d'ensemble exige que, lorsque nous considérons un ensemble, A, nous sachions toujours dire, étant donné un élément quelconque x si $x \in A$ ou non. La notion de type ne l'exige pas. Elle permet juste que l'on formule des jugements, du genre $a : A$, pour dire que a est de type A, mais ne nous demande pas nos raisons pour proposer un tel jugement ! Nous n'avons donc pas besoin de définir un *type* à partir de critères rigides. Je peux ainsi émettre le jugement que tel objet est du type $[vivant]$ même si je n'ai pas à ma disposition une définition précise de la notion d'être vivant. La théorie des types se préoccupe des *jugements*, pas des définitions extensionnelles des types. Evidemment, elle s'intéresse aussi aux preuves que nous pouvons avoir de ces jugements ! Mais ces preuves peuvent dériver d'autres jugements posés comme axiomes ou comme hypothèses (ce sont alors des jugements hypothétiques), sans que l'on sache toujours quelle est au juste l'extension d'un type ! A vrai dire, on se moque un peu des extensions...
Noter que demander si quelqu'individu est un $[enfant]$ n'a de sens que si l'individu en question est un $[human]$ (pas si c'est un nombre entier par exemple !), de même demander si un individu est $[human]$ n'a de sens que s'il est $[animate]$, et que par ailleurs, on admet généralement que $[enfant]$ est un sous-type de $[human]$, or si nous définissons $[enfant]$ comme $[human] \rightarrow prop$ et $[human]$ comme $[animate] \rightarrow prop$, puisque $[human]$ est un sous-type de $[animate]$, nous obtenons (par contravariance) que... $[human]$ est un sous-type de $[enfant]$! On comprendra donc que les noms communs ne soient pas considérés comme des propriétés, c'est-à-dire des fonctions, mais comme des types propres (qu'on peut toujours considérer comme sous-types d'un type *univers* associé généralement aux noms communs, CN). Ceci dit, le problème sera reporté au niveau des verbes. Il est difficile de faire autrement que considérer un verbe, par exemple intransitif, comme une propriété ayant pour domaine un certain type. Par exemple *être pair* est défini pour les nombres et seulement pour les nombres. *Penser* est défini pour les $[human]$ et seulement pour eux, alors que *manger* est défini pour tous les $[vivants]$. *Penser* appartient aux activites humaines, alors que *manger* appartient aux activités des vivants. On a évidemment envie de dire que les activités humaines constituent un sous-type des activités des vivants, autrement dit de définir un typage des verbes qui dépend des types de leurs actants, seulement voilà, la contravariance empêche de le faire de manière cohérente.
Ces questions ont trait de manière générale aux problèmes de *sous-typage*. Si on évite le recours à la théorie des ensembles, qui permettrait facilement de définir A comme sous-type de B en disant que l'extension de A est un sous-ensemble de l'extension de B, il faudra formuler explicitement des *jugements* de sous-typage par le biais de fonctions dites fonctions de *coercion*, et évidemment il faudra vérifier la cohérence de ces jugements. Autrement dit, étant donnée une théorie des types $\mathcal{T}$, l'étendre en une théorie $\mathcal{T}[\mathcal{C}]$ où $\mathcal{C}$ est une famille de jugements de sous-typage. Il n'est pas dit qu'on puisse pour autant résoudre les problèmes vus précédemment.
Nous avons vu également dans la section précédente que le polymorphisme était possible via des variables de types et une quantification sur ces variables, ainsi peut-on avoir des objets d'un type $\Pi X.T[X]$ où $T[X]$ est un type avec variable. Le problème qui apparaît ici est le fait que, si Π parcourt tous les types... il se rencontre nécessairement lui-même dans son parcours ! ainsi une instance particulière du type $\Pi X.T[X]$ sera $T[\Pi X.T[X]]$. Une théorie qui, comme ici, fait référence aux objets en train d'être définis dans la définition qu'elle donne de ces objets est dite *imprédica-*

tive. En général, l'imprédicativité peut poser des problèmes. Nous verrons que, dans le système F, ces problèmes sont résolus.

2 Usage de la théorie constructive des types

La loi absolue de la TCT est qu'on ne saurait parler d'une chose sans l'avoir rendue explicite soit par une méthode de construction, soit par une preuve. Une preuve est constructive au sens où elle permet de construire des objets : ce sont de véritables programmes qui permettent de calculer ce dont la preuve est preuve de l'existence. Il faut donc reprendre les règles et les compléter avec les objets construits. Les règles d'introduction sont associées à des *constructeurs* et les règles d'élimination à des *sélecteurs*. Par exemple, introduire une conjonction, c'est introduire un opérateur de *pairing* (fabriquer un couple à partir de deux éléments donnés), éliminer une conjonction c'est "casser" le couple, autrement dit *projeter* le couple sur chacune de ses composantes.

$$\frac{a \ : \ A \qquad b \ : \ B}{(a,b) \ : \ A \wedge B}\,[\wedge I] \qquad \frac{u \ : \ A \wedge B}{p(u) \ : \ A}\,[\wedge E]_1 \qquad \frac{u \ : \ A \wedge B}{q(u) \ : \ B}\,[\wedge E]_2$$

Comme une conjonction est interprétée par un couple, il n'y a pas de différence entre une conjonction de deux propositions et un produit cartésien de deux ensembles (une proposition étant un ensemble : l'ensemble de ses preuves). Il existe une autre règle d'élimination : si en faisant deux hypothèses A et B, on prouve une proposition C, alors on peut décharger les deux hypothèses en même temps grâce à $A \wedge B$ (ou $A \times B$). La règle logique est :

$$\frac{\begin{array}{cc} & \begin{array}{cc}[A] & [B] \\ \cdot & \cdot \\ \cdot & \cdot \end{array} \\ A \wedge B & C \end{array}}{C}$$

On voit que les deux demi-règles introduites plus haut sont des cas particuliers de cette règle, ceux pour lesquels respectivement $C = A$ et $C = B$:

$$\frac{\begin{array}{cc} \begin{array}{cc}[A] & [B] \\ \cdot & \cdot \\ \cdot & \cdot \end{array} & \\ A \wedge B & A \end{array}}{A} \qquad \frac{\begin{array}{cc} \begin{array}{cc}[A] & [B] \\ \cdot & \cdot \\ \cdot & \cdot \end{array} & \\ A \wedge B & B \end{array}}{B}$$

Maintenant supposons que les hypothèses A et B soient respectivement associées aux variables x et y et que la preuve de C à partir de ces hypothèses soit une fonction d de x et de y : $d(x,y)$, supposons que c soit la preuve associée à $A \wedge B$, l'utilisation de cette règle permet de combiner c et $d(x,y)$. On notera E la fonction qui réalise cette action. Le résultat est alors $E(c,(x,y)d(x,y))$.

On écrit (x, y) devant $d(x, y)$ pour indiquer que les variables en question deviennent liées lors de l'application de la règle. On a :

$E(c, (x, y)x) = p(c)$ et $E(c, (x, y)y) = q(c)$

La règle finale s'écrit :

$$\dfrac{\begin{array}{ccc} & [x : A] & \qquad\qquad [y : B] \\ & \vdots & \qquad\qquad \vdots \\ c : A \wedge B & d(x, y) : C((x, y)) & \end{array}}{E(c, (x, y)d(x, y)) : C(c)}$$

où la conclusion C est explicitement représentée comme dépendant du couple d'hypothèses (x, y). La règle substitue alors l'élément c donné du produit à la place du couple (x, y) obtenu à partir des hypothèses A et B.

Introduire une disjonction, c'est utiliser une *injection* i de A vers $A + B$ si on a une preuve de A ou une injection j de B vers $A + B$ si on a une preuve de B. Cela se représente par :

$$\dfrac{a \ : \ A}{i(a) \ : \ A \vee B}\,[\vee I] \qquad\qquad \dfrac{b \ : \ B}{j(b) \ : \ A \vee B}\,[\vee I]$$

L'éliminer c'est utiliser un schéma analogue au précédent pour la conjonction sauf en ce que, cette fois, c'est chaque hypothèse (A ou B) indépendamment qui doit permettre de déduire C.

$$\dfrac{\begin{array}{ccc} [x : A] & & [y : B] \\ \vdots & & \vdots \\ c : A \vee B \quad d(x) : C(i(x)) & & e(y) : C(j(y)) \end{array}}{D(c, (x)d(x), (y)e(y))) : C(c)}$$

On a : $D(i(a), (x)d(x), (y)e(y))) = d(a)$ et $D(j(b), (x)d(x), (y)e(y))) = e(b)$

Cela signifie que C dépend d'un élément qui soit vient de A si c'est A dont on a une preuve, soit vient de B si c'est B dont on a une preuve.

Il est bien sûr intéressant d'étendre ces règles aux quantificateurs. Commençons par la quantification existentielle. La particularité étrange de la TCT est que l'existentielle y est associée à la conjonction, tout simplement parce que :

avoir une preuve de $(\exists x \in A)\ B(x)$ *c'est avoir un* **couple** *formé d'un élément* $a \ : \ A$ **et** *d'une preuve de* $B(a)$

donc l'ensemble des preuves de $(\exists x \in A)\ B(x)$ est l'ensemble des couples (a, b) où a est un élément de A et b un élément de $B(a)$. Si tous les $B(a)$ sont identiques, on retrouve l'idée de produit

cartésien, mais en général, on a une généralisation de ce type de produit où la deuxième composante dépend de l'élément choisi dans la première. D'une façon générale, on notera $(\Sigma x\ :\ A)B(x)$ ce produit.
Règle d'introduction :

$$\frac{a\ :\ A \qquad b\ :\ B(a)}{(a,b)\ :\ (\Sigma x\ :\ A)B(x)}\,[\Sigma I]$$

Règles d'élimination :

$$\frac{c\ :\ (\Sigma x\ :\ A)B(x)}{p(c)\ :\ A}\,\Sigma E \qquad\qquad \frac{c\ :\ (\Sigma x\ :\ A)B(x)}{q(c)\ :\ B(p(c))}\,\Sigma E$$

Règle d'élimination plus générale :

$$\frac{\begin{array}{cc} & (x\ :\ A, y\ :\ B(a)) \\ & \vdots \\ c\ :\ (\Sigma x\ :\ A)B(x) & d(x,y)\ :\ C((x,y)) \end{array}}{E(c,(x,y)d(x,y))\ :\ C(c)}\,\Sigma E$$

avec :

$$E(c,(x,y)x) = p(c)\ :\ A$$
$$E(c,(x,y)y) = q(c)\ :\ B(p(c))$$

Quantification universelle :
On a de façon similaire à l'existentielle :

$$\frac{\begin{array}{c}(x\ :\ A) \\ \vdots \\ b(x)\ :\ B(x)\end{array}}{(\lambda x)b(x)\ :\ (\Pi x\ :\ A)B(x)}\,\Pi I$$

ce qui signifie :
si à partir d'une hypothèse x de type A, on peut prouver $B(x)$ (donc au moyen d'une preuve $b(x)$), x n'apparaissant dans aucune autre hypothèse, alors on a prouvé $(\Pi x\ :\ A)B(x)$. $(\Pi x\ :\ A)B(x)$ est identique à $A \Rightarrow B$ dans le cas où B ne dépend pas de l'hypothèse x. En termes d'ensembles, il s'agit de l'ensemble des applications de A dans B, ou plus précisément de toutes les méthodes permettant d'obtenir une preuve de $B(a)$ à partir d'une preuve a de A.
Règles de formation :

$$\frac{\begin{array}{cc} & (x\ :\ A) \\ & \vdots \\ A\ :\ Ens & B(x)\ :\ Ens \end{array}}{(\Sigma x\ :\ A)B(x)\ :\ Ens}\,\Sigma F \qquad\qquad \frac{\begin{array}{cc} & (x\ :\ A) \\ & \vdots \\ A\ :\ Ens & B(x)\ :\ Ens \end{array}}{(\Pi x\ :\ A)B(x)\ :\ Ens}\,\Pi F$$

3 De la Théorie Constructive des Types au langage ordinaire

3.1 Des évènements aux objets-preuves

L'un des intérêts majeurs de l'approche présentée ici réside dans sa richesse vis à vis du problème de la caractérisation du sens d'une proposition. Le point de vue frégéen se résume en une caractérisation très pauvre : une proposition dénote soit le vrai, soit le faux. Ce n'est que la théorie des mondes possibles qui permet de l'affiner : la signification d'une proposition est alors l'ensemble des mondes possibles où elle reçoit la valeur "vrai". Mais la théorie des mondes possibles est une monstruosité... De quels mondes possibles s'agit-il ? Où s'arrête le possible ? Y a-t-il des mondes possibles ayant des lois logiques différentes de nos lois dans le monde actuel ? On en vient à définir.... des mondes possibles impossibles ! etc. etc.
En réalité, la logique intuitionniste donne directement au langage ce qu'on va chercher à grand frais dans les mondes possibles, à savoir une théorie où la signification d'une proposition peut s'identifier à un ensemble, seulement il ne s'agit pas d'un ensemble de "mondes", mais beaucoup plus concrètement, d'un ensemble de *preuves*. Bien sûr, *hors mathématiques*, cela semble très spéculatif : quelles preuves pour une proposition ordinaire ? Aarne Ranta (p. 54) discute ce point en reprenant l'exemple suivant à Davidson :

> The sentence *Amundsen flew over the North Pole* is made true by a flight made by Amundsen over the North Pole.

Ce faisant, il tend à identifier la *preuve* d'une affirmation élémentaire à *l'évènement* qui produit ce dont elle parle. D'autres approches, comme celle de Mulligan, Simons et Smith, identifieraient plutôt la *preuve* à un "*truthmaker*", ce qu'on pourrait traduire en français par "*vérifacteur*"[2][3].
Ranta prend en compte les objections selon lesquelles la notion d'évènement demeure trop indécise : une guerre, par exemple, est-elle un seul évènement ou bien des millions ? La Théorie des Types permet de considérer autant de *types* d'évènements que souhaité. Il suffit simplement chaque fois de spécifier ce que c'est qu'être un objet du type spécifié (à partir d'éléments canoniques) et ce que c'est pour deux objets du même type d'être égaux. On peut ainsi envisager un type pour les guerres et un type pour les tirs. Une autre objection relevée consiste en ce que, à la différence des mathématiques, les objets qui nous intéressent (ici des guerres et des tirs, mais aussi des vols en avion ou des voyages au Pôle Nord) ne sont jamais *pleinement présentés*. Un nombre entier est pleinement représenté par son expression canonique, mais il n'y a pas à première vue d'élément canonique à quoi se ramènent une guerre ou un vol en avion... Ranta dit même :

> *Even the longest encyclopedic text will leave an infinity of properties open, and it is a puzzling question what expression, if any, would determine Africa as a continent.*

[2] et non "vérificateur", le *vérifacteur* étant ce qui rend vrai alors que le *vérificateur* est ce qui vérifie la vérité, mais on sent bien dans cette alternance - qui se joue sur deux lettres - la différence entre deux options philosophiques, *constructivisme* et *réalisme*. Nous n'entrerons pas ici dans ce débat.

[3] L'idée aristotélicienne reprise par Mulligan et al. consiste à mettre l'accent sur le rapport entre un *moment* d'une chose, ou substance, qui, en tant que moment est un accident, et un élément de cette substance, qui fait figure de *fondement*. Ainsi, ce qui rend vraie la phrase *Jean est triste*, c'est la tristesse de Jean.

Bien entendu ces critiques concernent aussi la logique classique et la sémantique formelle.
Les réponses à ces objections consistent à suggérer que nous travaillions avec des *types* plutôt qu'avec des *ensembles* (les ensembles nécessitent des règles d'introduction faisant usage de la définition canonique des éléments, ce n'est pas le cas des *types* en général), à développer des techniques d'approximation pour atteindre des objets non pleinement présentés (comme on le fait avec les nombres réels), ou bien à se concentrer sur des modèles de langage très restreints, autrement dit des "jeux de langage" au sens de Wittgenstein, des cas donc où on n'a pas besoin de la définition exhaustive de ce qu'est un *humain* pour manipuler la notion (à partir d'un ensemble fini en ce cas, permettant de dire "est un humain" ce qui appartient à tel ensemble donné en énumération). A ces solutions, nous en ajouterons plus loin une autre qui consisterait à *prendre acte de l'infinitude des notions du langage courant*, sous la forme de la considération d'objets-preuves infinis, avec l'idée que bien entendu, localement, nous n'en captons qu'une "fenêtre" finie. Une telle fenêtre serait obtenue comme partie de l'objet infini interagissant avec des objets du même genre, mais qui, eux, seraient finis. En assimilant *preuves* et *méthodes de construction*, cela reviendrait donc à prendre en compte des sortes de processus de preuve infinis, ce qu'autorisera la *ludique*. Concrètement, nous n'avons pas de caractérisation finie de ce qu'est une guerre, mais nous savons utiliser la notion parce que nous sommes capables de sélectionner dans sa caractérisation potentiellement infinie quelques éléments qui font que l'objet que nous avons en vue (une soi-disant guerre) passe avec succès (ou non) un certain nombre de *tests*, ces tests étant eux-mêmes inclus dans des processus de preuve (qui eux-mêmes peuvent être testés etc.).

3.2 Contextes

En Théorie des Types, un contexte est une suite d'hypothèses de la forme :

$$x_1 \ : \ A_1, x_2 \ : \ A_2(x_1), ..., x_n \ : \ A_n(x_1, ..., x_n)$$

Implicitement chaque hypothèse dépend de la suite des précédentes. Par exemple, on pourrait imaginer un contexte justifiant une phrase comme *Pierre a encore fait une fugue*. Un tel contexte commencerait par une fugue (donc un "évènement" de type *fugue*) une première fois, puis continuerait par une autre fugue, mais cette fois exécutée dans le contexte de la précédente (ce qui justifie que dès ce moment, le locuteur puisse dire *encore*) et ainsi de suite, jusqu'à une fugue effectuée dans le contexte de toutes les autres et repérée par un objet-preuve proche du temps présent (voir plus loin la représentation du temps). Les contextes seront désignés par des lettres grecques majuscules (Γ, Δ, Λ...). Ce qui est donné dans un contexte Γ, ce sont des jugements hypothétiques, qui ont la forme :

$$J(x_1, ..., x_n) \ \ (\Gamma)$$

Si on a un élément (un objet preuve) qui prouve une proposition dans un certain contexte, cela s'écrit :

$$a(x_1, ..., x_n) \ : \ A(x_1, ..., x_n) \ \ (\Gamma)$$

En "oubliant" la preuve, on écrit :

$$A(x_1, ..., x_n)\ vraie\ \ (\Gamma)$$

Une distinction importante que fait Ranta est celle qui s'établit entre ce qui, dans un contexte, est donné de manière *actuelle* et ce qui est donné de manière *potentielle*. Cette distinction avait déjà été faite par Martin-Löf :

> *Actual judgements are those that have been made already. Judgements that can be made by applying iteratively the rules of type theory, but have not been made, are only potential, and there is no reason why all of them should ever be actually made by anyone.*

On pense évidemment à tous les jugements qui peuvent provenir de l'application des règles d'introduction des quantificateurs, lesquelles ont leur traduction dans le langage. Par exemple,

A man walks in the park. He whistles

a la représentation :

$$(\Sigma z\ :\ (\Sigma x\ :\ man)(x\ walks))(p(z)\ whistles)$$

Le fait que ce soit une proposition a été prouvé à partir d'une hypothèse $x\ :\ man$ et d'une hypothèse $z\ :\ (\Sigma x\ :\ man)(x\ walks)$, autrement dit grâce au contexte :

$$(x\ :\ man)(z\ :\ (\Sigma x\ :\ man)(x\ walks))$$

Une fois que la phrase est formée, c'est un jugement actuel. Mais évidemment à partir des mêmes hypothèses, on aurait pu faire beaucoup d'autres dérivations, qui, tant qu'elles ne sont pas faites, restent des jugements potentiels.

Les jugements peuvent de plus être comparés par rapport à leur "actualité / potentialité". Chaque argument auquel s'applique un foncteur est *plus actuel* que le jugement obtenu au cours de cette application. Par exemple, si on a :

$$c(x_1, ..., x_n)\ :\ (\Sigma x\ :\ A(x_1, ..., x_n))B(x_1, ..., x_n, x)$$

alors on a aussi le jugement :

$$p(c(x_1, ..., x_n))\ :\ A(x_1, ..., x_n)$$

(où p est toujours la première projection), mais ce dernier est moins actuel que le précédent.

Les objets donnés dans un contexte forment un univers de discours sur lequel la comparaison d'actualité met une relation d'ordre partiel. On retrouvera ici une notion d'arbre.

Noter que cela renvoie à l'idée suivante : les *sélecteurs* comme p, q, $apply$ donnent toujours des jugements moins actuels que celui qu'on analyse au moyen d'eux. C'est ce que l'on retrouvera aussi en ludique : l'analyse suppose un temps postérieur à l'énonciation proprement dite.

3.3 Pronoms anaphoriques

Les pronoms anaphoriques sont l'exemple typique d'expressions dépendant du contexte. Par exemple, l'usage de *il* n'est possible en Français que dans le contexte où un humain masculin est donné. Etant donné un tel humain, a, il peut être utilisé pour a. En considérant $masc$ comme type, on a les règles :

$$\frac{a \ : \ masc}{il(a) \ : \ masc} \qquad\qquad \frac{a \ : \ masc}{il(a) = a \ : \ masc}$$

La première règle autorise l'introduction d'une nouvelle constante, dépendant d'un terme de type $masc$, la seconde règle précise que il est, de fait, la fonction identité. Evidemment aucune de ces règles ne dit comment aller rechercher le terme dont le pronom dépend, ni comment le passage à la surface linguistique permet d'abandonner la mention de la dépendance de sorte que seul il subsiste... En fait, la détermination du terme viendra du contexte au sens précis donné ci-dessus. Par exemple, lorsqu'on formalise la phrase :

(1) *quand un homme marche, il siffle*

on formalise d'abord l'expression *un homme marche*, ce qui donne :

$$(\Sigma x \ : \ masc)\ marche(x)$$

ensuite on considère la phrase *il siffle* dans le contexte fourni par une preuve de la phrase précédente, autrement dit comme dépendante de :

$$z \ : \ (\Sigma x \ : \ masc)\ marche(x)$$

La phrase (1) est en effet une fonction qui, à toute preuve z du fait que un homme marche associe une preuve du fait que ce même homme siffle. On doit alors extraire de z ce qui peut apparaître comme étant précisément cet homme, dont il est question. Puisque $(\Sigma x \ : \ masc)\ marche(x)$ est l'ensemble des couples formés d'un x de type $masc$ et d'une preuve de $marche(x)$, on obtient l'homme en question par projection de z sur sa première composante, autrement dit comme $p(z)$. Ainsi le contexte fournit-il précisément un individu de type $masc$. Par définition du fonctionnement anaphorique, il est alors l'expression en surface de $il(p(z))$. Cela permet d'obtenir, en utilisant la règle d'introduction de Π, la formule :

$$(\Pi z \ : \ (\Sigma x \ : \ masc)\ marche(x))(siffle(il(p(z)))$$

autrement dit, puisque il exprime l'identité :

$$(\Pi z \ : \ (\Sigma x \ : \ masc)\ marche(x))(siffle(p(z))$$

Cette manière de faire va s'avérer particulièrement féconde dans le traitement des fameuses phrases de Geach (ou *donkey sentences*).

3.4 Présupposition

Le cas général de la présupposition est celui d'une proposition dont la formation dépend de l'existence d'une preuve pour une autre proposition, autrement dit :

$$B(x) \ : \ prop \ (x \ : \ A)$$

qui devient, si on supprime la preuve :

$$B \ : \ prop \ (A \ vraie)$$

Par exemple, on a :

Pierre a arrêté de fumer : prop ((Pierre fumait) vraie)

3.5 Donkey sentences

L'un des résultats les plus spectaculaires de la Théorie Constructive des Types est la manière dont elle résout le problème des *donkey sentences*. On sait qu'il s'agit des phrases du genre :

si Pierre possède un âne, il le bat
tout fermier qui possède un âne le bat

Comme dans le cas ci dessus de *Un homme marche dans la rue. Il siffle*, on part de l'affirmation que Pierre a un âne, ce qui se traduit par :

$$z \ : \ (\Sigma x \ : \textit{âne})\ \textit{Pierre possède}\ x$$

Or nous savons qu'une preuve de $(\Sigma x \ : \ A)B$ est constituée d'un couple (u, v) où u est une méthode pour trouver x dans A et v une preuve de B (dépendant éventuellement de x). Donc si on applique le projecteur p à z, on obtient l'L'ne dont il est question, et le projecteur q donne la deuxième composante, autrement dit la preuve que *Pierre possède x. Il le bat* se traduit donc par *Pierre bat* $p(z)$. On obtient donc :

$$z \ : \ (\Sigma x \ : \textit{âne})\ (\textit{Pierre possède}\ x)(\text{Pierre bat}\ p(z))$$

et par application de Π, qui va donner une proposition (liage de toutes les variables) :

$$(\Pi z \ : \ (\Sigma x \ : \textit{âne})\ (\textit{Pierre possède}\ x)(\textit{Pierre bat}\ p(z)))$$

Dans le cas de *si un fermier possède un âne, il le bat*, on obtient :

$$(\Pi z \ : \ (\Sigma y \ : \text{fermier})(\Sigma x \ : \textit{âne})\ (y\ \textit{possède}\ x)(p(z)\ \text{bat}\ p(q(z))))$$

3.6 Théorie des types de bas niveau et Théorie de haut niveau

Dans ce qui précéde, nous avons travaillé en fait avec une théorie dont le seul type était Ens, le type des ensembles, identifié ici avec le type $Prop$ des propositions. Une théorie de plus haut niveau considèrerait que Ens n'est qu'un type parmi d'autres, et qu'il nous faut partir d'un type beaucoup plus général qui serait le type... $Type$! En ce cas, les règles seraient des règles très générales valables au niveau de tous les types, et les règles particulières relatives aux ensembles comme types particuliers seraient des jugements au sein de cette théorie. C'est ce que Ranta traite au chapitre 8 de son livre. Cette théorie dite " de haut niveau" admet, comme la théorie de bas niveau, quatre sortes de jugements :

$Jugement$		$signifie$:
$\alpha \ : \ type$	_	$\alpha \ est \ un \ type$
$\alpha = \beta \ : \ type$	$\alpha \ : \ type, \beta \ : \ type$	$\alpha \ et \ \beta \ sont \ des \ types \ égaux$
$a \ : \ \alpha$	$\alpha \ : \ type$	$aest \ un \ objet \ de \ \alpha$
$a = b \ : \ \alpha$	$\alpha \ : \ type, a \ : \ \alpha, b \ : \ \alpha$	$a \ et \ b \ sont \ des \ objets \ égaux \ de \ \alpha$

et trois schémas de règles, qui permettent de dire ce qu'on déclare comme type et comment on effectue un produit (dépendant) :

$$Ens \ : \ Type \qquad \frac{A \ : \ Ens}{A \ : \ Type} \qquad \frac{\begin{array}{c}(x \ : \ \alpha) \\ \vdots \\ \alpha \ : \ type \ \ \beta \ : \ type\end{array}}{(x \ : \ \alpha)\beta : \ type}$$

Dans ce cadre, une règle de formation de la théorie de bas niveau, comme :

$$\frac{\begin{array}{c}(x \ : \ A) \\ \vdots \\ A \ : \ Ens \ \ B(x) \ : \ Ens\end{array}}{(\Sigma x \ : \ A)B(x) \ : \ Ens}$$

s'écrit comme un jugement :

$$\Sigma \ : \ (X \ : \ Ens)((X)Ens)Ens$$

Etant donnés $A \ : \ Ens$ et $B \ : \ (A)Ens$, on peut leur appliquer Σ pour former le quantifieur : $\Sigma(A) \ : \ ((A)Ens)Ens$, puis l'ensemble $\Sigma(A, B)$.

4 Computational Semantics in Type Theory

4.1 Syntaxe abstraite et syntaxes concrètes

Dans son article- semi tutoriel de 2003, A. Ranta présente une implémentation d'un formalisme intégrant grammaire (au sens classique) et Théorie Constructive des Types. Ce formalisme implémenté est dénommé GF (*Grammatical Formalism*). L'idée de fond est l'opposition entre syntaxe

abstraite et syntaxe concrète, la première décrivant des structures abstraites (par exemple des arbres, ou des termes) et la seconde n'étant que l'image par un morphisme de ces structures abstraites exprimée dans un langage de chaînes. Une autre composante est bien sđr la forme logique, qui peut elle aussi être traitée comme une sorte de "syntaxe concrète", le langage de traduction étant alors un langage de λ-termes et de formules logiques. Là où une grammaire de clauses à la Prolog s'exprime au moyen de *clauses*, qui sont des déclarations s'exprimant dans un langage proche de la logique des prédicats (cf. "la liste L est une phrase (S) si la liste L_1 est un syntagme nominal (SN), la liste L_2 un syntagme verbal (SV) et L est la concaténation de L_1 et L_2"), une grammaire dans GF est une liste de *jugements*. Ces jugements permettent de définir des fonctions, par exemple :

fun $Pred \ : \ NP \rightarrow VP \rightarrow S$

se lit : $Pred$ est une fonction (**fun**) qui, à un objet de catégorie NP associe une fonction qui à un objet de catégorie VP associe un objet de catégorie S. Un tel jugement, appartenant à la syntaxe abstraite, est accompagné d'un autre jugement, dit de linéarisation, servant à exprimer la projection sur la syntaxe concrète :

lin $Pred\ np\ vp = \{s = np.s \ ++ \ vp.s\}$

qui se lit : la linéarisation de l'objet obtenu en appliquant la fonction $Pred$ à deux objets successivement, l'un noté np et l'autre vp s'obtient en concaténant la linéarisation de np et celle de vp. La valeur de cette linéarisation n'est à vrai dire pas une simple chaîne mais (ce qu'indiquent les accolades) une structure d'enregistrement ("record"). s est un attribut à valeur supposée de type chaîne.
Chaque règle usuelle de grammaire (dans un format du genre "grammaire hors contexte") est ainsi associée à une fonction ($Pred, Compl, Indef, Raise, Every$), jusqu'aux atomes, c'est-à-dire les mots du lexique, qui sont des fonctions sans arguments (d'arité 0), autrement dit des constantes. Une phrase comme :

every woman loves Bill

a pour représentation syntaxique abstraite :

$$Pred(Every(woman))((Compl(Love))(Raise(Bill)))$$

L'article donne la manière de convertir cette représentation autant sous forme de chaîne (de manière à retrouver *every woman loves Bill*) que sous forme de formule de logique des prédicats. Pour cela, la logique des prédicats est elle-même redéfinie dans le cadre de GF.
Elle repose sur deux catégories déclarées :

cat $Prop$
cat Ent

et deux types de fonctions :

fun $And, Or, If \ : \ Prop \rightarrow Prop \rightarrow Prop$
fun $All, Exist \ : \ (Ent \rightarrow Prop) \rightarrow Prop$

Il est intéressant ici de noter comment sont définis les quantificateurs. $(Ent \rightarrow Prop)$ est le type d'objets qui sont des fonctions des entités dans les propositions, autrement dit ce sont des *propriétés*. Donc un quantificateur est défini comme une fonction qui à toute propriété P associe une proposition. On a alors aussi bien $All(P)$ que $Exist(P)$, l'une pour traduire que P est vraie de toutes les entités et l'autre pour dire qu'il existe au moins une entité vérifiant P.
La traduction en logique des prédicats (dans cette syntaxe particulière) se fait alors au moyen d'une fonction d'interprétation associée à chaque catégorie. Cette fonction est définie dans les objets de la catégorie en question et à valeurs dans les dénotations possibles de cette catégorie. Par exemple, un SV a pour dénotation une fonction qui, à une entité individuelle associe une proposition, un nom propre a pour dénotation une entité individuelle etc. D'où :

fun $iS \ : \ S \rightarrow Prop$
fun $iNP \ : \ NP \rightarrow (Ent \rightarrow Prop) \rightarrow Prop$
fun $iVP \ : \ VP \rightarrow Ent \rightarrow Prop$
fun $iTV \ : \ TV \rightarrow Ent \rightarrow Ent \rightarrow Prop$
fun $iCN \ : \ CN \rightarrow Ent \rightarrow Prop$
fun $iPN \ : \ PN \rightarrow Ent$

avec pour chacune une définition explicite :

def $iS(Pred\ Q\ F) = iNP(Q)(\lambda x.iVP(F)(x))$
def $iVP(Compl\ F\ Q)(x) = iNP(Q)(\lambda y.iTV(F)(x)(y))$
def $iNP(Every\ A)(F) = All(\lambda x.If(iCN(A)(x))(F(x)))$
def $iNP(Indef\ A)(F) = Exist(\lambda x.And(iCN(A)(x))(F(x)))$
def $iNP(Raise\ a)(F) = F(iPN(a))$

On peut alors facilement calculer que :

$$iS(Pred(Every(woman))(Compl(Love)(Raise(Bill)))) =$$
$$All(\lambda x.If(iCN(woman)(x)(iTV(Love)(x)(iPN(Bill)))))$$

Des règles de linéarisation adéquates transforment ensuite cette "formule" en une formule plus classique de logique des prédicats du premier ordre :

$$(\forall x)(woman(x) \Rightarrow love(x, Bill))$$

Nous ne traiterons pas ici l'aspect sémantique en termes de valeurs booléennes (sémantique véri-conditionnelle) pour nous intéresser davantage à la sémantique en termes de preuves.

4.2 De l'usage des types dépendants

On a vu plus haut que, dans certains cas, un jugement de théorie des types dépend d'un contexte (une suite d'autres jugements représentés par les variables de preuve associées). Par exemple, le type *élément* ne saurait exister en lui-même, il dépend toujours d'un l'ensemble. On a en ce cas une déclaration

cat $El\ (Ens)$

ceci exprime que le type El dépend d'un objet de type Ens.
La règle de formation associée à Σ, que nous avons plus haut, peut s'interpréter comme une extension de la règle de formation d'une conjonction au cas de types dépendants. Elle peut elle-même être représentée en Théorie des Types par une fonction :

fun $\Sigma\ :\ (A\ :\ Ens) \rightarrow (El\ A \rightarrow Ens) \rightarrow Ens$

autrement dit, à un ensemble A, elle associe une fonction qui, à toute fonction qui à un élément de A associe un ensemble, associe un ensemble. Essayons de comprendre cette définition. $\Sigma(A)(B)$ représente donc un ensemble (un ensemble de preuves) associé à un ensemble A et à une famille d'ensembles B indexée par A. Noter que $Prop = Ens$ à partir du moment où on considère une proposition comme un ensemble (l'ensemble de ses preuves). Une famille d'ensembles indexée par un ensemble A est donc exactement la même chose qu'une famille de propositions dépendant d'un paramètre à valeurs dans A, autrement dit : un prédicat défini sur le domaine A. Donc on peut dire que défini ainsi, $\Sigma(A)(B)$ est bien une proposition définie à partir d'un ensemble et d'un prédicat.
On peut linéariser les règles d'introduction, par exemple pour introduire Σ, on écrit qu'un élément d'un ensemble formé avec Σ consiste en un couple d'un élément du domaine (A) et d'une preuve du prédicat appliqué à cet élément :

fun $pair\ :\ (A\ :\ Ens) \rightarrow (B\ :\ El\ A \rightarrow Ens) \rightarrow (a\ :\ El\ A) \rightarrow El\ (B(a)) \rightarrow El(\Sigma(A)(B))$

cela se lit : on obtient un élément de l'ensemble $\Sigma(A)(B)$ ($El(\Sigma(A)(B))$ au moyen d'un ensemble A, d'une fonction B qui à tout élément de A associe un ensemble (une proposition), d'un élément a de A et d'une preuve de $B(a)$.
La règle d'élimination de Σ introduit les fonctions de projection qui permettent de séparer d'un côté un élément de A et de l'autre la preuve de $B(a)$:

fun $p\ :\ (A\ :\ Ens) \rightarrow (B\ :\ El(A) \rightarrow Ens) \rightarrow El(\Sigma(A)(B)) \rightarrow El(A)$
fun $q\ :\ (A\ :\ Ens) \rightarrow (B\ :\ El(A) \rightarrow Ens) \rightarrow (c\ :\ El(\Sigma(A)(B)) \rightarrow El(B(p(A)(B)(c))))$

Noter bien ce qui se passe dans cette dernière définition : q est une fonction qui prend en argument un ensemble (le domaine A), un prédicat (la fonction qui à tout élément de A associe une proposi-

tion, et un élément c particulier de l'ensemble $\Sigma(A)(B)$. Elle donne comme résultat un élément de l'ensemble obtenu en appliquant la fonction B à un élément de A obtenu à partir de la projection p s'appliquant d'abord à A, puis à B puis à ce même c.
Les fonctions p et q sont ensuite définies de telle sorte que :
$p(pair(A)(B)(a)(b)) = a$ et
$q(pair(A)(B)(a)(b)) = b$.
On ne s'attardera pas ici sur les règles de linéarisation (qui permettent d'écrire une formule obtenue : $\Sigma(A)(B)$ sous la forme plus habituelle $\Sigma(x\ :\ A)B(x)$).
On procède de la même manière pour Π que pour Σ.

fun $\Pi\ :\ (A\ :\ Ens) \rightarrow (El(A) \rightarrow Ens) \rightarrow Ens$
fun $\lambda\ :\ (A\ :\ Ens) \rightarrow (B\ :\ El(A) \rightarrow Ens) \rightarrow ((x\ :\ El(A) \rightarrow El(B(x)) \rightarrow El(\Pi(A)(B))$
fun $app\ :\ (A\ :\ Ens) \rightarrow (B\ :\ El(A) \rightarrow Ens) \rightarrow El(\Pi(A)(B)) \rightarrow (a\ :\ Al(A)) \rightarrow Al(B(a))$
def $app(A)(B)(\lambda(A)(B)(b))(a) = b(a)$

Ranta donne l'exemple suivant. Considérons la phrase qui, en mathématiques, dit que pour tout nombre réel x, il existe un cercle dont le rayon est égal à x. Elle s'écrit :

$$(1)\ (\Pi x\ :\ R)(\Sigma y\ :\ Cercle)(rayon(y) = x)$$

Il ne faut pas la confondre avec :

$$(2)\ (\Pi x\ :\ R)(\Sigma y\ :\ Cercle)(rayon(x) = y)$$

ceci simplement parce que bien évidemment, x n'est pas un cercle, mais un nombre ! Autrement dit, les opérateurs Π et Σ permettent de spécifier le type des arguments d'une fonction. En linguistique, le cas se présentera souvent. Notons d'abord que les NP et VP peuvent se relativiser par rapport à des ensembles donnés. Ainsi, on aura :

cat $NP(Ens)$
cat $VP(Ens)$
cat $TV(Ens)(Ens)$

cela permet de dire par exemple que tel verbe (par exemple *manger*) est dépendant par rapport à certains types d'argument (par exemple l'ensemble des êtres animés et l'ensemble des choses comestibles). Il en résulte évidemment une définition de fonction d'interprétation différente de celle que nous avons donnée plus haut. Par exemple désormais un nom commun (CN) s'interprète comme une fonction qui associe à la catégorie CN un objet de type Ens, cet ensemble étant celui dont dépend le nom commun considéré. Un VP s'interprète comme une fonction qui, à un ensemble A associe une fonction qui au VP défini en référence à cet A associe une fonction qui à tout élément de A associe un ensemble (= une proposition). Par exemple un $VP(+anime)$ est interprété comme associant à tout animé (et seulement aux animés) une proposition obtenue en

appliquant ce VP au sujet animé. On a les fonctions suivantes définies relativement aux règles de la grammaire :

fun $Pred\ :\ (A\ :\ Ens) \rightarrow NP(A) \rightarrow VP(A) \rightarrow S$
fun $Compl\ :\ (A, B\ :\ Ens) \rightarrow TV(A)(B) \rightarrow NP(B) \rightarrow VP(A)$
fun $Every\ :\ (A\ :\ CN) \rightarrow NP(iCN(A))$
fun $Indef\ :\ (A\ :\ CN) \rightarrow NP(iCN(A))$
fun $Raise\ :\ (A\ :\ Ens) \rightarrow PN(A) \rightarrow NP(A)$

avec la fonction d'interprétation ainsi définie :

def $iNP(X)(Raise(A)(a))(B) = B(iPN(A)(a))$
def $iNP(X)(Every(A))(B) = \Pi(iCN(A))(B)$
def $iNP(X)(Indef(A))(B) = \Sigma(iCN(A))(B)$
def $iS(Pred(A)(Q)(F)) = iNP(A)(Q)(\lambda x.iVP(A)(F)(x))$
def $iVP(X)(Compl(A)(B)(F)(Q))(a) = iNP(B)(Q)(\lambda y.iTV(A)(B)(F)(a)(y))$

Considérons par exemple la phrase *Pierre dort*. Elle utilise :

fun $Pierre\ :\ PN(+anim)$
fun $dort\ :\ VP(+anim)$

et sans doute aussi une règle de grammaire supplémentaire servant à insérer des verbes intransitifs.

fun $Intrans\ :\ (A\ :\ Ens) \rightarrow IV(A) \rightarrow VP(A)$

Autrement dit : $Intrans$ est une fonction qui prend en argument un ensemble A et qui transforme le verbe intransitif $IV(A)$ en le syntagme verbal $VP(A)$. A est ici un ensemble qui spécifie le type d'argument auquel le verbe intransitif peut s'appliquer. L'interprétation de cette règle est :

def $iVP\_\ Intrans(A)(F)(a) = iIV(A)(F(a))$

Les fonctions d'interprétation des terminaux sont supposées être la fonction identique, quel que soit le domaine A dont ils dépendent, ce qui fait que la règle ci-dessus peut s'écrire plus simplement :

def $iVP\_\ Intrans(A)(F)(a) = F(a)$

La phrase *Pierre dort* est ainsi la linéarisation d'une structure :
$Pred(+anim)(Raise(+anim)(Pierre))(Intrans(+anim)(dort))$.
Voici comment on calcule son interprétation :

$iS(Pred(+anim)(Raise(+anim)(Pierre))(Intrans(+anim)(dort))) =$
$iNP(+anim)(Raise(+anim)(Pierre))(\lambda x.iVP(+anim)(Intrans(+anim)(dort))(x)) =$
$\lambda x.iVP(+anim)(Intrans(+anim)(dort))(x)(iPN(+anim)(Pierre)) =$
$\lambda x.dort(x)(iPN(+anim)(Pierre)) =$
$\lambda x.dort(x)(Pierre) =$
$dort(Pierre)$

Afin d'interpréter des phrases comme les *donkey sentences*, il faut introduire les structures en *If* et *And*, de la manière suivante :

fun $If\ :\ (A\ :\ S) \rightarrow (El(iS(A)) \rightarrow S) \rightarrow S$
fun $And\ :\ (A\ :\ S) \rightarrow (El(iS(A)) \rightarrow S) \rightarrow S$

If apparaît alors comme associant à une phrase A et à un prédicat B défini sur le domaine d'interprétation de A, une phrase.

def $iS(If(A)(B)) = \Pi(iS(A))(\lambda x.iS(B(x)))$
def $iS(And(A)(B)) = \Sigma(iS(A))(\lambda x.iS(B(x)))$

La phrase obtenue par la fonction If s'interprète comme l'application de $\Pi(iS(A))$ (quantification sur le domaine associé à la phrase A, autrement dit l'ensemble des preuves de A) à la fonction $\lambda x.iS(B(x))$, autrement dit comme un terme dont la linéarisation donnera $(\Pi z\ :\ A)B(z)$ à savoir "pour toute preuve z de A, il existe une preuve de $B(z)$".
Une phrase comme :

if a farmer owns a donkey, he beats it

est donc la linéarisation de la structure :

$If(Pred(Indef(farmer))(Compl(owns)(Indef(donkey))))(Pred(Pron(farmer))(Compl(beat)(Pron(donkey))))$

où toutefois, la fonction $Pron$ n'a pas encore été explicitée. Nous voyons pour cela d'ailleurs que les types dépendants sont nécessaires. En effet les pronoms dépendent de leur antécédent dans leur forme (genre, nombre...). Nous définissons la fonction $Pron$ de la manière suivante :

fun $Pron\ :\ (A\ :\ CN) \rightarrow El(iCN(A)) \rightarrow PN(iCN(A))$
def $iPN\_\ Pron(\_)(a) = a$

Le sens de ceci est le suivant :
La fonction$Pron$ prend comme argument un nom commun A (représenté par un ensemble) et un élément a de cet ensemble et elle retourne un élément de A qui est justement a (fonction identité). Ainsi $Pron(Donkey)$ associe à tout élément a qui est un L'ne cet élément lui-même (et n'est pas défini en un x qui n'est pas un L'ne).

Il faut donc inclure les paramètres dans la structure abstraite. Considérons la première proposition du If :

A farmer owns a donkey

Elle provient de l'application de la fonction $Pred$. Auparavant, cette fonction prenait comme argument un terme de catégorie NP et un terme de catégorie VP pour donner un terme de catégorie S, et son interprétation ($iS(Pred\ Q\ F)$) était obtenue en appliquant l'interprétation du NP (iNP) fourni par Q à la fonction obtenue par abstraction sur l'argument x de l'interprétation du VP fourni par F. Maintenant, il y a un argument supplémentaire A, qui est le paramètre commun au NP et au VP. L'interprétation du NP fourni par Q dépend de cet A, tout comme celle du VP fourni par F. Dans *A farmer owns a donkey*, le NP est *A farmer* et le VP est *owns a donkey*. On peut admettre que le domaine A est donc *farmer*. L'interprétation de *a farmer* maintenant est obtenue à partir de la fonction $Indef$. Auparavant, cette fonction associait un NP à un CN (nom commun), maintenant elle associe au domaine d'un nom commun (par exemple *farmer*) un NP dépendant de cet ensemble ($NP(iCN(A))$).
On obtient donc :
$Pred(iCN(farmer))(Indef(farmer))(Compl...)$
Même démarche pour $Compl$, qui nous permet d'obtenir la structure abstraite :
$Pred(iCN(farmer))(Indef(farmer))(Compl(iCN(farmer))(iCN(donkey))(own(Indef(donkey))))$
Pour traiter maintenant la phrase *if a farmer owns a donkey, he beats it*, nous devons partir d'un arbre abstrait qui contient toutes les informations nécessaires. Quand on considère $If(A)(B)$, d'après le typage de la fonction If, A est une phrase, par exemple celle que nous venons d'obtenir et qui est représentée par l'arbre abstrait ci-dessus, mais B est une *fonction* qui, à tout élément de l'ensemble associé à cette phrase (donc à toute preuve de cette phrase) associe une phrase. La phrase finale est obtenue au moyen de ces deux ingrédients.
Pronominaliser *A farmer* à partir de la phrase précédente se ramène à appliquer la fonction $Pron$, dépendant du type *farmer* (ce qui nous assurera que l'interprétation du pronom sera bien un fermier !) à un individu particulier.
Cet individu particulier est introduit dans la phrase qui sert d'antécédent (*A farmer owns a donkey*), plus précisément, étant donnée une preuve z de cette phrase, l'individu en question est obtenu par projection p de cette preuve (puisqu'on sait qu'en ce cas, ce qu'on obtient c'est bien un individu, la deuxième projection, q, nous donnant une preuve que la deuxième partie de la phrase est vraie de cet individu). Cette projection maintenant est dépendante des types comme les autres fonctions. Elle dépend de deux types : celui de la première composante de la preuve et celui de la deuxième composante, qui s'écrivent respectivement :

- $farmer$
- $\lambda x'.\Sigma(donkey)(\lambda y.own(x', y)$

Si z est une preuve de la phrase antécédent, le pronom s'écrit donc :
$Pron(farmer)(p(farmer)(\lambda x'.\Sigma(donkey)(\lambda y.own(x', y)))(z))$
ce qu'en abrégé évidemment, nous pouvons nous contenter d'écrire :
$Pron(farmer)(p(z))$!

En tant que conséquent dans la phrase donnée, *he beats it* doit pouvoir s'appliquer à la preuve fournie par z. On notera alors que cette phrase est elle-même obtenue par la fonction $Pred$, mais appliquée cette fois à un pronom (qui sera susceptible de $Raise$, option par défaut des NP), puis au produit d'une application de $Compl$. Etudions tout de suite $Compl$. Il y a évidemment un deuxième pronom (*it*) dans cette partie, traité comme le précédent. Ce pronom s'écrit en abrégé :
$Pron(donkey)(p(q(x)))$ (on passe les détails !)
Ce qui donne finalement, compte tenu de la règle concernant If, que la phrase A s'applique à l'abstraction sur sa preuve de la phrase B, ce qui donne comme structure abstraite :

$If(Pred(farmer)(Indef(farmer))(Compl(farmer)(donkey)(own(Indef(donkey)))))$
$(\lambda z.Pred(farmer)(Raise(farmer)(Pron(farmer)(p(z))))$
$(Compl(farmer)(donkey)(beat(Raise(donkey)(Pron(donkey)(p(q(z)))))))$

ou, de façon encore plus abrégée :

$If(Pred(Indef(farmer))(Compl(own(Indef(donkey)))))$
$(\lambda z.Pred(Raise(Pron(farmer)(p(z))))(Compl(beat(Raise(Pron(donkey)(p(q(z))))))))$

Cet arbre abstrait peut ensuite être interprété au moyen des fonction iX, et on obtient bien alors :

$$(\Pi z\ :\ (\Sigma x\ :\ farmer)(\Sigma y\ :\ donkey)own(x,y))\ beat(p(z),p(q(z)))$$

4.3 Problèmes

La démarche que nous avons développée ci-dessus convient parce que nous n'avions qu'une seule variable à lier dans $Pred$ et une seule dans $Compl$, et ces deux variables étaient hiérarchisées ($Compl$ est contenu dans $Pred$), mais nous n'aurions pas pu obtenir l'autre lecture d'une phrase comme *Every man loves a woman*, celle où l'indéfini a la portée large. Pour résoudre cette question, Montague utilisait des règles dites *Quantifying In*. Mais l'inconvénient de ces règles d'un point de vue computationnel est qu'elles rendent plus difficile la phase de *parsing*.

à continuer...

5 Types avec Records

Un certain nombre d'auteurs (Cooper, Ginzburg...) ont introduit la notion de *record* dans les types, afin que ceux-ci soient *structurés*. Un Type-Record est, de fait, le type d'objets qui sont structurés à la manière des matrices attribut-valeur (ou des structures de traits dans les grammaires d'unification telles LFG ou HPSG).
Un *record* est un ensemble de champs qui assignent des entités à des labels qui sont partiellement

ordonnés au moyen de l'idée de *dépendance*. Par exemple :

$$\left[\begin{array}{c} l_1 = v_1 \\ l_2 = v_2 \\ \ldots \\ l_n = v_n \end{array}\right]$$

Un tel objet est d'un certain *type*, à savoir les objets valides, c'est-à-dire ceux pour lesquels les champs sont remplis par des valeurs d'un type donné. Par exemple, si v_1 doit être de type T_1 etc. le record précédent sera du type suivant :

$$\left[\begin{array}{c} l_1 : T_1 \\ l_2 : T_2 \\ \ldots \\ l_n : T_n \end{array}\right]$$

Supposons que nous voulions par exemple représenter la signification de *Pierre a vu Paul*, en tant que jugement, cet énoncé dépend de deux individus préalablement donnés : *Pierre* et *Paul*. C'est dire qu'une preuve p de la proposition dépend de ces deux données. On écrit cela :

$$p : a\_vu(pierre, paul) \;\; (pierre : Ind), (paul : Ind)$$

Ce faisant, on traite $a\_vu(pierre, paul)$ et Ind comme des types. Les contextes (données de pierre et de paul) ne sont pas à proprement parler des objets en soi de la théorie des types, mais on peut les réifier au moyen de records. Le type correspondant à l'énoncé *Pierre a vu Paul* sera donc :

$$\left[\begin{array}{lcl} x & : & Ind \\ y & : & Ind \\ prf & : & a\_vu(x, y) \end{array}\right]$$

il correspond à une réponse à la question "qu'est-ce qu'une preuve d'un acte de voir ?", et un objet de ce type est par exemple :

$$\left[\begin{array}{lcl} x & = & pierre \\ y & = & paul \\ prf & = & p \end{array}\right]$$

où $pierre : Ind$, $paul : Ind$ et $p : a\_vu(pierre, paul)$.
R. Cooper a montré que la Théorie des Types était un excellent moyen pour reformuler la théorie dite "des Situations et des Attitudes", due à Barwise et Perry. On peut y exprimer par exemple la notion de *contrainte* telle qu'elle apparaît entre des situations différentes. Par exemple, un dicton tel que "pas de fumée sans feu" se traduit sous la forme d'une contrainte qui lie toutes les situations où il y a de la fumée. Si l'univers des *records* contient un enregistrement r :

$$r : \left[\begin{array}{lcl} l & : & Loc \\ s & : & fumée(l) \end{array}\right]$$

qui contient deux champs, l'un pour dire que l est un lieu, l'autre pour dire que s est du type d'une preuve qu'il y a de la fumée en ce lieu, alors il contient aussi :

$$r' : \left[\begin{array}{lcl} l & : & Loc \\ s & : & feu(l) \end{array} \right]$$

encore qu'évidemment, il n'y ait pas ici de lien entre les deux l... d'où l'usage de la notation "en chemin" à l'intérieur d'un *record*. Si l'on veut exprimer que le lieu l de r' est précisément celui mentionné dans le premier *record*, alors on introduira une équation :

$$l = r.l$$

ou bien directement $r.l$ à la place de l dans le deuxième :

$$r' : [\ s \ : \ feu(r.l)\ ]$$

rendant ainsi r' dépendant de r. Cette contrainte peut alors être *internalisée* sous la forme d'une *fonction* qui, à tout record r du premier type, associe un record r' du second, le type de cette fonction étant :

$$(r : \left[\begin{array}{lcl} l & : & Loc \\ s & : & fumée(l) \end{array} \right]) \quad [\ s \ : \ feu(r.l)\ ]$$

Cela exprime le fait que si un *record* précis existe, du type de l'argument, comme :

$$r = \left[\begin{array}{lcl} l & = & sur\ la\ colline \\ s & = & fumée\ sur\ la\ colline \end{array} \right]$$

alors la fonction s'applique et lui associe :

$$[\ s \ = \ feu\ sur\ la\ colline\ ]$$

De même, exprimer que *tous les chats miaulent* se fait au moyen d'une fonction d'un certain type, le type des fonctions qui associent à chaque chat, son miaulement, ce type étant en même temps qualifié de *contrainte*.

$$\left[\begin{array}{l} c = (r : \left[\begin{array}{lcl} x & : & Ind \\ s & : & chat(x) \end{array} \right]) \quad [\ s \ : \ miaule(r.x)\ ] : Type \\ s : contrainte(c) \end{array} \right]$$

Bien noter qu'ici, le type de la fonction est :

$$(r : \left[\begin{array}{lcl} x & : & Ind \\ s & : & chat(x) \end{array} \right]) \quad [\ s \ : \ miaule(r.x)\ ]$$

En introduisant c, on utilise la facilité d'exprimer un type *singleton* (c'est-à-dire ne possédant qu'un seul "habitant"), comme dans :

$$[l = a \ : \ T]$$

(voir ce que nous avons déjà fait pour les pronoms) qui dit que l'objet l de type T est introduit par son identité avec un objet constant a. Ici, donc, un objet preuve c est introduit, c'est un type (autrement dit il est du type *Type* !), et le deuxième s est une preuve que ce type est une contrainte[4]. Au passage, cet exemple introduit le type *Type*... qui est lui-même un type. Noter qu'on pourrait écrire :

$$Type \ : \ Type$$

au même titre que l'on peut parler d'un ensemble de tous les ensembles qui, donc, se contiendrait lui-même, ce qui s'écrirait :

$$\mathcal{E} \in \mathcal{E}$$

avec tous les désastres que l'on sait !
Cooper, à la suite d'autres (Coquand), propose alors d'utiliser une *stratification*. Cela veut dire que l'on définit une échelle des types $Type^0, Type^1, ..., Type^n, Type^{n+1}, ...$ de sorte que l'on ait, pour tout n :

$$Type^n \ : \ Type^{n+1}$$

ce qui n'est plus contradictoire. La construction des types se fait alors de manière inductive. $Type^0$ est le type des types qui ne sont pas construits en fonction d'autres types, mais si on cherche le type de ce type, alors on voit qu'il est défini en termes de tous les types d'ordre 0, il est donc d'ordre 1, autrement dit :

$$Type^0 \ : \ Type^1$$

et ainsi de suite (comme l'on construit la suite des ordinaux[5]).
Une autre construction possible apparaît également, en termes de fonction. Elle est importante car elle a à voir avec le phénomène de la présupposition : les noms propres ont, comme on sait, un effet présuppositionnel. Dire que "Pauline a cueilli des fleurs" présuppose l'existence d'un individu prénommé "Pauline". La présentation classique en théorie des types, comme dans le livre de Ranta, serait :

$$z \ : \ a\ cueilli\ des\ fleurs(x) \quad (prénommé(x, Pauline)\ vraie)(x \ : \ Ind)$$

En termes de *records*, on peut aussi voir cela comme une fonction qui, à tout contexte :

$$r : \left[\begin{array}{lcl} x & : & Ind \\ c & : & prénommé(x, Pauline) \end{array} \right]$$

associe :

$$\left[\ c' \ : \ a\_cueilli\_des\_fleurs(r.x) \ \right]$$

[4]bien que l'on ne sache pas très bien dire ce que peut bien être la preuve qu'un type est une contrainte !

[5]Et donc comme pour les ordinaux, cette hiérarchie se poursuit jusqu'aux transfinis... ce qui fait évidemment "beaucoup trop" de types !

Cette fonction se déduit en fait de la règle d'introduction de Π vue plus haut. En théorie des types "classique", elle pourrait s'écrire :

$$\lambda x : Ind.\lambda c : prénommé(x, Pauline)\ a\_cueilli\_des\_fleurs(x)$$

de type $(\Pi x : Ind)(\Pi c : prénommé(x, Pauline))\ a\_cueilli\_des\_fleurs(x)$
où l'on note que la manière dont l'individu est prénommé n'apparaît que de manière marginale, en une sorte d'effet de bord de la définition, puisque la preuve c n'est pas explicitement utilisée dans le jugement lui-même (qui fait référence à x, non à c)[6]. En théorie avec *records*, Cooper représente le jugement hypothétique par la fonction :

$$\lambda r : \left[\begin{array}{lcl} x & : & Ind \\ c & : & prénommé(x, Pauline) \end{array} \right] \quad (\left[\begin{array}{lcl} z & : & a\_cueilli\_des\_fleurs(r.x) \end{array} \right])$$

Le type d'une telle fonction s'écrit alors $(R)RecType$ où R est un type record particulier. Ici, nous pourrions l'écrire :

$$(\left[\begin{array}{lcl} x & : & Ind \\ c & : & prénommé(x, Pauline) \end{array} \right])RecType$$

Il ne faut pas confondre les types de fonctions (par exemple le type des fonctions d'un record de type "fumée" vers un record de type "pluie") avec les fonctions d'un type vers un autre. Les types de fonctions ont la syntaxe : $(r : Type1)(r' : Type2)$ alors que les fonctions ont la syntaxe : $\lambda r : Type1(Type2)$.

[6] Peut-être faut-il voir là une manière de traiter la question de la multidimensionnalité des énoncés, au sens de T. Potts.