Un Vernaculaire modulaire pour le Calcul des Constructions

Gilles Dowek

Un Vernaculaire modulaire pour le Calcul des Constructions

Rapport de D.E.A.

Gilles Dowek

1989

Je tiens avant tout à remercier Gérard Huet d'avoir bien voulu être mon directeur de stage.

Il m'a d'abord proposé ce sujet à la fois théorique et débouchant sur une application concrète, mais il m'a aussi permis de m'intégrer à l'équipe du Calcul des Constructions et à participer à ses activités informatiques et logiques.

Je tiens ensuite à remercier Christine Paulin-Mohring et Thierry Coquand pour les nombreuses discussions au cours desquelles j'ai peu à peu compris les bases du Calcul des Constructions et desquelles sont sorties de nombreuses idées de ce mémoire.

Je tiens enfin à remercier l'équipe CAML, d'avoir mis au point ce langage grâce auquel la partie implémentation de ce stage s'est déroulée sans souffrance ni angoisse.

Table des matières

Introduction

Le Calcul des Constructions

Le Calcul des Constructions est un formalisme permettant d'exprimer des informations de nature mathématique (un texte mathématique). Il peut être utilisé dans des applications mathématiques et informatiques.

Dans les applications mathématiques on considère le texte comme une donnée sur laquelle on peut faire un certain nombre d'opérations. La première est de vérifier que ce texte est correct (c'est-à-dire ne comporte pas d'erreurs). En effet, un résultat mathématique est fiable quand il a été non seulement démontré, mais aussi quand sa démonstration à été relue par plusieurs personnes, et qu'elles garantissent ne pas avoir trouvé d'erreur. (Par exemple la rumeur que la conjecture de Fermat était cassée a couru plusieurs semaines avant que l'on se rende compte que cette démonstration était fausse.) Plutôt que de compter sur la loi des grands nombres de relectures, une telle vérification de correction peut être faite automatiquement dans le Calcul des Constructions. Une deuxième utilisation de nature mathématique est l'aide à la démonstration, le mathématicien pouvant laisser à l'ordinateur le soin de démontrer certains lemmes simples, voire dans le futur des théorèmes complexes. On peut aussi imaginer un système de traitement de texte évolué qui présenterait le texte mathématique sous une forme optimale, en tenant compte de sa sémantique. On pourrait enfin comme il est suggéré dans [16] utiliser un tel système pour traduire automatiquement un texte mathématique d'une langue dans une autre. (On sait que ce genre de traduction demande une approche sémantique du texte, et si cette approche est très difficile dans le cas d'un texte quelconque, elle est peut-être plus abordable pour un texte mathématique.)

Dans les applications informatiques on considère le langage mathématique comme un langage de communication entre l'homme et la machine. C'est en effet dans ce langage (ou dans un langage très proche) qu'on a l'habitude d'exprimer naturellement les algorithmes, les spécifications, les requêtes, etc. Bien sûr une application informatique parmi les plus importantes est la vérification de preuves de programmes, car si les démonstrations mathématiques sont en général relues par un nombre suffisant de mathématiciens, ce qui garantit à moyen terme leur correction, les preuves de programmes ne sont en général relues que par leur auteur (qui est souvent aussi l'auteur du programme) et il y a des chances que la preuve comporte les mêmes erreurs que le programme.

Présenter dans un langage uniforme un algorithme et sa preuve permet

d'éviter de séparer l'un de l'autre : par exemple si on cherche à écrire une fonction f qui trie une liste finie, on va en général définir f par récurrence sur la taille de la liste, c'est-à-dire définir une suite f_0 , f_1, etc., f_i s'appliquant aux listes de longueur inférieure à i. Prouver cet algorithme consiste à démontrer qu'il vérifie bien la propriété $\forall x S(x, f(x))$ où $S(x, y)$ signifie que y est une permutation de la liste x et que y est triée. Cette démonstration se fait aussi par récurrence en prenant comme hypothèse de récurrence que f_i vérifie $\forall x(l(x) \leq i) \Rightarrow S(x, f_i(x))$. Au total on a démontré qu'il existe une fonction f qui vérifie la propriété voulue et cette démonstration est constructive, en ce sens qu'elle exhibe l'objet f.

En fait il est plus naturel de mélanger la définition de f et sa preuve de correction dans une récurrence unique. L'hypothèse de récurrence que l'on prend alors est que l'on a exhibé une fonction f_i qui s'applique aux listes de longueur inférieure à i, et que cette fonction f_i vérifie bien la propriété voulue. On n'exhibe plus ici la fonction f, mais on sent intuitivement que l'information nécessaire pour construire une telle fonction est contenue de façon diffuse dans la démonstration.

On peut maintenant remarquer que les propositions :

$\exists f \forall x S(x, f(x))$

et :

$\forall x \exists y S(x, y)$

sont équivalentes (l'équivalence de ces deux formes est l'axiome du choix).

Au lieu de démontrer la première, démontrons la deuxième. Cette démonstration se fait toujours par récurrence, mais l'hypothèse de récurrence est maintenant :

$\forall x(l(x) \leq i) \Rightarrow \exists y S(x, y)$

Ici l'information nécessaire pour construire une fonction f est encore plus diffuse que dans le cas précédent, mais la démonstration reste constructive pourvu qu'à chaque étape de récurrence on construise effectivement la liste triée à partir des listes (plus courtes) fournies par l'hypothèse de récurrence.

Dans cette nouvelle forme x apparaît comme l'entrée de la fonction et y comme la sortie. S apparaît donc comme une spécification du programme f. On a donc vu sur cet exemple une relation très générale, que l'on peut exprimer par le slogan :

Démonstration Constructive de Spécification = Programme + Preuve de Programme

C'est-à-dire que dans la démonstration de la proposition :

$\forall x \exists y S(x, y)$

se cache l'algorithme f et aussi sa preuve de correction, c'est-à-dire une preuve de :

$\forall x S(x, f(x))$

Le Calcul des Constructions permet d'extraire cette information d'une preuve.

Donnons, pour bien sentir la différence, un exemple de démonstration non constructive : soit $A = \{e + \pi, e\pi\}$. On peut démontrer qu'il existe un élément de A qui est transcendant. En effet si $e + \pi$ et $e\pi$ étaient tous les deux algébriques, alors π serait algébrique, or π est transcendant. Cette démonstration est non constructive en ce sens qu'elle ne donne pas l'élément de A qui est transcendant. On ne sait toujours pas après cette démonstration si $e + \pi$ est transcendant ni si $e\pi$ est transcendant. Ce qui dans cette démonstration est responsable de la non-constructivité est l'utilisation de la technique de démonstration par l'absurde. On peut démontrer que si l'on s'interdit ces démonstrations par l'absurde (ou ce qui est équivalent, le tiers exclus) on peut toujours extraire d'une preuve d'existence un témoin de cette existence, et d'une preuve de $\forall x \exists y S(x, y)$ un programme qui répond à la spécification S.

Cette possibilité d'extraire des programmes des démonstrations mathématiques rend ces démonstrations exécutables c'est-à-dire que les démonstrations sont des programmes et le langage mathématique un langage de programmation. Ainsi la preuve ci-dessus est un programme de tri. De même la démonstration de la proposition : pour tous entiers a et b ($b \neq 0$) il existe deux entiers q et r tels que $r < b$ et $a = bq + r$ est un programme de division euclidienne. De plus, si la démonstration est juste (ce qui peut se vérifier automatiquement) ce programme ne peut pas comporter d'erreur puisque la démonstration contient aussi une preuve du programme. Le langage mathématique est donc un langage de programmation où les programmes sont garantis corrects.

Bien entendu cette distinction entre utilisation mathématique et utilisation informatique n'est guère plus pertinente que celle qui oppose données et programmes. On peut très bien utiliser le Calcul des Constructions pour rédiger un théorème à la fois publiable et exécutable, comme le sont beaucoup de théorèmes d'analyse numérique par exemple.

Le Vernaculaire du Calcul des Constructions

Un système comme le Calcul de Constructions ne peut être utilisé par une communauté importante que s'il utilise un langage suffisamment proche du langage dans lequel les mathématiciens ont l'habitude d'écrire leurs résultats.

Une première approche consiste à écrire les démonstrations dans un des dialectes de la déduction naturelle. Mais la déduction naturelle, malgré son nom n'est qu'une approximation très grossière de la façon dont les

mathématiciens écrivent. Le langage utilisé par les mathématiciens (nous appellerons ce langage : vernaculaire mathématique [1]) est en fait assez mal connu. Les langages étudiés par les logiciens (déduction naturelle, calcul des séquents, etc.) sont des stylisations du langage réel. Les logiciens n'ont jamais prétendu que l'on pouvait démontrer des théorèmes non triviaux dans ces langages. Le rôle de ces langages stylisés est autre : c'est de permettre de démontrer des méta-théorèmes (théorèmes sur les théorèmes). C'est un acte de foi universellement admis parmi les mathématiciens que tout ce qui s'écrit dans les livres de mathématiques pourrait si l'on disposait d'assez de temps et de papier être traduit dans un des langages formalisés étudiés par les logiciens. (C'est d'autant plus un acte de foi que d'une part le vernaculaire mathématique est un concept flou et que d'autre part on peut affirmer cette croyance sans connaître aucun de ces langages formels.)

Quand on cherche à écrire l'interface utilisateur d'un système comme le Calcul des Constructions, la croyance générale en ce principe n'est plus d'aucun secours. Ce n'est plus la traductabilité qui est intéressante, mais la traduction effective. Nous devrons donc chercher à définir un langage formel qui soit suffisamment proche de la façon naturelle d'écrire pour un mathématicien afin de lui demander le minimum d'effort d'expression.

Un des problèmes qui se posent quand on veut définir un langage de preuve basé sur la déduction naturelle est celui de la notion de contexte. En effet la déduction naturelle permet d'établir la vérité de jugements du type : "Dans le contexte Γ on peut montrer P", or en mathématiques, le contexte d'une phrase est l'endroit où elle se trouve dans ce texte, c'est-à-dire l'ensemble des phrases (essentiellement des axiomes) qui la précédent. Il est alors difficile d'interpréter une règle comme $\Rightarrow$-intro qui dit que si une proposition Q est démontrable dans le contexte Γ, P alors la proposition $P \Rightarrow Q$ est démontrable dans le contexte Γ. Dans un texte mathématique le contexte ne peut être ainsi augmenté et diminué de façon arbitraire. On sent ici que l'on ne peut pas poser de façon définitive un axiome P, ou en d'autres termes qu'il faut pouvoir définir un axiome local. Le problème de la portée de cet axiome se pose alors, et comme la portée d'un objet ne peut se

1. Selon Girodet une *langue vernaculaire* est *une langue nationale ou dialecte, par opposition au latin, langue officielle de l'Eglise.* Le terme de *Mathematical vernacular* a été utilisé la première fois par N.G. De Bruijn, il le définit comme *the very precise mixture of words and formulas used by mathematicians in their better moments, wheras the "official" mathematical language is taken to be some formal system that uses formulas only.* Ici nous distinguerons *vernaculaire mathématique* et *vernaculaire du Calcul des Constructions*, le premier étant le but que nous nous cherchons à atteindre et le second l'approximation que nous savons implémenter.

définir que dans un langage hiérarchisé, nous sommes donc amenés à définir un mécanisme de structuration du texte mathématique.

1 Le Calcul des Constructions

1.1 Classification des symboles et des phrases

Dans cette section nous définirons les notions de proposition, axiome, théorème, etc. Ces définitions ne prétendent pas présenter un quelconque intérêt d'un point de vue logique ou mathématique, leur unique objet est de présenter ce que ces notions signifient *dans l'implémentation du Calcul des Constructions* et ce qui est correct ou non dans ce système.

1.1.1 Symboles

Traditionnellement on distingue en logique deux sortes de symboles : les symboles permettant de construire des termes et les symboles permettant de construire des propositions.

Algèbre universelle et λ-calcul

Les symboles permettant de construire des termes du premier ordre sont : les symboles d'individu, les symboles de fonction et les variables.

Termes du premier ordre : arbres Les termes sont des arbres dont les nœuds non terminaux sont des symboles de fonction et les feuilles des symboles d'individu et des variables. Par exemple le langage de l'arithmétique comporte les symboles : 0, S, $+$ et $*$. Les arbres 0, $(S\ 0)$, $(*\ (S\ (S\ 0))\ (S\ x))$ sont des termes du premier ordre.

Types Traditionnellement un modèle d'une théorie est un ensemble, et les symboles d'individu et les variables dénotent des éléments de cet ensemble.

Dans le Calcul des Constructions il est possible de définir des symboles d'individus et des variables qui appartiennent à des ensembles distincts. Pour indiquer à quel ensemble ils appartiennent, on associe un type à chaque variable et symbole d'individu. Par exemple le type des entiers naturels est Nat et 0 est de type Nat. Cela se note : $0 : Nat$. Si la variable x dénote un entier on écrit $x : Nat$.

Les symboles de fonction sont en général caractérisés par leur arité par exemple l'arité de S est 1 et celle de $+$ et $*$ est 2. Ici, comme les variables et les symboles d'individus sont typés, nous devons aussi typer les symboles de fonction pour éviter de construire des termes absurdes, par exemple il faut garantir que S s'applique toujours à un sous-terme du type Nat. Nous posons donc $S : Nat \to Nat$, $+ : Nat \to Nat \to Nat$, $* : Nat \to Nat \to Nat$.

Les types des symboles de fonctions sont construits avec les types atomiques et la flèche et ils sont appelés types fonctionnels.

Les termes du premier ordre sont donc des arbres typés.

Types en tant que termes Les types sont des arbres dont les nœuds non terminaux sont toujours la flèche et les feuilles des types atomiques. Ces arbres peuvent être considérés comme des termes. Pour distinguer ces arbres des autres termes, on leur donne le type $Type$. $Type$ est donc le type de tous les types, la flèche est un symbole de fonction de type $Type \to Type \to Type$ et les types atomiques sont des symboles d'individus du type $Type$.

Termes d'ordre supérieur : λ-calcul typé Dans une théorie, nous voulons exprimer non seulement des propositions concernant les éléments du modèle, mais aussi des propositions concernant des sous-ensembles et des fonctions sur cet ensemble. Par exemple pour exprimer qu'un ordre est complet, nous voulons écrire : tout sous-ensemble a un plus grand élément. Dans une théorie d'ordre supérieur les termes désignent non seulement des éléments du modèle, mais aussi des fonctions sur cet ensemble et des prédicats. Les termes d'ordre supérieur sont obtenus en étendant la structure d'arbres typés en un λ-calcul typé.

Dans le Calcul des Constructions on utilise un λ-calcul explicitement typé, dans lequel les variables sont typées dans leur lieur. De plus on note $[x : A]t$ ce qui est noté $\lambda x_A.t$ dans la notation de Church, c'est-à-dire la fonction $x \mapsto t$.

Par exemple en arithmétique le duplicateur est noté : $[x : Nat](* \ (S \ (S \ 0)) \ x)$.

Un terme peut être typé dans un contexte où toutes ses variables libres sont typées, par exemple $(* \ (S \ (S \ 0)) \ (S \ x))$ est typable dans tout contexte où $x : Nat$. Son type est Nat car si x dénote un entier alors $(* \ (S \ (S \ 0)) \ (S \ x))$ dénote aussi un entier. $[x : Nat](* \ (S \ (S \ 0)) \ (S \ x))$ est typable dans le contexte vide, puisqu'il ne comporte pas de variable libre. Son type est $Nat \to Nat$ puisque ce terme dénote une fonction qui à tout entier associe un entier.

Dans le λ-calcul du Calcul des Constructions les termes sont explicitement typés, ce qui permet de définir pour chaque terme un type principal.

Comme les variables de type ne sont pas des variables distinctes des variables du λ-calcul (elles ne sont distinguées que par le fait qu'elles sont du type $Type$), rien n'empêche d'abstraire une variable de type dans un terme. Par exemple le terme $[x : A]x$ est de type $A \to A$ dans tout contexte où A est une variable de type $Type$. Rien n'empêche d'abstraire ce terme par rapport à la variable A. On forme ainsi le terme $[A : Type][x : A]x$. Ce terme dénote une fonction de deux arguments dont le type du deuxième dépend de la valeur du premier. Si on tente de donner un type à ce terme on voudrait lui donner le type $Type \to ....$ Le problème est qu'on ne peut pas connaître ce qui suit la flèche puisque c'est le type du deuxième argument et que ce type dépend de la valeur du premier.

On doit donc étendre le système de type en ajoutant des types (c'est-à-dire des termes au type $Type$) pour les termes comme $[A : Type][x : A]x$.

La règle de formation des types est la suivante : si $\Gamma, x : T$ permet de typer $T' : Type$ alors Γ permet de typer $(x : T)T' : Type$ (lire produit des T' pour x dans T).

Nous pouvons maintenant utiliser ces types pour typer les termes tels que celui ci-dessus, on ajoute la règle : si $\Gamma, x : T$ type $t : T'$ (où x peut apparaître dans t et T') alors Γ type $[x : T]t$ du type $(x : T)T'$.

La flèche apparaît donc comme un cas particulier du produit. $T \to T'$ est $(x : T)T'$ quand x est une variable qui n'apparaît pas dans T'.

$[A : Type][x : A]x$ est donc de type $(A : Type)(A \to A)$.

Règles de typage Nous pouvons à présent écrire les règles de typage du λ-calcul du Calcul des Constructions. Ces règles permettent de construire des termes typées dans un contexte Γ. Un contexte est un ensemble de couples (x, T) où x est un symbole de variable et T un type. Nous noterons $\Gamma \vdash t : T$ le jugement que t peut être typé de type T dans le contexte Γ.

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \textit{Acces variable}$$

$$\frac{\Gamma \vdash T : Type \quad \Gamma, x : T \vdash T' : Type}{\Gamma \vdash (x : T)T' : Type} \qquad \textit{Produit de types}$$

$$\frac{\Gamma \vdash T : Type \quad \Gamma, x : T \vdash T' : Type \quad \Gamma \vdash t : T'}{\Gamma \vdash [x : T]t : (x : T)T'} \qquad \textit{Abstraction}$$

$$\frac{\Gamma \vdash u : (x : T)T' \quad \Gamma \vdash v : T}{\Gamma \vdash (u\ v) : T'[x \leftarrow T]} \qquad \textit{Application}$$

Propositions

Les symboles permettant de construire des propositions sont : les symboles de prédicats, les connecteurs logiques et les quantificateurs.

Symboles de Prédicat La façon la plus simple de former une proposition est d'appliquer un symbole de prédicat à une séquence de termes. Par exemple, si P est un symbole de prédicat à une place de type T et t un terme de type T alors $(P\ t)$ est une proposition. Les propositions ainsi construites sont des arbres à un nœud non terminal qui est le symbole de prédicat et dont les feuilles sont des termes.

Propositions en tant que termes Dans le Calcul des Constructions, ces arbres sont considérés comme des termes. Pour distinguer ces termes des autres on leur donne un type particulier : le type $Prop$. Un symbole de prédicat à n places de types $t_1 ... t_n$ est donc un symbole d'individu de type $t_1 \rightarrow ... \rightarrow t_n \rightarrow Prop$. Il faut bien sûr étendre le λ-calcul du Calcul des Constructions pour prendre en compte ces termes du type $Prop$, nous ne donnons pas ici les règles de ce calcul étendu car nous allons l'étendre encore dans les paragraphes suivants.

Connecteurs Un autre moyen de former des propositions est d'utiliser un connecteur logique comme $\wedge, \vee, \Rightarrow, \Leftrightarrow$ et $\neg$. Prenons l'exemple de l'implication : $\Rightarrow$. Soient P et Q deux propositions, $P \Rightarrow Q$ est aussi une proposition. La flèche d'implication apparaît alors comme un symbole de fonction de type $Prop \rightarrow Prop \rightarrow Prop$. Pour le moment cette flèche $\Rightarrow$ ne doit pas être confondue avec celle des types $(\rightarrow)$, la relation entre les deux sera mise en évidence par la suite.

Quantificateurs Un autre moyen de former des propositions est de quantifier des propositions par rapport à des variables. Prenons l'exemple de la quantification universelle.

Soit P une proposition et x une variable qui peut être libre dans P. Il est possible de quantifier la proposition P par rapport à la variable x. En vernaculaire mathématique on écrit cela $\forall x : T \quad P$. Dans le vernaculaire du

Calcul des Constructions cela s'écrit $(x : T)P$. Si la variable x est libre dans P elle ne l'est plus dans $(x : T)P$.

Pour pouvoir continuer à représenter les propositions comme des termes, il faut étendre les règles de formation des termes de type $Prop$. La règle de formation est la suivante : si $\Gamma, x : T$ permet de typer $P : Prop$ alors Γ permet de typer $(x : T)P : Prop$.

Cette notation ne doit pas être confondue avec celle utilisée pour les produits de types, la relation qui existe entre des deux sera mise en évidence par la suite.

Nous n'avons ici étudié que deux symboles logiques en oubliant les autres connecteurs et quantificateurs. Cela vient du fait qu'il est possible dans un système non prédicatif comme le Calcul des Constructions (c'est-à-dire un système ou les propositions sont des termes et où il est possible de quantifier sur ces termes) de définir ces symboles à l'intérieur même du langage.

Nous nous intéresserons donc dans un premier temps uniquement aux propositions formées avec ces deux symboles logiques, puis nous décrirons dans la section *Symboles logiques* la façon dont les autres symboles sont définis et utilisés dans le Calcul des Constructions.

1.1.2 Phrases

Classification

En vernaculaire on distingue quatre sortes de phrases :

- Les Axiomes qui sont des propositions.
- Les Théorèmes qui sont des paires de propositions et preuves.
- Les Définitions de constantes qui introduisent un nouveau symbole pour un terme. (Par exemple : "$Soit\ 1 = (S\ 0)$").
- Les Déclarations de variables qui introduisent un nouveau symbole qui représente un quelconque élément de son type (Par exemple : "*Soit n un entier*").

Chacune de ces phrases définit un nouveau symbole qui peut être utilisé par la suite.

- Un axiome associe à son nom une proposition.
- Un théorème associe à son nom une proposition et une preuve.
- Une constante définie associe à son nom un terme et un type.
- Une variable déclarée associe à son nom un type.

Les notions de proposition, terme et type ont été formalisées dans les paragraphes précédents. Celle de preuve le sera par la suite.

Symboles du langage

En vernaculaire tout symbole est soit un mot clé soit un symbole de l'une de ces quatre sortes, les symboles permettant de construire les termes et les propositions aussi.

Les variables libres dans les termes sont bien sûr des variables (c'est-à-dire des symboles introduits par une déclaration). Le seul connecteur introduit est la flèche d'implication ($\Rightarrow$) et c'est un mot clé. Le seul quantificateur introduit est le quantificateur universel et c'est un mot clé (en fait comme on n'écrit pas explicitement ce quantificateur, c'est plus où moins un mot clé internalisé dans le structure des termes). La flèche des types ($\rightarrow$) est elle aussi un mot clé, ainsi que les types $Prop$ et $Type$.

Il reste donc à comprendre ce que sont les symboles d'individus, de fonction, de prédicat et les types atomiques.

Par exemple en arithmétique, ces symboles sont 0, S, $+$, $*$, $=$, $<$ et Nat.

Ces symboles ne sont pas des mots clés puisqu'ils appartiennent à une théorie particulière, ce ne sont pas bien entendu des axiomes ni des théorèmes, ce sont donc des constantes ou des variables.

Dans les théories axiomatiques on les considère implicitement comme des variables.

Il peut sembler étrange de considérer 0 comme une variable puisqu'une variable représente un élément quelconque de son type et que 0 n'est pas un entier arbitraire.

La différence entre 0 et un x quelconque est que si nous définissons entièrement l'arithmétique nous serons amenés à poser des axiomes concernant 0 et aucun concernant x.

Donc dans une théorie axiomatique 0, S, $+$, $*$, $=$, $<$ et Nat sont des variables, voici leur type :

$Nat : Type$
$0 : Nat$
$S : Nat \rightarrow Nat$
$+ : Nat \rightarrow Nat \rightarrow Nat$
$* : Nat \rightarrow Nat \rightarrow Nat$
$=: Nat \rightarrow Nat \rightarrow Prop$
$<: Nat \rightarrow Nat \rightarrow Prop$

Une autre façon de définir ces symboles est de les considérer comme des constantes, c'est-à-dire de trouver un modèle de la théorie en question dans le λ-calcul (par exemple les entiers de Church pour l'arithmétique).

En faisant ce choix on perd en généralité puisqu'on ne travaille plus dans un modèle arbitraire de la théorie, mais dans un modèle particulier sauf si

l'on arrive à prouver que tous les modèles sont isomorphes au modèle choisi.

En fait dans bien des cas, comme en arithmétique, cette perte de généralité n'est pas un problème puisqu'on perd les modèles non standards et que seul le modèle standard présente, en général, de l'intérêt.

Contexte

Si on veut utiliser un vocabulaire précis, il est important de distinguer plusieurs notions de contexte : nous avons déjà la notion de contexte au sens du λ-calcul (quand on dit : tel terme est typable dans tel contexte), nous allons introduire par la suite le sens de contexte au sens de la déduction naturelle, nous définissons ici la notion de contexte au sens du Calcul des Constructions.

Le contexte d'une phrase est la suite des phrases écrites avant elle.

Symboles autorisés C'est le contexte qui définit les symboles qui peuvent être utilisés dans une phrase. Par exemple, dans le contexte vide il est possible de définir une variable de proposition P (C'est-à-dire une variable de nom P et de type $Prop$). Puis dans le contexte formé de cette déclaration il est possible de poser un axiome u d'énoncé P, etc.

Poser un axiome d'énoncé P dans le contexte vide est absurde car P est un symbole indéfini.

Typage des termes Dans une phrase du texte, un terme t est typable s'il l'est dans le contexte (au sens du λ-calcul) obtenu en prenant les déclarations de variables du contexte de la phrase (au sens du Calcul des Constructions). Si une variable est déclarée plusieurs fois dans le contexte il ne faut prendre que la dernière déclaration. Il faut aussi considérer que l'on se place dans un λ-δ-calcul (c'est-à-dire un λ-calcul où il est possible de définir des constantes) et prendre comme contexte de constantes (au sens du λ-calcul) les définitions du contexte de la phrase. Comme pour les variables si une constante est définie plusieurs fois il ne faut considérer que la dernière de ces définitions.

1.2 Déduction naturelle et écriture des preuves

Nous allons maintenant détailler les mécanisme de la déduction naturelle et la façon de noter les preuves dans le Calcul des Constructions.

1.2.1 Déduction naturelle

La déduction naturelle est un système de règles qui permet de dériver des jugements $\Gamma \vdash P$ qui signifient intuitivement que P est démontable sous les hypothèses Γ.

Il est d'usage en déduction naturelle de considérer les contextes comme des ensembles de propositions (axiomes) et d'exprimer des conditions sur certaines variables dans la règle $\forall$-intro. Ici nous préférons considérer les contextes comme des ensembles d'axiomes et de déclarations de variables, ce qui permet de s'affranchir de ces conditions. Nous définissons donc un contexte comme un couple dont la première composante est un ensemble de déclarations de variables et la deuxième un ensemble d'axiomes.

Nous supposerons aussi que nous avons un contexte de constantes puisque nous autorisons l'utilisation de constantes dans les termes, nous ne considérons pas pour le moment les théorèmes comme faisant partie de ces contextes.

Comme nous avons mis les déclarations de variables dans les contextes et que nous voulons écrire des règles de déduction naturelle pour un système non prédicatif (c'est-à-dire où l'on peut quantifier sur les variables propositionnelles) nous imposerons que les variables propositionnelles utilisées dans les axiomes d'un contexte soient déclarées dans ce contexte.

Si Γ est un contexte où P est un terme de type $Prop$ nous noterons $\Gamma, (Ax\ u : P)$ le contexte Γ auquel on ajoute un axiome de nom u et d'énoncé P. De même si Γ est un contexte où T est un terme de type $Type$, on note $\Gamma, (Var\ x : T)$ le contexte Γ auquel on ajoute la déclaration de la variable x de type T.

Si Γ est un contexte bien formé où aucun axiome n'a le nom u et aucune variable le nom x et que P et T sont des termes de types respectifs $Prop$ et $Type$ dans Γ alors $\Gamma, (Ax\ u : P)$ et $\Gamma, (Var\ x : T)$ sont des contextes bien formés.

Si $\Gamma, (Ax\ u : P)$ est un contexte bien formé alors Γ aussi.

Si $\Gamma, (Var\ x : T)$ est bien formé alors Γ est bien formé si et seulement si x n'apparaît pas dans les axiomes de Γ

Si Γ est un contexte, sa première composante est un contexte (au sens du λ-calcul) dans lequel on peut typer des termes $t : T$, nous noterons $\Gamma \vdash' t : T$ le jugement affirmant que t est de type T dans la première composante de Γ.

Puisque l'on a restreint l'ensemble des symboles logiques utilisés à l'implication et au quantificateur universel, on peut se restreindre à cinq règles de déduction naturelle.

$$\frac{}{\Gamma \vdash P} \qquad \textit{lien axiome}$$

(S'il existe un axiome d'énoncé P dans Γ).

$$\frac{\Gamma \vdash P \Rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q} \qquad \Rightarrow -elim$$

$$\frac{\Gamma, (Ax\ u : P) \vdash Q}{\Gamma \vdash P \Rightarrow Q} \qquad \Rightarrow -intro$$

$$\frac{\Gamma \vdash (x : T)P \quad \Gamma \vdash' t : T}{\Gamma \vdash P[x \leftarrow t]} \qquad \forall - elim$$

$$\frac{\Gamma, (Var\ x : T) \vdash P}{\Gamma \vdash (x : T)P} \qquad \forall - intro$$

Relation avec le Calcul des Constructions

Dans une phrase du texte une proposition P peut être prouvée s'il existe une dérivation en déduction naturelle du séquent $\Gamma \vdash P$ où Γ est le contexte dont les variables sont les déclarations de variables du contexte (au sens du Calcul des Constructions), et l'ensemble des axiomes les axiomes du contexte (au sens du Calcul des Constructions) de la phrase. Le contexte de constantes est l'ensemble des constantes définies dans le contexte (au sens du Calcul des Constructions) de la phrase. Naturellement si une variable, un axiome ou une constante est définie plusieurs fois on ne garde que la dernière définition.

1.2.2 Preuves dénotées par des λ-termes : sémantique de Heyting

Considérons dans un premier temps les propositions construites avec des variables propositionnelles et la flèche ⇒.

Nous pouvons donc nous limiter à trois règles de déduction naturelle : le lien-axiome, ⇒-elim et ⇒-intro.

Par exemple :

— Dans un contexte Γ où A est une variable propositionnelle, il est possible de prouver : $A \Rightarrow A$.
$\Gamma, (Ax\ u : A) \vdash A$ (lien-axiome)
on peut en déduire $\Gamma \vdash A \Rightarrow A$ (⇒-intro).

— Dans un contexte Γ où A et B sont des variables propositionnelles et où il y a un axiome d'énoncé B, on peut prouver $A \Rightarrow B$.
$\Gamma, (Ax\ u : A) \vdash B$ (lien axiome),
on en déduit $\Gamma \vdash A \Rightarrow B$ ($\Rightarrow$-intro).

— Dans un contexte Γ où A et B sont des variables propositionnelles et où il y a un axiome d'énoncé $(A \Rightarrow A) \Rightarrow B$, on peut prouver B.
$\Gamma, (Ax\ u : A) \vdash A$ (lien axiome)
on peut donc en déduire $\Gamma \vdash A \Rightarrow A$ ($\Rightarrow$-intro)
$\Gamma \vdash (A \Rightarrow A) \Rightarrow B$ (lien axiome)
on déduit des deux propositions précédentes $\Gamma \vdash B$ ($\Rightarrow$-elim)

La sémantique de Heyting propose d'associer à chaque variable propositionnelle un ensemble (l'ensemble de ses justifications ou de ses preuves) et de définir récursivement la preuve de $P \Rightarrow Q$ comme une fonction qui associe une preuve de Q à toute preuve de P.

Ces fonctions sont naturellement dénotées par des λ-termes.

Pour chaque axiome on définit une variable qui est par définition une preuve de cet axiome. (Par convention on choisit pour cette variable le même nom que celui de l'axiome).

Une proposition est dite prouvable dans un contexte Γ si on peut en exhiber une preuve, c'est-à-dire un λ-terme dans lequel les variables libres sont les preuves des axiomes de Γ.

Par exemple $A \Rightarrow A$ a une preuve dans le contexte vide (un terme clos) qui est $[x]x$ puisque $[x]x$ associe à chaque preuve de A une preuve de A.

Soit b une variable qui est par définition une preuve de B. $[x]b$ est une preuve de $A \Rightarrow B$ car $[x]b$ associe à toute preuve de A une preuve de B.

Si f est une preuve de $(A \Rightarrow A) \Rightarrow B$ alors $(f([x]x))$ est une preuve de B car $[x]x$ est une preuve de $A \Rightarrow A$ et f associe à toute preuve de $A \Rightarrow A$ une preuve de B.

Une preuve d'une proposition P écrite en déduction naturelle peut facilement se traduire en un λ-terme.

Montrons ce théorème par une récurrence sur la taille de la preuve écrite en déduction naturelle.

Si la dernière règle utilisée est le lien axiome alors P est un axiome de Γ et la variable p introduite pour P est une preuve de P.

Si la dernière règle utilisée est la règle $\Rightarrow$-elim alors on connaît par hypothèse de récurrence une preuve f de $Q \Rightarrow P$ et une preuve u de Q, $(f\ u)$ est une preuve de P puisque f associe à toute preuve de Q une preuve de P.

Si la dernière règle utilisée est $\Rightarrow$-intro alors P est de la forme $Q \Rightarrow R$. L'hypothèse de récurrence nous donne un λ-terme r preuve de R dans

le contexte $\Gamma, (Ax\ x : Q)$. Les variables libres de r sont soit x, soit dans l'ensemble des variables preuves des axiomes de Γ. Le λ-terme $[x]r$ est une preuve de $Q \Rightarrow R = P$ dans le contexte Γ.

1.2.3 Isomorphisme de Curry-Howard

Considérons à présent les types des preuves d'une proposition P.

Pour cela nous introduisons pour chaque variable propositionnelle A une variable de type que nous notons aussi A. On peut donc associer à chaque proposition un type en unifiant la flèche d'implication ($\Rightarrow$) et la flèche des types ($\rightarrow$).

Quand on déclare un axiome P, on introduit une nouvelle variable p. On peut décider que cette variable est de type P. Il est alors facile de voir que le type d'une preuve d'une proposition quelconque est cette proposition elle-même.

Par exemple la proposition $A \Rightarrow A$ a une preuve qui est $[x : A]x$ et le type de $[x : A]x$ est $A \rightarrow A$.

Dans un contexte Γ où $(A \Rightarrow A) \Rightarrow B$ est un axiome f, la proposition B a une preuve qui est $(f([x : A]x))$ et dans le contexte $f : (A \rightarrow A) \rightarrow B$ le terme $(f([x : A]x))$ est de type B.

Il est facile de montrer qu'on peut interpréter n'importe quel terme typé de type T, comme une preuve de T.

Cet isomorphisme entre types et propositions et preuves et termes est appelé isomorphisme de Curry-Howard.

Dans le Calcul des Constructions on ne distingue pas les fléches $\rightarrow$ et $\Rightarrow$, les deux symboles se notent `->`. Ici pour plus de clarté nous écrirons $\rightarrow$ ou $\Rightarrow$ selon que le terme formé est de type $Type$ ou $Prop$.

1.2.4 Quantification universelle

La sémantique de Heyting suggère de représenter une preuve de $\forall x : T\ \ P$ par une fonction qui associe à tout terme t de T une preuve de $(P[x \leftarrow t])$.

Ces preuves ressemblent à celles de $Q \Rightarrow P$. La principale différence est que quand f est une preuve de $Q \Rightarrow P$ et t une preuve de Q, t apparaît dans la preuve $(f\ t)$ de P, mais pas dans la proposition P elle-même alors que quand f est une preuve de $\forall x : T\ \ P$ le terme t apparaît dans la preuve $(f\ t)$ et aussi dans la proposition $(P[x \leftarrow t])$. Nous utilisons ici le possibilité d'avoir des types dépendants dans le λ-calcul du Calcul des Constructions puisque le type $(f\ t)$ dépend de la valeur de t c'est-à-dire que f est du type $(x : T)(P\ x)$ (Produit pour x dans T des $(P\ x)$).

Nous pouvons maintenant étendre l'algorithme de traduction de preuves écrites en déduction naturelle en λ-termes :

Si la dernière règle utilisée est $\forall$-elim alors nous avons par l'hypothèse de récurrence une preuve u de $(x : T)P$ et t un terme de type T. $(u\ t)$ est une preuve de $P[x \leftarrow t]$ puisque u associe à chaque terme t de type T une preuve de $P[x \leftarrow t]$.

Si la dernière règle utilisée est $\forall$-intro, alors soit x une variable de type T, on a une preuve u (ou x peut apparaître) de P, $[x : T]u$ est une preuve de $(x : T)P$ et x n'est plus libre dans $[x : T]u$.

Comme nous avons noté la quantification universelle comme le produit, l'isomorphisme de Curry-Howard s'étend aux propositions utilisant la quantification universelle. Toute preuve écrite en déduction naturelle peut donc être traduite en un λ-terme et réciproquement tout λ-terme typé peut s'interpréter comme une preuve de son type.

1.2.5 Preuves-termes dans le Calcul des Constructions

Pour noter les preuves des théorèmes nous allons introduire dans le Calcul des Constructions une nouvelle sorte de terme : les preuves.

Si T est un terme de type $Type$ on peut déclarer une variable x de type T, on peut également utiliser une variable liée de type T dans un terme. Par exemple le terme $[x : T]x$ est de type $T \to T$. Le terme $T \to T$ est de type $Type$.

Nous allons autoriser le même mécanisme pour les termes de type $Prop$: si P est de type $Prop$ nous pouvons déclarer une variable de type P, ou utiliser une variable liée de type P. Par exemple $[x : P]x$ est de type $P \Rightarrow P$ et $P \Rightarrow P$ est de type $Prop$.

Il faut faire attention ici à un abus de langage que nous commettons : le terme "typé' a deux sens :

- d'une part un type est un terme t tel qu'il existe un autre terme u (clos ou non) tel que $u : t$, ou ce qui est équivalent, si on peut déclarer une variable de type t, en ce sens $P \Rightarrow P$ est un type,
- d'autre part nous appelons type tout terme du type $Type$. En ce sens $T \to T$ est un type mais pas $P \Rightarrow P$.

Tout type (premier sens) est un type (deuxième sens) ou une proposition.

Genre des termes

Les termes dont le type du type est $Type$ sont appelés des objets (Object). Ceux dont le type du type est $Prop$ sont appelés des preuves (Proof).

Types de $Prop$ et $Type$

Les seuls termes dont nous n'avons pas encore donné le type sont les termes $Prop$ et $Type$ eux-mêmes.

Le terme $Prop$ est par convention de type $Type$.

Si on pose le terme $Type$ de type $Type$ on obtient un système incohérent où il devient possible de construire un paradoxe [4]. On est donc obligé de poser une nouvelle constante $Type_2$ telle que $Type : Type_2$, et une autre $Type_3$ telle que $Type_2 : Type_3$, etc, c'est le mécanisme des univers emboîtés. Néanmoins du point de vue de l'utilisateur $Type$ est un mot qui désigne génériquement les types $Type$, $Type_2$, ... ce qui permet de poser $Type : Type$.

Chaque occurrence du mot clé $Type$ doit être vue comme une occurrence d'un mot $Type_i$ et le système vérifie automatiquement que dans chaque déclaration i peut être instancié conformément aux règles $Type_i : Type_{i+1}$. Si ce n'est pas possible c'est que le texte est un paradoxe du système contradictoire avec $Type : Type$ et le système le signale.

Equivalence Variable-Axiome et Constante-Théorème

Variable-Axiome Comme nous l'avons vu plus haut les variables libres dans les preuves doivent être prises parmi les axiomes et les variables libres dans les objets parmi les variables déclarées. Axiomes et variables se distinguent en fait très peu. La seule différence réside dans le fait que les variables sont des objets (c'est-à-dire le type de leur type est $Type$) et les Axiomes sont des preuves (c'est-à-dire le type de leur type est $Prop$).

Constante-Théorème Les constantes permettent de considérer les termes de genre objet dans un λ-δ-calcul (c'est-à-dire un λ-calcul qui autorise la définition et l'utilisation de constantes) ce qui permet d'abréger leur expression. Il est normal de demander la même chose pour les preuves. Quand on a un terme t de type P on peut décider de l'abréger par un symbole u. Dans ce cas un terme t' utilisant u est δ-équivalent au terme $t'[u \leftarrow t]$ (c'est-à-dire est équivalent modulo le remplacement du nom symbolique des constantes par leur valeur).

Il est à noter que u est un théorème d'énoncé P et de preuve t. Donc l'équivalent de la notion de constante pour les preuves est la notion de théorème. Cela veut dire que l'on peut réutiliser un théorème dans la démonstration d'un autre théorème.

Par exemple soit u est le théorème $(A : Prop)(A \Rightarrow A)$ de preuve $[A : Prop][x : A]x$, on cherche une preuve du théorème $(P \Rightarrow Q) \Rightarrow (P \Rightarrow Q)$, une preuve est $[x : P \Rightarrow Q]x$.

En introduisant une coupure dans cette preuve on obtient $([A : Prop][x : A]x)(P \Rightarrow Q)$. En effet l'introduction d'un radical ("redex") dans le λ-terme correspond à l'introduction d'une coupure dans la preuve en déduction naturelle, puisqu'on montre d'abord $A \Rightarrow A$ dans un contexte où A est une variable propositionnelle (preuve : $[x : A]x$) puis on utilise la règle $\forall$-intro pour montrer $(A : Prop)(A \Rightarrow A)$ (preuve : $[A : Prop][x : A]x$) pour enfin utiliser la règle $\forall$-elim pour montrer $(P \Rightarrow Q) \Rightarrow (P \Rightarrow Q)$ (preuve : $([A : Prop][x : A]x)(P \Rightarrow Q)$) alors qu'en éliminant la coupure de cette preuve on montre directement $(P \Rightarrow Q) \Rightarrow (P \Rightarrow Q)$ en faisant la même preuve que pour $A \Rightarrow A$ mais dans le cas particulier où $A = P \Rightarrow Q$ ce qui donne comme terme de preuve $[x : P \Rightarrow Q]x$ qui est la forme normale de $([A : Prop][x : A]x)(P \Rightarrow Q)$.

Une fois la coupure introduite on peut remarquer qu'un sous-terme de la preuve est le théorème u, ce qui donne la preuve $(u(P \Rightarrow Q))$. On peut donc utiliser la règle $\forall$-elim avec u au lieu de redémontrer u dans le cas particulier où $A = P \Rightarrow Q$. Ce qui est bien l'essence de l'utilisation d'un théorème déjà prouvé.

Règles de typage

Nous pouvons maintenant donner les règles définitives du λ-calcul du Calcul des Constructions qui permettent de rendre compte des termes-propositions et des termes-preuves que nous avons introduits. Tout d'abord nous devons ajouter une règle indiquant que $Prop$ est de type $Type$, puis guidés par l'isomorphisme de Curry-Howard écrire pour le type $Prop$ les mêmes règles que celles écrites plus haut pour le type $Type$. Pour cela nous allons écrire ces règles de façon polymorphe : les symboles K et K' désignent les mots $Type$ ou $Prop$.

$$\frac{}{Prop : Type}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \textit{Acces variable}$$

$$\frac{\Gamma \vdash T : K \quad \Gamma, x : T \vdash T' : K'}{\Gamma \vdash (x : T)T' : K'} \qquad \textit{Produit et Quantification}$$

$$\frac{\Gamma \vdash T : K \quad \Gamma, x : T \vdash T' : K' \quad \Gamma \vdash t : T'}{\Gamma \vdash [x : T]t : (x : T)T'} \qquad \textit{Abstraction}$$

$$\frac{\Gamma \vdash u : (x : T)T' \quad \Gamma \vdash v : T}{\Gamma \vdash (u\ v) : T'[x \leftarrow T]} \qquad \textit{Application}$$

1.2.6 Symboles logiques

Nous avons dit qu'il est possible dans un système non prédicatif comme le Calcul des Constructions (c'est-à-dire un système où les propositions sont des termes et où il est possible de quantifier sur ces termes) de définir les connecteurs et quantificateurs autres que $\Rightarrow$ et $\forall$ à l'intérieur même du langage.

Nous allons décrire ici succinctement la façon dont ces symboles sont définis et utilisés dans le Calcul de Constructions.

Exemple : conjonction

En déduction naturelle le symbole $\wedge$ est primitif et trois règles régissent son comportement.

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \qquad \wedge - intro$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \wedge - elim1$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \qquad \wedge - elim2$$

Dans le Calcul des Constructions ce symbole n'est pas primitif, il est défini à l'intérieur du système. Il existe en effet des termes du λ-calcul $and, conj, proj1$ et $proj2$ tels que :

— si u :A et v :B alors (conj u v) :(and A B),
— si w :(and A B) alors (proj1 A B w) :A,
— si w :(and A B) alors (proj2 A B w) :B.

Ces termes sont :

— $and = [A : Prop][B : Prop](C : Prop)((A \Rightarrow B \Rightarrow C) \Rightarrow C)$
— $proj1 = [A : Prop][B : Prop][x : (and\ A\ B)](x\ A\ [y : A][z : B]y)$
— $proj2 = [A : Prop][B : Prop][x : (and\ A\ B)](x\ A\ [y : A][z : B]z)$
— $conj = [v : A][w : B][C : Prop][f : A \Rightarrow B \Rightarrow C](f\ v\ w)$

On peut de ce fait définir $A \wedge B$ comme une abréviation de $(and\ A\ B)$ et on a les résultats suivants :

— si on peut exhiber une preuve de A et une preuve de B dans Γ alors on peut exhiber dans Γ une preuve de $A \wedge B$

— si on peut exhiber une preuve de $A \wedge B$ dans Γ alors on peut exhiber dans Γ une preuve de A

— si on peut exhiber une preuve de $A \wedge B$ dans Γ alors on peut exhiber dans Γ une preuve de B

Donc on peut traduire toute preuve écrite en déduction naturelle dans le Calcul des Constructions puisqu'on sait y simuler chacune des règles de déduction naturelle.

Réciproquement si on a une preuve u d'une proposition A dans le Calcul des Constructions on peut prouver cette proposition en déduction naturelle.

On montre cette proposition par récurrence sur le couple (u, A), la complexité de u décroissant à chaque étape.

- Si $u = (conj\ b\ c)$ et que $A = B \wedge C$ alors on a par l'hypothèse de récurrence une preuve en déduction naturelle de B et une preuve de C, on peut donc utiliser la règle $\wedge$-intro pour construire une preuve de $B \wedge C$.

- Si ce terme est $(proj1\ A\ B\ u)$ alors u est une preuve de $A \wedge B$. On a par l'hypothèse de récurrence une preuve de $A \wedge B$ et on construit une preuve de A avec la règle $\wedge$-elim1.

- Idem si le terme commence par $(proj2\ A\ B\ u)$.

- Dans tous les autres cas on traduit le terme guidé par l'isomorphisme de Curry-Howard comme ci-dessus.

Notons ici qu'il faut distinguer le cas où on veut montrer $A \wedge B$ et le cas où on veut montrer $(C : Prop)((A \Rightarrow B \Rightarrow C) \Rightarrow C)$ qui sont distincts en déduction naturelle. Dans le premier cas il faut utiliser une règle $\wedge$-intro et dans le second faire deux $\Rightarrow$-elim puis une $\Rightarrow$-intro et un $\forall$-intro. C'est pour cela qu'il faut faire une récurrence sur le couple (terme,type) et non seulement sur le terme.

Autres symboles

La disjonction, l'absurdité, la négation et la quantification existentielle fonctionnent selon le même principe.

Si A et B sont deux propositions $(or\ A\ B)$ est la proposition "A ou B". Cette proposition peut aussi se noter $A \vee B$.

Si u est une preuve de A alors $(or\_introl\ A\ B\ u)$ est une preuve de $A \vee B$.

Si v est une preuve de B alors $(or\_intror\ A\ B\ v)$ est une preuve de $A \vee B$.

Si u est une preuve de $A \vee B$, v une preuve de $(A \Rightarrow K)$ et w une preuve de $(B \Rightarrow K)$ alors $(u\ K\ v\ w)$ est une preuve de K.

La proposition "Absurdé' se note $\{\}$.

Si u est une preuve de $\{\}$ alors $(u\ A)$ est une preuve de A.

Si A est une proposition $(not\ A)$ est la proposition "non A". $(not\ A)$ peut aussi se noter~A. Comme en déduction naturelle $(not\ A)$ est une abréviation pour la proposition $(A \Rightarrow \{\})$.

Si A est une proposition ayant éventuellement une variable libre x de type T alors $(ex\ T\ [x : T]A)$ est la proposition "il existe x de type T tel que A"). Cette proposition peut aussi se noter $< T > Ex\ [x : T]A$.

Si t est de type T et u est une preuve de $A[x \leftarrow t]$ alors $(ex\_intro\ T\ ([x : T]A)\ t\ u)$ est une preuve de $< T > Ex\ [x : T]A$.

Si u est une preuve de $< T > Ex\ [x : T]A$, v une preuve de $(x : T)(A \Rightarrow K)$ et que x n'est pas libre dans K alors $(u\ K\ v)$ est une preuve de K.

1.3 Vernaculaire et Machine Constructive

1.3.1 Vernaculaire

Nous allons maintenant voir la syntaxe permettant de déclarer une variable, définir une constante, poser un axiome et prouver un théorème dans le vernaculaire du Calcul des Constructions.

Variables

```
Parameter <nom de la variable>~: type de la variable>.
```

Constantes

```
Definition <nom de la constante> = <terme>.
```

Axiome

```
Axiom <nom de l'axiome>~: <enonce de l'axiome>.
```

Théorèmes

```
Theorem <nom du theoreme> <enonce du theoreme> Proof <preuve du theoreme>.
```

1.3.2 Machine Constructive

Un tel texte écrit en vernaculaire peut être vérifié automatiquement dans le Calcul des Constructions. Pour cela on le considère comme une suite d'instructions interprétées par une machine : la machine constructive. Cette machine manipule un environnement `ENV` qui est une liste de déclarations de variables, d'axiomes, de constantes et de théorèmes :

```
type constr ... ;; (* type des lambda-termes *)
type declaration =  Vardecl of variable
                  | Constdecl of constant

and variable = Decl of string * judgement
and constant = Def of string * judgement
and judgement = Judge of constr * constr * level
and level = Object | Proof ;;
```

Dans un environnement donné, la machine lit une phrase de vernaculaire. Si cette phrase n'est pas correcte, la machine déclenche une exception, si cette phrase est correcte la machine modifie son environnement pour en tenir compte.

Si le texte de vernaculaire est correct la machine terminera donc son interprétation, et si ce texte est incorrect la machine déclenchera une exception.

Nous supposerons que :

- si ENV et t sont donnés on sait vérifier que t est typable dans ENV et dans ce cas produire son type principal, (Nous noterons $Ty(e,t)$ le fait que t soit typable dans l'environnement e et $PT(e,t)$ son type principal),

- si e est un environnement et que t et t' sont deux termes bien formés et typables dans e on sait vérifier qu'ils sont équivalents (c'est-à-dire $\alpha\beta\eta\delta$-équivalents). Nous noterons ceci Verif(e,t,t'), par exemple Verif(e,PT(e,t),t') signifie que t est le type de t'.

$$\frac{ENV = e \;\; TEXT = (Parameter\ x : t) :: r \;\; Ty(e,t)}{ENV = (Vardecl(Decl(x, Judge(t, Type, Object)))) :: e \;\; TEXT = r}$$

$$\frac{ENV = e \;\; TEXT = (Definition\ x = t) :: r \;\; Ty(e,t)}{ENV = (Constdecl(Decl(x, Judge(t, PT(e,t), Object)))) :: e \;\; TEXT = r}$$

$$\frac{ENV = e \;\; TEXT = (Axiom\ x : t) :: r \;\; Ty(e,t)}{ENV = (Vardecl(Decl(x, Judge(t, Prop, Proof)))) :: e \;\; TEXT = r}$$

$$\frac{ENV = e \;\; TEXT = (Theorem\ x\ t'\ Proof\ t) :: r \;\; Ty(e,t) \;\; Ty(e,t') \;\; Verif(e, PT(e,t), t')}{ENV = (Constdecl(Decl(x, Judge(t, t', Proof)))) :: e \;\; TEXT = r}$$

Remarquons que la règle choisie est toujours décidée par le sommet de la pile TEXT, c'est-à-dire que pour un élément u au sommet de la pile, une règle au plus peut s'appliquer. Quand aucune règle ne s'applique deux cas peuvent se produire : soit la pile TEXT est vide ce qui signifie que le texte à été entièrement vérifié et qu'il est correct, soit une condition $Ty(e,t)$ ou $Verif(e, PT(e,t), t')$ n'est pas vérifiée ce qui signifie qu'un terme ou une preuve est mal construit, ou qu'une preuve est correcte mais n'est pas la preuve du théorème énoncé.

En fait toutes ces règles correspondent à plusieurs transitions successives de la machine. Pour décrire plus finement le fonctionnement de la machine, nous allons traduire les preuves dans un langage de plus bas niveau (command) et interpréter cette traduction.

On ajoute un registre à la machine : le registre VAL et un type d'éléments de l'environnement : les conjectures (Cast).

```
| Cast of judgement
```

Traduction du vernaculaire en commandes :

On traduit phrase par phrase :

Parameter x :t.	[construct t ; def_var x]
Definition x=t.	[construct t ; def_const x]
Axiom x :t	[construct t ; def_ax x]
Theorem x t proof t'	[construct t ; conjecture ; construct t' ;verify ;def_th x]

Règles de transition de la machine :

$$\frac{ENV = e \;\; TEXT = (construct\ t) :: r \;\; Ty(e,t)}{ENV = e \;\; TEXT = r \;\; VAL = t}$$

$$\frac{ENV = e \;\; TEXT = (def\_var\ x) :: r \;\; VAL = t}{ENV = (Vardecl(Decl(x, Judge(t, Type, Object)))) :: e \;\; TEXT = r}$$

$$\frac{ENV = e \;\; TEXT = (def\_const\ x) :: r \;\; VAL = t}{ENV = (Constdecl(Decl(x, Judge(t, PT(e,t), Object)))) :: e \;\; TEXT = r}$$

$$\frac{ENV = e \;\; TEXT = (def\_ax\ x) :: r \;\; VAL = t}{ENV = (Vardecl(Decl(x, Judge(t, Prop, Proof)))) :: e \;\; TEXT = r}$$

$$\frac{ENV = e \;\; TEXT = (conjecture) :: r \;\; VAL = t}{ENV = (Cast(Jugde(t, Prop, Proof))) :: e \;\; TEXT = r}$$

$$\frac{ENV = (Cast(Judge(t', Prop, Proof))) :: e \;\; TEXT = (verify) :: r \;\; VAL = t \;\; Verif(e, PT(e,t),}{ENV = e \;\; TEXT = r \;\; VAL = t}$$

$$\frac{ENV = e \;\; TEXT = (def\_th\ x) :: r \;\; VAL = t}{ENV = (Constdecl(Decl(x, Judge(t, PT(e,t), Proof)))) :: e \;\; TEXT = r}$$

La règle choisie est toujours guidée par le sommet de la pile TEXT.

On vérifiera que les transitions dues à l'interprétation d'un terme t dans le premier système mènent au même état que l'interprétation de sa traduction dans le deuxième système.

Remarquons que l'opération consistant à prendre un terme dans le texte à vérifier qu'il est typable et à le mettre dans le registre VAL que nous considérons comme une opération élémentaire de la machine pourrait être encore décomposée. Cela demanderait de détailler le type *constr* des λ-termes et à ajouter à cet ensemble de règles les règles de typage du λ-calcul utilisé dans le Calcul des Constructions.

2 Un Vernaculaire modulaire pour le Calcul des Constructions

Le vernaculaire sommaire que nous avons décrit ci-dessus demande à l'utilisateur de traduire sa preuve en un λ-terme pour pouvoir l'exprimer. Cette traduction a deux inconvénients : d'une part c'est un travail fastidieux qui devrait être fait par une machine, d'autre part ce travail demande à l'utilisateur de connaître l'isomorphisme de Curry-Howard, que l'utilisateur ne veut sûrement pas apprendre.

2.1 Local/Global

2.1.1 Ecrire une preuve en déduction naturelle

La première possibilité qui permet à l'utilisateur de ne pas traduire entièrement sa preuve sous forme d'un λ-terme est l'utilisation de la δ-conversion (possibilité de démontrer un théorème puis de l'utiliser par la suite).

Certaines preuves peuvent être ainsi exprimées quasiment en déduction naturelle.

La règle $\Rightarrow$-elim

Par exemple on a trois axiomes $u : A \Rightarrow B \Rightarrow C$, $v : A$ et $w : B$. On veut démontrer un théorème z d'énoncé C. La preuve en déduction naturelle est :

$$\dfrac{\dfrac{\Gamma \vdash A \Rightarrow B \Rightarrow C \quad \Gamma \vdash A}{\Gamma \vdash B \Rightarrow C} \qquad \Gamma \vdash B}{\Gamma \vdash C}$$

On peut traduire cette preuve en le λ-terme $(u\ v\ w)$ et donc écrire en vernaculaire :

```
Theorem z C Proof (u v w).
```

Une autre façon de faire est de remarquer que l'on applique deux fois la règle $\Rightarrow$-elim et trois fois le lien axiome. Oublions pour le moment les liens-axiome et décomposons le raisonnement en deux étapes : d'abord prouver le théorème y d'énoncé $B \Rightarrow C$, et ensuite prouver le théorème z.

```
Theorem y (B -> C) Proof (u v).

Theorem z C Proof (y w).
```

Chacune de ces étapes demande l'application d'une seule règle, de ce fait le λ-terme preuve est toujours de la même sorte : pour la règle $\Rightarrow$-elim c'est toujours l'application d'une preuve à une autre. On pourrait donc en changeant très superficiellement la syntaxe écrire ce texte.

```
Theorem y (B -> C) Proof by -> -elim from u and v.

Theorem z C Proof by -> -elim from y and w.
```

Ce qui est une façon d'écrire la preuve en déduction naturelle, qui ne demande pas de connaître l'isomorphisme de Curry-Howard.

La règle ∀-elim

La règle ∀-elim peut s'utiliser selon le même processus : on décide que la phrase `by particularisation of u to t` est une abréviation de $(u\ t)$.

Le lien axiome

En déduction naturelle, si on a deux théorèmes A et $A \Rightarrow B$ on peut utiliser la règle $\Rightarrow$-elim. Si A et $A \Rightarrow B$ sont deux axiomes alors on ne peut pas utiliser directement la règle $\Rightarrow$-elim, il faut d'abord transformer ces axiomes en théorèmes avec la règle du lien axiome.

Dans le vernaculaire du Calcul des Constructions, on autorise d'utiliser les axiomes comme les théorèmes dans l'écriture des termes-preuves. Cette règle du lien axiome n'est plus utile sauf dans le cas trivial où on veut avoir un théorème de même énoncé qu'un axiome. Par exemple si on a un axiome $u : A$ et que l'on veut un théorème d'énoncé A il suffit de dire que u est une preuve de A. On peut donc décider que `from the axiom u` est une abréviation du terme-preuve u.

Les règles d'introduction

Cette façon d'écrire les preuves en déduction naturelle ne peut pas s'étendre aux règle d'introduction, en effet dans la façon de noter les preuves comme ci-dessus on ne parle pas du contexte Γ simplement, on déclare les axiomes du contexte, puis on énonce les théorèmes qui sont tous prouvés dans le même contexte. Ceci n'est plus possible avec les règles d'introduction puisque leur essence même est de raisonner sur différents contextes.

2.1.2 Quelques remarques sur la façon de dénoter la conséquence

Ce problème que l'on rencontre avec les règles d'introduction vient du fait qu'il y a en déduction naturelle trois symboles pour dénoter la conséquence. Un symbole dans le langage : la flèche ($\Rightarrow$) et deux dans le métalangage : le signe de déduction ($\vdash$) et la barre utilisée pour écrire les règles (—). Dans les mathématiques usuelles on en distingue au plus deux : l'implication, et dans le métalangage la barre que l'on traduit par : telle proposition est démontrable à partir de telle autre. En d'autres termes, alors qu'on veut que les règles du métalangage permettent de déduire les propositions les unes des autres, la déduction naturelle nous propose de déduire des séquents les uns des autres, les séquents étant eux-mêmes des déductions.

Par rapport au calcul des séquents, la déduction naturelle minimise ce désagrément. En effet dans la règle $\Rightarrow$-elim :

$$\frac{\Gamma \vdash P \Rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q}$$

On a le même contexte dans tous les séquents, ce qui permet d'écrire cette règle :

$$\frac{P \Rightarrow Q \quad P}{Q}$$

Le problème est que cette remarque n'est pas valable pour toutes les règles et c'est justement les règles à contexte variable qui posent des problèmes pour le vernaculaire.

Il ne semble pas qu'il soit possible de définir un système de déduction dont toutes les règles soient locales et qui n'ait que deux symboles pour dénoter la conséquence. (Il serait sans doute intéressant d'essayer de donner un sens précis à cette phrase, voire de la démontrer.)

Une solution proposée est celle qui consiste à écrire les règles de déduction naturelle en rayant les hypothèses et les variables quand on utilise les deux règles d'introduction. Ce système de déduction n'a plus que deux symboles ($\Rightarrow$ et —) mais les règles ne sont plus locales puisque l'hypothèse ou la variable que l'on raye peut être loin de la barre d'application de la règle.

Cette solution n'est pas satisfaisante car elle ne correspond à rien dans la pratique mathématique mais elle nous indique que si le langage mathématique se contente de deux symboles c'est sans doute parce que les mécanismes sous-jacents ne sont pas non plus locaux.

2.1.3 Contextes ou Contexte

En déduction naturelle un contexte n'est rien de plus qu'un ensemble d'axiomes et de déclarations. Dans un texte mathématique ordinaire le contexte n'est pas un ensemble quelconque d'axiomes et de déclarations mais l'ensemble des axiomes et déclarations situés avant une phrase donnée dans le texte.

La propriété fondamentale qui distingue les textes mathématiques des preuves écrites dans les systèmes de déduction est qu'il existe dans les textes mathématiques un ordre total sur l'ensemble des phrases énoncées alors qu'il n'y a dans les systèmes de déduction qu'un ordre partiel sur cet ensemble.

Cette linéarité (au sens non mathématique du terme) a sans doute à voir avec la non-localité des mécanismes de déduction sous-jacents aux textes de mathématiques.

2.1.4 Pour en revenir aux règles d'introduction

Une façon de comprendre la règle $\Rightarrow$-intro est la suivante : Soit Γ le contexte actuel, si dans un autre livre où le contexte (au sens du Calcul des Constructions) est Γ, P on peut montrer Q, alors on peut dans le contexte actuel déduire $P \Rightarrow Q$.

Pour justifier la déduction de $P \Rightarrow Q$ on peut recopier dans le livre actuel la preuve de Q qu'on aurait écrite dans l'autre livre. Le temps de cette preuve on peut utiliser l'axiome P. Cet axiome P a donc une portée limitée dans le livre.

Cette façon de définir une portée aux hypothèses est un mécanisme non local, mais son effet est limité à une partie du texte, ce contrôle de la non-localité est possible du fait de la linéarité du texte mathématique.

Pour utiliser cette règle il faut donc procéder en trois temps :

1. annoncer que l'on suppose un nouvel axiome P,

2. démontrer Q, c'est-à-dire prouver un nouveau théorème $w : Q$

3. Cesser de supposer l'axiome P tout en transformant le théorème w pour que son énoncé devienne $P \Rightarrow Q$

Au moment où l'on modifie le théorème w de façon à ce que son énoncé devienne $P \Rightarrow Q$ il faut aussi supprimer du contexte l'axiome P.

En effet, quand on a une preuve w de l'énoncé Q qui utilise l'axiome $x : P$, il ne faut pas se contenter de remplacer w par $w' = [x : P]w$ est une preuve de $P \Rightarrow Q$, car cette preuve n'est pas écrite dans le bon contexte. Rappelons que nous voulons considérer que le contexte est l'ensemble des axiomes et variables valides au moment où l'on écrit le théorème (et non

uniquement les variables libres du λ-terme-preuve, que l'utilisateur ne veut pas voir) et donc bien que x ne soit pas libre dans w' la proposition $P \Rightarrow Q$ serait, si l'on ne supprimait pas l'hypothèse x, démontrée dans le contexte voulu augmenté de l'hypothèse P.

2.1.5 Ouvrir les parenthèses sans les fermer

La solution adoptée dans la première version du vernaculaire consistait à noter explicitement les trois opérations : par exemple pour prouver $A \Rightarrow A$ dans un contexte où A est une proposition, il fallait écrire :

```
Axiom x:A.

Theorem w A Proof x.

Discharge x.
```

`Discharge x` indiquait que x devait être éliminée et de ce fait w modifié en $[x : P]x : (P \Rightarrow P)$.

Dans un texte mathématique les deux premières phases sont assez explicites : "Pour prouver $A \Rightarrow B$ supposons A et prouvons B". La troisième est toujours implicite. Il est clair pour le lecteur que quand on a terminé de prouver B, A n'est plus une hypothèse valide, qu'on ne peut pas utiliser le théorème B et en revanche qu'on peut utiliser $A \Rightarrow B$. Le fait qu'un mathématicien ne distingue pas prouver B sous l'hypothèse A et prouver $A \Rightarrow B$ montre qu'on confond même ordinairement la notion mathématique d'implication avec celle métamathématique de déduction, c'est-à-dire les symboles $\Rightarrow$ et —. Ceci vient du fait que l'implication intuitionniste est l'internalisation dans le langage de la notion de déduction.

Puisque la troisième phase du raisonnement est implicite dans un texte mathématique elle doit aussi l'être dans le vernaculaire du Calcul des Constructions. Nous devrons donc ajouter la possibilité de définir une hypothèse ou une variable locale à une preuve, et supprimer les règles $\Rightarrow$-intro et $\forall$-intro.

2.1.6 Un peu de syntaxe

Ecriture des phrases en plusieurs lignes

Pour introduire la syntaxe qui permet de définir des hypothèses et des variables locales nous devons modifier la syntaxe du vernaculaire. Toutes les phrases que nous écrivions en une ligne le sont maintenant en deux ou trois.

```
Parameter <nom de la variable>~: <type de la variable>.
```

s'écrit désormais :

```
Parameter <nom de la variable>.
Inhabits <type de la variable>.

Definition <nom de la constante> = <terme>.
```

s'écrit désormais :

```
Definition <nom de la constante>.
Body <terme>.

Axiom <nom de l'axiome>~: <enonce de l'axiome>.
```

s'écrit désormais :

```
Axiom <nom de l'axiome>.
Assumes <enonce de l'axiome>.

Theorem <nom du theoreme> <enonce du theoreme> Proof <preuve du theoreme>.
```

s'écrit désormais :

```
Theorem <nom du theoreme>.
Statement <enonce du theoreme>.
Proof <preuve du theoreme>.
```

Il est maintenant plus difficile de traduire un texte de vernaculaire en commandes pour la machine constructive. Peut-être faut-il même modifier un peu cette machine. Nous n'allons pas détailler ces modifications puisque nous allons encore enrichir le vernaculaire.

Hypothèses et variables locales

Pour déclarer une hypothèse ou une variable locale à une preuve, il suffit d'insérer la déclaration de cette hypothèse ou de cette variable entre la ligne `Statement` et la ligne `Proof`. Par exemple :

Commençons par déclarer une variable propositionnelle B :

```
Parameter B.
Inhabits Prop.
```

puis prouvons le théorème I :

```
Theorem I.

Statement B->B.

      Hypothesis x.
      Assumes B.

Proof x.
```

Le mot clé `Hypothesis` à remplacé `Axiom`, ce qui indique que x est une hypothèse locale.

Pour prouver $B \Rightarrow B$ nous supposons B et nous prouvons B. Ce qui est puisque que B est une hypothèse.

Remarquons que la preuve donnée pour le théorème est x c'est-à-dire une preuve de B, et que l'énoncé du théorème est $B \Rightarrow B$. Donc l'hypothèse B est locale à la preuve mais pas à l'énoncé du théorème. Il est également possible de déclarer une hypothèse locale à tout le théorème.

```
Theorem I'.

      Hypothesis x.
      Assumes B.

Statement B.

Proof x.
```

Ici l'hypothèse est posée avant l'énoncé du théorème. L'énoncé est donc B (au lieu de $B \Rightarrow B$) et va être modifié en $B \Rightarrow B$ quand l'hypothèse x sera effacée.

Donnons enfin un exemple avec une variable : pour démontrer $(C : Prop)(C \Rightarrow C)$ nous posons une variable propositionnelle C, puis nous supposons l'hypothèse C et enfin nous montrons C.

```
Theorem I''.

Statement (C:Prop)(C->C).

      Variable C.
      Inhabits Prop.

      Hypothesis x.
      Assumes C.

Proof x.
```

Le mot clé `Variable` à remplacé `Parameter` pour indiquer que la variable C est locale.

2.1.7 Lemmes et définitions locales

La possibilité de définir des variables et des hypothèses locales à une preuve nous a permis de nous débarrasser dans le langage superficiel des règles $\forall$-intro et $\Rightarrow$-intro. Mais la hiérarchisation du texte mathématique est un phénomène plus général. Une autre indication que donne la structure hiérarchique du texte est l'importance relative des résultats. Par exemple,

pour démontrer un théorème T on peut avoir besoin de définir un concept qui n'est pertinent que pour cette preuve, de même on peut avoir besoin de montrer un lemme qui n'a pas d'intérêt en soi sinon d'aider à montrer le théorème en question. Ici cela revient à définir une constante (dans le premier cas une constante de genre objet, dans le second de genre preuve) qui vit le temps d'une preuve et dont le nom symbolique est remplacée par la valeur dans la preuve une fois que celle-ci est terminée.

Pour écrire de tels définitions et lemmes locaux il suffit de les insérer entre la ligne `Statement` et `Proof` du théorème, en remplaçant le mot clé `Definition` par `Local` et `Theorem` par `Remark`. On peut aussi insérer une définition locale entre la ligne `Theorem` et la ligne `Statement`, pour utiliser cette définition comme abréviation dans l'énoncé du théorème (la définition est alors locale au théorème en entier : énoncé et preuve). En revanche, insérer un lemme entre la ligne `Theorem` et `Statement` n'a pas grand sens.

2.1.8 Eléments locaux à un axiome, une variable ou une définition

Dans les paragraphes précédents, nous avons vu que l'on pouvait déclarer des éléments (hypothèses, variables, lemmes et définitions) locaux à un théorème. Dans ce paragraphe, nous allons voir qu'il est possible d'en déclarer aussi à des axiomes, des variables et des définitions. Pour cela il suffit d'insérer les phrases de déclaration d'éléments locaux entres les deux lignes de la phrase de définition de l'élément global.

Eléments locaux à une définition

Une *définition* locale à une définition permet d'écrire un corps de la définition plus court, le nom symbolique de la constante locale est remplacé par sa valeur dans la constante globale.

Une *variable* $x : T$ locale à une définition $u = t$ définit en fait la constante u comme $[x : T]t$.

Un *lemme* local à une définition n'a pas grand sens.

Une *hypothèse* locale à une définition permet de définir plus simplement des objets dépendant de preuves comme la racine carrée. En effet, pour former le terme $(sqrt\ x)$ il faut d'abord s'assurer que x est positif, c'est-à-dire que *sqrt* est un objet qui prend en argument un réel et une preuve de la positivité de ce réel.

Eléments locaux à un axiome

Si une *hypothèse* P est locale à un axiome Q c'est l'axiome $P \Rightarrow Q$ qui est en fait posé.

Si une *variable* $x : T$ est locale à un axiome Q c'est l'axiome $(x : T)Q$ qui est en fait posé.

Une définition locale à un axiome peut aider à écrire un axiome plus court, le nom symbolique est ensuite remplacé par sa valeur dans l'axiome.

Un lemme local à un axiome n'a pas grand sens.

Eléments locaux à une déclaration de variable

Si une *variable* $x : T$ est locale à une déclaration de variable $y : T'$ alors la variable y est en fait déclarée de type $T \Rightarrow T'$ (où $(x : T)T'$ si x intervient aussi dans le type T').

Une *définition* locale à une déclaration de variable pourrait aider à écrire un type plus court pour cette variable.

Un lemme local à une déclaration de variable n'a pas grand sens.

Une *hypothèse* locale à une déclaration de variable peut permettre de déclarer plus simplement une variable dénotant un objet dépendant d'une preuve.

2.1.9 Mécanisme général de localité

Dans les paragraphes précédents nous avons vu que l'on pouvait définir un élément d'une sorte quelconque local à un autre élément lui aussi d'une sorte quelconque. Comme il y a quatre sortes d'éléments et deux façons de déclarer un élément local à un théorème (local à la preuve ou à tout le théorème) cela fait vingt possibilités. Comme quatre d'entre elles n'ont pas de sens, cela devrait faire seize cas à traiter.

En fait ces vingt cas obéissent à un même mécanisme que nous allons détailler ici. Bien sûr pour guider l'implémentation nous suivrons ce mécanisme général, mais du point de vue de l'utilisateur, c'est l'étude de toutes les combinaisons ci-dessus qui est importante et non le mécanisme général.

Rappelons que les théorèmes sont des constantes, que les preuves sont les valeurs de ces constantes et les énoncés des théorèmes les types de ces constantes. Seul le genre distingue les théorèmes des autres constantes. De même les axiomes sont des variables, les énoncés de ces axiomes sont les types de ces variables.

Nous n'avons donc que deux sortes d'éléments : les constantes qui ont une valeur et un type et les variables qui n'ont qu'un type.

Nous pouvons donc ramener nos vingt cas à deux :

Quand une *constante* est locale à un élément, on remplace, quand la constante est éliminée, le nom symbolique par la valeur de la constante dans le type et la valeur (s'il en a une) de l'élément en question.

Quand une *variable* $x : T$ est locale à un élément, elle est abstraite dans cet élément, c'est-à-dire que si le type de l'élément est T' il devient $(x : T)T'$ quand la variable est éliminée et si l'élément a une valeur v, celle-ci devient $[x : T]v$.

Un cas particulier est celui où une variable est abstraite uniquement dans la preuve d'un théorème sans être abstraite dans son énoncé. Dans ce cas la preuve n'est pas une preuve de l'énoncé avant l'abstraction et il faut d'abord abstraire la variable dans la preuve (sans toucher à l'énoncé) avant de vérifier que la preuve est bien une preuve de l'énoncé.

2.1.10 Equivalence avec le calcul des prédicats intuitionniste

Maintenant que nous avons la possibilité de déclarer des variables et des axiomes locaux à une preuve, nous pouvons supprimer les règles $\forall$-intro et $\Rightarrow$-intro de l'ensemble des règles de déduction naturelle utilisées pour construire les termes-preuve. Il est important de vérifier qu'en modifiant ainsi le vernaculaire, en l'enrichissant d'une part du mécanisme de localité et en l'amputant d'autre part de deux règles, on ne change pas son pouvoir d'expression, c'est-à-dire que nous devons vérifier que les propositions démontrables en vernaculaire et en déduction naturelle (par exemple) sont les mêmes.

Correction

Pour montrer que toute preuve écrite en vernaculaire peut se traduire en déduction naturelle, on remarque que l'interprétation d'une preuve en vernaculaire d'une proposition P revient à construire un λ-terme-preuve de P. Nous détaillerons par la suite la méthode de construction. On peut comme nous l'avons vu déduire de ce terme une preuve en déduction naturelle de P.

Il est à noter que si on veut démontrer formellement ce théorème il faut complètement décrire la sémantique du vernaculaire (sémantique que nous allons esquisser par la suite). Et démontrer que le λ-terme construit à partir d'une preuve de P est de type P. Il y a beaucoup de travail à faire pour décrire cette sémantique et prouver sa correction, mais l'enjeu de ce travail n'est pas négligeable : en extrayant un programme d'une preuve constructive

de la correction de la sémantique du vernaculaire, on réaliserait l'amorçage ("boot-strap") du Calcul des Constructions.

Complétude

La réciproque de ce théorème est plus simple. Si on a une preuve d'une proposition P écrite en déduction naturelle, nous allons écrire un lemme pour chaque étape de la démonstration. Il suffit donc de faire une récurrence sur la longueur de la preuve. Notons tout de même qu'après avoir introduit une variable ou une hypothèse, nous devons montrer des lemmes avant d'arriver à la règle d'introduction qui abstrait ces hypothèses et ces variables. La possibilité de définir des lemmes locaux à un théorème et des hypothèses et variables locales à ce lemme, etc, n'est pas superflue, cette possibilité du vernaculaire est nécessaire pour traduire simplement les preuves de déduction naturelle en vernaculaire. Ce qui est important n'est pas que ces lemmes locaux soient éliminés hors du théorème global et que leur nom symbolique soit remplacé par leur valeur, mais la possibilité après une introduction de variable ou d'hypothèse de prouver des résultats intermédiaires utilisant eux-aussi des règles d'introduction avant de prouver le théorème final. En fait, il est possible de montrer la complétude sans utiliser cette possibilité et en autorisant de donner un arbre (c'est-à-dire plus qu'une simple application) comme preuve d'un théorème, mais il faudrait introduire de nombreuses coupures peu naturelles dans la preuve en vernaculaire.

Montrons formellement ce théorème :

Soit π une preuve en déduction naturelle du séquent $\Gamma \vdash P$. Nous allons construire un texte de vernaculaire qui est correct dans un contexte (au sens du Calcul des Constructions) où les axiomes et les variables de Γ sont déclarés et qui définit un théorème u d'énoncé P.

— Si la dernière règle de π est le lien-axiome, alors il existe dans Γ un axiome u d'énoncé P. Le texte de vernaculaire :

```
Remark u.
Statement P.
Proof u.
```

a les propriétés requises.

— Si la dernière règle de π est $\Rightarrow$-elim, alors il existe une proposition Q et deux preuves en déduction naturelle π_1 et π_2 de $\Gamma \vdash Q \Rightarrow P$ et $\Gamma \vdash Q$. Par hypothèse de récurrence, il existe deux textes $T1$ et

$T2$ qui définissent deux symboles $u1$ et $u2$ qui sont des preuves de $Q \Rightarrow P$ et Q. Le texte suivant :

```
T1

T2

Remark u.
Statement P.
Proof (u1 u2).
```

a les propriétés requises.

— Si la dernière règle de π est $\forall$-elim, alors il existe une proposition Q, une preuve π_1 et un terme t tels que π_1 est une preuve de $\forall x : T \;\; Q$ et $Q[x \leftarrow t] = P$. Par hypothèse de récurrence, il existe un texte $T1$ qui définit un symbole $u1$ preuve de $\forall x : T \;\; Q$. Le texte :

```
T1

Remark u.
Statement P.
Proof (u1 t).
```

a les propriétés requises.

— Si la dernière règle de π est un $\Rightarrow$-intro, alors il existe deux propositions Q et R, et une preuve π_1 telles que $P = Q \Rightarrow R$ et π_1 est une preuve de $\Gamma, (Ax\ h : Q) \vdash R$. Par hypothèse de récurrence, il existe un texte $T1$ qui définit un symbole $u1$ preuve de R dans un contexte où les variables et les axiomes de Γ et h sont déclarées. Le texte :

```
Remark u.
Statement P.
      Hypothesis h.
      Assumes Q.
      T1.
Proof u1.
```

a les propriétés requises.

— Si la dernière règle de π est un $\forall$-intro, alors il existe une proposition Q, une variable x, un type T et une preuve π_1 telles que $P = \forall x :$

$T\ \ Q$ et π_1 est une preuve de $\Gamma, (Var\ x : T) \vdash Q$. Par hypothèse de récurrence, il existe un texte $T1$ qui définit un symbole $u1$ preuve de Q dans un contexte où les variables et les axiomes de Γ et x sont déclarées. Le texte :

```
Remark u.
Statement P.
      Variable x.
      Inhabits T.
      T1.
Proof u1.
```

a les propriétés requises.

2.2 Sections

2.2.1 Partage d'hypothèse

Dans un livre, il est d'usage de démontrer dans un chapitre unique des résultats qui partagent une même hypothèse. Par exemple dans un livre de théorie des groupes on veut pouvoir consacrer un chapitre aux groupes commutatifs. Tous les théorèmes que l'on montre sont de la forme :

```
Statement ((Commutatif G) -> P).
```

(G est une variable globale)

Pour prouver un tel théorème, nous devons comme nous l'avons vu supposer localement que G est commutatif et prouver P.

Il est d'usage de ne pas répéter cette hypothèse pour chaque théorème, mais d'écrire au début du chapitre : "Dans ce chapitre on suppose G commutatif", puis d'écrire les théorèmes :

```
Statement P.
```

Si on a besoin de cette hypothèse dans une preuve on fait référence au fait qu'elle est supposée dans tout le chapitre.

A l'intérieur du chapitre le théorème est considéré comme étant P, et après le chapitre il est considéré comme $((Commutatif\ \ G) \Rightarrow P)$.

L'hypothèse ($Commutatif\ \ G$) est donc une hypothèse locale, mais au lieu d'être locale à un élément elle est locale à toute une section.

Pour permettre de définir des hypothèses, des variables, des lemmes et des définitions locaux à une section, nous allons ajouter au langage des instructions délimitant les sections.

Le début de la section se note :

`Section <nom de la section>.`

La fin :

`End <nom de la section>.`

Par exemple :

```
Section Groupes_commutatifs.
...
End Groupes_commutatifs.
```

Dans une telle section toutes les phrases commençant par les mots : `Hypothesis`, `Variable`, `Remark` et `Local` définissent des éléments locaux à la section. Hors de la section, ils sont effacés et les autres éléments sont modifiés de façon à ce que tout se passe comme si ces éléments étaient locaux à tous les éléments de la section.

2.2.2 Interprétation d'un mécanisme dans un autre

Les deux mécanismes que nous avons présentés ci-dessus (la possibilité de définir des éléments locaux à un élément et celle de définir des éléments locaux à une section) sont redondants, il est en effet facile de simuler l'un d'eux dans un système qui autorise l'autre.

Naturellement dans le vernaculaire nous autorisons les deux mécanismes, mais la machine constructive ne doit en connaître qu'un.

Interpréter le mécanisme de section dans celui des éléments locaux à un élément

De la façon dont nous l'avons introduite, la possibilité de définir une section est un procédé pour définir simultanément des éléments locaux à plusieurs éléments. On peut donc réécrire le texte avec ces éléments locaux à chaque élément qui le suit :

Par exemple :

```
Section Groupes_commutatifs.

Hypothesis c.
Assumes (Commutatif G).

Theorem th1.
Statement <statement 1>.
Proof <proof 1>.

Theorem th2.
Statement <statement 2>.
Proof <proof 2>.

End Groupes_commutatifs.
```

peut se réécrire en :

```
Theorem th1.
      Hypothesis c.
      Assumes (Commutatif G).
Statement  <statement 1>.
Proof  <proof 1>.

Theorem th2.
      Hypothesis c.
      Assumes (Commutatif G).
Statement <statement 2>.
Proof <proof 2>[th1 <- (th1 c)].
```

La substitution de $(th1\ c)$ à $th1$ vient du fait que dans $th2$ on a utilisé $th1$ comme preuve de $< statement\ 1 >$. Alors que maintenant $th1$ est une preuve de $(Commutatif\ \ G) \Rightarrow < statement\ 1 >$ et donc $(th1\ c)$ est une preuve de $< statement\ 1 >$.

Interpréter le mécanisme d'éléments locaux à un élément dans celui de section

Réciproquement il est toujours possible d'utiliser le mécanisme de définition d'éléments locaux à un élément en considérant chaque phrase de vernaculaire comme une petite section.

```
Theorem th1.

      Hypothesis c.
      Assumes <hyp>.

Statement <statement>.
Proof <Proof>.
```

Comme :

```
Section th1.

Hypothesis c.
Assumes <hyp>.

Theorem th1.
Statement <statement>.
Proof <Proof>.

End th1.
```

La première traduction comporte l'inconvénient d'obliger une réécriture assez complexe sur la preuve avant sa vérification, alors que cette réécriture est plus simple dans la deuxième traduction (nous verrons même qu'en modifiant un peu la machine constructive on peut s'affranchir de cette réécriture).

L'inconvénient de la deuxième traduction est de compliquer sensiblement la machine constructive en ajoutant un mécanisme de fermeture de section qui abstrait les variables éliminées et remplace les noms symboliques des constantes éliminées par leur valeur dans les bons éléments.

Grossièrement la première demande du travail avant la vérification de la preuve et la deuxième pendant cette vérification. Nous avons choisi la deuxième.

2.2.3 Force des éléments

Jusqu'ici nous nous sommes contentés de déclarer des éléments locaux à des éléments globaux. Nous voulons aussi pouvoir déclarer des éléments locaux à d'autres éléments locaux, par exemple une hypothèse locale à un lemme :

```
Remark r1.

     Hypothesis c.
     Assumes <hyp>.

Statement <statement>.

Proof <Proof>.
```

se traduit en :

```
Section r1.

      Hypothesis c.
      Assumes <hyp>.

Remark r1.
Statement <statement>.
Proof <Proof>.

End r1.
```

Le problème avec cette traduction est que le lemme $r1$ est désormais local à la section $r1$, c'est-à-dire que rien ne va sortir de la section $r1$. Si on réécrit ce texte en `Theorem r1`, le théorème va être global alors que l'on veut qu'il soit local, mais local à la section qui est juste au-dessus de la section $r1$.

L'information élémentaire : local/global est donc insuffisante et nous allons la remplacer par un entier qui indique la section exacte à laquelle est limitée la portée de l'élément.

A tout point du texte (c'est-à-dire à chaque instant de la vérification du texte) on considère un nombre SEC qui est le nombre de sections ouvertes. Quand on définit un élément local dans cette section on donne à cet élément la force SEC. Si l'élément est global on lui donne la force 0. Quand on ferme une section, la valeur de SEC passe de s à $s-1$ et on supprime tous les éléments qui ont une force supérieure ou égale à s. Ainsi quand on vérifie :

```
Remark r1.

      Hypothesis c.
      Assumes <hyp>.

Statement <statement>.
Proof <Proof>.
```

dans une situation où SEC $= s$, on associe à $r1$ la force s et à c la force $s+1$, ainsi quand on ferme la section $s+1$ (c'est-à-dire quand on a fini de vérifier $r1$) l'hypothèse c est abstraite et $r1$ survit, jusqu'à la prochaine fermeture de section, ce qui est bien ce qu'on voulait.

2.3 Vernaculaire et machine constructive

2.3.1 Sémantique de Section et End

Pour formaliser la sémantique de *Section* et *End* nous allons ajouter un nouveau type d'élément dans l'environnement : les Têtes de section (Section)

```
| Section of string;;
```

Quand nous rencontrerons une tête de section de nom s nous empilerons l'élément (Section s) au sommet de la pile.

Quand nous rencontrerons une fin de section nous modifierons les éléments situés dans la pile ENV au dessus de la dernière tête de section de façon à supprimer les éléments de force supérieure ou égale à s. Quand une constante ou un lemme est supprimé on remplace son nom symbolique par sa valeur dans tous les éléments définis après lui et qui l'utilisent. Quand une variable ou une hypothèse est supprimée on fonctionnalise tous les éléments définis après elle et qui l'utilise. (C'est-à-dire que si on supprime $u : T$

et que $v : P$ est un élément défini après u et qui l'utilise alors v devient $[u : T]v : (u : T)P)$ et quand on fonctionnalise un élément v en u on modifie toutes ses occurrences dans les objets au-dessus de lui en remplaçant v par $(v\ u)$.

Par exemple si w était égal à $(v\ x)$, w devient $(v\ u\ x)$ car v a été fonctionnalisé et en fait devient ensuite $[u : T](v\ u\ x)$ car u est éliminé.

Nous ajoutons deux instructions à la machine constructive et traduisons le vernaculaire ainsi :

Section x.	[open_section x]
End x.	[close_section]

2.3.2 Modifier la machine constructive pour ne pas faire explicitement la réécriture de preuve

L'expression de la sémantique du vernaculaire en terme de réécriture de preuves est une bonne façon d'exprimer une spécification mais pas un bon moyen d'implémenter le vernaculaire.

Nous allons donc tenter de donner une sémantique à chacune des instructions : `Parameter`, `Variable`, `Inhabits`, `Definition`, `Local`, `Body`, `Axiom`, `Hypothesis`, `Assumes`, `Theorem`, `Remark`, `Statement`, `Proof` de façon à ce qu'une phrase écrite en plusieurs lignes :

```
Theorem t.
Statement <statement>.
Proof <Proof>.
```

ait la sémantique de `Theorem t  <statement> Proof <Proof>.` définie plus haut et que :

```
Theorem t.

      Hypothesis c.
      Assumes <hyp>.

Statement <statement>.
Proof <Proof>.
```

ait la même sémantique que :

```
Section t.

Hypothesis c.
Assumes <hyp>.

Theorem t.
Statement <statement>.
Proof <Proof>.

End t.
```

(avec le calcul des forces décrit plus haut).

Nous allons commencer par donner la sémantique de `Parameter` et `Inhabits`. Nous voulons que :

```
Parameter x.
Inhabits t.
```

ait la même sémantique que celle définie précédemment pour `Paramter x:t.` et que l'on puisse introduire des éléments locaux entre les deux lignes. En fait dans ces deux instructions c'est seulement à la deuxième que l'on peut mettre l'élément $Vardecl$ dans l'environnement. La première instruction `Parameter` sert uniquement à ouvrir la section. Au moment où nous interpréterons `Inhabits`, il faudra définir l'élément $Vardecl$ et fermer la section. Le seul problème est qu'il faudra au moment de l'interprétation de `Inhabits` se souvenir de deux choses : le nom de la variable et sa force (qui est portée en partie par le mot clé `Parameter` qui indique que la variable est globale). Retrouver le nom est chose facile puisque c'est le nom de la dernière section ouverte, mais il faut stocker la portée dans l'environnement.

Pour cela on ajoute encore une sorte d'éléments à l'environnement : les portées (Scope) :

```
| Scope of strength
```

Nous traduirons donc :

Parameter x.	[push_scope 0 ; open_section x]
Inhabits t.	[construct t ; def_var ; pop_scope ;close_section]

Notons que *def_var* est maintenant une commande de la machine sans arguments, et qu'il faudra qu'elle trouve les informations de nom et de portée de l'environnement.

De même pour la définition de variable locale :

Variable x.	[push_current_scope ; open_section x]

On peut de même donner la traduction des instructions utilisées pour poser des axiomes et définir des constantes.

Axiom x.	[push_scope 0 ; open_section x]
Assumes t.	[construct t ; def_ax ; pop_scope ;close_section]
Hypothesis.	[push_current_scope ; open_section x]
Definition x.	[push_scope 0 ; open_section x]
Body t.	[construct t ; def_const ; pop_scope ;close_section]
Local x.	[push_current_scope ; open_section x]

Théorèmes :

Le cas des théorèmes est rendu un peu plus difficile par le fait qu'il y a trois instructions et que l'on veut pouvoir écrire les deux variantes :

```
Theorem I.

Statement B->B.

      Hypothesis x.
      Assumes B.

Proof x.



Theorem I'.
```

```
      Hypothesis x.
      Assumes B.

Statement B.

Proof x.
```

C'est-à-dire que l'on veut avoir certains éléments locaux à la preuve et d'autres à tout le théorème (preuve et énoncé). Prenons un exemple où l'on a les deux :

```
Theorem I.

      Variable B.
      Inhabits Prop.

Statement B->B.

      Hypothesis x.
      Assumes B.

Proof x.
```

Quand on rencontre `Proof x` on commence par mettre la valeur x dans VAL puis on abstrait l'hypothèse x dans VAL ce qui donne [x :B]x, puis on vérifie le type de VAL avec $B \Rightarrow B$ puis on abstrait B dans VAL ce qui donne $[B : Prop][x : B]x : (B : Prop)(B \Rightarrow B)$ et enfin on empile ce théorème dans l'environnement. Il faut donc ouvrir deux sections : une au moment de `Theorem`, l'autre au moment de `Statement` et fermer ces deux sections au moment de `Proof`. De plus la première fermeture de section doit abstraire x non seulement dans les éventuels éléments de l'environnement qui seraient définis après elle mais aussi dans VAL. Nous devons donc ajouter une instruction *close_section_val* à la machine constructive. Enfin nous allons donner à la première section le nom I (le nom du théorème) et à la deuxième le nom $proof\_of\_I$. Nous devons donc ajouter une instruction à la machine qui ouvre une section dont le nom est "proof_of_"x où x est le nom de la dernière section ouverte.

Theorem x.	[push_scope 0 ; open_section x]
Statement t.	[construct t ; conjecture ; open_section_proof]
Proof t.	[construct t ;close_section_var ;verify ;def_th ;close_section]
Remark x.	[push_current_scope ; open_section x]

2.3.3 Règles de transition

Nous avons donc défini une machine dont le jeu d'instructions est le suivant :

- construct t
- def_var
- def_ax
- def_const
- def_th
- conjecture
- verify
- open_section x
- open_section_proof
- close_section
- close_section_var
- push_scope n
- push_current_scope
- pop_scope

A ces 14 instructions correspondent 14 règles de transition :

Règle *construct* :

$$\frac{ENV = e \;\; TEXT = (construct\ t) :: r \;\; Ty(e,t) \;\; SEC = s}{ENV = e \;\; TEXT = r \;\; VAL = t \;\; SEC = s}$$

Règle def_var :

$$\frac{ENV = e \;\; TEXT = (def\_var) :: r \;\; VAL = t \;\; SEC = s}{ENV = (Vardecl(Decl(current\_section(e), Judge(t, Type, Object))), current\_scope(e)) :: e \;\; TEXT = r \;\; SEC = s}$$

Règle def_const :

$$\frac{ENV = e \;\; TEXT = (def\_const) :: r \;\; VAL = t \;\; SEC = s}{ENV = (Constdecl(Decl(current\_section(e), Judge(t, PT(e,t), Object))), current\_scope(e)) :: e \;\; TEXT = r \;\; SEC = s}$$

Règle def_ax :

$$\frac{ENV = e \;\; TEXT = (def\_ax) :: r \;\; VAL = t \;\; SEC = s}{ENV = (Vardecl(Decl(current\_section(e), Judge(t, Prop, Proof))), current\_scope(e)) :: e \;\; TEXT = r \;\; SEC = s}$$

Règle def_th :

$$\frac{ENV = e \;\; TEXT = (def\_th) :: r \;\; VAL = t \;\; SEC = s}{ENV = (Constdecl(Decl(current\_section(e), Judge(t, PT(e,t), Proof))), current\_scope(e)) :: e \;\; TEXT = r \;\; SEC = s}$$

Règle *conjecture* :

$$\frac{ENV = e \;\; TEXT = (conjecture) :: r \;\; VAL = t \;\; SEC = s}{ENV = (Cast(Judge(t, Prop, Proof))) :: e \;\; TEXT = r \;\; SEC = s}$$

Règle $verify$:

$$\frac{ENV = (Cast(Judge(t', Prop, Proof))) :: e \;\; TEXT = (verify) :: r \;\; VAL = t \;\; Verif(e, PT(e,t), t') \;\; SEC = s}{ENV = e \;\; TEXT = r \;\; VAL = t \;\; SEC = s}$$

Règle *open_section* :

$$\frac{ENV = e \;\; TEXT = (open\_section \; x) :: r \;\; SEC = s}{ENV = (Section \; x) :: e \;\; TEXT = r \;\; SEC = s + 1}$$

Règle *open_section_proof* :

$$\frac{ENV = e \;\; TEXT = (open\_section\_proof) :: r \;\; SEC = s}{ENV = (Section \text{“}proof\_of\_\text{''}(current\_section(e)) :: e \;\; TEXT = r \;\; SEC = s + 1}$$

Règle *close_section* :

$$\frac{ENV = l@[Section \; x]@l' \;\; No \; section \; in \; l \;\; TEXT = (close\_section) :: r \;\; SEC = s}{ENV = update(l, s)@l' \;\; TEXT = r \;\; SEC = s}$$

Règle *close_section_var* :

$$\frac{ENV = l@[Section \; x]@l' \;\; No \; section \; in \; l \;\; TEXT = (close\_section\_var) :: r \;\; SEC = s \;\; VAL = t}{ENV = tl(update(t :: l, s))@l' \;\; TEXT = r \;\; SEC = s \;\; VAL = hd(update(t :: l, s))}$$

Règle *push_scope* :

$$\frac{ENV = e \;\; TEXT = (push\_scope \; n) :: r \;\; SEC = s}{ENV = (Scope \; n) :: e \;\; TEXT = r \;\; SEC = s}$$

Règle *push_current_scope* :

$$\frac{ENV = e \;\; TEXT = (push\_current\_scope) :: r \;\; SEC = s}{ENV = (Scope(s)) \;\; TEXT = r \;\; SEC = s}$$

Règle *pop_scope* :

$$\frac{ENV = l@[Scope \; n]@l' \;\; No \; scope \; in \; l \;\; TEXT = (pop\_scope) :: r \;\; SEC = s}{ENV = l@l \;\; TEXT = r \;\; SEC = s}$$

Les fonctions $current\_section(e)$ et $current\_scope(e)$ renvoient le nom de la dernière section ouverte et la force portée par le dernier élément *Scope* mis dans l'environnement.

Les deux règles *close_section* et *close_section_var* utilisent une fonction *update* qui élimine les éléments locaux et qui abstrait les variables éliminées et remplace les noms symboliques des constantes éliminées par leur valeur dans les bons éléments.

Cette fonction est assez lourde à décrire, peut-être vaut-il mieux se contenter de la description informelle du mécanisme de fermeture de section donné plus haut.

2.3.4 Non-localité des règles de transition

Remarquons que les règles *close_section* et *close_section_var* sont non locales en ce sens qu'elles dépilent tous les éléments du sommet de la pile ENV jusqu'à la première tête de section pour tous les modifier et les réempiler. Pour cela on est obligé d'utiliser une petite pile annexe qui garde les éléments dépilés ou si l'on préfère d'accéder à des éléments qui ne sont pas au sommet de la pile.

Il est donc sans doute possible de décrire le mécanisme actuel de la machine constructive par un système de règles locales mais avec une structure d'environnement plus complexe.

Cette non-localité vient du fait que l'on veut écrire par exemple une hypothèse puis plusieurs théorèmes puis abstraire cette hypothèse dans tous les théorèmes. Si on veut garder un mécanisme de section (c'est-à-dire de partage d'hypothèses entre différents objets globaux) cette non-localité est incontournable.

En revanche cette non-localité ne provient pas de la non localité de la déduction naturelle notée avec un seul symbole du métalangage pour dénoter la conséquence. En effet, bien que l'on veuille pouvoir noter par un lemme toutes les étapes de la dérivation en déduction naturelle (c'est-à-dire n'avoir que des preuves de lemme qui soient u ou $(u\ v)$ où u et v sont des symboles déjà définis), il est tout à fait possible, *si on ne veut garder aucun de ces lemmes,* d'avoir un mécanisme local de fermeture de section : si tous les éléments locaux ne sont attachés qu'à un élément global, au moment où l'on ferme une section il y a au sommet de la pile *un* élément global et des éléments locaux. Il suffit de mettre l'élément global dans le registre VAR, et de les dépiler un par un en remplaçant les noms symboliques des constantes par leur valeur et en abstrayant les variables dans la valeur courante VAR. Puis une fois la tête de section dépilée on empile la valeur courante VAR dans l'environnement. On a alors un mécanisme de localité qui permet de s'affranchir des règles $\Rightarrow$-intro et $\forall$-intro mais on a perdu les mécanismes de partage d'éléments et de structuration du texte mathématique. Tout dépend donc de l'usage que l'on veut faire d'un mécanisme de sections.

3 Un exemple : le Théorème de Tarski

3.1 Le Théorème de Tarski en vernaculaire mathématique

Nous pouvons maintenant donner un exemple de théorème écrit dans le vernaculaire que nous avons défini.

Nous allons montrer que sur un ensemble complètement partiellement ordonné toute fonction croissante a un point fixe.

Soit A un ensemble. Soit $\prec$ un ordre partiel complet sur A. Soit f une fonction croissante.

Considérons l'ensemble U = $\{x | x \prec f(x)\}$.

Cet ensemble a une borne supérieure M (l'ordre est complet).

Nous allons monter que M est un point fixe de f.

Remarque 1 Si $y \in U$ alors $y \prec f(M)$.

Si y est dans U alors $y \prec M$ donc f étant croissante $f(y) \prec f(M)$ comme par ailleurs $y \prec f(y)$ puisque y est dans U on en déduit $y \prec f(M)$ par transitivité.

Remarque 2 $M \prec f(M)$.

$f(M)$ est un majorant de U (remarque précédente) et M est le plus petit de ces majorants.

Remarque 3 $f(M) \prec f(f(M))$.

Car f est croissante.

Remarque 4 $f(M) \prec M$.

D'après la remarque précédente $f(M) \in U$ donc M étant un majorant de U $f(M) \prec M$.

Remarque 5 $f(M) = M$.

Par antisymétrie d'après les remarques 2 et 4.

3.2 Le Théorème de Tarski en vernaculaire du Calcul des Constructions

```
Parameter A.
Inhabits Type.

Parameter R.
Inhabits A->A->Prop.

Parameter Eq.
Inhabits A->A->Prop.

Axiom Assym.
Assumes (x:A)(y:A)((R x y) -> (R y x) -> (Eq x y)).

Axiom Trans.
Assumes (x:A)(y:A)(z:A)((R x y) -> (R y z) -> (R x z)).
```

```
Parameter f.
Inhabits A->A.

Axiom Incr.
Assumes (x:A)(y:A)((R x y) -> (R (f x) (f y))).

Definition Lub.
Body [m:A][S:A->Prop](and ((x:A) ((S x) -> (R x m)))
                          ((y:A)  (((x:A) ((S x)->(R x y))) -> (R m y)))).

Axiom Complete.
Assumes (S:A->Prop)(<A> Ex ([x:A] (Lub x S))).

Definition Under.
Body [x:A](R x (f x)).

Theorem Exist_lub_under.
Statement <A> Ex ([m:A] (Lub m Under)).
Proof (Complete Under).

Theorem Tarski1.

Statement ((M:A) ((Lub M Under) -> <A> Ex  ([m:A] (Eq m (f m))))).

   Variable M.
   Inhabits A.

   Hypothesis LeastUp.
   Assumes (Lub M Under).

   Remark Up.
   Statement (x:A) ((R x (f x)) -> (R x M)).
   Proof (proj1 ((x:A) ((R x (f x)) -> (R x M)))
                ((x:A)(((y:A)(R y (f y))->(R y x))->(R M x)))
                LeastUp).

   Remark Least.
   Statement (x:A)(((y:A)(R y (f y))->(R y x))->(R M x)).
   Proof (proj2 ((x:A) ((R x (f x)) -> (R x M)))
```

```
                ((x:A)(((y:A)(R y (f y))->(R y x))->(R M x)))
                LeastUp).

   Remark One.

   Statement (y:A)(Under y)->(R y (f M)).

      Variable y.
      Inhabits A.

      Hypothesis v.
      Assumes (Under y).

      Remark T.
      Statement (R y M).
      Proof (Up y v).

      Remark T'.
      Statement (R (f y) (f M)).
      Proof (Incr y M T).

   Proof (Trans y (f y) (f M) v T').

   Remark Two.
   Statement (R M (f M)).
   Proof (Least (f M) One).

   Remark Three.
   Statement (R (f M) (f (f M))).
   Proof (Incr M (f M) Two).

   Remark Four.
   Statement (R (f M) M).
   Proof (Up (f M) Three).

   Remark Five.
   Statement (Eq M (f M)).
   Proof (Assym M (f M) Two Four).

Proof (ex_intro A ([m:A] (Eq m (f m))) M Five).
```

```
Theorem Tarski.
Statement <A> Ex ([x:A](Eq x (f x))).
Proof (Exist_lub_under (<A> Ex ([x:A](Eq x (f x))))  Tarski1).
```

Conclusion : Quelques directions futures

L'exemple du théorème de Tarski montre qu'il reste bien du travail pour que vernaculaire mathématique et vernaculaire du Calcul des Constructions se rejoignent, bien que la preuve en vernaculaire mathématique ait été écrite très soigneusement.

On peut classer les différences en plusieurs catégories :

Mécanismes élémentaires contre règles de déduction

Un système tel que le vernaculaire du Calcul des Constructions se caractérise par deux choses : l'ensemble des opérations élémentaires (prouver un théorème, poser un axiome, etc.) et l'ensemble des règles de déduction utilisées. Les systèmes de déduction (déduction naturelle, etc.) privilégient au maximum le second au détriment du premier, en donnant un ensemble de règles riche mais en n'autorisant qu'une seule opération élémentaire : prouver un théorème. De ce fait, du point de vue logique, ces systèmes se prêtent bien à l'étude puisque la notion de preuve est assez facilement formalisable. En revanche, la présence de règles inhabituelles rend difficile l'écriture et la lecture des preuves non triviales. Ici nous avons rajouté des opérations élémentaires (définir un élément local à un autre, ouvrir et fermer une section) et supprimé des règles de déduction ($\Rightarrow$-intro, $\forall$-intro). La puissance d'expression du langage reste comme nous l'avons vu inchangée. Un tel système se prête moins bien à la métathéorie du calcul des prédicats intuitionniste, mais mieux à l'écriture de preuves.

Dans la preuve écrite en vernaculaire mathématique nous avons montré l'existence d'une borne supérieure à U puis nous l'avons appelée M, dans la suite du raisonnement nous avons utilisé le fait que M était était borne supérieure de U puis quand nous avons montré que M était point fixe de f nous avons conclu qu'il existait un point fixe à f.

Dans le vernaculaire du Calcul des Constructions nous avons montré qu'il existait une borne sup à U puis nous avons eu l'idée d'utiliser la règle

$\exists$-elim. Pour cela nous avons dû montrer le théorème $Tarski1$ d'énoncé :

```
((M:A) ((Lub M Under) -> <A> Ex  ([m:A] (Eq m (f m))))).
```

Pour montrer ce théorème nous avons dû poser une variable M et supposer que cette variable était borne supérieure de U puis montrer qu'elle était point fixe de f puis en déduire que f avait un point fixe, et enfin une fois le théorème $Tarski1$ terminé, montrer le théorème de Tarski en utilisant la règle $\exists$-elim.

Ce fonctionnement de le règle $\exists$-elim introduit une cassure peu naturelle dans le texte. Il faut autoriser dans le vernaculaire du Calcul des Constructions un mécanisme similaire à celui du vernaculaire mathématique. Ce mécanisme déduisant automatiquement les règles à appliquer pour construire le λ-terme preuve. Ici encore on rajoutera un mécanisme et on supprimera des règles.

On pourrait de même faire le procès de la règle $\vee$-elim qui devrait être remplacée par le mécanisme de la démonstration par cas.

Culture mathématique

Dans le texte écrit en vernaculaire mathématique certaines notions comme celle de borne supérieure sont utilisées sans être définies. Ici on suppose que le lecteur a une certaine culture mathématique, ce que nous ne pouvons faire dans le Calcul des Constructions. Pour permettre de telles preuves allusives dans le Calcul des Constructions il faudra écrire un prélude qui définira un ensemble de connaissances élémentaires qui sont en général supposées dans les textes de mathématiques.

Une partie importante de la culture mathématique consiste en la connaissance d'un vocabulaire de base. Ici nous n'avons pas utilisé la notion d'ensemble, représentant l'ensemble U par le prédicat $Under$, $(Under\ x)$ étant la proposition $(x \in U)$. Pour écrire naturellement des preuves, il faut autoriser dans le Calcul des Constructions l'utilisation du vocabulaire de la théorie naive des ensembles. Cette possibilité n'implique en rien un choix sur les fondements des mathématiques adoptés. Il est tous à fait possible d'utiliser le vocabulaire de la théorie des ensembles et de l'algèbre des parties d'un ensemble sans prendre parti pour la thèse "Tout objet est un ensemblé' ou "$\pi \in e$ a un sens". Comme il est montré dans [10], une théorie typée des ensembles comme celle décrite dans [1] semble un outil adéquat dans le Calcul des Constructions.

Langue naturelle

En Vernaculaire mathématique beaucoup de formules sont exprimées par des phrases, ainsi on préfère écrire x est dans U que $x \in U$, ou M est un point fixe de f plutôt que $f(M) = M$ ou $Fixpoint(M, f)$.

Décidabilité des preuves et application à l'écriture de preuves allusives

Comme nous l'avons vu les preuves sont maintenant dans le vernaculaire du Calcul des Constructions comme des λ-termes sans abstraction, c'est-à-dire comme des arbres. (Naturellement il peut rester des abstractions dans la partie objet des preuves, par exemple dans la règle $\forall$-elim, mais si on reste au premier ordre par exemple il n'y a plus du tout d'abstractions.)

Quand on a un ensemble A de variables typées et une proposition P, il est indécidable de savoir s'il existe ou non un λ-terme t, dont les variables libres sont dans A et qui soit de type P (Théorème de Church). Mais en affaiblissant un peu le Calcul des Prédicats, on obtient parfois des systèmes décidables : l'arithmétique de Pressburger, le calcul propositionnel, la géométrie élémentaire, etc.

On peut aussi affaiblir le calcul des prédicats en supprimant des règles de déduction naturelle. Par exemple, si on supprime les règles $\Rightarrow$-intro et $\forall$-intro et que l'on reste au premier ordre, tester s'il existe un arbre de type P dont les variables sont dans A devient peut-être décidable. Il devrait être alors possible d'écrire comme preuve de la remarque 5 : "par antisymétrie d'après les remarques 2 et 4", la machine calculant que la preuve est $(Assym\ M\ (f\ M)\ Two\ Four)$. On voit ici une utilisation que l'on peut faire de la décidabilité d'un sous-système du calcul des prédicats : permettre de noter de façon allusive des preuves dans ce sous-système, en laissant à l'utilisateur le travail d'écrire ce qui est hors du système décidable (ici l'utilisateur choisit les variables et les hypothèses à introduire). Bien sûr, pour un théorème non trivial c'est le travail difficile qui est laissé à l'utilisateur, mais celui-ci apprécie quand même de ne pas avoir à faire le travail trivial en plus du travail difficile.

On voit donc sur cet exemple que si le vernaculaire mathématique autorise les preuves allusives, c'est parce que le calcul des prédicats a d'importantes parties décidables voire triviales (ce qui n'est pas la même chose comme en témoigne la géométrie élémentaire), et que la raison pour laquelle on s'autorise l'ellipse : A et $A \Rightarrow B$ donc trivialement B et non : les axiomes de Peano, donc trivialement la conjecture de Fermat, n'est pas uniquement

culturelle, mais aussi mathématique.

Actuellement, il existe dans le Calcul des Constructions un système interactif de synthèse de preuves qui construit la preuve formelle d'un théorème à partir de sa conclusion et de directions de recherche : les tactiques. Nous l'avons vu, un Vernaculaire de haut niveau se doit d'intégrer un mécanisme de synthèse de preuves pour combler les étapes triviales du raisonnement. Le vernaculaire et la synthèse devront donc se rejoindre en un langage unique. En revanche le rôle de l'interactivité dans un système tel que le Calcul des Constructions est moins bien compris. Il faudra comprendre s'il faut deux modes de fonctionnement (l'un, interactif de recherche de démonstration et un autre non interactif de saisie d'une démonstration éventuellement lacunaire) ou si on peut se contenter d'un seul mode dans lequel le mécanisme de synthèse est utilisé pour trouver la preuve dans le premier cas et uniquement pour combler les lacunes dans le second.

Références

[1] N.G. De Bruijn "The Mathematical Vernacular, A Language For Mathematics With Typed Sets" Procedings, Workshop on Programming Logic, Marstrand, Sweden, 1987

[2] N.G. De Bruijn "The Mathematical Vernacular : Examples" Non publié.

[3] Th. Coquand. "Une théorie des constructions." Thèse de troisième cycle, Université Paris VII, 1985.

[4] Th. Coquand. "An analysis of Girard's paradox." First IEEE Symposium on Logic in Computer Science, Boston, June 1986, 227–236.

[5] Th. Coquand. "Metamathematical investigations of a Calculus of Constructions" in The Calculus of Constructions, Collected papers, INRIA, 1989.

[6] Th. Coquand, G. Huet. "Constructions : A Higher Order Proof System for Mechanizing Mathematics." EUROCAL85, Linz, Springer-Verlag LNCS 203 (1985).

[7] T. Coquand, G. Huet. "The Calculus of Constructions." Information and Computation, Volume 76, 1988.

[8] G. Dowek "A Vernacular Syllabus" in The Calculus of Constructions, Collected papers, INRIA, 1989.

[9] J.Y. Girard "Types and Proofs" translated and with apendices by P. Taylor and Y. Lafont. Cambrige University Press, 1989.

[10] H. Herbelin "Le théorème de Schoeder-Bernstein dans le Calcul des Constructions". Mémoire de D.E.A. Paris VII, 1988.

[11] G. Huet "A Uniform Approach to Type Theory" Rapport de recherche INRIA no 795 , Février 88. Courses Notes, Institute on Logical Foundation of Functional Programming, University of Texas at Austin, June 1987. To appear, Addison-Wesley, 1988.

[12] G. Huet "The Constructive Engine" in The Calculus of Constructions, Collected papers, INRIA, 1989.

[13] Ch. Paulin-Mohring. "Extraction de programmes dans le Calcul des Constructions." Thèse, Université Paris VII, 1989.

[14] Ch. Paulin-Mohring. "Extracting Fω programs from proofs in the Calculus of Construction." Proceedings of POPL 1989.

[15] D. Simon "Checking Natural Language Proofs" 9th International Conference on Automated Deduction.

[16] R.S. Strichartz "An International Language for Mathematics." The Mathematical Inteligencer, Vol 11, No 1, 1989.