an intuitionistic lambda calculus with exceptions

Georges Mounier

Université de savoie
U.F.R. Sciences Fondamentales et Appliquées

Thèse

pour obtenir le grade de

Docteur de l'Université de Savoie

Discipline : Mathématiques

présentée et soutenue publiquement le 19 février 1999

par Monsieur Georges MOUNIER

Titre :

Un λ-calcul intuitionniste avec exceptions

Directeur de thèse : Monsieur René David

Devant le jury formé de :

Monsieur René David
Monsieur Gilles Dowek
Monsieur Philippe de Groote
Monsieur Gérard Huet, président
Monsieur François Lamarche

Sommaire

Préambule

Dans mon mémoire de D.E.A, j'ai décrit une machine abstraite, extension de la machine à environnement de Krivine ([3]), qui permettait de réduire les termes du $\lambda\mu$-calcul de Parigot ([20]). Et j'ai implémenté cette machine dans un programme écrit dans le langage Caml ([26]).

En continuité avec ce premier travail, j'ai commencé ma thèse avec l'idée d'une recherche sur les mécanismes de contrôle en $\lambda\mu$-calcul ou dans un autre système adapté.

Je connaissais les travaux de Manoury et Simonot ([16]) de description d'une stratégie automatique pour prouver la terminaison de fonctions définies par des équations récursives. Je savais qu'ils avaient implémenté cette stratégie dans un programme - Propre - qui permettait de passer des spécifications d'une fonction à un programme pour cette fonction. Et j'envisageais de prolonger leur travail en y intégrant la gestion d'un mécanisme de contrôle particulier : les exceptions.

Je recherchais donc un système de λ-calcul permettant d'étendre le système AF2 de Krivine ([10]) en y intégrant la gestion des exceptions, et j'ai commencé par étudier quelques uns des systèmes qui ont été proposés en vue de donner une interprétation logique des mécanismes de contrôle des langages fonctionnels.

Comme on le verra dans l'introduction, aucun des systèmes que j'ai examinés ne convenait au but que je m'étais fixé. Aucun ne procurait les outils nécessaires pour exprimer de façon équationnelle les spécifications des fonctions qui m'intéressaient : des fonctions qui, pour être plus efficaces, déroutaient le flux normal de l'évaluation en utilisant des mécanismes d'échappement.

Insatisfait des solutions existantes, j'ai donc été amené à construire un nouveau système de λ-calcul typé dans lequel on puisse exprimer les spécifications d'une fonction qui utilise les exceptions et en prouver la terminaison. Dans cette construction j'ai été guidé autant par la façon dont le langage Caml gére les exceptions que par l'aspect logique du problème.

Le résultat est un λ-calcul dont le socle est le système AF2, mais qui l'augmente d'un ensemble de constructions (λ-termes, termes d'individus, types) permettant de gérer les exceptions.

Ce nouveau système a les propriétés attendues habituellement d'un λ-calcul

typé : confluence, forte normalisation, conservation du type par réduction, et constitue un cadre adéquat pour prouver la terminaison de fonctions.

L'automatisation de la preuve de terminaison n'a pas été tentée, elle dépassait le cadre de la recherche engagée.

Un résultat surprenant a été obtenu sur ce système : il n'est pas basé sur la logique classique mais sur la logique intuitionniste (plus précisément c'est le cas d'une traduction du système logique dans le calcul des propositions du second ordre). Cela pourrait remettre en cause la vision dominante des mécanismes de contrôle comme nécessairement tous liés à la logique classique : ce n'est peut être pas le cas des exceptions.

Chapitre 1

Introduction

On sait aujourd'hui l'importance de disposer de programmes dont la correction est prouvée. L'une des façons d'obtenir cette garantie de bon fonctionnement est de construire les programmes dans le cadre d'un langage de programmation un peu particulier : le λ-calcul typé. Alors, la correspondance entre formules logiques et types, aussi nommée isomorphisme de Curry-Howard, permet d'associer à la preuve d'une formule T un programme de type T.

Longtemps cette correspondance a été restreinte à la logique intuitionniste. Pourtant on savait ([9]) que toute formule $\sum_1^0$ prouvable dans l'arithmétique de Peano est aussi prouvable dans l'arithmétique de Heyting, et, du point de vue du programmeur, la constructivité est nécessaire seulement pour cette classe de formules.

Pour étendre la correspondance entre preuves et programmes à la logique classique, une première solution est d'utiliser une traduction (comme celle de Gödel ou celle de Kolmogorov) pour traduire la preuve classique en preuve intuitionniste et ensuite extraire de la preuve intuitionniste le programme. Mais, avec cette méthode, le comportement algorithmique du programme obtenu peut être très éloigné de celui que la preuve classique laissait présager.

Une autre solution consiste à trouver des techniques pour extraire directement le contenu constructif des preuves classiques.

Depuis que Griffin ([7]) a proposé un typage de l'opérateur $\mathcal{C}$ de Felleisen ([15]), montrant ainsi que la logique classique était reliée, à travers l'isomorphisme de Curry-Howard, aux opérateurs de contrôle des langages de programmation fonctionnelle, de nombreux auteurs ont introduit de nouveaux systèmes en vue d'étudier le contenu calculatoire de la logique classique, ou de donner une interprétation logique des mécanismes de contrôle des langages fonctionnels. Nous étudions plus en détail trois de ces systèmes : $\lambda^{\rightarrow}_{exn}$ le calcul de P. de Groote ([5]), le λ_c-calcul de Krivine ([13]) et le $\lambda\mu$-calcul de Parigot ([20]).

1.1 $\lambda_{exn}^{\rightarrow}$ de P. de Groote

Dans [5] P. de Groote décrit un λ-calcul simplement typé basé sur un mécanisme de capture des exceptions à la ML.

Le langage ML ([24]) ne diffère du langage Caml - étudié plus loin (paragraphe 1.4) - que sur des points non significatifs pour notre propos, aussi nous commençons directement par l'analyse théorique de la gestion des exceptions en ML telle que la fait P. de Groote.

1.1.1 Exceptions en ML

La **construction** des exceptions peut se décrire par la règle de typage :
si $\Gamma \Vdash m{:}Exn$ alors $\Gamma \Vdash (raise\ m){:}T$ pour tout type T.

La **propagation** des exceptions peut se représenter par les deux règles de réduction :
$(v\ (raise\ m)) \rightarrow (raise\ m)$
$((raise\ m)\ n) \rightarrow (raise\ m)$
où v représente une valeur, et m et n des termes quelconques.

La **capture** des exceptions peut être décrite par la règle de typage:
si $\Gamma \Vdash m{:}T$ et $\Gamma \Vdash n{:}Exn \rightarrow T$ alors $\Gamma \Vdash m\ handle\ n{:}T$
(la construction $m\ handle\ n$ de ML est l'équivalent de $try\ m\ with\ n$ de Caml).

1.1.2 Modification de ML

A cause de la règle de typage de $(raise\ m)$, de Groote identifie le type des exceptions Exn avec le type logique $\bot$. Mais alors, si on veut préserver la consistance logique du système de typage, la déclaration d'un constructeur d'exceptions ne peut être effectuée au niveau supérieur : elle n'a de sens que si elle peut être déchargée. D'autre part la règle de typage de *handle* doit intégrer le fait que n n'est pas une expression mais un filtrage.

Une nouvelle règle de typage corrige ces problèmes :
si $\Gamma, y{:}A \rightarrow Exn \vdash m{:}T$ et $\Gamma, x{:}A \vdash n{:}T$ alors $\Gamma \vdash let\ y{:}\neg A\ in\ m\ handle\ (y\ x) \mapsto n\ tel{:}T$.
Si on identifie le type Exn avec le type $\bot$, et si on se souvient que $A \rightarrow \bot \stackrel{def}{=} \neg A$, on voit que cette règle est une règle d'élimination du $\vee$ dans le cas du tiers exclu, donc appartient à la logique classique.

1.1.3 Le λ-calcul

L'ensemble T des types est défini par la grammaire :
$T{:}:= A \mid \bot \mid (T \rightarrow T)$ où A désigne un type de base.

L'ensemble t des termes est défini, à partir des λ-variables notées x, et des variables d'exception notées y par :

$t::= x \mid y \mid \lambda x\, t \mid (t\ t) \mid (raise\ t) \mid let\ y{:}\neg A\ in\ t\ handle\ (y\ x) \mapsto t\ tel.$

L'ensemble v des valeurs est lui défini par $v::= x \mid y \mid \lambda x\, t \mid (y\ v)$.

Les **règles de typage** sont l'axiome, l'introduction et l'élimination du connecteur $\rightarrow$ ainsi que les deux règles vues plus haut : typage de *raise* et *handle*.

Le système logique correspondant à ces règles de typage est consistant par rapport à la logique classique (propositionnelle), il est même complet.

Les premières **règles de réduction** proposées par de Groote sont les suivantes (v, v_1 et v_2 sont des valeurs, m et n des termes) :

- la β-réduction par valeur $(\lambda x\ m\ v) \rightarrow m[x:=v]$
- deux règles de réduction de raise

  $(v_1\ (raise\ v_2)) \rightarrow (raise\ v_2)$

  $((raise\ v)\ m) \rightarrow (raise\ v)$
- trois règles de réduction de *handle*

  $let\ y{:}\neg A\ in\ v\ handle\ (y\ x) \mapsto n\ tel \rightarrow v$

  $let\ y{:}\neg A\ in\ (raise\ (y\ v))\ handle\ (y\ x) \mapsto n\ tel \rightarrow n[x:=v]$

  $let\ y{:}\neg A\ in\ (raise\ (z\ v))\ handle\ (y\ x) \mapsto n\ tel \rightarrow (raise\ (z\ v))$ (si $z \neq y$).

Elles peuvent être reliées à des **réductions de preuve**. Donnons en deux exemples.

La première règle de réduction de *raise* a pour interprétation logique le passage de la preuve :

$$\dfrac{\Gamma \vdash v_1 : A \rightarrow B \qquad \dfrac{\Gamma \vdash v_2 : \bot}{\Gamma \vdash (raise\ v_2) : A}\ Raise}{\Gamma \vdash (v_1\ (raise\ v_2)) : B}\ App$$

à la preuve :

$$\dfrac{\Gamma \vdash v_2 : \bot}{\Gamma \vdash (raise\ v_2) : B}\ Raise$$

La seconde règle de réduction de *handle* a pour interprétation logique le

passage de la preuve :

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash v : A}{\Gamma, y : \neg A \vdash (y\ v) : \bot}\, App}{\Gamma, y : \neg A \vdash (raise\ (y\ v)) : T}\, Raise \qquad \Gamma, x : A \vdash n : T}{\Gamma \vdash let\ y : \neg A\ in\ (raise\ (y\ v))\ handle\ (y\ x) \mapsto n\ tel : T}\, Handle$$

à la preuve :

$$\dfrac{\Gamma, x : A \vdash n : T \qquad \Gamma \vdash v : A}{\Gamma \vdash n[x := v] : T}\, Subs$$

Malheureusement ces règles ne donnent pas la propriété de conservation du type : à cause du caractère local de la déclaration de y, des variables liées peuvent devenir libres dans la réduction. P. de Groote modifie donc les règles de réduction et obtient avec les nouvelle règles, que nous n'exposerons pas ici, la confluence et la préservation du type, ce qui lui assure que l'évaluation d'un terme typé, ne donnera pas une exception non capturée. Et ceci dans un λ-calcul typé dont il montre que la sémantique opérationnelle reste équivalente à celle de ML pour toutes les valeurs non exceptionnelles.

En résumé, $\lambda^{\rightarrow}_{exn}$ est un λ-calcul basé sur la logique classique : sur une interprétation calculatoire de l'élimination du tiers exclu. Il reflète, au prix de quelques complications, la sémantique du langage ML.

Mais il ne peut convenir à notre projet parce qu'il est basé sur le calcul des propositions et non sur le calcul des prédicats : les types de notre système doivent évidemment inclure des termes d'individus, si on veut pouvoir l'utiliser pour prouver des programmes. Nous avons tenté d'étendre $\lambda^{\rightarrow}_{exn}$ en un sytème où les types seraient des formules du calcul des prédicats du second ordre, mais nous n'avons pas réussi dans cette tentative qui est évoquée au paragraphe 1.5.2.

1.2 Le λ_c-calcul de Krivine

Dans [12] Krivine décrit une extension à la logique classique du système AF2 ([10], [14]) : il ajoute aux λ-termes habituels une constante C de type $\forall X(\neg\neg X \to X)$.

Les **règles de réduction** sont celles de la C-réduction de tête :

- $(\lambda x\ u\ t\ t_1 \cdots t_n) \vdash_c (u[x := t]\ t_1 \cdots t_n)$
- $(C\ t\ t_1 \cdots t_n) \vdash_c (t\ \lambda x\ (x\ t_1 \cdots t_n))$.

L'instruction $exit[.]$ du langage de programmation C peut être représentée par le terme $\lambda y\,(C\ \lambda x\, y)$ dans le sens suivant : soit t un terme dont un

sous terme est $(C\ \lambda x\ u)$; si, durant l'exécution (la C-réduction) de t, le sous terme arrive en tête, t a pris la forme $(C\ \lambda x\ u\ t_1 \cdots t_n)$ et va donc se réduire à $(\lambda x\ u\ \lambda x\ (x\ t_1 \cdots t_n))$ puis à u.

Le système de **typage** est celui de AF2 dans lequel la constante C a toujours le type $\forall X\,(\neg\neg X \to X)$, et on note $\Gamma \vdash_c t{:}T$ au lieu de $\Gamma, C{:}\forall X\,(\neg\neg X \to X) \vdash_{AF2} t{:}T$.

Dans ce système on peut attribuer au terme $\lambda y\,(C\ \lambda x\ y)$, présenté plus haut comme une représentation de l'instruction $exit[.]$, le type $\forall X\,(\bot \to X)$ qui est la règle de l'absurde de la logique intuitionniste.

Mais, en λ_c-calcul , seul le type $\bot$ est conservé dans la C-réduction.

Dans AF2 (et dans le système que nous construirons), on sait qu'un terme t de type $N[a]$ dans le contexte vide se réduit à un entier de Church. Ce n'est plus vrai en λ_c-calcul calcul, on doit utiliser les opérateurs de mise en mémoire (définition 8.1.1) pour obtenir l'entier de Church à partir d'un terme de type $N[a]$.

Le résultat exact est précisé par le lemme suivant : soit T est un opérateur de mise en mémoire pour $\underline{N}$ et f une variable quelconque : si $\vdash_c t{:}N[\boldsymbol{s}^n(\mathbf{0})]$ alors $(T\ f\ t) \to_c (f\ u)$ où u est se réduit à l'entier de Church $\underline{n}$.

En fait on a un résultat plus fort : pour chaque entier n, il existe un λ-terme u_n, équivalent à $\underline{n}$, tel que si $\vdash_c t{:}N[\boldsymbol{s}^n(\mathbf{0})]$ alors $(T\ f\ t) \to_c (f\ u)$ où u est obtenu à partir de u_n à l'aide d'une substitution : $u = u_n[x_1{:=}t_1, \cdots, x_n{:=}t_n]$.

Deux caractéristiques du λ_c-calcul peuvent être génantes si on veut l'utiliser comme base d'un système comme celui que nous projetons.

La stratégie de réduction du λ_c-calcul est fixée : on doit réduire de tête, ce qui signifie que les règles de réduction sont globales, elle s'appliquent seulement au terme tout entier.

S'il est logiquement satisfaisant, le λ_c-calcul n'assure que partiellement la conservation du type (seulement pour le type $\bot$) ce qui oblige à considérer que seuls les programmes de type $\bot$ sont exécutables, les programmes d'un autre type sont des modules qui servent à construire les programmes exécutables mais ne sont pas exécutables eux mêmes. Et on doit envisager l'existence d'un programme E (une sorte de système d'exploitation) de type $N \to \bot$ à qui on confiera l'entier à calculer et qui en supervisera l'exécution et affichera le résultat ou le transmettra à un autre programme.

Pour ces deux raisons, le λ_c-calcul n'est pas un bon modèle des langages de programmation fonctionnels. C'est, comme le dit d'ailleurs J. L. Krivine ([12]), un modèle des langages de programmation impérative.

1.3 Le $\lambda\mu$-calcul de M. Parigot

Michel Parigot a proposé ([20]) un λ-calcul typé: le $\lambda\mu$-calcul , fondé sur un système de déduction particulier la déduction naturelle classique (CND).

1.3.1 La déduction naturelle classique

La CND est un système de déduction naturelle à conclusions multiples obtenu en particularisant la «free-deduction» ([19]).

Les formules sont construites avec les opérateurs logique : $\neg, \rightarrow, \forall$ (quantificateur universel du premier et du second ordre).

Les règles de réduction (présentées avec des séquents) sont l'axiome et l'introduction et l'élimination des connecteurs $\rightarrow$, $\forall$ et $\neg$. Le système est classique parce que les séquents sont à conclusions multiples.

L'interprétation algorithmique de la CND suit la procédure d'élimination des coupures. Mais, dans ce système, il y a deux coupures : une coupure logique qui est celle de la déduction naturelle intuitionniste et une coupure structurelle spécifique de la CND. La coupure structurelle apparaît lorsque la conclusion est momentanément laissée inactive, puis est reprise comme formule principale dans l'application d'une règle; c'est la possibilité que l'on s'est donné d'avoir dans les séquents plusieurs termes en partie droite qui est à l'origine de cette coupure.

La coupure structurelle relative au connecteur $\rightarrow$ est la suivante. La preuve :

$$\dfrac{\dfrac{\dfrac{}{\Gamma_1 \vdash A \rightarrow B, \Delta_1}(d_1)}{\Gamma_2 \vdash A \rightarrow B, \Delta_2}(d_2) \quad \dfrac{}{\Gamma_3 \vdash A, \Delta_3}(d_3)}{\Gamma_2, \Gamma_3 \vdash B, \Delta_2, \Delta_3} R$$

se réduit en la preuve :

$$\dfrac{\dfrac{\dfrac{}{\Gamma_1 \vdash A \rightarrow B, \Delta_1}(d_1) \quad \dfrac{}{\Gamma_3 \vdash A, \Delta_3}(d_3)}{\Gamma_1, \Gamma_3 \vdash B, \Delta_1, \Delta_3} R}{\Gamma_2, \Gamma_3 \vdash B, \Delta_2, \Delta_3}(d_2)$$

1.3.2 Le $\lambda\mu$-calcul

La présentation que nous donnons ci-dessous du $\lambda\mu$-calcul n'est pas exactement celle de [20], mais c'est celle qui est aujourd'hui généralement adoptée (voir par exemple [4] ou [23]).

La syntaxe des **termes** est la suivante :
$T ::= x | \lambda x\, T | (T\, T) | \mu\alpha\, [\beta]T$ (x est une λ-variable, α et β deux μ-variables).

La **réduction** comporte trois règles :

- deux règles déduites de la réduction des deux coupures
  - β-réduction : $(\lambda x\ u\ v) \rightarrow_\mu u[x:=v]$
  - μ-réduction : $(\mu\alpha\ u\ v) \rightarrow_\mu \mu\alpha\ u[[\alpha]w:=[\alpha](w\ v)]$
    dans u on remplace chaque sous-terme w de nom α par $(w\ v)$
- et une règle de renommage des μ-variables : $[\alpha]\mu\beta\ u \rightarrow_\mu u[\beta:=\alpha]$.

La réduction ainsi définie est confluente.

Un contexte est de la forme Γ, Δ où Γ désigne l'ensemble des déclarations des λ-variables et Δ désigne l'ensemble des déclarations des μ-variables, avec la condition : si $\alpha:B \in \Delta$ alors B est de la forme $\neg A$.

Les règles de **typage** sont, outre l'axiome, l'introduction et l'élimination des connecteurs $\rightarrow$, $\forall$, les deux règles :

μ-abstraction si $\Gamma, \Delta, \alpha : \neg A \vdash_\mu t : \bot$ alors $\Gamma, \Delta \vdash_\mu \mu\alpha\ t:A$

μ-application si $\Gamma, \Delta, \alpha : \neg A \vdash_\mu t : A$ alors $\Gamma, \Delta \vdash_\mu [\alpha]\ t:\bot$

On montre que le type est préservé dans la réduction.

On peut écrire en $\lambda\mu$-calcul calcul des termes susceptibles de représenter les opérateurs de contrôle classiques.

Le terme $C = \lambda y\,\mu\alpha\ [\beta](y\ \lambda x\,\mu\delta\ [\alpha]x)$ a le type $\forall X(\neg\neg X \rightarrow X)$ et le fonctionnement suivant : $(C\ u\ t_1 \cdots t_n) \rightarrow_\mu \mu\alpha[\beta](u\ \lambda x\,\mu\delta[\alpha](x\ t_1 \cdots t_n))$. Il a donc un comportement proche de celui de l'opérateur C de Felleisen ([15]).

Le terme $CC = \lambda y\,\mu\alpha\ [\alpha](y\ \lambda x\,\mu\delta\ [\alpha]x)$ a le type $((A \rightarrow B) \rightarrow A) \rightarrow A$ et le fonctionnement suivant :

$(CC\ u\ t_1 \cdots t_n) \rightarrow_\mu \mu\alpha\ [\alpha]((u\ \lambda x\,\mu\delta\ [\alpha](x\ t_1 \cdots t_n))\ t_1 \cdots t_n)$. Il a donc un comportement proche de celui de l'opérateur *call/cc* du langage Scheme.

En $\lambda\mu$-calcul (comme en λ_c-calcul), un terme t de type $N[a]$ dans le contexte vide ne se réduit pas nécessairement à un entier de Church. Mais les opérateurs de mise en mémoire peuvent servir à reconnaître l'entier de Church caché derrière le terme t (ce résultat a été exposé dans la paragraphe relatif au λ_c-calcul).

Nous verrons plus loin plus en détail en quoi le $\lambda\mu$-calcul ne convenait pas à notre projet, disons simplement que son indifférence au équations - il ne comporte pas de règle permettant de remplacer un terme d'individu par un autre terme égal - ne nous permettait pas de lui faire jouer le rôle d'un système de preuve de terminaison de fonctions.

Dans la construction de mon système, si j'ai commencé en étudiant les sytèmes de λ-calcul existants, j'ai toujours eu présent à l'esprit, non comme

un modèle mais comme un exemple de réalisation possible, un langage de programmation fonctionnel qui pratique la synthèse de type et qui possède un mécanisme de contrôle basé sur les exceptions : Caml. Nous donnons donc une description informelle de la gestion des exceptions dans ce langage.

1.4 Exceptions en Caml

Déclaration En Caml ([27]) les erreurs sont des valeurs à part entière du langage, elles appartiennent à un type somme prédéfini `exn` et on les appelle des valeurs exceptionnelles.

Il existe en Caml des constructeurs d'exceptions prédéfinis, par exemple `Failure` est un constructeur prédéfini d'exceptions de type `string -> exn`. Mais on peut ajouter de nouveaux constructeurs au type `exn`, ils sont alors déclarés comme dans le cas d'un type somme. Par exemple la déclaration : `exception alpha of int` crée un constructeur `alpha` de type `int -> exn`.

Levée Les exceptions sont construites à l'aide de la fonction prédéfinie `raise` qui a le type `exn -> a` (pour un type a quelconque).

L'exception est déclenchée, on dit aussi dressée ou levée (comme dans «lever une perdrix»), lorsque l'expression `raise (alpha 3)` est évaluée : le calcul en cours est interrompu et l'expression `raise (alpha 3)` est propagée. Si, dans les calculs suspendus, l'exception n'est pas récupérée (voir plus loin), l'évaluation s'arrête au niveau global. L'exception et la valeur transportée sont alors affichées à ce niveau. Par exemple :

```
1 + raise (alpha (2+3));;
# Uncaught exception: alpha 5
```

Capture Une exception peut être récupérée et analysée de façon à décider de quelle manière terminer le calcul qui a provoqué la levée de l'exception. La construction `try e with filtre` permet cette capture des exceptions : si `e` s'évalue sans déclenchement d'exception, c'est la valeur de `e` qui est retournée, si une valeur exceptionnelle est levée pendant l'évaluation de `e`, alors cette valeur est filtrée avec les clauses de `filtre` (qui doit donc opérer sur des valeurs de type `exn`).

Exemple Considérons la fonction `tete` qui renvoie la tête d'une liste si la liste n'est pas vide et une valeur exceptionnelle sinon. Ecrivons une première fonction `tete1` :

```
let tete1 = function
[ ] -> raise (Failure "vide")
| x::_ -> x ;;
```

Le compilateur lui attribue le type : `# tete1 : 'a list -> 'a = <fun>`. La fonction `tete1` accepte donc comme paramètre aussi bien une liste d'entiers :

```
tete1 [1; 2] ;;
#- : int = 1
```

qu'une liste de booléens :

```
tete1 [true; false] ;;
#- : bool = true
```

dans le cas d'une liste vide elle renvoie une exception :

```
tete1 [] ;;
# Uncaught exception: Failure "vide"
```

Mais l'exception peut être capturée :

```
let tete l = try (tete1 l) with (Failure "vide") -> 0 ;;
# tete : int list -> int = <fun>
```

Remarquer que, cette fois, le type de la fonction indique qu'elle opère exclusivement sur les listes d'entiers (pour que la valeur renvoyée dans le cas général c'est à dire la tête de la liste soit cohérente avec la valeur 0 renvoyée en cas de liste vide). L'exception est bien capturée :

```
tete [] ;;
#- : int = 0
```

Cependant, dans cet exemple, la capture de l'exception est artificielle puisque, lorsque la liste est vide, que peut-on renvoyer d'autre qu'une exception?

Il faut remarquer que Caml a considéré la fonction `tete1` comme bien typée, puisque `raise (Failure "tete")` est, d'après la règle de typage de `raise`, d'un type quelconque. Pourtant l'évaluation de `tete1` avec comme paramètre la liste vide produit une valeur exceptionnelle non capturée. On voit donc que le fait qu'une fonction soit typée par Caml d'un type *a priori* sans surprise ne nous donne aucune garantie que l'évaluation de cette fonction ne produira pas une valeur exceptionnelle. C'est, de notre point de vue, une caractéristique de Caml que nous devrons impérativement éliminer dans notre système si nous voulons que le typage serve de preuve de correction des programmes.

Produit des entiers d'un arbre Le type d'un arbre binaire d'entiers se définit en Caml par :

```
type arbre =
  F of int
| T of arbre * arbre ;;
# Type arbre defined.
```

On peut ensuite définir récursivement le produit des entiers contenus dans un arbre :

```
let rec prod = function
  (F n) -> n
| T (g ,d) -> (prod g) * (prod d) ;;
# prod : arbre -> int = <fun>

prod (T(T((F 1), (F 0)), (F 2))) ;;
#- : int = 0
```

Le **produit** dit **rapide** est un exemple classique d'utilisation des exceptions, non pour signaler une erreur, mais comme un mécanisme normal de programmation : pour accélérer le calcul lorsque la donnée s'avère d'une forme particulière. Nous le programmons dans notre λ-calcul au paragraphe 8.3.2.

En Caml il se programme en deux temps. La fonction `prodrap1` renvoie une exception dès qu'une feuille de l'arbre renferme la valeur 0, remarquer que dans ce cas le calcul du produit des entiers est interrompu : la valeur exceptionnelle est propagée. La fonction `prodrap` se contente de capturer l'exception éventuelle produite par `prodrap1` et, dans ce cas, de rendre la valeur 0.

```
let rec prodrap1 = function
 (F n) -> if (n =0) then (raise (alpha 0)) else n
| T (g, d) -> (prodrap1 g) * (prodrap1 d) ;;
# prodrap1 : arbre -> int = <fun>

let prodrap a = try (prodrap1 a) with (alpha x) -> x ;;
# prodrap : arbre -> int = <fun>
```

La fonction `prodrap1`, bien que le type calculé par Caml pour le résultat soit `int`, peut produire une exception non capturée :

```
prodrap1 (T(T((F 1), (F 0)), (F 2))) ;;
# Uncaught exception: alpha 0
```

Ce n'est plus le cas de la fonction `prodrap`, elle capture l'exception :

```
prodrap (T(T((F 1), (F 0)), (F 2))) ;;
#- : int = 0
```

Cet exemple montre qu'en Caml, le type - le même pour les deux fonctions - ne peut rendre compte pleinement du comportement - différent pour les deux fonctions.

1.5 Un autre système?

Comme nous l'avons dit dans le préambule, aucun des systèmes examinés ne nous a semblé convenir comme base d'un système de preuve de terminaison de fonctions, si on veut pouvoir utiliser un mécanisme de contrôle comme les exceptions. En effet aucun ne nous donnait les outils nécessaires pour exprimer de façon équationnelle les spécifications d'une fonction intégrant des exceptions.

1.5.1 Produit rapide

Pour montrer ce que nous souhaitons pouvoir faire, reprenons l'exemple du produit rapide, que nous avons proposé en Caml (1.4), et que nous traitons en détail au paragraphe 8.3.2.

Le produit des éléments d'un arbre binaire d'entiers est défini classiquement par les équations : $h(feu(n)) = n$ et $h(tree(g,d)) = prod(h(g), h(d))$.

Le calcul de la fonction h est cependant rendu plus efficace si on remarque que lorsque l'on rencontre, sur l'une des feuilles de l'arbre, l'entier zéro, il est inutile de continuer : le résultat final est alors connu, c'est zéro. Mais comment exprimer formellement cette définition intuitive du produit rapide?

Une première voie possible serait de spécifier le produit rapide par une formule mathématique. Par exemple le produit efficace $pe(l)$ des éléments d'une liste l d'entiers pourrait être défini par : $(A \rightarrow pe(l) = 0) \vee (\neg A \rightarrow pe(l) = p(l))$ où $p(l)$ est le produit habituel des éléments d'une liste et A la formule $\exists n(n \in N \rightarrow item(n,l) = 0)$ (avec $item(n,l)$ n-ième élément de la liste l). Reste à extraire de cette spécification un λ-terme . C'est proche de ce que fait un système comme Coq ([2]), et c'est sans doute une piste prometteuse, mais nous n'avons pas exploré cette voie, car nous voulions rester dans un cadre proche de AF2.

L'autre possibilité, celle que avons choisie, consiste à prolonger le mode de définition des fonctions utilisé dans AF2, les équations (ou formules équationnelles), en créant de nouvelles constantes pour écrire les termes d'individus nécessaires. Les équations définissant le produit efficace des entiers d'un arbre prennent alors l'allure informelle suivante.

- $f'(feu(n)) = 0'$ si n est égal à zéro et $f'(feu(n)) = n$ sinon (le terme $0'$ représente le fait qu'on a rencontré zéro dans l'arbre et qu'on ne doit donc pas poursuivre le calcul)

- $f'(tree(g,d)) = prod'(f'(g), f'(d))$ avec $prod'(x,y) =$ dans le cas où x ou y est $0'$ alors $0'$, dans les autres cas $prod(x,y)$ (la valeur $0'$ ne doit pas être traitée comme un entier, en sa présence les calculs sont interrompus, et $0'$ est propagé)

– $f(a)$ est le résultat du calcul suivant : si $f'(a)$ est $0'$ alors $f(a)$ est zéro sinon $f(a)$ est $f'(a)$

Aucun des calculs examinés précédemment : $\lambda^{\rightarrow}_{exn}$, $\lambda\mu$-calcul ou λ_c-calcul ne nous permettait d'exprimer directement de façon satisfaisante les spécifications d'une fonction comme le produit rapide. Le premier, en effet, est un calcul simplement typé ce qui est rédhibitoire. Le second ne prend pas en compte de façon explicite les équations. Le λ_c-calcul utilise le même type d'équations que AF2, trop restreintes pour l'usage que nous voulons en faire.

Nous avons fait de nombreuses tentatives pour étendre ces systèmes de façon à pouvoir y gérer les exceptions. Il serait fastidieux de les décrire ici. Nous nous contenterons de donner, dans chacun des trois systèmes, un exemple de terme calculant le produit rapide et de montrer les difficultés qu'il y aurait à le typer.

1.5.2 Exemples de termes

En $\lambda^{\rightarrow}_{exn}$ calcul Le terme « naturel » pour le produit rapide des entiers d'un arbre est :

$t \stackrel{def}{=} \lambda a\, let\ y : \neg N\, in\ (a\ v\ u)\ handle\ (y\ x) \mapsto x\ tel$ avec $v = \lambda g\, \lambda d\, (P\ y\ x)$ et $u = \lambda n\, ((Z\ n)\ (raise\ (y\ 0))\ n)$ (le terme Z effectue le test à zéro d'un entier, et le terme P calcule le produit de deux entiers : voir page 99 pour une définition plus précise).

Il a le bon comportement. Soit d l'arbre $\lambda f\, \lambda x\, (f\ (f\ (x\ 1)\ (x\ 0))\ (x\ 2))$, alors $(d\ v\ u) \rightarrow \cdots (raise\ (y\, 0))$ et donc $(t\ d) \rightarrow let\ y : \neg N in (raise\ (y\, 0)) handle\ (y\ x) \mapsto x\ tel \rightarrow 0$.

Examinons son typage, dans une généralisation de $\lambda^{\rightarrow}_{exn}$ qui permettrait de traiter le type de données des entiers (N) et celui des arbres d'entiers (AN) - mais sans utiliser de termes d'individus.

Montrons que le terme t a, dans le contexte vide, le type $AN \rightarrow N$:

– $y : \neg N \vdash (raise\ (y\ 0)) : N$ donc $y : \neg N \vdash u : N \rightarrow N$, et $\vdash v : N, N \rightarrow N$, donc $a : AN, y : \neg N \vdash (a\ v\ u) : N$;

– on a aussi (2) $x : N \vdash x : N$;

– donc (typage de $handle$) $a : AN \vdash let\ y : \neg N in\ (a\ v\ u)\ handle\ (y\ x) \mapsto x\ tel : N$.

Mais les difficultés apparaissent lorsqu'on tente de passer à un typage avec termes d'individus.

En effet, si l'on veut typer $(y\ 0)$, il faut que y puisse avoir le type $\neg N[0]$. Et d'autre part il faut, à cause de (2), que x puisse avoir le type $N[f(a)]$ ($f(a)$ est le produit rapide des entiers de l'arbre a).

Il nous faudrait donc trouver un type A tel que $y : \neg A \vdash y : \neg N[0]$ et $x : A \vdash x : N[f(a)]$. Ce serait possible si on pouvait exprimer de façon formelle le raisonnement «lorsque $(raise\ (y\ 0))$ est utilisé c'est que l'arbre a possède une feuille nulle et alors $f(a) = 0$» car alors on pourrait choisir $A = N[f(a)]$.

Modifions le terme t pour obtenir le terme $t' \stackrel{def}{=} \lambda a\ : \neg N iny(ayvyu)$ $handle\ (y\ x) \mapsto 0\ tel$ avec le même sous-terme v que dans t et avec $u = \lambda n\,((Z\ n)\ (raise\ (y\ 3))\ n)$. Le terme t' calcule lui aussi le produit rapide, mais comment le typer?

1.5.3 En λ_c-calcul ou $\lambda\mu$-calcul

Un terme possible pour le produit rapide des entiers d'un arbre :

$t \stackrel{def}{=} \lambda a\,(\mathcal{CC}\ \lambda f\,(a\ v\ u))$ avec $v = \lambda g\,\lambda d\,(T\ g\ (T\ d\ \lambda x\,\lambda y\,(P\ y\ x)))$ et $u = \lambda n\,((Z\ n)\ (f\ 0)\ n)$.

En λ_c-calcul , on peut considérer que $\mathcal{CC}$ est une constante de type $((A \to B) \to A) \to A$ qui vérifie la règle de réduction $(\mathcal{CC}\ t\ t_1 \cdots t_n) \succ (t\ \lambda x\,\mathcal{A}(x\ t_1 \cdots t_n)\ t_1 \cdots t_n)$. où $\mathcal{A}$ est elle même une constante et vérifie la règle de réduction $(\mathcal{A}\ t\ t_1 \cdots t_n) \succ t$.

En $\lambda\mu$-calcul , $\mathcal{CC}$ et $\mathcal{A}$ sont deux termes qui ont été définis au paragraphe 1.3.2, ils vérifient les règles de réduction précédentes.

Le terme t a le bon comportement. En effet, avec les mêmes notations qu'au paragraphe 1.5.2, on a :

$(t\ d) \to (d\ v\ u')$ où $u' = u[f := \lambda x\,\mathcal{A}(x\ g)]$.

Or $(d\ v\ u') \to \cdots (\lambda x\,\mathcal{A}(x\ g)\ 0)$, donc $(t\ d) \to (\lambda x\,\mathcal{A}x\ 0) \to \mathcal{A}0 \to 0$.

Essayons de typer t. Dans le système de typage F, enrichi de la constante $\mathcal{CC}$ de type $((A \to B) \to A) \to A$, le terme t a, dans le contexte vide, le type $AN \to N$.

En effet : $f : N \to N \vdash u : N \to N$ et $\vdash v : N, N \to N$ donc, en remplaçant X par N dans $AN \stackrel{def}{=} \forall X\,((X, X \to X), (N \to X) \to X)$, on obtient $a : AN \vdash (a\ v\ u) : N$ soit $a : AN \vdash \lambda f\,(a\ v\ u) : (N \to N) \to N$ et donc $(\mathcal{CC}\ \lambda f\,(a\ v\ u)) : N$.

Mais le terme t' obtenu en remplaçant, dans t, le sous-terme v par $\lambda g\,\lambda d\,(P\ y\ x)$, aurait lui aussi le type $AN \to N$. Pourtant il n'a pas le bon fonctionnement. Comment justifier la nécessité des opérateurs de mise en mémoire?

D'autres difficultés adviennent lorsqu'on tente de typer t dans le système AF2 enrichi de $\mathcal{CC}$, toujours de type $((A \to B) \to A) \to A$.

Quel type donner à f? Nécessairement le type $N[0] \to N[?]$ si l'on veut pouvoir typer $(f\ 0)$, mais aussi le type $N[f(a)] \to N[?]$ si l'on veut que $(CC\ \lambda f\,(a\ v\ u))$ ait le type $N[f(a)]$.

Là aussi - comme pour le $\lambda_{\underset{exn}{\to}}$ calcul - le système devra permettre d'exprimer de façon formelle le raisonnement «lorsque $(f\ 0)$ est utilisé c'est que l'arbre a possède une feuille nulle et alors $f(a) = 0$» si on veut pouvoir donner un type à la variable f.

Le terme t peut s'écrire uniquement à l'aide de la constante C, puisqu'on sait qu'on peut choisir $CC = \lambda x\,(C\ \lambda y\,(y\ (x\ \lambda z\,\mathcal{A}\ (y\ z))))$ et $\mathcal{A} = \lambda y\,(C\ \lambda x\,y)$. Le terme obtenu est typable dans le système F enrichi de la constante C de type $\neg\neg X \to X$. Mais, dans le λ_c-calcul ou le $\lambda\mu$-calcul , il n'est pas plus facile à typer que t.

N'ayant pas pu utiliser les systèmes de λ-calcul précédents, nous proposons dans la suite un nouveau système de λ-calcul typé - $Ex2$ - construit comme une extension de AF2, et qui intègre un mécanisme de gestion des exceptions ce qui permet de construire des programmes prouvés plus efficaces.

1.6 Plan

Le reste du document est organisé de la façon suivante.

Le chapitre 2 décrit le λ-calcul pur et montre la confluence de la réduction.

Le chapitre 3 définit le système de typage: les types sont des formules d'un calcul des prédicats du second ordre avec constantes (de relations et de fonctions).

Le chapitre 4 montre la forte normalisation de $Ex2$ en la ramenant à celle d'un système de λ-calcul typé dérivé où les formules sont sans premier ordre.

Dans le chapitre 5 nous prouvons une propriété de conservation faible du type: la présence dans le système de typage d'une règle d'introduction de $\vee$ empêche la préservation du type par réduction, une forme de réduction uniforme doit être construite par laquelle le type sera préservé.

Dans le chapitre 6 nous montrons que $Ex2$ constitue un cadre satisfaisant pour la programmation. Si un terme est de type N, alors il se réduit à un entier de Church: le type N - par exemple - garantit donc qu'on ne peut obtenir une exception non capturée. Si les équations des fonctions à programmer sont consistantes sur leurs ensembles de définition, alors un terme qui a le bon type constitue bien un bon programme c'est à dire calcule bien la fonction dont il a le type.

Dans le chapitre 7 nous relions réduction des λ-termes et réduction des preuves, et nous montrons que la formule $\neg\neg X \to X$ n'est pas démontrable dans $Ex2$.

Le chapitre 8 décrit quelques exemples de fonctions faisant usage des exceptions et donne des termes de $Ex2$ qui constituent des programmes (prouvés) pour ces fonctions.

Chapitre 2

Le λ-calcul pur

Nous allons contruire un nouveau λ-calcul qui nous permettra d'utiliser des exceptions (d'entiers, de booléens ...), et que nous nommerons pour cette raison $Ex2$. Dans ce chapitre, nous définirons successivement la construction des termes et leur réduction, et nous montrerons la confluence de cette réduction.

2.1 Les termes

En plus des termes du λ-calcul ordinaire, nous utilisons de nouvelles constructions dans le but de représenter les exceptions.

2.1.1 Dénominations de type

Chaque algèbre de termes peut être nommée par un symbole que nous appellerons symbole de données. Par exemple, N et B sont les symboles de données associés respectivement aux algèbres de termes des entiers et des booléens. Nous écrirons D pour désigner un symbole de données quelconque.

Nous appellerons **dénomination de type** un symbole de données indicé, comme N_1 ou B_2 et nous considérerons que les deux dénominations de type N_1 et N_2 sont différentes.

Nous tolèrerons enfin de ne pas noter l'indice d'une dénomination de type si, dans les termes utilisés, il n'y a qu'une seule dénomination de type associée à un symbole de données particulier.

2.1.2 Termes

Définition 2.1.1 *On dispose d'un ensemble dénombrable de variables.*

La syntaxe des éléments de l'ensemble T est donnée par la grammaire suivante, où x désigne une variable, et α une dénomination de type :

$T ::= x \mid \lambda x\, T \mid (T\ T) \mid \varepsilon_\alpha\langle T\rangle \mid \tau_\alpha\langle T, T\rangle$. *L'ensemble Λ_E des termes de $Ex2$ est le quotient de l'ensemble T par la relation d'α-équivalence ([10]).*

S'il n'y a pas d'ambiguïté, on notera aussi $u\ v$ le terme $(u\ v)$, et $u\ t_1 \cdots t_n$ le terme $((u\ t_1)\ \cdots t_n)$.

2.2 Réduction

On définit, sur Λ_E, une relation de réduction notée $\boldsymbol{\delta}$ ou $\rightarrow_\delta$.

Définition 2.2.1 *On dit qu'un terme est un redex s'il est de l'une des formes suivantes, et on définit alors son contracté :*

- $(\lambda x\ u\ v)$ *de contracté* $u[x:=v]$.
- $(\varepsilon_\alpha\langle u\rangle\ v)$ *de contracté* $\varepsilon_\alpha\langle u\rangle$
- $\tau_\alpha\langle\lambda x\ u, v\rangle$ *de contracté* $\lambda x\ u$
- $\tau_\alpha\langle\varepsilon_\alpha\langle u\rangle, v\rangle$ *de contracté* $(v\ u)$
- $\tau_\alpha\langle\varepsilon_\beta\langle u\rangle, v\rangle$ *de contracté* $\varepsilon_\beta\langle u\rangle$ *(si* $\alpha \neq \beta$*)*

On peut vérifier que, si deux redex sont α*-équivalents, leurs contractés le sont aussi. La notion de contracté est donc bien définie sur* Λ_E.

On définit ensuite une relation binaire $\boldsymbol{\delta_0}$ (notée aussi $\rightarrow_{\delta_0}$) sur Λ_E : on aura $t\ \boldsymbol{\delta_0}\ t'$ si t' est obtenu en contractant un redex de t. On peut donner de $\boldsymbol{\delta_0}$ une définition plus opérationnelle.

Définition 2.2.2 *Si t et t' sont éléments de* Λ_E, *on définit* $t\ \boldsymbol{\delta_0}\ t'$ *par induction sur la complexité de t :*

- *si t est une variable,* $t\ \boldsymbol{\delta_0}\ t'$ *est faux*
- *si* $t = \lambda x\ u$, *alors* $t\ \boldsymbol{\delta_0}\ t'$ *ssi* $t' = \lambda x\ u'$ *et* $u\ \boldsymbol{\delta_0}\ u'$
- *si* $t = \varepsilon_\alpha\langle u\rangle$, *alors* $t\ \boldsymbol{\delta_0}\ t'$ *ssi* $t' = \varepsilon_\alpha\langle u'\rangle$ *et* $u\ \boldsymbol{\delta_0}\ u'$
- *si* $t = (u\ v)$ *alors* $t\ \boldsymbol{\delta_0}\ t'$ *ssi*
  - *ou* $t' = (u'\ v)$ *avec* $u\ \boldsymbol{\delta_0}\ u'$
  - *ou* $t' = (u\ v')$ *avec* $v\ \boldsymbol{\delta_0}\ v'$
  - *ou* $u = \lambda x\ w$ *et* $t' = w[x:=v]$
  - *ou* $u = \varepsilon_\alpha\langle w\rangle$ *et* $t' = \varepsilon_\alpha\langle w\rangle$
- *si* $t = \tau_\alpha\langle u, v\rangle$ *alors* $t\ \boldsymbol{\delta_0}\ t'$ *ssi*
  - *ou* $t' = \tau_\alpha\langle u', v\rangle$ *avec* $u\ \boldsymbol{\delta_0}\ u'$
  - *ou* $t' = \tau_\alpha\langle u, v'\rangle$ *avec* $v\ \boldsymbol{\delta_0}\ v'$

- *ou* $u = \lambda x\, w$ *et* $t' = \lambda x\, w$
- *ou* $u = \varepsilon_\alpha\langle w\rangle$ *et* $t' = (v\ w)$
- *ou* $u = \varepsilon_\beta\langle w\rangle$ *avec* $\beta \neq \alpha$ *et* $t' = \varepsilon_\beta\langle w\rangle$

On définit ensuite la relation $\boldsymbol{\delta}$, *notée aussi* $\rightarrow_\delta$, *(*δ*-réduction) comme la plus petite relation binaire réflexive et transitive contenant* $\boldsymbol{\delta_0}$, *et la* δ*-équivalence (notée* $\simeq_\delta$*) comme la plus petite relation d'équivalence contenant* $\boldsymbol{\delta_0}$.

On peut caractériser assez simplement la δ-réduction , après avoir rappelé ce que veut dire passer au contexte pour une relation sur Λ_E.

Définition 2.2.3 *Une relation binaire* R *sur* Λ_E *passe au contexte si elle est réflexive et si on a :*

- $t\, R\, t' \Rightarrow \lambda x\, t\, R\, \lambda x\, t'$
- $t\, R\, t' \Rightarrow \varepsilon_\alpha\langle t\rangle\, R\, \varepsilon_\alpha\langle t'\rangle$
- $u\, R\, u', v\, R\, v' \Rightarrow (u\ v)\, R\, (u'\ v')$
- $u\, R\, u', v\, R\, v' \Rightarrow \tau_\alpha\langle u, v\rangle\, R\, \tau_\alpha\langle u', v'\rangle$

Lemme 2.2.4 *La* δ*-réduction est la plus petite relation binaire transitive* R *qui passe au contexte et telle que, quels que soient les termes* u *et* v*, quelles que soient la variable* x *et les dénominations de type (différentes)* α *et* β*, on ait :*

- $(\lambda x\, u\ v)\ R\ \ u[x := v]$
- $(\varepsilon_\alpha\langle u\rangle\ v)\, R\, \varepsilon_\alpha\langle u\rangle$
- $\tau_\alpha\langle \lambda x\, u, v\rangle\, R\, \lambda x\, u$
- $\tau_\alpha\langle \varepsilon_\beta\langle u\rangle, v\rangle\, R\, \varepsilon_\beta\langle u\rangle$
- $\tau_\alpha\langle \varepsilon_\alpha\langle u\rangle, v\rangle\, R\, (v\ u)$.

Preuve Il est clair que la δ-réduction passe au contexte. Inversement, soit R une relation binaire qui passe au contexte et telle qu'on ait les réductions citées dans l'énoncé du lemme. On montre facilement, par induction sur t, que $t\, \boldsymbol{\delta_0}\, t' \Rightarrow t\, R\, t'$. La relation R contient donc $\boldsymbol{\delta_0}$, mais alors R qui est transitive contient $\boldsymbol{\delta}$ puisque $\boldsymbol{\delta}$ est la plus petite relation transitive contenant $\boldsymbol{\delta_0}$. □

Définition 2.2.5 *Un terme est dit normal s'il ne contient aucun redex.*

Nous préciserons plus loin la forme générale d'un terme normal.

2.3 Confluence de la réduction

Nous montrons l'équivalent du théorème de Church-Rosser pour $Ex2$.

Théorème 2.3.1 *La δ-réduction est confluente.*

On en déduit que les termes normalisables ont une forme normale unique.

La δ-réduction est un système de réécriture. Au prix de quelques reformulations des règles, on peut faire de ce système un CRS (combinatory reduction system) orthogonal et en déduire la confluence. Cette méthode de preuve est présentée dans l'annexe B, mais nous avons préféré donner, ci-dessous, une preuve plus classique de la confluence de la δ-réduction, en utilisant la technique des réductions parallèles.

Pour prouver le théorème nous allons construire une relation $\boldsymbol{\rho}$ dont nous montrerons successivement qu'elle est confluente et que sa cloture transitive est la relation de δ-réduction . Cela suffira à prouver notre théorème puisque nous savons que si une relation binaire est confluente, alors sa cloture transitive l'est aussi ([10]).

2.3.1 Une autre réduction

Définition 2.3.2 *Les éléments de $\boldsymbol{\rho}$ sont les couples de termes obtenus en appliquant les règles suivantes et elles seules :*

réflexivité : $t \,\boldsymbol{\rho}\, t$

$\boldsymbol{\rho}_c$ *(ρ passe au contexte) :*

1. $u \,\boldsymbol{\rho}\, u' \Rightarrow \lambda x\, u \,\boldsymbol{\rho}\, \lambda x\, u'$
2. $u \,\boldsymbol{\rho}\, u' \Rightarrow \varepsilon_\alpha\langle u\rangle \,\boldsymbol{\rho}\, \varepsilon_\alpha\langle u'\rangle$
3. $u \,\boldsymbol{\rho}\, u', v \,\boldsymbol{\rho}\, v' \Rightarrow (u\ v) \,\boldsymbol{\rho}\, (u'\ v')$
4. $u \,\boldsymbol{\rho}\, u', v \,\boldsymbol{\rho}\, v' \Rightarrow \tau_\alpha\langle u, v\rangle \,\boldsymbol{\rho}\, \tau_\alpha\langle u', v'\rangle$

$\boldsymbol{\rho}_\lambda$: $u \,\boldsymbol{\rho}\, u', v \,\boldsymbol{\rho}\, v' \Rightarrow (\lambda x\, u\ v) \,\boldsymbol{\rho}\, u'[x := v']$

$\boldsymbol{\rho}_\varepsilon$: $u \,\boldsymbol{\rho}\, u' \Rightarrow (\varepsilon_\alpha\langle u\rangle\ v) \,\boldsymbol{\rho}\, \varepsilon_\alpha\langle u'\rangle$

$\boldsymbol{\rho}_\tau$: *si* $u \,\boldsymbol{\rho}\, u', v \,\boldsymbol{\rho}\, v'$ *alors :*

1. $\tau_\alpha\langle\varepsilon_\alpha\langle u\rangle, v\rangle \,\boldsymbol{\rho}\, (v'\ u')$

2. $\tau_\alpha\langle\varepsilon_\beta\langle u\rangle, v\rangle \; \boldsymbol{\rho} \; \varepsilon_\beta\langle u'\rangle$ *si* $\alpha \neq \beta$
3. $\tau_\alpha\langle\lambda x\, u, v\rangle \; \boldsymbol{\rho} \; \lambda x\, u'$

Le premier résultat est presqu'immédiat.

Lemme 2.3.3 *La cloture transitive de* $\boldsymbol{\rho}$ *(*$\boldsymbol{\rho}^*$*) est égale à la* δ*-réduction .*

Preuve

- On montre $t \to_\rho t' \Rightarrow t \to_\delta t'$ par récurrence sur la complexité de t, en distinguant selon la dernière règle appliquée.
  - Si c'est la règle λ, on a $u \to_\rho u', v \to_\rho v'$ donc par hypothèse de récurrence $u \to_\delta u', v \to_\delta v'$ et donc (passage au contexte) $\lambda x\, u \to_\delta \lambda x\, u'$ puis $(\lambda x\, u\, v) \to_\delta (\lambda x\, u'\, v')$. Le lemme 2.2.4 nous donne $(\lambda x\, u'\, v') \to_\delta u'[x := v']$ et permet de conclure.
  - Si c'est la règle ϵ, on a $u \to_\rho u'$ donc par hypothèse de récurrence $\varepsilon_\alpha\langle u\rangle \to_\rho \varepsilon_\alpha\langle u'\rangle$ puis $(\varepsilon_\alpha\langle u\rangle\; v) \to_\delta (\varepsilon_\alpha\langle u'\rangle\; v) \to_\delta \varepsilon_\alpha\langle u'\rangle$.
  - Si c'est la règle τ, on a $u \to_\rho u', v \to_\rho v'$ et, dans le premier cas, par passage au contexte, $\tau_\alpha\langle\varepsilon_\alpha\langle u\rangle, v\rangle \to_\delta \tau_\alpha\langle\varepsilon_\alpha\langle u'\rangle, v'\rangle$, puis (lemme 2.2.4) $\tau_\alpha\langle\varepsilon_\alpha\langle u'\rangle, v'\rangle \to_\delta (v'\; u')$ (les deux autres cas ne sont pas traités car très semblables).

    De $\boldsymbol{\rho} \subset \boldsymbol{\delta}$ on déduit $\boldsymbol{\rho}^* \subset \boldsymbol{\delta}^* = \boldsymbol{\delta}$.
- On montre $t \to_{\delta_0} t' \Rightarrow t \to_\rho t'$ par récurrence sur la complexité de t. Il suffit d'examiner les différents cas de la définition 2.2.2. Par exemple si $t = (\lambda x\, u\, v)$ et $t' = u[x := v]$, on applique la règle $\boldsymbol{\rho}_\lambda$ à $u \to_\rho u$ et $v \to_\rho v$.

  De $\boldsymbol{\delta_0} \subset \boldsymbol{\rho}$ on déduit $\boldsymbol{\delta_0}^* \subset \boldsymbol{\rho}^*$, et donc $\boldsymbol{\delta} \subset \boldsymbol{\rho}^*$ puisque $\boldsymbol{\delta}$ est la cloture réflexive de $\boldsymbol{\delta_0}^*$ et $\boldsymbol{\rho}^*$ est réflexive.

□

Quelques lemmes sont nécessaires avant de pouvoir montrer la confluence de la relation $\boldsymbol{\rho}$.

2.3.2 Lemmes

Le premier lemme permet de donner la forme de t', réduit de t par la relation $\boldsymbol{\rho}$, selon celle de t.

Lemme 2.3.4 *Soient deux termes t et t' tels que* $t \to_\rho t'$ *:*

- *si* $t = x$ *alors* $t' = x$
- *si* $t = \lambda x\, u$ *alors* $t' = \lambda x\, u'$ *et* $u \to_\rho u'$

- *si* $t = (u\ v)$ *alors*
  - *soit* $t' = (u'\ v')$ *et* $u \to_\rho u', v \to_\rho v'$
  - *soit* $t' = w'[x:=v']$ *et* $u = \lambda x\, w$ *avec* $v \to_\rho v', w \to_\rho w'$
  - *soit* $t' = \varepsilon_\alpha\langle w'\rangle$ *et* $u = \varepsilon_\alpha\langle w\rangle$ *avec* $w \to_\rho w'$
- *si* $t = \varepsilon_\alpha\langle u\rangle$ *alors* $t' = \varepsilon_\alpha\langle u'\rangle$ *et* $u \to_\rho u'$
- *si* $t = \tau_\alpha\langle u, v\rangle$ *alors*
  - *soit* $t' = \tau_\alpha\langle u', v'\rangle$ *et* $u \to_\rho u', v \to_\rho v'$
  - *soit* $t' = \lambda x\, w'$ *et* $u = \lambda x\, w$ *avec* $w \to_\rho w'$
  - *soit* $t' = \varepsilon_\beta\langle w'\rangle$ *et* $u = \varepsilon_\beta\langle w\rangle$ *avec* $w \to_\rho w'$ *et* $\alpha \neq \beta$
  - *soit* $t' = (v'\ w')$ *et* $u = \varepsilon_\alpha\langle w\rangle$ *avec* $v \to_\rho v', w \to_\rho w'$

Preuve Le lemme se démontre sans difficulté par récurrence sur le nombre de règles utilisées dans la réduction $t\ \boldsymbol{\rho}\ t'$, en considérant la dernière règle. □

Lemme 2.3.5 *Si* $u \to_\rho u'$ *et* $v \to_\rho v'$ *alors* $u[x:=v] \to_\rho u'[x:=v']$

Preuve Le lemme se démontre par récurrence sur la longueur de la réduction $u \to_\rho u'$, on distingue selon la dernière règle utilisée.

- Si $u = u'$, on montre le résultat immédiat $u[x:=v] \to_\rho u[x:=v']$ par récurrence sur la complexité de u.
- Si c'est le passage au contexte :
  - dans le cas 1, on a $u = \lambda y\, w$ et $u' = \lambda y\, w'$ avec $w \to_\rho w'$; de $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x:=v] \to_\rho w'[x:=v']$ et donc $(\boldsymbol{\rho}_{c_1})\ \lambda y\, w[x:=v] \to_\rho \lambda y\, w'[x:=v']$ soit $u[x:=v] \to_\rho u'[x:=v']$
  - dans le cas 2, on a $u = \varepsilon_\alpha\langle w\rangle$ et $u' = \varepsilon_\alpha\langle w'\rangle$ avec $w \to_\rho w'$; de $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x:=v] \to_\rho w'[x:=v']$ et donc $(\boldsymbol{\rho}_{c_2})\ \varepsilon_\alpha\langle w[x:=v]\rangle \to_\rho \varepsilon_\alpha\langle w'[x:=v']\rangle$ soit $u[x:=v] \to_\rho u'[x:=v']$
  - dans le cas 3, on a $u = (w\ t)$ et $u' = (w'\ t')$ avec $w \to_\rho w'$ et $t \to_\rho t'$; de $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x:=v] \to_\rho w'[x:=v']$ et $t[x:=v] \to_\rho t'[x:=v']$ et donc $(\boldsymbol{\rho}_{c_3})\ (w[x:=v]\ t[x:=v]) \to_\rho (w'[x:=v']\ t'[x:=v'])$ soit $u[x:=v] \to_\rho u'[x:=v']$
  - dans le cas 4, on a $u = \tau_\alpha\langle w, t\rangle$ et $u' = \tau_\alpha\langle w', t'\rangle$ avec $w \to_\rho w'$ et $t \to_\rho t'$; de $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x:=v] \to_\rho w'[x:=v']$ et $t[x:=v] \to_\rho t'[x:=v']$ et donc $(\boldsymbol{\rho}_{c_4})\ \tau_\alpha\langle w[x:=v], t[x:=v]\rangle \to_\rho \tau_\alpha\langle w'[x:=v'], t'[x:=v']\rangle$ soit $u[x:=v] \to_\rho u'[x:=v']$.

- Si c'est la règle λ, on a $u = (\lambda y\, w\ t)$ et $u' = w'[y := t']$ avec $w \to_\rho w'$ et $t \to_\rho t'$. De $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x := v] \to_\rho w'[x := v']$ et $t[x := v] \to_\rho t'[x := v']$ donc $(\boldsymbol{\rho}_\lambda)$ $(\lambda y\, w[x := v]\, t[x := v]) \to_\rho w'[x := v'][y := t'[x := v']]$ soit $u[x := v] \to_\rho u'[x := v']$, puisque $\lambda y\, w[x := v] = (\lambda y\, w)[x := v]$ et $w'[x := v'][y := t'[x := v']] = w'[y := t'][x := v']$.

- Si c'est la règle ϵ, on a $u = (\varepsilon_\alpha\langle w\rangle\ t)$ et $u' = \varepsilon_\alpha\langle w'\rangle$ avec $w \to_\rho w'$. De $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x := v] \to_\rho w'[x := v']$ donc $(\boldsymbol{\rho}_\epsilon)$ $u = (\varepsilon_\alpha\langle w[x := v]\rangle\ t[x := v])$ et $u' = \varepsilon_\alpha\langle w'[x := v']\rangle$, soit $u[x := v] \to_\rho u'[x := v']$.

- Si c'est la règle τ, si $u \to_\rho u'$, $v \to_\rho v'$ alors :

  - dans le cas 1, on a $u = \tau_\alpha\langle\varepsilon_\alpha\langle w\rangle, t\rangle$ et $u' = (t'\ w')$ avec $w \to_\rho w'$ et $w \to_\rho w'$; de $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x := v] \to_\rho w'[x := v']$ et $t[x := v] \to_\rho t'[x := v']$ donc $(\boldsymbol{\rho}_{\tau_1})$ $\tau_\alpha\langle\varepsilon_\alpha\langle w[x := v]\rangle, t[x := v]\rangle \to_\rho (t'[x := v']\ w'[x := v'])$
  - dans le cas 2, on a $u = \tau_\alpha\langle\varepsilon_\beta\langle w\rangle, t\rangle$ et $u' = \varepsilon_\beta\langle w'\rangle$ avec $w \to_\rho w'$ et $\alpha \neq \beta$; de $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x := v] \to_\rho w'[x := v']$ donc $(\boldsymbol{\rho}_{\tau_2})$ $\tau_\alpha\langle\varepsilon_\beta\langle w[x := v]\rangle, t[x := v]\rangle \to_\rho \varepsilon_\beta\langle w'[x := v']\rangle$
  - dans le cas 3, on a $u = \tau_\alpha\langle\lambda y\, w, t\rangle$ et $u' = \lambda y\, w'$ avec $w \to_\rho w'$; de $v \to_\rho v'$ on déduit par hypothèse de récurrence $w[x := v] \to_\rho w'[x := v']$ donc $(\boldsymbol{\rho}_{\tau_3})$ $\tau_\alpha\langle\lambda y\, w[x := v], t[x := v]\rangle \to_\rho \lambda y\, w'[x := v']$

□

2.3.3 Confluence de $\boldsymbol{\rho}$

Lemme 2.3.6 *La relation* $\boldsymbol{\rho}$ *est confluente.*

Preuve Partant de $t \to_\rho t'$ et $t \to_\rho t''$ nous cherchons T tel que $t' \to_\rho T$ et $t'' \to_\rho T$, la preuve de l'existence de T est faite par récurrence sur la complexité de t (on utilise abondamment le lemme 2.3.4).

- Si t est de longueur 1, c'est une variable donc $t = t' = t''$.

- Si $t = \lambda x\, u$ alors $t' = \lambda x\, u'$ et $t'' = \lambda x\, u''$ avec $u \to_\rho u'$ et $u \to_\rho u''$ par hypothèse de récurrence il existe un terme U tel que $u' \to_\rho U$ et $u'' \to_\rho U$ et il suffit de choisir $T = \lambda x\, U$.

- Si $t = (u\ v)$, distinguons plusieurs cas.

  - Soit $t' = (u'\ v')$ et $t'' = (u''\ v'')$ avec $u \to_\rho u'$,$u \to_\rho u''$, $v \to_\rho v'$,$v \to_\rho v''$; par hypothèse de récurrence , il existe U et V tels que $u' \to_\rho U$, $u'' \to_\rho U$, $v' \to_\rho V$, $v'' \to_\rho V$ et il suffit de choisir $T = (U\ V)$.

- Soit $u = \lambda x\, w$ et $t' = (u'\ v')$ et $t'' = w''[x:=v'']$ $t' = (u'\ v')$ et $t'' = (u''\ v'')$ avec $u \to_\rho u', w \to_\rho w''$. Mais si $u = \lambda x\, w$ et $u \to_\rho u'$ alors $u' = \lambda x\, w'$ avec $w \to_\rho w'$ et donc $t' = (\lambda x\, w'\ v')$. Par hypothèse de récurrence , il existe U et W tels que $u' \to_\rho U$, $u'' \to_\rho U$, $w' \to_\rho W$, $w'' \to_\rho W$; on a donc $(\boldsymbol{\rho})\ (u'\ v') \to_\rho W[x:=V]$ et de même $w''[x:=v''] \to_\rho W[x:=V]$, il suffit donc de choisir $T = W[x:=V]$.
- Soit $u = \varepsilon_\alpha\langle w\rangle$ et $t' = (\varepsilon_\alpha\langle w'\rangle\ v')$ et $t'' = \varepsilon_\alpha\langle w"\rangle$ et ce cas se traite comme le précédent.
- Soit $u = \lambda x\, w$ et $t' = w'[x:=v']$ et $t'' = w''[x:=v'']$, et ce cas se traite comme le suivant.
- Soit $u = \varepsilon_\alpha\langle w\rangle$ et $t' = \varepsilon_\alpha\langle w'\rangle$ et $t'' = \varepsilon_\alpha\langle w"\rangle$ avec $w \to_\rho w'$et $w \to_\rho w''$; par hypothèse de récurrence , il existe W tel que $w' \to_\rho W$ et $w'' \to_\rho W$ et donc il suffit de choisir $T = \varepsilon_\alpha\langle W\rangle$.

- Le cas $t = \varepsilon_\alpha\langle u\rangle$ se traite comme le cas $t = \lambda x\, u$.

- Si $t = \tau_\alpha\langle u, v\rangle$ nous devons aussi distinguer plusieurs cas.
  - Si $t' = \tau_\alpha\langle u', v'\rangle$ et $t'' = \tau_\alpha\langle u'', v''\rangle$ alors le raisonnement est le même que dans le cas $t' = (u'\ v')$ et $t'' = (u''\ v'')$.
  - Si $u = \lambda x\, w$, nous devons examiner deux cas :
    - ou $t' = \tau_\alpha\langle \lambda x\, w', v'\rangle$ et $t'' = \lambda x\, w"$ avec $w \to_\rho w', w \to_\rho w'', v \to_\rho v'$; par hypothèse de récurrence , il existe W tel que $w' \to_\rho W$ et $w'' \to_\rho W$, on a donc $\tau_\alpha\langle \lambda x\, w', v'\rangle \to_\rho \lambda x\, W$ et $\lambda x\, w" \to_\rho \lambda x\, W$, il suffit donc de choisir $T = \lambda x\, W$;
    - ou $t' = lbxw'$ et $t'' = \lambda x\, w"$ avec $w \to_\rho w'$et $w \to_\rho w''$ par hypothèse de récurrence , il existe W tel que $w' \to_\rho W$ et $w'' \to_\rho W$, on a donc $\lambda x\, w' \to_\rho \lambda x\, W$ et $\lambda x\, w" \to_\rho \lambda x\, W$, il suffit donc de choisir $T = \lambda x\, W$.
  - Si $u = \varepsilon_\beta\langle w\rangle$, les deux cas se traitent comme les deux précédents.
  - Si $u = \varepsilon_\alpha\langle w\rangle$, il y a aussi deux cas à examiner :
    - ou $t' = \tau_\alpha\langle \varepsilon_\alpha\langle w'\rangle, v'\rangle$ et $t'' = (v"\ \varepsilon_\alpha\langle w''\rangle)$ avec $w \to_\rho w', w \to_\rho w''$, $v \to_\rho v'$, $v \to_\rho v''$; par hypothèse de récurrence , il existe V et W tels que $v' \to_\rho V$,$v'' \to_\rho V$, $w' \to_\rho W$, $w'' \to_\rho W$, on a $\tau_\alpha\langle \varepsilon_\alpha\langle w'\rangle, v'\rangle \to_\rho (V\ W)$ et $(v"\ \varepsilon_\alpha\langle w''\rangle) \to_\rho (V\ W)$, il suffit donc de choisir $T = (V\ W)$;
    - ou $t' = (v'\ \varepsilon_\alpha\langle w'\rangle)$ et $t'' = (v"\ \varepsilon_\alpha\langle w''\rangle)$ avec $w \to_\rho w', w \to_\rho w'', v \to_\rho v'$, $v \to_\rho v''$; par hypothèse de récurrence , il existe V et W tels que $v' \to_\rho V$,$v'' \to_\rho V$, $w' \to_\rho W$, $w'' \to_\rho W$, on a $(v'\ \varepsilon_\alpha\langle w'\rangle) \to_\rho (V\ W)$ et $(v"\ \varepsilon_\alpha\langle w''\rangle) \to_\rho (V\ W)$, il suffit donc de choisir $T = (V\ W)$.

□

2.4 Forme des termes

Nous introduisons une notation qui va nous permettre de regrouper d'une part les termes $\lambda x\, t$ et $\varepsilon_\alpha\langle t\rangle$, d'autre part les termes $\tau_\alpha\langle u, v\rangle$ et (u v).

Notation 2.4.1 *Nous noterons $\overrightarrow{\lambda\epsilon}$ une suite (éventuellement vide) de λ ou de ε. Ainsi, par exemple, le terme $\lambda x_1\, \lambda x_2\, \varepsilon_{\alpha_3}\langle\varepsilon_{\alpha_4}\langle\lambda x_5\, t\rangle\rangle$ sera dit de la forme $\overrightarrow{\lambda\epsilon}\, t$.*

Les termes $(u\ v)$ et $\tau_\alpha\langle u, v\rangle$ seront eux respectivement notés $\lfloor A, u, v\rfloor$ et $\lfloor T_\alpha, u, v\rfloor$.

Nous pouvons alors trouver une forme générale pour les termes.

Lemme 2.4.2 *Tout terme s'écrit d'une façon et d'une seule sous la forme :*

$$\overrightarrow{\lambda\epsilon}\ \lfloor X_n, \lfloor X_{n-1}, \cdots \lfloor X_1, r, t_1\rfloor \cdots, t_{n-1}\rfloor, t_n\rfloor$$

où n est un entier éventuellement nul, et où, pour tout i tel que $1 \leq i \leq n$, X_i est soit A soit T_α (pour une certaine dénomination de type α), et t_i un terme, où enfin r est soit une variable soit un redex.

Preuve Le lemme se prouve par récurrence sur la complexité du terme t :

- si t est une variable, il est bien de la forme voulue
- si t est $\lambda x\, u$ ou $\varepsilon_\alpha\langle u\rangle$, il suffit d'appliquer l'hypothèse de récurrence
- si t est $(u\ v)$
  - si u est $\lambda y\, w$ ou $\varepsilon_\alpha\langle w\rangle$ alors t est un redex donc de la forme souhaitée
  - sinon on applique l'hypothèse de récurrence à u qui est donc égal à $\lfloor X_n, \cdots \lfloor X_1, r, t_1\rfloor \cdots, t_n\rfloor$ et ainsi t est $\lfloor A, \lfloor X_n, \cdots \lfloor X_1, r, t_1\rfloor \cdots, t_n\rfloor, v\rfloor$
- si t est $\tau_\alpha\langle u, v\rangle$ alors
  - si u est $\lambda y\, w$ ou $\varepsilon_\beta\langle w\rangle$ avec α égal ou non à β alors t est un redex donc de la forme souhaitée
  - sinon par hypothèse de récurrence $u = \lfloor X_n, \cdots \lfloor X_1, r, t_1\rfloor \cdots, t_n\rfloor$ et donc $t = \lfloor T_\alpha, \lfloor X_n, \cdots \lfloor X_1, r, t_1\rfloor \cdots, t_n\rfloor, v\rfloor$

□

Et nous en déduisons, trivialement, la forme des termes normaux.

Lemme 2.4.3 *Un terme est normal si et seulement s'il est de la forme :*

$$\overrightarrow{\lambda\epsilon}\lfloor X_n, \lfloor X_{n-1}, \cdots \lfloor X_1, x, t_1\rfloor \cdots, t_{n-1}\rfloor, t_n\rfloor$$

où x est une variable et où, pour tout i tel que $1 \leq i \leq n$, X_i est soit A soit T_α (pour une certaine dénomination de type α), et t_i un terme normal.

Chapitre 3

Le système de typage

Nous poursuivons la construction du λ-calcul $Ex2$ en décrivant dans ce chapitre son système de typage, système qui peut être vu comme une extension du système AF2 (arithmétique fonctionnelle du second ordre) de Krivine ([10]) et Leivant ([14]).

3.1 Les types

Nous avons tout d'abord à décrire un langage du second ordre dont les formules seront les types de notre système de typage.

3.1.1 Les termes d'individus

Définition 3.1.1 *Tout langage comprendra :*

- *des symboles qui seront utilisés pour exprimer les termes des algèbres de termes (pour les algèbres de termes N et B que nous utiliserons comme exemples dans la suite, nous utiliserons donc les symboles $\mathbf{0}, \boldsymbol{v}, \boldsymbol{f}$ 0-aires et $\boldsymbol{s}$ unaire);*

- *des symboles qui seront utilisés pour exprimer les exceptions: r_α, t_α, cas_α respectivement unaire, binaire, ternaire (pour chaque dénomination de type α).*

Ces deux premières catégories de symboles de fonctions constituent la partie commune à tous les langages ; pouront y être joints des symboles de fonctions particuliers (pour représenter les fonctions sur lesquelles on souhaite travailler).

Les termes d'individus sont définis au moyen des deux règles :

- *chaque variable d'individu, chaque symbole de constante (symbole de fonction 0-aire) est un terme*

– *si f est un symbole de fonction n-aire et si $t_1, \cdots, t_n$ sont des termes d'individu, alors $f(t_1, \cdots, t_n)$ est un terme.*

Ainsi nous écrirons, par exemple, $cas_\alpha(r_\alpha(\boldsymbol{s}(\mathbf{0})), \boldsymbol{v}, \boldsymbol{f})$.

Remarque 3.1.2 *Dire que le symbole t_α est binaire n'est pas tout à fait exact. On devrait dire que $t \stackrel{def}{=} t_\alpha(u,(x,v))$ est un terme si x est une variable d'individu et u et v sont des termes d'individu. En procédant ainsi, on n'a pas, stricto sensu, défini un langage du premier ordre, car le deuxième argument de t_α est défini à renommage près : $t_\alpha(u,(x,v))$ et $t_\alpha(u,(y,v[x:=y]))$ sont considérés comme égaux. On pourrait utiliser pour t la notation $t_\alpha(u, \lambda x\, v)$, à condition de bien noter que ce n'est pas une écriture du second ordre puisqu'on ne peut évidemment pas écrire $t_\alpha(u,(\lambda x\, v\ w))$.*

En pratique nous écrirons $t_\alpha(u, x \to v)$ ou $t_\alpha(u, x \to f(x))$ ou même $t_\alpha(u, f)$. Et, dans un souci de concision, nous écrirons $t_\alpha(t)$ pour $t_\alpha(t, x \to x)$.

3.1.2 Les formules

Nous utilisons des symboles de relation spécifiques : pour dénoter le type des exceptions.

Définition 3.1.3 *Tout langage comprendra les symboles de relation suivants et eux seulement:*

– *pour chaque dénomination de type α un symbole de relation unaire E^α*

– *pour chaque couple formé d'un ensemble (éventuellement vide) $L \stackrel{def}{=} \alpha_1, \cdots, \alpha_n$ de dénominations de type , et d'un symbole de données D, un symbole de relation unaire D^L*

L'ensemble de formules $\mathcal{A}$ s'obtient par application d'une des règles:

– *si X est une variable de relation n-aire, $t_1, \cdots, t_n$ des termes, alors $X(t_1, \cdots, t_n) \in \mathcal{A}$ (si n=1, on notera indifféremment X(t) ou Xt)*

– *si B est un symbole de relation (unaire donc), t un terme, alors $B[t] \in \mathcal{A}$*

– *si F et G sont éléments de $\mathcal{A}$, alors $(F \to G) \in \mathcal{A}$*

– *si F est élément de $\mathcal{A}$, alors $\forall x F$ et $\forall X F$ sont éléments de $\mathcal{A}$ (si x est une variable d'individu, X une variable de relation).*

On définit une relation d'équivalence sur $\mathcal{A}$ en mettant en relation chaque formule $D^\emptyset[x]$, pour tout symbole de données D, avec la formule définissant dans AF2 le type de données correspondant à D. Ainsi, par exemple, on a

pour tout terme t *:* $N^{\emptyset}[t] \simeq N[t]$ *et* $B^{\emptyset}[t] \simeq B[t]$ *où* $B[x] \stackrel{def}{=} \forall X\,[X\,\boldsymbol{v}, X\,\boldsymbol{f} \to X\,x]$ *et* $N[x] \stackrel{def}{=} \forall X\,(\forall y(X\,y \to X\,\boldsymbol{s}(y)), X\,\boldsymbol{0} \to X\,x)$.

L'ensemble Φ_E *des formules du système* $Ex2$ *est le quotient de l'ensemble* $\mathcal{A}$ *par la relation d'équivalence précédente.*

Mais, en l'absence d'ambiguïté, nous noterons indifféremment: $N^{\emptyset}[t]$ ou $N[t]$.

Nous abrègerons éventuellement $A_1 \to (A_2 \to (\cdots (A_k \to B)\cdots))$ en $A_1, A_2, \cdots, A_k \to B$.

Lemme 3.1.4 *Toute formule* F *de* Φ_E *s'écrit :* $\forall v_1 \cdots v_n \breve{F}$ *où* $v_1, \cdots, v_n$ *sont des variables d'individu ou de relation et* $\breve{F}$ *une formule ne commençant pas par* $\forall$ *(l'entier* n *peut être éventuellement nul).*

Toute formule ne commençant pas par $\forall$ *est d'une et d'une seule des formes suivantes :* $B \to C$ *ou* $E^{\alpha}[u]$ *ou* $D^{L}[u]$ *(*$L \neq \emptyset$*) ou* $X(x_1 \cdots x_n)$ *(*X *variable de relation n-aire).*

Preuve La preuve de ce lemme est triviale. □

Remarquer que la formule $D^{\emptyset}[t]$, en vertu de l'équivalence évoquée dans la définition 3.1.3, peut s'écrire $\forall X(B \to C)$:

par exemple si $F = N^{\emptyset}[u] = \forall X(\forall y(X\,y \to X\,\boldsymbol{s}(y)), X\,\boldsymbol{0} \to X\,u)$, alors $\breve{F}$ est de la forme $B \to C$.

Nous définirons ensuite, comme il est d'usage, les autres connecteurs logiques et l'égalité de deux termes d'individu (selon la méthode de Leibnitz).

Notation 3.1.5 *Nous utliserons les notations suivantes : la formule* $\forall X\,X$ *est notée* $\bot$ *;*

la formule $F \to \bot$ *est notée* $\neg F$ *;*

la formule $\forall X\,((F \to X), (G \to X) \to X)$ *est notée* $F \vee G$ *;*

la formule $\forall X\,((F, G \to X) \to X)$ *est notée* $F \wedge G$ *;*

la formule $\forall X\,(\forall \nu(F \to X) \to X)$ *est notée* $\exists \nu F$ *;*

la formule $\forall X\,(X\,u \to X\,v)$ *est notée* $u = v$.

3.2 Le système de typage

La principale différence entre $Ex2$ et AF2, outre la manipulation d'exceptions, est que nous utiliserons une égalité (entre termes d'individus) conditionnelle : notre égalité sera conditionnée par des hypothèses (des formules) ; et pour manipuler ces égalités conditionnelles nous utiliserons un système logique *ad hoc*.

3.2.1 Le système logique

Nous définissons donc un système logique du second ordre.

Définition 3.2.1 *Etant donné un ensemble fini de formules de* Φ_E : $\mathcal{A} \stackrel{def}{=} A_1 \cdots A_n$, *on dira que la formule* A *est conséquence de* $\mathcal{A}$, *compte tenu des équations de base* $\mathcal{E}$, *si on a pu obtenir* $\mathcal{A} \Vdash^{\mathcal{E}} A$ *au moyen des règles suivantes.*

Leg $\mathcal{A} \Vdash^{\mathcal{E}} A$ *si* A *est un cas particulier d'un axiome égalitaire, ou d'une équation de base (voir plus loin pour la définition de ces termes).*

Lvar $\mathcal{A} \Vdash^{\mathcal{E}} A_i$ *pour tout* i *vérifiant* $1 \leq i \leq n$.

Labs *Si* $\mathcal{A}, B \Vdash^{\mathcal{E}} C$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} B \to C$.

Lapp *Si* $\mathcal{A} \Vdash^{\mathcal{E}} B \to C$ *et* $\mathcal{A} \Vdash^{\mathcal{E}} B$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} C$.

L$\forall I_1$ *Si* $\mathcal{A} \Vdash^{\mathcal{E}} A$ *et si* x *n'est pas libre dans* $\mathcal{A}$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} \forall x A$.

L$\forall E_1$ *Si* $\mathcal{A} \Vdash^{\mathcal{E}} \forall x A$ *alors, pour tout terme d'individu* t, $\mathcal{A} \Vdash^{\mathcal{E}} A[x := t]$.

L$\forall I_2$ *Si* $\mathcal{A} \Vdash^{\mathcal{E}} A$ *et si* X *n'est pas libre dans* $\mathcal{A}$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} \forall X A$.

L$\forall E_2$ *Si* $\mathcal{A} \Vdash^{\mathcal{E}} \forall X A$ *où* X *est une variable de relation n-aire, alors, pour toute formule* F, $\mathcal{A} \Vdash^{\mathcal{E}} A[X(x_1, \cdots, x_n) := F]$.

Leq *Si* $\mathcal{A} \Vdash^{\mathcal{E}} A[x := t_1]$ *et si* $\mathcal{A} \Vdash^{\mathcal{E}} t_1 = t_2$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} A[x := t_2]$.

Lexc *Si* $\mathcal{A} \Vdash^{\mathcal{E}} D[a]$ *avec* $\alpha = D_i$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} E^{\alpha}[r_{\alpha}(a)]$.

Lprop *Si* $\mathcal{A} \Vdash^{\mathcal{E}} E^{\alpha}[a]$ *alors, pour tout type* T, $\mathcal{A} \Vdash^{\mathcal{E}} T \to E^{\alpha}[a]$.

Ltry *Si* $\mathcal{A} \Vdash^{\mathcal{E}} D^{L+\alpha}[a]$ *et si* $\mathcal{A} \Vdash^{\mathcal{E}} \forall x(F[x] \to D^{L}[b])$ *avec* $\alpha = F^{j}$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} D^{L}[t_{\alpha}(a, x \to b)]$.

L$\vee I1$ *Si* $\mathcal{A} \Vdash^{\mathcal{E}} E^{\alpha}[a]$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} D^{\alpha}[a]$.

L$\vee I2$ *Si* $\mathcal{A} \Vdash^{\mathcal{E}} D^{L}[a]$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} D^{L+\alpha}[a]$.

L$\vee E$ *Si* $\mathcal{A}, D^{L}[a] \Vdash^{\mathcal{E}} T$, *si* $\mathcal{A}, E^{\alpha}[a] \Vdash^{\mathcal{E}} T$ *et si* $\mathcal{A} \Vdash^{\mathcal{E}} D^{L+\alpha}[a]$ *alors* $\mathcal{A} \Vdash^{\mathcal{E}} T$.

D et F sont des symboles de données , α une dénomination de type , L est un ensemble de dénominations de type ne contenant pas α et $L + \alpha$ désigne la réunion des ensembles L et $\{\alpha\}$.

En l'absence d'ambiguïté $\Vdash^{\mathcal{E}}$ *sera abrégé en* $\vdash$.

La règle Leq n'est pas indispensable, puisqu'elle est conséquence des autres règles, cependant elle a été conservée pour la symétrie avec le système de typage. Prouver que Leq est déductible des autres règles est simple : dans $\mathcal{A} \Vdash^{\mathcal{E}} (t_1 = t_2) \stackrel{def}{=} \forall X (X t_1 \to X t_2)$, il suffit de remplacer X. par $A[x:=.]$ (règle L$\forall E_2$) pour obtenir $A[x:=t_1] \to A[x:=t_2]$ et conclure.

Précisons maintenant cette notion d'axiome égalitaire. Nous avons besoin d'axiomes pour exprimer les liens entre les divers symboles utilisés dans la formation des termes d'individu : nous les nommerons axiomes égalitaires, ce sont des relations d'égalités (ou de non égalité) entre termes d'individus que nous postulerons.

Définition 3.2.2 *Les axiomes égalitaires sont au nombre de cinq:*

casE $E^{\alpha}[x] \to cas_{\alpha}(x,y,z) = y$

casD $D^{L}[x] \to cas_{\alpha}(x,y,z) = z$

trapE $D^{\emptyset}[x] \to t_{\alpha}(r_{\alpha}(x), f) = f(x)$ *si* $\alpha = D_i$

trapD $D^{L}[x] \to t_{\alpha}(x, f) = x$

pluralité $\neg(\mathbf{0} = \boldsymbol{s}(\mathbf{0}))$

D est un symbole de données , α une dénomination de type , et L un ensemble de dénominations de type ne contenant pas α.

Nous devons aussi préciser ce que nous entendons par équation de base. Comme dans AF2, le typage se fera compte tenu d'un ensemble d'équations de la forme $a = b$ où a et b sont des termes d'individu. Par exemple nous utiliserons les deux équations $add(\mathbf{0}, y) = y$ et $add(\boldsymbol{s}(x), y) = \boldsymbol{s}(add(x, y))$ pour définir l'addition de deux entiers.

Selon le point de vue, on peut voir les équations de base comme des axiomes égalitaires particuliers (sans hypothèses), ou les axiomes égalitaires comme des équations de base nécessaires.

Un cas particulier d'un axiome égalitaire (d'une équation de base) est la formule obtenue en remplaçant dans l'axiome égalitaire (l'équation de base) les variables d'individu par des termes quelconques.

3.2.2 Le système de typage

Définition 3.2.3 *Un contexte Γ s'écrit $x_1{:}A_1, \cdots, x_n{:}A_n$ où $x_1, \cdots, x_n$ sont des variables distinctes du λ-calcul et $A_1, A_2, \cdots, A_n$ des formules de Φ_E . On appelle contenu logique de Γ, notée $\widehat{\Gamma}$ l'ensemble de formules $A_1, A_2, \cdots, A_n$.*

Etant donné un terme t de Λ_E , un ensemble d'équations de base $\mathcal{E}$, un contexte $\Gamma \stackrel{def}{=} x_1{:}A_1, \cdots, x_n{:}A_n$, on dira que « t est typable de type A dans le

contexte Γ, compte tenu des équations $\mathcal{E}$ », (noté: $\Gamma \vdash^{\mathcal{E}} t : A$), si le typage a été obtenu au moyen des règles suivantes.

Tvar $\Gamma \vdash^{\mathcal{E}} x_i : A_i$ *pour tout* i *vérifiant* $i \leq n$.

Tabs *Si* $\Gamma, x : B \vdash^{\mathcal{E}} u : C$ *alors* $\Gamma \vdash^{\mathcal{E}} \lambda x\, u : B \to C$

Tapp *Si* $\Gamma \vdash^{\mathcal{E}} u : B \to C$ *et* $\Gamma \vdash^{\mathcal{E}} v : B$. *alors* $\Gamma \vdash^{\mathcal{E}} (u\ v) : C$.

T$\forall I_1$ *Si* $\Gamma \vdash^{\mathcal{E}} u : A$ *et si* x *n'est pas libre dans* Γ *alors* $\Gamma \vdash^{\mathcal{E}} u : \forall x A$.

T$\forall E_1$ *Si* $\Gamma \vdash^{\mathcal{E}} u : \forall x A$ *alors, pour tout terme d'individu* t, $\Gamma \vdash^{\mathcal{E}} u : A[x := t]$.

T$\forall I_2$ *Si* $\Gamma \vdash^{\mathcal{E}} u : A$ *et si* X *n'est pas libre dans* Γ *alors* $\Gamma \vdash^{\mathcal{E}} u : \forall X A$.

T$\forall E_2$ *Si* $\Gamma \vdash^{\mathcal{E}} u : \forall X A$ *où* X *est une variable de relation n-aire, alors, pour toute formule* F, $\Gamma \vdash^{\mathcal{E}} u : A[X(x_1, \cdots, x_n) := F]$.

Teq *Si* $\Gamma \vdash^{\mathcal{E}} u : A[x := t_1]$ *et si* $\widehat{\Gamma} \Vdash^{\mathcal{E}} t_1 = t_2$ *alors* $\Gamma \vdash^{\mathcal{E}} u : A[x := t_2]$.

Texc *Si* $\Gamma \vdash^{\mathcal{E}} u : D[a]$ *alors* $\Gamma \vdash^{\mathcal{E}} \varepsilon_\alpha \langle u \rangle : E^\alpha[r_\alpha(a)]$.

Tprop *Si* $\Gamma \vdash^{\mathcal{E}} u : E^\alpha[a]$ *alors, pour tout type* T, $\Gamma \vdash^{\mathcal{E}} u : T \to E^\alpha[a]$.

Ttry *Si* $\Gamma \vdash^{\mathcal{E}} u : D^{L+\alpha}[a]$ *et si* $\Gamma \vdash^{\mathcal{E}} v : \forall x (F[x] \to D^L[b])$ *avec* $\alpha = F^j$ *alors* $\Gamma \vdash^{\mathcal{E}} \tau_\alpha \langle u, v \rangle : D^L[t_\alpha(a, x \to b)]$.

T$\vee I1$ *Si* $\Gamma \vdash^{\mathcal{E}} u : E^\alpha[a]$ *alors* $\Gamma \vdash^{\mathcal{E}} u : D^\alpha[a]$.

T$\vee I2$ *Si* $\Gamma \vdash^{\mathcal{E}} u : D^L[a]$ *alors* $\Gamma \vdash^{\mathcal{E}} u : D^{L+\alpha}[a]$.

T$\vee E$ *Si* $\Gamma, x : D^L[a] \vdash^{\mathcal{E}} u : T$, *si* $\Gamma, x : E^\alpha[a] \vdash^{\mathcal{E}} u : T$ *et si* $\Gamma \vdash^{\mathcal{E}} v : D^{L+\alpha}[a]$ *alors* $\Gamma \vdash^{\mathcal{E}} u[x := v] : T$.

D et F sont des symboles de données , α une dénomination de type , et L un ensemble de dénominations de type ne contenant pas α.

En l'absence d'ambiguïté, $\vdash^{\mathcal{E}}$ sera abrégé en $\vdash$.

Nous pouvons faire une remarque pour éclaircir le lien entre système logique et système de typage : la seule règle de typage qui utilise le système logique est la règle Teq, donc seules les preuves du système logique dont la conclusion est de la forme $\widehat{\Gamma} \Vdash^{\mathcal{E}} t_1 = t_2$ seront utilisées dans les typages.

Les sept premières règles de typage du système AF2 ([10]) sont reprises à l'identique : de Tvar à T$\forall E_2$. La huitième Teq (remplacement des termes d'individus) est modifiée de façon à intégrer une égalité sous conditions (de type).

Les autres règles sont nouvelles, quelques remarques informelles peuvent aider à en comprendre l'usage.

- La règle Texc permet de construire une valeur exceptionnelle à partir d'un terme ayant pour type un type de données .

- La régle Tprop va permettre de propager les exceptions placées en tête comme nous le verrons dans le lemmme 3.4.3.

- Si on considère $D^{L+\alpha}[t]$ comme une notation pour $D^L[t] \cup E^\alpha[t]$, les règles T$\vee I1$ et T$\vee I2$ apparaissent comme des cas particuliers des règles habituelles d'introduction du $\vee$, et la règle T$\vee E$ comme un cas particulier de la règle habituelle d'élimination de $\vee$.

- La règle Ttry est aussi une règle d'élimination du $\vee$, mais pas du tout classique.

Les règles de typage peuvent être classées en :

- règles qui ne changent pas le λ-terme : T$\forall I_1$, T$\forall I_2$, T$\forall E_1$, T$\forall E_2$, Teq, Tprop, T$\vee I1$, T$\vee I2$

- règles dans lesquelles le λ-terme a été changé et où le λ-terme final a une forme particulière : Tabs, Tapp, Texc, Ttry où le terme est respectivement de la forme $\lambda x\ u$, $(u\ v)$, $\varepsilon_\alpha\langle u\rangle$, $\tau_\alpha\langle u, v\rangle$.

Outre Tvar une seule règle échappe à la classification précédente : T$\vee E$, dans cette règle, le terme final est modifié sans être pour autant d'une forme particulière.

Nous pouvons préciser les liens entre système de typage et système logique.

Lemme 3.2.4 *Si* $\Gamma \vdash^{\mathcal{E}} t : T$ *alors* $\widehat{\Gamma} \Vdash^{\mathcal{E}} T$.

Preuve La preuve est immédiate puisque le système logique est le décalque du système de typage en ce sens que chacune de ses règles L_i peut être obtenue en supprimant dans la règle T_i du système de typage les termes du λ-calcul présents. □

3.3 Mise en garde

Le système $Ex2$ est proche de AF2, mais en diffère suffisamment pour créer quelques surprises. Pour faciliter la lecture des chapitres suivants, en mettant en garde contre des automatismes de pensée, nous dressons une liste de propriétés « délicates ». Il s'agit de propriétés vérifiées dans les systèmes de λ-calcul habituels (en particulier AF2) qui sont fausses dans le système $Ex2$ (ou dont nous ne savons pas si elles sont vraies ou fausses). Chacun de ces points sera repris dans le cours du texte, au moment où il apparaîtra.

Variables libres Dans le typage d'un terme t, on ne peut pas (dans $Ex2$) supprimer du contexte les variables qui n'apparaissent pas libres dans t. En effet, dans notre système, une hypothèse peut être utilisée - dans la règle Teq - sans qu'on en garde trace dans le terme. Plus précisément, le résultat «si $\Gamma, x_1{:}A_1, \cdots, x_n{:}A_n \vdash t : A$ et si x_i n'est pas libre dans t $(1 \leq i \leq n)$ alors $\Gamma \vdash t : A$» est vrai dans AF2 mais n'est pas vrai dans $Ex2$, la preuve en est faite dans la remarque 5.1.9.

Dédoublement des variables Il est possible de typer $(x\ x)$ dans le contexte $x : N^\alpha[z]$, mais nous conjecturons (remarque 5.3.1) qu'il n'est pas possible de typer $(x\ y)$ dans le contexte $x : N^\alpha[z], y : N^\alpha[z]$.

Si cette conjecture s'avère exacte, on pourra en déduire que le dédoublement d'une variable n'est pas permis en $Ex2$. Plus précisément on aura le résultat suivant « si $\Gamma, x : A \vdash^{\mathcal{E}} t[x_1{:=}x, x_2{:=}x] : T$ alors il n'est pas nécessairement vrai que $\Gamma, x_1 : A, x_2 : A \vdash^{\mathcal{E}} t : T$ ».

Typage de $(u\ v)$ Nous n'avons pas démontré le résultat « si $(u\ v)$ est typable de type T dans le contexte Γ, alors il existe un type A tel que $\Gamma \vdash u : A \to T$ et $\Gamma \vdash v : A$ » ; le prouver n'est pas aussi immédiat qu'on pourrait le croire (remarque 6.1.2).

Nous terminons ce chapitre en énonçant quelques résultats élémentaires dont nous ferons un usage constant dans la suite.

3.4 Premiers résultats

3.4.1 Utilitaires

Lemme 3.4.1 *Si un terme t est typable dans le contexte* $\Gamma = x_1{:}A_1, \cdots, x_n{:}A_n$, *alors les seules variables libres de t appartiennent à* $\{x_1 \cdots x_n\}$.

Preuve La preuve se fait par récurrence sur la longueur du typage. Le cas où la dernière règle est T$\forall E$ est le seul à ne pas être trivial. On a

$$(a)\Gamma, x : D^L[a] \vdash u : T;\ (b)\Gamma, x : E^\alpha[a] \vdash u : T;\ (c)\Gamma \vdash v : D^{L+\alpha}[a]$$

On en déduit par hypothèse de récurrence que les variables libres de u appartiennent à $\{x, x_1 \cdots x_n\}$ et celles de v à $\{x_1 \cdots x_n\}$, donc celles de $u[x{:=}v]$ à $\{x_1 \cdots x_n\}$. □

Lemme 3.4.2 (Cas particulier de T$\vee E$) *Si* $\Gamma, x : D^L[a] \vdash u : T$ *et si* $\Gamma, x : E^\alpha[a] \vdash u : T$ *alors* $\Gamma, x : D^{L+\alpha}[a] \vdash u : T$.

Preuve Soit y une variable différente de x et qui n'appartient ni à Γ, ni à u. On a $\Gamma, x : D^L[a], y : D^{L+\alpha}[a] \vdash u : T$ et $\Gamma, x : E^\alpha[a], y : D^{L+\alpha}[a] \vdash u : T$. On a aussi (Tvar) $\Gamma, y : D^{L+\alpha}[a] \vdash y : D^{L+\alpha}[a]$. En appliquant la règle T$\vee E$ on obtient $\Gamma, y : D^{L+\alpha}[a] \vdash u[x := y] : T$. Il reste à renommer les variables pour retrouver la conclusion du lemme. □

Lemme 3.4.3 (Propagation) *De* $\Gamma \vdash u : E^\alpha[a]$ *et* $\Gamma \vdash v : T$ *(T type quelconque), on peut déduire* $\Gamma \vdash (u\ v) : E^\alpha[a]$.

Preuve De $\Gamma \vdash u : E^\alpha[a]$, on déduit (Tprop) $\Gamma \vdash u : T \to E^\alpha[a]$ et il suffit d'appliquer Tapp pour conclure. □

3.4.2 Propriétés des égalités

Nous examinons tout d'abord quelques propriétés de la relation d'égalité entre termes d'individus.

Lemme 3.4.4 *La relation d'égalité conditionnelle* $\mathcal{A} \Vdash u = v$ *est une relation d'équivalence.*

Si $\mathcal{A} \Vdash u = v$ *et* $\mathcal{A} \Vdash a = b$ *alors* $\mathcal{A} \Vdash u[x := a] = v[x = b]$ *(si x n'est pas libre dans* $\mathcal{A}$*).*

Preuve Rappelons que $u = v$ est une abréviation de la formule $\forall X\,(X\,u \to X\,v)$. Nous ne prouverons pas que l'égalité est une relation d'équivalence (c'est classique). Mais nous montrons la deuxième propriété.

- De $\mathcal{A} \Vdash u = v$ on déduit (L$\forall I_1$ et L$\forall I_2$) $\mathcal{A} \Vdash \forall x(u = v)$ puis $\mathcal{A} \Vdash (u = v)[x := a]$ c'est à dire $\mathcal{A} \Vdash \forall X\,(X\,u[x := a] \to X\,v[x := a])$ soit $\mathcal{A} \Vdash u[x := a] = v[x = a]$.
- On montre $\mathcal{A} \Vdash a = b \Rightarrow \mathcal{A} \Vdash v[x := a] = v[x = b]$ (si x n'est pas libre dans $\mathcal{A}$) par récurrence sur la complexité de v.
  - C'est évident si v est une constante, ou une variable (égale ou non à x).
  - Si $v = g(t_1, \cdots, t_n)$ alors par hypothèse de récurrence on a $\mathcal{A} \Vdash t_i[x := a] = t_i[x := b]$ $(1 \le i \le n)$ donc $\mathcal{A} \Vdash g(t_1[x := a], \cdots, t_n[x := a]) = g(t_1[x := b], \cdots, t_n[x := b])$ c'est à dire $\mathcal{A} \Vdash v[x := a] = v[x = b]$.
- La transitivité de l'égalité permet de conclure.

□

Nous démontrons ensuite deux propriétés de l'égalité qui peuvent être vues comme des axiomes égalitaires supplémentaires.

Lemme 3.4.5 *Soient D un symbole de données , α une dénomination de type et L un ensemble de dénominations de type ne contenant pas α. On a :*

cas2 $\Vdash (E^{\alpha}[x] \to A = B), (D^{L}[x] \to C = D), D^{L+\alpha}[x] \to cas_{\alpha}(x, A, C) = cas_{\alpha}(x, B, D)$

casfonc $\Vdash D^{L+\alpha}[x] \to cas_{\alpha}(x, f(A), f(B)) = f(cas_{\alpha}(x, A, B))$

x est une variable d'individu, A, B, C, D sont des termes et f une fonction.

Preuve Commençons par prouver cas2. Posons $\mathcal{A} \stackrel{def}{=} E^{\alpha}[x] \to A = B$, $D^{L}[x] \to C = D$.

- En utilisant l'axiome égalitaire casE, on obtient (1) $\mathcal{A}, E^{\alpha}[x] \Vdash cas_{\alpha}(x, A, C) = cas_{\alpha}(x, B, D))$ puisque $\mathcal{A}, E^{\alpha}[x] \Vdash cas_{\alpha}(x, A, C) = A$ et $\mathcal{A}, E^{\alpha}[x] \Vdash cas_{\alpha}(x, B, D) = B$.

- En utilisant l'axiome égalitaire casD, on obtient (2) $\mathcal{A}, D^{L}[x] \Vdash cas_{\alpha}(x, A, C) = cas_{\alpha}(x, B, D))$ puisque $\mathcal{A}, D^{L}[x] \Vdash cas_{\alpha}(x, A, C) = C$ et $\mathcal{A}, D^{L}[x] \Vdash cas_{\alpha}(x, B, D) = D$.

- On utilise alors la version logique du cas particulier de la règle T$\vee E$ (lemmes 3.4.2 et 3.2.4) appliquée à (1) et (2) pour obtenir $\mathcal{A}, D^{L+\alpha}[x] \Vdash cas_{\alpha}(x, A, C) = cas_{\alpha}(x, B, D))$ et conclure.

La preuve de casfonc a la même structure :

- $E^{\alpha}[x] \Vdash cas_{\alpha}(x, f(A), f(B)) = f(cas_{\alpha}(x, A, B))$ puisque sous l'hypothèse $E^{\alpha}[x]$, on a $cas_{\alpha}(x, f(A), f(B)) = f(A)$ et $f(cas_{\alpha}(x, A, B)) = f(A)$ (CasE).

- $D^{L}[x] \Vdash cas_{\alpha}(x, f(A), f(B)) = f(cas_{\alpha}(x, A, B))$ puisque sous l'hypothèse $D^{L}[x]$, on a $cas_{\alpha}(x, f(A), f(B)) = f(B)$ et $f(cas_{\alpha}(x, A, B)) = f(B)$ (CasD).

- On conclut en utilisant le lemme 3.4.2.

□

3.4.3 Conséquences de l'axiome pluralité

L'axiome égalitaire **pluralité** peut paraître énigmatique, mais certaines de ses conséquences éclairent sa signification. La première indique que le type d'un terme ne peut être à la fois exception et type de données .

Lemme 3.4.6 *Si D est un symbole de données , et α et β deux dénominations de type différentes, alors :*

- $\Vdash \forall x(E^\alpha[x] \to \neg D[x])$
- $\Vdash \forall x(D[x] \to \neg E^\alpha[x])$
- $\Vdash \forall x(E^\alpha[x] \to \neg E^\beta[x])$.

Preuve Nous montrons seulement la première partie du lemme, la deuxième partie en est une conséquence directe et la troisième se prouve de la même façon.

Nous savons (axiome égalitaire casE) que : $E^\alpha[x] \Vdash cas_\alpha(x, \mathbf{0}, \boldsymbol{s}(\mathbf{0})) = \mathbf{0}$

Nous savons (axiome égalitaire casD) que : $D[x] \Vdash cas_\alpha(x, \mathbf{0}, \boldsymbol{s}(\mathbf{0})) = \boldsymbol{s}(\mathbf{0})$

On a donc (transitivité de l'égalité) $E^\alpha[x], D[x] \Vdash \boldsymbol{s}(\mathbf{0}) = \mathbf{0}$, et l'axiome égalitaire pluralité nous permet d'obtenir $E^\alpha[x], D[x] \Vdash \bot$ et de conclure. □

La seconde conséquence de l'axiome **pluralité** est que deux termes de type pour l'un exception et pour l'autre type de données ne peuvent être égaux.

Lemme 3.4.7 *Si $\alpha \neq \beta$ alors $\Vdash \forall x \forall y(E^\alpha[x], E^\beta[y] \to \neg(x = y)))$ et si $\alpha \notin L$ alors $\Vdash \forall x \forall y(E^\alpha[x], D^L[y] \to \neg(x = y)))$.*

Preuve Nous prouvons ici aussi la première partie du lemme, la preuve de sa seconde partie étant semblable.

D'une part $E^\alpha[x], E^\beta[y], x = y \Vdash E^\alpha[y]$ (Lvar et Leq).

D'autre part $E^\alpha[x], E^\beta[y], x = y \Vdash \neg E^\alpha[y]$ par application du lemme 3.4.6.

On en déduit $E^\alpha[x], E^\beta[y], x = y \Vdash \bot$ ce qui permet de conclure. □

Chapitre 4

Forte normalisation

Pour montrer la forte normalisation de $Ex2$, nous construisons un nouveau système $Ex2_F$ dont nous montrons la forte normalisation, ce qui, apès avoir montré que typabilité dans $Ex2$ implique typabilité dans $Ex2_F$, nous permet de conclure. Nous suivons ainsi la méthode utilisée par Jean Louis Krivine pour le système AF2 ([10]).

4.1 Le système $Ex2_F$

Nous définissons un système $Ex2_F$ qui est au système $Ex2$ ce que le système F est à AF2 : c'est à dire un système où « le premier ordre a été supprimé».

4.1.1 Les formules

Les λ-termes de $Ex2_F$ sont exactement les λ-termes de $Ex2$: les éléments de Λ_E . Les formules de $Ex2_F$ sont obtenues à partir de celles de Φ_E en «supprimant le premier ordre».

Définition 4.1.1 *Le langage de $Ex2_F$ comprend les constantes propositionnelles suivantes:*

- *pour chaque dénomination de type α la constante E^α*
- *pour chaque couple formé d'un ensemble (éventuellement vide) $L \stackrel{def}{=} \alpha_1, \cdots, \alpha_n$ de dénominations de type , et d'un symbole de données D, la constante D^L.*

L'ensemble de formules $\mathcal{A}^F$ s'obtient par application d'une des règles:

- *une variable propositionnelle, une constante propositionnelle sont des éléments de $\mathcal{A}^F$*
- *si F et G sont éléments de $\mathcal{A}^F$, alors $(F \to G)$ est élément de $\mathcal{A}^F$*

– *si F est élément de $\mathcal{A}^F$, alors $\forall X F$ est élément de $\mathcal{A}^F$ (X variable propositionnelle).*

On définit une relation d'équivalence sur $\mathcal{A}^F$ en mettant en relation chaque formule $D^\emptyset$, pour tout symbole de données D, avec la formule D définissant dans le système F le type de données correspondant. Ainsi, par exemple, on a: $N^\emptyset \simeq \forall X((X \to X) \to (X \to X))$ et $B^\emptyset \simeq \forall X(X \to (X \to X))$. L'ensemble des formules Φ_E^F est le quotient de l'ensemble $\mathcal{A}^F$ par la relation d'équivalence précédente.

En l'absence d'ambiguïté, nous noterons indifféremment: $D^\emptyset$ ou D.

4.1.2 Le typage

Il suffit de reprendre les règles de typage du sytème $Ex2$ et d'y supprimer tout ce qui a trait au premier ordre.

Définition 4.1.2 *Etant donné un terme t de Λ_E, un contexte $\Gamma \stackrel{def}{=} x_1{:}A_1, \cdots, x_n{:}A_n$, on dira que « t est typable, dans le système $Ex2_F$, de type A dans le contexte Γ » (noté $\Gamma \vdash_F t : A$), si le typage a été obtenu au moyen des règles suivantes.*

T'var $\Gamma \vdash_F x_i : A_i$ *pour tout i vérifiant $1 \leq i \leq n$.*

T'abs *Si* $\Gamma, x : B \vdash_F u : C$ *alors* $\Gamma \vdash_F \lambda x\, u : B \to C$.

T'app *Si* $\Gamma \vdash_F u : B \to C$ *et* $\Gamma \vdash_F v : B$ *alors* $\Gamma \vdash_F (u\ v) : C$.

T'$\forall I_2$ *Si* $\Gamma \vdash_F u : A$ *et si X n'est pas libre dans* Γ *alors* $\Gamma \vdash_F u : \forall X\, A$.

T'$\forall E_2$ *Si* $\Gamma \vdash_F u : \forall X A$ *alors, pour toute formule F,* $\Gamma \vdash_F u : A[X := F]$.

T'exc *Si* $\Gamma \vdash_F u : D$ *et si* $\alpha = D_i$, *alors* $\Gamma \vdash_F \varepsilon_\alpha \langle u \rangle : E^\alpha$.

T'prop *Si* $\Gamma \vdash_F u : E^\alpha$ *alors, pour tout type T,* $\Gamma \vdash_F u : T \to E^\alpha$.

T'try *Si* $\Gamma \vdash_F u : D^{L+\alpha}$ *et si* $\Gamma \vdash_F v : F \to D^L$ *alors* $\Gamma \vdash_F \tau_\alpha \langle u, v \rangle : D^L$ *($\alpha = F^j$).*

T'$\vee I1$ *Si* $\Gamma \vdash_F u : E^\alpha$ *alors* $\Gamma \vdash_F u : D^\alpha$.

T'$\vee I2$ *Si* $\Gamma \vdash_F u : D^L$ *alors* $\Gamma \vdash_F u : D^{L+\alpha}$.

T'$\vee E$ *Si* $\Gamma, x : D^L \vdash_F u : T$, *si* $\Gamma, x : E^\alpha \vdash_F u : T$ *et si* $\Gamma \vdash_F v : D^{L+\alpha}$ *alors* $\Gamma \vdash_F u[x := v] : T$.

D, F sont des symboles de données , α une dénomination de type , L un ensemble de dénominations de type ne contenant pas α.

4.2 Le théorème de forte normalisation

4.2.1 Typabilité dans $Ex2$ et $Ex2_F$

Proposition 4.2.1 *Si un terme de Λ_E est typable dans le système $Ex2$ alors il est typable dans le système $Ex2_F$.*

Preuve Nous associons à chaque formule F du système $Ex2$ une formule F' du système $Ex2_F$ obtenue en «supprimant» dans F la «partie du premier ordre». Plus précisément F' est définie par induction sur F :

- si F est une variable propositionnelle X, alors $F' = X$; si $F = E^\alpha[a]$ alors $F' = E^\alpha$; si $F = D^L[a]$ alors $F' = D^L$.
- si $F = A \to B$ alors $F' = A' \to B'$
- si $F = \forall X\, A$ alors $F' = \forall X\, A'$

Si, dans $Ex2$, $x_1{:}A_1, \cdots, x_n{:}A_n \vdash^{\mathcal{E}} t{:}A$, on peut remplacer dans la preuve chaque règle T_i de $Ex2$ par la règle T_i' de $Ex2_F$ (les règles T$\forall I_1$, T$\forall E_1$ et Teq étant elles supprimées), chaque formule F de Φ_E par la formule F' de Φ_E^F. On obtient alors une preuve dans le système $Ex2_F$ de $x_1{:}A_1', \cdots, x_n{:}A_n' \vdash_F t{:}A'$. □

Remarque 4.2.2 *Dans [10], Krivine montre la réciproque de la proposition précédente, simplement en faisant remarquer que les règles de typage du système F forment un sous système des règles de typage de AF2. Mais ce raisonnement n'est plus vrai dans notre construction. En effet les formules du système $Ex2_F$ ne constituent plus une partie de l'ensemble des formules de $Ex2$: E^α n'est pas une formule de $Ex2$. Est ce qu'on pourrait rajouter E^α aux formules de $Ex2$? C'est une question que nous avons laissée ouverte.*

4.2.2 Forte normalisation dans $Ex2$ et $Ex2_F$

Rappelons tout d'abord qu'un terme t est normalisable s'il existe un terme t' normal auquel t puisse se réduire, et qu'un terme t est fortement normalisable s'il n'existe aucune suite infinie de termes $t_0 = t,\ t_1, \cdots, t_n, \cdots$ telle que pour tout $i \geq 0$ on ait $t_i \to_\delta t_{i+1}$.

Théorème 4.2.3 (Forte normalisation de $Ex2_F$) *Si t est un terme de Λ_E typable dans le système $Ex2_F$ alors t est fortement normalisable.*

Preuve La preuve de ce théorème commence au paragraphe suivant et occupe le reste du chapitre. □

De ce théorème nous déduisons le théorème de forte normalisation du système $Ex2$.

Théorème 4.2.4 (Forte normalisation de $Ex2$) *Si t est un terme de Λ_E typable dans le système $Ex2$ alors t est fortement normalisable.*

Preuve En effet, le terme t est typable dans le système $Ex2_F$, d'après la proposition 4.2.1, et le théorème précédent permet de conclure. □

La suite du chapitre est consacrée à la preuve du théorème de forte normalisation du système $Ex2_F$ comme annoncé.

4.3 Ensemble des termes fortement normalisables

4.3.1 Saturations

Nous définissons cinq propriétés de saturation pour un ensemble de λ-termes.

Définition 4.3.1 *On note $\mathcal{N}$ l'ensemble des termes de Λ_E fortement normalisables et $\mathcal{E}^\alpha$ l'ensemble des termes de $\mathcal{N}$ qui ont un réduit de la forme $\varepsilon_\alpha\langle u\rangle$.*

Un ensemble A de λ-termes sera dit :
- sat1 si

$$\forall u, t_1, \cdots, t_n \in \Lambda_E\,, \forall t \in \mathcal{N}((u[x:=t]\ t_1 \cdots t_n) \in A \Rightarrow (\lambda x\ u\ t\ t_1 \cdots t_n) \in A)$$

- sat2 si pour toute dénomination de type α, A est $sat2_\alpha$ c'est à dire

$$\forall v, t_1, \cdots, t_n \in \Lambda_E\,, \forall t \in \mathcal{N}((t\ t_1 \cdots t_n) \in A, t \notin \mathcal{E}^\alpha, v \in \mathcal{N} \Rightarrow (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \in A)$$

- sat3 si pour toute dénomination de type α, A est $sat3_\alpha$ c'est à dire

$$\forall v, t, t_1, \cdots, t_n \in \Lambda_E\,(t \in \mathcal{E}^\alpha, \forall w(t \to_s \varepsilon_\alpha\langle w\rangle \Rightarrow (v\ w\ t_1 \cdots t_n) \in A) \Rightarrow (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \in A)$$

- Rclos si $\forall t(t \in A \Rightarrow \forall t'(t \to_\delta t' \Rightarrow t' \in A))$;
- Eclos si $\forall t(t \in A \Rightarrow \forall v(v \in \mathcal{N} \Rightarrow (t\ v) \in A))$.

Remarquons que la propriété pour un ensemble d'être sat1 est nommée par Krivine dans [10] «être $\mathcal{N}$-saturé».

4.3.2 Premières propriétés des ensembles saturés

Définition 4.3.2 *Soient A et B deux sous-ensembles de Λ_E, la notation $A \to B$ désigne l'ensemble des éléments t de Λ_E tels que, pour tout u de A, on ait $(t\ u) \in B$.*

Lemme 4.3.3 *Soient A et B deux sous-ensembles de Λ_E, $A_i (i \in I)$ une famille de sous-ensembles de Λ_E,*

- *si B est sat1 (respectivement sat2, sat3, Rclos) alors $A \to B$ est sat1 (respectivement sat2, sat3, Rclos);*
- *si chacun des A_i est sat1 (respectivement sat2, sat3, Rclos) alors $\bigcap_{i\in I} A_i$ est sat1 (respectivement sat2, sat3, Rclos);*
- *si chacun des A_i est sat1 (respectivement sat2, Rclos, Eclos) alors $\bigcup_{i\in I} A_i$ est sat1 (respectivement sat2, Rclos, Eclos).*

Preuve Prouvons la première partie du lemme pour la propriété sat1. Supposons que $(u[x:=t]\ t_1 \cdots t_n) \in A \to B$, on a alors $(\lambda x\ u\ t\ t_1 \cdots t_n) \in A \to B$. En effet si $a \in A$ alors $(u[x:=t]\ t_1 \cdots t_n a) \in B$ et B est sat1, donc $(\lambda x\ u\ t\ t_1 \cdots t_n\ a) \in B$. La démonstration est la même pour sat2, sat3 et Rclos.

Prouvons la deuxième partie du lemme pour sat2_α. Supposons que $(t\ t_1 \cdots t_n) \in \bigcap_{i\in I} A_i, t \notin \mathcal{E}^\alpha, v \in \mathcal{N}$. Pour tout i de I, puisque A_i est sat2_α, $(\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \in A_i$, donc $(\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \in \bigcap_{i\in I} A_i$. La démonstration est la même pour sat1, sat3 et Rclos.

Prouvons la troisième partie du lemme pour Rclos. Supposons que $t \in \bigcup_{i\in I} A_i$, on a donc, par exemple, $t \in A_k (k \in I)$. Mais A_k est Rclos donc $t' \in A_k$ si $t \to_\delta t'$. La démonstration est la même pour sat1, sat2, Eclos. □

Mais on ne peut pas affirmer directement que si deux sous-ensembles A et B de Λ_E sont sat3 alors $A \cup B$ est sat3. En effet, l'hypothèse est dans ce cas

$$\forall w(t \to_\delta \varepsilon_\alpha\langle w\rangle \Rightarrow (v\ wt_1 \cdots t_n) \in A \cup B)$$

et donc on peut avoir $t \to_\delta \varepsilon_\alpha\langle w_1\rangle, t \to_\delta \varepsilon_\alpha\langle w_2\rangle$ et $(v\ w_1 t_1 \cdots t_n) \in A, (v\ w_2 t_1 \cdots t_n) \in B$. On ne peut donc pas utiliser le fait que A et B sont sat3.

Nous verrons plus loin qu'il nous suffira de montrer que $A \cup B$ est sat3 (si A et B le sont) pour certains ensembles A et B particuliers.

4.3.3 Propriétés élémentaires de $\mathcal{E}^\alpha$

Lemme 4.3.4 *L'ensemble $\mathcal{E}^\alpha$ a les propriétés suivantes.*

1) Si $t \in \mathcal{E}^\alpha$ alors la forme normale de t est de la forme $\varepsilon_\alpha\langle u\rangle$ (avec $u \in \mathcal{N}$).

2) Si $t \in \mathcal{E}^\alpha$ et si $t \to_\delta t'$ alors $t' \in \mathcal{E}^\alpha$.

3) Si $t \in \mathcal{E}^\alpha$ alors t ne peut se réduire ni à $\lambda x\ u$, ni à $\varepsilon_\beta\langle u\rangle$ pour $\alpha \neq \beta$, ni à $(x\ t_1 \cdots t_n)$ où x est une variable.

4) Si $t \notin \mathcal{E}^\alpha$ et si $t \to_\delta t'$ alors $t' \notin \mathcal{E}^\alpha$.

Preuve 1) D'après la définition de $\mathcal{E}^\alpha$, t a un réduit $\varepsilon_\alpha\langle v\rangle$ donc une forme normale $\varepsilon_\alpha\langle u\rangle$ et, bien sûr, u est fortement normalisable puisque t l'est.

2) La forme normale de t est $\varepsilon_\alpha\langle u\rangle$, et puisque $\boldsymbol{\delta}$ est confluente, t' se réduit à cette forme normale.

3) En effet, sinon, la forme normale de t ne serait pas $\varepsilon_\alpha\langle u\rangle$.

4) En effet, sinon, on aurait $t \to_\delta t' \to_\delta \varepsilon_\alpha\langle u\rangle$. □

4.3.4 Propriétés de $\mathcal{N}$

Proposition 4.3.5 *L'ensemble $\mathcal{N}$ est sat1, sat2, sat3, Rclos.*

Preuve

$\mathcal{N}$ est sat1. Nous montrons que

$$(u[x:=t]\ t_1 \cdots t_n) \in \mathcal{N}, t \in \mathcal{N} \Rightarrow T \stackrel{def}{=} (\lambda x\ u\ t\ t_1 \cdots t_n) \in \mathcal{N}$$

par induction sur $p+q$, où p est la longueur de la plus longue chaîne de réduction de $(u[x:=t]\ t_1 \cdots t_n)$ et q la longueur de la plus longue chaîne de réduction de t.

Pour cela nous montrons que tous les réduits à un pas T' de T sont fortement normalisables, ce qui prouve que T l'est aussi.

- Soit $T' = (u[x:=t]\ t_1 \cdots t_n)$ et alors T' est dans $\mathcal{N}$ par hypothèse.
- Soit $T' = (\lambda x\ u'\ t\ t_1 \cdots t_n)$ avec $u \to_{\delta_0} u'$. On a $u[x:=t] \to_\delta u'[x:=t]$, et $(u[x:=t]\ t_1 \cdots t_n) \to_\delta (u'[x:=t]\ t_1 \cdots t_n)$. Donc p' longueur de la plus longue chaîne de réduction de $(u'[x:=t]\ t_1 \cdots t_n)$ vérifie $p' < p$ et on obtient $(\lambda x\ u'\ t\ t_1 \cdots t_n) \in \mathcal{N}$ par hypothèse de récurrence .
- Soit $T' = (\lambda x\ u\ t\ t_1 \cdots t_{k-1} t'_k t_{k+1} \cdots t_n)$ avec $t_k \to_{\delta_0} t'_k$ et la preuve est la même que ci-dessus.
- Soit $T' = (\lambda x\ u\ t'\ t_1 \cdots t_n)$ avec $t \to_{\delta_0} t'$. Alors q' longueur de la plus longue chaîne de réduction de t' vérifie $q' < q$.

  On a $u[x:=t] \to_\delta u[x:=t']$ et $(u[x:=t]\ t_1 \cdots t_n) \to_\delta (u[x:=t']\ t_1 \cdots t_n)$. Donc $(u[x:=t']\ t_1 \cdots t_n) \in \mathcal{N}$ et p' longueur de la plus longue chaîne de réduction de $(u[x:=t']\ t_1 \cdots t_n)$ vérifie $p' \leq p$.

  On obtient donc $(\lambda x\ u\ t'\ t_1 \cdots t_n) \in \mathcal{N}$ par hypothèse de récurrence .

$\mathcal{N}$ est sat2.

- Informellement : le terme $T = (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n)$ ne peut être non fortement normalisable sans que dans la réduction infinie présumée il y ait réduction du « $\tau_\alpha\langle,\rangle$ » puisque les sous termes $t, v, t_1, \cdots, t_n$ sont

fortement normalisables. Or si $t \notin \mathcal{E}^\alpha$, t ne peut se réduire à $\varepsilon_\alpha\langle u\rangle$ (si t fortement normalisable se réduisait à $\varepsilon_\alpha\langle u\rangle$, u serait fortement normalisable et donc on aurait $t \in \mathcal{E}^\alpha$), donc si T se réduit il se réduira à $(t'\ t'_1 \cdots t'_n)$ qui est fortement normalisable comme réduit de $(t\ t_1 \cdots t_n)$.

- Formellement : nous montrons que

$$(t\ t_1 \cdots t_n) \in \mathcal{N}, t \notin \mathcal{E}^\alpha, v \in \mathcal{N} \Rightarrow T \stackrel{def}{=} (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \in \mathcal{N}$$

par induction sur $p+q$, où p est la longueur de la plus longue chaîne de réduction de $(t\ t_1 \cdots t_n)$ et q la longueur de la plus longue chaîne de réduction de v.

Pour cela nous montrons que tous les réduits à un pas T' de T sont fortement normalisables, ce qui prouve que T l'est aussi.

  - Soit $T' = (\tau_\alpha\langle t', v\rangle\ t_1 \cdots t_n)$ avec $t \rightarrow_{\delta_0} t'$ et l'hypothèse de récurrence s'applique car $(t\ t_1 \cdots t_n) \rightarrow_\delta (t'\ t_1 \cdots t_n)$ donc $(t'\ t_1 \cdots t_n) \in \mathcal{N}$ avec $t' \notin \mathcal{E}^\alpha$ et p' longueur de la plus longue chaîne de réduction de $(t'\ t_1 \cdots t_n)$ vérifie $p' < p$ car $(t\ t_1 \cdots t_n) \rightarrow_\delta (t'\ t_1 \cdots t_n)$.
  - Soit $T' = (\tau_\alpha\langle t, v'\rangle\ t_1 \cdots t_n)$ avec $v \rightarrow_{\delta_0} v'$ et l'hypothèse de récurrence s'applique car $v' \in \mathcal{N}$ et q' longueur de la plus longue chaîne de réduction de v vérifie $q' < q$.
  - Soit $T' = (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_{k-1}\ t'_k\ t_{k+1} \cdots t_n)$ avec $t_k \rightarrow_{\delta_0} t'_k$ et l'hypothèse de récurrence s'applique car $(t\ t_1 \cdots t_{k-1}\ t'_k\ t_{k+1} \cdots t_n) \in \mathcal{N}$ et p' longueur de la plus longue chaîne de réduction de $(t\ t_1 \cdots t_{k-1}\ t'_k\ t_{k+1} \cdots t_n)$ vérifie $p' < p$.
  - Soit $T' = (t\ t_1 \cdots t_n)$ qui est par hypothèse fortement normalisable .

$\mathcal{N}$ est sat3.

- Informellement : ici aussi, le terme $T = (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n)$ ne peut être non fortement normalisable sans que dans la réduction infinie présumée il y ait réduction du « $\tau_\alpha\langle,\rangle$ ». Or si $t \in \mathcal{E}^\alpha$, t ne pourra se réduire ni à $\lambda x\ u$ ni à $\varepsilon_\beta\langle u\rangle$ avec $\alpha \neq \beta$, donc, s'il y a réduction du « $\tau_\alpha\langle,\rangle$ », t se reduira à $\varepsilon_\alpha\langle w\rangle$ et T se réduira à $(\tau_\alpha\langle\varepsilon_\alpha\langle w\rangle, v'\rangle\ t'_1 \cdots t'_n)$ puis à $(v'\ wt'_1 \cdots t'_n)$ mais on sait (hypothése) que $(v\ wt_1 \cdots t_n)$ est fortement normalisable donc $(v'\ wt'_1 \cdots t'_n)$ l'est aussi et donc la seule potentialité de réduction infinie s'avère interdite.

- Formellement : nous montrons que

$$t \in \mathcal{E}^\alpha, \forall w(t \rightarrow_\delta \varepsilon_\alpha\langle w\rangle \rightarrow (v\ w\ t_1 \cdots t_n) \in \mathcal{N}) \Rightarrow T \stackrel{def}{=} (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \in \mathcal{N}$$

par induction sur $p + Q$ où p est la longueur de la plus longue chaîne de réduction de t et $Q = \sum q_w$ où, pour w tel que $t \to_\delta \varepsilon_\alpha\langle w\rangle$, q_w est la longueur de la plus longue chaîne de réduction de $(v\ w\ t_1 \cdots t_n)$.

On peut remarquer que les w tels que $t \to_\delta \varepsilon_\alpha\langle w\rangle$ sont en nombre fini car d'une part $t \in \mathcal{E}^\alpha \subset \mathcal{N}$ donc aucune chaîne de réduction de t n'est infinie, d'autre part la δ-réduction est une relation dont l'arbre est à branchement fini donc, selon le lemme de Koenig, l'arbre des réductions de t est fini.

Nous montrons que tous les réduits à un pas T' de T sont fortement normalisables, ce qui prouve que T l'est aussi.

- Soit $T' = (\tau_\alpha\langle t', v\rangle\ t_1 \cdots t_n)$ avec $t \to_{\delta_0} t'$. L'hypothèse de récurrence s'applique car :
  - $t' \in \mathcal{E}^\alpha$ (lemme 4.3.4) et si $t' \to_\delta \varepsilon_\alpha\langle w\rangle$ alors $t \to_\delta \varepsilon_\alpha\langle w\rangle$ donc $(v\ w\ t_1 \cdots t_n) \in \mathcal{N}$;
  - si p' est la longueur de la plus longue chaîne de réduction de t' on a $p' < p$;
  - l'ensemble des w tels que $t' \to_\delta \varepsilon_\alpha\langle w\rangle$ est inclus dans l'ensemble des w tels que $t \to_\delta \varepsilon_\alpha\langle w\rangle$ donc si $Q' = \sum q_w$ pour w tel que $t' \to_\delta \varepsilon_\alpha\langle w\rangle$ alors $Q' \leq Q$.
- Soit $T' = (\tau_\alpha\langle t, v'\rangle\ t_1 \cdots t_n)$ avec $v \to_{\delta_0} v'$. Si $t \to_\delta \varepsilon_\alpha\langle w\rangle$ alors $(v\ w\ t_1 \cdots t_n) \in \mathcal{N}$ et donc $(v'\ w\ t_1 \cdots t_n) \in \mathcal{N}$.

  L'hypothèse de récurrence s'applique car, pour chaque w tel que $t \to_\delta \varepsilon_\alpha\langle w\rangle$, q'_w longueur de la plus longue chaîne de réduction de $(v'\ w\ t_1 \cdots t_n)$ vérifie $q'_w < q_w$ donc $Q' < Q$
- Soit $T' = (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_{k-1}\ t'_k\ t_{k+1} \cdots t_n)$ avec $t_k \to_{\delta_0} t'_k$. L'hypothèse de récurrence s'applique aussi car, comme dans le cas précédent, $q'_w < q_w$.
- Comme t appartient à $\mathcal{E}^\alpha$, il n'est égal ni à $\lambda x\, w$ ni à $\varepsilon_\beta\langle w\rangle$ si $\beta \neq \alpha$ (lemme 4.3.4). Donc le seul cas qu'il reste à envisager est $t = \varepsilon_\alpha\langle w\rangle$ et $T' = (v\ w\ t_1 \cdots t_n)$. Il suffit alors d'appliquer l'hypothèse $t\, \boldsymbol{\delta}\, \varepsilon_\alpha\langle w\rangle \Rightarrow (v\ w\ t_1 \cdots t_n) \in \mathcal{N}$.

$\mathcal{N}$ est Rclos.

En effet, sinon, de la réduction infinie de t' on déduit évidemment une réduction infinie de t. □

Remarque : $\mathcal{N}$ n'est évidemment pas Eclos, puisque, par exemple, si $\delta = \lambda x\,(x\ x)$ on a $\delta \in \mathcal{N}$ et $(\delta\ \delta) \notin \mathcal{N}$.

4.4 L' ensemble $\mathcal{N}_1$

4.4.1 Définitions

Définition 4.4.1 *On note $\mathcal{N}_0$ l'ensemble des termes de la forme $(x\ t_1 \cdots t_n)$ où x est une variable et où $t_1, \cdots, t_n$ sont des termes fortement normalisables et $\mathcal{N}_1$ le plus petit ensemble contenant $\mathcal{N}_0$ qui soit à la fois sat1, sat2, sat3 et Rclos.*

Plus formellement, on construit la suite (croissante) d'ensembles $(A_i)_{i\in N}$:

$$A_0 = \mathcal{N}_0; \quad \forall i \in N (A_{i+1} = A_i \cup B_i \cup C_i \cup D_i \cup F_i)$$

où les ensembles B_i, C_i, D_i, F_i sont définis ainsi:

$$B_i = \{(\lambda x\ u\ tt_1 \cdots t_n) \mid (u[x{:=}t]\ t_1 \cdots t_n) \in A_i\}$$

$$C_i = \{(\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \mid (t\ t_1 \cdots t_n) \in A_i, t \notin \mathcal{E}^\alpha, v \in \mathcal{N}\}$$

$$D_i = \{(\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \mid t \in \mathcal{E}^\alpha, \forall w (t \rightarrow_\delta \varepsilon_\alpha\langle w\rangle \Rightarrow (v\ wt_1 \cdots t_n) \in A_i\}$$

$$F_i = \{t' \mid t \in A_i\, t \rightarrow_\delta t'\}$$

et on définit $\mathcal{N}_1$ comme $\bigcup_{i\in N} A_i$.

4.4.2 Premières propriétés de $\mathcal{N}_1$

Proposition 4.4.2 *L'ensemble $\mathcal{N}_1$ est sat1, sat2, sat3, Rclos, Eclos et inclus dans $\mathcal{N}$.*

Preuve

- $\mathcal{N}_1$ est sat1, sat2, sat3.

  C'est une conséquence de sa definition. Montrons par exemple que $\mathcal{N}_1$ est sat3.

  Soit $t \in \mathcal{E}^\alpha$ tel que $\forall w (t \rightarrow_\delta \varepsilon_\alpha\langle w\rangle \Rightarrow (v\ w\ t_1 \cdots t_n) \in \mathcal{N}_1)$ est ce que $T \stackrel{def}{=} (\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \in \mathcal{N}_1$?

  Les w tels que $t \rightarrow_\delta \varepsilon_\alpha\langle w\rangle$ sont en nombre fini (même raisonnement que dans la preuve de la proposition 4.3.5). Nous nommons $w_1, \cdots, w_m$ les w tels que $t \rightarrow_\delta \varepsilon_\alpha\langle w\rangle$.

  Pour chaque entier j de 1 à m, de $(v\ w_j t_1 \cdots t_n) \in \mathcal{N}_1$ on déduit qu'il existe un entier p_j tel que $(v\ w_j t_1 \cdots t_n) \in A_{p_j}$.

  Soit q le plus grand de tous les entiers p_j pour j de 1 à m.

  Pour chaque entier j de 1 à m: $(v\ w_j t_1 \cdots t_n) \in A_{p_j} \subset A_q$.

  Donc par définition de la suite des A_i on a $T \in A_{q+1} \subset \mathcal{N}_1$.

- $\mathcal{N}_1$ est Rclos est immédiat.

- $\mathcal{N}_1$ est Eclos.

  On montre que pour tout i ($i \in N$), A_i est Eclos par recurrence sur i.

  - C'est évidemment vrai pour $A_0 \stackrel{def}{=} \mathcal{N}_0$.
  - Supposons que A_i est Eclos et montrons que A_{i+1} l'est aussi. Soit $T \in A_{i+1}, V \in \mathcal{N}$ montrons que $(T\ V) \in A_{i+1}$.
    - Si $T \in A_i$ c'est vrai par hypothèse de récurrence .
    - Si $T = (\lambda x\ u\ t\ t_1 \cdots t_n)$ avec $(u[x:=t]\ t_1 \cdots t_n) \in A_i$, on a $(u[x:=t]\ t_1 \cdots t_n\ v) \in A_i$ par hypothèse de récurrence et donc $(T\ v) \in B_i \subset A_{i+1}$.
    - Si $T = (\tau_\alpha\langle t, u\rangle\ t_1 \cdots t_n)$ avec $(t\ t_1 \cdots t_n) \in A_i, t \notin \mathcal{E}^\alpha, u \in \mathcal{N}$ alors par hypothèse de récurrence $(t\ t_1 \cdots t_n\ V) \in A_i$ et donc $(T\ V) \in A_{i+1}$.
    - si $T = (\tau_\alpha\langle t, u\rangle\ t_1 \cdots t_n)$ avec $t \in \mathcal{E}^\alpha$ et $\forall w(t \to_\delta \varepsilon_\alpha\langle w\rangle \Rightarrow (u\ wt_1 \cdots t_n) \in A_i)$ alors par hypothèse de récurrence $\forall w(t \to_\delta \varepsilon_\alpha\langle w\rangle \to (u\ wt_1 \cdots t_n V) \in A_i)$ donc $(T\ V) \in A_{i+1}$.
    - Si $T = t'$ avec $t \in A_i$ et $t \to_\delta t'$ alors par hypothèse de récurrence $(T\ V) \in A_i$ et $(t\ V) \to_\delta (t'\ V) = (T\ V)$ donc $(T\ V) \in A_{i+1}$.

  D'autre part, on sait que que si A_i est Eclos pour tout i de N, alors $\bigcup_{i\in N} A_i$ est Eclos (lemme 4.3.3), donc $\mathcal{N}_1$ est Eclos.

- $\mathcal{N}_1$ est inclus dans $\mathcal{N}$. C'est immédiat, en effet $\mathcal{N}$ contient $\mathcal{N}_0$, est sat1, sat2, sat3 et Rclos et $\mathcal{N}_1$ est le plus petit ensemble ayant ces cinq propriétés.

□

4.5 Interprétation

Une interprétation I est une fonction qui associe à chaque type T de Φ_E^F un sous-ensemble de Λ_E que nous noterons: I(T) ou $|T|_I$ ou même $|T|$ s'il n'y a pas d'ambiguïté.

4.5.1 Candidats de réductibilité

Définition 4.5.1 *L'ensemble $\mathcal{R}$ des candidats de réductibilité est l'ensemble des parties F de Λ_E qui vérifient $\mathcal{N}_0 \subset F \subset \mathcal{N}$ et F sat1, sat2, sat3 et Rclos.*

On peut remarquer que tout candidat de réductibilité contient $\mathcal{N}_1$. En effet $\mathcal{N}_1$ est le plus petit ensemble contenant $\mathcal{N}_0$ qui soit sat1, sat2, sat3 et Rclos.

Nous définissons ensuite un ensemble de λ-termes que nous utiliserons pour définir l'interprétation de la formule E^α.

4.5.2 L'ensemble $\mathcal{F}^\alpha$

Nous avons vu (définition 4.1.1) qu'à chaque symbole de données D on peut associer la formule définissant dans $Ex2_F$ le type de données correspondant, formule que nous noterons aussi D. Par exemple $B = \forall X(X, X \to X)$, $N = \forall X((X \to X), X \to X)$ et $LN = \forall X(N, X \to X), X \to X))$.

On associe ensuite à la formule D une partie $||D||$ de Λ_E, qui est définie par induction sur D.

Contentons nous ici de donner trois exemples:

- $||B|| = \cap_{F \in \mathcal{R}}(F \to (F \to F))$ et
- $||N|| = \cap_{F \in \mathcal{R}}((F \to F) \to (F \to F))$.
- $||LN|| = \cap_{F \in \mathcal{R}}((||N||, F \to F), F \to F)$.

On remarque immédiatement que pour chaque symbole de données D l'ensemble $||D||$ est Rclos.

Définition 4.5.2 *Si α est la dénomination de type D_i, l'ensemble $\mathcal{F}^\alpha$ est l'ensemble des termes t de Λ_E tels que :*

i) t est fortement normalisable

ii) t a un réduit de la forme $\varepsilon_\alpha\langle w\rangle$ avec $w \in ||D||$

iii) t n'a aucun réduit de la forme $\varepsilon_\alpha\langle w\rangle$ avec $w \notin ||D||$

Remarque 4.5.3 *Le lecteur curieux peut se demander si la condition iii) est nécéssaire. Nous devons avouer notre ignorance sur ce point : nous n'avons pas réussi à montrer que les conditions i) et ii) impliquent la condition iii). Nous ne savons pas non plus s'il peut exister un terme t de Λ_E qui se réduise à t' avec $t' \in ||D||$ et qui se réduise à t" avec $t'' \notin ||D||$.*

On montre ensuite quelques propriétés de $\mathcal{F}^\alpha$.

Proposition 4.5.4 *L'ensemble $\mathcal{F}^\alpha$ est sat1, sat2, sat3, Rclos et Eclos.*

Preuve Soit $\alpha = D_i$ une dénomination de type .

$\mathcal{F}^\alpha$ est sat1. Supposons que $(u[x:=t]\ t_1 \cdots t_n) \in \mathcal{F}^\alpha, t \in \mathcal{N}$ nous devons montrer que $T = (\lambda x\ u\ tt_1 \cdots t_n) \in \mathcal{F}^\alpha$.

- La condition i) est satisfaite puisque $\mathcal{N}$ est sat1.
- La condition ii) l'est car $(\lambda x\ u\ tt_1 \cdots t_n) \to_\delta (u[x:=t]\ t_1 \cdots t_n)$.
- On montre la condition iii) par l'absurde. Si on avait $T \to_\delta \varepsilon_\alpha\langle w\rangle$ avec $w \notin ||D||$ ce ne pourrait être que par une réduction de la forme :

  $$T \to_\delta (\lambda x\ u'\ t't'_1 \cdots t'_n) \to_\delta (u'[x:=t']\ t'_1 \cdots t'_n) \to_\delta \varepsilon_\alpha\langle w\rangle$$

  ce qui contredit le fait que $(u[x:=t]\ t_1 \cdots t_n) \in \mathcal{F}^\alpha$.

$\mathcal{F}^\alpha$ est sat2. Nous montrons que $\mathcal{F}^\alpha$ est sat2$_\beta$ (α et β sont deux dénominations de type égales ou non).

Supposons que $(t\ t_1 \cdots t_n) \in \mathcal{F}^\alpha, t \notin \mathcal{E}^\beta, v \in \mathcal{N}$ nous devons montrer que $T = (\tau_\beta\langle t, v\rangle\ t_1, \cdots, t_n) \in \mathcal{F}^\alpha$.

- La condition i) est satisfaite puisque $\mathcal{N}$ est sat2.
- La condition ii) est plus délicate. Quelle peut être la forme normale de t ?
  - Ce ne peut être ni $(x\ v_1 \cdots v_n)$ ni $\tau_\gamma\langle a, b\rangle$ car alors on n'aurait pas $(t\ t_1 \cdots t_n) \to_\delta \varepsilon_\beta\langle w\rangle$.
  - Ce ne peut pas être $\varepsilon_\beta\langle w\rangle$ car $t \notin \mathcal{E}_\beta$.
  - Ce peut être $\lambda x\ w$ ou $\varepsilon_\alpha\langle w\rangle$ si $\alpha \neq \beta$ mais dans les deux cas T va pouvoir se réduire de la même façon : $T \to_\delta (\tau_\beta\langle \bar{t}, v\rangle\ t_1 \cdots t_n) \to_\delta (\bar{t}\ t_1 \cdots t_n)$ (en notant $\bar{t}$ la forme normale de t).
    Or $(t\ t_1 \cdots t_n) \to_\delta \varepsilon_\beta\langle w\rangle$ avec $w \in ||D||$ donc, puisque la δ-réduction est confluente, $(\bar{t}\ t_1 \cdots t_n) \to_\delta \varepsilon_\alpha\langle \bar{w}\rangle$ et finalement $T \to_\delta \varepsilon_\alpha\langle \bar{w}\rangle$ où $\bar{w}$ forme normale de w appartient à $||D||$ puisque $||D||$ est Rclos.
- Passons à la condition iii). Sans réduction du $\tau_\beta\langle,\rangle$, T ne pourrait se réduire à $\varepsilon_\alpha\langle\rangle$, donc la seule réduction qui contredirait la condition iii) est $T \to_\delta (\tau_\beta\langle t', v'\rangle\ t'_1, \cdots, t'_n) \to_\delta (t'\ t'_1, \cdots, t'_n) \to_\delta \varepsilon_\alpha\langle u\rangle$ avec $u \notin ||D||$ car t' ne peut être de la forme $\varepsilon_\beta\langle a\rangle$ puisque $t \notin \mathcal{E}^\beta$.

  Mais si cela était, on aurait $(t\ t_1 \cdots t_n) \to_\delta (t'\ t'_1, \cdots, t'_n) \to_\delta \varepsilon_\alpha\langle u\rangle$ ce qui n'est pas possible puisque $(t\ t_1 \cdots t_n) \in \mathcal{F}^\alpha$.

$\mathcal{F}^\alpha$ est sat3. Soit $t \in \mathcal{E}^\beta$ tel que $\forall w(t \to_\delta \varepsilon_\beta\langle w\rangle \Rightarrow (v\ wt_1 \cdots t_n) \in \mathcal{F}^\alpha)$. Est ce que $T \stackrel{def}{=} (\tau_\beta\langle t, v\rangle\ t_1 \cdots t_n) \in \mathcal{F}^\alpha$?

- La condition i) se déduit de $\mathcal{N}$ est sat3.

- La condition ii) est vérifiée car si $t \in \mathcal{E}^\beta$ alors $t \to_\delta \varepsilon_\beta\langle a\rangle$ donc

$$T \to_\delta (\tau_\beta\langle\varepsilon_\beta\langle a\rangle, v\rangle\ t_1 \cdots t_n) \to_\delta (v\ at_1 \cdots t_n) \in \mathcal{F}^\alpha$$

- La condition iii) se montre par récurrence sur p + Q où p est la longueur de la plus longue chaîne de réduction de t et Q la somme des q_w pour w tel que $t \to_\delta \varepsilon_\beta\langle w\rangle$ avec q_w longueur de la plus longue chaîne de réduction de $(v\ wt_1 \cdots t_n)$ (les w sont en nombre fini : voir preuve de la proposition 4.3.5). On examine les réduits à un pas T' de T et on montre qu'aucun ne se réduit à $\varepsilon_\alpha\langle u\rangle$ avec $u \notin ||D||$.
  - Si $T' = (\tau_\beta\langle t', v\rangle\ t_1 \cdots t_n)$ l'hypothèse de récurrence s'applique car $t' \in \mathcal{E}^\beta$ (lemme 4.3.4), p' longueur de la plus longue chaîne de réduction de t' vérifie $p' < p$, l'ensemble des w tels que $t' \to_\delta \varepsilon_\beta\langle w\rangle$ est inclus dans l'ensemble des w tels que $t \to_\delta \varepsilon_\beta\langle w\rangle$.
  - Si $T' = (\tau_\beta\langle t, v'\rangle\ t_1 \cdots t_n)$ l'hypothèse de récurrence s'applique parce que pour chaque w tel que $t \to_\delta \varepsilon_\beta\langle w\rangle$, la longueur de la plus longue chaîne de réduction de $(v'\ wt_1 \cdots t_n)$ est inféreure à la longueur de la plus longue chaîne de réduction de $(v\ wt_1 \cdots t_n)$.
  - De même si $T' = (\tau_\beta\langle t, v\rangle\ t_1, \cdots, t_{k-1}, t'_k, t_{k+1} \cdots, t_n)$.
  - S'il y a réduction du $\tau_\beta\langle,\rangle$, c'est que $t = \varepsilon_\beta\langle w\rangle$ parce que t appartient à $\mathcal{E}^\beta$ et ne peut donc être ni de la forme $\lambda x\, u$, ni de la forme $\varepsilon_\alpha\langle u\rangle$ avec $\alpha \neq \beta$. Et on a alors $T \to_\delta (v\ wt_1 \cdots t_n)$ qui ne peut par hypothèse se réduire à $\varepsilon_\alpha\langle u\rangle$ avec $u \notin ||D||$.

$\mathcal{F}^\alpha$ est Rclos. Nous nous contenterons de montrer la condition ii) : si $t \to_\delta \varepsilon_\alpha\langle i\rangle$ alors $t' \to_\delta \varepsilon_\alpha\langle\bar{i}\rangle$ où $\bar{i}$ est la forme normale de i, et si $i \in ||D||$ alors $\bar{i} \in ||D||$.

$\mathcal{F}^\alpha$ est Eclos. Soit $t \in \mathcal{F}^\alpha, v \in \mathcal{N}$, montrons que $T \stackrel{def}{=} (t\ v) \in \mathcal{F}^\alpha$.

- La condition i) est satisfaite. En effet sinon pour avoir $(t\ v) \notin \mathcal{N}$ il faudrait puisque t et v sont fortement normalisables que l'application $(t\ v)$ soit réduite. Mais $t \to_\delta \lambda x\, u$ n'est pas possible puisque $t \in \mathcal{F}^\alpha$. Et si $t \to_\delta \varepsilon_\beta\langle u\rangle$ alors on aura $T \to_\delta (\varepsilon_\beta\langle u\rangle\ v) \to_\delta \varepsilon_\beta\langle u\rangle \in \mathcal{N}$.

- La condition ii) est évidemment satisfaite.

- Montrons la condition iii) par l'absurde. Pour que T se réduise à $\varepsilon_\alpha\langle u\rangle, u \notin ||D||$, la seule possibilité est qu'il y ait réduction de $(t\ v)$. Or si t' est un réduit de t, il ne peut être d'aucune des formes $\lambda x\, u$ ou $\varepsilon_\beta\langle u\rangle$ avec $\beta \neq \alpha$ donc la seule possibilité est $T \to_\delta (\varepsilon_\alpha\langle a\rangle\ v') \to_\delta \varepsilon_\alpha\langle a\rangle$. Mais alors, puisque $T \to_\delta \varepsilon_\alpha\langle u\rangle, u \notin ||D||$, on a $t \to_\delta \varepsilon_\alpha\langle a\rangle \to_\delta \varepsilon_\alpha\langle u\rangle$ ce qui est impossible.

□

Lemme 4.5.5 *Pour toutes dénominations de type α et β différentes, et tout symbole de données D :*

- *1)* $\mathcal{F}^\alpha \subset \mathcal{E}^\alpha$
- *2)* $\mathcal{N}_1 \cap \mathcal{E}^\alpha = \emptyset$
- *3)* $||D|| \cap \mathcal{E}^\alpha = \emptyset$
- *4)* $\mathcal{E}^\alpha \cap \mathcal{E}^\beta = \emptyset$.

Preuve

- La première propriété est évidente.
- Montrons la deuxième :
  - la forme normale d'un terme de $\mathcal{E}^\alpha$ est $\varepsilon_\alpha\langle u\rangle$;
  - la forme normale d'un terme de $\mathcal{N}_1$ est : soit $(x\ t_1 \cdots t_n)$ soit $\tau_\alpha\langle u, v\rangle$: (cela se montre par induction sur la construction de $\mathcal{N}_1$) ;
  - l'incompatibilité des formes normales prouve que les deux ensembles $\mathcal{N}_1$ et $\mathcal{E}^\alpha$ sont disjoints.
- Prouvons la troisième, pour la dénomination de type $\alpha = N_i$. Soit $t \in \mathcal{E}^\alpha$. Si t appartenait à $||N|| \stackrel{def}{=} \cap_{F\in\mathcal{R}}\ ((F \to F) \to (F \to F))$, t appartiendrait à $(\mathcal{N}_1 \to \mathcal{N}_1) \to (\mathcal{N}_1 \to \mathcal{N}_1)$, puisque $\mathcal{N}_1$ est un candidat de réductibilité. On aurait donc, pour $a \in \mathcal{N}_1 \to \mathcal{N}_1$ et $b \in \mathcal{N}_1$, $(t\ a\ b) \in \mathcal{N}_1$. Mais puisque la forme normale de t est $\varepsilon_\alpha\langle c\rangle$, celle de $(t\ a\ b)$ est aussi $\varepsilon_\alpha\langle c\rangle$ ce qui est contradictoire avec $(t\ a\ b) \in \mathcal{N}_1$.
- Pour prouver la quatrième propriété, il suffit de voir que les formes normales des termes des deux ensembles ne sont pas compatibles.

□

4.5.3 Interprétations

Nous pouvons maintenant définir ce qu'est une interprétation.

Définition 4.5.6 *Une interprétation est une fonction de Φ_E^F dans $\mathcal{R}$; les images des variables propositionnelles étant données, les règles suivantes définissent l'interprétation d'une formule quelconque :*

- $I(A \to B) \stackrel{def}{=} I(A) \to I(B)$
- $I(\forall X\, B) \stackrel{def}{=} \cap_{F\in\mathcal{R}}\, I[X := F](A)$ *où* $I[X := F]$ *est l'interprétation définie par* $I[X := F](X) \stackrel{def}{=} F$ *et* $I[X := F](Y) \stackrel{def}{=} I(Y)$ *si* $Y \neq X$

- $|E^\alpha| \stackrel{def}{=} \mathcal{N}_1 \cup \mathcal{F}^\alpha$.
- $|D^{L+\alpha}| \stackrel{def}{=} |D^L| \cup |E^\alpha|$
- $|D^\emptyset| \stackrel{def}{=} |D|$

  L est un ensemble de dénominations de type éventuellement vide, α une dénomination de type n'appartenant pas à L, et D un symbole de données.

On peut remarquer que l'interprétation de la formule $N \stackrel{def}{=} \forall X((X \to X) \to (X \to X))$ est, d'après notre définition, $\cap_{F \in \mathcal{R}}((F \to F) \to (F \to F)) \stackrel{def}{=} ||N||$; et, plus généralement, que, pour tout symbole de données D, $|D| = ||D||$.

On vérifie facilement que toute formule close a une interprétation unique (puisqu'indépendante des interprétations des variables propositionnelles).

4.5.4 Toute interprétation est un candidat de réductibilité

Nous examinons tout d'abord les propriétés de $|E^\alpha|$.

Proposition 4.5.7 *Pour toute dénomination de type α, $|E^\alpha|$ est un candidat de réductibilité Eclos.*

Preuve

- On a évidemment $\mathcal{N}_0 \subset |E^\alpha|$. D'autre part, $\mathcal{N}_1$ et $\mathcal{F}^\alpha$ sont des sous ensembles de $\mathcal{N}$ donc $|E^\alpha|$ aussi.
- Les ensembles $\mathcal{N}_1$ et $\mathcal{F}^\alpha$ sont sat1, sat2, Rclos, Eclos et ces propriétés «passent à la réunion» donc $|E^\alpha|$ est sat1, sat2, Rclos et Eclos.
- Montrer que $|E^\alpha|$ est sat3 est un peu plus difficile. Soit $t \in \mathcal{E}^\alpha$ tel que $\forall w(t \to_\delta \varepsilon_\alpha\langle w\rangle \Rightarrow (v\ w t_1 \cdots t_n) \in |E^\alpha|)$ est ce que $T \stackrel{def}{=} (\tau_\beta\langle t, v\rangle\ t_1 \cdots t_n) \in |E^\alpha|$?

  Les ensembles $\mathcal{N}_1$ et $\mathcal{F}^\alpha$ sont disjoints, considérons les w_j tels que $t \to_\delta \varepsilon_\alpha\langle w_j\rangle$. Nous montrons plus loin que les termes $(v\ w_j t_1 \cdots t_n)$ sont tous soit dans $\mathcal{N}_1$ soit dans $\mathcal{F}^\alpha$ ce qui nous permet d'appliquer la saturation3 de $\mathcal{N}_1$ ou $\mathcal{F}^\alpha$ pour conclure que $|E^\alpha|$ est sat3.

  Il nous reste à prouver l'affirmation que les $(v\ w_j t_1 \cdots t_n)$ sont tous soit dans $\mathcal{N}_1$ soit dans $\mathcal{F}^\alpha$. S'ils ne l'étaient pas, avec par exemple $(v\ w_1\ t_1 \cdots t_n) \in \mathcal{N}_1$ et $(v\ w_2\ t_1 \cdots t_n) \in \mathcal{F}^\alpha$, les deux termes $\varepsilon_\alpha\langle w_1\rangle$ et $\varepsilon_\alpha\langle w_2\rangle$ auraient la même forme normale (celle de t) et donc w_1 et w_2 auraient aussi même forme normale $\bar{w}$. On en déduirait $(v\ w_1\ t_1 \cdots t_n) \to_\delta (v\ \bar{w}\ t_1 \cdots t_n)$ et $(v\ w_2\ t_1 \cdots t_n) \to_\delta (v\ \bar{w}\ t_1 \cdots t_n)$, mais on aurait alors

$(v\ \bar{w}\ t_1 \cdots t_n) \in \mathcal{N}_1$ puisque $\mathcal{N}_1$ est Rclos, et $(v\ \bar{w}\ t_1 \cdots t_n) \in \mathcal{F}^\alpha$ puisque $\mathcal{F}^\alpha$ est Rclos et c'est impossible d'après le lemme 4.5.5.

□

Nous examinons ensuite les propriétés de saturation des interprétations. Seule sat3 n'est pas immédiate.

Proposition 4.5.8 *Pour tout type T et toute interprétation I, $|T|_I$ est un candidat de réductibilité.*

Preuve Nous montrons tout d'abord que $|T|_I$ est sat1, sat2, Rclos et qu'on a $\mathcal{N}_0 \subset |T|_I \subset \mathcal{N}$. La preuve est la même pour toutes ces propriétés, écrivons la pour sat1.

L'interprétation d'une variable propositionnelle est un candidat de réductibilité donc sat1. L'interprétation de E^α est sat1. Le lemme 4.3.3 permet de conclure compte tenu de la définition des interprétations.

Il est un peu plus difficile de montrer que $|T|_I$ est sat3 puisque, comme nous l'avons vu, on ne peut pas affirmer que si A et B sous-ensembles de $\mathcal{N}$ sont sat3 alors $A \cup B$ est sat3. Mais nous savons (proposition 4.5.7) que pour toute dénomination de type β, $|E^\beta|$ est sat3, il suffit donc de montrer que, pour tout ensemble de dénominations de type L, $|D^L|$ est sat3.

- $|D|$ est sat3_α. Ecrivons la preuve pour $D = N$, elle est très semblable pour un autre symbole de données. Rappelons que $|N| = \cap_{F \in \mathcal{R}}((F \to F) \to (F \to F))$. Chaque candidat de réductibilité F est sat3_α. On en déduit par «passage à la flèche» que $F \to F$ et $((F \to F) \to (F \to F))$ sont aussi sat3_α et par «passage à l'intersection» que $|N|$ est sat3_α.

- $|D^L|$ est sat3_α. La preuve se fait par récurrence sur le nombre d'éléments de L, ce qui revient à montrer que si $|D^L|$ est sat3_α, alors $|D^{L+\beta}|$ est sat3_α (α et β deux dénominations de type égales ou non).

  Soit donc $t \in \mathcal{E}^\alpha$ tel que $\forall w(t \to_\delta \varepsilon_\alpha\langle w\rangle \Rightarrow (v\ wt_1 \cdots t_n) \in |D^{L+\beta}|)$. Est ce que $(\tau_\alpha\langle t, v\rangle\ t_1 \cdots t_n) \in |D^{L+\beta}|$? Si nous considérons les w_j tels que $t \to_\delta \varepsilon_\alpha\langle w_j\rangle$ (ils sont en nombre fini car t est fortement normalisable), on peut affirmer (nous le montrons plus loin) que les $(v\ w_j t_1 \cdots t_n)$ sont soit tous dans $|D^L|$ soit tous dans $\mathcal{F}^\beta$ et donc on peut appliquer la propriété sat3_α à l'un de ces deux ensembles et conclure.

  Prouvons par l'absurde l'affirmation précédente: on ne peut avoir des $(v\ w_j t_1 \cdots t_n)$ à la fois dans $|D^L|$ et dans $\mathcal{F}^\beta$.

  Supposons que l'on ait $(v\ w_1 t_1 \cdots t_n) \in |D^L|$ et $(v\ w_2 t_1 \cdots t_n) \in \mathcal{F}^\beta$. Les termes w_1 et w_2 ont même forme normale $\bar{w}$ puisque les deux termes $\varepsilon_\alpha\langle w_1\rangle$ et $\varepsilon_\alpha\langle w_2\rangle$ ont même forme normale : celle de t. On en déduit

donc $(v\ w_1\ t_1 \cdots t_n) \to_\delta (v\ \bar{w}\ t_1 \cdots t_n)$ et $(v\ w_2\ t_1 \cdots t_n) \to_\delta (v\ \bar{w}\ t_1 \cdots t_n)$, ce qui prouve que $(v\ \bar{w}\ t_1 \cdots t_n) \in |D^L|$ puisque $|D^L|$ est Rclos et que $(v\ \bar{w}\ t_1 \cdots t_n) \in \mathcal{F}^\beta$ puisque $\mathcal{F}^\beta$ est Rclos. Et c'est impossible d'après le lemme 4.5.5.

□

4.6 Lemme d'adéquation

On montre le résultat habituellement appelé lemme d'adéquation (dont nous faisons un théorème).

Théorème 4.6.1 (Adéquation) *Soit I une interprétation.*
Si (1) $\Gamma \stackrel{def}{=} x_1{:}A_1, \cdots, x_n{:}A_n \vdash_F u : A$, *et si* $t_i \in |A_i|_I\ (1 \leq i \leq n)$ *alors* $u' \stackrel{def}{=} u[x_1{:=}t_1, \cdots, x_n{:=}t_n] \in |A|_I$.

Preuve Pour tout terme t nous noterons t' le terme $t[x_1{:=}t_1, \cdots, x_n{:=}t_n]$. Le théorème se prouve par induction sur le typage (1). Considérons la dernière règle utilisée.

- Si c'est la règle T'exc, on a $\alpha = Di$ et $\Gamma \vdash_F w : D$ donc par hypothèse de récurrence $w' \in |D|_I$ et donc $u' = \varepsilon_\alpha\langle w'\rangle \in \mathcal{F}^\alpha \subset |E^\alpha|_I$. En effet u' vérifie les trois conditions caractérisant l'appartenance à $\mathcal{F}^\alpha$. Seule iii) n'est pas évidente : si on a $u' \to_\delta \varepsilon_\alpha\langle a\rangle$ alors $w' \to_\delta a$ et $a \in |D|_I$ puisque $|D|_I$ est Rclos.

- Si c'est la règle T'prop, on a $\Gamma \vdash_F u : E^\alpha$, et de $u' \in |E^\alpha|$ on déduit $u' \in |T|_I \to |E^\alpha|$. En effet de $|E^\alpha|_I$ Eclos et $|T|_I \subset \mathcal{N}$ on déduit $|E^\alpha|_I \subset |T|_I \to |E^\alpha|_I = |T \to E^\alpha|_I$.

- Si c'est l'une des règles $T' \vee I1$ ou $T' \vee I2$ le résultat est immédiat puisqu'on interprète $D^{L+\alpha}$ comme la réunion des interprétations de D^L et de E^α.

- Si c'est la règle T'$\vee E$, on a $\Gamma, x : D^L \vdash_F u : T$ et $\Gamma, x : E^\alpha \vdash_F u : T$ et encore $\Gamma \vdash_F v : D^{L+\alpha}$ donc par hypothèse de récurrence $u[x_1{:=}t_1, \cdots, x_n{:=}t_n, x{:=}a] \in |T|_I$ pour tout a dans $|D^L|_I$, $u[x_1{:=}t_1, \cdots, x_n{:=}t_n, x{:=}b] \in |T|_I$ pour tout b dans $|E^\alpha|_I$ et $v[x_1{:=}t_1, \cdots, x_n{:=}t_n] \in |D^{L+\alpha}|_I = |D^L|_I \cup |E^\alpha|_I$.

  Considérons le terme $t'' \stackrel{def}{=} (u[x{:=}v])[x_1{:=}t_1, \cdots, x_n{:=}t_n]$. Les propriétés des substitutions permettent d'affirmer que $t'' = u[x_1{:=}t_1, \cdots, x_n{:=}t_n, x{:=}v[x_1{:=}t_1, \cdots, x_n{:=}t_n]]$.
  Que $v[x_1{:=}t_1, \cdots, x_n{:=}t_n]$ soit dans $|D^L|_I$ ou dans $|E^\alpha|_I$ on aura donc bien $t'' \in |T|_I$.

- Si c'est la règle T'try, on a $\Gamma \vdash_F u : D^{\beta,\alpha}$ et $\Gamma \vdash_F v : G \to D^{\beta})$ avec $\alpha = G^i$ (nous examinons un cas qui n'est pas le plus général, mais qui contient le seul point intéressant de la preuve). Par hypothèse de récurrence $v' \in |G|_I \to |D^L|_I$, $u' \in |D^{\beta,\alpha}|_I = |D|_I \cup |E^\beta|_I \cup |E^\alpha|_I$, et nous distinguons trois cas.

  - Si $u' \in |D|_I$ alors $u' \notin \mathcal{E}^\alpha$ et donc $\tau_\alpha\langle u', v'\rangle \in |D|_I$ puisque v' est fortement normalisable et $|D|_I$ sat2.
  - Si $u' \in |E^\beta|_I$, la preuve précédente peut être répétée puisque $|E^\beta|_I$ est sat2.
  - Si $u' \in |E^\alpha|_I$ nous devons distinguer deux cas :
    - soit $u' \in \mathcal{N}_1$ et $\mathcal{N}_1$ sat2 permet de conclure que $\tau_\alpha\langle u', v'\rangle \in \mathcal{N}_1 \subset |D|_I \subset |D^\beta|_I$;
    - soit $u' \in \mathcal{F}^\alpha$ et, par définition de $\mathcal{F}^\alpha$, on sait que si $u' \to_\delta \varepsilon_\alpha\langle a\rangle$ alors $a \in |G|_I$ et donc $(v\ a) \in |D^\beta|_I$; on en déduit (puisque $|D^\beta|_I$ est sat3) $\tau_\alpha\langle u', v'\rangle \in |D^\beta|_I$.

- Le cas où la dernière règle utilisée est l'une des règles T'var, T'abs, T'app, T'$\forall I_2$, T'$\forall E_2$ c'est à dire une des règles du système F se traite comme dans [10].

□

Du théorème d'adéquation, on déduit aisément le théorème de forte normalisation du système $Ex2_F$.

On considère l'interprétation I définie par $|X|_I = \mathcal{N}$ pour toute variable propositionnelle. Si $\Gamma \stackrel{def}{=} x_1{:}A_1, \cdots, x_n{:}A_n \vdash_F u : A$, on a $u = u[x_1{:=}x_1 \cdots x_n{:=}x_n] \in |A|_I \subset \mathcal{N}$, par application du théorème d'adéquation, puisque $x_i \in \mathcal{N}^0 \subset |A_i|_I$.

Chapitre 5

Conservation du type

L'objet de ce chapitre est de prouver la propriété dite « subject reduction ».

5.1 Clarification des preuves

Notre but est d'éliminer certaines façons d'utiliser les règles de typage qui s'avèrent inutiles et qui compliqueraient nos raisonnements si elles étaient conservées.

Nous définissons ainsi deux utilisations particulières de la règle T$\vee E$ dont nous montrerons plus tard qu'elles ne sont pas indispensables.

Définition 5.1.1 *Une utilisation de la règle T$\vee E$ de la forme : si* (a) $\Gamma, x : D^L[a] \vdash x : T$, *si* (b) $\Gamma, x : E^\alpha[a] \vdash x : T$ *et si* (c) $\Gamma \vdash v : D^{L+\alpha}[a]$ *alors* (d) $\Gamma \vdash x[x:=v] = v : T$ *est dite « avec u variable principale ».*

Un terme de Λ_E est dit « redexeur » s'il est de l'une des deux formes $\lambda x\, u$ ou $\varepsilon_\alpha\langle u\rangle$.

Une utilisation de la règle T$\vee E$ dans laquelle le terme v n'est pas redexeur est dite « profitable ».

Une preuve avec utilisation non profitable de la règle T$\vee E$ est dite « trouble », sinon elle est dite « claire ».

5.1.1 Lemmes sur les variables

Le premier lemme examine, dans un cas particulier, comment le type d'une variable peut évoluer à partir du type initial de la variable dans le contexte.

Lemme 5.1.2 *Si* $\Gamma, x : A \vdash x : T$ *alors :*

- *si* $A = Xa$ *alors* $T = \forall y_1 \cdots y_p Xb$ *(avec* $y_i \neq X$ *pour tout* i *tel que* $1 \leq i \leq p$*) et* $\widehat{\Gamma}, A \Vdash a = b$

- *si* $A = \forall y(X\,y \to X\,\boldsymbol{s}(y))$ *alors* $T = \forall y_1 \cdots y_p(X\,a \to X\,b)$ *(avec* $y_i \neq X$ *pour tout* i *tel que* $1 \leq i \leq p$*) et* $\widehat{\Gamma}, A \Vdash b = \boldsymbol{s}(a)$.

Preuve Nous prouvons la première partie du lemme, par récurrence sur la longueur du typage. La dernière règle ne peut être aucune des règles Tabs, Tapp, Texc, Ttry, Tprop, T$\vee I1$, T$\vee I2$. Examinons les autres possibilités.

- Si la dernière règle est Tvar le résultat est trivial.
- Si c'est T$\forall I_1$ ou T$\forall I_2$, l'hypothèse de récurrence permet de conclure (on ne peut évidemment pas introduire $\forall X$ puisque la variable X n'est pas libre dans le contexte).
- Si c'est T$\forall E_1$, on a de $\Gamma, x : A \vdash x : B \stackrel{def}{=} \forall z C$ déduit $\Gamma, x : A \vdash x : T \stackrel{def}{=} C[z:=t]$. Par hypothèse de récurrence on a $B = \forall z \forall y_1 \cdots y_p X\,b$ avec $\widehat{\Gamma}, A \Vdash a = b$, et $T = \forall y_1 \cdots y_p X\,b[z:=t]$. Mais $\widehat{\Gamma}, X\,a \Vdash b[z:=t] = a[z:=t] = a$ puisque z n'est pas libre dans $\Gamma, X\,a$.
- Si c'est T$\forall E_2$ le résultat est trivial puisqu'on passe du type $\forall Z \forall y_1 \cdots y_p X\,b$ avec $Z \neq X$ au type $\forall y_1 \cdots y_p X\,b$.
- Si c'est Teq, de $\Gamma, x : A \vdash x : \forall y_1 \cdots y_p X\,b[z:=u]$ et $\widehat{\Gamma}, A \Vdash u = v$ on déduit $\Gamma \vdash x : \forall y_1 \cdots y_p X\,b[z:=v]$. Mais par hypothèse de récurrence on a $\widehat{\Gamma}, A \Vdash b[z:=u] = a$ et puisque $\widehat{\Gamma}, A \Vdash b[z:=u] = b[z:=v]$ on obtient $\widehat{\Gamma}, A \Vdash b[z:=v] = a$.
- Si c'est T$\vee E$, on a

  $(a)\Gamma, x : A, y : D^L[c] \vdash u : T$; $(b)\Gamma, x : A, y : E^\alpha[c] \vdash u : T$; $(c)\Gamma, x : A \vdash v : D^{L+\alpha}[c]$

  et l'égalité $u[y:=v] = x$ nous amène à distinguer deux cas.

  - Soit $u = x$, et alors l'hypothèse de récurrence nous donne $T = \forall x_1 \cdots x_n X\,b$ ainsi que $\widehat{\Gamma}, A, D^L[c] \Vdash b = a$ et $\widehat{\Gamma}, A, E^\alpha[c] \Vdash b = a$. Avec $\widehat{\Gamma}, A \Vdash D^{L+\alpha}[c]$, on peut alors obtenir (L$\vee E$) $\widehat{\Gamma}, A \Vdash b = a$.
  - Soit $u = y$ et $v = x$, et (c) s'écrit alors $\Gamma, x : A \vdash x : D^{L+\alpha}[c]$ ce qui, par hypothèse de récurrence, n'est pas possible.

La preuve de la seconde partie du lemme est identique. □

Nous précisons ensuite les types possibles pour une variable, dans un contexte où elle a pour type la formule $D^L[a]$.

Lemme 5.1.3 *Soient* D *un symbole de données,* α *une dénomination de type,* L *un ensemble de dénominations de type.*

1. *Si* $\Gamma, x : D^L[a] \vdash x : T$ *alors* T *est de l'une des deux formes* $T_1 = \forall \nu_1, \cdots, \nu_n(A \to B)$ *ou* $T_2 = \forall \nu_1, \cdots, \nu_n D^{L+L'}[b]$ *(et alors* $\widehat{\Gamma}, D^L[a] \Vdash b = a$*).*

2. *Si* $\Gamma, x : E^\alpha[a] \vdash x : T$ *alors* T *est de l'une des formes :*

$$\forall \nu_1, \cdots, \nu_n E^\alpha[b] \text{ ou } \forall \nu_1, \cdots, \nu_n (A \to E^\alpha[b]) \text{ ou } \forall \nu_1, \cdots, \nu_n D^{L'+\alpha}[b]$$

avec $\widehat{\Gamma}, E^\alpha[a] \Vdash b = a$.

A et B sont des formules, L un ensemble de dénominations de type, l'entier n peut être éventuellement nul et $\nu_1, \cdots, \nu_n$ *sont des variables d'individu ou de relation.*

Preuve Les deux parties du lemme se prouvent parallèlement, par récurrence sur le typage. Nous rédigeons seulement la preuve de la première partie (la preuve de la seconde partie est très semblable) dans le cas où $D = N$.

Nous posons $\Gamma' = \Gamma, x : N^L[a]$ et nous examinons la dernière règle.

- La dernière règle ne peut être ni Tabs ni Tapp ni Texc ni Ttry.
- Dans le cas de la règle Tvar, le résultat est immédiat, puisque $N^\emptyset[a]$ s'écrit aussi $\forall X\,(A \to B)\ (N^\emptyset[a] = \forall X\,(\forall y(X\,y \to X\,\boldsymbol{s}(y)), X\,\mathbf{0} \to X\,a))$.
- Dans le cas des règles T$\forall I_1$, T$\forall I_2$ l'hypothèse de récurrence permet de conclure.
- Si la règle est T$\forall E_1$, on a donc de $\Gamma' \vdash x : A \stackrel{def}{=} \forall z T_i$ déduit $\Gamma' \vdash x : B \stackrel{def}{=} T_i[z := t]$;
  - si i=1, le résultat est prouvé puisque B est alors de type T_1 ;
  - si i=2, on a $A = \forall z, \nu_1, \cdots, \nu_n N^{L+L'}[b]$ et $\widehat{\Gamma'} \Vdash b = a$ et $B = \forall \nu_1, \cdots, \nu_n N^{L+L'}[b[z := t]]$, mais $\widehat{\Gamma'} \Vdash b[z := t] = a[z := t] = a$ puisque z n'est pas libre dans Γ'.
- Si la règle est T$\forall E_2$, avec les mêmes notations, le cas i=2 est trivial si $L + L' \neq \emptyset$ sinon il faut remarquer qu'on peut éventuellement passer du type $N[b] = \forall X\,(\forall y(X\,y \to X\,\boldsymbol{s}(y)), X\,\mathbf{0} \to X\,b)$ au type $\forall y(X\,y \to X\,\boldsymbol{s}(y)), X\,\mathbf{0} \to X\,b)[X\,z := F]$ qui est de la forme T_1 ; le cas $i = 1$ est trivial.
- Si la règle est Teq, le cas i=1 est trivial. Si i=2, de $\Gamma' \vdash x : \forall \nu_1, \cdots, \nu_n N^{L+L'}[b[z := u]]$ et $\widehat{\Gamma'} \Vdash u = v$ on a déduit $\Gamma' \vdash x : \forall \nu_1, \cdots, \nu_n N^{L+L'}[b[z := v]]$. Mais, par hypothèse de récurrence, on a $\widehat{\Gamma'} \Vdash [b[z := u] = a$ et, puisque $\widehat{\Gamma'} \Vdash [b[z := u] = [b[z := v]$, on obtient $\widehat{\Gamma'} \Vdash [b[z := v] = a$.
- La règle peut être Tprop seulement dans la seconde partie du lemme, mais le résultat est alors trivial.
- Si la règle est T$\vee I2$, le résultat est évident.

– Le cas de la règle T$\vee E$ est le plus long. On a

$$(a)\Gamma', y : F^{L''}[c] \vdash u : T;\ (b)\Gamma', y : E^{\beta}[c] \vdash u : T;\ (c)\Gamma' \vdash v : F^{L''+\beta}[c]$$

On doit distinguer deux cas.

– Soit u=x. Le cas i=1 est alors trivial et si i=2 on a $T = \forall \nu_1, \cdots, \nu_n$ $N^{L+L''}[b]$ avec

$$\widehat{\Gamma}', F^{L''}[c] \Vdash a = b;\ \widehat{\Gamma}', E^{\beta}[c] \Vdash a = b;\ \widehat{\Gamma}' \Vdash F^{L''}[c]$$

et donc, en utilisant la règle L$\vee E$, $\widehat{\Gamma}' \Vdash a = b$.

– Soit u=y et v=x. On déduit de (c), en appliquant l'hypothèse de récurrence que F=D. Le cas i=1 est alors évident en appliquant l'hypothèse de récurrence à (a). Dans le cas i=2, l'hypothèse de récurrence nous donne :

  – appliquée à (a) et (b) : $T = \forall \nu_1, \cdots, \nu_n F^{L''+\beta+l}[c]$ et (d) $\widehat{\Gamma}', y : F^{L''}[c] \Vdash b = c$ et (e) $\widehat{\Gamma}', y : E^{\beta}[c] \Vdash b = c$
  – appliquée à (c) : (f) $\widehat{\Gamma}' \Vdash b = a$.

  D'autre part on a (g) $\widehat{\Gamma}' \Vdash F^{L''+\beta}[c]$.

  En appliquant L$\vee E$ à (d), (e) et (g) on obtient : $\widehat{\Gamma}' \Vdash b = c$ qui combiné avec (f) nous donne $\widehat{\Gamma}' \Vdash a = c$. Enfin on a $L'' + \beta + l \supset L'' + \beta \supset L$ et $\alpha \in L'' + \beta + l$.

□

Nous pouvons alors envisager une première simplification des preuves.

Lemme 5.1.4 *Toute preuve contenant une ou plusieurs utilisations « avec u variable principale » de la règle T$\vee E$ peut être transformée en une preuve qui ne contient plus aucune utilisation de cette sorte, et, si la preuve de départ est claire, alors la preuve obtenue l'est aussi.*

Preuve Nous utilisons les notations de la définition 5.1.1.

Dans un premier temps nous montrons que dans une utilisation « avec u variable principale » de la règle T$\vee E$ le type T est égal à $\overrightarrow{\forall} D^{L+\alpha+L'}[b]$ avec L' éventuellement vide et $\widehat{\Gamma} \Vdash b = a$.

En effet, d'après le lemme 5.1.3, le seul type possible pour T doit être de la forme :

– d'une part $\forall \nu_1, \cdots, \nu_n (A \to B)$ ou $\forall \nu_1, \cdots, \nu_n D^{L+L'}[b]$

– et d'autre part $\forall \nu_1, \cdots, \nu_n E^{\alpha}[b]$ ou $\forall \nu_1, \cdots, \nu_n (T \to E^{\alpha}[b])$ ou $\forall \nu_1, \cdots, \nu_n$ $D^{L+\alpha}[b]$.

Et de $\widehat{\Gamma}, D^L[a] \Vdash b = a$, $\widehat{\Gamma}, E^\alpha[a] \Vdash b = a$ et $\widehat{\Gamma} \Vdash D^{L+\alpha}[a]$ on peut déduire (L$\vee E$) $\widehat{\Gamma} \Vdash b = a$.

Nous montrons ensuite, par récurrence sur la longueur du typage à transformer T, qu'un typage contenant une ou plusieurs utilisations «avec u variable principale» de la règle T$\vee E$ peut être transformé en un typage qui ne contient plus aucune utilisation de cette sorte (et que cette transformation préserve la clarté de la preuve).

Si la dernière règle du typage T n'est pas une utilisation «avec u variable principale» de la règle T$\vee E$, il suffit de transformer les preuves des prémisses (hypothèse de récurrence) et d'appliquer la dernière règle à ces prémisses qui sont sans utilisation «avec u variable principale» de la règle T$\vee E$. Remarquer que la clarté de T est préservée par ces transformations.

Le seul cas à étudier est donc celui où la fin du typage est :

si (a) $\Gamma, x : D^L[a] \vdash x : T$, si (b) $\Gamma, x : E^\alpha[a] \vdash x : T$ et si (c) $\Gamma \vdash v : D^{L+\alpha}[a]$ alors (d) $\Gamma \vdash u[x := v] = v : T$.

D'après le résultat ci-dessus $T = \forall y_1 \cdots y_n D^{L+\alpha+L'}[b]$ et $\widehat{\Gamma} \Vdash b = a$. Par hypothèse de récurrence, (c) peut être remplacée par une preuve $(c)'$ $\Gamma \vdash v : D^{L+\alpha}[a]$ sans utilisation «avec u variable principale» de la règle T$\vee E$. Il va nous suffire de poursuivre la preuve de (c)' sans utiliser la règle T$\vee E$.

En utilisant la règle T$\vee I2$, s'il le faut plusieurs fois, on obtient $\Gamma \vdash v : D^{L+\alpha+L'}[a]$. Ensuite, puisque aucun des $y_1, \cdots, y_n$ n'est libre dans Γ, on utilise, éventuellement plusieurs fois, la règle T$\forall I1$ ou T$\forall I2$ pour obtenir $\Gamma \vdash v : \forall y_1 \cdots y_n D^{L+\alpha+L'}[a]$. Il ne reste plus qu'à utiliser Teq pour obtenir $\Gamma \vdash v : \forall y_1 \cdots y_n D^{L+\alpha+L'}[b]$.

Remarquer que si T est claire, (c) est claire, donc, par hypothèse de récurrence, (c') l'est aussi et la preuve transformée de T est donc claire. □

5.1.2 Lemmes sur les termes redexeurs

Nous commençons par un résultat qui indique des types impossibles pour $\lambda x\, u$ ou $\varepsilon_\alpha\langle u\rangle$.

Lemme 5.1.5 *On ne peut obtenir aucun des typages suivants :*

- $\Gamma \vdash \lambda x\, u : \overrightarrow{\forall}\, X(t_1, \cdots, t_n)$, $\Gamma \vdash \lambda x\, u : \overrightarrow{\forall}\, E^\alpha[a]$
- $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \overrightarrow{\forall}\, X(t_1, \cdots, t_n)$.

Preuve Nous rédigeons seulement la preuve de la première partie du lemme, la seconde étant semblable. Cette preuve se fait par récurrence sur le typage, examinons donc la dernière règle.

- La dernière règle ne peut être ni Tvar, ni Texc, ni Ttry (le terme ne convenant pas) ni Tprop, ni T$\vee I1$, ni T$\vee I2$ (le type ne convenant pas).

- Si la règle est T$\forall E_1$, T$\forall E_2$,T$\forall I_1$, T$\forall I_2$ ou Teq, la preuve est directe par hypothèse de récurrence (remarquer que si l'on veut obtenir $\Gamma \vdash \lambda x\, u : \overrightarrow{\forall} X(x_1, \cdots, x_p)$ par application de la règle T$\forall E_2$, il faut partir de $\Gamma \vdash \lambda x\, u : \forall Y\, \overrightarrow{\forall} Y(y_1, \cdots, y_p)$.

- Pour traiter le cas de T$\vee E$, on peut supposer que la preuve est sans utilisation avec «avec u variable principale» de la règle T$\vee E$, et alors on sait que si le terme final «$w[y:=v]$» est $\lambda x\, u$, c'est que w est de la forme $\lambda x\, t$. Mais alors on devrait avoir $\Gamma, y : D^L[a] \vdash \lambda x\, t : \overrightarrow{\forall} X(t_1, \cdots, t_n)$ ou $\Gamma, y : D^L[a] \vdash \lambda x\, t : \overrightarrow{\forall} E^\alpha[a]$ ce qui est interdit par hypothèse de récurrence.

□

Nous montrons un lemme permettant de « remonter » le typage d'une exception.

Lemme 5.1.6 *Soit $\alpha = D^k$ une dénomination de type et $\nu_1, \cdots, \nu_n$ des variables d'individu ou de relation ($n \in N$).*

Si $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \forall \nu_1, \cdots, \nu_n E^\alpha[a]$ alors il existe b tel que $\Gamma \Vdash a = r_\alpha(b)$ et $\Gamma \vdash u : D[b]$.

Preuve Nous faisons la preuve par récurrence sur le typage, en considérant donc la dernière règle qui ne peut être ni Tvar, ni Tabs, ni Tapp, ni Ttry - le terme ne convenant pas - ni Tprop, ni T$\vee I1$, ni T$\vee I2$ - le type ne convenant pas.

- Si la dernière règle est T$\forall I_1$ ou T$\forall I_2$, l'hypothèse de récurrence permet de conclure.

- Si la dernière règle est T$\forall E_1$, on a de $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \forall z, \nu_1, \cdots, \nu_n E^\alpha[a]$ déduit $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \forall \nu_1, \cdots, \nu_n E^\alpha[a[z:=t]]$. L'hypothèse de récurrence nous dit qu'il existe b tel que $\Gamma \vdash u : D[b]$ et $\widehat{\Gamma} \Vdash a = r_\alpha(b)$.

  On en déduit $\Gamma \vdash u : \forall z D[b]$ puis $\Gamma \vdash u : D[b[z:=t]]$ par application successive de T$\forall I_1$ et T$\forall E_1$, z n'étant pas libre dans Γ ; et $\widehat{\Gamma} \Vdash a[z:=t] = r_\alpha(b[z:=t])$ en utilisant le lemme 3.4.4.

- Si la dernière règle est T$\forall E_2$, on remarque qu'on ne peut pas avoir dans la prémisse, d'après le lemme 5.1.5, le type $\overrightarrow{\forall} X(x_1, \cdots, x_n)$, on a donc du partir de $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \forall X, \nu_1, \cdots, \nu_n E^\alpha[a]$ et l'hypothèse de récurrence permet de conclure.

- Si la dernière règle est Teq, on a de $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \forall \nu_1, \cdots, \nu_n E^\alpha[a[z:=t_1]]$ et $\Gamma \Vdash t_1 = t_2$ déduit $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \forall \nu_1, \cdots, \nu_n E^\alpha[a[z:=t_2]]$. L'hypothèse de récurrence permet d'affirmer qu'il existe b tel que $\Gamma \vdash u : D[b]$ et $\widehat{\Gamma} \Vdash a[z:=t_1] = r_\alpha(b)$. Le lemme 3.4.4 nous donne $\widehat{\Gamma} \Vdash a[z:=t_1] = a[z:=t_2]$ et la transitivité de l'égalité permet de conclure.

- Si la dernière règle est Texc, le résultat est immédiat.

- Si la dernière règle est T$\vee E$, on n'a qu'un seul cas à examiner, puisqu'on peut supposer que la preuve a été transformée en une preuve sans utilisation «avec u variable principale» de T$\vee E$.

  On a $(a)\ \Gamma, x : F^L[c] \vdash \varepsilon_\alpha\langle w\rangle : \forall\nu_1, \cdots, \nu_n E^\alpha[a]$, $(b)\ \Gamma, x : E^\beta[c] \vdash \varepsilon_\alpha\langle w\rangle : \forall\nu_1, \cdots, \nu_n E^\alpha[a]$ et $(c)\ \Gamma \vdash v : F^{L+\beta}[c]$.

  - L'hypothèse de récurrence appliquée à (a) nous dit qu'il existe b' tel que (1) $\Gamma, x : F^L[c] \vdash w : D[b']$ et (2) $\widehat{\Gamma}, F^L[c] \Vdash a = r_\alpha(b')$.
  - L'hypothèse de récurrence appliquée à (b) nous dit qu'il existe b" tel que (3) $\Gamma, x : E^\beta[c] \vdash w : D[b'']$ et (4) $\widehat{\Gamma}, E^\beta[c] \Vdash a = r_\alpha(b'')$.
  - On déduit de (1) et (2) (casD) (1') $\Gamma, x : F^L[c] \vdash w : D[cas_\alpha(c, b'', b')]$ et (2') $\widehat{\Gamma}, F^L[c] \Vdash a = r_\alpha(cas_\alpha(c, b'', b'))$.
  - On déduit de (3) et (4) (casE) (3') $\Gamma, x : E^\beta[c] \vdash w : D[cas_\alpha(c, b'', b')]$ et (4') $\widehat{\Gamma}, E^\beta[c] \Vdash a = r_\alpha(cas_\alpha(c, b'', b'))$.

  On conclut en appliquant d'une part T$\vee E$ à (1'), (3') et (c), d'autre part L$\vee E$ à (2'), (4') et (c') $\widehat{\Gamma} \Vdash F^{L+\beta}[c]$ (déduit de (c) à l'aide du lemme 3.2.4).

□

Nous montrons ensuite un résultat qui permet de simplifier le type obtenu pour un terme redexeur.

Lemme 5.1.7 *Soient a un terme d'individu, D un symbole de données, α et β des dénominations de type, égales ou non, L un ensemble de dénominations de type, n un entier éventuellement nul, et $\nu_1, \cdots, \nu_n$ des variables d'individu ou de relation.*

- *Si* (1) $\Gamma \vdash \lambda x\, u : \forall\nu_1, \cdots, \nu_n D^L[a]$ *alors* (2) $\Gamma \vdash \lambda x\, u : Da$.
- *Si* (1) $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \forall\nu_1, \cdots, \nu_n E^\beta[a]$ *alors* $\beta = \alpha$ *et* (2) $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : E^\alpha[a]$.
- *Si* (1) $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \forall\nu_1, \cdots, \nu_n D^L[a]$ *alors* $\alpha \in L$ *et* (2) $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : E^\alpha[a]$.

Et - c'est la version forte du résultat - s'il existe une preuve claire de (1), alors il existe une preuve claire de (2).

Preuve Nous rédigeons seulement la preuve de la première partie du lemme, dans sa version forte.

Remarquons tout d'abord que, si $L = \emptyset$, on peut, à partir de $\Gamma \vdash \lambda x\, u : \forall\nu_1, \cdots, \nu_n D^L[a]$, en appliquant T$\forall E_1$ ou T$\forall E_2$, obtenir $\Gamma \vdash \lambda x\, u : \forall y_2 \cdots y_n\ D^L[a][y_1 := y_1]$ soit $\Gamma \vdash \lambda x\, u : \forall y_2 \cdots y_n D^L[a]$; et donc, après n étapes,

$\Gamma \vdash \lambda x\ u : D^L[a]$ qui est le résultat souhaité. On vérifie que si (1) est claire, (2) qui la prolonge avec quelques règles T$\forall E$ l'est aussi.

Dans la suite, on supposera donc que $L \neq \emptyset$. Nous faisons la preuve de la version forte de la première partie du lemme par récurrence sur le typage (1), qui peut être supposé sans utilisation «avec u variable principale» de la règle T$\vee E$ (lemme 5.1.4), en considérant donc la dernière règle.

- Si cette dernière règle est T$\forall I_1$, T$\forall I_2$, on applique l'hypothèse de récurrence et on obtient directement le résultat cherché qui est donc une preuve claire.

- Si la règle est T$\forall E_1$, on a de $\Gamma \vdash \lambda x\ u : \forall z, \nu_1, \cdots, \nu_n D^L[a]$ déduit $\Gamma \vdash \lambda x\ u : \forall \nu_1, \cdots, \nu_n D^L[a[z:=t]]$. L'hypothèse de récurrence nous donne $\Gamma \vdash \lambda x\ u : D[a]$ d'où $\Gamma \vdash \lambda x\ u : \forall z D[a]$ puis $\Gamma \vdash \lambda x\ u : D[a[z:=t]]$ par application successive de T$\forall I_1$ et T$\forall E_1$. Là aussi la clarté de la preuve est préservée.

- Si la règle est T$\forall E_2$, il suffit de remarquer qu'on ne peut pas partir, d'après le lemme 5.1.5, de $\overrightarrow{\forall} X(x_1, \cdots, x_n)$, on a donc du partir de $\Gamma \vdash \lambda x\ u : \forall X, \nu_1, \cdots, \nu_n D^L[a]$ et l'hypothèse de récurrence permet de conclure.

- Si la règle est Teq, on a de $\Gamma \vdash \lambda x\ u : \forall \nu_1, \cdots, \nu_n D^L[a[z:=u]]$ et $\Gamma \Vdash u = v$ déduit $\Gamma \vdash \lambda x\ u : \forall \nu_1, \cdots, \nu_n D^L[a[z:=v]]$. L'hypothèse de récurrence donne $\Gamma \vdash \lambda x\ u : D[a[z:=u]]$ et il suffit d'appliquer Teq.

- La règle T$\vee I1$ n'est pas utilisable d'après le lemme précédent et le cas de T$\vee I2$ est trivial.

- Si la règle est T$\vee E$, puisque la preuve (1) est sans utilisation «avec u variable principale» de T$\vee E$, nous n'avons qu'un cas à examiner.

  On a (a) $\Gamma, x : F^{L'}[b] \vdash \lambda y\ u : \forall \nu_1, \cdots, \nu_n D^L[a]$, (b) $\Gamma, x : E^{\alpha}[b] \vdash \lambda y\ u : \forall \nu_1, \cdots, \nu_n D^L[a]$ et (c) $\Gamma \vdash v : F^{L'+\alpha}[b]$. L'hypothèse de récurrence appliquée à (a) et (b) donne (a') $\Gamma, x : F^{L'}[b] \vdash \lambda y\ u : D[a]$, (b') $\Gamma, x : E^{\alpha}[b] \vdash \lambda y\ u : D[a]$. Et l'application de T$\vee E$ à (a'), (b') et (c) donne le résultat cherché.

  Remarquer que, si la preuve (1) est claire, d'une part elle peut être transformée en preuve claire sans utilisation «avec u variable principale» de T$\vee E$ (lemme 5.1.4), d'autre part que v n'est pas redexeur et donc puisque, par hypothèse de récurrence, (a') et (b') sont claires, on a bien obtenu une preuve (2) claire.

□

5.1.3 Lemme de substitution dans le terme

Nous montrons une version forte d'un résultat connu sous le nom de lemme de substitution dans le terme.

Lemme 5.1.8 (Substitution dans le terme) *Soit* $\Gamma' \stackrel{def}{=} \Gamma, x_1{:}A_1, \cdots, x_n{:}A_n$ *un environnement, si* (1) $\Gamma' \vdash t : A$ *et si* (2i) $\Gamma \vdash t_i : A_i (1 \leq i \leq n)$, *alors* (3) $\Gamma \vdash t[x_1{:=}t_1, \cdots, x_n{:=}t_n] : A$. *Et - c'est la version forte du résultat - s'il existe des preuves claires de (1) et* (2i) $(1 \leq i \leq n)$, *alors on peut construire une preuve claire de (3).*

Preuve Nous prouvons directement la version forte du lemme par récurrence sur le couple (p, Q) où p est la longueur de la preuve (1) et $Q = \sum_{i=1}^{n} q_i$ avec q_i longueur de la preuve (2i). On examine la dernière règle de (1). Pour tout terme t, on note t' le terme $t[x_1{:=}t_1, \cdots, x_n{:=}t_n]$.

- Si la dernière règle est Tvar, on distingue deux cas.
  - Si t est l'une des variables $x_1, \cdots, x_n$ soit x_k, alors $A = A_k$ et le résultat (3) $\Gamma \vdash t_k : A_k$ est identique à l'hypothèse (2k) et donc clair.
  - Sinon $t = y$ et alors l'hypothèse (1) indique que la déclaration $y : A$ est dans Γ, le résultat $\Gamma \vdash y[x_1{:=}t_1, \cdots, x_n{:=}t_n] = y : A$ s'obtient directement par Tvar, il est donc clair.
- Si c'est Tabs ou Tapp, il suffit d'appliquer l'hypothèse de récurrence puis la même règle. Par exemple dans le cas de Tapp, on a $\Gamma' \vdash u : B \to A, \Gamma' \vdash v : B$ et donc $\Gamma' \vdash (u\ v) : A$ d'où, par hypothèse de récurrence, $\Gamma \vdash u' : B \to A$ clair et $\Gamma \vdash v' : B$ clair et il suffit alors d'appliquer la règle Tapp pour obtenir une preuve claire de (3).
- Si c'est T$\forall E_1$ ou T$\forall E_2$, alors on a comme prémisse $\Gamma' \vdash u : \forall \nu A$ et donc, par hypothèse de récurrence, $\Gamma \vdash u' : \forall \nu A$ clair et il suffit d'appliquer la même règle pour conclure.
- Si c'est T$\forall I_1$ ou T$\forall I_2$, alors la prémisse est $\Gamma' \vdash u : A$; par hypothèse de récurrence $\Gamma \vdash u' : A$ clair et là aussi la même règle donne le résultat.
- si c'est Teq, on a $\Gamma' \vdash u : A[y{:=}a]$ et $\widehat{\Gamma'} \Vdash a = b$. L'hypothèse de récurrence nous donne $\Gamma \vdash u' : A[y{:=}a]$ clair et de (2i) je déduis $\widehat{\Gamma} \Vdash A_i (1 \leq i \leq n)$, donc $\widehat{\Gamma} \Vdash a = b$, et il suffit d'appliquer la règle Teq pour conclure.
- Dans le cas des règles Texc, Tprop, Ttry, T$\vee I1$, T$\vee I2$ le résultat est évident en appliquant l'hypothèse de récurrence puis la même règle.

- Reste la règle T$\vee E$. On avait

  $$(a)\Gamma', y : D^L[a] \vdash u : A;\ (b)\Gamma', y : E^\alpha[a] \vdash u : A;\ (c)\Gamma' \vdash v : D^{L+\alpha}[a]$$

  L'hypothèse de récurrence nous donne

  $$(a')\Gamma, y : D^L[a] \vdash u' : A;\ (b')\Gamma, y : E^\alpha[a] \vdash u' : A;\ (c')\Gamma \vdash v' : D^{L+\alpha}[a]$$

  et on sait que ces trois preuves sont claires.

  L'application de T$\vee E$ à ces trois typages donne $\Gamma \vdash u'[y{:=}v'] : A$ avec $u'[y{:=}v'] = (u[x_1{:=}t_1, \cdots, x_n{:=}t_n])[y{:=}v[x_1{:=}t_1, \cdots, x_n{:=}t_n]] = (u[y{:=}v][x_1{:=}t_1, \cdots, x_n{:=}t_n])$ puisque d'après (2) x_i n'est pas libre dans t_i $(1 \leq i \leq n)$.

  Mais l'application de la règle T$\vee E$ peut rendre (3) trouble, ce qui nous oblige à distinguer deux cas.

  - Si dans (c), v n'est égal à aucun des $x_1, \cdots, x_n$ ou si $v = x_k$, t_k n'étant pas redexeur alors, puisque v ne peut être lui même redexeur (car (1) est claire), v' ne sera pas redexeur et l'application de T$\vee E$ à (a'), (b'), (c') sera profitable.
  - Dans l'autre cas, $v = x_k$ et $t_k = \lambda x\, u$ (ou $\varepsilon_\alpha\langle u\rangle$). On peut appliquer l'hypothèse de récurrence à (c) et $(2i)$, $(1 \leq i \leq n)$ puisque la longueur de (c) est inférieure à celle de (1). On a donc une preuve claire de (d) $\Gamma \vdash x_k[x_1{:=}t_1, \cdots, x_n{:=}t_n] = t_k : D^{L+\alpha}[a]$. Distinguons encore deux cas.
    - Si $t_k = \lambda x\, u$, le lemme 5.1.7 nous permet de déduire de (d) une preuve de (e) $\Gamma \vdash t_k : D[a]$ et d'affirmer que (e) est claire. On utilise ensuite la règle T$\vee I2$ pour obtenir (f) $\Gamma \vdash t_k : D^L[a]$ claire aussi. Et on conclut en appliquant l'hypothèse de récurrence à (a), $(2i)$, $(1 \leq i \leq n)$ et (f) ce qui donne une preuve claire de $\Gamma \vdash u[x_1{:=}t_1, \cdots, x_n{:=}t_n, y{:=}t_k] : A$ où l'on reconnait (3).
    - Si $t_k = \varepsilon_\alpha\langle u\rangle$, le même lemme nous permet de déduire de (d) $(f')\Gamma \vdash t_k : E^\alpha[a]$ et d'affirmer que (f') est claire. On conclut alors en appliquant l'hypothèse de récurrence à (b), $(2i)$, $(1 \leq i \leq n)$ et (f') ce qui donne une preuve claire de $\Gamma \vdash u[x_1{:=}t_1, \cdots, x_n{:=}t_n, y{:=}t_k] : A$ où l'on reconnait (3).

  □

Remarque 5.1.9 *Le résultat « si $\Gamma, x : B \vdash t : A$ et si x n'est pas libre dans t, alors* (3) *$\Gamma \vdash t : A$ », qui est vrai dans AF2, n'est plus vrai dans Ex2 .*

Preuve La signification intuitive de ce résultat est la suivante : s'il n'y a pas trace de l'hypothèse B dans le terme t - trace de la preuve de A -, alors je peux prouver A, toujours avec comme trace t, en excluant l'hypothèse B de mon contexte. Mais dans notre système, une hypothèse peut être utilisée sans qu'on en garde trace dans le terme : c'est le cas dans la règle Teq.

Voici un exemple. Soient $\alpha = N_1$ et $\beta = N_2$ deux dénominations de type (différentes), et soit $a = r_\alpha(\mathbf{0})$. On a (1) $x : E^\beta[a], y : N^\beta[t_\alpha(a)] \vdash y : N^\beta[a]$
et on n'a pas (2) $y : N^\beta[t_\alpha(a)] \vdash y : N^\beta[a]$.

L'affirmation (1) se déduit de $x : E^\beta[a], y : D^\beta[t_\alpha(a)] \vdash y : D^\beta[t_\alpha(a)]\ (Tvar)$, par application de la règle Teq, puisque $E^\beta[a] \Vdash D^\beta[a]$ $(T \vee I1)$ et $\Vdash D^\beta[a] \to t_\alpha(a) = a$ $(trapD)$.

Prouvons pour finir que l'on n'a pas $(2) y : D^\beta[t_\alpha(a)] \vdash y : D^\beta[a]$.

- De $\Vdash N[0]$ on déduit (trapE) $\Vdash t_\alpha(a) = \mathbf{0}$ donc (Leq) $\Vdash N[t_\alpha(a)]$ puis (L$\vee I2$) (1) $\Vdash N^\beta[t_\alpha(a)]$.
- On n'a pas $\Vdash N^\beta[a]$ car sinon on aurait (Ltry) $\Vdash N[t_\beta(a)]$ et, puisque $\Vdash N^\alpha[a]$ et (trapD) $\Vdash N^\alpha[a] \to t_\beta(a) = a$, on aurait (Leq) $\Vdash N[a]$ et nous verrons plus loin (théorème de caractérisation des entiers 6.1.6) que c'est impossible.

On n'a donc pas $N^\beta[t_\alpha(a)] \Vdash N^\beta[a]$. Et on ne peut donc pas avoir (lemme 3.2.4) $y : N^\beta[t_\alpha(a)] \vdash y : N^\beta[a]$. □

5.1.4 Clarification des preuves

Nous montrons que toute preuve peut être rendue claire.

Proposition 5.1.10 *S'il existe une preuve de $\Gamma \vdash t : A$ alors il existe une preuve de $\Gamma \vdash t : A$ qui est claire.*

Preuve Si nous supposons, hypothèse de récurrence, que toute preuve de longueur inférieure ou égale à n est transformable en preuve claire, alors considérons une preuve de longueur $n+1$ et montrons qu'elle aussi peut être rendue claire.

Si la dernière règle utilisée n'est pas T$\vee E$ ou si c'est T$\vee E$ utilisée de façon profitable, alors l'hypothèse de récurrence permet de conclure.

Sinon, on a $v = \lambda x\, u$ (ou $\varepsilon_\alpha\langle u\rangle$) et

$$(a)\ \Gamma, x : D^L[a] \vdash u : T;\ (b)\ \Gamma, x : E^\alpha[a] \vdash u : T;\ (c)\ \Gamma \vdash v : D^{L+\alpha}[a]$$

Par hypothèse de récurrence ces preuves peuvent être transformées en preuves claires (a'), (b') et (c'). Le lemme 5.1.7 nous permet d'obtenir, à partir

de (c') une preuve claire (d) $\Gamma \vdash v : D[a]$ (ou $(e')\Gamma \vdash v : E^{\alpha}[a]$), puis, dans le cas où $v = \lambda x\, u$, par utilisation de la règle T$\vee I2$, une preuve claire (e) $\Gamma \vdash v : D^{L}[a]$. Et le lemme de substitution dans le terme (5.1.8), appliqué à (a') (ou (b')) et (e) (ou (e')) toutes les deux claires nous donne une preuve claire de $\Gamma \vdash u[x{:=}v] : T$. □

5.2 Lemmes divers

Nous montrons tout d'abord deux lemmes de substitution dans les formules.

Lemme 5.2.1 *Soient u un terme d'individu, x une variable d'individu, X une variable de relation n-aire et F une formule.*

1. *Si* (1) $\Gamma \vdash t : A$, *alors* (2) $\Gamma[x{:=}u] \vdash t : A[x{:=}u]$
2. *Si(1)* $\Gamma \vdash t : A$, *alors* (2) $\Gamma[X(x_1, \cdots, x_n){:=}F] \vdash t : A[X(x_1, \cdots, x_n){:=}F]$

Et, si le typage (1) a une longueur l, alors il existe un typage de (2) qui est aussi de longueur l.

Preuve Les deux parties du lemme se prouvent par récurrence sur le typage (1), en examinant donc la dernière règle.

Ecrivons la preuve de la première partie du lemme, en reprenant la preuve de [10] et en la complétant dans le cas des nouvelles règles.

Posons $\Gamma' \stackrel{def}{=} \Gamma[x{:=}u]$.

- Le résultat est immédiat si la dernière règle est Tvar, Tabs, Tapp, TExc, T$\vee I1$, T$\vee I2$, Tprop, Ttry ou T$\vee E$.
- Si la dernière règle est T$\forall I_1$, on a, de $\Gamma \vdash t : A'$, déduit $\Gamma \vdash t : A \stackrel{def}{=} \forall y A'$ (y étant une variable d'individu non libre dans Γ). L'hypothèse de récurrence nous donne $\Gamma' \vdash t : A'[x{:=}u]$, et on appplique T$\forall I_1$ pour obtenir $\Gamma' \vdash t : \forall y A'[x{:=}u]$ ce qui est le résultat cherché (on peut supposer que $y \notin u$).
- Si la dernière règle est T$\forall E_1$, on a, de $\Gamma \vdash t : \forall y A'$, déduit $\Gamma \vdash t : A \stackrel{def}{=} A'[y{:=}v]$. L'hypothèse de récurrence nous donne $\Gamma' \vdash t : \forall y A'[x{:=}u]$, donc $\Gamma' \vdash t : A'[x{:=}u][y{:=}v']$ (T$\forall E_1$). En choisissant $v' = v[x{:=}u]$, on obtient $\Gamma' \vdash t : A'[x{:=}u][y{:=}v[x{:=}u]] = A'[y{:=}v][x{:=}u] = A[x{:=}u]$ puisque y n'appartient pas à u.
- Si la dernière règle est T$\forall E_2$, on a, de $\Gamma \vdash t : \forall X A'$, déduit $\Gamma \vdash t : A \stackrel{def}{=} A'[X\, x_1, \cdots, x_n{:=}F]$. L'hypothèse de récurrence nous donne $\Gamma' \vdash t : \forall X A'[x{:=}u]$, donc $\Gamma' \vdash t : A'[x{:=}u][X\, x_1, \cdots, x_n{:=}F]$ (T$\forall E_2$) soit

$\Gamma' \vdash t : A'[X\,x_1, \cdots, x_n := F][x := u]$ car $x_1, \cdots, x_n$ peuvent être choisis non dans u.

- Le cas où la dernière règle est T$\forall I_2$ se traite comme celui de la règle T$\forall I_1$.

- Si la dernière règle est Teq, on a, de $\Gamma \vdash t : A'[y := v]$ et $\widehat{\Gamma} \Vdash v = w$, déduit $\Gamma \vdash t : A'[y := w]$. L'hypothèse de récurrence nous donne $\Gamma' \vdash t : A'[y := v][x := u]$ et $A'[y := v][x := u] = A'[x := u][y := v]$ car y peut être choisi non dans u. De $\widehat{\Gamma} \Vdash v = w$ on déduit (hypothèse de récurrence partie logique) $\widehat{\Gamma}[x := u] \Vdash v[x := u] = w[x := u]$. On conclut, à l'aide de la règle Teq, que $\Gamma' \vdash t : A'[x := u][y := w[x := u]]$ soit $\Gamma' \vdash t : A'[y := w][x := u]$.

La preuve de la seconde partie du lemme est la même sauf dans le cas où la dernière règle est T$\forall E_1$ que nous examinons.

On a, de $\Gamma \vdash t : \forall Y\, A'$, déduit $\Gamma \vdash t : A \stackrel{def}{=} A'[Y\,y_1, \cdots, y_p := G]$. L'hypothèse de récurrence nous donne $\Gamma[X\,x_1, \cdots, x_n := F] \vdash t : \forall Y\, A'[X\,x_1, \cdots, x_n := F]$ donc (T$\forall E_1$) $\Gamma[X\,x_1, \cdots, x_n := F] \vdash t : A'[X\,x_1, \cdots, x_n := F][Y\,y_1, \cdots, y_p := G[X\,x_1, \cdots, x_n := F]]$ c'est à dire $\Gamma[X\,x_1, \cdots, x_n := F] \vdash t : A'[Y\,y_1, \cdots, y_p := G][X\,x_1, \cdots, x_n := F]$ puisque Y n'est pas libre dans F. □

Mais le remplacement, dans le type, d'un terme d'individu par un terme qui lui est égal ne peut pas être aussi simple que dans AF2, puisque notre égalité est conditionnelle.

Lemme 5.2.2 *Soient u et v des termes d'individu, x une variable d'individu, Γ et Δ deux contextes.*

Si $\Gamma, \Delta[x := u] \vdash t : A[x := u]$ *et* $\widehat{\Gamma} \Vdash u = v$, *alors* $\Gamma, \Delta[x := v] \vdash t : A[x := v]$.

On peut remarquer que, si $\Delta = \emptyset$, on retrouve le lemme 3 de AF2 ([10] page 144).

Preuve On peut prouver directement ce lemme à partir du lemme 5.2.1.

Posons $\Delta = y_1 : B_1, \cdots, y_n : B_n$. De $\Gamma, \Delta[x := u] \vdash t : A[x := u]$, on déduit (Teq) $\Gamma, \Delta[x := u] \vdash t : A[x := v]$, soit $\Gamma, y_1 : B_1[x := u], \cdots, y_n : B_n[x := u] \vdash t : A[x := v]$ ou par renommage $\Gamma, z_1 : B_1[x := u], \cdots, z_n : B_n[x := u] \vdash t[y_1 := z_1, \cdots, y_n := z_n] : A[x := v]$ et donc par affaiblissement
(1) $\Gamma, y_1 : B_1[x := v], \cdots, y_n : B_n[x := v],\ z_1 : B_1[x := u], \cdots, z_n : B_n[x := u] \vdash t[y_1 := z_1, \cdots, y_n := z_n] : A[x := v]$.

Mais (2i) $\Gamma, y_i : B_i[x := v] \vdash y_i : B_i[x := u]$ (Tvar puis Teq). Il suffit alors d'appliquer le lemme de substitution dans le terme à (1) et (2i) ($1 \leq i \leq n$) pour obtenir $\Gamma, y_1 : B_1[x := v], \cdots, y_n : B_n[x := v] \vdash t[y_1 := z_1, \cdots, y_n := z_n][z_1 := y_1, \cdots, z_n := y_n] = t : A[x := v]$. □

Nous montrons enfin un résultat qui permet de remonter la preuve lorsque le type d'un terme redexeur est un type «flèche».

Lemme 5.2.3 *Soient A et B deux formules quelconques.*

1. *Si* $\Gamma \vdash \lambda x\, u : \overrightarrow{\forall}(A \to B)$ *alors* $\Gamma, x : A \vdash u : B$.
2. *Si* $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : \overrightarrow{\forall}(A \to B)$ *alors* $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : B$.

Preuve Nous montrons la première partie du lemme, par récurrence sur le typage, en examinant donc la dernière règle.

- La dernière règle ne peut être ni Tvar, ni Texc, ni Tapp, ni Ttry puisque le terme typé est $\lambda x\, u$, ni T$\vee I1$, ni T$\vee I2$ le type ne convenant pas.
- Si c'est l'une des règles Tabs, T$\forall I_1$, T$\forall I_2$, le résultat est trivial. Et ce ne peut être la règle Tprop car sinon on aurait $\Gamma \vdash \lambda x\, u : E^\alpha[a]$ ce que le lemme 5.1.5 interdit.
- Si la dernière règle est T$\forall E_1$, on a, de $\Gamma \vdash \lambda x\, u : \forall z\, \overrightarrow{\forall}(A \to B)$, déduit $\Gamma \vdash \lambda x\, u : \overrightarrow{\forall}(A[z:=t] \to B[z:=t])$. L'hypothèse de récurrence nous donne $\Gamma, x : A \vdash u : B$, et le lemme 5.2.1 nous permet d'en déduire $\Gamma[z:=t], x : A[z:=t] \vdash u : B[z:=t]$, ce qui est bien le résultat cherché puisque z n'est pas libre dans Γ et donc $\Gamma[z:=t] = \Gamma$.
- Si la dernière règle est T$\forall E_2$, on a, de $\Gamma \vdash \lambda x\, u : \forall X C$, déduit $\Gamma \vdash \lambda x\, u : \overrightarrow{\forall}(A \to B)$. On sait donc que C est de l'une des deux formes $\overrightarrow{\forall} Y(y_1 \cdots y_n)$ ou $\overrightarrow{\forall}(A' \to B')$, mais, d'après le lemme 5.1.5, la première n'est pas possible. On a donc $\Gamma \vdash \lambda x\, u : \forall X (A' \to B')$, avec $A'[X(x_1, \cdots, x_n) := F] = A$ et $B'[X(x_1, \cdots, x_n) := F] = B$ ce qui par hypothèse de récurrence donne $\Gamma, x : A' \vdash u : B'$. Le lemme 5.2.1 donne alors $\Gamma[X(x_1, \cdots, x_n) := F], x : A'[X(x_1, \cdots, x_n) := F] \vdash u : B'[X(x_1, \cdots, x_n) := F]$ ce qui est le résulat cherché puisque X n'est pas libre dans Γ et donc $\Gamma[X(x_1, \cdots, x_n) := F] = \Gamma$.
- Si la dernière règle est Teq, on a, de $\Gamma \vdash \lambda x\, u : \overrightarrow{\forall}(A[y:=a] \to B[y:=a])$ et $\widehat{\Gamma} \Vdash a = b$, déduit $\Gamma \vdash \lambda x\, u : \overrightarrow{\forall}(A[y:=b] \to B[y:=b])$. L'hypothèse de récurrence nous donne $\Gamma, x : A[y:=a] \vdash u : B[y:=a]$ et le lemme 5.2.2 permet là aussi de conclure que $\Gamma, x : A[y:=b] \vdash u : B[y:=b]$.
- Si la dernière règle est T$\vee E$, nous avons un seul cas à examiner puisqu'on peut supposer que la preuve est sans utilisation de T$\vee E$ «avec u variable principale». On a donc, de $\Gamma, z : D^L[c] \vdash \lambda x\, w : T$ et $\Gamma, z : E^\alpha[c] \vdash \lambda x\, w : T$ et aussi $\Gamma \vdash v : D^{L+\alpha}[c]$, déduit $\Gamma \vdash \lambda x\, w[y:=v] : T$, avec $T = \overrightarrow{\forall}(A \to B)$. L'hypothèse de récurrence donne $\Gamma, z : D^L[c], x : A \vdash w : B$ et $\Gamma, z : E^\alpha[c], x : A \vdash w : B$ puis, à l'aide de la règle T$\vee E$, on obtient $\Gamma, x : A \vdash w[y:=v] : B$.

La preuve de la deuxième partie du lemme est sans difficulté.

- La dernière règle ne peut être ni Tvar, ni Tabs, ni Tapp, ni Ttry puisque le terme typé est $\varepsilon_\alpha\langle u\rangle$, ni Texc, ni T$\vee I1$, ni T$\vee I2$ le type ne convenant pas.
- Le cas des règles T$\forall E_1$, T$\forall E_2$, T$\forall I_1$, T$\forall I_2$, Teq et T$\vee E$ se traite comme dans la première partie.
- Si la dernière règle est Tprop, on a obtenu $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : T \to E^\alpha[a]$ à partir de $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : E^\alpha[a]$ et le lemme est bien vérifié.

□

5.3 Conservation du type

Nous commencerons par pointer une difficulté que pose la règle T$\vee E$.

Remarque 5.3.1 *Dans le terme $u[x:=v]$ (règle T$\vee E$), le terme v peut être dupliqué. Un redex de v peut donc être dupliqué, et ces instances d'un même redex, qui proviennent de la même preuve (celle de v), doivent être traitées de la même façon, donc réduites simultanément, si on veut que tout se passe bien, en particulier que le type se conserve par réduction.*

Ce problème est évoqué dans [1] où l'exemple suivant, du à Pierce ([22]), est proposé.

Quatre termes sont examinés :

$t \stackrel{def}{=} \lambda x\,\lambda y\,\lambda z\,(x\ (\lambda t\,t\ (y\ z))\ (\lambda t\,t\ (y\ z)))$ et $u \stackrel{def}{=} \lambda x\,\lambda y\,\lambda z\,(x\ (y\ z)\ (y\ z))$

$t' \stackrel{def}{=} \lambda x\,\lambda y\,\lambda z\,(x\ (\lambda t\,t\ (y\ z))\ (y\ z))$

et $t'' \stackrel{def}{=} \lambda x\,\lambda y\,\lambda z\,(x\ (y\ z)\ (\lambda t\,t\ (y\ z)))$.

On vérifie que $t \to_\beta t' \to_\beta u$ et $t \to_\beta t'' \to_\beta u$.

Le système de typage utilisé est un système simple utilisant les seuls connecteurs $\wedge$, $\vee$ et $\to$, et dans lequel la règle T$\vee E$ est présente (sous une forme plus générale que dans $Ex2$).

Dans ce système, on type facilement t et u de type :

$(\sigma \to \sigma \to \tau) \wedge (\rho \to \rho \to \tau) \to (\mu \to (\sigma \vee \rho)) \to \mu \to \tau$.

Mais on ne peut pas typer t' ou t". Le type n'est pas conservé : ni par β-réduction, ni par β-expansion.

Dans $Ex2$ le problème se pose de la même manière : examinons l'exemple suivant.

Soit $v \stackrel{def}{=} (z\ (\lambda t\,t\ \varepsilon_\alpha\langle\underline{0}\rangle)\ \underline{1})$ et $v' \stackrel{def}{=} (z\ \varepsilon_\alpha\langle\underline{0}\rangle\ \underline{1})$.

On a évidemment $v \to_\beta v'$ donc $(v\ v) \to_\beta (v\ v') \to_\beta (v'\ v')$ et $(v\ v) \to_\beta (v'\ v) \to_\beta (v'\ v')$.

On peut typer les termes $(v\ v)$ et $(v'\ v')$:

- dans le contexte $z : B[z], y : N[x]$, le terme $(y\ y)$ a le type $N[f(x)]$ (où f est la fonction $x \to x^x$) donc le type $N^\alpha[f(x)]$ (T$\vee I2$), donc le type $N^\alpha[cas_\alpha(x, x, f(x))]$ (casD) ;
- dans le contexte $z : B[z], y : E^\alpha[x]$, le terme $(y\ y)$ a le type $E^\alpha[x]$ donc le type $N^\alpha[x]$ (T$\vee I1$), donc le type $N^\alpha[cas_\alpha(x, x, f(x))]$ (casE) ;
- mais $z : B[z] \vdash v : N^\alpha[t]$ avec $t \stackrel{def}{=} Si(z, r_\alpha(\mathbf{0}), \boldsymbol{s}(\mathbf{0}))$ (la fonction Si est l'alternative bien connue définie par $Si(\boldsymbol{v}, a, b) = a$ et $Si(\boldsymbol{f}, a, b) = b$), et de même $z : B[z] \vdash v' : N^\alpha[t]$;
- on en déduit, en utilisant la règle T$\vee E$, $z : B[z] \vdash (v\ v) : N^\alpha[cas_\alpha(t, t, f(t))]$ et aussi $z : B[z] \vdash (v'\ v') : N^\alpha[cas_\alpha(t, t, f(t))]$.

Il ne semble pourtant pas possible (nous n'en avons pas encore la preuve) de typer $(v\ v')$ ou $(v'\ v)$. Essayons de préciser où se situe la difficulté.

- Il est possible de typer $(x\ x)$ dans le contexte $x : N^\alpha[x]$ (comme cela a été fait ci-dessus), seuls deux cas sont à examiner :
  - dans le contexte $x : N[x]$, le terme $(x\ x)$ a le type $N^\alpha[cas_\alpha(x, x, f(x))]$
  - dans le contexte $x : E^\alpha[x]$, le terme $(x\ x)$ a le type $N^\alpha[cas_\alpha(x, x, f(x))]$
  - donc (lemme 3.4.2 dans le contexte $x : N^\alpha[x]$ le terme $(x\ x)$ a le type $N^\alpha[cas_\alpha(x, x, f(x))]$
- Nous conjecturons qu'il n'est pas possible de typer $(x\ y)$ dans le contexte $x : N^\alpha[x], y : N^\alpha[y]$. A défaut d'une preuve, examinons quelques arguments informels.

  La seule possibilité de typage est l'utilisation de la règle T$\vee E$, c'est à dire la recherche d'un même type dans les deux contextes (1) $x : N[x], y : N^\alpha[y]$ et (2) $x : E^\alpha[x], y : N^\alpha[y]$. Et le typage dans le contexte (1) est lui aussi nécessairement ramené au typage dans les deux contextes (a) $x : N[x], y : E^\alpha[y]$ et (b) $x : N[x], y : N[y]$.
  - Mais, si $(x\ y)$ est typable dans le contexte (a) de type T, alors le lemme de substitution dans le terme (5.1.8) nous autorise à affirmer que $\vdash t_1 \stackrel{def}{=} (\underline{1}\ \varepsilon_\alpha\langle\underline{0}\rangle) : T[x{:=}\boldsymbol{s}(\mathbf{0}), y{:=}r_\alpha(\mathbf{0})]$
  - De même, si $(x\ y)$ est typable dans le contexte (b) de type T, alors $\vdash t_2 \stackrel{def}{=} (\underline{1}\ \underline{0}) : T[x{:=}\boldsymbol{s}(\mathbf{0}), y{:=}\mathbf{0}]$.
  - Mais $t_1 \to_\delta t'_1 \stackrel{def}{=} \lambda x\, \varepsilon_\alpha\langle\underline{0}\rangle$, et $t_2 \to_\delta t'_2 \stackrel{def}{=} \underline{0}$. Or nous savons (théorème 6.1.8) que t'_1 ne peut avoir pour type $N^L[a]$. Donc, si le type se conserve par réduction, le type T ne peut être un type $N^L[t]$. Mais le raisonnement n'est pas complet puisque les deux termes t'_1 et t'_2 peuvent avoir le même type $A \to (E^\alpha[r_\alpha(\mathbf{0})] \to E^\alpha[r_\alpha(\mathbf{0})])$.

- Nous conjecturons même qu'il n'est pas possible de typer $(x\ y)$ dans le contexte $x : N^\alpha[z], y : N^\alpha[z]$. Examinons là aussi quelques arguments informels.

  Dans ce nouveau contexte, la difficulté précédente ne devrait plus se présenter : on ne devrait plus avoir à prendre en compte le cas $x : N[z], y : E^\alpha[z]$ puisqu'il n'est pas possible d'avoir $\vdash t_1 : N[a]$ et $\vdash t_2 : E^\alpha[a]$.

  Les seuls cas à prendre en compte, comme dans le typage du terme $(x\ x)$ dans le contexte $x : N^\alpha[x]$, devraient être $x : N[z], y : N[z]$ et $x : E^\alpha[z], y : E^\alpha[z]$. Malheureusement nous conjecturons que les règles de typage ne permettent pas d'exprimer les considérations précédentes et que $(x\ y)$ ne peut pas plus être typé dans le contexte $x : N^\alpha[z], y : N^\alpha[z]$ que dans le contexte $x : N^\alpha[x], y : N^\alpha[y]$.

- Le typage de $(x\ x)$ dans le contexte $x : N^\alpha[x]$ permet le typage de $(v\ v)$ et $(v'\ v')$ dans le contexte $z : B[z]$. Et l'impossibilité de typer $(x\ y)$ dans le contexte $x : N^\alpha[z], y : N^\alpha[z]$ explique l'impossibilité de typer $(v\ v')$ dans le contexte $z : B[z]$.

Comme dans [1], nous résoudrons la difficulté en traitant de façon uniforme les redex de v dupliqués dans $u[x := v]$, c'est à dire en remplaçant la δ-réduction par une forme de réduction parallèle.

5.3.1 Réduction parallèle

Nous définissons une forme de réduction parallèle (ou uniforme), dans laquelle toutes les instances d'un même redex sont réduites ensemble. Précisons d'abord le vocabulaire.

Définition 5.3.2 *Notons S(u) l'ensemble des sous-termes d'un terme u de* Λ_E. *Un sous ensemble E de S(u) est dit primaire (sur u) si* $\forall x \in E,\ \forall y \in S(u)$
$(x \simeq_\alpha y \leftrightarrow y \in E)$ *(la notation* $\simeq_\alpha$ *désigne la relation d'*α*-équivalence entre termes).*

On remarque que si un sous ensemble primaire n'est pas vide c'est une classe d'équivalence pour la relation $\simeq_\alpha$.

Nous énonçons maintenant quelques propriétés triviales (que donc nous ne démontrerons pas) des ensembles primaires.

Lemme 5.3.3 *Si E est un ensemble primaire sur le terme* $\lambda x\, u$ *(ou* $\varepsilon_\alpha\langle u\rangle$*) alors* $E \cap S(u)$ *est primaire sur u.*

Si E est un ensemble primaire sur le terme $(u\ v)$ *(ou* $\tau_\alpha\langle u, v\rangle$*) alors* $E \cap S(u)$ *est primaire sur u et* $E \cap S(v)$ *primaire sur v.*

Nous pouvons maintenant définir notre réduction parallèle.

Définition 5.3.4 *Soit t un λ-terme . Nous dirons que $t \|_0 t'$ si t' est obtenu à partir de t en appliquant la relation $\boldsymbol{\delta_0}$ à chacun des éléments d'un ensemble primaire E (sur t) de redex de t.*

La cloture transitive de la relation $\|_0$ est notée $\|$ et sera dite réduction parallèle.

Il faut noter que les éléments de l'ensemble primaire E sont disjoints puisque α-équivalents. Donc réduire par $\boldsymbol{\delta_0}$ chacun des redex de E a un sens et l'ordre dans lequel cette réduction est faite est sans influence sur le terme obtenu.

On remarque qu'avec notre définition la relation $\|_0$ est réflexive puisqu'un ensemble primaire de redex peut être vide, et donc que $\|$ est aussi réflexive.

Montrons quelques propriétés de la réduction parallèle.

Lemme 5.3.5 *Soient deux λ-termes t et t' tels que $t \|_0 t'$.*

1. *Si $t = \lambda x\ u$ alors $t' = \lambda x\ v$ et $u \|_0 v$.*
2. *Si $t = \varepsilon_\alpha\langle u\rangle$ alors $t' = \varepsilon_\alpha\langle v\rangle$ et $u \|_0 v$.*
3. *Si $t = (u\ v)$ alors :*
   - *soit $t' = (u'\ v')$ avec $u \|_0 u'$ et $v \|_0 v'$;*
   - *soit $u = \lambda x\ w$ et $t' = w[x := v]$;*
   - *soit $u = \varepsilon_\alpha\langle w\rangle$ et $t' = \varepsilon_\alpha\langle w\rangle$.*
4. *Si $t = \tau_\alpha\langle u, v\rangle$ alors :*
   - *soit $t' = \tau_\alpha\langle u', v'\rangle$ avec $u \|_0 u'$ et $v \|_0 v'$;*
   - *soit $u = \lambda x\ w$ et $t' = \lambda x\ w$;*
   - *soit $u = \varepsilon_\beta\langle w\rangle$ avec $\alpha \neq \beta$ et alors $t' = \varepsilon_\beta\langle w\rangle$;*
   - *soit $u = \varepsilon_\alpha\langle w\rangle$ et $t' = (v\ w)$.*
5. *Si $t = u[x := v]$ et si v n'est pas un terme redexeur alors $t' = u'[x := v']$ avec $u \|_0 u'$ et $v \|_0 v'$.*

Preuve Les deux premiers cas sont évidents, et le quatrième se traite comme le troisième. Prouvons donc le lemme dans les cas 3 et 5.

Soit E l'ensemble (primaire) de redex qui est réduit dans la réduction ($\|_0$) de t en t'.

Dans le troisième cas, $t = (u\ v)$ et on distingue deux cas : si t lui même appartient à E alors E se réduit au seul redex t, et ce redex est donc réduit ; sinon le lemme 5.3.3 permet de conclure.

Dans le dernier cas, puisque v n'est pas redexeur, la substitution ne crée aucun redex mais il peut y avoir des redex modifiés et des redex de v peuvent être dupliqués. Puisque E est un ensemble de redex α-équivalents, nous parlerons du redex r représenté par E. On doit distinguer deux cas.

- Soit le terme v contient (ou est égal à) une (au moins) instance du redex r. Il peut y avoir aussi dans u des instances de r, mais elles ne peuvent contenir la variable x et donc aucune instance de r, dans u ou dans v, n'est modifiée par le remplacement de x par v.

  Donc, si on nomme u' le terme obtenu à partir de u en réduisant (au sens de $\boldsymbol{\delta_0}$) toutes les instances de r dans u, et v' le terme obtenu à partir de v en réduisant (au sens de $\boldsymbol{\delta_0}$) toutes les instances de r dans v, alors on a bien $u \parallel_0 u'$, $v \parallel_0 v'$, et $t' = u'[x := v']$.

- Soit le redex r est issu du remplacement, dans un redex R de u, de x par v, et alors il contient strictement v.

  Nous examinons tout d'abord le cas où $t = r$. Posons $R \stackrel{def}{=} (\lambda y\ a\ b)$. On a alors $t = (\lambda y\ a[x := v]\ b[x := v])$ et donc $t' = a[x := v][y := b[x := v]] = a[y := b][x := v]$ car y n'est pas libre dans v. Si on pose $u' = a[y := b]$, on a alors $t' = u'[x := v]$ avec $u \parallel_0 u'$.

  Dans le cas général - plusieurs instances de r dans t - le résultat précédent reste valable parce que les instances de r sont disjointes et leur réduction est donc indépendante.

  □

Remarquer que si $u[x := v] \rightarrow_{\delta_0} t'$ et si v n'est pas un terme redexeur alors on n'a pas nécéssairement $t' = u'[x := v']$ avec $u \rightarrow_{\delta_0} u'$ et $v \rightarrow_{\delta_0} v'$. En effet dans t' une seule copie d'un redex de v peut avoir été réduite. Et c'est cette difficulté avec $\boldsymbol{\delta_0}$ qui rend nécessaire la réduction parallèle.

5.3.2 Conservation du type en réduction parallèle

Nous montrons que la réduction parallèle conserve le type.

Théorème 5.3.6 *Soient t et t' deux λ-termes tels que* $t \parallel t'$, *si* $\Gamma \vdash t : A$ *alors* $\Gamma \vdash t' : A$.

Preuve Nous montrons le résultat lorsque $t \parallel_0 t'$, d'où l'on déduit directement le lemme. La preuve se fait par récurrence sur le typage qui peut être supposé sans utilisation de T$\vee E$ «avec u variable principale» et clair.

On examine la dernière règle et on utilise le lemme 5.3.5 pour distinguer les différentes formes possibles pour t'.

- Si la dernière règle est Tvar, c'est que t=t' et le résultat est évident.

- Si c'est une des règles qui ne changent pas le terme : T$\forall E_1$, T$\forall E_2$, T$\forall I_1$, T$\forall I_2$, Teq, Tprop, T$\vee I1$, T$\vee I2$, le résultat est direct par hypothèse de récurrence.

- Si c'est Tabs, on a donc obtenu $\Gamma \vdash \lambda x\, u : A \to B$, à partir de $\Gamma, x : A \vdash u : B$. Mais $t = \lambda x\, u \parallel_0 t'$ implique $t' = \lambda x\, u'$ et $u \parallel_0 u'$. Par hypothèse de récurrence on a $\Gamma, x : A \vdash u' : B$. On en deduit (Tabs) $\Gamma \vdash \lambda x\, u' : A \to B$.

- Dans le cas de la règle Tapp, on a déduit (3) $\Gamma \vdash (u\ v) : B$ de (1) $\Gamma \vdash u : A \to B$ et (2) $\Gamma \vdash v : A$, et on peut alors distinguer plusieurs cas.

  - Si $t' = (u'\ v')$ avec $u \parallel_0 u'$ et $v \parallel_0 v'$, on a, par hypothèse de récurrence, $\Gamma \vdash u' : A \to B$ et $\Gamma \vdash v' : A$. Il suffit d'appliquer la règle Tapp pour conclure.
  - Si $u = \lambda x\, w$ et $t' = w[x := v]$, alors, de $\Gamma \vdash \lambda x\, w : A \to B$, on déduit (lemme 5.2.3) (4) $\Gamma, x : a \vdash w : B$. L'application du lemme de substitution dans le terme (5.1.8) à (2) et (4) donne le résultat cherché.
  - Si $u = \varepsilon_\alpha\langle w\rangle$ et $t' = \varepsilon_\alpha\langle w\rangle$, alors de $\Gamma \vdash \varepsilon_\alpha\langle w\rangle : A \to B$ on déduit (lemme 5.2.3) $\Gamma \vdash \varepsilon_\alpha\langle w\rangle : B$.

- Si la dernière règle est Texc, de $\Gamma \vdash u : D[a]$ on a déduit $\Gamma \vdash \varepsilon_\alpha\langle u\rangle : E^\alpha[r_\alpha(a)]$. Mais $t' = \varepsilon_\alpha\langle u'\rangle$, avec $u \parallel_0 u'$ et l'hypothèse de récurrence nous donne $\Gamma \vdash u' : D[a]$. L'application de la règle Texc permet de conlure.

- Si la dernière règle est Ttry, de $\Gamma \vdash u : D^{L+\alpha}[a]$ et $\Gamma \vdash v : \forall x(F[x] \to D^L[b])$ avec $\alpha = F^i$ on a déduit $\Gamma \vdash \tau_\alpha\langle u, v\rangle : D^L[t_\alpha(a, x \to b)]$. et nous devons, ici aussi, distinguer plusieurs cas.

  - Si $t' = \tau_\alpha\langle u', v'\rangle$ avec $u \parallel_0 u'$ et $v \parallel_0 v'$, on a, par hypothèse de récurrence, $\Gamma \vdash u' : D^{L+\alpha}[a]$ et $\Gamma \vdash v' : \forall x(F[x] \to D^L[b])$. Il suffit d'appliquer la règle Ttry pour conclure.
  - Si $u = \lambda x\, w$ et $t' = \lambda x\, w$, alors, de $\Gamma \vdash \lambda x\, w : D^{L+\alpha}[a]$, on déduit (lemme 5.1.7) $\Gamma \vdash \lambda x\, w : D[a]$ puis (T$\vee I2$) (4) $\Gamma \vdash \lambda x\, w : D^L[a]$. Mais l'axiome égalitaire trapD : $\Vdash D[a] \to t_\alpha(a, x \to b) = a$ nous donne (5) $\widehat{\Gamma} \Vdash t_\alpha(a, x \to b) = a$. Et l'application de Teq à (4) et (5) donne $\Gamma \vdash \lambda x\, w : D^L[t_\alpha(a, x \to b)]$.

- Si $u = \varepsilon_\beta\langle w\rangle$ avec $\alpha \neq \beta$ et $t' = \varepsilon_\beta\langle w\rangle$, alors de $\Gamma \vdash \varepsilon_\beta\langle w\rangle : D^{L+\alpha}[a]$ on déduit (lemme 5.1.7) $\beta \in L$ et $\Gamma \vdash \varepsilon_\beta\langle w\rangle : E^\beta[a]$ puis (T$\vee I2$) (4) $\Gamma \vdash \varepsilon_\beta\langle w\rangle : D^L[a]$. Mais l'axiome égalitaire trapE : $\Vdash E^\beta[a] \to t_\alpha(a, x \to b) = a$ nous donne (5) $\widehat{\Gamma} \Vdash t_\alpha(a, x \to b) = a$. Et l'application de Teq à (4) et (5) donne $\Gamma \vdash \varepsilon_\beta\langle w\rangle : D^L[t_\alpha(a, x \to b)]$.
- Si $u = \varepsilon_\alpha\langle w\rangle$ et $t' = (v\ w)$, alors de $\Gamma \vdash \varepsilon_\alpha\langle w\rangle : D^{L+\alpha}[a]$ on déduit (lemme 5.1.7) $\Gamma \vdash \varepsilon_\alpha\langle w\rangle : E^\alpha[a]$. On sait alors (lemme 5.1.6), si $\alpha = F^i$ par exemple, qu'il existe c tel que $\widehat{\Gamma} \Vdash a = r_\alpha(c)$ et $\Gamma \vdash w : F[c]$. En appliquant la règle Tapp, compte tenu du type de v, on a $\Gamma \vdash (v\ w) : D^L[b[x:=c]]$. Mais l'axiome égalitaire trapD : $\Vdash F[c] \to t_\alpha(r_\alpha(c), x \to b) = b[x:=c]$ nous donne $\Gamma \vdash (v\ w) : D^L[t_\alpha(r_\alpha(c), x \to b)]$ qui est le résultat cherché.

- Si la dernière règle est T$\vee E$, de $\Gamma, x : D^L[a] \vdash u : T$; $\Gamma, x : E^\alpha[a] \vdash u : T$ et $\Gamma \vdash v : D^{L+\alpha}[a]$ on a déduit $\Gamma \vdash u[x:=v] : T$. Puisque la preuve peut être supposée claire (proposition 5.1.10), v n'est pas redexeur. De $t' = u'[x:=v']$, avec $u \parallel_0 u'$ et $v \parallel_0 v'$, on tire, par hypothèse de récurrence, $\Gamma, x : D^L[a] \vdash u' : T$; $\Gamma, x : E^\alpha[a] \vdash u' : T$ et $\Gamma \vdash v' : D^{L+\alpha}[a]$; il suffit alors d'appliquer T$\vee E$ pour conclure.

□

5.3.3 Lien entre δ-réduction et réduction parallèle

Nous énonçons quelques propriétés, pour la plupart triviales, permettant de relier réduction parallèle ($\parallel$) et δ-réduction.

Lemme 5.3.7 *Pour tout λ-terme t :*

- *(1) si $t \parallel_0 t'$ alors $t \to_\delta t'$, (2) si $t \parallel t'$ alors $t \to_\delta t'$;*
- *(3) si $t \to_{\delta_0} t_1$ alors il existe t_2 tel que $t \parallel_0 t_2$ et $t_1 \parallel_0 t_2$;*

Preuve La preuve de (1) est immédiate : par définition de $\parallel_0$, les redex réduits sont disjoints et peuvent donc être réduits l'un après l'autre. Et (2) s'en déduit immédiatement.

La preuve de (3) est aussi simple : soit r le redex réduit dans $t \to_{\delta_0} t_1$, appelons E l'ensemble primaire (sur t) contenant r, il suffit de choisir pour t_2 le terme obtenu (à partir de t) en réduisant E. □

On pourrait montrer, en utilisant la même méthode qu'au chapitre 2, que la réduction parallèle est confluente. On en déduirait alors, immédiatement, le résultat « si u est normal et si $t \to_\delta u$ alors $t \parallel u$ ». Nous ne ferons pas la preuve de la confluence de la réduction parallèle, car nous n'avons pas besoin de ce résultat.

Le lemme suivant suffira pour montrer une conservation faible du type.

Lemme 5.3.8 *Pour tout λ-terme t fortement normalisable : si $t \to_\delta u$ et si u est normal alors $t \parallel u$.*

Preuve Le lemme se prouve par récurrence sur $l(t)$ longueur de la plus longue chaîne de δ-réduction de t. L'hypothèse est : u est normal et $t \to_{\delta_0} t_1 \to_\delta u$. D'après (3), il existe t_2 tel que $t \parallel_0 t_2$ et $t_1 \parallel_0 t_2$. On a $t_2 \to_\delta u$, car la δ-réduction est confluente et $l(t_2) < l(t)$, l'hypothèse de récurrence s'applique et nous donne $t_2 \parallel u$ donc $t \parallel u$. □

5.3.4 Conservation du type

La δ-réduction ne permet pas toujours la conservation du type (remarque 5.3.1), mais nous montrons que si nous la poursuivons jusqu'à son terme alors le type est conservé.

Théorème 5.3.9 (Conservation faible du type) *Soit t un λ-terme, si $\Gamma \vdash t : A$ alors $\Gamma \vdash \bar{t} : A$ (la notation $\bar{t}$ désigne la forme normale de t).*

Preuve

Si $\Gamma \vdash t : A$, alors, d'après le théorème de forte normalisation de $Ex2$ (4.2.4), t est fortement normalisable. Le lemme 5.3.8 nous permet d' affirmer que $t \parallel \bar{t}$ et donc (théorème 5.3.6) que $\Gamma \vdash \bar{t} : A$. □

Nous verrons au chapitre suivant que notre théorème de conservation faible du type est suffisant pour que nous puissions utiliser le système $Ex2$ comme un système de preuve de programmes.

Chapitre 6

Théorèmes pour programmer

Le but de ce chapitre est de préciser à quelles conditions pourra être obtenu dans le système $Ex2$ un λ-terme qui constitue un programme pour une fonction donnée par un ensemble d'équations.

Nous commençons par un résultat qui donne une caractérisation des termes de type entier.

6.1 Caractérisation des termes de type entier

Un certain nombre de lemmes doivent tout d'abord être établis.

6.1.1 Lemmes

Nous explorons quel type peut obtenir une variable, selon le type qu'elle avait dans le contexte. Et nous commençons par énoncer des impossibilités de typage.

Lemme 6.1.1 *Dans le contexte* $\Gamma, x : X(a)$ *on ne peut pas typer le terme* t *s'il a pour sous-terme* $(x\ w)$*, ou s'il a pour sous-terme* $\tau_\alpha\langle x, w\rangle$*.*

Dans le contexte $\Gamma, f : \forall y((Xy \to Xs(y))$ *on ne peut pas typer le terme* t *s'il a pour sous-terme* $\tau_\alpha\langle f, w\rangle$*.*

Preuve Le schéma de preuve «naturel» serait :

si dans le contexte $\Gamma, x : X(a)$ on a typé le terme t qui a pour sous-terme $(x\ w)$, alors, dans ce même contexte, $(x\ w)$ est typable et donc, dans ce contexte, x est typable de type $A \to B$. Et nous pouvons conclure puisque cette dernière proposition est interdite par le lemme 5.1.2.

Mais le résultat *a priori* évident : «si $t = (a\ b)$ est typable de type T dans le contexte Γ, alors il existe un type A tel que $\Gamma \vdash a : A \to T$ et $\Gamma \vdash b : A$» n'est pas immédiat comme nous le voyons plus loin (remarque 6.1.2).

Aussi, à défaut de pouvoir utiliser ce résultat, nous montrons le lemme par l'absurde, par récurrence sur la longueur du typage.

Montrons la première partie, dans le cas d'un sous-terme $(x\ w)$.

La dernière règle ne peut être Tvar. Et si la dernière règle ne modifie pas le terme, ou dans le cas des règles Tabs, Texc, Ttry, l'hypothèse de récurrence permet de conclure. Il nous reste donc à examiner deux règles.

- Si la dernière règle est Tapp, le terme t ne peut être égal à $(x\ w)$ car alors on aurait $\Gamma, x : Xa \vdash x : A \to B$ ce que le lemme 5.1.2 interdit. Mais si $t = (u\ v)$ avec $u \neq x$, il suffit d'appliquer l'hypothèse de récurrence à u et v et de conclure puisque $(x\ w)$ est sous-terme de u ou de v.

- Si la dernière règle est T$\vee E$, on a

  $(a)\Gamma, x : A, y : D^L[c] \vdash u : T;\ (b)\Gamma, x : A, y : E^\alpha[c] \vdash u : T;\ (c)\Gamma, x : A \vdash v : D^{L+\alpha}[c]$

  Nous savons que $t \stackrel{def}{=} u[y := v]$ a un sous-terme s égal à $(x\ w)$. Selon la position du sous-terme s par rapport aux copies de v qui remplacent y dans u, on peut distinguer plusieurs cas.

  - Soit s est inclus dans v (et $y \in u$), ce qui n'est pas possible puisque (c) montre que v est typable dans un contexte qui contient $x : A$ et l'hypothèse de récurrence l'interdit.
  - Soit s est extérieur aux remplacés v de y, c'est à dire s est un sous-terme de u, mais alors l'hypothèse de récurrence permet de conclure.
  - Soit s contient un des remplacés v de y, et deux cas sont à étudier :
    - v est dans la partie gauche de s, c'est à dire $v = x$ et u a un sous-terme $(y\ w')$, mais, avec l'hypothèse $x : Xa$, le lemme 5.1.2 interdit d'obtenir pour $v = x$ le type $D^{L+\alpha}[c]$;
    - v est dans la partie droite de s, c'est à dire u a un sous-terme $(x\ w')$, mais l'hypothèse de récurrence permet d'affirmer que c'est impossible.

La preuve de la première partie, dans le cas d'un sous-terme $\tau_\alpha\langle x, w\rangle$, est presque identique : seul le cas de la règle Ttry est à reprendre.

Le terme t ne peut être égal à $\tau_\alpha\langle x, w\rangle$ car alors on aurait $\Gamma, x : Xa \vdash x : D^{L+\alpha}[a]$ ce que le lemme 5.1.2 interdit. Donc $t = \tau_\beta\langle u, v\rangle$ avec $u \neq x$, et alors il suffit d'appliquer l'hypothèse de récurrence à u et v et de conclure puisque $\tau_\alpha\langle x, w\rangle$ est sous-terme de u ou de v.

La deuxième partie du lemme se prouve comme la première. □

Remarque 6.1.2 *Nous n'avons pas démontré le résultat : si* $\Gamma \vdash (a\ b) : T$, *alors on peut trouver un type* A *tel que* $\Gamma \vdash a : A \to T$ *et* $\Gamma \vdash b : A$.

A cause de la règle T$\vee E$, la preuve n'en est pas élémentaire.

En effet, si on tente de le montrer par récurrence sur la longueur du typage, lorsque la dernière règle est T$\vee E$, on a

$$(a)\Gamma, y : D^L[c] \vdash (a'\ b') : T;\ (b)\Gamma, y : E^\alpha[c] \vdash (a'\ b') : T;\ (c)\Gamma \vdash v : D^{L+\alpha}[c]$$

Et l'application de l'hypothèse de récurrence donne, d'une part, $\Gamma, y : D^L[c] \vdash a' : A_1 \to T$ et $\Gamma, y : D^L[c] \vdash b' : A_1$, d'autre part, $\Gamma, y : E^\alpha[c] \vdash a' : A_2 \to T$ et $\Gamma, y : E^\alpha[c] \vdash a' : A_2$. Mais, si nous ne savons rien sur le rapport existant entre A_1 et A_2, nous ne pouvons pas prouver le lemme, en tout cas de cette façon.

Le lemme suivant, construit de façon *ad hoc* en vue de notre théorème sur les entiers, utilise une notation déja vue (2.4.1) pour faire rentrer dans un même cadre l'application et le «try».

Lemme 6.1.3 *Soit Γ un contexte et $\Gamma' \stackrel{def}{=} \Gamma, x : X\mathbf{0}, f : \forall y(Xy \to Xs(y))$. Si $\Gamma' \vdash t \stackrel{def}{=} \lfloor A_n, \cdots \lfloor A_1, f, t_1 \rfloor, \cdots t_n \rfloor : T$ avec $n \neq \mathbf{0}$, alors n=1 et $T = \forall x_1 \cdots x_p X a$ (avec $x_i \neq X$ pour tout i tel que $1 \leq i \leq p$) et on a $\widehat{\Gamma'} \Vdash a = s(b)$ et $\Gamma' \vdash t_1 : Xb$.*

Preuve Prouvons le lemme par récurrence sur le typage, en examinant la dernière règle.

- La dernière règle ne peut être ni Tvar, ni Tabs, ni Texc à cause de la forme du terme t.
- La dernière règle ne peut être ni Tprop, ni T$\vee I1$, ni T$\vee I2$: appliquer l'hypothèse de récurrence à la prémisse, et voir que le type ne convient pas.
- Dans le cas des règles T$\forall I_1$, T$\forall I_2$, T$\forall E_1$ ou T$\forall E_1$, l'hypothèse de récurrence permet de conclure.
- Si le dernière règle est Teq, de $\Gamma' \vdash t : T[y{:=}u]$ et $\widehat{\Gamma'} \Vdash u = v$, on a déduit $\Gamma' \vdash t : T[y{:=}v]$. Mais, par hypothèse de récurrence, on a $T[y{:=}u] = \forall x_1 \cdots x_n X a$ avec $\widehat{\Gamma'} \Vdash a = s(b)$ et $\Gamma' \vdash t_1 : Xb$. On a donc $T = \forall x_1 \cdots x_n X a'$ avec $a \stackrel{def}{=} a'[y{:=}u]$ et le type final est $T[y{:=}v] = \forall x_1 \cdots x_n X a'[y{:=}v]$.

  Mais de $\widehat{\Gamma'} \Vdash u = v$, on déduit (lemme 3.4.4) $\widehat{\Gamma'} \Vdash a'[y{:=}v] = a'[y{:=}u]$, puisque y peut être choisi non dans $\widehat{\Gamma'}$. On a donc bien (transitivité de l'égalité) $\widehat{\Gamma'} \Vdash a'[y{:=}v] = \boldsymbol{s}(b)$.
- Si la dernière règle est Tapp, on a $t = (u\ t_n)$ et $\Gamma' \vdash u : B \to T$ et $\Gamma' \vdash t_n : B$. Mais n ne peut être différent de 1, car $u = \lfloor A_{n-1}, \cdots \lfloor A_1, f, t_1 \rfloor, \cdots t_{n-1} \rfloor$ ne peut, par hypothèse de récurrence, avoir un type flèche. L'entier n

est donc 1, et on a $\Gamma' \vdash f : B \to T$ et $\Gamma' \vdash t_1 : B$ ce qui entraine, d'après la deuxième partie du lemme 5.1.2, $B = X\,c$ et $T = X\,d$ avec $\widehat{\Gamma'} \Vdash d = s(c)$.

- La dernière règle ne peut être Ttry car alors on devrait avoir ou bien

$$\Gamma' \vdash \lfloor A_{n-1}, \cdots \lfloor A_1, f, t_1 \rfloor, \cdots t_{n-1} \rfloor : D^{L+\alpha}[a]$$

ce qui, par hypothèse de récurrence, n'est pas possible ou bien $\Gamma' \vdash f : D^{L+\alpha}[a]$ ce que le lemme 5.1.2 interdit.

- Dans le cas de la règle T$\vee E$, on a $(a)\Gamma, x : X\,\mathbf{0}, f : \forall y(X\,y \to X\,s(y)), y : D^L[c] \vdash u : T$, $(b)\Gamma, x : X\,\mathbf{0}, f : \forall y(X\,y \to X\,s(y)), y : E^\alpha[c] \vdash u : T$, et $(c)\Gamma, x : X\,\mathbf{0}, f : \forall y(X\,y \to X\,s(y)) \vdash v : D^{L+\alpha}[c]$.

  Si $u[y{:=}v] = \lfloor A_n, \cdots \lfloor A_1, f, t_1 \rfloor, \cdots t_n \rfloor$, que peut-on dire de u?

  Deux cas sont possibles :

  - soit il existe m ($m \leq n$) tel que $u = \lfloor A_n, \cdots \lfloor A_{m+1}, y, u_{m+1} \rfloor, \cdots u_n \rfloor$ et $v = \lfloor A_m, \cdots \lfloor A_1, f, t_1 \rfloor, \cdots t_m \rfloor$ avec pour tout i ($1 \leq i \leq m$) l'égalité $u_i[y{:=}v] = t_i$;
  - soit $u = \lfloor A_n, \cdots \lfloor A_1, f, u_1 \rfloor, \cdots u_n \rfloor$, avec pour tout i $1(\leq i \leq n)$ $u_i[y{:=}v] = t_i$.

  Si on est dans le premier cas, on ne peut avoir $m \neq \mathbf{0}$ car, par hypothèse de récurrence, le type de v, si $v = \lfloor C, f, t \rfloor$, ne peut être $D^{L+\alpha}[c]$. Mais alors $m = \mathbf{0}$ et donc $v = f$, ce qui est interdit (lemme 5.1.2), car alors on aurait $\Gamma, x : X\,\mathbf{0}, f : \forall y(X\,y \to X\,s(y)) \vdash f : D^{L+\alpha}[c]$).

  Reste à examiner le second cas : $u = \lfloor A_n, \cdots \lfloor A_1, f, u_1 \rfloor, \cdots u_n \rfloor$. Par hypothèse de récurrence $n = 1$ et $T = \forall x_1 \cdots x_p X\,a$ avec $x_i \neq X$ pour tout i tel que $1 \leq i \leq p$.

  D'une part $\widehat{\Gamma'}, D^L[c] \Vdash a = s(b)$ et $\widehat{\Gamma'}, E^\alpha[c] \Vdash a = s(b)$ ce qui en appliquant L$\vee E$ avec $\widehat{\Gamma'} \Vdash D^{L+\alpha}[c]$ nous donne $\widehat{\Gamma'} \Vdash a = s(b)$.

  D'autre part $\Gamma', y : D^L[c] \vdash u_1 : X\,b$ et $\Gamma', y : E^\alpha[c] \vdash u_1 : X\,b$ ce qui en appliquant T$\vee E$ avec $\Gamma' \vdash v : D^{L+\alpha}[c]$ nous donne $\Gamma' \vdash t_1 \stackrel{def}{=} u_1[y{:=}v] : X\,b$.

□

6.1.2 Formules égalitairement sans conséquence

Dans la preuve d'une égalité, certaines hypothèses s'avèrent inutiles, et leur forme suffit pour le savoir. Mais définissons tout d'abord ce que nous entendons par inutiles.

Définition 6.1.4 *Un ensemble de formules $\mathcal{B}$ est dit égalitairement sans conséquence, relativement à un second ensemble de formules $\mathcal{A}$ si, pour toute égalité $(a = b)$, lorsque $\mathcal{A}, \mathcal{B} \Vdash a = b$ alors $\mathcal{A} \Vdash a = b$.*

Nous pouvons maintenant énoncer que certaines formules vont être «inutiles».

Lemme 6.1.5 *Tout ensemble de formules dans lequel chaque formule est de la forme $\forall x_1 \cdots x_n X a$ ou $\forall y_1 \cdots y_p (A \to \forall x_1 \cdots x_n X a)$ (si $x_i \neq X$ pour tout i tel que $1 \leq i \leq n$ et $y_i \neq X$ pour tout i tel que $1 \leq i \leq p$) est égalitairement sans conséquence relativement à tout ensemble de formules dans lequel la variable X n'est pas libre.*

Preuve Considérons un ensemble $\mathcal{A}$ de formules dans lequel la variable X n'est pas libre, et, par exemple, les deux formules $B = \forall x_1 \cdots x_n X a$ et $C = \forall y_1 \cdots y_p (A \to \forall z_1 \cdots z_m X b)$. De $\mathcal{A}, B, C \Vdash t_1 = t_2$ on déduit (Labs) $\mathcal{A} \Vdash B, C \to t_1 = t_2$ puis, en utilisant la règle L$\forall I_2$, ce qui est possible puisque X n'est pas libre dans $\mathcal{A}$, on obtient $\mathcal{A} \Vdash \forall X (B, C \to (t_1 = t_2))$, puis (L$\forall E_2$) en remplaçant Xt par la formule $\forall Y (Y \to Y)$ notée V, on obtient $F \Vdash B[Xt{:=}V], C[Xt{:=}V] \to (t_1 = t_2))$, remarquer que $(t_1 = t_2)[Xt{:=}V] = (t_1 = t_2)$.

Mais $B[Xt{:=}V] = \forall x_1 \cdots x_n V$ et $C[Xt{:=}V] = \forall y_1 \cdots y_p (A[x{:=}v] \to \forall x_1 \cdots x_n V)$ et on a de façon évidente $\Vdash B[Xt{:=}V]$ et $\Vdash C[Xt{:=}V]$. Donc, par application de Lapp, on obtient $\mathcal{A} \Vdash (t_1 = t_2)$. □

6.1.3 Le théorème de caractérisation des entiers

Nous pouvons démontrer que les seuls termes de type entier, dans le contexte vide, sont les entiers de Church.

Théorème 6.1.6 (Caractérisation des entiers) *Si $\vdash t : N[a]$ alors il existe un entier n (éventuellement nul) tel que $t \to_\delta \underline{n}$ et $\Vdash a = \boldsymbol{s}^n(\mathbf{0})$ (ou $t \to_\delta \lambda f\, f$ et $\Vdash a = \boldsymbol{s}(\mathbf{0})$).*

Rappelons que $\boldsymbol{s}^n(\mathbf{0})$ désigne le n-ième succeseur de zéro et que $\underline{n}$ est l'entier de Church $\lambda f\, \lambda x\, (f^n\ x)$.

On peut aussi caractériser les termes de type booléen: Si $\vdash t : Bool[a]$ alors soit $\Vdash a = \boldsymbol{v}$ et $t \to_\delta \lambda x\, \lambda y\, x$ soit $\Vdash a = \boldsymbol{f}$ et $t \to_\delta \lambda x\, \lambda y\, y$.

La définition que nous avons choisie pour $N[a]$ (définition 3.1.3) est $\forall X (\forall y (Xy \to X\boldsymbol{s}(y)), X\mathbf{0} \to Xa)$. Ce choix est celui de AF2. Il donne cependant, dans $Ex2$ aussi bien que dans AF2, une preuve moins satisfaisante que le choix $N[a] = \forall X (X\mathbf{0}, \forall y (Xy \to X\boldsymbol{s}(y)) \to Xa)$ qui lui évite le cas $t = \lambda f\, f$. Mais le cas $t = \lambda f\, f$ n'est pas très génant puisque nous savons que $\lambda f\, f$ est η-équivalent à $\underline{1}$.

Preuve Supposons que $\vdash t : N[a]$, alors t est fortement normalisable puisque typable, il a donc une forme normale notée $\bar{t}$, et on a $\vdash \bar{t} : N[a]$ par conservation du type. D'autre part, $\bar{t}$ est clos puisque typable dans le contexte vide. Dans notre preuve, nous prenons donc comme hypothèses : t est un terme normal et clos tel que $\vdash t : N[a]$.

Nous savons, puisqu'il est clos, que t est de l'une des deux formes $\lambda f\, u$ ou $\varepsilon_\alpha\langle u\rangle$ (lemme 2.4.2).

Mais si on avait $t = \varepsilon_\alpha\langle u\rangle$, alors de $\vdash t : \forall X(X\mathbf{0}, \forall y(Xy \to X\boldsymbol{s}(y)) \to Xa)$ on déduirait (T$\forall E_2$) $\vdash t : X\mathbf{0}, \forall y(Xy \to X\boldsymbol{s}(y)) \to Xa$ puis (lemme 5.2.3) $\vdash t : \forall y(Xy \to X\boldsymbol{s}(y)) \to Xa$ et $\vdash t : Xa$ ce qui est impossible (lemme 5.1.5).

On a donc $t = \lambda f\, u$ et on en déduit (T$\forall E_2$ et lemme 5.2.3) $f : \forall y(Xy \to X\boldsymbol{s}(y)) \vdash u : X\mathbf{0} \to Xa$.

On peut affirmer que :

- Si u est une variable c'est la variable f et on a alors (lemme 5.1.2) $\forall y(Xy \to X\boldsymbol{s}(y)) \Vdash a = \boldsymbol{s}(o)$ et donc (lemme 6.1.5) $\Vdash a = \boldsymbol{s}(o)$.

- Le terme u n'est pas de la forme $\varepsilon_\alpha\langle w\rangle$ car sinon on obtiendrait (lemme 5.2.3) $f : \forall y(Xy \to X\boldsymbol{s}(y)) \vdash \varepsilon_\alpha\langle w\rangle : Xa$ ce qui est impossible (lemme 5.1.5).

Si u n'était pas de la forme $\lambda x\, w$, on aurait, puisque u est normal et contient comme seule variable libre f (lemme 2.4.3), avec les notations de 2.4.1 : $u = \lfloor X_n, \lfloor X_{n-1}, \cdots \lfloor X_1, f, t_1\rfloor, t_{n-1}\rfloor, t_n\rfloor$ (où chacun des X_i est soit A soit T et où les t_i sont des termes). Mais le lemme 6.1.3 nous dit que ce n'est pas possible si on veut que u ait pour type $X\mathbf{0} \to Xa$, ce qui rend l'alternative u n'est pas de la forme $\lambda x\, w$ impossible.

Donc $u = \lambda x\, v$ et on obtient (lemme 5.2.3) : $f : \forall y(Xy \to X\boldsymbol{s}(y)), x : X\mathbf{0} \vdash v : Xa$.

Nous allons maintenant montrer, par récurrence sur la complexité de v, que, si $x : X\mathbf{0}, f : \forall y(Xy \to X\boldsymbol{s}(y)) \vdash v : Xa$, alors il existe un entier n (éventuellement nul) tel que $v = (f^n\ x)$ et $\Vdash a = \boldsymbol{s}^n(\mathbf{0})$, résultat qui nous permettra de conclure notre preuve. Examinons quel peut être v ?

Si v est un terme élémentaire c'est à dire une variable :

- v ne peut être f (lemme 5.1.2) ;

- v peut être x, et alors on en déduit (toujours le lemme 5.1.2) $X\mathbf{0}, \forall y(Xy \to X\boldsymbol{s}(y)) \Vdash a = \mathbf{0}$ qui nous amène (lemme 6.1.5) à $\Vdash a = \mathbf{0}$, et notre résultat est bien vérifié.

Sinon v ne peut être ni $\lambda y\, w$, ni $\varepsilon_\alpha\langle w\rangle$ (lemme 5.1.5), donc v est de la forme $v = \lfloor X_n, \cdots \lfloor X_1, w, t_1\rfloor, \cdots t_n\rfloor$ avec w égal à f ou x. Mais w ne

peut être égal à x car, alors, dans un contexte où $x : X\mathbf{0}$, avec comme sous-terme $\lfloor X_1, x, t_1 \rfloor$, v ne pourrait être typable (lemme 6.1.1). Et v ne peut être égal à f si $X_1 = T$, en vertu du même lemme. Donc finalement $v = \lfloor X_n, \cdots \lfloor A, f, t_1 \rfloor, \cdots t_n \rfloor$.

Nous utilisons alors le lemme 6.1.3 pour en déduire que $v = (f\ t_1)$ avec $X\mathbf{0}, \forall y(Xy \to X\boldsymbol{s}(y)) \Vdash a = \boldsymbol{s}(b)$, et $x : X\mathbf{0}, f : \forall y(Xy \to X\boldsymbol{s}(y)) \vdash t_1 : Xb$.

Mais l'hypothèse de récurrence nous permet d'affirmer qu' il existe un entier p (éventuellement nul) tel que $t_1 = (f^p\ x)$ et $\Vdash b = \boldsymbol{s}^p(\mathbf{0})$, et nous obtenons bien finalement $v = (f^{p+1}\ x)$ et, à partir de $\Vdash b = \boldsymbol{s}^p(\mathbf{0})$ et $X\mathbf{0}, \forall y(Xy \to X\boldsymbol{s}(y)) \Vdash a = \boldsymbol{s}(b)$ (donc - lemme 6.1.5 - $\Vdash a = \boldsymbol{s}(b)$), nous obtenons $\Vdash b = \boldsymbol{s}^{p+1}(\mathbf{0})$. □

Le théorème de caractérisation des entiers nous permet donc d'affirmer que si un λ-terme est de type entier (ou booléen ...), son exécution ne peut aboutir à une exception non capturée. C'est un résultat indispensable si on veut pouvoir utiliser $Ex2$ pour programmer, mais qui n'était pas vérifié dans le langage Caml (paragraphe 1.4).

6.1.4 Caractérisation des exceptions

Les termes de type exception ($E^\alpha[a]$) sont bien ceux que nous attendions.

Théorème 6.1.7 *Si $\vdash t : E^\alpha[a]$ avec $\alpha = D$ alors il existe un λ-terme u et un terme d'individu b tels que $t \to_\delta \varepsilon_\alpha\langle u\rangle$, $\vdash u : D[b]$ et $\Vdash a = r_\alpha(b)$.*

Preuve Comme dans la preuve du théorème de caractérisation des entiers (6.1.6), on peut supposer le terme t normal et clos. Puisque t est clos, on sait (lemme 2.4.2) qu' il est de l'une des deux formes $\lambda x\ u$ ou $\varepsilon_\beta\langle u\rangle$. La première forme n'est pas possible (lemme 5.1.5), on a donc $t = \varepsilon_\beta\langle u\rangle$. Le lemme 5.1.7 nous dit que $\alpha = \beta$, et le lemme 5.1.6 permet de conclure. □

Nous pouvons maintenant utiliser les deux théorèmes précédents pour caractériser les termes dont le type est $N^L[a]$.

Théorème 6.1.8 *Si $\vdash t : N^{\alpha,\beta}[a]$ avec $\alpha = N$ et $\beta = B$ alors :*

- *soit il existe un entier n (éventuellement nul) tel que $t \to_\delta \underline{n}$ et $\Vdash a = \boldsymbol{s}^n(\mathbf{0})$ (ou $t \to_\delta \lambda f\ f$ et $\Vdash a = \boldsymbol{s}(\mathbf{0})$)*
- *soit il existe un entier n (éventuellement nul) tel que $t \to_\delta \varepsilon_\alpha\langle\underline{n}\rangle$ et $\Vdash a = r_\alpha(\boldsymbol{s}^n(\mathbf{0}))$ (ou $t \to_\delta \varepsilon_\alpha\langle\lambda x\ x\rangle$ et $\Vdash a = r_\alpha(\boldsymbol{s}(\mathbf{0}))$*
- *soit $t \to_\delta \varepsilon_\beta\langle\underline{V}\rangle$ et $\Vdash a = r_\alpha(\boldsymbol{v})$, ou $t \to_\delta \varepsilon_\beta\langle\underline{F}\rangle$ et $\Vdash a = r_\alpha(\boldsymbol{f})$.*

Le théorème est énoncé dans un cas particulier pour éviter des formulations pénibles, mais il peut évidemment se généraliser au cas où l'hypothèse est $\vdash t : D^L[a]$.

Preuve On peut supposer le terme t normal et clos, il est donc de l'une des deux formes $\lambda x\, u$ ou $\varepsilon_\gamma\langle u\rangle$.

Si $t = \lambda x\, u$, le lemme 5.1.7 nous donne $\vdash t = N[a]$ et le théorème de caractérisation des entiers (6.1.6) permet de conclure.

Si $t = \varepsilon_\gamma\langle u\rangle$, le lemme 5.1.7 nous dit que $\gamma = \alpha$ et $\vdash t : E^\alpha[a]$ ou $\gamma = \beta$ et $\vdash t : E^\beta[a]$, il suffit alors d'appliquer le théorème 6.1.7 pour conclure. □

6.2 Théorème de programmation

Jusqu'ici nous sommes restés dans un cadre strictement syntaxique. Mais le théorème de programmation (6.2.5), que nous démontrons plus loin, ne peut se formuler sans la notion de modèle.

6.2.1 Modèles

Rappelons ce qu'est pour nous un langage $\mathcal{L}$. Nous disposons d'un ensemble de symboles de données (par exemple N, B), avec lesquels sont construites des dénominations de type (par exemple N_1, N_2, B_1). A partir d'une dénomination de type α, on construit le symbole de relation (unaire): $E^\alpha[.]$, et à partir d'une liste de dénominations de type L, et d'un symbole de données D, on construit le symbole de relation (unaire) $D^L[.]$. Nous disposons aussi de symboles d'individus, comme indiqué dans la définition 3.1.1. La donnée des symboles de fonction et de relation constitue ce qu'on appelle un langage $\mathcal{L}$.

Nous nommerons prémodèle du langage $\mathcal{L}$ ce que classiquement on nomme modèle plein du second ordre de $\mathcal{L}$ ([10]). Et nous définirons un modèle comme un prémodèle particulier.

Définition 6.2.1 *Un modèle $\mathcal{M}$ de $\mathcal{L}$ est un prémodèle qui vérifie les cinq conditions suivantes :*

(C1) *si $\alpha = D_i$, alors $|E^\alpha| = \{r_\alpha(t); t \in |\mathcal{M}|$ et $\mathcal{M} \models D[t]\}$*

(C2) *$\mathcal{M} \models D^{L+\alpha}[t]$ ssi $\mathcal{M} \models D^L[t]$ ou $\mathcal{M} \models E^\alpha[t]$*
$\mathcal{M} \models D^\emptyset[t]$ ssi $\mathcal{M} \models D[t]$ où $D[x]$ est la formule de AF2 associée au symbole de données D.

(C3) *si $\mathcal{M} \models E^\alpha[x]$ alors $|cas_\alpha(x, y, z)| = |y|$*
si $\mathcal{M} \models D^L[x]$ alors $|cas_\alpha(x, y, z)| = |z|$ si $\alpha \notin L$

(C4) *si $\mathcal{M} \models D[x]$ alors $|t_\alpha(r_\alpha(x), f)| = |f(x)|$*
si $\mathcal{M} \models D^L[x]$ alors $|t_\alpha(x, f)| = |x|$ si $\alpha \notin L$

(C5) $\mathcal{M} \models \neg(\mathbf{0} = \boldsymbol{s}(\mathbf{0}))$

Une fois ces conditions imposées aux modèles, il est facile de montrer que notre système logique est sûr.

Lemme 6.2.2 *Si* $\Vdash^{\mathcal{E}} F$ *alors, pour tout modèle* $\mathcal{M}$ *de* $\mathcal{E}$*, on a* $\mathcal{M} \models F$.

Preuve La preuve se fait par récurrence sur la longueur du typage $\Vdash^{\mathcal{E}} F$, on distingue selon la dernière règle. Examinons les cas les moins évidents.

- Si la dernière règle est Leg, on doit distinguer deux cas :
  - dans le cas où F est une égalité de base, on a $\mathcal{M} \models F$ puisque $\mathcal{M}$ est un modèle de $\mathcal{E}$,
  - dans le cas où F est un axiome égalitaire on a aussi $\mathcal{M} \models F$, il suffit d'utiliser la condition C3 dans le cas de l'axiome égalitaire casE ou CasD, la condition C4 pour trapE ou trapD et C5 pour l'axiome égalitaire pluralité.
- Si la dernière règle est Lexc, la condition C1 assure le résultat.
- Si la dernière règle est L$\vee I1$, L$\vee I2$ ou encore L$\vee E$, la condition C2 permet de conclure.
- Si la dernière règle est Ltry, on a de $\Vdash^{\mathcal{E}} D^{L+\alpha}[a]$ et $\Vdash^{\mathcal{E}} \forall x(F[x] \rightarrow D^{L}[f(x)]$ obtenu $\Vdash^{\mathcal{E}} D^{L}[t_\alpha(a,f)]$. Par hypothèse de récurrence $\mathcal{M} \models D^{L+\alpha}[a]$, et C2 permet de distinguer deux cas.
  - Si $\mathcal{M} \models E^{\alpha}[a]$, alors (C1) $\mathcal{M} \models a = r_\alpha(t)$ et $\mathcal{M} \models F[t]$. Mais, par hypothèse de récurrence, $\mathcal{M} \models F[t] \rightarrow D^{L}[f(t)]$ et donc $\mathcal{M} \models D^{L}[f(t)]$. On utilise C3 pour obtenir $\mathcal{M} \models t_\alpha(r_\alpha(t), f) = f(t)$ et donc (Leq) $\mathcal{M} \models D^{L}[t_\alpha(a,f)]$.
  - Si $\mathcal{M} \models D^{L}[a]$, le raisonnement est semblable.

□

Rappelons qu'un modèle standard est un modèle dont l'ensemble de base est $\Lambda_E / \simeq_{\boldsymbol{\delta}}$.

Dans AF2, si on veut pouvoir programmer, on doit imposer des conditions sur l'interprétation, dans tout modèle standard, des symboles $\boldsymbol{s}, \mathbf{0}, \boldsymbol{v}, \boldsymbol{f} \cdots$ qui interviennent dans les formules définissant les types entiers, booléens .... Nous imposerons les mêmes conditions que nous nommerons conditions sur les types de données.

Définition 6.2.3 *Un modèle standard* $\mathcal{M}$ *vérifie :*

- *la condition sur le type de données N si pour tout entier n, l'interprétation de* $\boldsymbol{s}^n(\mathbf{0})$ *est l'entier de Church* $\underline{n}$

- *la condition sur le type de données B si l'interprétation de* $\boldsymbol{v}$ *est* $\underline{V} \stackrel{def}{=} \lambda x\, \lambda y\, x$, *et celle de de* $\boldsymbol{f}$ *est* $\underline{F} \stackrel{def}{=} \lambda x\, \lambda y\, y$

Les conditions sur les autres types de données sont semblables.

On peut construire un modèle standard vérifiant les conditions précédentes.

Lemme 6.2.4 *Il existe un modèle standard vérifiant les conditions sur les types de données N et B.*

Preuve Définissons notre modèle standard, i et j sont des entiers quelconques.

- Si $\alpha = D_i$ alors $|E^\alpha| = \{\varepsilon_\alpha\langle t\rangle; t \in |D|\}$.
- Pour tout ensemble de dénominations de type L, $|D^{L+\alpha}| = |D^L| \cup |E^\alpha|$.
- On a $|\boldsymbol{v}| = \lambda x\, \lambda y\, x$, $|\boldsymbol{f}| = \lambda x\, \lambda y\, y$ et, pour tout entier n, $|\boldsymbol{s}^n(\mathbf{0})| = \underline{n}$.
- Pour $\alpha = N_i$, on définit $|r_\alpha(t)| = \varepsilon_\alpha\langle t\rangle$ si $t = \boldsymbol{s}^n(\mathbf{0})$ et $|r_\alpha(t)| = t$ sinon ;

  pour $\alpha = B_j$, on définit $|r_\alpha(t)| = \varepsilon_\alpha\langle t\rangle$ si $t = \boldsymbol{v}$ ou $t = \boldsymbol{f}$ et $|r_\alpha(t)| = t$ sinon.

- Si, par exemple, $\alpha = N_i$ et $\beta = B_j$:

  $|cas_\alpha(t, x, y)| = |y|$ pour $t = \boldsymbol{s}^n(\mathbf{0}), \boldsymbol{v}, \boldsymbol{f}, r_\beta(\boldsymbol{v})$ ou $r_\beta(\boldsymbol{f})$,

  $|cas_\alpha(t, x, y)| = |x|$ pour $t = r_\alpha(\boldsymbol{s}^n(\mathbf{0}))$

  et $|cas_\alpha(t, x, y)| = t$ sinon.

- Si, par exemple, $\alpha = N_i$ et $\beta = B_j$:

  $|t_\alpha(t, f)| = |t|$ pour $t = \boldsymbol{s}^n(\mathbf{0}), \boldsymbol{v}, \boldsymbol{f}, r_\beta(\boldsymbol{v})$ ou $r_\beta(\boldsymbol{f})$,

  $|t_\alpha(r_\alpha(t), f)| = |f(t)|$ pour $t = \boldsymbol{s}^n(\mathbf{0})$

  et $|t_\alpha(t, f)| = t$ sinon.

Il est facile de s'assurer que ce modèle vérifie les conditions (C1) à (C5) et les conditions sur les types de données N et B. □

6.2.2 Théorème de programmation

Nous avons besoin d'un théorème garantissant qu'un terme du type correct est bien un programme pour la fonction que l'on souhaite programmer.

Théorème 6.2.5 (de programmation) *Soient $D_1[x_1], \cdots, D_n[x_n], F[y]$ des formules prises parmi les formules définissant un type de données syntaxique. Soit $\mathcal{M}$ un modèle standard du système d'équations $\mathcal{E}$, vérifiant les conditions sur les types de données $D_1, \cdots, D_n, F$.*

Si t est un terme tel que $\vdash^{\mathcal{E}} t : \forall x_1 \cdots x_n (D_1[x_1], \cdots, D_n[x_n] \rightarrow F[f(x_1 \cdots x_n)])$ alors t est un programme pour la fonction f c'est à dire :

$\mathcal{M} \models (t\ u_1 \cdots u_n) = f(u_1, \cdots, u_n)$ si les termes $u_1, \cdots, u_n$ sont dans les ensembles de termes du λ-calcul définis par les formules $D_1, \cdots, D_n$.

Preuve Pour fixer les idées, nous ferons la preuve pour un terme t tel que $\vdash^{\mathcal{E}} t : \forall x \forall y (N[x], N[y] \rightarrow N[f(x, y)])$.

Soient $u = \boldsymbol{s}^n(\mathbf{0})$ et $v = \boldsymbol{s}^m(\mathbf{0})$. On a $\vdash_{\mathcal{E}} (t\ u\ v) : N[f(\boldsymbol{s}^n(\mathbf{0}), \boldsymbol{s}^m(\mathbf{0}))]$.

Le théorème de caractérisation des entiers (6.1.6) permet d'affirmer qu'il existe un entier k tel que $(t\ u\ v) \rightarrow_\delta \underline{k}$ et $\Vdash f(\boldsymbol{s}^n(\mathbf{0}), \boldsymbol{s}^m(\mathbf{0})) = \boldsymbol{s}^k(\mathbf{0})$.

Mais, puisque le système logique est sûr (lemme 6.2.2), les termes $f(\boldsymbol{s}^n(\mathbf{0}), \boldsymbol{s}^m(\mathbf{0}))$ et $\boldsymbol{s}^k(\mathbf{0})$ sont égaux dans tout modèle. On a donc $\mathcal{M} \models f(\boldsymbol{s}^n(\mathbf{0}), \boldsymbol{s}^m(\mathbf{0})) = \boldsymbol{s}^k(\mathbf{0})$.

Dans le modèle standard $\mathcal{M}$, on a $\mathcal{M} \models (t\ u\ v) = \underline{k}$. Et comme $\mathcal{M}$ vérifie la condition sur le type de données N, on a $\mathcal{M} \models \boldsymbol{s}^k(\mathbf{0}) = \underline{k}$. On en déduit $\mathcal{M} \models (t\ u\ v) = \underline{k} = \boldsymbol{s}^k(\mathbf{0}) = f(\boldsymbol{s}^n(o), \boldsymbol{s}^m(o))$ ce qui clôt la preuve. □

6.2.3 Prolongement de modèle

Prouver qu'il existe un modèle standard vérifiant le système d'équations $\mathcal{E}$ peut être fastidieux, puisque cela oblige à interpréter chaque fonction sur $\Lambda_E / \simeq_{\boldsymbol{\delta}}$ tout entier. Heureusement, nous montrons qu'il n'est pas nécéssaire de construire un modèle complet, interpréter les fonctions sur leur ensemble de définition suffit.

Nous reprenons la structure du raisonnement de [10] (page 154), pour montrer comment étendre un modèle des fonctions à programmer ayant pour ensemble de base l'ensemble de définition de ces fonctions en un modèle standard.

Considérons des fonctions $\phi_1, \cdots, \phi_n$ de $(N^\alpha)^{k_i}$ dans N^α, où $N^\alpha \stackrel{def}{=} N \cup E^\alpha$, $N = \{\underline{n}; n \in N\}$ et $E^\alpha = \{\varepsilon_\alpha\langle\underline{n}\rangle; n \in N\}$.

Soit $\mathcal{L}$ le langage du second ordre ayant pour seuls symboles de fonctions : $\boldsymbol{s}, \mathbf{0}, r_\alpha, cas_\alpha, t_\alpha, f_1, \cdots, f_n$ (f_i d'arité k_i et $\alpha = N$).

Nous supposons qu'il existe un modèle $\mathcal{M}^N$ vérifiant les conditions sur le type de données N, d'ensemble de base N^α, où le symbole f_i est interprété

par la fonction ϕ_i $(1 \leq i \leq n)$. Soit $\mathcal{E}$ l'ensemble des égalités vraies dans $\mathcal{M}^N$.

Considérons le modèle $\mathcal{M}^T$ suivant : son ensemble de base est $\mathcal{T}/\simeq$ où $\mathcal{T}$ est l'ensemble des termes clos du langage $\mathcal{L}$ et où $u \simeq v$ ssi $u = v$ est un cas particulier d'une équation de $\mathcal{E}$, $E^\alpha = \{r_\alpha(\boldsymbol{s}^n(\mathbf{0})); n \in N\}$ et les symboles de fonctions sont interprétés par eux mêmes.

On vérifie facilement que $\mathcal{M}^T$ est un modèle au sens de la définition 6.2.1.

Nous définissons un isomorphisme i de $|\mathcal{M}^T|$ sur $|\mathcal{M}^N|$: si $\overline{u} \in |\mathcal{M}^T|$ alors $i(\overline{u}) = |u|_{\mathcal{M}^N}$. L'application i est en effet injective car $i(\overline{u}) = i(\overline{v})$ signifie $|u|_{\mathcal{M}^N} = |v|_{\mathcal{M}^N}$ ce qui prouve que $u = v$ est un cas particulier d'une équation de $\mathcal{E}$ et donc $\overline{u} = \overline{v}$. Elle est surjective car $\underline{n} = i(\boldsymbol{s}^n(\mathbf{0}))$ et $\varepsilon_\alpha\langle\underline{n}\rangle = i(r_\alpha(\boldsymbol{s}^n(\mathbf{0})))$.

Considérons le langage $\mathcal{L}'$ obtenu en rajoutant au langage $\mathcal{L}$ une suite infinie $c_0, \cdots, c_p, \cdots$ de symboles de constante. Selon le même mode de construction que $\mathcal{M}^T$, mais à partir cette fois du langage $\mathcal{L}'$, nous construisons le modèle $\mathcal{M}^{T'}$.

Nous vérifions que $\mathcal{M}^{T'}$ est un modèle de $\mathcal{E}$. En effet soit $u = v$ une équation de $\mathcal{E}$ dans laquelle les termes u, v contiennent les seules variables $x_1, \cdots, x_n$. On doit vérifier que $\mathcal{M}^{T'} \models u[x_1 := t_1, \cdots, x_n := t_n] =$
$v[x_1 := t_1, \cdots, x_n := t_n]$ pour tous termes $t_1, \cdots, t_n$ de $\mathcal{L}'$ mais cela signifie que
$u[x_1 := t_1, \cdots, x_n := t_n] = v[x_1 := t_1, \cdots, x_n := t_n]$ est un cas particulier d'une équation de $\mathcal{E}$ et c'est donc vrai.

Nous montrons ensuite que l'ensemble $\mathcal{M}^{T'} \setminus \mathcal{M}^T$ est infini dénombrable.

- On n'a pas $c_k \simeq t$ avec $t \in \mathcal{L}$. En effet, dans le cas contraire, $c_k = t$ serait un cas particulier d'une équation de $\mathcal{E}$. L'équation $x = t$ appartiendrait donc à $\mathcal{E}$ ce qui n'est pas possible puisque $x = t$ ne peut être vérifié dans $\mathcal{M}^N$, dont l'ensemble de base est infini dénombrable.

- On n'a pas non plus $c_i \simeq c_j$ si $i \neq j$, car, sinon, l'équation $x = y$ appartiendrait à $\mathcal{E}$ ce qui est impossible puisque $\mathcal{M}^N$ ne la vérifie pas.

Puisque $\Lambda_E / \simeq_{\boldsymbol{\delta}} \setminus |\mathcal{M}^N|$ est infini dénombrable, on peut prolonger la bijection i de $|\mathcal{M}^T|$ sur $|\mathcal{M}^N|$ en une bijection de $|\mathcal{M}^{T'}|$ sur $\Lambda_E / \simeq_{\boldsymbol{\delta}}$. Et on définit ainsi sur $\Lambda_E / \simeq_{\boldsymbol{\delta}}$ une structure de modèle de $\mathcal{E}$ vérifiant la condition sur le type de données N, modèle qui est une extension du modèle $\mathcal{M}^N$.

On peut donc choisir comme ensemble d'équations $\mathcal{E}$, l'ensemble des équations vérifiées par les fonctions que l'on cherche à programmer (y compris $\boldsymbol{s}, \mathbf{0}, r_\alpha, cas_\alpha, t_\alpha$) sur leur ensemble de définition (N^α dans le raisonnement ci-dessus). On est assuré de l'existence d'un modèle standard (vérifiant les

conditions sur les types de données) sur lequel l'ensemble d'équations $\mathcal{E}$ est vérifié, le théorème de programmation (6.2.5) peut donc s'appliquer.

Le raisonnement que nous venons de faire a utilisé une hypothèse assez particulière : l'existence d'un modèle $\mathcal{M}^N$ d'ensemble de base N^α. Un raisonnement moins particularisé, où l'hypothése serait la plus générale : l'existence d'un modèle des fonctions sur leur ensemble de définition, nécessiterait l'utilisation de modèles multi-sortés. Il n'a pas été entrepris.

Chapitre 7

Logique

Deux questions de logique se posent à propos du système $Ex2$.

7.1 Réduction des λ-termes et réduction des preuves

Nous savons qu'il existe entre les termes de notre λ-calcul $Ex2$ et les preuves qui leur sont associées dans la relation de typage une bijection. Cette correspondance (dite de Curry-Howard) est en fait un isomorphisme : à la réduction des λ-termes correspond la réduction des preuves associées. Examinons le cas de nos cinq règles de réduction (définition 2.2.1), L est un ensemble de dénominations de type contenant β et ne contenant pas α, $\alpha = F^i$.

La réduction de $(\lambda x\ u\ v)$ en $u[x:=v]$ correspond à la réduction de la preuve :

$$\dfrac{\dfrac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x\ u : A \to B}\, Tabs \qquad \Gamma \vdash v : A}{\Gamma \vdash (\lambda x\ u\ v) : B}\, Tapp$$

en la preuve :

$$\dfrac{\Gamma, x : A \vdash u : B \quad \Gamma \vdash v : A}{\Gamma \vdash u[x \leftarrow v] : B}\, Subs$$

La réduction de $(\varepsilon_\alpha\langle u\rangle\ v)$ en $\varepsilon_\alpha\langle u\rangle$ correspond à la réduction de la preuve :

$$\dfrac{\dfrac{\Gamma \vdash \varepsilon_\alpha\langle u\rangle : E^\alpha[a]}{\Gamma \vdash \varepsilon_\alpha\langle u\rangle : T \to E^\alpha[a]}\, Tprop \qquad \Gamma \vdash v : T}{\Gamma \vdash (\varepsilon_\alpha\langle u\rangle\ v) : E^\alpha[a]}\, Tapp$$

en la preuve :

$$\Gamma \vdash \varepsilon_\alpha\langle u\rangle : E^\alpha[a]$$

La réduction de $\tau_\alpha\langle \lambda x\, u, v\rangle$ en $\lambda x\, u$ correspond à la réduction de la preuve :

$$\dfrac{\dfrac{\Gamma \vdash \lambda x\, u : D^L[a]}{\Gamma \vdash \lambda x\, u : D^{L+\alpha}[a]}\, T \vee I2 \qquad \Gamma \vdash v : \forall x(F[x] \to D^L[f(x)])}{\Gamma \vdash \tau_\alpha\langle \lambda x\, u, v\rangle : D^L[t_\alpha(a,f)]}\, Ttry$$

en la preuve :

$$\dfrac{\Gamma \vdash \lambda x\, u : D^L[a] \qquad D^L[a] \to a = t_\alpha(a,f)\ (trapD)}{\Gamma \vdash \lambda x\, u : D^L[t_\alpha(a,f)]}\, Teq$$

La réduction de $\tau_\alpha\langle \varepsilon_\beta\langle u\rangle, v\rangle$ en $\varepsilon_\beta\langle u\rangle$ correspond à la réduction de la preuve :

$$\dfrac{\dfrac{\Gamma \vdash \varepsilon_\beta\langle u\rangle : D^L[a]}{\Gamma \vdash \varepsilon_\beta\langle u\rangle : D^{L+\alpha}[a]}\, T \vee I2 \qquad \Gamma \vdash v : \forall x(F[x] \to D^L[f(x)])}{\Gamma \vdash \tau_\alpha\langle \varepsilon_\beta\langle u\rangle, v\rangle : D^L[t_\alpha(a,f)]}\, Ttry$$

en la preuve :

$$\dfrac{\Gamma \vdash \varepsilon_\beta\langle u\rangle : D^L[a] \qquad D^L[a] \to a = t_\alpha(a,f)\ (trapD)}{\Gamma \vdash \varepsilon_\beta\langle u\rangle : D^L[t_\alpha(a,f)]}\, Teq$$

La réduction de $\tau_\alpha\langle \varepsilon_\alpha\langle u\rangle, v\rangle$ en $(v\ u)$ correspond à la réduction de la preuve :

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash u : F[a]}{\Gamma \vdash \varepsilon_\alpha\langle u\rangle : E^\alpha[r_\alpha(a)]}\, Texc}{\Gamma \vdash \varepsilon_\alpha\langle u\rangle : D^{L+\alpha}[r_\alpha(a)]}\, T \vee I1 \qquad \Gamma \vdash v : \forall x(F[x] \to D^L[f(x)])}{\Gamma \vdash \tau_\alpha\langle \varepsilon_\alpha\langle u\rangle, v\rangle : D^L[t_\alpha(r_\alpha(a),f)]}\, Ttry$$

en la preuve :

$$\dfrac{\dfrac{\Gamma \vdash v : F[a] \to D^L[f(a)] \qquad \Gamma \vdash u : F[a]}{\Gamma \vdash (v\ u) : D^L[f(a)]}\, Tapp \qquad F[a] \to f(a) = t_\alpha(r_\alpha(a),f)\ (trapE)}{\Gamma \vdash (v\ u) : D^L[t_\alpha(r_\alpha(a),f)]}\, Teq$$

Remarquons que la réduction de la preuve :

$$\dfrac{\dfrac{\Gamma \vdash v : D^L[a]}{\Gamma \vdash v : D^{L+\alpha}[a]}\, T \vee I2 \qquad \Gamma, x : D^L[a] \vdash u : T \quad \Gamma, x : E^\alpha[a] \vdash u : T}{\Gamma \vdash u[x \leftarrow v] : T}\, T \vee E$$

en la preuve :

$$\dfrac{\Gamma \vdash v : D^L[a] \quad \Gamma, x : D^L[a] \vdash u : T}{\Gamma \vdash u[x \leftarrow v] : T}\, Subs$$

ne correspond pas à une réduction de termes : le λ-terme est le même dans les deux cas. Cependant cette réduction de preuve est utilisée dans le processus de clarification des preuves (proposition 5.1.10). Il en est de même dans le cas d'une utilisation de la règle T$\vee I1$ suivie d'une utilisation de la règle T$\vee E$.

7.2 Logique de $Ex2$

Depuis le travail de Griffin ([7]), on relie opérateurs de contrôle et logique classique. Le système $Ex2$ intègre un opérateur de contrôle un peu particulier - les exceptions - aussi une question se pose naturellement à son sujet : quelle est sa logique? Précisons tout d'abord le sens de cette question.

Définition 7.2.1 *Etant donné une formule A et un ensemble fini de formules : $\mathcal{A} \stackrel{def}{=} A_1, A_2, \cdots, A_n$ de Φ_E, un ensemble d'équations $\mathcal{E}$, on dira que A est conséquence logique (dans $Ex2$) de $\mathcal{A}$, compte tenu de $\mathcal{E}$, et on écrira $\mathcal{A} \vdash^{\mathcal{E}}_L A$ s'il existe un λ-terme t et des variables $x_1, \cdots, x_n$ tels que $x_1 : A_1, \cdots, x_n : A_n \vdash^{\mathcal{E}} t : A$.*

Etant donné une formule B et un ensemble fini de formules : $\mathcal{B} \stackrel{def}{=} B_1, \cdots, B_n$ de Φ^F_E, on dira que B est conséquence logique (dans $Ex2_F$) de $\mathcal{B}$ et on écrira $\mathcal{B} \vdash^F_L B$ s'il existe un λ-terme t et des variables $x_1, \cdots, x_n$ tels que $x_1 : B_1, \cdots, x_n : B_n \vdash_F t : A$.

Le système de typage de $Ex2$ inclut un système logique de déduction (notée $\Vdash^{\mathcal{E}}$) utilisé pour prouver l'égalité des termes d'individus. Ci-dessus nous avons défini la relation de conséquence logique dans $Ex2$ ($\vdash^{\mathcal{E}}_L$) à partir de la relation de typage, sans faire référence à la relation $\Vdash^{\mathcal{E}}$. Il semble donc nécessaire de préciser le lien entre ces deux notions.

On a immédiatement le résultat : si $\mathcal{A} \vdash^{\mathcal{E}}_L A$ alors $\mathcal{A} \Vdash^{\mathcal{E}} A$. En effet l'hypothése signifie $x_1 : A_1, \cdots, x_n : A_n \vdash^{\mathcal{E}} t : A$, et de ce typage dans $Ex2$ on peut déduire directement une preuve de $A_1, A_2, \cdots, A_n \Vdash^{\mathcal{E}} A$ (lemme 3.2.4).

Mais la réciproque n'est pas vraie. En effet on a $\Vdash \neg(\mathbf{0} = \boldsymbol{s}(\mathbf{0}))$ (axiome égalitaire pluralité). Mais on peut montrer qu'il n'existe pas de terme t tel que $\vdash \neg(\mathbf{0} = \boldsymbol{s}(\mathbf{0}))$. Le raisonnement est le même que celui de la preuve du théorème de caractérisation des entiers (6.1.6) : on prouve qu'il n'existe pas de terme t fortement normalisable et clos de type $\forall X(X\mathbf{0} \to X\boldsymbol{s}(\mathbf{0})) \to \forall Y Y$.

Nous montrons ensuite un résultat qui relie les notions de conséquence logique dans $Ex2$ ($\vdash_L^{\mathcal{E}}$) et dans $Ex2_F$ ($\vdash_L^F$).

Lemme 7.2.2 *On désigne par A' la formule de Φ_E^F obtenue en « supprimant » dans la formule A de Φ_E la « partie du premier ordre » (voir la proposition 4.2.1 pour une définition plus précise de A'), et par $\mathcal{A}'$ l'ensemble de formules $A'_1, \cdots, A'_n$ (les notations sont celles de la définition 7.2.1).*

On a le résultat : si $\mathcal{A} \vdash_L^{\mathcal{E}} A$ alors $\mathcal{A}' \vdash_L^F A'$.

Preuve La preuve se déduit de façon immédiate de celle de la proposition 4.2.1. □

Puis nous traduisons les formules de Φ_E^F en formules de la logique propositionnelle intuitionniste du second ordre ($Lp2$).

Définition 7.2.3 *Si A est une formule de Φ_E^F sa traduction $\overline{A}$ est définie ainsi :*

- *si $\alpha = D_i$ alors $\overline{E^\alpha} = D$ où D est la formule de $Lp2$ associée au symbole de données D (par exemple $N = \forall X((X \to X) \to (X \to X))$) ;*
- *si L est un ensemble de dénominations de type, α une dénomination de type et D un symbole de données, $\overline{D^{L+\alpha}} = \overline{D^L} \vee \overline{E^\alpha}$ et $\overline{D^\emptyset} = D$;*
- *si X est une variable propositionnelle et A, B deux formules de Φ_E^F alors $\overline{X} = X$, $\overline{A \to B} = \overline{A} \to \overline{B}$, et $\overline{\forall X A} = \forall X \overline{A}$.*

On vérifie facilement que si $A \in \Phi_E^F$ alors $\overline{A}$ est une formule de $Lp2$ et que si A est une formule de $Lp2$ alors $A = \overline{A}$.

Nous pouvons maintenant relier conséquence logique dans $Ex2_F$ ($\vdash_L^F$) et déduction en logique propositionnelle intuitionniste du second ordre ($\vdash_{Lp2}$).

Lemme 7.2.4 *Etant donné une formule A et un ensemble fini de formules: $\mathcal{A} \stackrel{def}{=} A_1, A_2, \cdots, A_n$ de Φ_E^F, si (1) $\mathcal{A} \vdash_L^F A$ alors $\overline{\mathcal{A}} \vdash_{Lp2} \overline{A}$.*

Preuve La preuve se fait par récurrence sur le typage (1), nous examinons donc la dernière règle en nous limitant aux cas non triviaux.

- Si la dernière règle est T'exc, alors, de $\mathcal{A} \vdash_L^F D$, nous avons déduit $\mathcal{A} \vdash_L^F E^\alpha$. Mais puisque $\alpha = D_i$ on sait que $\overline{E^\alpha} = D$ et l'hypothèse de récurrence permet de conclure.

- Si la dernière règle est T'prop, le résultat est évident puisque si $\mathcal{B} \vdash_{Lp2} B$ alors $\mathcal{B} \vdash_{Lp2} C \to B$ pour toute formule C.
- Si la dernière règle est T'$\vee I1$ ou T'$\vee I2$, il suffit d'utiliser l'une des règles d'introduction du $\vee$ (dans $Lp2$) : si $\mathcal{B} \vdash_{Lp2} B$ alors $\mathcal{B} \vdash_{Lp2} C \vee B$ et $\mathcal{B} \vdash_{Lp2} B \vee C$.
- Si la dernière règle est T'$\vee E$, on utilise la règle (élimination du $\vee$): si $\mathcal{B}, C_1 \vdash_{Lp2} B$ et $\mathcal{B}, C_2 \vdash_{Lp2} B$ et $\mathcal{B} \vdash_{Lp2} C_1 \vee C_2$ alors $\mathcal{B} \vdash_{Lp2} B$.
- Si la dernière règle est T'try, alors, de $\mathcal{A} \vdash_L^F D^{L+\alpha}$ et $\mathcal{A} \vdash_L^F F \to D^L$, nous avons déduit $\mathcal{A} \vdash_L^F D^L$ ($\alpha = F_i$). Puisque $\overline{F} = F$ et $\overline{D^{L+\alpha}} = \overline{D^L} \vee \overline{E^\alpha} = \overline{D^L} \vee F$, l'hypothèse de récurrence nous donne $\overline{\mathcal{A}} \vdash_{Lp2} \overline{D^L} \vee F$ et $\overline{\mathcal{A}} \vdash_{Lp2} F \to \overline{D^L}$ d'où l'on peut déduire $\overline{\mathcal{A}} \vdash_{Lp2} \overline{D^L}$.

□

Les deux lemmes précédents ont une conséquence intéressante.

Proposition 7.2.5 *Pour toute formule A et tout ensemble fini de formules* $\mathcal{A} \stackrel{def}{=} A_1, A_2, \cdots, A_n$ *du calcul des propositions du second ordre, on a* $\mathcal{A} \vdash_L A$ *si et seulement si* $\mathcal{A} \vdash_{Lp2} A$

Preuve La preuve de la partie directe est simple. Soit en effet A une formule de $Lp2$, telle que $\mathcal{A} \vdash_L A$. La formule A vérifie $A = A' = \overline{A}$ et de même $\mathcal{A} = \mathcal{A}' = \overline{\mathcal{A}}$, donc (lemme 7.2.2) $\mathcal{A} \vdash_L^F A$ et (lemme 7.2.4) $\mathcal{A} \vdash_{Lp2} A$.

La réciproque se prouve tout aussi facilement. En effet toute formule de $Lp2$ est aussi une formule de $Ex2$ et toute règle de démonstration de $Lp2$ est aussi une règle de $Ex2$, donc la preuve de A dans $Lp2$ est une preuve de A dans $Ex2$. □

On en déduit qu'une formule non démontrable en logique propositionnelle intuitionniste du second ordre ne l'est pas non plus dans $Ex2$. En particulier les formules $\neg\neg X \to X$ et $((A \to B) \to A) \to A$ ne sont pas démontrables dans $Ex2$. La logique de $Ex2$ est la logique intuitionniste.

Chapitre 8

Exemples

Dans ce chapitre nous utilisons à plusieurs reprises des fonctions et des λ-termes particuliers de AF2. Rappelons leur définition.

- La fonction Si est l'alternative bien connue : $Si(\boldsymbol{v}, a, b) = a$ et $Si(\boldsymbol{f}, a, b) = b$.
- La fonction zerop est la fonction habituelle de test à zéro d'un entier : $zerop(\mathbf{0}) = \boldsymbol{v}$ et $zerop(\boldsymbol{s}(x)) = \boldsymbol{f}$; on notera Z un λ-terme de type $\forall n(N[n] \rightarrow B[zerop(n)])$.
- Les fonction add, prod et div sont les fonctions somme, produit et division euclidienne de deux entiers ; on notera A, P et D des termes de types respectifs $\forall x \forall y(N[x], N[y] \rightarrow N[add(x, y)])$, $\forall x \forall y(N[x], N[y] \rightarrow N[prod(x, y)])$ et $\forall x \forall y(N[x], N[y] \rightarrow N[div(x, y)])$.
- La fonction videp est la fonction qui teste si une liste est vide ou non : $videp(\emptyset) = \boldsymbol{v}$ et $videp(cons(t, q)) = \boldsymbol{f}$; on notera Z' un λ-terme de type $\forall x(L[x] \rightarrow B[videp(x)])$.

8.1 Opérateurs de mise en mémoire

Commençons par quelques rappels définitions.

8.1.1 Dans AF2

Nous parlerons dans la suite d'opérateurs de mise en mémoire sur les entiers (de Church), mais tout ce qui sera dit pourrait être transposé sans difficultés à d'autres ensembles comme les booléens, ou les listes d'entiers ... (pour une définition plus générale voir [18]).

Un opérateur de mise en mémoire sur les entiers est caractérisé par son comportement sur ces entiers.

Définition 8.1.1 *On dit que T λ-terme clos est un opérateur de mise en mémoire pour $\underline{N}$ (ensemble des entiers de Church) si et seulement si, pour tout $\underline{n}$ de $\underline{N}$, il existe un λ-terme $\tau_n \simeq_\beta \underline{n}$ tel que, pour tout $\theta_n \simeq_\beta \underline{n}$, il existe une substitution σ telle que $(T\ \theta_n f) \succ (f\ \sigma(\tau_n))$ (f est une nouvelle variable).*

Rappelons que, dans AF2, on peut trouver, pour chaque type de données, un ou plusieurs opérateurs de mise en mémoire ([18]).

Par exemple le terme $T_1 \stackrel{def}{=} \lambda n\,(n\ G\ \delta)$ où $G \stackrel{def}{=} \lambda x\,\lambda y\,(x\ \lambda z\,(y\ (\underline{S}\ z)))$ et $\delta \stackrel{def}{=} \lambda f\,(f\ \underline{0})$ est un opérateur de mise en mémoire pour les entiers de Church, il vérifie $(T_1\ \theta_n f) \succ (f\ \underline{S}^n \underline{0})$.

Le terme $T_2 \stackrel{def}{=} \lambda n\,\lambda f\,(n\ H\ f\ \underline{0})$ où $H = \lambda x\,\lambda y\,(x\ (\underline{S}\ y))$ est un autre opérateur de mise en mémoire pour les entiers, de même fonctionnement que T_1.

Les termes $\underline{0}$ et $\underline{S}$ sont les termes de ce nom dans AF2, de types respectifs $N[\mathbf{0}]$ et $\forall x(N[x] \to N[\boldsymbol{s}(x)])$. On pourra choisir $\underline{0} \stackrel{def}{=} \lambda f\,\lambda x\,x$ et $\underline{S} \stackrel{def}{=}$ $\lambda n\,\lambda f\,\lambda x\,(f\ (n\ f\ x))$.

Un théorème de Krivine précise les types caractéristiques des opérateurs de mise en mémoire.

Théorème 8.1.2 *Si, dans AF2, le terme T a, dans le contexte vide, le type $\forall x(N^*[x] \to \neg\neg N[x])$, alors T est un opérateur de mise en mémoire pour $\underline{N}$.*

Pour chaque formule F, $\neg F \stackrel{def}{=} F \to O$ et la formule F^*, une des transformées de Gödel de F, est obtenue en remplaçant chaque formule atomique A de F par $\neg A$ (O est un symbole de relation 0-aire).

On a donc $N^*[x] = \forall X\,(\forall y(\neg X\,y \to \neg X\,\boldsymbol{s}(y)), \neg X\,\mathbf{0} \to \neg X\,x)$.

Les opérateurs de mise en mémoire ont été introduits par Krivine pour simuler «l'appel par valeur» dans le cadre de «l'appel par nom» et nous les utiliserons pour cet usage: obliger les paramètres à passer en tête de façon à forcer la propagation des exceptions.

8.1.2 Dans $Ex2$

Dans la suite T désignera l'un des termes T_1 ou T_2 opérateurs de mise en mémoire pour les entiers de Church.

Nous montrons un lemme sur le typage de termes contenant T.

Lemme 8.1.3 *Quelle que soit la dénomination de type α:*

- *si $\Gamma \vdash t : N[n]$ et $\Gamma \vdash F : \forall x(N[x] \to N[f(x)])$ alors $\Gamma \vdash (T\ t\ F) : N[f(n)]$;*
- *$\vdash T : \forall x(E^\alpha[x] \to E^\alpha[x])$ (nous dirons que T propage les exceptions);*

– *si* $\Gamma \vdash t_1 : N[a]$ *et* $\Gamma \vdash t_2 : E^\alpha[b]$ *alors* $\Gamma \vdash (T\ t_1\ t_2) : E^\alpha[b]$.

Preuve Le premier résultat est vrai dans AF2 ([18]) et on vérifie facilement qu'il reste vrai dans Ex2 puisqu'il ne fait intervenir aucune des caractéristiques nouvelles de Ex2.

Rédigeons la preuve du deuxième résultat, pour l'opérateur T_1 par exemple. Dans le contexte vide G et δ sont typables ([18]) de types respectifs A_1 et A_2, donc, de $x : E^\alpha[x] \vdash x : A_1 \to E^\alpha[x]$ (Tvar et Tprop) et $x : E^\alpha[x] \vdash G : A_1$, on déduit (règle Tapp) $x : E^\alpha[x] \vdash (x\ G) : E^\alpha[x]$, puis, par le même raisonnement, $x : E^\alpha[x] \vdash (x\ G\ \delta) : E^\alpha[x]$ et donc le résultat cherché.

La preuve du troisième résultat est très simple. De $\Gamma \vdash t_2 : E^\alpha[b]$, on déduit (Tprop) $\Gamma \vdash t_2 : N[a] \to E^\alpha[b]$. Partant de $\Gamma \vdash t_1 : N[a]$, on montre facilement ([18]) que $\Gamma \vdash t_1 : N^*[a]$ et de $\vdash T : N^*[a] \to ((N[a] \to E^\alpha[b]) \to E^\alpha[b])$ (remplacement de O par $E^\alpha[b]$) on déduit facilement le résultat en appliquant deux fois la règle Tapp. □

8.2 Petits exemples et utilitaires

8.2.1 Projections

Dans AF2

On définit, dans AF2, la conjonction de la façon suivante : $F \wedge G \stackrel{def}{=} \forall X\,((F, G \to X) \to X)$. On peut alors obtenir (pour les couples (entier, booléen) par exemple) :

– un constructeur de couple

$$\vdash c \stackrel{def}{=} \lambda x\,\lambda y\,\lambda f\ (f\ x\ y) : \forall x \forall y (N[x], B[y] \to N[x] \wedge B[y])$$

– et les deux projections

$$\vdash p_1 \stackrel{def}{=} \lambda p\ (p\ \lambda x\,\lambda y\,x) : \forall x \forall y (N[x] \wedge B[y] \to N[x])$$

$$\vdash p_2 \stackrel{def}{=} \lambda p\ (p\ \lambda x\,\lambda y\,y) : \forall x \forall y (N[x] \wedge B[y] \to B[y])$$

Mais, en utilisant les exceptions, nous pouvons, dans notre système, créer de nouveaux termes pour les projections.

Dans *Ex2*

Exemple 8.2.1 *Les termes* $\pi_1 \stackrel{def}{=} \lambda p\,\tau_\alpha\langle(p\ \lambda x\,\varepsilon_\alpha\langle x\rangle), \lambda z\,z\rangle$ *et* $\pi_2 \stackrel{def}{=} \lambda p\,\tau_\beta\langle(p\ \lambda x\,\lambda y\,\varepsilon_\beta\langle y\rangle), \lambda z\,z\rangle$ *conviennent respectivement pour première et deuxième projection* ($\alpha = N$ *et* $\beta = B$).

Preuve Commençons par la première projection. On sait que $x : N[x] \vdash \varepsilon_\alpha\langle x\rangle : E^\alpha[r_\alpha(x)]$ et donc (Tprop) $x : N[x] \vdash \varepsilon_\alpha\langle x\rangle : B[y] \to E^\alpha[r_\alpha(x)]$. On en déduit $\vdash \lambda x\, \varepsilon_\alpha\langle x\rangle : N[x], B[y] \to E^\alpha[r_\alpha(x)]$.

Si p a le type $N[x] \wedge B[y]$ donc le type $(N[x], B[y] \to E^\alpha[r_\alpha(x)]) \to E^\alpha[r_\alpha(x)]$, alors $(p\ \lambda x\, \varepsilon_\alpha\langle x\rangle)$ a le type $E^\alpha[r_\alpha(x)]$ donc $N^\alpha[r_\alpha(x)]$ et $\tau_\alpha\langle(p\ \lambda x\, \varepsilon_\alpha\langle x\rangle), \lambda z\, z\rangle$ le type $N[t_\alpha(r_\alpha(x), z \to z)]$ soit, puisque $\Vdash N[x]$, le type $N[x]$ (axiome égalitaire TrapD).

La preuve dans le cas de π_2 est un peu différente puisqu'elle n'utilise pas la règle Tprop.

On sait que $y : N[y] \vdash \varepsilon_\beta\langle y\rangle : E^\beta[r_\beta(y)]$ donc $\vdash \lambda x\ \lambda y\, \varepsilon_\beta\langle y\rangle : N[x], B[y] \to E^\beta[r_\beta(y)]$.

Et, puisque p a le type $N[x] \wedge B[y]$ donc le type $(N[x], B[y] \to E^\beta[r_\beta(y)]) \to E^\beta[r_\beta(y)]$, alors $(p\ \lambda x\ \lambda y\, \varepsilon_\beta\langle y\rangle)$ a le type $E^\beta[r_\beta(y)]$ donc $B^\beta[r_\beta(y)]$.
Et $\tau_\beta\langle(p\ \lambda x\ \lambda y\, \varepsilon_\beta\langle y\rangle), \lambda z\, z\rangle$ a alors le type $B[t_\beta(r_\beta(y), z \to z)]$ soit, puisque $\Vdash B[y]$, le type $B[y]$ (axiome égalitaire TrapD). □

Examinons le fonctionnement de ces projections sur le couple $C \stackrel{def}{=} \lambda f\, (f\ \underline{1}\ \underline{F})$.

- On a $(\pi_1\ C) \to_\delta \tau_\alpha\langle(C\ \lambda x\, \varepsilon_\alpha\langle x\rangle), \lambda z\, z\rangle \to_\delta \tau_\alpha\langle(\lambda x\, \varepsilon_\alpha\langle x\rangle\ \underline{1}\ \underline{F}), \lambda z\, z\rangle$
  $\to_\delta \tau_\alpha\langle(\varepsilon_\alpha\langle\underline{1}\rangle\ \underline{F}), \lambda z\, z\rangle \to_\delta \tau_\alpha\langle\varepsilon_\alpha\langle\underline{1}\rangle, \lambda z\, z\rangle \to_\delta (\lambda z\, z\ \underline{1}) \to_\delta \underline{1}$

- On a $(\pi_2\ C) \to_\delta \tau_\beta\langle(C\ \lambda x\ \lambda y\, \varepsilon_\beta\langle y\rangle), \lambda z\, z\rangle \to_\delta \tau_\beta\langle(\lambda x\ \lambda y\, \varepsilon_\beta\langle y\rangle\ \underline{1}\ \underline{F}), \lambda z\, z\rangle$
  $\to_\delta \tau_\beta\langle\varepsilon_\beta\langle\underline{F}\rangle, \lambda z\, z\rangle \to_\delta (\lambda z\, z\ \underline{F}) \to_\delta \underline{F}$.

Ces deux termes sont inspirés de projections construites par P. de Groote ([5]). Il qualifie ses projections de classiques car, dans son système de typage (simple), il leur attribue les types $((A \to (B \to \bot)) \to \bot) \to A$ et $((A \to (B \to \bot)) \to \bot) \to B$ soit en logique classique $(A \wedge B) \to A$ et $(A \wedge B) \to B$.

8.2.2 Prolongement de fonctions sur les types de données

Notre premier exemple illustrera la prolongation d'une fonction de N dans N.

Exemple 8.2.2 *Considérons un λ-terme F_1 tel que $\vdash F_1 : \forall x(N[x] \to N[f(x)])$ où f est une fonction de N dans N. Alors le terme $t = \lambda x\, (T\ x\ F_1)$ a dans le contexte vide le type $\forall x(N^\alpha[x] \to N^\alpha[g(x)])$ où la fonction g est définie par l'équation $g(x) = cas_\alpha(x, x, f(x))$.*

Preuve D'une part, d'après le lemme 8.1.3, $x : N[x] \vdash (T\ xF_1) : N[f(x)]$. On en déduit, en utilisant l'axiome égalitaire casD $(N[x] \to cas_\alpha(x, a, b) = b)$ et la règle Teq, le typage: (1) $x : N[x] \vdash (T\ xF_1) : N[cas_\alpha(x, x, f(x))]$.

D'autre part on a $x : E^\alpha[x] \vdash (T\ x) : E^\alpha[x]$ (8.1.3), et donc (Tprop et Tapp) $x : E^\alpha[x] \vdash (T\ xF_1) : E^\alpha[x]$. On utilise ensuite la règle T$\vee I1$ pour

obtenir $x : E^\alpha[x] \vdash (T\ xF_1) : N^\alpha[x]$, puis l'axiome égalitaire casE ($E^\alpha[x] \to cas_\alpha(x,a,b) = a$) et la règle Teq pour obtenir (2) $x : E^\alpha[x] \vdash (T\ xF_1) : N^\alpha[cas_\alpha(x,x,f(x))]$.

On déduit de (1) et (2), par utilisation du cas particulier de la règle T$\vee E$ (lemme 3.4.2), $x : N^\alpha[x] \vdash (T\ xF_1) : N^\alpha[cas_\alpha(x,x,f(x))]$ et la règle Tabs donne le résultat. □

Nous sommes partis d'une fonction f de N dans N, pour laquelle le terme F_1 constituait un programme. La fonction f a été prolongée par une fonction g de N^α dans N^α qui propage les exceptions en ce sens que $\Vdash N[x] \to g(x) = f(x)$ et $\Vdash E^\alpha[x] \to g(x) = x$. On a alors trouvé un terme qui constitue un programme pour la fonction g.

Il est plus difficile de prolonger une fonction de $N \times N$ vers N, parce qu'il faut choisir, lorsqu'on se trouve en présence de deux exceptions, laquelle des deux propager. C'est l'objet de notre deuxième exemple.

Exemple 8.2.3 *Considérons un λ-terme F_2 tel que $\vdash F_2 : \forall x \forall y (N[y], N[x] \to N[f(x,y)])$ où f est une fonction de $N \times N$ dans N. Alors le terme $t = \lambda x\, \lambda y\, (T\ x\ (T\ y\ F_2))$ a dans le contexte vide le type: $\forall x \forall y (N^\alpha[x], N^\beta[y] \to N^{\alpha,\beta}[g(x,y)]$ où la fonction g est définie par l'équation $g(x,y) = cas_\alpha(x,x,cas_\beta(y,y,f(x,y)))$.*

Preuve Posons $v = (T\ y\ F_2)$ et $u = (T\ x\ v)$. Nous allons successivement typer u dans le contexte $x : N[x], y : N^\beta[y]$ puis dans le contexte $x : E^\alpha[x], y : N^\beta[y]$.

- Commençons par le contexte $x : N[x], y : N^\beta[y]$.
  - On a (1) $x : N[x], y : N[y] \vdash u : N[f(x,y)]$.
    En effet $\vdash T : N^*[y] \to ((N[y] \to A) \to A)$ avec $A = N[x] \to N[f(x,y)]$ donc $y : N[y] \vdash (T\ yF_2) : N[x] \to N[f(x,y)]$.
    Mais $\vdash T : N^*[x] \to ((N[x] \to B) \to B)$ avec $B = N[f(x,y)]$.
    Donc $x : N[x], y : N[y] \vdash (T\ x) : (N[x] \to N[f(x,y)]) \to N[f(x,y)]$ et finalement $x : N[x], y : N[y] \vdash (T\ x\ v) : N[f(x,y)]$.
  - On a (2) $x : N[x], y : E^\beta[y] \vdash u : E^\beta[y]$.
    En effet $y : E^\beta[y] \vdash (T\ y) : E^\beta[y]$ d'après le lemme 8.1.3, puis $y : E^\beta[y] \vdash v : E^\beta[y]$ d'après le lemme 3.4.3. Et le résultat s'en déduit en utilisant à nouveau le lemme 8.1.3.

  Nous transformons (1) en $x : N[x], y : N[y] \vdash u : N^\beta[f(x,y)]$ (T$\vee I1$) puis en (3) $x : N[x], y : N[y] \vdash u : N^\beta[cas_\beta(y,y,f(x,y))]$ en utilisant Teq et l'axiome égalitaire casD.

  De même nous transformons (2) en $x : N[x], y : E^\beta[y] \vdash u : N^\beta[y]$ (T$\vee I2$) puis en (4) $x : N[x], y : E^\beta[y] \vdash u : N^\beta[cas_\beta(y,y,f(x,y))]$ en utilisant Teq et l'axiome égalitaire casE.

Le cas particulier de la règle T$\vee E$ (lemme 3.4.2) permet alors d'obtenir (*a*) $x : N[x], y : N^{\beta}[y] \vdash u : N^{\beta}[cas_{\beta}(y, y, f(x, y))]$.

- Passons au contexte $x : E^{\alpha}[x], y : N^{\beta}[y]$.
  - On a (1) $x : E^{\alpha}[x], y : N[y] \vdash u : E^{\alpha}[x]$.
    En effet, dans ce contexte, $(T\ x)$ a le type $E^{\alpha}[x]$ et, puisque v est typable dans ce contexte, l'application de Tprop et Tapp permet de conclure.
  - On a (2) $x : E^{\alpha}[x], y : E^{\beta}[y] \vdash u : E^{\alpha}[x]$. En effet $y : E^{\beta}[y] \vdash v : E^{\beta}[y]$, d'après le lemme 8.1.3. et le raisonnement se poursuit comme ci-dessus.

  Le cas particulier de la règle T$\vee E$ (lemme 3.4.2) nous permet d'obtenir (*b*) $x : E^{\alpha}[x], y : N^{\beta}[y] \vdash u : E^{\alpha}[x]$.

- Il nous reste à «fusionner» les typages (a) et (b).
  - Par utilisation tout d'abord de T$\vee I2$ puis de Teq et de l'axiome égalitaire casD, (a) se transforme en (*c*) $x : N[x], y : N^{\beta}[y] \vdash u : N^{\alpha,\beta}[cas_{\alpha}(x, x, cas_{\beta}(y, y, f(x, y)))]$.
  - Par utilisation tout d'abord de T$\vee I1$ et T$\vee I2$ puis de Teq et de l'axiome égalitaire casE, (b) se transforme en (*d*) $x : E^{\alpha}[x], y : N^{\beta}[y] \vdash u : N^{\alpha,\beta}[cas_{\alpha}(x, x, cas_{\beta}(y, y, f(x, y)))]$.
  - A partir de (c) et (d), le cas particulier de la règle T$\vee E$ (lemme 3.4.2) nous permet d'obtenir
    $x : N^{\alpha}[x], y : N^{\beta}[y] \vdash u : N^{\alpha,\beta}[cas_{\alpha}(x, x, cas_{\beta}(y, y, f(x, y)))]$
    et donc de conclure.

□

Il n'est pas toujours nécéssaire d'utiliser les opérateurs de mise en mémoire pour prolonger aux exceptions une fonction sur un type de données. Considérons la fonction « puissance n-ième de deux » définie sur N par $f(\mathbf{0}) = \boldsymbol{s}(\mathbf{0}), f(\boldsymbol{s}(x)) = mult(f(x), \boldsymbol{s}(\boldsymbol{s}(\mathbf{0})))$.

Dans AF2 (et dans $Ex2$) on peut montrer ([10]) que, dans le contexte vide, le terme $t = \lambda x\,(x\ \underline{2})$ a le type $\forall x(N[x] \to N[f(x)])$. On a donc $x : N[x] \vdash (x\ \underline{2}) : N[f(x)]$ et donc $x : N[x] \vdash (x\ \underline{2}) : N^{\alpha}[cas_{\alpha}(x, x, f(x))]$.

Mais on a aussi (Tprop et Tapp) $x : E^{\alpha}[x] \vdash (x\ \underline{2}) : E^{\alpha}[x]$ et donc (T$\vee I1$ puis Teq et l'axiome égalitaire casE) $x : E^{\alpha}[x] \vdash (x\ \underline{2}) : N^{\alpha}[cas_{\alpha}(x, x, f(x))]$. Soit, en appliquant là aussi le cas particulier de la règle T$\vee E$, $x : N^{\alpha}[x] \vdash (x\ \underline{2}) : N^{\alpha}[cas_{\alpha}(x, x, f(x))]$. Et finalement $\vdash t : \forall x(N^{\alpha}[x] \to N^{\alpha}[cas_{\alpha}(x, x, f(x))]$.

On peut dire que le terme t est un programme pour la fonction g de N^{α} dans N^{α} définie par $g(x) = cas_{\alpha}(x, x, f(x))$ qui est une prolongation de la fonction f aux exceptions.

8.3 Produit rapide

8.3.1 Produit rapide des éléments d'une liste

Nous nous proposons de calculer le produit des éléments d'une liste d'entiers. Et pour le calculer efficacement, nous utiliserons les exceptions.

Rappelons que le type des listes d'entiers est :

$$LN[x] \stackrel{def}{=} \forall X(\forall y \forall z(N[y], Xz \to X\,cons(y,z)), X\emptyset \to Xx)$$

Equations Les équations définissant la fonction f sont :
$f(l) = t_\alpha(f'(l), x \to x)$ et (définition par induction de f')
$f'(\emptyset) = 1$; $f'(cons(n,q)) = Si(zerop(n), r_\alpha(\mathbf{0}), prodv(n, f'(q)))$
(avec $prodv(x,y) = cas_\alpha(y, y, prod(x,y))$).

Exemple 8.3.1 *Le terme* $t \stackrel{def}{=} \lambda l\, \tau_\alpha\langle(l\ V\ \underline{1}), \lambda x\, x\rangle$ *où* $V = \lambda n\, \lambda z\, ((Z\ n)\ \varepsilon_\alpha\langle\mathbf{0}\rangle\ (P'\ n\ z))$ *et* $P' = \lambda x\, \lambda y\, (T\ y\ (P\ x))$ *a dans le contexte vide le type* $\forall l(LN[l] \to N[f(l)])$.

Preuve

Nous montrons tout d'abord que $l : LN[l] \vdash w \stackrel{def}{=} (l\ V\ \underline{0}) : N^\alpha[f'(l)]$.

- Commençons par le typage de V.
  - Tout d'abord $\vdash P' : \forall x \forall y(N[x], N^\alpha[y] \to N^\alpha[cas_\alpha(y,y,prod(x,y))]$.
    En effet, $x : N[x] \vdash (P\ x) : N[y] \to N[prod(x,y)]$ donc, d'apès l'exemple 8.2.2, $x : N[x], y : N^\alpha[y] \vdash (T\ y\ (P\ x)) : N^\alpha[cas_\alpha(y,y,prod(x,y))]$.
  - On en déduit
    $n : N[n], z : N^\alpha[f'(q)] \vdash (P'\ n\ z) : N^\alpha[cas_\alpha(f'(q), f'(q), prod(n, f'(q)))]$.
    Puis, en utilisant la règle Teq et les égalités définissant f',
    $$n : N[n], z : N^\alpha[f'(q)] \vdash (P'\ n\ z) : N^\alpha[prodv(n, f'(q))]$$
  - Mais $\vdash \varepsilon_\alpha\langle\underline{0}\rangle : E^\alpha[r_\alpha(\mathbf{0})]$ donc (T$\vee I2$) $\vdash \varepsilon_\alpha\langle\underline{0}\rangle : N^\alpha[r_\alpha(\mathbf{0})]$
  - Donc
    $n : N[n], z : N^\alpha[f'(q)] \vdash ((Z\ n)\ \varepsilon_\alpha\langle\underline{0}\rangle\ (P'\ n\ z)) : N^\alpha[SI(zerop(n), r_\alpha(\mathbf{0}), prodv(n, f'(q)))]$

  On a alors le typage de V: $\forall n \forall q(N[n], N^\alpha[f'(q)] \to N^\alpha[f'(cons(n,q))]$.
- Passons au terme $\underline{1}$. On a $\vdash \underline{1} : N[1]$ et donc (Teq) $\vdash \underline{0} : N[f'(\emptyset)]$ puis $\vdash \underline{1} : N^\alpha[f'(\emptyset)]$ (T$\vee I2$).
- Mais, en remplaçant dans le type $LN[l]$, X. par $N^\alpha[f'(.)]$, on obtient :
  $l : LN[l] \vdash l : \forall n \forall q(N[n], N^\alpha[f'(q)] \to N^\alpha[f'(cons(n,q))]), N^\alpha[f'(\emptyset)] \to N^\alpha[f'(l)]$
  Ce qui nous donne bien $l : LN[l] \vdash (l\ V\ \underline{1}) : N^\alpha[f'(l)]$.

De $l : LN[l] \vdash w : N^\alpha[f'(l)]$ et $\vdash \lambda x\, x : \forall x(N[x] \to N[x])$ on déduit (Ttry) $l : LN[l] \vdash \tau_\alpha\langle w, \lambda x\, x\rangle : N^\alpha[t_\alpha(f'(l), x \to x)]$ soit (Teq) $l : LN[l] \vdash \tau_\alpha\langle w, \lambda x\, x\rangle : N^\alpha[f(l)]$ ce qui conclut notre preuve. □

8.3.2 Produit rapide des éléments d'un arbre

Nous calculons maintenant le produit des éléments d'un arbre binaire d'entiers, et là aussi nous utilisons les exceptions.

Rappelons que le type des arbres binaires d'entiers (dans la variante où les entiers sont accrochés aux feuilles) est :

$$AN[x] \stackrel{def}{=} \forall X(\forall y \forall z(Xy, Xz \to Xtree(y,z)), \forall n(N[n] \to Xfeu(n)) \to Xx)$$

Equations Les équations définissant notre produit des entiers d'un arbre sont :

$f(a) = t_\alpha(f'(a))$ et (définition par induction de f')
$f'(feu(n)) = Si(zerop(n), r_\alpha(\mathbf{0}), n)$; $f'(tree(g,d)) = prodvv(f'(g), f'(d))$
(avec $prodvv(x,y) = cas_\alpha(x, x, cas_\alpha(y, y, prod(x,y)))$).

Exemple 8.3.2 *Le terme* $t \stackrel{def}{=} \lambda a\, \tau_\alpha\langle(a\ V\ U), \lambda z\, z\rangle$ *où*
$V = \lambda g\, \lambda d\, (T\ g\ (T\ d\ \lambda x\, \lambda y\, (P\ y\ x)))$ *et* $U = \lambda n\, ((Z\ n)\ \varepsilon_\alpha\langle\underline{0}\rangle\ n)$
a dans le contexte vide le type $\forall a(AN[a] \to N[f(a)])$.

Preuve Nous montrons tout d'abord que $a : AN[a] \vdash (a\ V\ U) : N^\alpha[f'(a)]$.

- Le typage $\vdash V : \forall x \forall y(N^\alpha[x], N^\alpha[y] \to N^\alpha[prodvv(x,y)]$ est une conséquence directe de l'exemple 8.2.3 car $\vdash \lambda x\, \lambda y\, (P\ y\ x) : N[x], N[y] : N[prod(y,x)]$.

  On en déduit (Teq) $\vdash V : \forall g \forall d(N^\alpha[f'(g)], N^\alpha[f'(d)] \to N^\alpha[f'(tree(g,d))]$.

- Passons au terme U. Il a le type $\forall n(N[n] \to N^\alpha[Si(zerop(n), r_\alpha(\mathbf{0}), n)]$ soit (Teq) $\forall n(N[n] \to N^\alpha[f'(feu(n))]$.

- Mais, en remplaçant dans le type $AN[a]$ la formule X. par $N^\alpha[f'(.)]$, on obtient le type

  $$\forall x \forall y(N^\alpha[f'(x)], N^\alpha[f'(y)] \to N^\alpha[f'(tree(x,y))], \forall n(N[n] \to N^\alpha[f'(feu(n))] \to N^\alpha[f'(a)]$$

  Ce qui nous donne bien $a : AN[a] \vdash (a\ V\ U) : N^\alpha[f'(a)]$.

De $a : AN[a] \vdash (a\ V\ U) : N^\alpha[f'(a)]$ on déduit, en appliquant Ttry,

$$a : AN[a] \vdash \tau_\alpha\langle(a\ V\ U), \lambda z\, z\rangle : N^\alpha[t_\alpha(f'(a), z \to z)]$$

soit (Teq) $a : AN[a] \vdash \tau_\alpha\langle(a\ V\ U), \lambda z\, z\rangle : N^\alpha[f(a)]$ qui est le résultat cherché. □

8.3.3 Produit rapide et produit simple

Nous nous proposons de montrer l'égalité du produit rapide précédemment défini et du produit habituel.

Produit habituel Le produit des éléments d'un arbre binaire d'entiers est défini classiquement par les équations : $h(feu(n)) = n$ et $h(tree(g,d)) = prod(h(g), h(d))$.

Nous montrons que les fonctions h (produit) et f (produit rapide) sont égales.

Preuve Nous nous plaçons dans l'hypothèse où a est un arbre binaire d'entiers ($AN[a]$) et nous voulons prouver $f(a) = h(a)$.

Nous utiliserons les cinq résultats préalables suivants : (1) $N[f(a)]$, (2) $N^\alpha[f'(a)]$, (3) $E^\alpha[f'(a)] \rightarrow f'(a) = r_\alpha(\mathbf{0})$, (4) $E^\alpha[f'(a)] \rightarrow h(a) = \mathbf{0}$, (5) $N[f'(a)] \rightarrow f'(a) = h(a)$. Les résultats (1) et (2) ont été prouvés au paragraphe 8.3.2. Et nous prouvons (3), (4) et (5) simultanément par induction sur a.

Si a est une feuille : feu(n), la preuve se fait par récurrence sur l'entier n.

- Si n est $\mathbf{0}$, $f'(a) = f'(feu(\mathbf{0})) = r_\alpha(\mathbf{0})$ et $h(a) = h(feu(\mathbf{0})) = \mathbf{0}$ ce qui prouve (3) et (4). On a $E^\alpha[f'(a)]$ donc (lemme 3.4.6) $\neg N[f'(a)]$ ce qui prouve (5).

- Si n est $\boldsymbol{s}(p)$, on a $f'(a) = n = h(a)$ - ce qui prouve (5) - et $N[f'(a)]$ donc (lemme 3.4.6) $\neg E^\alpha[f'(a)]$ - ce qui prouve (3) et (4).

On suppose que les propriétés (3) (4) et (5) sont vérifiées par les arbres g et d et on les montre pour l'arbre a égal à $tree(g,d)$. Rappelons que $f'(tree(g,d)) = cas_\alpha(f'(g), f'(g), cas_\alpha(f'(d), f'(d), prod(f'(g), f'(d))))$. Nous raisonnons par disjonction des cas.

- Si $E^\alpha[f'(g)]$ alors $f'(a) = f'(g)$ (casE) et $f'(g) = r_\alpha(\mathbf{0})$ (hypothèse de récurrence) - ce qui prouve (3) - $h(g) = \mathbf{0}$ (hypothèse de récurrence) donc $h(a) = prod(h(g), h(d)) = \mathbf{0}$ (équations de prod) - ce qui prouve (4). Puisque $E^\alpha[f'(a)]$, on a (lemme 3.4.6) $\neg N[f'(a)]$ et donc (5).

- Si $N[f'(g)]$ on doit distinguer selon $f'(d)$.

  - Si $E^\alpha[f'(d)]$ alors $f'(a) = f'(d)$ (casD et casE) et $f'(d) = r_\alpha(\mathbf{0})$ (hypothèse de récurrence) - ce qui prouve (3) - $h(d) = \mathbf{0}$ (hypothèse de récurrence) donc $h(a) = prod(h(g), h(d)) = \mathbf{0}$ (équations de prod) - ce qui prouve (4). Et puisque $E^\alpha[f'(a)]$, le lemme 3.4.6) nous donne $\neg N[f'(a)]$ et (5).

- Si $N[f'(d)]$ alors $f'(a) = prod(f'(g), f'(d))$, et donc $N[f'(a)]$ ce qui implique (lemme 3.4.6) $\neg E^{\alpha}[f'(a)]$ et donc (3) et (4). Par hypothèse de récurrence $f'(g) = h(g)$ et $f'(d) = h(d)$, on obtient donc (5) puisque $h(a) = prod(h(g), h(d))$.
- L'application de la règle L$\vee E$, sachant que $N^{\alpha}[f'(d)]$ nous garantit (3) (4) et (5) dans l'hypothèse $N[f'(g)]$.

- Une nouvelle application de la règle L$\vee E$, sachant que $N^{\alpha}[f'(g)]$ montre que les propriétés (3) (4) et (5) sont vérifiées, ce qui termine la preuve des cinq résultats préalables.

Puisqu'on sait que $f(a) = t_{\alpha}(f'(a))$, on déduit de (3), (4) et trapE :
$E^{\alpha}[f'(a)] \Vdash f(a) = t_{\alpha}(r_{\alpha}(\mathbf{0})) = \mathbf{0} = g(a)$,
et de (5) et trapD : $N[f'(a)] \Vdash f(a) = t_{\alpha}(f'(a)) = f'(a) = g(a)$.
Il suffit alors d'appliquer la règle L$\vee E$ aux deux résultats précédents et à $N^{\alpha}[f'(a)]$ pour conclure que $\Vdash f(a) = h(a)$. □

8.4 Modularité

A travers deux exemples, nous montrons la possibilité d'utiliser des «sous-programmes» sans en connaître autre chose que les spécifications.

8.4.1 Recherche dans un agenda

On possède un agenda c'est à dire une liste de répertoires qui sont eux mêmes des listes de couples (nom, numéro de téléphone). On dispose d'un programme de recherche dans un répertoire qui à partir d'un nom nous fournit le numéro de téléphone correspondant si le nom figure dans le répertoire et une exception sinon. On veut construire un programme de recherche dans l'agenda qui puisse parcourir successivement chacun des répertoires.

Equations La fonction f de recherche dans l'agenda est définie par les équations : $f(\emptyset, n) = r_{\alpha}(\underline{1})$ et $f(cons'(l, q), n) = t_{\alpha}(g(l, n), x \to f(q, n))$ où g est la fonction de recherche dans un répertoire.

La fonction g rend un entier si le numéro est trouvé ou $r_{\alpha}(p)$ avec p entier arbitraire et $\alpha = N$ sinon.

La valeur renvoyée par f en cas de recherche infructueuse dans l'agenda est une exception mais l'entier associé $\underline{1}$ est là aussi arbitraire.

Types On définit le type agenda ($A[x]$) comme le type des listes de répertoires et le type répertoire ($R[x]$) comme le type des listes de couples d'entiers, soit si C désigne le type couple d'entiers :

$$R[x] \stackrel{def}{=} \forall X\,(\forall y \forall z(C[y], X\,z \to X\,cons(y, z)), X_{\emptyset} \to X\,x)$$

$$A[x] \stackrel{def}{=} \forall X\,(\forall y \forall z(R[y], X\,z \to X\,cons'(y,z)), X_\emptyset \to X\,x)$$

Exemple 8.4.1 *le terme* $t = \lambda a\,\lambda n\,(a\ V\ U)$ *où* $V = \lambda l\,\lambda q\,\tau_\alpha\langle(G\ l\ n), \lambda x\,q\rangle$ *et* $U = \varepsilon_\alpha\langle\underline{1}\rangle$ *a dans le contexte vide le type* $\forall a \forall n(A[a], N[n] \to N^\alpha[f(a)])$ *si* G *a le type* $\forall l \forall n(R[l], N[n] \to N^\alpha[g(l,n)])$.

Preuve

- Nous montrons tout d'abord que:

$$n : N[n] \vdash V : \forall l \forall q(R[l], N^\alpha[f(q,n)] \to N^\alpha[f(cons'(l,q),n)])$$

  En effet $l : R[l], n : N[n] \vdash (G\ l\ n) : N^\alpha[g(l,n)]$ donc $l : R[l], n : N[n], z : N^\alpha[z] \vdash \tau_\alpha\langle(G\ l\ n), \lambda x\ z\rangle : N^\alpha[t_\alpha(g(l,n), x \to z)]$.

- Il est facile de vérifier que: $n : N[n] \vdash U : N^\alpha[f(\emptyset, n)]$.

- On en déduit aisément (induction sur a) que $a : A[a], n : N[n] \vdash (a\ V\ U) : N^\alpha[f(a)])$ ce qui termine la preuve.

□

Nous proposons, pour la recherche dans un agenda, un autre terme basé sur d'autres équations.

Equations La fonction h de recherche dans l'agenda est définie cette fois par les équations:
$h(l,n) = t_\beta(h'(l,n))$ avec $h'(\emptyset, n) = r_\alpha(\underline{1})$ et $h'(cons(l,q),n) = cas_\beta(j(g(l,n)), j(g(l,n)), h'(q,n))$ où $j(x) = cas_\alpha(x, t_\alpha(x), r_\beta(x))$.

Terme On obtient, de façon directe à partir des nouvelles équations, comme terme de type $\forall a \forall n(A[a], N[n] \to N^\alpha[h(a)])$ le nouveau terme $t' = \lambda a\,\tau_\beta\langle(a\ V\ U), \lambda x\,x\rangle$ avec $U = \varepsilon_\alpha\langle\underline{1}\rangle$ et $V = \lambda l\,\lambda q\,(T\ (H\ (G\ l))\ \lambda z\,q)$ (où $H = \lambda x\,\tau_\alpha\langle(T\ x\ \lambda y\,\varepsilon_\beta\langle y\rangle), \lambda z\,z\rangle$).

Il est assez facile de vérifier que t' a bien le type indiqué. Il est plus intéréssant de montrer que les deux fonctions de recherche dans un agenda sont égales. Nous prouvons ci-dessous l'égalité des deux fonctions f et h.

Preuve Rappelons que la fonction g rend un entier si le numéro de téléphone est trouvé ou $r_\alpha(p)$ avec p entier arbitraire et $\alpha = N$ sinon. On a donc $\Vdash \forall l \forall n(R[l], N[n] \to N^\alpha[g(l,n)])$.

Nous montrons que $f(a,n) = h(a,n)$ par induction sur l'agenda a.

On a $h(\emptyset, n) = t_\beta(h'(\emptyset, n)) = t_\beta(r_\alpha(\underline{1})) = r_\alpha(\underline{1}) = f(\emptyset, n)$ en utilisant l'axiome égalitaire trapD.

On montre ensuite que $f(cons(l,q),n) = h(cons(l,q),n)$ si $f(q,n) = h(q,n)$. Posons $g = g(l,n)$ pour simplifier les écritures, et distinguons deux cas.

- Si $N[g]$, alors $f(cons(l,q),n) = t_\beta(g, x \to f(q,n)) = g$ (trapD).

  Et $h(cons(l,q),n) = t_\beta(h'(l,n)) = t_\beta(cas_\beta(j(g), j(g), h'(q,n)))$. Avec $j(g) = cas_\alpha(g, t_\alpha(g), r_\beta(g)) = r_\beta(g)$ (trapD), on obtient $h(cons(l,q),n) = t_\beta(cas_\beta(r_\beta(g), r_\beta(g), h'(q,n))) = t_\beta(r_\beta(g)) = g$ (casD puis trapE). L'égalité est donc prouvée dans ce premier cas.

- Si $E^\alpha[g]$ alors, par définition de g, $g = r_\alpha(a)$ avec $N[a]$.

  On a alors $f(cons(l,q),n) = t_\alpha(r_\alpha(a), x \to f(q,n)) = f(q,n)$ (trapE puisque $\alpha \neq \beta$). Et $h(cons(l,q),n) = t_\beta(cas_\beta(j(g), j(g), h'(q,n)))$.

  Avec $j(g) = cas_\alpha(g, t_\alpha(g), r_\beta(g)) = t_\alpha(g) = t_\alpha(r_\alpha(a)) = a$ (casE puis trapE), on obtient $h(cons(l,q),n) = t_\beta(cas_\beta(a, a, h'(q,n))) = t_\beta(h'(q,n)) = h(q,n)$ (casD). L'égalité est donc prouvée dans ce second cas puisque $f(q,n) = h(q,n)$.

- La règle T$\vee E$ nous permet, à partir de $N[g] \to f(cons(l,q),n) = h(cons(l,q),n)$ et $E^\alpha[g] \to f(cons(l,q),n) = h(cons(l,q),n)$ d'une part, et de $\Vdash N^\alpha[g]$ d'autre part, d'obtenir $\Vdash f(cons(l,q),n) = h(cons(l,q),n)$ ce qui termine notre preuve.

□

8.4.2 Liste des maximums

On dispose d'un programme qui, à deux listes d'entiers (a_i) et (b_i), associe la liste $(max(a_i, b_i))$ si les deux listes ont même longueur et une exception sinon. Et on souhaite obtenir un programme qui, à une liste de listes d'entiers associe la liste obtenue en calculant le maximum des premiers termes de chacune des listes, puis le maximum des deuxièmes termes ... si les listes ont toutes même longueur et une exception sinon.

Equations La fonction f est définie par : $f(\emptyset) = \emptyset$ et

$$f(cons(l,q)) = cas_\alpha(f(q), f(q), Si(videp(f(q)), l, g(l, f(q))))$$

où g est la fonction de calcul de la liste des maximums de deux listes qui rend $r_\alpha(l)$ si les deux listes n'ont pas le même nombre d'éléments (l est une liste d'entiers arbitraire et $\alpha = L$).

Exemple 8.4.2 *On note L le type des listes d'entiers et LL le type des listes de listes d'entiers. Le terme* $t = \lambda l\,(l\ V\ \emptyset)$ *où* $V = \lambda t\,\lambda q\,(T_L\ q\ \lambda x\,(Z'\ x\ t\ (G\ t\ x)))$

a dans le contexte vide le type $\forall l(LL[l] \to N^\alpha[f(l)])$ *si* G *a le type* $\forall x \forall y(L[x], L[y] \to L^\alpha[g(x,y)]$ *et si* T_L *est un opérateur de mise en mémoire sur les listes d'entiers.*

Preuve Montrons que $\vdash V : \forall t \forall q(L[t], L^\alpha[f(q)] \to L^\alpha[f(cons(t,q))]$.

- Par définition de G, on a $x : L[x], t : L[t] \vdash (G\ t\ x) : L^\alpha[g(t,x)]$ donc $x : L[x], t : L[t] \vdash (Z'\ x\ t\ (G\ t\ x)) : L^\alpha[Si(videp(x), t, g(t,x))]$ soit $t : L[t] \vdash \lambda x\ (Z'\ x\ t\ (G\ t\ x)) : \forall x(L[x] \to L^\alpha[Si(videp(x), t, g(t,x))])$.
- Posons $W = (T_L\ q\ \lambda x\ (V\ x\ t\ (G\ t\ x)))$.

  D'une part $t : L[t], q : E^\alpha[f(q)] \vdash W : E^\alpha[f(q)]$ en utilisant le lemme 8.1.3.

  D'autre part $t : L[t], q : L[f(q)] \vdash W : L^\alpha[Si(videp(f(q)), t, g(t, f(q)))]$.

  Donc en utilisant T$\vee I1$, Teq et le cas particulier de T$\vee E$ (lemme 3.4.2)

  $t : L[t], q : L^\alpha[f(q)] \vdash W : L^\alpha[cas_\alpha(f(q), f(q), Si(videp(f(q)), t, g(t, f(q))))]$

  ce qui donne à V le type prévu.

$\emptyset$ a évidemment le type $L^\alpha[\emptyset]$.

En remplaçant, dans le type LL[l], X. par $L^\alpha[f(.)]$ on obtient

$l : LL[l] \vdash l : \forall y \forall z(L[y], L^\alpha[f(z)] \to L^\alpha[f(cons(y,z))], L^\alpha[\emptyset] \to L^\alpha[f(l)]$

Il est alors facile de conclure que $l : LL[l] \vdash (l\ V\ U) : L^\alpha[f(l)]$. □

8.5 Evaluation d'une expression numérique

Lorsqu'un interpréteur évalue une expression numérique, il peut détecter des sous-expressions inévaluables comme 1/0 et dans ce cas il doit rendre une exception comme résultat de son évaluation.

Opérateurs On commence par définir le type opérateur : $O[x] \stackrel{def}{=} \forall X\ (Xa, Xm, Xd \to Xx)$, type qu'on peut considérer comme un booléen à trois valeurs. On construit ensuite facilement un « branchement » selon la valeur de l'opérateur :

$\vdash B \stackrel{def}{=} \lambda o\, \lambda x\, \lambda y \lambda z\ (o\ x\ y\ z) : \forall o, x, y, z(O[o], N^\alpha[x], N^\alpha[y], N^\alpha[z] \to N^\alpha[t(o,x,y,z)])$

où la fonction t est définie par les équations $t(a,x,y,z) = x$; $t(m,x,y,z) = y$; $t(d,x,y,z) = z$.

Preuve La preuve du typage de B est évidente : dans le type O[o] remplacer X. par $N^\alpha[t(.,x,y,z)]$.

Remarquons que si, dans le type de B, N^α était remplacé par N ou $N^{\alpha,\beta}$ (par exemple) le résultat serait encore vrai. □

Calcul On définit le type calcul ($C[x]$) comme type d'un arbre dans lequel les opérateurs sont les noeuds et les entiers les feuilles :

$$C[x] \stackrel{def}{=} \forall X(\forall o \forall g \forall d(O[o], Xg, Xd \to Xtree(g,o,d)), \forall n(N[n] \to Xfeu(n)) \to Xx)$$

Equations La fonction f d'évaluation du calcul est définie par les équations :
$f(feu(n)) = n$ et $f(tree(g,o,d)) = t(o, a(f(g), f(d)), m(f(g), f(d)), d(f(g), f(d)))$.

Les fonctions a et m sont définies comme les prolongations aux exceptions des fonctions addition et multiplication : $a(x,y) = cas_\alpha(x, x, cas_\alpha(y, y, add(x,y)))$ et $m(x,y) = cas_\alpha(x, x, cas_\alpha(y, y, mult(x,y)))$.

La fonction d prolonge aux exceptions la fonction division et distingue le cas de division par zéro :
$d(x,y) = cas_\alpha(x, x, cas_\alpha(y, y, Si(zerop(y), r_\alpha(x), div(x,y))))$.

Exemple 8.5.1 *Le terme* $t = \lambda c\,(c\ V\ \lambda x\ x)$ *où* $V = \lambda g\,\lambda o\,\lambda d\,(B\ o\ A'\ M'\ D')$, $A' = (T\ g\ (T\ d\ \lambda x\,\lambda y\,(A\ y\ x)))$, $M' = (T\ g\ (T\ d\ \lambda x\,\lambda y\,(P\ y\ x)))$ *et* $D' = (T\ g\ (T\ d\ \lambda x\,\lambda y\,(Z\ x\ \varepsilon_\alpha\langle x\rangle\ (D\ y\ x))))$ *a dans le contexte vide le type* $\forall a(C[a] \to N^\alpha[f(a)])$.

Le typage ne présente pas de difficultés puisque le terme proposé calque assez précisément les équations, il est laissé à l'initiative du lecteur.

On peut trouver un terme légèrement différent pour l'évaluation d'un calcul : $t' = \lambda c\,(c\ V'\ \lambda x\,x)$ où $V' = \lambda g\,\lambda o\,\lambda d\,(T\ g\ (T\ d\ W))$ et $W = \lambda x\,\lambda y\,(B\ o\ (A\ y\ x)(P\ y\ x)(Z\ x\ \varepsilon_\alpha\langle x\rangle\ (D\ y\ x)))$.

Son typage s'effectue en montrant qu'en modifiant légérement les équations de définition, on obtient «la même» fonction f. Voici les modifications à apporter aux équations :

$f(tree(g,o,d)) = cas_\alpha(f(g), f(g), cas_\alpha(f(d), f(d), \tau))$ où
$\tau = t(o, add(f(g), f(d)), mult(f(g), f(d)), d(f(g), f(d)))$ et
$d(x,y) = SI(zerop(y), r_\alpha(x), div(x,y))$.

Une remarque pour conclure : la valeur rendue dans le cas de division $x/0$ est $\varepsilon_\alpha\langle x\rangle$ ce qui permettrait, dans le cas d'un programme réel, d'afficher le message « Erreur division de x par 0 ».

Chapitre 9

Conclusion

9.1 Bilan

Notre projet était de réaliser un système de λ-calcul dans lequel on puisse définir des fonctions avec exceptions et prouver leur terminaison, l'avons nous mené à bien ?

9.1.1 Résultats

Nous avons construit un nouveau λ-calcul typé en ajoutant au système AF2 les constructions nécessaires pour exprimer et gérer les exceptions.

- De nouveaux λ-termes $\varepsilon_\alpha\langle u\rangle$ et $\tau_\alpha\langle u, v\rangle$ ont été créés pour représenter les exceptions et le mécanisme de leur capture (la propagation n'est pas dénotée spécifiquement au niveau des λ-termes elle est assurée par les opérateurs de mise en mémoire).

- De nouveaux termes d'individus ont été créés pour dénoter les valeurs exceptionnelles et leur capture. La simplification de l'écriture des termes d'individus (même non clos) devenait indispensable, elle est assurée par un système logique de gestion d'égalités avec hypothèses de type (dites conditionnelles).

- Le typage prend en compte les exceptions en leur attribuant un type particulier E et offre le moyen de regrouper types de données (D) et exceptions à travers un type $D \vee E$ que des règles d'introduction et d'élimination de $\vee$ permettent de manipuler.

- De nouvelles règles (incluant la β-réduction), associées, à travers l'isomorphisme de Curry / Howard, à l'élimination des coupures dans les preuves permettent de réduire les λ-termes .

Ce nouveau système a les propriétés attendues d'un λ-calcul typé. La réduction est confluente. Les termes typés sont fortement normalisables. La

conservation du type lors d'une réduction n'est pas directement garantie : la règle de typage T$\vee E$ (élimination de $\vee$) introduit en effet certaines complications dans la conservation du type pour le terme $u[x:=v]$. Mais nous avons construit une réduction uniforme - si elle réduit, dans $u[x:=v]$, un redex de v, elle réduit nécessairement toutes ses occurrences - qui vérifie, elle, la propriété de conservation du type. Cela nous permet d'obtenir une propriété de conservation faible du type : le type d'un terme est aussi celui de sa forme normale. Nous n'avons pas besoin d'autre chose pour pouvoir utiliser $Ex2$ comme système de preuve de programmes.

Nous avons obtenu une caractérisation simple des termes de type entier (ou de tout autre type de données) : ils sont équivalents aux entiers de Church. On ne peut donc pas obtenir dans $Ex2$, comme il est possible de le faire dans le $\lambda\mu$-calcul ou le λ_c-calcul , de preuves « classiques » de $N[\boldsymbol{s}^n(\mathbf{0})]$. Et nous avons la certitude que si un terme est de type entier, alors sa réduction ne pourra pas déboucher sur une exception non capturée : garantie indispensable si on veut pouvoir utiliser $Ex2$ comme environnement de programmation. Les termes de type exception d'entier sont aussi simples à caractériser : leur ensemble est isomorphe à celui des entiers de Church.

Nous avons obtenu ensuite un résultat garantissant la possibilité d'utiliser $Ex2$ pour programmer : à condition que les équations des fonctions à programmer aient un modèle (sur leur ensemble de définition), un terme du bon type ($\forall x(N[x] \to N[f(x)])$] dans le cas le plus simple) constitue bien un bon programme c'est à dire calcule bien la fonction dont il a le type. Et ce résultat est appliqué dans de nombreux exemples de programmes faisant usage des exceptions.

Pour en terminer, une traduction des formules de $Ex2$ dans un calcul (intuitionniste) des propositions du second ordre montre que la logique du système $Ex2$ n'est pas la logique classique mais la logique intuitionniste.

9.1.2 Lacunes

Sans que cela remette en cause les résultats que nous venons d'énoncer - tous ont été démontrés - certaines questions n'ont pas encore trouvé de réponse.

Nous ne savons pas si la typabilité dans $Ex2_F$ implique la typabilité dans $Ex2$ (remarque 4.2.2).

Nous ne savons pas si, dans la définition de $\mathcal{F}^\alpha$, la condition iii) est nécessaire (remarque 4.5.3).

Nous n'avons pas démontré le résultat sur les types de a et b lorsque $(a\ b)$ est typable (remarque 6.1.2).

Nous n'avons pas donné d'exemple où la non conservation du type par δ-réduction soit prouvée (remarque 5.3.1).

Toutes ces questions demanderaient donc à être étudiées, leur résolution

pouvant préciser certains propriétés du système.

9.2 Perspectives

Un travail futur, à partir des résultats énoncés dans cette thèse, peut, selon nous, s'orienter dans deux grandes directions.

9.2.1 Vérification automatique de programmes

On peut envisager une traduction de programmes écrits dans le langage Caml (ou plutôt dans une version simplifié mais conservant les exceptions - mini-Caml - de ce langage) en ensembles d'équations (respectant la syntaxe des équations de $Ex2$). On peut aussi envisager la traduction réciproque. Une fois faite la preuve de terminaison des équations (dans $Ex2$), on pourrait ainsi obtenir, à condition que la traduction respecte certaines propriétés, une preuve du programme mini-Caml.

La définition d'un sous ensemble du langage Caml susceptible d'être traduit automatiquement en équations de $Ex2$ n'est sans doute pas simple : le filtrage de Caml est trop peu contraint pour se traduire aisément, la propagation des exceptions n'est pas explicite en Caml ... Mais, à voir l'exemple du produit rapide (annexe A) où programme Caml, équations et λ-termes sont syntaxiquement très proches, l'entreprise ne paraît pas désespérée.

La question de l'automatisation du passage des équations aux λ-termes mérite aussi d'être posée, elle pourrait s'envisager comme un prolongement du programme Propre ([16]).

9.2.2 Logique classique ou logique intuitionniste ?

On désigne habituellement sous le nom d'opérateurs de contrôle un ensemble de mécanismes de contrôle qui ont été ajoutés aux langages de programmation fonctionnels pour leur apporter certaines caractéristiques des langages de programmation impératifs. Depuis Griffin ([7]), le lien entre l'opérateur $\mathcal{C}$ de Felleisen ([15]) et la logique classique est acquis et de nombreux travaux ont confirmé ce lien pour d'autres opérateurs (par exemple `call\cc` du langage Scheme).

Pourtant notre travail montre qu'un mécanisme de contrôle particulier, les exceptions, peut être intégré à un λ-calcul sans que la logique de celui ci ne devienne pour autant classique. Notre système permet, en effet, d'utiliser les exceptions, selon un mécanisme proche de celui du langage de programmation Caml, tout en restant dans le cadre de la logique intuitionniste. Dans $Ex2$, les exceptions apparaissent d'ailleurs comme un type de données bis : elles sont toujours liées à un type de données particulier et elles en constituent une sorte de calque.

Ces résultats demanderaient à être énoncés de façon plus formelle. Mais ils permettent de préciser les questions que l'on doit se poser si on veut préciser le lien entre logique classique et opérateurs de contrôle : avons nous inclus dans $Ex2$ la totalité du mécanisme des exceptions - le sens précis à donner à cette phrase est lui même à définir -, les autres mécanismes de contrôle sont-ils plus puissants, et si oui en quoi, que le mécanisme des exceptions, sont-ils tous équivalents?

Annexe A

Comparaison de $Ex2$ et Caml

Nous avons dit dans le préambule qu'une des sources de notre inspiration dans la création de $Ex2$ était le langage Caml (décrit au paragraphe 1.4). Dans quelle mesure sommes nous restés fidèles à notre « modèle » ? C'est la question que nous examinons maintenant.

Une comparaison formelle n'est pas possible, puisque nous n'avons pas défini formellement la sémantique de Caml. Nous nous limiterons donc à mettre en rapport, informellement, les constructions relatives aux exceptions dans le langage Caml d'une part et le système de λ-calcul typé $Ex2$ d'autre part en distinguant dans ce dernier deux modes de description des programmes : les équations et les λ-termes.

A.1 Déclaration

Le système $Ex2$ ne permet pas - au contraire de Caml - de déclarer des exceptions, mais on peut considérer qu'il y a pour chaque type de données une infinité d'exceptions prédéclarées. Autrement dit à la dénomination de type $\alpha = N_i$ de $Ex2$ correspond la déclaration `exception alpha of N` de Caml.

Remarquer aussi qu'en Caml on peut définir une exception constante : `exception essai` ce qui n'est pas possible en $Ex2$ où une exception est toujours liée à un type de données.

A.2 Construction

En Caml, la construction d'une valeur exceptionnelle s'effectue en deux temps : `(alpha 3)` a le type `exn` puisque `alpha` a le type `N -> exn` ; `raise (alpha 3)` est une valeur exceptionnelle, tout type peut lui être attribué.

En $Ex2$ la construction intermédiaire `(alpha 3)` n'existe pas, la valeur exceptionnelle est le λ-terme $\varepsilon_\alpha\langle\underline{3}\rangle$ qui correspond au terme d'individu $r_\alpha(\boldsymbol{s}^3(\mathbf{0}))$.

Pour comprendre les choix faits dans $Ex2$, il faut bien voir les deux problèmes que pose Caml.

Une fonction `f` définie par `let f n = if (n = 0) then (raise (alpha 3)) else n` a, en Caml, le type `N -> N` alors que son exécution peut donner lieu à une exception non capturée.

Les deux expressions `(alpha 3)` et `(alpha 4)` ont le même type `exn`, ce qui permet de donner à `raise (alpha 3)` et à `raise (alpha 4)` le type `N`. Mais, dans le type `exn` de `(alpha 3)` et `(alpha 4)`, l'information sur l'entier - 3 ou 4 - est perdue, on ne sait donc plus attribuer à `raise (alpha 3)` et `raise (alpha 4)` le bon entier. On ne peut pas étendre le typage en un typage avec termes d'individus.

Le typage de l'objet intermédiaire `(alpha 3)` posant problème, dans $Ex2$ cet objet intermédiaire problématique a été supprimé.

A.3 Propagation

La propagation des exceptions est implicite en Caml : la survenue d'une exception (quelle qu'elle soit) dans l'évaluation d'une expression interrompt le calcul. Mais, dans le cas de plusieurs exceptions, que fait Caml ? Le résultat de l'évaluation de `raise (alpha 3) * raise (alpha 4)` dépend de la façon dont le compilateur évalue l'expression `a * b`.

Dans $Ex2$, la propagation des exceptions est réalisée par des constructions utilisant les opérateurs de mise en mémoire. La propagation de la valeur exceptionnelle du terme t dans le calcul de $f(t)$ est réalisée par le λ-terme $(T\ t\ F)$, si F réalise f (exemple 8.2.2). Et, dans le cas de plusieurs valeurs exceptionnelles, il est possible de controler laquelle sera propagée ((exemple 8.2.3).

Mais, dans le cas de la propagation, termes d'individus et λ-termes divergent sensiblement. En effet la construction $cas_\alpha(t,u,v)$ permet davantage que la propagation : elle permet un traitement différent selon que t est une exception de nom α ou non, et, dans le premier cas, le traitement n'est pas réduit à la propagation, la syntaxe des termes d'individus permet d'écrire le terme $cas_\alpha(t,g(t),f(t))$. Cependant la réalisation de $cas_\alpha(t,u,v)$ par un λ-terme n'est pas directe.

Si on voulait retrouver, dans le domaine de la propagation des exceptions, l'isomorphisme qui existe ailleurs entre termes d'individus et λ-termes, il faudrait utiliser une notation plus restrictive que $cas_\alpha(t,u,v)$.

A.4 Capture

Dans $Ex2$, la capture est dénotée par le terme d'individu $t_\alpha(a,f)$ et par le λ-terme $\tau_\alpha\langle u,v\rangle$, et ici les deux constructions semblent bien isomorphes.

Dans $Ex2$, on capture nécessairement toutes les exceptions de même nom et elles seules. Si on veut capturer plusieurs sortes d'exceptions, plusieurs $\tau\langle,\rangle$ doivent être imbriqués.

Le langage Caml utilise la construction `try t with m` où `m` est un filtrage dont la syntaxe est très libre. On peut capturer une partie seulement des exceptions de nom `alpha` et une partie des exceptions de nom `beta` comme dans l'exemple suivant :

```
try t with (alpha 1) -> a
         | (alpha 2) -> b
         | (beta 1) -> c ;;
```

La possibilité, dans Caml, d'utiliser un filtrage très peu contraint n'a pas son équivalent en $Ex2$, mais cette différence n'est pas liée au traitement des exceptions, elle constitue une des différences importantes entre un langage comme Caml et un système de λ-calcul typé comme AF2.

Annexe B

La δ-réduction : un système de réécriture

On peut considérer la δ-réduction comme un « Combinatory reduction system » ou CRS ([8]), avec, par exemple, les règles de réécriture suivantes :

symboles

- symboles de fonctions, avec leur arité : `t` (0), `v` (0), `lam` (1), `app` (2), `exc` (2), `try` (3), `try-eps` (4), `comp` (2), `type` (2) ;
- metavariables : D et D' pour les types de données , K et K' pour les indices, A et B pour les types, U et V pour les expressions ;

règles

- `app(lam([x] U(x)), V) -> U(V)`
- `app(exc(A,U), V) -> exc(A, U)`
- `try(A, lam([x] U(x), V) -> lam([x] U(x)`
- `try(A, eps(B,U), V) -> try-eps(comp(A,B), B, U, V)`
- `try-eps(t, B, U, V) -> app(V, U)`
- `try-eps(f, B, U, V) -> exc(B, U)`
- `comp(type(D, K), type(D', K')) -> and( egal(D, D'), egal(K, K')).`

Quelques commentaires aideront à retrouver dans ces règles les règles de la δ-réduction.

Les notations `lam([x] U(x)`, `app(U, V)`, `exc(A,U)`, `try(A, U, V)` désignent respectivement les λ-termes $\lambda x\ u$, $(u\ v)$, $\varepsilon_\alpha\langle u\rangle$ et $\tau_\alpha\langle u, v\rangle$.

La règle `try(A, eps(A,U), V) -> app(V, U)` n'est pas permise si on veut que le système de règles soit un CRS, nous faisons donc appel à une fonction auxiliaire `try-eps` qui comparera les dénominations de type A et B pour décider si elles sont égales et selon le résultat réécrira `try(A, eps(B,U), V)`.

Une dénomination de type est un couple : (type de données , entier), et deux dénominations de type (D, K) et (D', K') sont égales si D est égal à D' et K à K'.

Les règles de réécriture des fonctions `and` et `egal` ne sont pas explicitées.

Il est aisé de montrer, mais nous ne le ferons pas dans le cadre de ce travail, que le système de règles ci-dessus définit un CRS orthogonal. Le résultat « tout CRS orthogonal est confluent» ([8]) permet alors d'obtenir la confluence de la δ-réduction.

Index

Bibliographie

[1] F. Barbanera and S. Berardi. Extracting constructive content from classical logic via control-like reductions. In *Proc. Int. Conf. on Typed Lambda Calculi and Applications*, volume LNCS 664, pages 45–59. Springer, 1993.

[2] T. Coquand and G. Huet. The calculus of constructions. In *Information and Computation*, volume 76. 1988.

[3] P. Crègut. *Machines à environnement pour la réduction symbolique et l'évaluation partielle.* PhD thesis, Université de Paris 7, 1991.

[4] P. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *Proc. of the 5th Int. Conf. on Logic Programming and Automated Reasoning*, LNCS 822, pages 31–43. Springer, 1994.

[5] P. de Groote. A simple calculus of exception handling. In *Second Int. Conf. on Typed Lambda Calculi and Applications*, LNCS, pages 201–215, 1995.

[6] M. Dezani-Ciancaglini F. Cardone and U. de'Liguoro. Combining type disciplines. *Annals of Pure and Applied Logic*, 66:197–230, 1994.

[7] T. Griffin. A formulae-as-types notion of control. In *Proc. ACM Conf. Principle of Programming Languages*, pages 47–58. ACM Press, 1990.

[8] V. van Oostrom J. W. Klop and F. van Raamsdonk. Combinatory reduction systems : introduction and survey. In *TCS 121.* 1993.

[9] G. Kreisel. Mathematical significance of consistency proofs. *J. Symb. Logic*, 23:155–182, 1958.

[10] J.-L. Krivine. *Lambda-Calcul, Types et Modèles.* Masson, 1990.

[11] J.-L. Krivine. Opérateurs de mise en mémoire et traduction de Gödel. *Archive for Mathematical Logïc*, 30:241–267, 1990.

[12] J.-L. Krivine. Classical logic, storage operators and second order λ-calculus. *Annals of Pure and Applied Logic*, 68:53–78, 1994.

[13] J.-L. Krivine and Michel Parigot. Programming with proofs. *Inf. Process. Cybern.*, EIK 26(3):149–167, 1990.

[14] D. Leivant. Reasoning about functional programs and complexity classes associated with type disciplines. In *24th Annual Symp. on Found. of Comp. Sc.*, pages 460–469, 1983.

[15] E. Kohlbecker M. Felleisen, D. P. Friedman and B. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987.

[16] P. Manoury and M. Simonot. *Des Preuves de Totalité de Fonctions comme Synthèse de Programmes.* PhD thesis, Université de Paris VII.

[17] C. R. Murthy. *Extracting Constructive Content from Classical Proofs.* PhD thesis, Cornell University, 1991.

[18] K. Nour. *Opérateurs de Mise en Mémoire en Lambda-calcul pur et typé.* PhD thesis, Université de Savoie, 1993.

[19] M. Parigot. Free deduction : an analysis of computations in classical logic. In *LNCS 592*. 1991.

[20] M. Parigot. $\lambda\mu$-calculus : an algorithmic interprétation of classical natural deduction. In *Proc. Int. Conf. on Logic Programming and Automated Reasoning*, pages 190–201. Springer, 1992.

[21] M. Parigot. Strong normalization for second order classical natural deduction. In *Logic in Computer Sciences*, pages 39–46, 1993.

[22] B. Pierce. Preliminary investigation of a calculus with intersection and union types. Internal report, Carnegie Mellon University, 1990.

[23] W. Py. *Confluence en $\lambda\mu$-calcul* . PhD thesis, Université de Savoie, 1998.

[24] M. Tofte R. Milner and R. Harper. *The Definition of Standard ML.* The MIT Press, 1990.

[25] N. J. Rehof and M. H. Sorensen. The λ_Δ-calculus. In *Theoretical Aspects of Computer Software*, pages 516–542. 1994.

[26] P. Weis and X. Leroy. *Le Langage Caml.* InterEditions, 1993.

[27] P. Weis and X. Leroy. *Manuel de Référence du Langage Caml.* InterEditions, 1993.