Les lambda-termes typés.

November 28, 2001

Abstract

Après avoir décrit les types, je décris un système de λ-termes typés construites de variables munies de types (sans contexte). Ce système est utile pour resoudre le problème de synthèse de types pour les λ-termes avec contexte. Après je décris un système de termes avec contexte et explique le lien entre deux systèmes. A la fin je considère le proibleme de synthèse de terme quand un contexte et un type sont donnés.

1 Types

Les types d'un λ-calcul appliqué sont les formules construites avec:

- Les constantes de type, appellées aussi les *atomes*, on va les designer (sauf quelques exceptions ou la notation est déjà établie) par $a, b, ...$;
- Notamment parmi les constantes il y a le constante I ("objet terminal");
- Les variables $X, Y, ...$ (variables de type);
- Les connecteurs $\to$ et $\wedge$.

On écrit souvent $A_1 \to A_2 \to ...A_n \to A$ pour $A_1 \to (A_2 \to ...(A_n \to A)...)$.

Remarque. On peut en général avoir plusieur connecteurs, par exemple, $\to$, $\wedge$ et $\vee$.

Remarque. A l'ordinaire $A \to B$ est interpreté comme l' éspace des fonctions de A à B, $A \wedge B$ comme le produit et $\vee$ comme la somme (l'union disjointe) de types A et B.

2 Une représentation de λ-termes typés utilisant des variables typées.

Dans cette représentation le contexte (voir plus loin) n'est pas nécessaire. (Il reste implicite.) Conceptuellement, cette représentation est la plus simple. Cette représentation est utile pour vérifier si un terme est bien formé et pour trouver son type.

Les termes sont les mots dans l'alphabet suivant:

- Un ensemble des variables typées (on dit aussi "identificateurs") pour chaque type A $x{:}A, y{:}A, ...$;
- les constantes de termes $k_1{:}\sigma_1, k_2{:}\sigma_2 ...$, notamment pour chaque A, B les projections $p_1{:}A \wedge B \to A$, $p_2{:}A \wedge B \to B$;
- le symbole λ;
- le constructeur de paire $<, >$;
- $(,)$ parenthèses.

Remarque. Les expressions $x{:}A$, $y{:}A$, $k{:}A$ dans les termes sont traitées comme un symbole unique, on ne peut pas separer une variable de son type. Par exemple, $x{:}A$ et $x{:}B$ sont considerées comme des variables differentes.

Notation. Pour faire la lecture plus facile, on écrit souvent les types comme les exposants x^A, y^A, k^A.....

On note Λ_A la classe des termes de type A. Si $t \in \Lambda_A$ on ecrit aussi $t{:}A$ ou t^A.

La classe

$$\Lambda = \cup\{\Lambda_A | A \ \ type\}$$

de tous les termes est définie inductivement:

- Pour chaque variable x^A, $x^A \in \Lambda_A$.
- Pour chaque constante $k_i^{A_i} \in \Lambda_{A_i}$;
- Si $t \in \Lambda_A$, $s \in \Lambda_B$ alors $< t, s > \in \Lambda_{A \wedge B}$
- Si $t \in \Lambda_{A \to B}, s \in \Lambda_A$, alors $(ts) \in \Lambda_B$;
- Si $t \in \Lambda_B$ et x^A est une variable, alors $(\lambda x^A.t) \in \Lambda_{A \to B}$.

On peut remarquer que l'utilisation des projections est compris dans le cas concernant l'application: si $t{:}A \wedge B$ alors $(p_1 t){:}A, (p_2 t){:}B$.

Les variables libres et liées.

L'ensemble des variables libres $FV(t)$ est defini inductivement (suivant le processus de construction). Les variables qui ne sont pas libres sont appellées liées (par λ correspondant).

- $FV(x^A) = \{x^A\}$ pour chaque variable;
- $FV(k_i) = \emptyset$
- $FV(ts) = FV(< t, s >) = FV(t) \cup FV(s)$;
- $FV(\lambda x^A.t) = FV(t) - x^A$.

Exemple 1 $(\lambda x^A.x^{A\to A})$ est un terme de type $A \to A \to A$. Notez que la variable $x^{A\to A}$ n'est pas la même que x^A et donc n'est pas liée par λx^A.

Exemple 2 Dans le terme $(\lambda x^A.(x^{A\to A}x^A))$ la variable x^A est liée et la variable $x^{A\to A}$ est libre.

Remarque. α-conversion. Il est possible que dans un terme bien formé le symbole λx^A a plusieurs occurrences (avec le même x^A). (On peut toujours reconstruire quel variable est liée par quel λx^A.) Il est permis de changer les noms de variables liées (on remplace x^A dans l'expression λx^A et tous les occurrences de x^A liées par ce *lambda* par y^A. Formellement on remplace $\lambda x^A.t$ par $\lambda y^A.[y^A/x^A]t$. Ici y^A ne doit pas être parmi les variables libres de t (les variables libres ne doivent pas être "capturées"). Cette transformation peut être appliquée aux sous-termes et s'appelle α-conversion.

A l'ordinaire on peut omettre certain parenthèses. On prend $ts_1...s_n$ pour $((...(ts_1)...s_n)$, $\lambda x^A.ts$ pour $(\lambda x^A.(ts))$ et $\lambda x_1^{A_1}....\lambda x_n^{A_n}.t$ pour $(\lambda x_1^{A_1}.(...(\lambda x_n^{A_n}.t)))$. Pour vérifier si une expression est un terme, on reintegre les parenthèses omises et vérifie si la définition inductive s'applique. En chaque étape on doit prendre en compte les types.

Exemple 3. L'expression $(x^{A\to B}(\lambda y^B.y^A))$ n'est pas un terme car le type de $\lambda y^B.y^A$ est $B \to A$. L'application d'un terme $x^{A\to B}$ à un terme $(\lambda y^B.y^A){:}B \to A$ n'est pas possible.

Exemple 4. L'expression $x^{(A\to B)\to B\to B}(\lambda y^A.z^B)x^B$ est un terme. On reintegre les parenthèses omises: $((x^{(A\to B)\to(B\to B)}(\lambda y^A.z^B))x^B)$.

Maintenant on vérifie que le résultat est un terme:

$$\dfrac{\dfrac{(var)}{x^{(A\to B)\to(B\to B)}} \quad \dfrac{\dfrac{(var)}{y^A} \quad \dfrac{(var)}{z^B}}{(\lambda y^A.z^B){:}A\to B}}{\dfrac{(x^{(A\to B)\to(B\to B)}(\lambda y^A.z^B)){:}B\to B \qquad \dfrac{(var)}{x^B}}{((x^{(A\to B)\to(B\to B)}(\lambda y^A.z^B))x^B){:}B}}$$

Remarque. J'ai utilisé l'écriture avec "barres" de l'arbre syntaxique. Il est difficile de faire differemment en LaTex. Les barres au sommet (avec (var)) ne sont pas necessaires, on peut voir ca comme commentaire (quel type de feuille on a). J'ajoute aussi le type au terme obtenu à chaque étape pour faciliter le contrôle de types.)

Contextes

A chaque terme défini dans cette section correspond un contexte qui consiste de toutes ses variables libres. En fait, parce que le type d'une variable est donné avec cette variable, ce contexte n'est pas necessaire dans ce genre de λ-calcul. On designe par $FV(t)$ ce contexte. ("Free Variables de t.)

Exemple 5. $FV(((x^{(A\to B)\to(B\to B)}(\lambda y^A.z^B))x^B)) = \{x^{(A\to B)\to(B\to B)}, x^B, z^B\}$.

Exemple 6 (Constantes). La constante $\mathbf{K}$ du calcul combinatoire non-typé est characterisée par $(\mathbf{K}P)Q = P$. Dans le λ-calcul typé on peut introduire les constantes $\mathbf{K}^{A\to(B\to A)}$ pour tous types A, B si on veut modeliser le comportement de $\mathbf{K}$.

Justification: les termes P, Q possèdent dans le λ-calcul typé les types A, B quelconques. Le type de $(\mathbf{K}P)Q$ doit être B d'où le type de $\mathbf{K}$ doit être $A \to (B \to A)$. (Si on prend en compte l'associativité à *droite* dans l'écriture de types, on peut écrire $A \to B \to A$.)

La constante $\mathbf{S}$, $((\mathbf{S})P)Q)R = (PR)(QR)$ est représentée par

$$\mathbf{S}^{(A\to B\to\rho)\to(A\to B)\to A\to\rho}$$

$(\forall A, B, \rho)$.

(Si on reintegre les parenthèses le type s'écrit comme $(A \to (B \to \rho)) \to ((A \to B) \to (A \to \rho))$.

Il est interessant comment on justifie le type de la constant $\mathbf{S}$ typée.

- Suppose que le type de $((\mathbf{S})P)Q)R$ est ρ.
- Alors le type de $(PR)(QR)$ doit être ρ.
- Suppose que $R{:}A$ et $QR{:}B$.

- Alors $P{:}(A \to (B \to \rho))$ et $Q{:}A \to B$ ce que nous donne le type de $\mathbf{S}$ parce que $((\mathbf{S})P)Q)R$ est un terme.

Exercice 1.(Non-corrigée.).

- Soit a un atome (constante-type). Reintegrer les parenthèses dans $a \to a \to a$, $((a \to a \to a) \to a \to a \to a$. Trouver un terme quelconque de chaque type
- Est $t = (\lambda x^{a\to a\to a}(x^{a\to a\to a}y^a y^a))$ un terme? (N'oubliez pas reintegrer les parenthèses!).
- Est $t = (\lambda x^{a\to a\to a}((x^{a\to a\to a}y^a)y^a))(\lambda x^a.x^a)$ un terme?

3 Une représentation de λ-termes typés utilisant des contextes.

C'est une représentation alternative à celle où les variables typées sont utilisées. Le type n'est plus une partie inséparable d'une variable.

Les types des variables sont introduites par déclarations qui sont assemblées dans un contexte Γ. (Si un terme sans contexte est donné, sa description n'est pas complète.)

Une déclaration c'est l'expression $x{:}A$, où x est une variable et A est un type quelconque. Un contexte legitime c'est l'ensemble des déclarations où chaque variable a au maximum une occurrence. (Mais les repetitions de types sont possibles!)

Remarque. (Pourquoi on utilise les termes avec contexte.)

- C'est plus proche à la pratique de programmation où on peut declarer les variables ou constants sans les utiliser dans un module (pour les utiliser plus tard etc.)
- C'est plus économique (on n'écrit pas le type avec chaque occurrence d'une variable).
- Il suffit de vérifier si un terme est bien formé et trouver son type un seul fois (en ce moment on peut utiliser les termes décrites dans la section précédent).

Les termes typés dans cette approche sont définis inductivement. On appelle $x, y, z, ...$ variables où identificateurs.

- Chaque constante k est un terme;
- Chaque variable est un terme;
- Si p et q sont termes, alors (pq) est un terme;
- Si r est un terme, A est un type et y est une variable, alors $(\lambda y{:}A.r)$ est un terme.

Si un contexte n'est pas donné seulement les variables liées et constantes possèdent un type. On peut vérifier seulement si la position des parenthèses est bonne et dans certain cas que les types des variables liées et peut être constants ne sont pas en accord.

Exemple. Dans $\lambda x{:}A.\lambda y{:}A.(xy)$ les types de x et de y ne sont pas en accord.

L'expression $\Gamma \vdash t{:}A$ est appellée un jugement. Elle signifie "le terme t a le type A dans le contexte Γ". Le jugement peut être vrai ou fausse.

Les vrais jugements sont définis via les règles suivantes:

$$\Gamma \vdash x{:}A \ (axiome)$$

ou $x{:}A \in \Gamma$.

$$\Gamma \vdash k{:}A \ (axiome)$$

ou k est une constante de type A.

$$\frac{\Gamma \vdash t{:}A}{\Gamma, y{:}B \vdash x{:}A} \ (wkn)$$

sous condition que $\Gamma, y{:}B$ reste un contexte legitime.

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash (ts){:}B} \ (application)$$

$$\frac{\Gamma, y : A \vdash t{:}B}{\Gamma \vdash \lambda y{:}A.t : (A \to B)}. \ (abstraction)$$

$$\frac{\Gamma \vdash t{:}A \quad \Gamma \vdash s{:}B}{\Gamma \vdash < t, s > : A \wedge B} \ (paire)$$

$$\frac{\Gamma \vdash t{:}A \wedge B}{\Gamma \vdash (p_1 t){:}A} \ \frac{\Gamma \vdash t{:}A \wedge B}{\Gamma \vdash (p_2 t){:}B} \ (projections)$$

On peut remarquer, que les dernières deux regles peuvent être exprimées via application.

3.1 Les problèmes de synthèse.

Si un jugement n'est pas complet, si Γ, t ou A n'est pas donné, on a un problème à resoudre.

Nous n'irons considerer que 2 variantes (il y en a plus):

- Quand A n'est pas donné (le synthèse de type): $\Gamma \vdash t{:}?$;
- Quand t n'est pas donné (le synthèse de terme): $\Gamma \vdash ?{:}A$.

Ces deux problèmes représentent (au sein d'un système formel, tel que le lambda-calcul typé) les problèmes qu'on rencontre quotidiennement dans le programmation (la-bas ces deux problèmes se posent moins formellement et certain effort peut être nécessaire pour les voir clairement).

- Le problème de synthèse de type corresponds à la vérification de typage dans un programme;
- Le problème de synthèse d'un terme corresponds au problème de synthèse d'un programme correspondant à une spécification. (L'expression $\Gamma \vdash ?{:}A$ représente une spécification, le terme inconnu est un programme à construire.)

3.2 Le problème de typage (ou de synthèse de types).

Etant donné une paire (Γ, t) où Γ est un contexte et t est un terme, peut-on vérifier, s'il existe un type B tel qu'on peut déduire le jugement $\Gamma \vdash t{:}B$?

Pour eviter certain difficultés, on doit utiliser d'abord l'α-conversion (changer, si necessaire, les noms des variables liés) pour qu'une variable x ne soit en même temps libre et liée et $\lambda x{:}A$ avec le même x ne soit utilisée deux fois ou plus.

Questions additionnelles:

- Est le type B unique?
- Peut-on trouver toutes les deductions (cette deuxième question est plus importante quand B n'est pas unique.)

Dans le λ-calcul appliqué qu'on considère, si le problème de typage a une solution par rapport à un contexte Γ et un terme t, le type B est unique.

a) **La synthèse de type (seulement).**

On a des algorithmes simples. Par example, on peut:

1) Vérifier, si Γ est legitime.

2) Vérifier, si toutes les variables libres de t appartiennent au Γ. (Si ce n'est pas le cas, le problème n'a pas de solution.)

3) Remplacer toutes les variables non-typées dans le terme t par les variables typées, comme en section précédent (en ajoutant les types qu'on trouve dans les declarations en Γ pour aux variables libres et dans $\lambda x{:}A$ aux variables liées). Soit t' l'expression qu'on obtient.

4) On vérifie si t' est un terme (on doit suivre le processus de construction) et trouve dans ce cas en même temps son type B. t' est un terme de type B si et seulement si t est le terme de type B en contexte Γ.

Exemple. $\Gamma = x{:}\theta, y{:}\psi$, $t = \lambda z{:}\phi.(zx)$. 1) Γ est legitime. 2)La seule variable libre de t est x, qui appartienne au Γ. 3) Le t' est

$$\lambda z^{\phi}.(z^{\phi}\ x^{\theta})$$

Maintenant, si $\phi \neq \theta \to \xi$ t' n'est pas bien formé. Si $\phi = \theta \to \xi$ pour ξ quelconque, on a

$$\dfrac{\dfrac{var}{z^{(\theta\to\xi)}} \quad \dfrac{\dfrac{var}{z^{(\theta\to\xi)}} \quad \dfrac{var}{x^{\theta}}}{(z^{(\theta\to\xi)}x^{\theta})}}{\lambda z^{(\theta\to\xi)}.(z^{(\theta\to\xi)}x^{\theta})}$$

(t' est un terme).

Exercice 2. $\Gamma = x{:}(\psi \to \psi), y{:}\psi$, $t = (\lambda x{:}(\psi \to \psi).(xy))x$. (Vérifier si t est un terme en contexte Γ et trouver son type.)

Indication. N'oubliez α-conversion.

3.3 La synthèse d'un deduction.

Si l'objectif est d'obtenir non seulement le type de t, mais aussi une déduction du jugement $\Gamma \vdash t{:}A$, on peut utiliser l'algorithme suivant.

Remarque. L'algorithme n'utilise pas la règle de *wkn* (affaiblissement). Cette règle est necessaire seulement si on veut trouver toutes les déductions.

On peut d'abord vérifier si le contexte est legitime, si le terme est bien formé et quel est son type.

Si c'est le cas, on peut determiner quel est la dernière règle de la deduction à partir de structure du terme (c'est à dire, on suit tout simplement le processus de construction).

Si le terme est t_1t_2 c'est la règle d'application; si le terme est l'abstraction, la règle est celle d'abstraction; si c'est variable ou constante, c'est un axiome.

La déduction est construite à partir de jugement $\Gamma \vdash t{:}A$ qu'on veut démontrer. (On remonte de $\Gamma \vdash t{:}A$ vers les axiomes et on reconstruit les règles.)

Exemple. Donné: $\Gamma = x{:}\theta, y{:}\theta \to \xi$, $t = \lambda z{:}\theta \to \xi.(zx)$. Deduction:

$$\dfrac{\dfrac{ax.}{x{:}\theta, y{:}\psi, z{:}(\theta \to \xi) \vdash z{:}(\theta \to \xi)} \quad \dfrac{ax.}{x{:}\theta, y{:}\psi, z{:}(\theta \to \xi) \vdash x{:}\theta}}{\dfrac{x{:}\theta, y{:}\psi, z{:}(\theta \to \xi) \vdash zx{:}\xi}{x{:}\theta, y{:}\psi \vdash \lambda z{:}(\theta \to \xi).(zx){:}(\theta \to \xi) \to \xi}}$$

Exercice 3. Trouver une déduction de

$$x{:}(A \to A) \vdash \lambda y{:}A.(x(x(xy))){:}(A \to A)$$

Exercice 4. Soit N une constante de type et $plus{:}N \to (N \to N)$ une constante de terme. Soit $\Gamma = x{:}N, y{:}(N \to \xi)$.

a) Est $t = (plus\, x)y$ un terme en contexte Γ?

b) Est $s = y((plus\, x)x)$ un terme en contexte Γ ?

Trouver le type et une déduction.

4 Le problème de synthèse d'un terme.

Problème. (Synthèse d'un terme.) Etant donné un contexte Γ et un type A, trouver un terme t tel que $\Gamma \vdash t{:}A$.

Le problème de synthèse d'un terme est beaucoup plus compliquée que le problème de synthèse d'un type.

Je vais considerer deux approches.

- La première approche utilise les déductions logiques.
- La deuxième utilise l'existence d'une forme normale.

(SUITE: 14.10.01)

4.1 Déductions logiques.

Si on prend un jugement $\Gamma \vdash t{:}A$ et éfface le terme t et toutes les variables dans les declarations $x{:}A \in \Gamma$ le résultat peut être consideré comme un proposition logique. (Selon l'isomorphisme de Curry-Howard les types peuvent être consideré comme les formules logiques.)

Si $A_1, ..., A_n, A$ sont les formules $A_1, ..., A_n \vdash A$ est compris comme l'implication $A_1 \to (A_2 \to ... \to A)...)$ ou $A_1 \wedge ... \wedge A_n \to A$. (Pour minimiser les modifications, je vais designer l'implication par $\to$.)

Dans un contexte legitime il n'y a pas de répétition des variables mais les répétition de types sont possibles. Si on note par A le resultat de suppression des variables dans les declarations de Γ, on ne peut pas plus considerer A comme un ensemble, mais comme un “multiset” (l'ensemble où chaque élément peut être present plusieur fois).

Les règles de déduction des jugements correspondent aux règles bien connues de la logique:

$$
\begin{array}{lll}
\Gamma \vdash x{:}A \ (x{:}A \in \Gamma) & \Sigma \vdash A \ (A \in \Sigma) & (axiome) \\[1ex]
\Gamma \vdash k{:}A \ (k \ const.\ de\, type\, A) & \Sigma \vdash A \ (A \ type\ d'un\ constante) & (axiome) \\[1ex]
\dfrac{\Gamma \vdash t{:}A}{\Gamma, y{:}B \vdash t{:}A} & \dfrac{\Sigma \vdash A}{\Sigma, B \vdash A} & (wkn) \\[2ex]
\dfrac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash (ts){:}B} & \dfrac{\Sigma \vdash A \to B \quad \Sigma \vdash A}{\Sigma \vdash B} & (\to -elimination) \\[2ex]
\dfrac{\Gamma, y : A \vdash t{:}B}{\Gamma \vdash \lambda y{:}A.t : (A \to B)} & \dfrac{\Sigma,\ A \vdash B}{\Sigma \vdash (A \to B)} & (\to -introduction) \\[2ex]
\dfrac{\Gamma \vdash t{:}A \quad \Gamma \vdash s{:}B}{\Gamma \vdash < t, s > : A \wedge B}\,(paire) & \dfrac{\Sigma \vdash A \quad \Sigma \vdash B}{\Sigma \vdash A \wedge B} & (\wedge - introduction) \\[2ex]
\dfrac{\Gamma \vdash t{:}A \wedge B}{\Gamma \vdash (p_1 t){:}A} \quad \dfrac{\Gamma \vdash t{:}A \wedge B}{\Gamma \vdash (p_2 t){:}B}\,(projections) & \dfrac{\Sigma \vdash A \wedge B}{\Sigma \vdash A} \quad \dfrac{\Sigma \vdash A \wedge B}{\Sigma \vdash B} & (\wedge - elim) \\[2ex]
\dfrac{\Gamma, x{:}B, y{:}B \vdash t{:}A}{\Gamma, x{:}B \vdash [x/x, y]t{:}A}(contr.) & \dfrac{\Sigma, B, B \vdash A}{\Sigma, B \vdash A} & (contr.)
\end{array}
$$

Une déduction c'est un arbre qui a les axiomes comme les feuilles et les règles comme les noeuds. (La racine c'est le proposition qu'on déduit.) A l'ordinaire on écrit les déductions en utilisant la notation avec barres.

Exemple.

$$
\dfrac{\dfrac{ax.}{(a \to a) \to c \vdash (a \to a) \to c} \quad \dfrac{\dfrac{ax.}{(a \to a) \to c, a \vdash a}}{(a \to a) \to c \vdash (a \to a)}(\to -intro)}{(a \to a) \to c \vdash c}(\to -elim)
$$

Ici $A = (a \to a), B = c, A = (a \to a) \to c$.

A partir d'un déduction logique d de $\Sigma \vdash A$ on peut construire par recurrence un terme $t(d)$ de type A dans un contexte Γ. Pour obtenir Γ de Σ on rajoute les variables (chaque formule A est remplacée par une declaration de type $x{:}A$). Pour chaque occurrence de A dans Σ on doit choisir une variable qui n'etait pas utilisée avant.

Le terme $t(d)$ est défini au choix de noms des variables près.

L'algorithme est plus facile à comprendre si on écrit les règles pour les termes (correspondantes aux règles logiques) avec contextes et types.

$$\Sigma \vdash A \qquad \Gamma_\Sigma \vdash x{:}A \ (x{:}A \in \Gamma_\Sigma) \qquad (axiome)$$

$$\Sigma \vdash A \ (A \ type\ d'un\ constante) \qquad \Gamma \vdash k{:}A \ (k\ const.\ de\, type\ A) \qquad (axiome)$$

$$\frac{\Sigma \vdash A}{\Sigma, B \vdash A} \qquad \frac{\Gamma_\Sigma \vdash t{:}A}{\Gamma_\Sigma, y{:}B \vdash x{:}A} \qquad (wkn)$$

$$\frac{\Sigma \vdash \ A \to B \quad \Sigma \vdash A}{\Sigma \vdash B} \qquad \frac{\Gamma_\Sigma \vdash t : A \to B \quad \Gamma_\Sigma \vdash s : A}{\Gamma_\Sigma \vdash (ts){:}B} \qquad (application)$$

$$\frac{\Sigma,\ A \vdash B}{\Sigma \vdash\ (A \to B)} \qquad \frac{\Gamma_\Sigma, y : A \vdash t{:}B}{\Gamma_\Sigma \vdash\ \lambda y{:}A.t :(A \to B)} \qquad (abstraction)$$

$$\frac{\Sigma \vdash A \quad \Sigma \vdash B}{\Sigma \vdash A \wedge B} \qquad \frac{\Gamma_\Sigma \vdash t{:}A \quad \Gamma_\Sigma \vdash s{:}B}{\Gamma_\Sigma \vdash < t, s > :A \wedge B}$$

$$\frac{\Sigma \vdash A \wedge B}{\Sigma \vdash A} \quad \frac{\Sigma \vdash A \wedge B}{\Sigma \vdash B} \qquad \frac{\Gamma_\Sigma \vdash t{:}A \wedge B}{\Gamma_\Sigma \vdash (p_1 t){:}A} \quad \frac{\Gamma_\Sigma \vdash t{:}A \wedge B}{\Gamma_\Sigma \vdash (p_2 t){:}B}$$

$$\frac{\Sigma, B, B \vdash A}{\Sigma, B \vdash A} \qquad \frac{\Gamma_\Sigma, x{:}B, y{:}B \vdash t{:}A}{\Gamma_\Sigma, x{:}B \vdash [x/x, y]t{:}A}$$

Exemple. (Une déduction logique et une déduction de jugement correspondent.)

a)Déduction logique.

$$\dfrac{\dfrac{ax.}{(a \to a) \to c \vdash (a \to a) \to c} \quad \dfrac{\dfrac{ax.}{(a \to a) \to c, a \vdash a}}{(a \to a) \to c \vdash (a \to a)}(\to -intro.)}{(a \to a) \to c \vdash a}(\to -elim)$$

b) Déduction de jugement de typage:

$$\dfrac{\dfrac{ax.}{x{:}(a \to a) \to c \vdash x{:}(a \to a) \to c} \quad \dfrac{\dfrac{ax.}{x{:}(a \to a) \to c, y{:}a \vdash y{:}a}}{x{:}(a \to a) \to c \vdash \lambda y{:}a.y{:}(a \to a)}(abstr.)}{x{:}(a \to a) \to c \vdash x(\lambda y{:}a.y){:}a}(appl.)$$

Exercice 5 1) Construire un terme de déduction suivante:

$$\dfrac{\dfrac{\dfrac{\dfrac{ax.}{a \to (a \to c), a \vdash a \to (a \to c)} \quad \dfrac{ax.}{a \to (a \to c), a \vdash a}}{a \to (a \to c), a \vdash a \to c}(\to -elim.) \quad \dfrac{ax.}{a \to (a \to c), a \vdash a}}{a \to (a \to c), a \vdash c}(\to -elim.)}{a \to (a \to c) \vdash a \to c}(\to -intro.)$$

Maintenant on peut resoudre le probleme de synthèse de terme si on sait comment construire une déduction (si une déduction logique est connue, on peut construire son terme).

4.2 Comment chercher une déduction logique.

S'il existe une déduction quelconque, il existe une déduction d qui a la forme suivante:

- Il n'y a pas de wkn dans d;
- Dans tous les règles de $\rightarrow$-elimintaion dans d la premice gauche est une axiome.

A la base de cette information on peut proposer l'algorithme suivant (pour rechercher une déduction de $\Sigma \vdash A$):

1. Voir, si A est "en tête" d'une formule en Σ, c'est à dire, il y a une formule $B_1 \rightarrow \ldots \rightarrow B_n \rightarrow A$ dans Σ;
2. Si c'est le cas, prendre la figure suivante comme une partie finale possible de déduction:

$$\cfrac{\cfrac{\cfrac{\cfrac{\Sigma \vdash B_1 \rightarrow \ldots \rightarrow B_n \rightarrow A \quad \cfrac{?}{\Sigma \vdash B_1}}{\Sigma \vdash B_2 \rightarrow \ldots \rightarrow B_n \rightarrow A} \quad \cfrac{?}{\Sigma \vdash B_2}}{\ldots\ldots}}{\Sigma \vdash B_n \rightarrow A} \quad \cfrac{?}{\Sigma \vdash B_n}}{\Sigma \vdash A}$$

3. Essayer d'appliquer l'algorithme à premices droites.
4. Si 2) n'est pas applicable, et $A = A_1 \rightarrow A_2$ prendre

$$\frac{\Sigma, A_1 \vdash A_2}{\Sigma \vdash A_1 \rightarrow A_2}$$

   comme une partie finale possible de déduction et appliquer l'algorithme à $\Sigma, A_1 \vdash A_2$.

On prend sans demonstration que si l'algorithme n'aboutit pas il n'y pas de deduction. Si la déduction existe on peut obtenir le terme comme décrit avant.

Exercice 6. Essayez de trouver une déduction de

$$a \to (b \to c) \vdash b \to (a \to c)$$

Les autres systèmes de lambda-calcul

Le lambda-calcul non-typé.

Je presente ici justement quelques remarques, en comparaison avec le calcul simplement typé.

Les termes sont definis comme suit:

- variables $x, y...$ sont des termes;
- si s et t sont des termes, alors $(s)t$ est un terme;
- si x est une variable et t un terme, alors $\lambda x.t$ est un terme (x est liée dans t).

Comme dans le calcul typé, l'expression $ts_1s_2...s_n$ est l'abréviation de $(...((t)s_1)...)s_n$, et $\lambda x_1...x_n.t$ est l'abréviation de $\lambda x_1.(\lambda x_2.(...(\lambda x_n.t)...))$.

Si on prend un terme simplement typé sans constantes et operation de paire et efface les types, le resultat est un terme non-typé dans le sens de cette definition.

Dans le lambda-calcul non-type les α-, β- et η-conversions sont definis comme dans le calcul typé.

$(\lambda x.t)s = [s/x]t\,(\beta)$, si x n'est pas libre dans t alors $\lambda x.(tx) = t\,(\eta)$, $\lambda x.t = \lambda x'.[x'/x]t\,(\alpha)$. Si on transforme le terme de gauche en terme de droite, c'est une réduction. Les réductions et conversions s'appliquent aux sous-termes.

Le calcul est fortement normalisable (par rapport à un système des réductions) si chaque sequence des réductions appliquée à un terme se termine. Le terme t qui n'admet pas de réductions est dit "normal" (par rapport a un système de reductions).

Le lambda-calcul simplement typé est fortement normalisable par rapport à β et η. La forme normale est unique à α-conversions près.

Le lambda-calcul non-typé n'est pas fortement normalisable, mais s'il existe pour un terme t une suite des β, η-réductions qui aboutit à une forme normale, cette forme normale est unique (à α-conversions près) et chaque suite des réductions se termine (par cette forme normale).

Exemple. Soit $t = \lambda x.(xx)$, alors tt ne se normalise pas (on obtient la suite infinie $tt \mapsto_\beta tt \mapsto_\beta$ .....

Les "numerals de Church" sont les termes $c_n = \lambda fx.f^n(x)$ (où $f^n(x)$ est $f(f...(f(x))..)$ (n applications). En particulièr, $c_0 = \lambda fx.x$ et $c_1 = \lambda fx.f(x)$.

Soit $S^+ = \lambda abc.b(abc)$, alors $S^+c_n = (\lambda abc.b(abc))(\lambda fx.f^n(x)) =_\beta$
$=_\beta \lambda bc.b((\lambda fx.f^n(x))bc) =_\beta \lambda bc.b(b^n(c)) = \lambda bc.b^{n+1}(c)$. En particulièr $S^+c_0 = c_1$.

Definition. Soit $f{:}N^p \to N$ une fonction numérique de p arguments. On dit que f est λ-definable (dans le calcul non-typé) s'il existe un λ-terme F si pour tous $n_1, ..., n_p$ $Fc_{n_1}...c_{n_p} = c_{f(n_1,...,n_p)}$.

Théorème. f est lambda-definable si et seulement si f est récursive.

Remarque. Il esiste plusieurs representations des entiers par lambda-termes, ce théorème reste vrai pour tous les représentations satisfaisants certains conditions générales.

Remarque. Cette definition peut être généralisée aux fonctions partielles.

Definition. Soit f une fonction partielle de p arguments. On dit que f est λ-definable s'il existe un λ-terme F tel que pour tous $n_1, ..., n_p$ $Fc_{n_1}...c_{n_p} = c_{f(n_1,...,n_p)}$ quand $f(n_1, .., n_p)$ est definie, sinon le terme $Fc_{n_1}...c_{n_p}$ n'a pas de forme normale.

Le théorème reste vrai.

Pour comparaison, on peut considerer les numerals dans le cacul typé. Pour chaque type A on pose

$$c_n = \lambda f{:}A \to A.\lambda x{:}A.f^n(x){:}(A \to A) \to (A \to A).$$

(Donc, la definition dépend de A maintenant.) On peut definir

$$S_A^+ = \lambda a{:}(A \to A) \to (A \to A).\lambda b{:}A \to A.\lambda c{:}A.b(abc){:}$$

$$((A \to A) \to (A \to A)) \to ((A \to A) \to (A \to A))$$

On peut definir le lambda-definabilité dans le lambda-calcul typé comme pour le calcul non-typé.

Mais la classe de fonctions qui sont definables en utilisant les lambda-termes seulement (sans, par exemple, la constante Rec de recursion et le type des entiers N) est beaucoup plus petit, loin de contenir tous les fonctions recursives.

Les autres systèmes de lambda-calcul typé.

En général on peut imaginer les dépendances suivantes entre des types et des termes.

$terme \to terme$, $type \to type$, $::$ $\to terme$ et $terme \to type$.

Par exemple, le terme ts (application) depénd de ses composantes t et s (le lambda-calcul simple typé et le calcul non-typé).

Le type $A \to B$ depénd de A et B.

Le système suivant de lambda-calcul donne des exemples de dependance (*terme/type*). (Ce système était développé pour modeliser polymorphisme, par exemple, dans les langages de famille ML).

Le système F (lambda-calcul de 2-ème ordre).

Dans ce système les contextes consistent de deux parties: une liste Γ de variables distinctes de type $X_1, ..., X_k$ et une liste Σ de déclarations de type $x_1{:}A_1, ..., x_m{:}A_m$ (variables de terme distinctes avec leur types).

Les types sont definis par $T{:} = X|(T \to S)|(\forall X.T)$ (où X denote une variable de type).

Par exemple, $\forall X.X, \forall X.(X \to X), (\forall Y.(Y \to X)) \to X$ sont des types.

Le type est legal par rapport à Γ si tous ses variables libres (non liées par $\forall$) sont dans Γ.

Les termes sont construits en utilisant l'application et deux symboles d'abstraction (pour variables de terme et de type) λ et Λ.

Règles de typage de système F:

$\Gamma, \Sigma \vdash x{:}A$ si $x{:}A \in \Sigma$ (axiome);

$$\frac{\Gamma, \Sigma, x{:}A \vdash t{:}B}{\Gamma, \Sigma \vdash (\lambda x{:}A.t){:}(A \to B)}$$

$$\frac{\Gamma, \Sigma \vdash t{:}A \to B \quad \Gamma, \Sigma \vdash s{:}A}{\Gamma, \Sigma \vdash ts{:}B}$$

$$\frac{\Gamma, X, \Sigma \vdash t{:}B}{\Gamma, \Sigma \vdash (\Lambda X.t){:}(\forall X.A)}\ (*)$$

(*) S'il n'y a pas de variable libre dans t telle que son type depend de X

$$\frac{\Gamma, \Sigma \vdash t{:}(\forall X.A) \quad \Gamma \vdash S{:}Type}{\Gamma, \Sigma \vdash (tS){:}[S/X]A}$$

On peut aussi avoir des constantes (dans ce cas on rajoute les axiomes additionnels).

Par exemple, pour chaque Γ on peut avoir $\Gamma \vdash N{:}Type$, $\Gamma \vdash R{:}Type$ et pour chaque Γ, Σ $\Gamma, \Sigma \vdash succ{:}N \to N$.

La règle d'application à un type est une représentation formelle de polymorphisme (c' est à dire, de la situation quand une opération peut être appliquée aux données de types differentes). Par exemple,

$$\frac{\Gamma, \Sigma \vdash t{:}\forall X.(X \to X) \quad \Gamma \vdash N{:}Type}{\Gamma, \Sigma \vdash (tN){:}N \to N}$$

ou bien

$$\frac{\Gamma, \Sigma \vdash t{:}\forall X.(X \to X) \quad \Gamma \vdash R{:}Type}{\Gamma, \Sigma \vdash (tR){:}R \to R}$$

Par l'application de terme t à un type on spécifie d'abord le type de données. On obtient une opération qui peut être appliquée aux données de type N, R etc.

On peut représenter des entiers dans le système F et definir des fonctions lambda-definables par rapport à ce système (ce système est beaucoup plus "forte" que le lambda-calcul simple typé).

Systèmes aux types depéndants.

On dit qu'un système de lambda-termes est un système aux types depéndants si les types dans ce système peuvent depéndre de termes.

Par exemple, on peut avoir les types $A(1), ..., A(n), ...$ où n est de type N (un entier).

Les exemples typiques (et importants) sont les types de somme et de produit depéndants.

Les règles correspondantes (pour illustration):

(produit)

$$\frac{\Delta \vdash A{:}Type \quad \Delta, x{:}A \vdash B{:}Type}{\Delta \vdash \Pi x{:}A.B{:}Type}$$

(B peut être depénd de x)

$$\frac{\Delta, x{:}A \vdash t{:}B}{\Delta \vdash \lambda x{:}A.t{:}\Pi x{:}A.B{:}Type}$$

Exemple. Voila un type-produit: $\Pi n{:}N.vector(n)$. Ce type représente le produit des espaces de vecteurs de dimensions 0, 1,..., n,....

(somme)

$$\frac{\Delta \vdash A{:}Type \quad \Delta, x{:}A \vdash B{:}Type}{\Delta \vdash \Sigma x{:}A.B{:}Type}$$

$$\frac{\Delta \vdash t{:}A \quad \Delta \vdash s{:}[t/x]B \quad \Delta, x{:}A \vdash B{:}Type}{\Delta \vdash < t, s >_{\Sigma x:A.B} {:}\Sigma x{:}A.B}$$

(paire)

$$\frac{\Delta \vdash t{:}\Sigma x{:}A.B}{\Delta \vdash p_1(t){:}A}$$

$$\frac{\Delta \vdash t{:}\Sigma x{:}A.B}{\Delta \vdash p_2(t){:}[p_1(t)/x]B}$$

(projections)

Par exemple, en utilisant la somme, on definit:

[Je vais rajouter des exemples interessantes mais vous n'avez pas besoin de ca à l'examen.]