Capítulo 2: Asistentes de Pruebas para Programadores

3. Cálculo de Construcciones

1. Cálculo λ simplemente tipado
Sintaxis

Términos

$e ::= x \mid \lambda x.e \mid (e_1\ e_2) \mid c$ $\qquad x \in \mathit{Var}$

$c \in \mathit{Const}$

Tipos

$\alpha ::= t \mid \alpha_1 \rightarrow \alpha_2$ $\qquad t \in \mathit{TConst}$

Contextos

$\Gamma ::= [\ ] \mid \Gamma, x{:}\alpha$

(o directamente $\Gamma ::= x_1{:}\alpha_1 \ldots x_n{:}\alpha_n$ $(n \geq 0)$)

Sistema de tipos

Juicios de la forma: $\Gamma \mid\!- e: \alpha$

"la expresión e tiene tipo α bajo el contexto Γ"

Reglas:

$$\frac{x{:}\alpha \in \Gamma}{\Gamma \mid\!- x{:}\alpha}\ \textbf{ctx}$$

$$\frac{\Gamma, x{:}\alpha \mid\!- e: \beta}{\Gamma \mid\!- \lambda x.e: \alpha \to \beta}\ \textbf{abs}$$

$$\frac{\Gamma \mid\!- e_1{:}\alpha \to \beta \quad \Gamma \mid\!- e_2: \alpha}{\Gamma \mid\!- (e_1\ e_2): \beta}\ \textbf{app}$$

Sistema de tipos
Ejemplos de términos tipados

- $\lambda x.x$: ?
- $\lambda x.\lambda y.\ x$: ?
- $\lambda x.\lambda y.\lambda z.((x\ y)\ z)$: ?
- $\lambda x.\lambda y.\lambda z.((x\ z)\ (z\ y))$: ?

- ? : $\alpha\to\beta\to\beta$
- ? : $(\alpha\to\beta\to\delta) \to \beta \to \alpha \to\delta$

Reducciones

- **Reducción β**

  $(\lambda x.e_1\ e_2) \to_\beta e_1[x := e_2]$ *β-redex*

- **Reducción η**

  $\lambda x.(f\ x) \to_\eta f$ *η-redex*

- **Reducción δ**

  Si c:= *d entonces c* $\to_\delta$ *d* *δ-redex (unfold)*

Reductiones - Ejemplos

- $((\lambda x.\lambda y.(x\ y)\ \lambda z.z)\ w)$

  $((\lambda x.\lambda y.(x\ y)\ \lambda z.z)\ w) \rightarrow_\beta (\lambda y.(\lambda z.z\ y)\ w)$

  $(\lambda y.(\lambda z.z\ y)\ w) \rightarrow_\beta (\lambda z.z\ w)$

  $(\lambda z.z\ w) \rightarrow_\beta w$

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

  $(\lambda x.(\lambda z.z\ x)\ y) \rightarrow_\eta (\lambda z.z\ y)$

  $(\lambda z.z\ y) \rightarrow_\beta y$

Reducciones infinitas

No siempre una cadena de reducciones β y η tiene fin. Ejemplos:

- Definimos $\boldsymbol{\Delta} = \lambda x.(x\ x)$
  - Entonces $(\Delta\ \Delta) \rightarrow_\beta (\Delta\ \Delta) \rightarrow_\beta (\Delta\ \Delta) \rightarrow_\beta \ldots\ldots$
- Definimos $\boldsymbol{\Delta_3} = \lambda x.(x\ x\ x)$
  - Entonces $(\Delta_3\ \Delta_3) \rightarrow_\beta (\Delta_3\ \Delta_3\ \Delta_3) \rightarrow_\beta \ldots\ldots$

Propiedad importante: ***tener tipo garantiza que no haya computaciones infinitas***

Dado un término e, si existe α tal que Γ |- e:α , entonces e es β-η-normalizable (toda cadena de reducciones temina en una ***forma normal***)

Cálculo λ simplemente tipado en Coq

- Constantes de tipo: nat, bool, etc.
- Las abstracciones se escriben con fun ... => ... El tipo de las variables aparece en las abstracciones:
  - fun x: nat => x : nat → nat
  - fun x: bool => x : bool → bool
  - fun (f:nat→bool)(x:nat) => f x : bool
  - fun (g:nat→nat→bool)(n:nat) => g n n : bool
  - fun (g:nat→nat→bool)(n:nat) => g n : nat → bool

2. Polimorfismo Paramétrico

- En los lenguajes funcionales (ML, Haskell) escribimos: $\lambda x.\ x : \alpha \rightarrow \alpha$

Variable de tipo!!

- En Coq, abstraemos el tipo. Los tipos de datos pertenecen a la clase Set:

fun (X:Set)(x:X) => x : forall X:Set, X → X

Abstrayendo en la clase Set se obtiene polimorfismo paramétrico

3. Tipos dependientes

- Polimorfismo paramétrico = abstraer variables de tipo (de Set).

- En forma más general: podemos abstraer variables de *cualquier tipo*.

  Tipos dependientes: un tipo puede *depender* no solo de otro tipo, sino también de un *objeto de cierto tipo.*

Tipos dependientes
Ejemplos

- Array: Set $\rightarrow$ nat $\rightarrow$ Set
  - **(Array bool 3)**
  - **zip: forall (A B:Set)(k:nat),**
    (Array A k)$\rightarrow$(Array B k)$\rightarrow$(Array (A*B) k)

- Matrix: Set $\rightarrow$nat$\rightarrow$nat$\rightarrow$Set
  - **(Matrix nat 5 4)**
  - **prod:forall n m k:nat,**
    (Matrix nat n k)$\rightarrow$(Matrix nat k m)$\rightarrow$(Matrix nat n m)

Lenguaje de programación de Coq

- Es un lambda cálculo tipado con las siguientes características:
  - es de orden superior
  - permite definir funciones paramétricas
  - tiene tipos dependientes

  CC

  - permite definir tipos inductivos

  CCI

  - permite definir tipos coinductivos

  CCI$^{\propto}$

Cálculo de Construcciones (CC)

Cálculo de Construcciones Inductivas = CC + tipos inductivos (CCI)

Cálculo de Construcciones Inductivas y Coinductivos =
CCI+tipos coinductivos (CCI$^{\propto}$)

4. Cálculo de Construcciones

Sintaxis

1. Términos

T ::= Set | Prop | Type		
|	x	variables
|	(T T)	aplicación
|	[x : T]T	abstracción
|	(x : T) T	producto
|	T → T	tipo de las funciones*

*** A → B abrevia (x:A)B cuando x no ocurre libre en B**

Ejemplos: [A:Set][x:A]x   [f: A → A → A][a:A](f a a)

Sintaxis (cont.)

2. Contextos

$$\Gamma ::= \; [\,] \;\; | \;\; \Gamma \, , \, x : T$$

3. Definiciones

$$\text{def} ::= \;\; C := T$$

Sintaxis - Alcance

La lambda abstracción y el producto son operadores de ligadura

Definición [alcance]

- en el término [x:A]B el alcance de la abstracción [x:A] es B.
- en el tipo (x:A) B el alcance del cuantificador (x:A) es B.

Las nociones de variable libre y ligada son las usuales.

Sintaxis - Sustitución

Definición [sustitución]

sean t y u dos términos y x una variable.

t[x := u] denota el término que resulta de sustituir todas las ocurrencias libres de x en t por el término u.

Otras notaciones: t[u/x], [u/x]t, t[x<-u], t{x:=u}

Asumiremos que la operación de sustitución evita la captura de variables renombrando adecuadamente las variables ligadas.

Reglas de Reducción

Definición [reducción beta y eta]

- ([x:T]f   a) $\rightarrow_\beta$ f [x := a]
- [x:T] (f x) $\rightarrow_\eta$ f

Definición

- Se define como $\Rightarrow_\beta$ y $\Rightarrow_\eta$ las relaciones correspondientes a la reescritura de un subtérmino utilizando $\rightarrow_\beta$ y $\rightarrow_\eta$. Se define $\Rightarrow_{\beta\eta}$ como la unión de ambas relaciones.
- Se define como $\Rightarrow_{\beta *}$ y $\Rightarrow_{\eta *}$ la clausura reflexiva, transitiva de $\Rightarrow_\beta$ y $\Rightarrow_\eta$ respectivamente.
- Idem para $\Rightarrow_{\beta\eta *}$

Reglas Reducción (cont.)

Definición [beta y eta conversión]

- Se definen $\equiv_{\beta\eta}$ como la clausura reflexiva, transitiva y simétrica de $\rightarrow_{\beta\eta^*}$
- Si t $\equiv_{\beta\eta}$u se dice que t y u son $\beta\eta$-convertibles

Definición [redex, forma normal]

- Un término t es un β-redex si t$\rightarrow_{\beta}$u y es un η-redex si t$\rightarrow_{\eta}$u
- Un término t está en β-forma normal si ninguno de sus terminos es un β-redex (idem para η-forma normal y $\beta\eta$-forma normal).

Juicio de tipabilidad en CC

Definición [juicio de tipado]

Los juicios t tiene tipo T en el contexto Γ (Γ |- t:T) y Γ es un contexto bien formado se definen en forma mutuamente recursiva

El conjunto S de clases de objetos se define como :

S = {Set, Prop, Type, Type1, Type2, ...}

El conjunto *R* de reglas de formación de productos es:

***R* =** $\{ <Prop,Prop,Prop>, <Set, Prop, Prop>, <Type_i, Prop, Prop>,$
$<Set, Set, Set>, <Prop, Set, Set>, <Type_i, Set, Set>,$
$<Set,Type_i,Type_i>, <Prop,Type_i,Type_i>,$
$<Type_i, Type_j, Type_{max\{i,j\}}> \mid i \in N \}$

Juicio de Tipabilidad (cont)
Reglas de Buena formación de Contextos

$$\frac{}{[\ ]\ \textbf{bien formado}}$$

$$\frac{\Gamma \vdash T{:}s \qquad\qquad x \notin \Gamma}{\Gamma,\ x{:}T\ \textbf{bien formado}}\ \text{DECLARACION DE VARIABLES}$$

Juicio de Tipabilidad (cont)
Reglas de Tipado I

$$\frac{}{[\,]\ |\!-\ \mathbf{Set:Type}}\ \mathbf{AX1}$$

$$\frac{}{[\,]\ |\!-\ \mathbf{Prop:Type}}\ \mathbf{AX2}$$

$$\frac{}{[\,]\ |\!-\ \mathbf{Tipe_i:Type}\ _{\mathbf{i+1}}}\ \mathbf{AXi}$$

$$\frac{\Gamma\ |\!-\ T{:}s_1 \qquad \Gamma,\ x{:}T\ |\!-\ U{:}s_2}{\Gamma\ |\!-\ (x{:}T)U : s_3}\ \langle S1,S2,S3\rangle \in R\ \ (\text{PROD})$$

Juicio de Tipabilidad (cont)

Reglas de Tipado II

$$\frac{\Gamma \text{ bien formado} \qquad x{:}t \in \Gamma}{\Gamma \mid\!- \ x : t}\ \text{VAR}$$

$$\frac{\Gamma \mid\!- (x{:}T)U{:}s \qquad \Gamma\,,\ x{:}T \mid\!- t{:}U}{\Gamma \mid\!- [x{:}T]t : (x{:}T)U}\ \text{ABSTR}$$

$$\frac{\Gamma \mid\!- f{:}(x{:}T)U \qquad \Gamma \mid\!- t{:}T}{\Gamma \mid\!- (f\ \ t) : U\ [x \leftarrow t]}\ \text{APP}$$

Juicio de Tipabilidad (cont)
Reglas de Tipado III

$$\frac{\Gamma \vdash t:T \qquad \Gamma \vdash U:s \qquad T \equiv_{\beta\eta} U}{\Gamma \vdash t:U}\ (\mathbf{CONV})$$

δ -Reducción

Definición [regla delta]

si C :=E es una definición incorporada al contexto Γ

t $\rightarrow_{\delta, \Gamma}$ t´ donde t´ es el resultado remplazar una ocurrencia del símbolo de constante C por E en t.

Observar que este remplazo no es una sustitución pues C es un símbolo de constante.

Las definiciones de forma normal y conversion vistas para $\beta\eta$ se definen de forma análoga para δ.

Ver: presentación de CC con δ en Cap. 4 del manual y [Paulin-Mohring96]

Coq: Tácticas vinculadas a la reducción

- Estrategias de reducción
  - cbv $flag_1$ $flag_2$
  - lazy $flag_1$ $flag_2$
    - donde $flag_i$ es el nombre de la reducción (beta o delta)
- Reducción Delta
  - unfold $id_1 \ldots id_n$
  - fold term
- Estrategias de reducción
  - compute **calcula la forma normal**
  - simpl **aplica β y luego δ a constantes transparentes**

Propiedades del Cálculo de Construcciones

Normalización

Si el juicio $\Gamma|\!-A:B$ es derivable entonces A tiene forma normal.

Confluencia

Si $\Gamma|\!-A:B$ y $A \rightarrow_{\beta\eta^*} A_1$ y $A \rightarrow_{\beta\eta^*} A_2$ entonces existe C tal que $A_1 \rightarrow_{\beta\eta^*} C$ y $A_2 \rightarrow_{\beta\eta^*} C$.

Más Propiedades del Cálculo de Construcciones

Decidibilidad del tipado

Existe un algoritmo que dado un término U y un contexto Γ:

- devuelve un término V tal que $\Gamma \vdash U : V$ es derivable, o bien
- reporta un error si no existe un término V tal que $\Gamma \vdash U : V$ pueda derivarse.

Problemas ligados al tipado

1. Chequeo de tipos:

$$\Gamma \mid\!- U : V \ ?$$

2. Inferencia de tipos:

Dados Γ y U existe V tal que $\Gamma \mid\!- U : V$?

3. Vacuidad de un tipo:

Dados Γ y V existe U tal que $\Gamma \mid\!- U : V$?

Los problemas 1 y 2 son decidibles en el cálculo de construcciones, 3 no lo es.