Especificación Formal en RSL

Binding

- Es una estructura de id. Agrupada
- bind = id o ($bind_1$, ..., $bind_n$) con n>= 2
- Se utiliza en:
  - Expresiones Let
  - Parámetros formales de funciones
  - Typings

Typing

- Tipo es una colección de valores
- Forma básica:
  - Binding : type_expr
- Forma derivada:
  - $(bind_1, ..., bind_n)$: $type\_expr_1$ x .. x $type\_expr_n$
  - $bind_1, ..., bind_n$: type_expr
- Utilización
  - Definiciones **value**
  - Expresiones Cuantificadas

Definiciones value

- Diferentes formas:
  - typing + axioms
  - definición de value explícito
    value pi : **Real** = 3.1415926535
  - definición de value implícito
    value pi : **Real** · x > 3.1
  - definición de función explícita
    value
      f : **Int** → **Int**
      f(x) ≡ x + 1
  - definición de función implícita
    value
      f : **Int** → **Int**
      f(x) as result post result > x

Funciones Totales

f : $type\_expr_1 \rightarrow type\_expr_2$

propiedad de función

- Definición Explícita

  value

  id : $type\_expr_1 \times ... \times type\_expr_n \rightarrow type\_expr_{n+1}$

  $id(id_1, ..., id_n) \equiv value\_expr$

- Definición por axiomas

  value

  id : $type\_expr_1 \times ... \times type\_expr_n \rightarrow type\_expr_{n+1}$

  axiom

  $\forall (id_1, ..., id_n) : type\_expr_1 \times ... \times type\_expr_n \cdot$
  $id(id_1, ..., id_n) \equiv value\_expr$

Funciones Parciales

f : $type\_expr_1 \xrightarrow{\sim} type\_expr_2$

Propiedades:

- ◆ no termina
- ◆ no determinística

- Definición Explícita

  $(value\_expr_2 \equiv \textbf{true}) \Rightarrow id(id_1, ..., id_n) \equiv value\_expr_1$

  Equivalente

  value

  id: $type\_expr_1 \times ... \times type\_expr_n \xrightarrow{\sim} type\_expr_{n+1}$

  $id(id_1, ..., id_n) \equiv value\_expr_1$ **pre** $value\_expr_2$

Expresiones de Función

- λ binding : type_expr · value_expr
  con type_expr $\xrightarrow{\sim}$ T
- Caso especial λ () · value_expr con **Unit** $\xrightarrow{\sim}$ T
- Operadores: = , ≠, °
- Funciones de Orden Superior
  value
  com : (**Real** →~ **Real**) → **Real** →~ **Real**
  com(f) ≡ λ i : **Int** · f(f(i))

com(λ i : **Real** · 2.0 * i) = λ i : **Real** · 2.0 * 2.0 * i
com(λ i : **Real** · 2.0 * i)(3.0) = 12.0
axiom f : (**Real** →~ **Real**), i:**Real** · com(f)(i) ≡ f(f(i)
axiom f : (**Real** →~ **Real**) · com(f) ≡ f º f
Ej. axiomas de PEANO

Currying - Uncurring

Currying

value
id:type_expr$_1$ → ... → type_expr$_n$ → type_expr
id(id$_1$)... (id$_1$) ≡ value_expr

value
com : (**Real** →~ **Real**) → **Real** →~ **Real**
com(f) ≡ λ i : **Int** · f(f(i))

Uncurrying

value
com2 : (**Real** →~ **Real**) x **Real** →~ **Real**
com2(f, i) ≡ f(f(i))

Aplicación:
com(f)(3.0)   com2(f, 3.0)

Definición Algebraica de Funciones

- Estándar

  $id(id_1, ..., id_n) \equiv value\_expr$
- Algebraico

  $id(value\_expr_1, ..., value\_expr_n) \equiv$ $value\_expr$
- Ejemplos
  - $f(f(x)) \equiv f(x)$ /* Idempotencia */
  - $f(g(x)) \equiv x$ /* g inversa de f */
  - Número Natural (suma, producto)
  - Diccionario

Set

- Set finitos e infinitos
- Expresiones de tipo Set

  type_expr**-set** type_expr**-infset**
- Operadores Set
- Expresiones de valores Set
- Ejemplos
  - Definición de Set
  - Administrador de Recursos
  - Diccionario - Refinamiento - Verificación
  - Parcela - Colindantes

List

- Expresiones de Tipo
- Expresiones de Valores
  - Extensión
  - Comprensión
- Indexación
- Operadores
- Ejemplos
  - Definición de Lista
  - Lista ordenada
  - Diccionario Ordenado

Conjunto, Lista y Producto

	Nro Elem	Tipo Elem	Dupl.	Orden
T-set	Var.	=	N	N
T-infset	Var.	=	N	N
T*	Var.	=	S	S
T^{ω}	Var.	=	S	S
$T_1 x ... x T_n$	Fijo	Var.	S	S

Mapas

- Expresiones de Tipo
- Expresiones de Valores
  - Extensión
  - Comprensión
- Aplicación
- Operadores
- Ejemplos
  - Definición de Mapa
  - Diccionario
  - Relación de Equivalencia
  - Parcelas - Historia

Subtipos

- {| binding : type_expr · logical_value_expr|}
- Equiv. con axiomas (valores y funciones)
- Relación transitiva
- Tipos Maximales
  - **Bool**, **Int**, **Real**, **Char**, **Unit**
  - Sorts
  - Constructores de tipo con tipos maximales
    - x, **-infset**, $^{\omega}$, $\to$~, $\to_m$
  - Id. de Tipo que son Abreviaturas de Tipos maximales
- No son maximales: **-set**, *, $\to$
- El chequeo de tipos sólo chequea tipos maximales

Tipos

- Abreviatura : **type** id = type_expr
- Sort : **type** id
- Variant **type** id == var_1 | ... | var_n
- Record

  type id ::

  d_1 : $type\_expr_1$

  ...

  d_n : $type\_expr_n$
- Union : **type** id = id_1 | ... | id_n
- Cualquier orden es permitido en **type**

Tipos

- Def_tipo_Abs = Tn
- Def_tipo = Tn '=' TE
- TE = Bool | Int | Nat | Char

  | TE x ... x TE

  | TE**-set** | TE**-infset**

  | TE* | TE^{ω}

  | TE $\rightarrow_m$ TE | TE $\rightarrow$~ TE | TE $\rightarrow$ TE
- Def_tipo_Var = Tn '==' id(id:TE, ..., id:TE)

Definición de Unión y Registro

type
Documento = Libro | Articulo,
Libro ::
titulo : Titulo
autor : Autor
editor : Editor,
Articulo ::
titulo : Titulo
autor : Autor
revista : Revista,
value
autor : Documento $\rightarrow$ Autor
autor(doc) $\equiv$ **case** doc **of**
mk_Libro(t, a, p) $\rightarrow$ a,
mk_Articulo(t, a, r) $\rightarrow$ a **end**

Definición de Registro

type
id ::
d_1 : $type\_expr_1$
...
d_n : $type\_expr_n$
values
mk_id : $type\_expr_1$ x .. x $type\_expr_n$ $\rightarrow$ id
/* mk_id(v_1, ..., v_n) donde v_i : $type\_expr_i$ */
d_1 : id $\rightarrow$ $type\_expr_1$
...
d_n : id $\rightarrow$ $type\_expr_n$

Registro versus Producto

type id :: d_1 : $type\_expr_1$ ... d_n : $type\_expr_n$

type id = $type\_expr_1$ x ... x $type\_expr_n$

- Se utiliza registro cuando la función destructora es útil
- Comparación

type A = Int, B = Int value a : A, b : B

axiom a = b /* Correcto */

type A, B value a : A, b : B

axiom a = b /* Incorrecto */

type A :: a1: Int, B :: b1: Int value a : A, b : B

axiom a = b /* Incorrecto */

mk_A(2) = mk_B(2) /* Incorrecto */

Definición de Unión

type id = id_1 | ... | id_n

value

$id\_from\_id_1 : id_1 \rightarrow id$

...

$id\_from\_id_n : id_n \rightarrow id$

$id\_to\_id_1 : id \xrightarrow{\sim} id_1$

...

$id\_to\_id_n : id \xrightarrow{\sim} id_n$

Definición de Variante

THOJA = **class**
type THoja == A4 | Carta
value
cm : THoja $\rightarrow$ Nat x Nat
cm(th) $\equiv$ **case** (th) **of**
A4 $\rightarrow$ (210, 297)
Carta $\rightarrow$ (216, 279)
end
end

Equivalente a

type THoja **value** A4, Carta : THoja
axiom
[Disjuntos] A4 $\neq$ Carta
[Inducción] $\forall$ p : THoja $\rightarrow$ Bool ·
p(A4) $\wedge$ p(Carta) $\Rightarrow$ $\forall$ h : THoja · p(c)

Definición de Variante (II)

- Ejemplo de List
  - con variante
  - equivalente usando axiomas sin variante
  - completar con apply_to_first, apply_all
  - Árbol Ordenado - Árbol Balanceado
- Uso del Wildcard
  - No es válido el Axioma de Inducción
- Registros y Uniones son Variantes

type Libro :: autor : Autor titulo : Titulo
type Libro == mk_Libro(autor:Autor, titulo :Titulo
type Documento = Libro | Articulo
type Doc ==
Doc_from_Libro(Doc_to_Libro : Libro) |
Doc_from_Articulo(Doc_to_Articulo : Articulo)

Subespecificación

Ejemplos de subespecificación

value x : **Int**

value x : **Int** · x ≠ 0

value x : **Int axiom** x ≠ 0

value f : **Int** → **Int axiom** ∀x : **Int** · f(x) > x

Ejemplos de No subespecificación

value x : **Int** = 0

value x : **Int axiom** x = 0

value f : **Int** → **Int axiom** ∀x : **Int** · f(x) ≡ x + 1

No determinismo

0 |˙| 1 |˙| 2

let x : Nat · x < 3 **in** x **end**

(0 |˙| 1 |˙| 2 ≡ 0 |˙| 1 |˙| 2) ≡ **true**

(0 |˙| 1 |˙| 2 = 0 |˙| 1 |˙| 2) ≡ **true** |˙| **false**

(0 |˙| 1 |˙| 2) - (0 |˙| 1 |˙| 2) ≡

(0-0) |˙| (0-1) |˙| (0-2) |˙| ... ≡ -2 |˙| -1 |˙| 0 |˙| ...

value choose : **Int-set** →~ **Int**

axiom ∀ x : **Int-set** · choose(s) ≡

let i : **Int** · i ∈ s **in** i **end pre** s ≠ {}

axiom ∀ x : **Int-set** · choose(s)

as i **post** i ∈ s **pre** s ≠ {}

Los predicados deben ser Convergentes

- Convergente = determinístico y definido
- Axiomas no determinísticos

axiom let b : **Bool in** b **end** ≡ **false**

- Los predicados de expresiones

∃ x : **Char** · x = **let** y : **Char in** y **end** ≡ **false**

- Comprensión:

{x | x : **Char** · x = **let** y : **Char in** y **end**} ≡ {}

Expresión Case

case value_expr **of**

pattern$_1$ → value_expr$_1$

...

pattern$_n$ → value_expr$_n$

end

- Puede dar **swap** si no existe un pattern matching existoso, o

  ser no determinísticos
- Ejemplo
  - Invertir una lista.

Patrones

- Patrones con Literales y comodines
  - Ej. de Fib
- Patrones de Listas
  - Calcular el perímetro de una figura
- Patrones con Producto
  - Tabla de verdad del y, o, ...
- Patrones de Nombres
  - Provincias - Regiones
- Patrones de Registro
  - Derivada.
- Patrones de Igualdad
  - Lista de las precipitaciones provinciales

Expresiones Let Explícitas

let binding = value_expr$_1$ **in** value_expr$_2$ **end**

let record_pattern = value_expr$_1$ **in** value_expr$_2$ **end**

let list_pattern = value_expr$_1$ **in** value_expr$_2$ **end**

Equivalencias

let (id1, id2) = value_expr$_1$ **in** value_expr$_2$ **end**

≡ **case** value_expr$_1$ **of** (id1, id2) →

let record_pattern = value_expr$_1$ ...

let list_pattern = value_expr$_1$ ...

let id = value_expr$_1$ ...

No Equiv a **case** value_expr$_1$ **of** id →

Expresiones Let Implícitas

Formas:
- **let** typing **in** value_expr **end**
- **let** binding : type_expr · value_expr$_1$ **in** value_expr$_2$ **end**

value elegir : E-**set** →~ E
elegir(s) ≡ **let** e:E · e ∈ s **in** e **end** **pre** s ≠ {}
elegir({e1, e2, e3}) ≡ e1 |-| e2 |-| e3
value elegir2 : E-**set** →~ E
elegir2(s) as e post e ∈ s **pre** s ≠ {}
(elegir2({e1, e2, e3}) ≡ {e1}) v
(elegir2({e1, e2, e3}) ≡ {e2}) v
(elegir2({e1, e2, e3}) ≡ {e3})

Expresiones Let Anidadas

let let_def$_1$ ... let_def$_n$ **in** value_expr **end**
más corto que
let let_def$_1$ **in**
    ...
    let let_def$_n$ **in** value_expr
    end
    ...
end

Sobrecarga

- Si hay varias definiciones del mismo id u operador pero con distintos tipos maximales.
- Ejemplo:
  - Máximo de dos números, de una lista y de un conjunto
- Sobrecarga de operadores definidos por el Usuario.
  - **elems** y $\in$ para árboles
- Operadores infijos pueden usarse como prefijos

  (infix_op) (value_expr$_1$, value_expr$_2$) $\equiv$
  value_expr$_1$ infix_op value_expr$_2$

Ambiguedad

value
empty : **Int-set**,
empty : **Int***,
is_empty : **Int-set** $\rightarrow$ **Bool**
is_empty : **Int*** $\rightarrow$ **Bool**
axiom
is_empty(empty)
Solución
axiom
is_empty(empty : **Int-set**),
is_empty(empty : **Int***)

Especificación Imperativa

variable

variable_def$_1$

...

variable_def$_n$

Def. Simple: id : type_expr := value_expr

Def. Múltiple: id$_1$, ..., id$_n$: type_expr

CONTADOR = **class**

variable contador : **Nat** := 0

value

incr : **Unit** $\rightarrow$ **write** contador **Nat**

incr() $\equiv$ contador := contador +1; contador

end

Especificación Imperativa

- Funciones con Acceso a variables

type_expr$_1$ $\rightarrow\sim$ acc_desc$_1$ ... acc_desc$_n$ type_expr$_2$

  - acc_desc$_i$
    - **read** id$_1$ ... id$_n$
    - **write** id$_1$ ... id$_n$
- Expresiones Imperativas
  - Asignación, secuencia (:=), expresiones de repetición (for, while, until,...), expr. If, ...
  - Efecto: Cambio de estado del contenido particular de variables.
- Expresiones puras : no tienen acceso a vbles.
- Expresiones de sólo lectura: no escribe a vbles.
- Orden de Evaluación: de izq. a derecha.

  <x:= 1; x, x:= 2; x> $\equiv$ x := 2; <1,2>

  x + (x:= x + 1; x) $\equiv$ x := x + 1; 2 * x - 1

Cuantificación sobre Estados

- Un axioma es verdadero si es verdadero en cualquier estado posible que satisface las definiciones de variables.
- Combinador "siempre"
  value_expr
- El valor de la expresión "siempre" es true si y sólo si para todos los estados que satisfacen las definiciones de variables, la expresión value_expr evalúa determinísticamente a true.
- **axiom** value_expr es equivalente a
- **axiom** value_expr

Concurrencia

- Composición
  - Secuencial
    value_expr1 ; value_expr2
  - Concurrente
    value_expr1 || value_expr2
- Es de tipo Unit
- Value_expri (1<= i <=2) es de tipo Unit
- Se recomienda que sean de asignación disjunta
- Razones para la concurrencia
  - Conceptual
  - Eficiencia

Definición de Canales

channel

channel_def$_1$ ...
channel_def$_n$

- Simple: id:type_expr
- Multiple: id$_1$, ..., id$_n$: type_expr
- Expresiones de Comunicación
  - Input expr : id?
  - Output expr: id ! value_expr
- Ejemplos

channel c:Int, variable x:Int ... x:= c? || c!5 ...

- La comunicación puede sincronizar con x:=5 o dar otro resultado si c es utilizada en otro lugar

(x:=c? || c!5) || c!7

- Puede dar x:=7; c!5 or ...