Short Records and Unions

Contents

Short record and union type definitions are short-hands for variant type definitions.

Overview of RSL Type Definitions

- abbreviation

  type id = type_expr

- sort

  type id

- variant

  type id == variant$_1$ | ... | variant$_n$

  where n $\geq$ 1

- short record

  type id ::
  d_1 : type_expr$_1$
  ⋮
  d_n : type_expr$_n$

  where n $\geq$ 1

- union

  type id = id$_1$ | ... | id$_n$

  where n $\geq$ 2

Type Declarations

type
type_definition$_1$
⋮
type_definition$_n$

Any order of definitions is allowed.

Cyclic abbreviation definitions is not allowed, e.g.

type
A = B, B = A-**set**

Name versus Structural Equivalence

Abbreviation definitions: structural equivalence.
Other type definitions: name equivalence.

type A = **Int**, B = **Int**
value a : A, b : B
axiom
/∗ the following axiom is type correct ∗/
a = b

type A , B
value a : A, b : B
axiom
/∗ the following axiom is not type correct ∗/
a = b

type A :: **Int**, B :: **Int**
value a : A, b : B
axiom
/∗ the following axioms are not type correct ∗/
a = b,
mk_A(2) = mk_B(2)

Short Record Definitions

Example

type
  Book ::
    title : Title
    author : Author
    publisher : Publisher

is short for

type
  Book ==
    mk_Book(
      title : Title,
      author : Author,
      publisher : Publisher)

Short Record Definitions

Generally

type
  id ::
    destr_id_1 : $\text{type\_expr}_1 \leftrightarrow \text{recon\_id}_1$
    :
    destr_id_n : $\text{type\_expr}_n \leftrightarrow \text{recon\_id}_n$

(where $n \geq 1$)

is short for

type
  id ==
    mk_id(
      destr_id_1 : $\text{type\_expr}_1 \leftrightarrow \text{recon\_id}_1$,
      :
      destr_id_n : $\text{type\_expr}_n \leftrightarrow \text{recon\_id}_n$)

Values of type id:

  $\text{mk\_id}(v_1, ..., v_n)$, where v_i : type_expr_i

Short Records versus Products

1. **type** id :: destr_id_1 : type_expr_1 ... destr_id_n : type_expr_n
2. **type** id = $\text{type\_expr}_1 \times ... \times \text{type\_expr}_n$

Construction of values:

1. $\text{mk\_id}(v_1, ..., v_n)$, where v_i : type_expr_i
2. $(v_1, ..., v_n)$, where v_i : type_expr_i

Destruction of values (v : id):

1. $\text{destr\_id}_i(v)$
2. **let** $(v_1, ..., v_n)$ = v **in** ... **end**

Type equivalence:

1. id and $\text{type\_expr}_1 \times ... \times \text{type\_expr}_n$ are distinct
2. id equivalent to $\text{type\_expr}_1 \times ... \times \text{type\_expr}_n$

Union definitions

Example

type Document = Book | Article

is short for

type
Document ==
  Document_from_Book(Document_to_Book : Book) |
  Document_from_Article(Document_to_Article : Article)

Union Definitions

Generally

type id = id_1 | ... | id_n, $n \geq 2$

is short for

type id ==
  $id\_from\_id_1(id\_to\_id_1 : id_1)$ |
  ... |
  $id\_from\_id_n(id\_to\_id_n : id_n)$

Values of type id:

$id\_from\_id_i(v_i)$, where $v_i : id_i$

Context Condition:

The maximal types of id_1, ..., id_n must be distinguishable

Omission of Union Constructors

Is allowed in patterns and expressions
iff their insertion is unique

Omission of Union Constructors

Example

```
type
  Document = Book | Article,
  Book ::
    title : Title
    author : Author
    publisher : Publisher,
  Article ::
    title : Title
    author : Author
    journal : Journal,
  Title, Author, Journal, Publisher

value

  f : Document → ...
  f(doc) ≡
    case doc of
                    mk_Book(t, a, p) → ...,
                    mk_Article(t, a, j) → ...
    end


  ... f(                mk_Book(t,a,j) ) ...
```


Omission of Constructors

Example

```
        A
    B       C
D       E       F

type
  A = B | C,  B = D | E,  C = E | F,
  D :: ...,  E :: ..., F :: ...

value
  f : A → T
  f(a) ≡
    case a of
        mk_D(x) → ...,
        mk_E(x) → ...,
        mk_F(x) → ...
    end
```

is not allowed. There are two possible insertions around mk_E(x):

```
A_from_B(B_from_E(mk_E(x)))
A_from_C(C_from_E(mk_E(x)))
```

Wildcards in Union Definitions

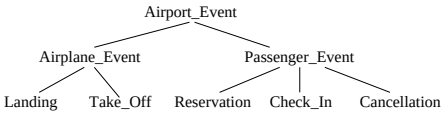
type Document = Book | Article | _

type id = id_1 | ... | id_n | _

short for:

type id ==
  $id\_from\_id_1(id\_to\_id_1 : id_1)$ |
  ... |
  $id\_from\_id_n(id\_to\_id_n : id_n)$ |
  _

Example: Airport



The book shows several ways of modelling the data of an airport system.

Since the airport concepts are hierarchical (in 3 levels), it is natural to use union types or variant types.

The choice of data model for Airport_Events has an impact on the complexity of functions (like flight_identification) operating on Airport_Events.

The conclusion is that it is shorter to use unions than variants when there are more than two levels.