Algebraic Data Types: ED($L^{\times,+}$)

Yu Zhang

http://staff.ustc.edu.cn/~yuzhang/tpl/
yuzhang@ustc.edu.cn

References

- PFPL
  - Chapters:
    - 10 Product Types, 11 Sum Types
      (Void & Unit, Boolean, Enumerate, Options)
- TAPL
- The algebra (and calculus!) of algebraic data types

Product Types

- Nullary and Binary products
  - Abstract syntax

Typ	τ	::=	`unit`	`unit`	nullary product 空积
			`prod`$(\tau_1;\tau_2)$	$\tau_1 \times \tau_2$	binary product 二元积
Exp	e	::=	`triv`	$\langle\rangle$	null tuple 空元组 (introduction)
			`pair`$(e_1;e_2)$	$\langle e_1, e_2\rangle$	ordered pair 有序对 (introduction)
			`pr[l]`(e)	$e \cdot \mathtt{l}$	left projection 左投影 (elimination)
			`pr[r]`(e)	$e \cdot \mathtt{r}$	right projection 右投影 (elimination)

  - Examples

$$\langle\rangle : \mathtt{unit}$$
$$\langle \mathtt{z}, \mathtt{z}\rangle : \mathtt{nat} \times \mathtt{nat}$$
$$\langle \mathtt{z}, (\mathtt{s}(\mathtt{z}), \mathtt{s}(\mathtt{z}))\rangle : \mathtt{nat} \times (\mathtt{nat} \times \mathtt{nat})$$
$$\langle \lambda(x:\mathtt{nat})\, x, \lambda(x:\mathtt{nat}\to\mathtt{nat})\, x\rangle : (\mathtt{nat}\to\mathtt{nat}) \times ((\mathtt{nat}\to\mathtt{nat}) \to \mathtt{nat}\to\mathtt{nat})$$

Examples

- $\lambda(x{:}\,\mathtt{nat}\times\mathtt{nat}).\, x\cdot\mathtt{l} + x\cdot\mathtt{r}$
  - 函数：对pair中的两个元素求和
- $\lambda(x{:}\,(\mathtt{nat}\to\mathtt{nat})\times\mathtt{nat}).\, x\cdot\mathtt{l}\ \ x\cdot\mathtt{r}$
  - 函数：接收由一个函数和一个自然数组成的pair，以该自然数为参数调用这个函数
- $(1,(2,(3,4))) : \mathtt{nat}\times(\mathtt{nat}\times(\mathtt{nat}\times\mathtt{nat}))$
  - pairs产生**4**元组

Nullary and Binary Products

- Statics

$$\frac{}{\Gamma \vdash \langle\rangle : \mathtt{unit}} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2\rangle : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e\cdot\mathtt{l} : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e\cdot\mathtt{r} : \tau_2}$$

Typ	τ	::=	`unit`	`unit`	nullary product
			`prod`$(\tau_1;\tau_2)$	$\tau_1 \times \tau_2$	binary product
Exp	e	::=	`triv`	$\langle\rangle$	null tuple
			`pair`$(e_1;e_2)$	$\langle e_1, e_2\rangle$	ordered pair
			`pr[l]`(e)	$e\cdot\mathtt{l}$	left projection
			`pr[r]`(e)	$e\cdot\mathtt{r}$	right projection

- Dynamics

Values introduction:

$$\frac{}{\langle\rangle\ \mathtt{val}} \qquad \frac{[e_1\ \mathtt{val}] \quad [e_2\ \mathtt{val}]}{\langle e_1, e_2\rangle\ \mathtt{val}}$$

Search transitions (omitted for lazy dynamics):

$$\left[\frac{e_1 \longmapsto e_1'}{\langle e_1, e_2\rangle \longmapsto \langle e_1', e_2\rangle}\right] \qquad \left[\frac{e_1\ \mathtt{val} \quad e_2 \longmapsto e_2'}{\langle e_1, e_2\rangle \longmapsto \langle e_1, e_2'\rangle}\right]$$

Search transitions Elimination:

$$\frac{e \longmapsto e'}{e\cdot\mathtt{l} \longmapsto e'\cdot\mathtt{l}} \qquad \frac{e \longmapsto e'}{e\cdot\mathtt{r} \longmapsto e'\cdot\mathtt{r}}$$

Instruction transitions Elimination:

$$\frac{[e_1\ \mathtt{val}] \quad [e_2\ \mathtt{val}]}{\langle e_1, e_2\rangle\cdot\mathtt{l} \longmapsto e_1} \qquad \frac{[e_1\ \mathtt{val}] \quad [e_2\ \mathtt{val}]}{\langle e_1, e_2\rangle\cdot\mathtt{r} \longmapsto e_2}$$

Finite Products

- Syntax

Typ	τ	::=	`prod`$([i \hookrightarrow \tau_i]_{i\in I})$	$\langle\tau_i\rangle_{i\in I}$	product
Exp	e	::=	`tpl`$([i \hookrightarrow e_i]_{i\in I})$	$\langle e_i\rangle_{i\in I}$	tuple
			`pr`$[i](e)$	$e\cdot i$	projection

- Statics

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash \langle i_1 \hookrightarrow e_1, \dots, i_n \hookrightarrow e_n\rangle : \langle i_1 \hookrightarrow \tau_1, \dots, i_n \hookrightarrow \tau_n\rangle} \quad (10.3a)$$

$$\frac{\Gamma \vdash e : \langle i_1 \hookrightarrow \tau_1, \dots, i_n \hookrightarrow \tau_n\rangle \quad (1 \le k \le n)}{\Gamma \vdash e\cdot i_k : \tau_k} \quad (10.3b)$$

- Dynamics

$$\frac{[e_1\ \mathtt{val} \quad \dots \quad e_n\ \mathtt{val}]}{\langle i_1 \hookrightarrow e_1, \dots, i_n \hookrightarrow e_n\rangle\ \mathtt{val}} \quad (10.4a)$$

$$\left[\left\{\begin{array}{ccccccc} e_1\ \mathtt{val} & \dots & e_{j-1}\ \mathtt{val} & e_1' = e_1 & \dots & e_{j-1}' = e_{j-1} \\ & & e_j \longmapsto e_j' & e_{j+1}' = e_{j+1} & \dots & e_n' = e_n \end{array}\right\}\right] \quad (10.4b)$$
$$\frac{}{\langle i_1 \hookrightarrow e_1, \dots, i_n \hookrightarrow e_n\rangle \longmapsto \langle i_1 \hookrightarrow e_1', \dots, i_n \hookrightarrow e_n'\rangle}$$

$$\frac{e \longmapsto e'}{e\cdot i \longmapsto e'\cdot i} \quad (10.4c)$$

$$\frac{[\langle i_1 \hookrightarrow e_1, \dots, i_n \hookrightarrow e_n\rangle\ \mathtt{val}]}{\langle i_1 \hookrightarrow e_1, \dots, i_n \hookrightarrow e_n\rangle\cdot i_k \longmapsto e_k} \quad (10.4d)$$

Nullary and Binary Sum

- Syntax

Typ	τ ::=	void	void	nullary sum
		$\mathtt{sum}(\tau_1;\tau_2)$	$\tau_1+\tau_2$	binary sum
Exp	e ::=	$\mathtt{abort}[\tau](e)$	$\mathtt{abort}(e)$	abort
introduction		$\mathtt{in}[\mathtt{l}][\tau_1;\tau_2](e)$	$\mathtt{l}\cdot e$	left injection
elimination		$\mathtt{in}[\mathtt{r}][\tau_1;\tau_2](e)$	$\mathtt{r}\cdot e$	right injection
		$\mathtt{case}(e;x_1.e_1;x_2.e_2)$	$\mathtt{case}\,e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1 \mid \mathtt{r}\cdot x_2\hookrightarrow e_2\}$	case analysis

Elimination of void

- Examples

$$\mathtt{l}\cdot\mathtt{z} : \mathtt{nat}+\tau$$
$$\mathtt{r}\cdot\mathtt{s}(\mathtt{z}) : \tau+\mathtt{nat}$$
$$\mathtt{l}\cdot\mathtt{r}\cdot\lambda\,(x:\mathtt{nat})\,x : (\tau_1+(\mathtt{nat}\to\mathtt{nat}))+\tau_2$$
$$\mathtt{case}\ \mathtt{l}\{\mathtt{nat}\to\mathtt{nat};\mathtt{nat}\}\cdot\lambda\,(x:\mathtt{nat})\,x\ \{\mathtt{l}\cdot x_1\hookrightarrow x_1(\mathtt{z}) \mid \mathtt{r}\cdot x_2\hookrightarrow x_2\}$$
$$\mathtt{bool}\triangleq\mathtt{unit}+\mathtt{unit}$$
$$\mathtt{true}\triangleq\mathtt{l}\cdot\langle\rangle$$
$$\mathtt{false}\triangleq\mathtt{r}\cdot\langle\rangle$$

Nullary and Binary Sum

- Statics

$$\frac{\Gamma\vdash e:\mathtt{void}}{\Gamma\vdash\mathtt{abort}(e):\tau}$$
$$\frac{\Gamma\vdash e:\tau_1}{\Gamma\vdash\mathtt{l}\cdot e:\tau_1+\tau_2}$$
$$\frac{\Gamma\vdash e:\tau_2}{\Gamma\vdash\mathtt{r}\cdot e:\tau_1+\tau_2}$$
$$\frac{\Gamma\vdash e:\tau_1+\tau_2\quad\Gamma,x_1:\tau_1\vdash e_1:\tau\quad\Gamma,x_2:\tau_2\vdash e_2:\tau}{\Gamma\vdash\mathtt{case}\,e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\mid\mathtt{r}\cdot x_2\hookrightarrow e_2\}:\tau}$$

Typ	τ ::=	void	void	
		$\mathtt{sum}(\tau_1;\tau_2)$	$\tau_1+\tau_2$	
Exp	e ::=	$\mathtt{abort}[\tau](e)$	$\mathtt{abort}(e)$	
		$\mathtt{in}[\mathtt{l}][\tau_1;\tau_2](e)$	$\mathtt{l}\cdot e$	
		$\mathtt{in}[\mathtt{r}][\tau_1;\tau_2](e)$	$\mathtt{r}\cdot e$	
		$\mathtt{case}(e;x_1.e_1;x_2.e_2)$	$\mathtt{case}\,e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\mid\mathtt{r}\cdot x_2\hookrightarrow e_2\}$	

- Dynamics

$$\frac{e\longmapsto e'}{\mathtt{abort}(e)\longmapsto\mathtt{abort}(e')}$$
$$\frac{e\longmapsto e'}{\mathtt{case}\,e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\mid\mathtt{r}\cdot x_2\hookrightarrow e_2\}\longmapsto\mathtt{case}\,e'\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\mid\mathtt{r}\cdot x_2\hookrightarrow e_2\}}$$
$$\frac{[e\ \mathtt{val}]}{\mathtt{l}\cdot e\ \mathtt{val}}\qquad\left[\frac{e\longmapsto e'}{\mathtt{l}\cdot e\longmapsto\mathtt{l}\cdot e'}\right]$$
$$\frac{[e\ \mathtt{val}]}{\mathtt{case}\,\mathtt{l}\cdot e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\mid\mathtt{r}\cdot x_2\hookrightarrow e_2\}\longmapsto[e/x_1]e_1}$$
$$\frac{[e\ \mathtt{val}]}{\mathtt{r}\cdot e\ \mathtt{val}}\qquad\left[\frac{e\longmapsto e'}{\mathtt{r}\cdot e\longmapsto\mathtt{r}\cdot e'}\right]$$
$$\frac{[e\ \mathtt{val}]}{\mathtt{case}\,\mathtt{r}\cdot e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\mid\mathtt{r}\cdot x_2\hookrightarrow e_2\}\longmapsto[e/x_2]e_2}$$

Finite Sums

- Syntax

Typ	τ ::=	$\mathtt{sum}(\{i\hookrightarrow\tau_i\}_{i\in I})$	$[\tau_i]_{i\in I}$	sum
Exp	e ::=	$\mathtt{in}[i](\vec{\tau})(e)$	$i\cdot e$	injection
		$\mathtt{case}(e;\{i\hookrightarrow x_i.e_i\}_{i\in I})$	$\mathtt{case}\,e\,\{i\cdot x_i\hookrightarrow e_i\}_{i\in I}$	case analysis

- Statics

$$\frac{\Gamma\vdash e:\tau_k\quad(1\le k\le n)}{\Gamma\vdash i_k\cdot e:[i_1\hookrightarrow\tau_1,\ldots,i_n\hookrightarrow\tau_n]}\qquad(11.3a)$$
$$\frac{\Gamma\vdash e:[i_1\hookrightarrow\tau_1,\ldots,i_n\hookrightarrow\tau_n]\quad\Gamma,x_1:\tau_1\vdash e_1:\tau\quad\ldots\quad\Gamma,x_n:\tau_n\vdash e_n:\tau}{\Gamma\vdash\mathtt{case}\,e\,\{i_1\cdot x_1\hookrightarrow e_1\mid\ldots\mid i_n\cdot x_n\hookrightarrow e_n\}:\tau}\qquad(11.3b)$$

- Dynamics

$$\frac{[e\ \mathtt{val}]}{i\cdot e\ \mathtt{val}}\qquad(11.4a)$$
$$\left[\frac{e\longmapsto e'}{i\cdot e\longmapsto i\cdot e'}\right]\qquad(11.4b)$$
$$\frac{e\longmapsto e'}{\mathtt{case}\,e\,\{i\cdot x_i\hookrightarrow e_i\}_{i\in I}\longmapsto\mathtt{case}\,e'\,\{i\cdot x_i\hookrightarrow e_i\}_{i\in I}}\qquad(11.4c)$$
$$\frac{i\cdot e\ \mathtt{val}}{\mathtt{case}\,i\cdot e\,\{i\cdot x_i\hookrightarrow e_i\}_{i\in I}\longmapsto[e/x_i]e_i}\qquad(11.4d)$$

Type Algebra

- void and unit types (nullary sum and product)
  - $|\tau\times\mathtt{unit}|=|\tau|$, $|\tau|$表示τ类型值的个数
  - $|\tau+\mathtt{void}|=|\tau|$
  - If $\tau=\mathtt{bool}$, then $\mathtt{bool}\times\mathtt{unit}$ has values:
    - (true, null), (false, null)
    - 将unit 加入pair，对数据结构并未增加额外的信息
  - $\mathtt{bool}+\mathtt{void}$ has values
    - in[l](bool;void)(true), in[l](bool;void)(false)

The algebra (and calculus!) of algebraic data types

Some Useful Sum Types

- Enumerate types
  - Type $\mathtt{card}\triangleq\mathtt{unit}+(\mathtt{unit}+(\mathtt{unit}+\mathtt{unit}))$
  - Introduction (values): hearts | spades |diamonds| clubs
    - $\mathtt{hearts}\triangleq\mathtt{in}[\mathtt{l}](\mathtt{card})\ \mathtt{null}$ $\quad\mathtt{l}\cdot\mathtt{null}$
    - $\mathtt{spades}\triangleq\mathtt{in}[\mathtt{r}](\mathtt{card})\ (\mathtt{in}[\mathtt{l}](\mathtt{unit};\mathtt{unit}+\mathtt{unit})\ \mathtt{null}),\ \ \mathtt{r}\cdot(\mathtt{l}\cdot\mathtt{null})$
    - …
  - Elimination
    - $\mathtt{case}(\mathtt{hearts};f_1;f_2;f_3;f_4)=f_1\ \mathtt{null}$
    - $\mathtt{case}(\mathtt{spades};f_1;f_2;f_3;f_4)=f_2\ \mathtt{null}$
    - …

Exp	e ::=	$\mathtt{abort}[\tau](e)$	$\mathtt{abort}(e)$
		$\mathtt{in}[\mathtt{l}][\tau_1;\tau_2](e)$	$\mathtt{l}\cdot e$
		$\mathtt{in}[\mathtt{r}][\tau_1;\tau_2](e)$	$\mathtt{r}\cdot e$
		$\mathtt{case}(e;x_1.e_1;x_2.e_2)$	$\mathtt{case}\,e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\mid\mathtt{r}\cdot x_2\hookrightarrow e_2\}$

Some Useful Sum Types

- Option types
  - Type $\mathtt{option}_\tau\triangleq\mathtt{unit}+\tau$
  - Introduction (values): null | just(M)
    - $\mathtt{null}\triangleq\mathtt{in}[\mathtt{l}](\mathtt{option}_\tau)\ \mathtt{null}$ $\quad\mathtt{l}\cdot\mathtt{null}$
    - $\mathtt{just}(M)\triangleq\mathtt{in}[\mathtt{r}](\mathtt{option}_\tau)\ M$ $\quad\mathtt{r}\cdot M$
    - …
  - Elimination
    - Typing $\mathtt{ifnull}_\tau:\mathtt{option}_\tau\to(\mathtt{unit}\to\rho)\to(\tau\to\rho)\to\rho$
    - Forms
      - $\mathtt{ifnull}_\tau\ \mathtt{null}\ \{\lambda\_:\mathtt{unit}.\,e_1\mid\lambda x:\tau.\,e_2\}\mapsto e_1$
      - $\mathtt{ifnull}_\tau\ \mathtt{just}(M)\{\lambda\_:\mathtt{unit}.\,e_1\mid\lambda x:\tau.\,e_2\}\mapsto e_2$

Some Useful Sum Types

- 理解空指针错误——**option**类型的意义之一
  - 在**OO**语言中，所有对象都是引用**(**指针**)**，对象的引用可能为空，不能通过空引用来访问对象的域类型
  - 如何避免空指针错误**?**
    一些语言提供空指针的检测函数 $\text{null}: \tau \rightarrow \text{bool}$
    if null(*e*) then …*error* … else …*ok* …
  - 但是空指针异常仍然普遍, 原因**: 1)**缺少空指针检测**; 2)**极少在程序的异常处进行空指针检测
  - 解决：用option_τ 描述类型为 τ 的可选值类型，其值或者为τ类型的值，或者为空.
    - 消去形式 $\text{ifnull}_\tau\ e\ \{\lambda\_ : \text{unit}. \ldots error \ldots | \lambda x{:}\tau. \ldots ok \ldots\}$

Lists in OCaml

let lst = [1;2;3]
let empty = []

[1;2;3]: int **list**
[]: t **list** for any type t

let longer = 5::lst
let another = 5::1::2::3::[]

If **e1 : t** and **e2 : t list** then **e1::e2 : t list**

```
let rec sum xs =
        match xs with
        | [] -> 0
        | h::t -> h + sum t
let six = sum lst
let zero = sum empty
```


Some Types in OCaml

	Define	Build	Access
Variant	`type`	Constructor name	Pattern matching
Record	`type`	Record expression with `{...}`	Pattern matching OR field selection with dot operator `.`
Tuple	N/A	Tuple expression with `(...)`	Pattern matching OR `fst` or `snd`

- **Variants**: one-of types aka **sum types**
  - type t = C1 | ... | Cn
- **Records, tuples**: each-of types aka **product types**
  - type t = {f1:t1; ...; fn:tn}
  - {f1=p1; ...; fn=pn} (e1,e2,...,en)
  - e.f

Lists are Just Variants

- OCaml effectively codes up lists as variants

  - **type** 'a list = [ ] | :: **of** 'a * 'a list

    - **list** is a type constructor parameterized on type variable **'a**
    - **[ ]** and **::** are constructors
    - Just a bit of syntactic magic in the compiler to use **[ ]** and **::** instead of alphabetic identifiers

Options are Just Variants

- OCaml effectively codes up options as variants

  - **type** 'a option = None | Some **of** 'a

    - **option** is a type constructor parameterized on type variable **'a**
    - **None** and **Some** are constructors

Exceptions are (mostly) Just Variants

- OCaml effectively codes up exceptions as slightly strange variants
  - **type** exn
  - **exception** MyNewException **of** string

    - Type **exn** is an *extensible* variant that may have new constructors added after its original definition
    - Raise exceptions with **raise e**, where **e** is a value of type **exn**
    - Handle exceptions with pattern matching, just like you would process any variant