Spring 2018

$\lambda^{\times,+,\rightarrow}$

Algebraic Data Types

Yu Zhang

Course web site: http://staff.ustc.edu.cn/~yuzhang/fopl

Recap

- $L^{\text{int,bool}}$
  - Grammar: type, term ---- *syntactically* valid
  - Semantics: judgements, rules
  - Dynamics
    - Values

      a val $\qquad \dfrac{}{z \text{ val}}(\text{V-z})$
    - Small-step semantics, *structural*)

      $t \mapsto t' \qquad \dfrac{t \mapsto t'}{s(t) \mapsto s(t')}(\text{D-s})$
  - Statics: Type system (typing rules)

    $t : \tau \qquad \dfrac{t : \text{int}}{s(t) : \text{int}}(\text{T-s})$
  - interpreter.ml

Recap (con't)

- $\lambda^{\rightarrow}$ (typed lambda calculus) $\quad \text{Type } \tau ::= \text{int} \mid \tau_1 \rightarrow \tau_2$ $\quad \text{Term } t ::= n \mid t_1 + t_2 \mid x \mid \lambda(x:\tau).t' \mid t_1 t_2$
  - Values

    $\dfrac{}{n \text{ val}}(\text{V-n}) \qquad \dfrac{}{\lambda(x:\tau).t \text{ val}}(\text{V-fn})$
  - Dynamics

    $\dfrac{t_1 \mapsto t_1'}{t_1 t_2 \mapsto t_1' t_2}(\text{D-app}_1) \qquad \dfrac{}{(\lambda(x:\tau).t_1)t_2 \mapsto [t_2 / x]t_1}(\text{D-app}_2)$

    $\dfrac{t_1 \mapsto t_1'}{t_1 + t_2 \mapsto t_1' + t_2}(\text{D-add}_1) \qquad \dfrac{t_1 \text{ val} \quad t_2 \mapsto t_2'}{t_1 + t_2 \mapsto t_1 + t_2'}(\text{D-add}_2)$

    $\dfrac{n_3 = n_1 + n_2}{n_1 + n_2 \mapsto n_3}(\text{D-add}_3)$

Recap (con't)

- $\lambda^{\rightarrow}$ (typed lambda calculus) $\quad \text{Type } \tau ::= \text{int} \mid \tau_1 \rightarrow \tau_2$ $\quad \text{Term } t ::= n \mid t_1 + t_2 \mid x \mid \lambda(x:\tau).t' \mid t_1 t_2$
  - Statics

    Γ: typing context

    $\dfrac{}{n : \text{int}}(\text{T-n}) \qquad \dfrac{\Gamma \vdash t_1 : \text{int} \quad \Gamma \vdash t_2 : \text{int}}{\Gamma \vdash t_1 + t_2 : \text{int}}(\text{T-add})$

    $\dfrac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda(x : \tau_1).t : \tau_1 \rightarrow \tau_2}(\text{T-fn}) \qquad \dfrac{\Gamma \vdash t_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1 \ t_2 : \tau_2}(\text{T-app})$

    $\dfrac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}(\text{T-var})$

    Introduction rule of function type (引入该类型的值)

    Elimination rule of function type (该类型的表达式上可以进行的操作)

Recap (con't)

- $L^{\text{int,bool}}$

  A language is *sound* if satisfying two theorems:
  - **Progress**:

    if $t : \tau$ then either t val or $\exists t'$ such that $t \mapsto t'$
  - **Preservation**:

    if $t : \tau$ and $t \mapsto t'$ then $t' : \tau$

    Prove the theorems by *induction* on the *typing rules*
- $\lambda^{\rightarrow}$ (typed lambda calculus)

  $\text{Type } \tau ::= \text{int} \mid \tau_1 \rightarrow \tau_2$

  $\text{Term } t ::= n \mid t_1 + t_2 \mid x \mid \lambda(x:\tau).t' \mid t_1 t_2$

References

- PFPL
  - Chapter 10 Product Types
  - Chapter 11 Sum Types

    (Void & Unit, Boolean, Enumerate, Options)
- TAPL
  - Chapter 11 Simple Extensions
- The algebra (and calculus!) of algebraic data types

Algebraic Data Types

- Product types
- Sum types (variants)

$$\text{Type } \tau ::= \cdots \mid \tau_1 \times \tau_2 \quad \text{product}$$
$$\mid \tau_1 + \tau_2 \quad \text{sum}$$

$$\text{Term } t ::= \cdots \mid (t_1, t_2) \quad \text{pair}$$
$$\mid t.d \quad \text{projection}$$
$$\mid \text{inj } t = d \text{ as } \tau \quad \text{injection}$$
$$\mid \text{case } t \{x_1 \rightsquigarrow t_1 \mid x_2 \rightsquigarrow t_2\} \quad \text{case}$$

$$\text{Direction } d ::= L \mid R \quad \text{left |right}$$

Product Types: Semantics

- Statics
  - Introduction rule $\dfrac{\Gamma \vdash t_1 : \tau_1 \quad \Gamma \vdash t_2 : \tau_2}{\Gamma \vdash (t_1, t_2) : \tau_1 \times \tau_2}$ (T-pair)
  - Elimination rules $\dfrac{\Gamma \vdash t : \tau_1 \times \tau_2}{\Gamma \vdash t.L : \tau_1}$ (T-project-L) $\quad \dfrac{\Gamma \vdash t : \tau_1 \times \tau_2}{\Gamma \vdash t.R : \tau_2}$ (T-project-R)
- Dynamics

$$\frac{t_1 \mapsto t_1'}{(t_1, t_2) \mapsto (t_1', t_2)} \text{(D-pair}_1\text{)} \quad \frac{t_1 \text{ val} \quad t_2 \mapsto t_2'}{(t_1, t_2) \mapsto (t_1, t_2')} \text{(D-pair}_2\text{)} \quad \frac{t_1 \text{ val} \quad t_2 \text{ val}}{(t_1, t_2) \text{ val}} \text{(D-pair}_3\text{)}$$

$$\frac{t \mapsto t'}{t.d \mapsto t'.d} \text{(D-proj}_1\text{)} \quad \frac{(t_1, t_2) \text{ val}}{(t_1, t_2).L \mapsto t_1} \text{(D-proj}_2\text{)} \quad \frac{(t_1, t_2) \text{ val}}{(t_1, t_2).R \mapsto t_2} \text{(D-proj}_3\text{)}$$

Product Types: Examples

$\lambda(x : \text{int} \times \text{int}).x.L + x.R$

$\lambda(x : (\text{int} \to \text{int}) \times \text{int}).x.L\, x.R$

$(1, (2, (3, 4))) : \text{int} \times (\text{int} \times (\text{int} \times \text{int}))$

Record Types: Syntactic Sugar

- Grammar

$$\text{Type } \tau ::= \cdots \mid \{l_1 : \tau_1, \cdots, l_k : \tau_k\}$$
$$\text{Term } t ::= \cdots \mid (l_1 = t_1, \cdots, l_k = t_k)$$
$$\mid t.l_i \quad (1 \le i \le k)$$

- Statics
  - Introduction rule
  - Elimination rules
- Dynamics

Product Types: Examples

$\lambda(x : \text{int} \times \text{int}).x.L + x.R$

- 函数: 对pair中的两个元素求和

$\lambda(x : (\text{int} \to \text{int}) \times \text{int}).x.L\, x.R$

- 函数: 接收由一个函数和一个整数组成的pair, 以该整数为参数调用这个函数

$(1, (2, (3, 4))) : \text{int} \times (\text{int} \times (\text{int} \times \text{int}))$

- 由pairs产生n元组

Sum Types: Examples

$(\text{inj } l = L \text{ as } \text{int} + (\text{int} \to \text{int})) : \text{int} + (\text{int} \to \text{int})$

$\text{case } (\text{inj } l = L \text{ as } \text{int} + (\text{int} \to \text{int}))\{x_1 \rightsquigarrow x_1 + 1 \mid x_2 \rightsquigarrow x_2\ 2\} \mapsto l + 1$

Sum Types: Examples

$(\text{inj}\ \text{l} = L\ \text{as int+(int} \to \text{int)}) : \text{int+(int} \to \text{int)}$

- 将左标记项包装成一个sum类型的值

$\text{case}\ (\text{inj}\ \text{l} = L\ \text{as int+(int} \to \text{int}))\{x_1 \rightsquigarrow x_1 + 1 \mid x_2 \rightsquigarrow x_2\ 2\} \mapsto \text{l} + 1$

- 用case算子对sum值分情况处理，当前是左标记项，故按 $x_1 \rightsquigarrow x_1 + 1$计算，得到 $\text{l} + 1$

Type Algebra

- **void** and **unit** types

$\text{Type}\ \tau ::= \cdots \mid \text{void} \mid \text{unit}$

$\text{Term}\ t ::= \cdots \mid \text{null}$

null : unit
void类型没有值

- 相等的类型

$|\tau \times \text{unit}| = |\tau| \qquad |\tau + \text{void}| = |\tau|$

若τ = bool， 有类型为 $\text{bool} \times \text{unit}$ 的项(true, null)、(false, null)

- 将unit 类型加入pair ，对数据结构并未增加额外的信息

类型 $\text{bool} + \text{void}$ 有两个值 $\text{inj true} = L\ \text{as bool} + \text{void}$
$\text{inj false} = L\ \text{as bool} + \text{void}$

[The algebra (and calculus!) of algebraic data types]

Sum Types: Statics

- Introduction rules

$$\frac{\Gamma \vdash t : \tau_1}{\Gamma \vdash \text{inj}\ t = L\ \text{as}\ \tau_1 + \tau_2 : \tau_1 + \tau_2}\ (\text{T-inject-L})$$

$$\frac{\Gamma \vdash t : \tau_2}{\Gamma \vdash \text{inj}\ t = R\ \text{as}\ \tau_1 + \tau_2 : \tau_1 + \tau_2}\ (\text{T-inject-R})$$

- Elimination rule

$$\frac{\Gamma \vdash t : \tau_1 + \tau_2 \quad \Gamma, x_1 : \tau_1 \vdash t_1 : \tau \quad \Gamma, x_2 : \tau_2 \vdash t_2 : \tau}{\Gamma \vdash \text{case}\ t\ \{x_1 \rightsquigarrow t_1 \mid x_2 \rightsquigarrow t_2\} : \tau}\ (\text{T-case})$$

Some Useful Sum Types

- Enumerate types
  - 例：扑克牌的花色
    - 类型 $\text{card} \triangleq \text{unit+(unit+(unit+unit))}$
    - 引入形式（值）： $\text{hearts} \mid \text{spades} \mid \text{diamonds} \mid \text{clubs}$

$\text{hearts} \triangleq \text{inj null} = L\ \text{as card}$
$\text{spades} \triangleq \text{inj (inj null} = L\ \text{as (unit+(unit+unit)))} = R\ \text{as card}$
.....

    - 消去形式

$\text{case hearts as card}\ \{f_1 \mid f_2 \mid f_3 \mid f_4\} = f_1\ *$
$\text{case spades as card}\ \{f_1 \mid f_2 \mid f_3 \mid f_4\} = f_2\ *$
.....

Sum Types: Dynamics

- inject

$$\frac{t \mapsto t'}{\text{inj}\ t = d\ \text{as}\ \tau \mapsto \text{inj}\ t' = d\ \text{as}\ \tau}\ (\text{D-inject}_1)$$

$$\frac{t\ \text{val}}{\text{inj}\ t = d\ \text{as}\ \tau\ \text{val}}\ (\text{D-inject}_2)$$

- case

$$\frac{t \mapsto t'}{\text{case}\ t\ \{x_1 \rightsquigarrow t_1 \mid x_2 \rightsquigarrow t_2\} \mapsto \text{case}\ t'\ \{x_1 \rightsquigarrow t_1 \mid x_2 \rightsquigarrow t_2\}}\ (\text{D-case}_1)$$

$$\frac{t\ \text{val}}{\text{case inj}\ t = L\ \text{as}\ \tau\ \{x_1 \rightsquigarrow t_1 \mid x_2 \rightsquigarrow t_2\} \mapsto [t / x_1]t_1}\ (\text{D-case}_2)$$

$$\frac{t\ \text{val}}{\text{case inj}\ t = R\ \text{as}\ \tau\ \{x_1 \rightsquigarrow t_1 \mid x_2 \rightsquigarrow t_2\} \mapsto [t / x_2]t_2}\ (\text{D-case}_3)$$

Some Useful Sum Types

- Option types
  - 类型 $\text{option}_\tau \triangleq \text{unit}+\tau$
  - 引入形式（值）： $\text{null} \mid \text{just}(M)$

$\text{null} \triangleq \text{inj null} = L\ \text{as option}_\tau$
$\text{just}(M) \triangleq \text{inj}\ M = R\ \text{as option}_\tau, \quad M : \tau$

  - 消去形式

$\text{ifnull}_\tau : \text{option}_\tau \to (\text{unit} \to \rho) \to (\tau \to \rho) \to \rho$
$\text{ifnull}_\tau\ \text{null}\ \{\lambda\_ : \text{unit}.e_1 \mid \lambda x : \tau.e_2\} \mapsto e_1$
$\text{ifnull}_\tau\ \text{just}(M)\ \{\lambda\_ : \text{unit}.e_1 \mid \lambda x : \tau.e_2\} \mapsto [M / x]e_2$

Some Useful Sum Types

理解空指针错误——option类型的意义之一

- **起因**：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.
- **如何避免空指针错误?**
  一些语言提供空指针的检测函数 null : $\tau \to$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$_\tau$ 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 ifnull$_\tau$ e {λ_:unit. …*error*…| $\lambda x{:}\tau$. …*ok*…}

Reduction Strategies

- 操作语义与符号解释器
  - 不确定的解释器：每一步可选择任意的子表达式进行归约.
  - 确定的解释器：每一步选择一个特定的"下一步归约".
  - 并行的解释器：可以把几个归约同时作用于不相交的子表达式.
- 确定的归约：归约策略
  - 归约策略：是项到项的部分函数F，它具有性质：
    如果$F(e)=e'$，那么$e \to e'$.
  - 求值函数：

$$eval_F(e_1) = \begin{cases} e_1 & \text{if } F(e_1) \text{ is not defined} \\ e_2 & \text{if } F(e_1) = e_1' \text{ and } eval_F(e_1') = e_2 \end{cases}$$

Reduction Strategies

- 三种归约策略
  - 例，若$e_1 \to e_1'$且$e_2 \to e_2'$，则$(\lambda x{:}\sigma.e_1)\ e_2$可以归约成
  - $[e_2/x]\ e_1$：**最左最外归约，最左归约，惰性归约**. 对于函数应用，称为**按名调用**
    若e_1中有形参x的多次出现，用e_2代换x会使函数体中有多个e_2，从而e_2的归约会重复多次.
  - $(\lambda x{:}\sigma.\ e_1)\ e_2'$：**急切归约.**对于函数应用，称为**按值调用**
    若e_2 无范式，则$(\lambda x{:}\sigma.\ e_1)\ e_2$的急切归约不会终止.
    若e_2在最终的结果中不使用，则最左归约方式因无需归约e_2而终止.
  - $(\lambda x{:}\sigma.e_1')\ e_2$：在提供$e_2$前试图**"优化"函数$\lambda x{:}\sigma.e_1$**