Algebraic Data Types: $L^{\times,+,\rightarrow}$

Yu Zhang

http://staff.ustc.edu.cn/~yuzhang/pldpa/

yuzhang@ustc.edu.cn

References

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

Outline

- Total Functions
  - System T of Higher-order Recursion
- Finite Data Types
  - Product type
  - Sum type

Algebraic Data Types: $L^{\times,+,\rightarrow}$

Outline

- Total Functions
  - System T of Higher-order Recursion
- Finite Data Types
  - Product type
  - Sum type

Algebraic Data Types: $L^{\times,+,\rightarrow}$

System T: *Gödel's T*

- E: $L^{\text{num,str}}$; T: $L^{\text{nat,}\rightarrow}$ (primitive recursion,原始递归式)
- Syntax

		抽象语法	具体语法		
Typ	τ ::=	nat	nat	naturals	自然数类型
		arr(τ_1; τ_2)	$\tau_1 \rightarrow \tau_2$	function	函数类型
Exp	e ::=	x	x	variable	变量
introduction		z	z	zero	零
		s(e)	s(e)	successor	后继
elimination		rec{e_0; $x.y.e_1$}(e)	rec e {z $\hookrightarrow$ e_0 | s(x) with $y \hookrightarrow e_1$}	recursion	递归式
introduction		lam{τ}($x.e$)	$\lambda\,(x:\tau)\,e$	abstraction	抽象
elimination		ap(e_1; e_2)	$e_1(e_2)$	application	应用

  - 对于每种类型（如nat），有如下两类表达式
    - 引入形式(introduction)：引入这种类型的值，如z和s(e)
    - 消去形式(elimination)：使用这种类型的值，如rec递归式

System T: *Gödel's T*

- E: $L^{\text{num,str}}$; T: $L^{\text{nat},\to}$ (primitive recursion,原始递归)
- Syntax
  - 递归式Recursor
    - $\text{rec}\{e_0; x.y.e_1\}(e)$ $\quad\quad$ $\text{rec}\ e\{\text{z} \hookrightarrow e_0 | s(x) \text{ with } y \hookrightarrow e_1\}$
    - e-fold iteration of the transformation $x.y.e_1$ starting from e_0,

      x : predecessor; y : result of the x-fold iteration; e_1: result of recursive call

      例：以求阶乘 $n!$ 为例, n次迭代 $x.y.e_1$是从$e_0 = \text{z}\ !$ (即0!)开始， x为n的前驱， y 为x次迭代结果（即$x!$ ）， e_1为递归调用的结果（即$n!$ ，为$s(x) * y$ ）
- Recursor递归式 vs. Iterator 迭代式 $\text{iter}\{e_0; y.e_1\}(e)$

  迭代式中是$y.e_1$而不是$x.y.e_1$， y: result of the recursive call, no binding for the predecessor x

System T

- Derivability vs. Admissibility
  - Derivability (可导性) $\Gamma \vdash_R \mathcal{K}$
    - $\mathcal{K}$ is derivable from rules $R \cup \Gamma$
  - Admissibility (可纳性、可容许性) $\Gamma \models_R \mathcal{J}$
    - $\vdash_R \Gamma$ implies $\vdash_R \mathcal{J}$

      if any of the hypotheses Γ are *not* derivable relative to R, then $\mathcal{J}$ is *vacuously* true
  - If $\Gamma \vdash_R \mathcal{J}$, then $\Gamma \models_R \mathcal{J}$; but the converse fails

$$\frac{}{\mathtt{zero\ even}}$$

$$\frac{b\ \mathtt{odd}}{\mathtt{succ}(b)\ \mathtt{even}}\ (2.8)$$

$$\frac{a\ \mathtt{even}}{\mathtt{succ}(a)\ \mathtt{odd}}$$

$\mathtt{succ(zero)\ even} \not\vdash_{(2.8)} \mathtt{zero\ odd}$, 不满足可导性

$\mathtt{succ(zero)\ even} \models_{(2.8)} \mathtt{zero\ odd}$

满足可纳性(valid), because the hypothesis is false

$\mathtt{succ(zero)\ even} \vdash_{(2.8)} \mathtt{succ(succ(zero))\ odd}$

System T

- Statics of T

$$\frac{}{\Gamma, x:\tau \vdash x:\tau}$$

$$\frac{}{\Gamma \vdash z:\mathrm{nat}} \qquad \frac{\Gamma \vdash e:\mathrm{nat}}{\Gamma \vdash s(e):\mathrm{nat}}$$

$$\frac{\Gamma \vdash e:\mathrm{nat} \quad \Gamma \vdash e_0:\tau \quad \Gamma, x:\mathrm{nat}, y:\tau \vdash e_1:\tau}{\Gamma \vdash \mathrm{rec}\{e_0; x.y.e_1\}(e):\tau}$$

假设Γ下可推导出e是nat类型、$e_0:\tau$，$\Gamma, x:\mathrm{nat}, y:\tau$下可推导出$e_1:\tau$，则Γ 下可推导出递归式$\mathrm{rec}\{e_0; x.y.e_1\}(e)$是$\tau$类型

$$\frac{\Gamma, x:\tau_1 \vdash e:\tau_2}{\Gamma \vdash \mathrm{lam}\{\tau_1\}(x.e):\mathrm{arr}(\tau_1;\tau_2)} \qquad \frac{\Gamma \vdash e_1:\mathrm{arr}(\tau_2;\tau) \quad \Gamma \vdash e_2:\tau_2}{\Gamma \vdash \mathrm{ap}(e_1;e_2):\tau}$$

  - Substitution (Admissibility)

    If $\Gamma \vdash e:\tau$ and $\Gamma, x:\tau \vdash e':\tau'$, then $\Gamma \vdash [e/x]e':\tau'$

    注：代换引理是可纳的，因为上面的陈述是 If XX then YY这种蕴含关系

System T

- Statics of T

$$\frac{}{\Gamma, x:\tau \vdash x:\tau} \qquad \frac{}{\Gamma \vdash z:\mathrm{nat}} \qquad \frac{\Gamma \vdash e:\mathrm{nat}}{\Gamma \vdash s(e):\mathrm{nat}}$$

假设Γ下可推导出e是nat类型、$e_0:\tau$，$\Gamma, x:\mathrm{nat}, y:\tau$下可推导出$e_1:\tau$，则Γ 下可推导出递归式$\mathrm{rec}\{e_0; x.y.e_1\}(e)$是$\tau$类型

$$\frac{\Gamma \vdash e:\mathrm{nat} \quad \Gamma \vdash e_0:\tau \quad \Gamma, x:\mathrm{nat}, y:\tau \vdash e_1:\tau}{\Gamma \vdash \mathrm{rec}\{e_0; x.y.e_1\}(e):\tau}$$

$$\frac{\Gamma, x:\tau_1 \vdash e:\tau_2}{\Gamma \vdash \mathrm{lam}\{\tau_1\}(x.e):\mathrm{arr}(\tau_1;\tau_2)} \qquad \frac{\Gamma \vdash e_1:\mathrm{arr}(\tau_2;\tau) \quad \Gamma \vdash e_2:\tau_2}{\Gamma \vdash \mathrm{ap}(e_1;e_2):\tau}$$

  - Substitution (Admissibility)

    If $\Gamma \vdash e:\tau$ and $\Gamma, x:\tau \vdash e':\tau'$, then $\Gamma \vdash [e/x]e':\tau'$

- Dynamics ([…] omitted for lazy dynamics)

  - Value $\quad \dfrac{}{z\ \mathrm{val}} \qquad \dfrac{[e\ \mathrm{val}]}{s(e)\ \mathrm{val}} \qquad \dfrac{}{\mathrm{lam}\{\tau\}(x.e)\ \mathrm{val}}$

  - Transition $\left[\dfrac{e \longmapsto e'}{s(e) \longmapsto s(e')}\right]$

Search transitions
决定指令执行次序
Instruction transitions
基础的求值步

方括号中的规则包含时代表eager求值，否则为lazy求值

$$\frac{[e_2\ \mathrm{val}\ ]}{\mathrm{ap}(\mathrm{lam}\{\tau\}(x.e); e_2) \longmapsto [e_2/x]e}$$

$$\frac{e_1 \longmapsto e_1'}{\mathrm{ap}(e_1;e_2) \longmapsto \mathrm{ap}(e_1';e_2)} \qquad \left[\frac{e_1\ val \quad e_2 \longmapsto e_2'}{\mathrm{ap}(e_1;e_2) \longmapsto \mathrm{ap}(e_1;e_2')}\right]$$

System T

- Dynamics ([…] omitted for lazy dynamics)
  - Transition

$$\frac{e \mapsto e'}{\mathrm{rec}\{e_0; x.y.e_1\}(e) \mapsto \mathrm{rec}\{e_0; x.y.e_1\}(e')}$$

$$\frac{}{\mathrm{rec}\{e_0; x.y.e_1\}(z) \mapsto e_0} \qquad \frac{s(e)\ val}{\mathrm{rec}\{e_0; x.y.e_1\}\big(s(e)\big) \mapsto [e, \mathrm{rec}\{e_0; x.y.e_1\}(e\,)/x\,, y]e_1}$$

First evaluate the argument e of the recursion

If the argument of the recursion is z, the recursion is evaluated to e_0

If the argument of the recursion is $s(e)$, substitute x with e and y with $\mathrm{rec}\{e_0; x.y.e_1\}(e)$ that appears freely in e_1

System T

- Dynamics ([…] omitted for lazy dynamics)
  - Transition

$$\frac{e \longmapsto e'}{\mathrm{rec}\{e_0; x.y.e_1\}(e) \longmapsto \mathrm{rec}\{e_0; x.y.e_1\}(e')}$$

$$\frac{}{\mathrm{rec}\{e_0; x.y.e_1\}(z) \longmapsto e_0} \qquad \frac{s(e)\ val}{\mathrm{rec}\{e_0; x.y.e_1\}\big(s(e)\big) \longmapsto [e, \mathrm{rec}\{e_0; x.y.e_1\}(e\,)/x\,, y]e_1}$$

- Canonical Forms(范式)

范式引理明确了什么类型具有什么样的值

  - If $e : \tau$ and e val, then
    - If $\tau = nat$, then e=z or $e = s(e')$ for some e'
    - If $\tau = \tau_1 \to \tau_2$, then $e = \lambda(x{:}\,\tau_1)e_2$ for some e_2
- Safety
  - If $e : \tau$ and $e \longmapsto e'$, then $e' : \tau$ Preservation：求值中保持类型不变
  - If $e : \tau$, then either e val or $e \longmapsto e'$ for some e'

Progress：良类型的表达式或者是值，或者能被继续求值

Examples for Recursion

- Doubling

```
datatype nat = z | s of nat

fun double z = z
  | double (s x) = s (s (double x))

fun double e =
  case e of
    z => z
  | s x => let val y = double x in s (s y) end
```

double = λ(e : nat) rec{ z; *x*.y.s (s (y)) } (e) define **double** in System T

double s(z) ↦ rec{z; *x.y*.s(s(y))}(s(z))	Application
↦ [z ,rec{z; *x.y*.s(s(y))}(z)/x, y] s(s(y))	Since s(z) val
= s(s(rec{z; x.y.s(s(y))}(z)))	Substitution
↦ s(s(z))	By evaluation rule of s()

Expressiveness of System T

- Function $f : \mathbb{N} \to \mathbb{N}$ is **definable** in T, iff

  there exists $e_f : \text{nat} \to \text{nat}$ such that for all $n \in \mathbb{N}$, $e_f(\bar{n}) \equiv \overline{f(n)} : \text{nat}$

- Definitional equality for **T**, $\Gamma \vdash e \equiv e' : \tau$

$$\frac{\Gamma, \mathrm{x}:\tau_1 \vdash e_2 : \tau_2 \quad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash \text{ap}(\text{lam}\{\tau_1\}(x.\, e_2); e_1) \equiv [e_1/x]e_2 : \tau_2} \qquad \frac{\Gamma \vdash e_0 : \tau \quad \Gamma, x : \text{nat}, y : \tau \vdash e_1 : \tau}{\Gamma \vdash \text{rec}\{e_0; x.\, y.\, e_1\}(z) \equiv e_0 : \tau}$$

$$\frac{\Gamma \vdash e_0 : \tau \quad \Gamma, x : \text{nat}, y : \tau \vdash e_1 : \tau}{\Gamma \vdash \text{rec}\{e_0; x.\, y.\, e_1\}(s(e)) \equiv [e, \text{rec}\{e_0; x.\, y.\, e_1\}(e\,)/x\,, y]e_1 : \tau} \qquad (9.5)$$

- Doubling function $d(n) = 2 \times n$ is definable in T by

$e_d : \text{nat} \longrightarrow \text{nat} \qquad e_d = \lambda(x : \text{nat})\ \text{rec}\ x\{z \hookrightarrow z | s(u)\ \text{with}\ v \hookrightarrow s(s(v))\}$

Proof: $e_d(\bar{0}) \equiv \bar{0} : \text{nat}$ (basis)

Assume $e_d(\bar{n}) \equiv \overline{d(n)} : \text{nat}$ $\qquad e_d(\overline{n+1}) \equiv s(s(e_d(\bar{n}))) \equiv \text{s}(\text{s}(\overline{2 \times n}))$

$= \overline{2 \times (n+1)} \equiv \overline{d(n+1)}$

Ackermann

- Well-known Ackermann function

$$A(0,n) = n+1$$
$$A(m+1,0) = A(m,1)$$
$$A(m+1,n+1) = A(m,A(m+1,n))$$

  - not first-order primitive recursive
  - but is higher-order primitive recursive

  is incompatible with the recursion $\mathrm{rec}\{e_0; x.\, y.\, e_1\}(e)$

    - its argument need be *deconstructed* at every step

- Ackermann is definable in T
  - $A(m+1,n)$ iterates n times $A(m,-)$, starting with $A(m,1)$
  - Define **it**: $(\mathrm{nat} \longrightarrow \mathrm{nat}) \longrightarrow \mathrm{nat} \longrightarrow \mathrm{nat} \longrightarrow \mathrm{nat}$ as
    $\lambda(f\colon \mathrm{nat} \longrightarrow \mathrm{nat})\ \lambda\ (n\colon \mathrm{nat})\ \mathrm{rec}\ n\ \{\mathrm{z} \hookrightarrow \mathbf{id}\ |\mathrm{s}(x)\ \mathrm{with}\ g \hookrightarrow \boldsymbol{f} \circ \boldsymbol{g}\}$
    - $\mathbf{id} = \lambda(x\colon \mathrm{nat})x$, $\boldsymbol{f} \circ \boldsymbol{g} = \lambda(x\colon \mathrm{nat})f(g(x))$
  - $\mathbf{it}(f)(\overline{n})(\overline{m}) \equiv f^{(n)}(\overline{m})\colon \mathrm{nat}$

Ackermann

- Ackermann is definable in T
  - Define **it**: $(\mathrm{nat} \longrightarrow \mathrm{nat}) \longrightarrow \mathrm{nat} \longrightarrow \mathrm{nat} \longrightarrow \mathrm{nat}$ as

    $\lambda(f\colon \mathrm{nat} \longrightarrow \mathrm{nat})\ \lambda\ (n\colon \mathrm{nat})\ \mathrm{rec}\ n\ \{\mathrm{z} \hookrightarrow \mathbf{id}\ |\mathrm{s}(x)\ \mathrm{with}\ g \hookrightarrow \boldsymbol{f} \circ \boldsymbol{g}\}$

    - $\mathrm{id} = \lambda(x\colon \mathrm{nat})x,\ f \circ g = \lambda(x\colon \mathrm{nat})f(g(x))$
  - $\mathrm{it}(f)(\bar{n})(\bar{m}) \equiv f^{(n)}(\bar{m})\colon \mathrm{nat}$
  - Ackermann $e_a\colon \mathrm{nat} \longrightarrow \mathrm{nat} \longrightarrow \mathrm{nat}$ is

    $\lambda(m\colon \mathrm{nat})\ \mathrm{rec}\ m\ \{z \hookrightarrow s\ \mid\ s(x)\ \mathrm{with}\ f \hookrightarrow \lambda(n\colon \mathrm{nat})\ \mathrm{it}(f)(n)(f(\bar{1}))\}$

    - $e_a(\bar{0})(\bar{n}) \equiv s(\bar{0})$
    - $e_a(\overline{m+1})(\bar{0}) \equiv e_a(\bar{m})(\bar{1})$
    - $e_a(\overline{m+1})(\overline{n+1}) \equiv e_a(\bar{m})\big(e_a\big(s(\bar{m})\big)(\bar{n})\big)$

    That is, the Ackermann function is definable in **T**

Outline

- Total Functions
  - System T of Higher-order Recursion
- Finite Data Types
  - Product type
  - Sum type

Algebraic Data Types: $L^{\times,+,\rightarrow}$

Product Types

- Nullary and Binary products
  - Abstract syntax

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

  - Examples

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

Examples

- $\lambda(x\colon \mathrm{nat} \times \mathrm{nat}).\, x \cdot \mathrm{l} + x \cdot \mathrm{r}$
- $\lambda(x\colon (\mathrm{nat} \to \mathrm{nat}) \times \mathrm{nat}).\, x \cdot \mathrm{l}\ \ x \cdot \mathrm{r}$
- $(1, (2, (3, 4)))\colon \mathrm{nat} \times (\mathrm{nat} \times (\mathrm{nat} \times \mathrm{nat}))$

Examples

- $\lambda(x\colon \mathrm{nat} \times \mathrm{nat}).\, x \cdot \mathrm{l} + x \cdot \mathrm{r}$
  - 函数：对pair中的两个元素求和

- $\lambda(x\colon (\mathrm{nat} \to \mathrm{nat}) \times \mathrm{nat}).\, x \cdot \mathrm{l}\ \ x \cdot \mathrm{r}$
  - 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
- $(1, (2, (3, 4)))\colon \mathrm{nat} \times (\mathrm{nat} \times (\mathrm{nat} \times \mathrm{nat}))$
  - pairs产生**4**元组

Nullary and Binary Products

- Statics $\dfrac{}{\Gamma \vdash \langle\rangle : \mathtt{unit}}$

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

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

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

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

Product type can represent *structure type in C*

- Dynamics

Values introduction

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

Search transitions Elimination

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

Search transitions (omitted for lazy dynamics)

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

Instruction transitions Elimination

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

Finite Products

Product type can represent *structure type in C*

- Syntax

$$\begin{array}{llllll} \mathsf{Typ} & \tau & ::= & \mathtt{prod}(\{i \hookrightarrow \tau_i\}_{i \in I}) & \langle \tau_i \rangle_{i \in I} & \text{product} \\ \mathsf{Exp} & e & ::= & \mathtt{tpl}(\{i \hookrightarrow e_i\}_{i \in I}) & \langle e_i \rangle_{i \in I} & \text{tuple} \\ & & & \mathtt{pr}[i](e) & e \cdot i & \text{projection} \end{array}$$

可以称为带标签的积类型，或者记录类型
I 是字段名（或域名）集合

- Statics

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

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

- Dynamics

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

$$\left[\frac{\left\{ \begin{array}{c} e_1 \ \mathsf{val} \quad \ldots \quad e_{j-1} \ \mathsf{val} \quad e'_1 = e_1 \quad \ldots \quad e'_{j-1} = e_{j-1} \\ e_j \longmapsto e'_j \quad e'_{j+1} = e_{j+1} \quad \ldots \quad e'_n = e_n \end{array} \right\}}{\langle i_1 \hookrightarrow e_1, \ldots, i_n \hookrightarrow e_n \rangle \longmapsto \langle i_1 \hookrightarrow e'_1, \ldots, i_n \hookrightarrow e'_n \rangle} \right] \quad (10.4b)$$

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

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

Nullary and Binary Sum

对τ_1类型的表达式e通过左注入(injection)构造器，构造出和类型$\tau_1 + \tau_2$的值

- Syntax

Typ	τ	::=	$\mathtt{void}$	$\mathtt{void}$	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
			$\mathtt{in}[\mathtt{r}]\{\tau_1;\tau_2\}(e)$	$\mathtt{r}\cdot e$	right injection
elimination			$\mathtt{case}(e;x_1.e_1;x_2.e_2)$	$\mathtt{case}\,e\,\{\mathtt{l}\cdot x_1 \hookrightarrow e_1 \mid \mathtt{r}\cdot x_2 \hookrightarrow e_2\}$	case analysis

Elimination of void

- Examples

$$\mathtt{l}\cdot\mathtt{z} : \mathtt{nat} + \tau$$

$$\mathtt{r}\cdot\mathtt{s}(\mathtt{z}) : \tau + \mathtt{nat}$$

$$\mathtt{l}\cdot\mathtt{r}\cdot\lambda(x:\mathtt{nat})\,x : (\tau_1 + (\mathtt{nat}\to\mathtt{nat})) + \tau_2$$

$$\mathtt{case}\ \mathtt{l}\{\mathtt{nat}\to\mathtt{nat};\mathtt{nat}\}\cdot\lambda(x:\mathtt{nat})\,x\ \{\mathtt{l}\cdot x_1 \hookrightarrow x_1(\mathtt{z}) \mid \mathtt{r}\cdot x_2 \hookrightarrow x_2\}$$

和类型的值（通过左注入构造的）

$$\mathtt{bool} \triangleq \mathtt{unit} + \mathtt{unit}$$

$$\mathtt{true} \triangleq \mathtt{l}\cdot\langle\rangle$$

$$\mathtt{false} \triangleq \mathtt{r}\cdot\langle\rangle$$

Nullary and Binary Sum

- Statics

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

$$\frac{\Gamma \vdash e : \mathtt{void}}{\Gamma \vdash \mathtt{abort}(e) : \tau}$$

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathtt{l} \cdot e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathtt{r} \cdot e : \tau_1 + \tau_2}$$

Sum type can represent *union type in C*

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

- Dynamics

$$\frac{e \longmapsto e'}{\mathtt{abort}(e) \longmapsto \mathtt{abort}(e')}$$

$$\frac{[e\ \mathrm{val}]}{\mathtt{l} \cdot e\ \mathrm{val}} \qquad \left[\frac{e \longmapsto e'}{\mathtt{l} \cdot e \longmapsto \mathtt{l} \cdot e'}\right]$$

$$\frac{[e\ \mathrm{val}]}{\mathtt{r} \cdot e\ \mathrm{val}} \qquad \left[\frac{e \longmapsto e'}{\mathtt{r} \cdot e \longmapsto \mathtt{r} \cdot e'}\right]$$

$$\frac{e \longmapsto e'}{\mathtt{case}\, e\, \{\mathtt{l} \cdot x_1 \hookrightarrow e_1 \mid \mathtt{r} \cdot x_2 \hookrightarrow e_2\} \longmapsto \mathtt{case}\, e'\, \{\mathtt{l} \cdot x_1 \hookrightarrow e_1 \mid \mathtt{r} \cdot x_2 \hookrightarrow e_2\}}$$

$$\frac{[e\ \mathrm{val}]}{\mathtt{case}\, \mathtt{l} \cdot e\, \{\mathtt{l} \cdot x_1 \hookrightarrow e_1 \mid \mathtt{r} \cdot x_2 \hookrightarrow e_2\} \longmapsto [e/x_1]e_1}$$

$$\frac{[e\ \mathrm{val}]}{\mathtt{case}\, \mathtt{r} \cdot e\, \{\mathtt{l} \cdot x_1 \hookrightarrow e_1 \mid \mathtt{r} \cdot x_2 \hookrightarrow e_2\} \longmapsto [e/x_2]e_2}$$

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

$$\frac{\Gamma \vdash e : \tau_k \quad (1 \leq k \leq n)}{\Gamma \vdash i_k \cdot e : [i_1 \hookrightarrow \tau_1, \ldots, i_n \hookrightarrow \tau_n]} \tag{11.3a}$$

$$\frac{\Gamma \vdash e : [i_1 \hookrightarrow \tau_1, \ldots, i_n \hookrightarrow \tau_n] \quad \Gamma, x_1 : \tau_1 \vdash e_1 : \tau \quad \ldots \quad \Gamma, x_n : \tau_n \vdash e_n : \tau}{\Gamma \vdash \mathsf{case}\, e\, \{i_1 \cdot x_1 \hookrightarrow e_1 \mid \ldots \mid i_n \cdot x_n \hookrightarrow e_n\} : \tau} \tag{11.3b}$$

- Dynamics

$$\frac{[e\ \mathsf{val}]}{i \cdot e\ \mathsf{val}} \tag{11.4a}$$

$$\left[\frac{e \longmapsto e'}{i \cdot e \longmapsto i \cdot e'}\right] \tag{11.4b}$$

$$\frac{e \longmapsto e'}{\mathsf{case}\, e\, \{i \cdot x_i \hookrightarrow e_i\}_{i \in I} \longmapsto \mathsf{case}\, e'\, \{i \cdot x_i \hookrightarrow e_i\}_{i \in I}} \tag{11.4c}$$

$$\frac{i \cdot e\ \mathsf{val}}{\mathsf{case}\, i \cdot e\, \{i \cdot x_i \hookrightarrow e_i\}_{i \in I} \longmapsto [e/x_i]e_i} \tag{11.4d}$$

Type Algebra

- void and unit types (nullary sum and product)
  - $|\tau \times \text{unit}| = |\tau|$, $|\tau|$表示 τ类型值的个数
  - $|\tau + \text{void}| = |\tau|$
  - If $\tau = \text{bool}$, then $\text{bool} \times \text{unit}$ 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
  - $\text{bool} + \text{void}$ has values
    - in[l](bool;void)(true), in[l](bool;void)(false)

The algebra (and calculus!) of algebraic data types

Some Useful Sum Types

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

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

Some Useful Sum Types

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

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 $\text{null}: \tau \to \text{bool}$

    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 ifnull_τ e $\{\lambda \_ : \text{unit}. \ldots error \ldots | \lambda x: \tau \ldots . ok \ldots\}$

Primitive Mutual Recursion原始互递归

- Using products to simplify primitive recursors in T
  - T: $\mathrm{rec}\{e_0; x.\, y.\, e_1\}(e)$
  - Using products: $\mathrm{iter}\{e_0; x.\, e_1\}(e)$,
    - x : the recursive result on the predecessor
    - define $\mathrm{rec}\{e_0; x.\, y.\, e_1\}(e)$ to be $e' \cdot \mathrm{r}$

      $e' \triangleq \mathrm{iter}\{\langle \mathrm{z}, e_0\rangle; x'.\langle \mathrm{s}(x'.\mathrm{l}), [x'.\mathrm{l}, x'.\mathrm{r}/x, y]e_1\rangle\}(e)$
- Mutual primitive recursion
  - $\lambda(n{:}\,\mathrm{nat})\mathrm{iter}\, n\{\mathrm{z} \hookrightarrow \langle 1,0\rangle | \mathrm{s}(b) \hookrightarrow \langle b.\mathrm{r}, b.\mathrm{l}\rangle\}$

$$e_{\mathtt{ev}} \triangleq \lambda\,(n : \mathtt{nat})\, e_{\mathtt{eo}}(n) \cdot \mathtt{l}$$

$$e_{\mathtt{od}} \triangleq \lambda\,(n : \mathtt{nat})\, e_{\mathtt{eo}}(n) \cdot \mathtt{r}.$$

$$e(0) = 1$$

$$o(0) = 0$$

$$e(n+1) = o(n)$$

$$o(n+1) = e(n)$$

THANKS