General Recursion

Yu Zhang

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

References

- PFPL
  - Chapters
    - 19 System PCF of Recursive Functions
    - * 20 System FPC of Recursive Types
- TAPL

Outline

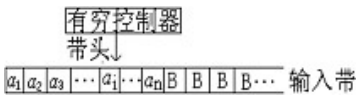
- Recursion theorem
- Recursive function
  - Example: factorial function in Lua
  - General recursor (fix_σ operator) and general recursion
    - (一般递归式和一般递归)
- Statics & Dynamics
- (Briefly intro.) Recursive types

Recursion Theorem

- Recursion theorem：computability theory
- Computable functions (可计算函数) f
  - f is computable if it can be calculated by a finite mechanical procedure
  - 1936 Church(美): computable functions are general recursive functions (一般递归函数)
    （**判定性问题** Entscheidungs problem ）
    - **和 Kleene在 20 世纪三十年代引入 untyped λ calculus**

Recursion Theorem

- Recursion theorem：computability theory
- Computable functions (可计算函数) f
  - 1936, Turing(英): 提出Turing机,any computable function is Turing computable —Church-Turing Thesis (论点)
  - 1936，Kleene(美) : 证明一般递归函数就是Turing机所计算的函数。



Recursion

- Example: factorial function in Lua

```
local function fact(n)
  if n == 0 then return 1
  else return n * fact(n - 1) end
end
```

Can we represent the recursion as any other kind of term?

```
local x = 0
local y = x + y
```

- A new kind of function where f is a variable: $\mathbf{fn}\ f(x:\tau).t$

  $\mathbf{fn}\ f(x:\mathbf{int}).\mathbf{if}\ x = 0\ \mathbf{then}\ 1\ \mathbf{else}\ x * f(x-1)$

Self-reference

Recursion: self-reference

- Dynamics

$$\frac{t_2 \text{ val}}{(\text{fn } f(x{:}\tau).\, t_1)\, t_2 \mapsto [t_2/x, (\text{fn } f(x{:}\tau).\, t_1)/f]t_1}$$

$$\begin{aligned}
&(\textbf{fn } f(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1))\ 2 \\
\mapsto\ &[2/x, (\textbf{fn } f(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1))/f] \\
&\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1) \\
\equiv\ &\textbf{if } 2 = 0 \textbf{ then } 1 \textbf{ else } 2 * \\
&(\textbf{fn } f(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1))(2-1) \\
\mapsto\ &\textbf{if false then } 1 \textbf{ else } 2 * \\
&(\textbf{fn } f(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1))(2-1) \\
\mapsto\ &2 * (\textbf{fn } f(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1))(2-1)
\end{aligned}$$

Need generalize the mechanism to work for any term!

General Recursion: *fixpoint* operator

- General recursion(一般递归式): $fix(F)$
  - $f : \mathbb{N} \to \mathbb{N}$, such that $f = F(f)$
  - f is defined to be $fix(F)$
  - fix: higher-order operator on functionals F
- Example $\quad \textbf{fn } f(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1)$

  Use fixpoint operator fix:

  $\textbf{fix}(\lambda(f : \textbf{int} \to \textbf{int}).\lambda(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1))$

  $F \triangleq \lambda(f : \text{int} \to \text{int}).\lambda(x : \text{int}).\text{if } = 0 \text{ then } 1 \text{ else } x * f(x-1)$
- What is the type of fix ?

General Recursion: *fixpoint* operator

- General recursion(一般递归式): $fix(F)$
  - f is defined to be $fix(F)$
- Example $\quad \textbf{fn } f(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1)$

  Use fixpoint operator fix:

  $\textbf{fix}(\lambda(f : \textbf{int} \to \textbf{int}).\lambda(x : \textbf{int}).\textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x * f(x-1))$

  $F \equiv \lambda(f : \text{int} \to \text{int}).\lambda(x : \text{int}).\text{if } = 0 \text{ then } 1 \text{ else } x * f(x-1)$
- What is the type of fix ?
  - $fix_\sigma : (\sigma \to \sigma) \to \sigma$
  - e.g. σ is int→int for the above factorial function example

Fixpoint

- **Fixpoint**: If $F : \sigma \to \sigma$ is a high order function, then the fixpoint of F is $x : \sigma$ where $F(x) = x$
- Fixpoint operator fix_σ
  - Equality Axiom: $fix_\sigma = \lambda(f : \sigma \to \sigma).\, f(fix_\sigma\, f)$
  - $fix_\sigma\, F = F\ (fix_\sigma\, f)$ generates a fixpoint of F
  - Reduction Rule: $fix_\sigma \mapsto \lambda(f : \sigma \to \sigma).\, f(fix_\sigma\, f)$
- Example: $fact \equiv fix_{\text{int}\to\text{int}}\, F$, $fact$ is a fixpoint of F
  - $fact\ \ n \equiv (fix\ F)\ n \Rightarrow F(fix\ F)\ n$

    $\equiv (\lambda(f : \text{int} \to \text{int}).\lambda(x : \text{int}).\text{if } = 0 \text{ then } 1 \text{ else } x * f(x-1))(fix\ F)\ n$

    $\Rightarrow \text{if } = 0 \text{ then } 1 \text{ else } n * (fix\ F)(n-1)$

Some Examples of Fixpoints

- 自然数上的平方函数的不动点
- 恒等函数的不动点
- 后继函数的不动点
- $F \equiv \lambda(f : \text{int} \to \text{int}).\lambda(x : \text{int}).\text{if } = 0 \text{ then } 1 \text{ else } x * f(x-1)$ 的不动点

Some Examples of Fixpoints

- 自然数上的平方函数的不动点
  - 0 and 1
- 恒等函数的不动点
  - 无数个
- 后继函数的不动点
  - 无
- $F \equiv \lambda(f : \text{int} \to \text{int}).\lambda(x : \text{int}).\text{if } = 0 \text{ then } 1 \text{ else } x * f(x-1)$ 的不动点
  - 阶乘函数

System PCF of Recursive Functions

- Syntax

Typ	τ	::=	nat	nat	naturals
			parr(τ_1; τ_2)	$\tau_1 \rightharpoonup \tau_2$	partial function
Exp	e	::=	x	x	variable
			z	z	zero
			s(e)	s(e)	successor
			ifz{e_0; $x.e_1$}(e)	ifz e {z $\hookrightarrow e_0$ | s(x) $\hookrightarrow e_1$}	zero test
			lam{τ}($x.e$)	$\lambda(x:\tau)\,e$	abstraction
			ap(e_1; e_2)	$e_1(e_2)$	application
			fix{τ}($x.e$)	fix $x:\tau$ is e	recursion

- fix{τ}($x.e$) is general recursion (一般递归)
- ifz{e_0; $x.e_1$}(e): branches according to whether e evaluates to z, binding the predecessor to x

PCF: Statics

- Statics

$$\frac{}{\Gamma, x:\tau \vdash x:\tau} \quad (19.1a)$$

$$\frac{}{\Gamma \vdash \mathtt{z} : \mathtt{nat}} \quad (19.1b)$$

$$\frac{\Gamma \vdash e : \mathtt{nat}}{\Gamma \vdash \mathtt{s}(e) : \mathtt{nat}} \quad (19.1c)$$

$$\frac{\Gamma \vdash e : \mathtt{nat} \quad \Gamma \vdash e_0 : \tau \quad \Gamma, x : \mathtt{nat} \vdash e_1 : \tau}{\Gamma \vdash \mathtt{ifz}\{e_0; x.e_1\}(e) : \tau} \quad (19.1d)$$

$$\frac{\Gamma, x:\tau_1 \vdash e : \tau_2}{\Gamma \vdash \mathtt{lam}\{\tau_1\}(x.e) : \mathtt{parr}(\tau_1;\tau_2)} \quad (19.1e)$$

$$\frac{\Gamma \vdash e_1 : \mathtt{parr}(\tau_2;\tau) \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathtt{ap}(e_1;e_2) : \tau} \quad (19.1f)$$

$$\frac{\Gamma, x:\tau \vdash e : \tau}{\Gamma \vdash \mathtt{fix}\{\tau\}(x.e) : \tau} \quad (19.1g)$$

PCF: Dynamics

- Values

$$\frac{}{\mathtt{z}\ \mathtt{val}} \qquad \frac{[e\ \mathtt{val}]}{\mathtt{s}(e)\ \mathtt{val}} \qquad \frac{}{\mathtt{lam}\{\tau\}(x.e)\ \mathtt{val}}$$

- Transitions

$$\left[\frac{e \longmapsto e'}{\mathtt{s}(e) \longmapsto \mathtt{s}(e')}\right] \quad (19.3a)$$

$$\frac{e \longmapsto e'}{\mathtt{ifz}\{e_0; x.e_1\}(e) \longmapsto \mathtt{ifz}\{e_0; x.e_1\}(e')} \quad (19.3b)$$

$$\frac{}{\mathtt{ifz}\{e_0; x.e_1\}(\mathtt{z}) \longmapsto e_0} \quad (19.3c)$$

$$\frac{\mathtt{s}(e)\ \mathtt{val}}{\mathtt{ifz}\{e_0; x.e_1\}(\mathtt{s}(e)) \longmapsto [e/x]e_1} \quad (19.3d)$$

$$\frac{e_1 \longmapsto e_1'}{\mathtt{ap}(e_1;e_2) \longmapsto \mathtt{ap}(e_1';e_2)} \quad (19.3e)$$

$$\left[\frac{e_1\ \mathtt{val} \quad e_2 \longmapsto e_2'}{\mathtt{ap}(e_1;e_2) \longmapsto \mathtt{ap}(e_1;e_2')}\right] \quad (19.3f)$$

$$\frac{[e_2\ \mathtt{val}]}{\mathtt{ap}(\mathtt{lam}\{\tau\}(x.e);e_2) \longmapsto [e_2/x]e} \quad (19.3g)$$

$$\frac{}{\mathtt{fix}\{\tau\}(x.e) \longmapsto [\mathtt{fix}\{\tau\}(x.e)/x]e} \quad (19.3h)$$

Normal-order Reduction

Unwinding the recursion

Recursive Types

- Examples: list
  - natlist = null: unit + cons: nat x natlist
    - 该等式蕴含递归定义
  - Recursive types: introduce a recursive operator μ
    natlist = $\mu\, t$.null: unit + cons: nat x t
    将natlist 定义为满足 t = unit + cons: nat x t 的无穷的类型
- What's relationship between $\mu\, t.\tau$ and $[\,\mu\, t.\tau/t\,]\tau$
  - Equiv-recursive(相等递归): $\mu\, t.\tau = [\,\mu\, t.\tau/t\,]\tau$
    - 问题：类型表达式会有无穷个
  - Iso-recursive(同构递归): $\mu\, t.\tau$与$[\,\mu\, t.\tau/t\,]\tau$同构但不等

Iso-Recursive Types

- Examples
  - Recursive type: natlist = $\mu\, t$. null:unit+cons: nat×t
  - Expansion (or unrolling)
    - null:unit+cons: nat×$\mu\, t$.(null:unit+ cons: nat×t)
- Syntax

Types τ ::= t | rec($t.\tau$) t是关于类型名的元变量

Expr's e ::= fold[$t.\tau$](e) | unfold(e)

抽象语法	具体语法	
rec($t.\tau$)	$\mu\, t.\tau$	递归类型，其展开式为[rec($t.\tau$) /t]τ
fold[$t.\tau$](e)	fold(e)	引入形式：折叠， e : [rec($t.\tau$) /t]τ
unfold(e)	unfold(e)	消去形式：展开， e : rec($t.\tau$)

Iso-Recursive Types

- Statics
  - 范型判断: $\Delta \mid \tau$ type
    - Δ是一组有限的形如t_i type的假设集合
    - t_i为类型变量

$$\frac{}{\Delta, t\ \mathtt{type} \mid t\ \mathtt{type}} \qquad \frac{\Delta \mid \tau_1\ \mathtt{type} \quad \Delta \mid \tau_2\ \mathtt{type}}{\Delta \mid \mathtt{arr}(\tau_1;\tau_2)\ \mathtt{type}} \qquad \frac{\Delta, t\ \mathtt{type} \mid \tau\ \mathtt{type}}{\Delta \mid \mathtt{rec}(t.\tau)\ \mathtt{type}}$$

  - 定型判断: $\Gamma \vdash e : \tau$

$$\frac{\Gamma \vdash e : [\mathtt{rec}(t.\tau)/t]\tau}{\Gamma \vdash \mathtt{fold}[t.\tau](e) : \mathtt{rec}(t.\tau)} \qquad \frac{\Gamma \vdash e : \mathtt{rec}(t.\tau)}{\Gamma \vdash \mathtt{unfold}(e) : [\mathtt{rec}(t.\tau)/t]\tau}$$

Iso-Recursive Types

- Dynamics

$$\frac{\{e\ \mathsf{val}\}}{\mathtt{fold}[t.\tau]\,(e)\ \mathsf{val}}$$

eager语义下，e是值，则其fold形式是值

$$\left\{\frac{e \mapsto e'}{\mathtt{fold}[t.\tau]\,(e) \mapsto \mathtt{fold}[t.\tau]\,(e')}\right\}$$

eager语义下,e未完成求值,则允许在fold下归约

$$\frac{e \mapsto e'}{\mathtt{unfold}(e) \mapsto \mathtt{unfold}(e')}$$

e未完成求值，则允许在unfold下归约

$$\frac{\{e\ \mathsf{val}\}}{\mathtt{unfold}(\mathtt{fold}[t.\tau]\,(e)) \mapsto e}$$

eager语义下,对值e执行fold再unfold，所得为e

- 惰性(lazy)语义 省去{}中的规则或前提
- 急切(eager)语义 包含{}中的规则或前提

- Safety(略)