General Recursion

Yu Zhang

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

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*$_\sigma$ 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

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

$$(\text{fn } f(x : \text{int}).\text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1))\ 2$$

$$\mapsto [2/x, (\text{fn } f(x : \text{int}).\text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1))/f]$$

$$\text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1)$$

$$\equiv \text{if } 2 = 0 \text{ then } 1 \text{ else } 2 *$$

$$(\text{fn } f(x : \text{int}).\text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1))(2-1)$$

$$\mapsto \text{if false then } 1 \text{ else } 2 *$$

$$(\text{fn } f(x : \text{int}).\text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1))(2-1)$$

$$\mapsto 2 * (\text{fn } f(x : \text{int}).\text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1))(2-1)$$

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

  Use fixpoint operator fix:

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

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

  Use fixpoint operator fix:

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

  $F \equiv \lambda(f: \text{int} \to \text{int}).\,\lambda(x: \text{int}).\,\text{if} = 0\ \text{then}\ 1\ \text{else}\ x * f(x-1)$
- 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 $\boldsymbol{F}$ is $x:\sigma$ where $F(x) = x$
- Fixpoint operator $\boldsymbol{fix_\sigma}$
  - **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*
  - $fact\ \ n \equiv (fix\ F)\ n\ \Rightarrow F(fix\ F)\ n$

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

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

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
- $F \equiv \lambda(f\colon \mathrm{int} \to \mathrm{int}).\,\lambda(x\colon \mathrm{int}).\,\mathrm{if}\ x = 0\ \mathrm{then}\ 1\ \mathrm{else}\ x * f(x-1)$

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

的不动点

  - 阶乘函数

System PCF of Recursive Functions

- Syntax

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

- $\mathtt{fix}\{\tau\}(x.e)$ is general recursion (一般递归)
- $\mathtt{ifz}\{e_0; x.e_1\}(e)$: 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

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

- Transitions

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

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

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

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

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

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

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

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

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 ::= $t \mid \mathrm{rec}(t.\tau)$ t是关于类型名的元变量

Expr's e ::= $\mathrm{fold}[t.\tau](e) \mid \mathrm{unfold}(e)$

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

Iso-Recursive Types

- Statics
  - **类型形成**判断：$\Delta \mid \tau \ \text{type}$
    - Δ是一组有限的形如$t_i \ \text{type}$的假设集合
    - t_i为类型变量

$$\frac{}{\Delta, t \ \text{type} \mid t \ \text{type}}$$

$$\frac{\Delta \mid \tau_1 \ \text{type} \quad \Delta \mid \tau_2 \ \text{type}}{\Delta \mid \texttt{arr}(\tau_1; \tau_2) \ \text{type}}$$

$$\frac{\Delta, t \ \text{type} \mid \tau \ \text{type}}{\Delta \mid \texttt{rec}(t.\tau) \ \text{type}}$$

  - **定型**判断：$\Gamma \vdash e : \tau$

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

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

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)

```
36  let rec typecheck (t : Term.t) : Type.t option =
37    match t with
38    | Term.Z -> Some (Type.Int)
39    | Term.S t' ->
40      let tau = typecheck t' in
41      (match tau with
42       | Some Type.Int -> Some (Type.Int)
43       | _ -> None)
44    | Term.True -> Some (Type.Bool)
45    | Term.False -> Some (Type.Bool)
46    | Term.If (t1, t2, t3) ->
47      (match typecheck t1 with
48       | Some Type.Bool ->
49         (match typecheck t2 with
50          | Some tau ->
51            (match typecheck t3 with
52             | Some tau' ->
53               if tau = tau' then Some tau
54               else None
55             | _ -> None)
56          | _ -> None)
57       | _ -> None)
58    | Term.Iszero t' ->
59      match typecheck t' with
60      | Some Type.Int -> Some Type.Bool
61      | _ -> None
```

- Interpreter.ml

Some, None, option

```
16  module Type = struct
17    type t =
18      | Int
19      | Bool
20  end
```

```
22  module Term = struct
23    type t =
24      | Z
25      | S of t
26      | True
27      | False
28      | If of t * t * t
29      | Iszero of t
30  end
```

```
63  exception Unreachable

68  let rec eval (t : Term.t) : Term.t =
69    match t with
70    | Term.Z -> Term.Z
71    | Term.S t' -> Term.S (eval t')
72    | Term.True -> Term.True
73    | Term.False -> Term.False
74    | Term.If (t1, t2, t3) ->
75      eval (match eval t1 with
76        | Term.True -> t2
77        | Term.False -> t3
78        | _ -> raise Unreachable)
79    | Term.Iszero t' ->
80      (match eval t' with
81        | Term.Z -> Term.True
82        | Term.S _ -> Term.False
83        | _ -> raise Unreachable)
```

```
22  module Term = struct
23    type t =
24      | Z
25      | S of t
26      | True
27      | False
28      | If of t * t * t
29      | Iszero of t
30  end
```

- Interpreter.ml

Some, None, option

```
16  module Type = struct
17    type t =
18      | Int
19      | Bool
20  end
```

```
85  let main () =
86    let t1 = Term.Z in
87    assert (typecheck t1 = Some Type.Int);
88    assert (eval t1 = Term.Z);
89
90    let t2 = Term.Iszero Term.Z in
91    assert (typecheck t2 = Some Type.Bool);
92    assert (eval t2 = Term.True);
93
94    let t3 = Term.Iszero Term.True in
95    assert (typecheck t3 = None)
96
97  let () = main ()
```

THANKS