Fall 2018

General Recursion

Yu Zhang

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

Recap

- $\lambda^{\times,+,\rightarrow}$
  - Terms: introduction and elimination forms of a type
  - Statics: introduction and elimination rules
  - Dynamics: values, structural small-step semantics
  - Type safety: progress and preservation
- Type Algebra
  - void (0), unit (1), bool (1+1), Maybe a (a+1), Either a b (a+b), (a,b) (axb), a->b (b^a)
- Useful sum types: enumerate, option$_\tau$,
  - Why using option$_\tau$?

Recap

- Reduction strategies $(\lambda u.\lambda v.v)((\lambda x.x\ x)(\lambda x.x\ x))$
  - Normal-order reduction: left-most, outer-most redex first
  - Applicative-order reduction: left-most, **inner-most** redex first

Recap

- Reduction strategies $(\lambda u.\lambda v.v)((\lambda x.x\ x)(\lambda x.x\ x))$
  - Normal-order reduction: left-most, outer-most redex first

    $(\lambda u.\lambda v.v)((\lambda x.x\ x)(\lambda x.x\ x))$

    $\rightarrow \lambda v.v$

  - Applicative-order reduction: left-most, **inner-most** redex first

    $(\lambda u.\lambda v.v)\ ((\lambda x.x\ x)(\lambda x.x\ x))$

    $\rightarrow (\lambda u.\lambda v.v)\ ((\lambda x.x\ x)(\lambda x.x\ x))$

    $\rightarrow \ldots$

Recap

- Reduction strategies similar to (but subtly different from) ***evaluation strategies*** in language theories
  - **Call-by-name** (like normal-order)
    - ALGOL 60
  - **Call-by-need** ("memorized version" of call-by-name)
    - Haskell, R, …
  - **Call-by-value** (like applicative-order)
    - C, …
  - …

arguments are not evaluated, but directly substituted into function body

called "lazy evaluation", avoids repeated evaluations

called "eager evaluation"

References

- PFPL
  - Chapter 9 System T of Higher-Order Recursion
    - System T: well-known as *Gödel's T*
  - Chapter 19 System PCF of Recursive Functions
  - * Chapter 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

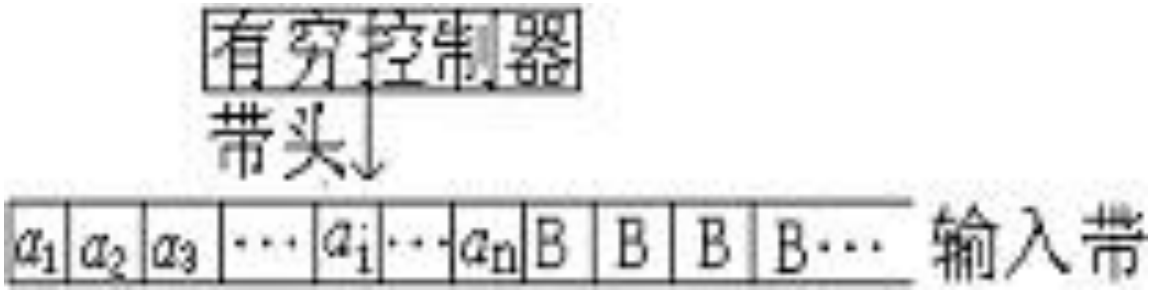
Recursion Theorem

- 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

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

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

Need generalize the mechanism to work for any term!

General Recursion: *fixpoint* operator

- General recursion(一般递归式): $\mathbf{fix}(\lambda(x:\sigma).t)$
  - Read this as "a term *t* where *x* = *t* "
- Example $\mathbf{fn}\ f(x:\mathbf{int}).\mathbf{if}\ x=0\ \mathbf{then}\ 1\ \mathbf{else}\ x * f(x-1)$

  Use fixpoint operator fix:

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

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

General Recursion: *fixpoint* operator

- General recursion(一般递归式): $\mathbf{fix}(\lambda(x:\sigma).t)$
  - Read this as "a term *t* where *x* = *t* "
- Example $\mathbf{fn}\ f(x:\mathbf{int}).\mathbf{if}\ x=0\ \mathbf{then}\ 1\ \mathbf{else}\ x * f(x-1)$

  Use fixpoint operator fix:

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

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

  $$\mathbf{fix}_\sigma : (\sigma\to\sigma)\to\sigma$$

  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: $\text{fix}_\sigma = \lambda f : \sigma \to \sigma . f(\text{fix}_\sigma\ f)$
    - $\text{fix}_\sigma F = F(\text{fix}_\sigma\ F)$ generates a fixpoint of F
  - Reduction Rule: $\text{fix}_\sigma \mapsto \lambda f : \sigma \to \sigma . f(\text{fix}_\sigma\ f)$
- Example: $fact \triangleq fix_{\text{int}\to\text{int}}\ F$, $fact$ is a fixpoint of F

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

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
- $\mathbf{F} \triangleq \lambda(f : \mathrm{int} \to \mathrm{int}).\lambda(n : \mathrm{int}).\mathbf{if}\ x = 0\ \mathbf{then}\ 1\ \mathbf{else}\ x * f(x-1)$ 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
- $\mathbf{F} \triangleq \lambda(f : \text{int} \to \text{int}).\lambda(n : \text{int}).\text{if } x = 0 \text{ then } 1 \text{ else } x * f(x-1)$ 的不动点
  - 阶乘函数

General Recursion: Semantics

- General recursion: $\mathbf{fix}(\lambda(x:\sigma).t)$
  - Fixpoint of *F*: fix(*F*) = *F*(fix(*F*))
- Statics

$$\frac{\Gamma \vdash t : \sigma \to \sigma}{\Gamma \vdash \mathbf{fix}(t) : \sigma}\,(\text{T-fix})$$

- Dynamics

$$\frac{t \mapsto t'}{\mathbf{fix}(t) \mapsto \mathbf{fix}(t')}\,(\text{D-fix}_1)$$

Which reduction strategy?

$$\frac{}{\mathbf{fix}(\lambda(x:\sigma).t) \mapsto [\mathbf{fix}(\lambda(x:\sigma).t)\ /\ x]t}\,(\text{D-fix}_2)$$

Unwinding the recursion

General Recursion: Semantics

- General recursion: $\mathbf{fix}(\lambda(x:\sigma).t)$
  - Fixpoint of *F*: fix(*F*) = *F*(fix(*F*))
- Statics

$$\frac{\Gamma \vdash t : \tau \to \tau}{\Gamma \vdash \mathbf{fix}(t) : \tau}\,(\text{T-fix})$$

- Dynamics

$$\frac{t \mapsto t'}{\mathbf{fix}(t) \mapsto \mathbf{fix}(t')}\,(\text{D-fix}_1)$$

Normal-order Reduction

$$\frac{}{\mathbf{fix}(\lambda(x:\sigma).t) \mapsto [\mathbf{fix}(\lambda(x:\sigma).t)\ /\ x]t}\,(\text{D-fix}_2)$$

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
  - 一般断言：$\Delta \mid \tau$ 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

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

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

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

  - 定型断言： $\Gamma \vdash e : \tau$

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

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

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)