PLC Review

TA - Sinong

Principles

- Programming languages have four properties:
  - Syntax
  - Names
  - Types
  - Semantics
- There are four main programming paradigms:
  - Imperative
  - Object-oriented
  - Functional
  - Logic (declarative)

Inductive Definition

- Inductive Definition
  - Axioms
  - Proper Rules
- Derivation (Tree)
- Proof by Induction
  - If X then A
  - Proof: by induction on the derivation of J
  - Use cases belongs to part of X

Theorem 3: If even2 n, then even n.

Proof: By induction on the derivation of even2 n.

Case: $$\frac{}{\mathit{even2}\quad Z}\,\mathit{even2Z}$$

even Z (by rule evenZ)

Case: $$\frac{\mathit{even2}\quad n}{\mathit{even2}\quad (S\,(S\,n))}\,\mathit{even2S}$$

(1) even n (by I.H.)

Need to prove: even (S (S n))

(2) odd (S n) (by (1), oddS)

(3) even (S (S n)) (by (2), evenS)

QED.

Lambda Calculus (Thoery)

- Basic Syntax
  - Variable
  - Function
  - Application
- Binding – Scope
  - Free Variable
  - Substitution
- Evaluation Strategies
  - Call by Value
  - Call by Name (Lazy Evaluation, *bonus point in Project*)
- Multi-Step Operation

Lambda Calculus (Practical)

- Tru = \t.\f. t
- Fls = \t.f. f
- And = \b.\c. b c Fls
- Succ = \n.\f.\x. f (n f x)

Simply-Typed Lambda Calculus

Syntax

e ::=		expressions:
	x	(variable)
|	true	(true value)
|	false	(false value)
|	if e1 then e2 else e3	(conditional)
|	\x : t . e	(abstraction)
|	e1 e2	(application)
v ::=		values:
	true	(true value)
|	false	(false value)
|	\x : t . e	(abstraction value)

Simply-Typed Lambda Calculus

t ::=		types:
	bool	(base boolean type)
|	$t_1 \rightarrow t_2$	(type of functions)
Γ ::=		contexts:
	.	(empty context)
|	Γ, x: t	(variable binding)

Simply-Typed Lambda Calculus

Typing

$[\Gamma \vdash e : t]$

$$\frac{x : t \in \Gamma}{\Gamma \mid\!- x : t} \quad \text{(T-Var)}$$

$$\frac{}{\Gamma \mid\!- \mathit{true} : \mathit{bool}} \quad \text{(T-True)}$$

$$\frac{}{\Gamma \mid\!- \mathit{false} : \mathit{bool}} \quad \text{(T-False)}$$

$$\frac{\Gamma \mid\!- e_1 : \mathit{bool} \quad \Gamma \mid\!- e_2 : t \quad \Gamma \mid\!- e_3 : t}{\Gamma \mid\!- \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t} \quad \text{(T-If)}$$

$$\frac{\Gamma, x : t_1 \mid\!- e_2 : t_2}{\Gamma \mid\!- \lambda x : t_1 . e_2 : t_1 \to t_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \mid\!- e_1 : t_{11} \to t_{12} \qquad \Gamma \mid\!- e_2 : t_{11}}{\Gamma \mid\!- e_1 \; e_2 : t_{12}} \quad \text{(T-App)}$$

Simply-Typed Lambda Calculus

Semantics

[e → e']

$$\frac{e_1 \rightarrow e_1'}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \rightarrow \text{if } e_1' \text{ then } e_2 \text{ else } e_3} \quad (\text{E - if0})$$

$$\frac{}{\text{if } \mathit{true} \text{ then } e_2 \text{ else } e_3 \rightarrow e_2} \quad (\text{E - if1})$$

$$\frac{}{\text{if } \mathit{false} \text{ then } e_2 \text{ else } e_3 \rightarrow e_3} \quad (\text{E - if2})$$

$$\frac{e_1 \rightarrow e_1'}{e_1\ e_2 \rightarrow e_1'\ e_2} \quad (\text{E - App1}) \qquad \frac{e_2 \rightarrow e_2'}{v_1\ e_2 \rightarrow v_1\ e_2'} \quad (\text{E - App2})$$

$$\frac{}{(\lambda x : t.e)\ v \rightarrow e[v / x]} \quad (\text{E - AppAbs})$$

Simply-Typed Lambda Calculus

- Properties of Simply-Typed Lambda Calculus
  - Uniqueness of Typing
  - Inversion for Typing
  - Well-typedness
  - Exchange Lemma
  - Weakening Lemma
- Progress Theorem
- Preservation Theorem

Simply-Typed Lambda Calculus

- Let Function
- Static and Dynamic Scoping
- Pairs
- Tuples
- Records
- Sums
- Recursive function
  - Fix-point Combinator(Y combinator)
  - Mutually recursive combinator(*<u>Bonus Point in Project</u>*)

Simply-Typed Lambda Calculus

- List
- Environment Model
  - .[x v] = x v
  - (E, x’, v’)[x v] = E, x v if x = x’
  - (E, x’, v’)[x v] = E, x’ v’, x v if x != x’

Imperative

- Lambda calculus with value
  - Y = ref 1
  - !Y
- References(Machine State)
  - M is a partial function from location to values
  - $\boldsymbol{\Sigma}$ is a type relation for memory store
- Sequence

Example Evaluation

Program:

```
let x = ref 3 in
    let y = x in
        x := (!x) +1;
        !y
```

```
(., let x = ref 3 in
    let y = x in
    x:= (!x) + 1;
    y) →
(l 3, let x = l in
          let y = x in
          x := (!x) + 1;
          !y) →
(l 3, let y = l in
          l := (!l) + 1;
          !y) →
(l 3, l := (!l) + 1; !l) →
(l 3, l := 3 + 1; !l) →
(l 3, l := 4; !l) →
(l 4, (); !l) → (l 4, !l) → (l 4, 4)
```

Imperative

- Type Safety
  - Weakening
  - Preservation Theorem
  - Progress Theorem
- While loop
- Factorial(Example)
- Exception Handling
- Exception

Memory Management

- (Bonus point in Project)
- Memory Organization
  - Static area
  - Run-time stack
  - Heap
- Garbage Collection
  - Reference counting (RC)
  - Mark-and-Sweep (MB)
  - Copy collection (Half Space)

Type Inference

- Step1: Add Type Schemes
- Step2: Generate Constraints
- Step3: Solve Constraints
- Step4: Generate Types

- Constraint Generation Rules -> a set of constraints
- Composition of Substitutions
  - Preservation of Typing under type substitution
  - Principle Solution

Type Inference

- Unification
  - (S, q)->(S’, q’)
  - Unification Machine
  - Termination
- Properties of Solution
- Let Polymorphism
  - Let id = \x.x in (id 25, id true)

Subtyping

- Subtype polymorphism
  - Universal Polymorphism
  - Existential Poly morphism
- Basic Properties
  - Reflexivity
  - Transitivity
- Top-Subsumption
  - Top
  - Sub

Subtyping

- Subtype polymorphism
  - Universal Polymorphism
  - Existential Poly morphism
- Basic Properties
  - Reflexivity
  - Transitivity
- Top-Subsumption
  - Top
  - Sub

Subtyping

- Tuples
- Sums
- Functions
- Properties
  - Canonical Forms Lemma
  - Progress Lemma
  - Inversion of Subtyping Lemma
  - Component Typing Lemma
  - Substitution Lemma
  - Preservation Lemma
- Polymorphism
  - Subtype polymorphism
  - Parametric polymorphism
  - Ad-hoc polymorphism

Object-Oriented Programming

- Data Abstraction
- Encapsulation

- Ada
- Smalltalk

Logical Programming

- Horn Clauses and Predicates
- Resolution and Unification

- Prolog

OCaml

- Tail recursion (*Bonus point in Project*)
- OCaml

Final Examination

- Dec 11st, 14:00
- 2 Hour
- ERB 129
- 1 **Letter Size double sides Handwriting** Cheat Sheet
- Short Answer Questions
  - Design, Proof, Programming

- https://www.uta.edu/administration/registrar/calendars/final-exams