Administrivia

- Homework 9 is due Thursday 11:59 PM
- Please start submitting late assignments ASAP
- No class next Monday (Patriot's Day)
- Project update due by next Friday

Polymorphism I: An Introduction

Type Theory and Mechanized Reasoning

Lecture 20 + 21

CAS CS 400 (Spring 2024)

At a High Level

The behavior of a function often doesn't depend heavily on the argument of the function

`id x = x`	it doesn't matter what x is

`reverse [] = []` `reverse (x :: xs) = xs ++ [x]`	it matters it's a list but it doesn't matter what's in the list

`sort [] = []` `sort (x :: xs) = insert x (sort x)`	it doesn't matter what's in the list <u>except</u> that they are orderable

Kinds of Polymorphism

$\boxed{\text{Ad Hoc Polymorphism}}$ Define an interface that can be implemented on different types

e.g. Haskell Type Classes

$\boxed{\text{Subtype Polymorphism}}$ Relate types hierachically and inherit functionality

e.g. Java Classes

FOCUS OF TODAY

$\boxed{\text{Parametric Polymorphism}}$ Define functions over abstract type variables

e.g. OCaml, Agda, Haskell, ..

System F ($\lambda 2$)

System F (General Info.)

- STLC + Type Variables + Type Abstraction

- Introduced by Jean-Yves Girard and John C. Reynolds (independently) in 1972

- CH-corresponds to 2nd Order IPL (IPL with quantification over propositional variables)

Recall: Domain-full v.s. Domain-free Abstraction

In domain-free STLC:

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \to B}$$

simple λ-term

In domain-full STLC:

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A. M : A \to B}$$

type-annoted λ-term

In STLC, the distinction is minor (we lose uniqueness of typability)

System F* (types and terms)

* also called (domain-full) $\lambda 2$

$\mathcal{V} \triangleq$ Set of variable symbols

$$Ty ::= \mathcal{V} \mid Ty \to Ty \mid \Pi \mathcal{V}.\, Ty$$

$$Tm ::= \mathcal{V} \mid \lambda \mathcal{V}^{Ty}.\, Tm \mid Tm\ Tm \mid \Lambda \mathcal{V}.\, Tm \mid Tm\ Ty$$

$$Kd ::= Type$$

* We will use capital letters for types and lower case letters for term variables

β-reduction:

$$(\lambda x^A.\, M)\, N \to_\beta M[N/x]$$

$$(\Lambda X.\, M)\, A \to_\beta M$$

System F

start

$$\frac{}{\vdash \mathrm{Type} : \mathrm{Kind}}$$

intro

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$$

weaken

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma, x : B \vdash M : A}$$

$x \notin \Gamma$, $s \in \{\mathrm{Type}, \mathrm{Kind}\}$

λ-abs

$$\frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash A \to B : \mathrm{Type}}{\Gamma \vdash \lambda x^A . M : A \to B}$$

λ-app

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M N : B}$$

Λ-abs

$$\frac{\Gamma, A : \mathrm{Type} \vdash M : B \qquad \Gamma \vdash B : \mathrm{Type}}{\Gamma \vdash \Lambda A . M : \Pi A . B}$$

Λ-app

type substitution

$$\frac{\Gamma \vdash M : \Pi A . B \qquad \Gamma \vdash C : \mathrm{Type}}{\Gamma \vdash M C : B[C/A]}$$

A Note on Kinds

Kind is the type of Type

This allows us to introduce type variables and term variables with the same rule

$$\dfrac{\dfrac{\vdash \mathrm{Type} : \mathrm{Kind}}{X : \mathrm{Type} \vdash X : \mathrm{Type}}\ (\text{intro})}{X : \mathrm{Type},\ x : X \vdash x : X}\ (\text{intro})$$

same rule

A Note on Type Substitution

Since types have variables, we can substitute these variables with concrete types.

eg. $(A \to A)[Int / A] = Int \to Int$

The definition works as you might expect:

* with consideration to captured variables

$$Y[A/X] = \begin{cases} A & X = Y \\ Y & o w \end{cases}$$

$$(B \to C)[A/X] = B[A/X] \to C[A/X]$$

$$(\Pi Y . B)[A/X] = \begin{cases} \Pi Y . B & Y = X \\ \Pi Y . B[A/X] & Y \text{ not free in } A^{*} \end{cases}$$

Example

Give a derivation of

$$\boxed{\vdash \Lambda A. \Lambda B. \lambda f^{A \to B}. \lambda x^{A}. f x : \Pi A. \Pi B. (A \to B) \to A \to B}$$

Example with Type Application

Derive

$$\boxed{\begin{array}{l} \vdash \Lambda A.\ \lambda f^{\Pi B.\ B \to A}.\ f\,(A \to A)\,(f\,A) \\ \quad : \Pi A.\ (\Pi B. B \to A) \to A \end{array}}$$

Basic Meta-Theory

$\boxed{\text{Thinning}}$ $\Gamma, \Delta \vdash M : A$ & $\gamma \vdash \text{Type} : \text{Kind} \Rightarrow \Gamma, \gamma, \Delta \vdash M : A$

$\boxed{\text{Correctness}}$ $\Gamma \vdash M : A \Rightarrow \Gamma \vdash A : \text{Type}$

$\boxed{\text{Type Preservation}}$ $\Gamma \vdash M : A$ & $M \to_\beta N \Rightarrow \Gamma \vdash N : A$

$\boxed{\text{Uniqueness}}$ $\Gamma \vdash M : A$ & $\Gamma \vdash M : B \Rightarrow A =_\alpha B$ *

$\boxed{\text{Strong Normalization}}$ * $\Gamma \vdash M : A \Rightarrow M \twoheadrightarrow_\beta N$ where N is a normal form (N cannot be reduced)

* This is more difficult

* types have bound variables

Connection to Agda

System F is the fragment of Agda in which the only named parameters are types.

$$\Pi A.\ A \to B \equiv (A : \mathrm{Set}) \to A \to B$$

$$\Pi A.\ (\Pi B.\ B \to A) \to A \equiv (A : \mathrm{Set}) \to ((B : \mathrm{Set}) \to B \to A) \to A$$

If you can write an Agda function with this type then you can write a System F term with this type.

demo

(in Agda)

Ways of Defining Polymorphic Type Systems

$\boxed{\text{Domainful}}$ $\lambda 2$ λ-abstractions are labeled with types.

e.g. $\vdash \Lambda A. \Lambda B. \lambda x^A. \lambda y^B. x : \Pi A. \Pi B. A \to B \to A$

$\boxed{\text{Domain-free, Explicit } \Lambda\text{-abstraction}}$ $\underline{\lambda 2}$ unlabeled λ-abs.

e.g. $\vdash \Lambda. \Lambda. \lambda x. \lambda y. x : \Pi A. \Pi B. A \to B \to A$

$\boxed{\text{Domain-free, Implicit } \Lambda\text{-abstraction}}$ $\lambda 2c$ unlabeled λ-abs. no Λ-abs.

e.g. $\vdash \lambda x. \lambda y. x : \Pi A. \Pi B. A \to B \to A$
$\vdash \lambda x. \lambda y. x : \Pi A. A \to \Pi B. B \to A$

Computational Problems in Type Theory

$\boxed{\text{Type Checking}}$ Given Γ, M and A, determine if $\Gamma \vdash M : A$.

$\boxed{\text{Type Inference}}$ Given Γ and M, determine if there is an A s.t. $\Gamma \vdash M : A$

$\lambda 2$ vs. $\underline{\lambda 2}$ vs. $\lambda 2_c$

A computational problem is **decidable** if there is <u>any</u> algorithm that solves.

e.g. SAT is decidable, the halting problem is undecidable

	Decidable checking?	Decidable inference?
$\lambda 2$	YES	YES
$\underline{\lambda 2}$	NO	NO
$\lambda 2c$	NO	NO

Logic in System F

Recall: Logical Connectives in STLC

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \iota_1 M : A \vee B} \; (\vee\text{-}I_1) \qquad \frac{\Gamma \vdash N : B}{\Gamma \vdash \iota_2 N : A \vee B} \; (\vee\text{-}I_2)$$

$$\frac{\Gamma \vdash M : A \vee B \qquad \Gamma, x : A \vdash N_1 : C \qquad \Gamma, x : B : N_2 : C}{\Gamma \vdash \text{case } M \; N_1 \; N_2 : C} \; (\vee\text{-}E)$$

$$\boxed{\begin{array}{l} \text{case } (\iota_1 M) \; N_1 \; N_2 \to_\beta N_1[M/x] \\ \text{case } (\iota_2 M) \; N_1 \; N_2 \to_\beta N_2[M/x] \end{array}}$$

We can define logical connectives within STLC by including new constructors and rules.

Recall: Data Types in the λ-Calculus

$$\iota_1 \equiv \lambda x. \lambda f. \lambda g. f x$$

$$\iota_2 \equiv \lambda x. \lambda f. \lambda g. g x$$

$$\text{case} \equiv \lambda u. \lambda f. \lambda g. u f g \equiv \lambda u. u$$

We can define the computational parts within the λ-calculus.

Example Show

$$\text{case}\ (\iota_1 M)\ (\lambda x. N_1)\ (\lambda x. N_2) \twoheadrightarrow_\beta M[N_1/x]$$

Example (Continued)

$$\text{case } (\iota_1 M)\ (\lambda x.\ N_1)\ (\lambda x.\ N_2) \twoheadrightarrow_\beta M[N_1/x]$$

Lambda Encodings and Types

What would the type of ι_1 be in STLC?

$$\iota_1 \equiv \lambda x.\, \lambda f.\, \lambda g.\, f\, x$$

$$: A \to \underbrace{\boxed{(A \to C) \to (B \to C) \to C}}_{A \vee_C B}$$

We get almost an encoding of $A \vee B$ but it is specific to the "output type."

In System F, we can generalize over C.

Disjunction in System F

$$\boxed{A \vee B \equiv \Pi C.(A \to C) \to (B \to C) \to C}$$

$A \vee B$ is a thing which can take an $A \to C$ function and a $B \to C$ function and give you a C.

$$\iota_1 \equiv \Lambda A.\Lambda B.\Lambda C.\lambda x^A.\lambda f^{A \to C}.\lambda g^{B \to C}.f\,x$$

$$\iota_2 \equiv \Lambda A.\Lambda B.\Lambda C.\lambda x^B.\lambda f^{A \to C}.\lambda g^{B \to C}.g\,x$$

$$\text{case} \equiv \Lambda A.\Lambda B.\Lambda C.\lambda u^{A \vee B}.\lambda f^{A \to C}.\lambda g^{B \to C}.u\,C\,f\,g$$

compared to

$$\text{case} \equiv \qquad \lambda u.\lambda f.\lambda g.u\,f\,g$$

Example

Derive $\boxed{A : \mathrm{Type},\ B : \mathrm{Type} \vdash \iota_1\, A\, B : A \to A \vee B}^{*}$

* this is meta-syntax

Conjunction in System F

$$\boxed{A \wedge B \equiv \Pi C . (A \to B \to C) \to C}$$

$A \wedge B$ is a thing which, given a way to convert an A and a B to a C, it gives you a C.

$$\text{pair} \equiv \Lambda A . \Lambda B . \Lambda C . \lambda x^{A} . \lambda y^{B} . \lambda f^{A \to B \to C} . f\, x\, y$$

$$\pi_1 \equiv \Lambda A . \Lambda B . \lambda p^{A \wedge B} . p\, A\, (\lambda x^{A} . \lambda y^{B} . x)$$

$$\pi_2 \equiv \Lambda A . \Lambda B . \lambda p^{A \wedge B} . p\, B\, (\lambda x^{A} . \lambda y^{B} . y)$$

Example

Derive $\boxed{A : \mathrm{Type},\ B : \mathrm{Type} \vdash \pi_1\, A\, B : A \wedge B \to A.}$

Check that $\boxed{\pi_1\, A B\ (\mathrm{pair}\ A B\ M\ N) \twoheadrightarrow_\beta M}$

Negation in System F

$$\boxed{\bot \equiv \Pi C . C}$$

$\bot$ is a thing which can construct a term of any type

$$\neg A \equiv A \to \bot$$

$$\text{explode} \equiv \Pi A . \Pi B . f^{A \to \bot} . y^{A} . (f\ y)\ B$$

This is not necessary to define, but we have

$$A : \text{Type},\ B : \text{Type} \vdash \text{explode}\ A\ B : \neg A \to A \to B$$

The Point

We don't include logical connectives in System F because we can define them within the system.

Example

Write the law of excluded middle in System F.

Example

Prove $\neg A \vee B \to A \to B$ in System F.

Natural Numbers in System F

Recall Church Numerals in the λ-calculus:

$$\text{zero} \equiv \lambda f.\ \lambda x.\ x$$

$$\text{one} \equiv \lambda f.\ \lambda x.\ f\ (f\ x)$$

$$n \equiv \lambda f.\ \lambda x.\ f^{n}\ x$$

$$\text{succ} \equiv \lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x)$$

In System F:

$$\text{Nat} \equiv \Pi C.\ (C \to C) \to C \to C$$

$$\text{zero} \equiv \Lambda C.\ \lambda f^{C \to C}.\ \lambda x^{C}.\ x$$

$$\text{succ} \equiv \lambda n^{\text{Nat}}.\ \Lambda C.\ \lambda f^{C \to C}.\ \lambda x^{C}.$$