Type Systems

Lecture 6: Existentials, Data Abstraction, and Termination for System F

Neel Krishnaswami
University of Cambridge

Polymorphism and Data Abstraction

- So far, we have used polymorphism to model datatypes and genericity
- Reynolds's original motivation was to model *data abstraction*

An ML Module Signature

```
module type BOOL = sig
  type t
  val yes : t
  val no : t
  val choose
    : t -> 'a -> 'a -> 'a
end
```

- We introduce an abstract type `t`
- There are two values, `yes` and `no` of type `t`
- There is an operation `choose`, which takes a `t` and two values, and switches between them.

An Implementation

```
module M1 : BOOL = struct
  type t = unit option
  let yes = Some ()
  let no = None
  let choose v ifyes ifno =
    match v with
    | Some () -> ifyes
    | None -> ifno
end
```

- Implementation uses option type over unit
- There are two values, one for true and one for false
- `choose` implemented via pattern matching

Another Implementation

```
module M2 : BOOL = struct
  type t = int
  let yes = 1
  let no = 0
  let choose b ifyes ifno =
    if b = 1 then
      ifyes
    else
      ifno
end
```

- Implement booleans with integers
- Use 1 for true, 0 for false
- Why is this okay? (Many more integers than booleans, after all)

Yet Another Implementation

```
module M3 : BOOL = struct
  type t =
    {f : 'a. 'a -> 'a -> 'a}
  let yes =
    {f = fun a b -> a}
  let no =
    {f = fun a b -> b}
  let choose b ifyes ifno =
    b.f ifyes ifno
end
```

- Implement booleans with Church encoding (plus some Ocaml hacks)
- Is this really the same type as in the previous lecture?

A Common Pattern

- We have a signature — `BOOL` — with an abstract type in it
- We choose a concrete implementation of that abstract type
- We implement the other operations (`yes`, `no`, `choose`) of the interface in terms of that concrete representation
- Client code cannot identify the representation type because it sees an abstract type variable `t` rather than the representation

Abstract Data Types in System F

$$\begin{array}{llll} \text{Types} & A & ::= & \ldots \mid \exists \alpha.\, A \\ \text{Terms} & e & ::= & \ldots \mid \text{pack}_{\alpha.B}(A, e) \mid \text{let pack}(\alpha, x) = e \text{ in } e' \\ \text{Values} & v & ::= & \text{pack}_{\alpha.B}(A, v) \end{array}$$

$$\frac{\Theta, \alpha \vdash B \text{ type} \qquad \Theta \vdash A \text{ type} \qquad \Theta; \Gamma \vdash e : [A/\alpha]B}{\Theta; \Gamma \vdash \text{pack}_{\alpha.B}(A, e) : \exists \alpha.\, B} \; \exists \text{I}$$

$$\frac{\Theta; \Gamma \vdash e : \exists \alpha.\, A \qquad \Theta, \alpha; \Gamma, x : A \vdash e' : C \qquad \Theta \vdash C \text{ type}}{\Theta; \Gamma \vdash \text{let pack}(\alpha, x) = e \text{ in } e' : C} \; \exists \text{E}$$

Operational Semantics for Abstract Types

$$\frac{e \leadsto e'}{\mathsf{pack}_{\alpha.B}(A, e) \leadsto \mathsf{pack}_{\alpha.B}(A, e')}$$

$$\frac{e \leadsto e'}{\mathsf{let}\ \mathsf{pack}(\alpha, x) = e\ \mathsf{in}\ t \leadsto \mathsf{let}\ \mathsf{pack}(\alpha, x) = e'\ \mathsf{in}\ t}$$

$$\frac{}{\mathsf{let}\ \mathsf{pack}(\alpha, x) = \mathsf{pack}_{\alpha.B}(A, v)\ \mathsf{in}\ e \leadsto [A/\alpha, v/x]e}$$

Data Abstraction in System F

$$\frac{\Theta, \alpha \vdash B \text{ type} \quad \Theta \vdash A \text{ type} \quad \Theta; \Gamma \vdash e : [A/\alpha]B}{\Theta; \Gamma \vdash \text{pack}_{\alpha.B}(A, e) : \exists \alpha.\, B} \exists I$$

$$\frac{\Theta; \Gamma \vdash e : \exists \alpha.\, A \quad \Theta, \alpha; \Gamma, x : A \vdash e' : C \quad \Theta \vdash C \text{ type}}{\Theta; \Gamma \vdash \text{let pack}(\alpha, x) = e \text{ in } e' : C} \exists E$$

- We have a signature with an abstract type in it
- We choose a concrete implementation of that abstract type
- We implement the operations of the interface in terms of the concrete representation
- Client code sees an abstract type variable α rather than the representation

Abstract Types Have Existential Type

- No accident we write $\exists\alpha. B$ for abstract types!
- This is exactly the same thing as existential quantification in second-order logic
- Discovered by Mitchell and Plotkin in 1988 – *Abstract Types Have Existential Type*
- But Reynolds was thinking about data abstraction in 1976...?

A Church Encoding for Existential Types

$$\frac{\Theta, \alpha \vdash B \text{ type} \qquad \Theta \vdash A \text{ type} \qquad \Theta; \Gamma \vdash e : [A/\alpha]B}{\Theta; \Gamma \vdash \text{pack}_{\alpha.B}(A, e) : \exists \alpha.\, B} \exists\text{I}$$

$$\frac{\Theta; \Gamma \vdash e : \exists \alpha.\, B \qquad \Theta, \alpha; \Gamma, x : B \vdash e' : C \qquad \Theta \vdash C \text{ type}}{\Theta; \Gamma \vdash \text{let pack}(\alpha, x) = e \text{ in } e' : C} \exists\text{E}$$

Original	Encoding
$\exists \alpha.\, B$	$\forall \beta.\, (\forall \alpha.\, B \to \beta) \to \beta$
$\text{pack}_{\alpha.B}(A, e)$	$\Lambda \beta.\, \lambda k : \forall \alpha.\, B \to \beta.\, k\ A\ e$
$\text{let pack}(\alpha, x) = e \text{ in } e' : C$	$e\ C\ (\Lambda \alpha.\, \lambda x : B.\, e')$

Reduction of the Encoding

$$
\begin{aligned}
&\text{let pack}(\alpha, x) = \text{pack}_{\alpha.B}(A, e) \text{ in } e' : C \\
&\qquad = \text{pack}_{\alpha.B}(A, e)\ C\ \ (\Lambda\alpha.\, \lambda x : B.\, e') \\
&\qquad = (\Lambda\beta.\, \lambda k : \forall\alpha.\, B \to \beta.\, k\ A\ e)\ C\ \ (\Lambda\alpha.\, \lambda x : B.\, e') \\
&\qquad = (\lambda k : \forall\alpha.\, B \to C.\, k\ A\ e)\ (\Lambda\alpha.\, \lambda x : B.\, e') \\
&\qquad = (\Lambda\alpha.\, \lambda x : B.\, e')\ A\ e \\
&\qquad = (\lambda x : [A/\alpha]B.\, [A/\alpha]e')\ e \\
&\qquad = [e/x][A/\alpha]e'
\end{aligned}
$$

System F, The Girard-Reynolds Polymorphic Lambda Calculus

$$\begin{array}{llll} \text{Types} & A & ::= & \alpha \mid A \to B \mid \forall \alpha.\, A \\ \text{Terms} & e & ::= & x \mid \lambda x : A.\, e \mid e\, e \mid \Lambda \alpha.\, e \mid e\, A \\ \text{Values} & v & ::= & \lambda x : A.\, e \mid \Lambda \alpha.\, e \end{array}$$

$$\frac{e_0 \rightsquigarrow e_0'}{e_0\, e_1 \rightsquigarrow e_0'\, e_1}\ \textsc{CongFun} \qquad \frac{e_1 \rightsquigarrow e_1'}{v_0\, e_1 \rightsquigarrow v_0\, e_1'}\ \textsc{CongFunArg}$$

$$\frac{}{(\lambda x : A.\, e)\, v \rightsquigarrow [v/x]e}\ \textsc{FunEval}$$

$$\frac{e \rightsquigarrow e'}{e\, A \rightsquigarrow e'\, A}\ \textsc{CongForall} \qquad \frac{}{(\Lambda \alpha.\, e)\, A \rightsquigarrow [A/\alpha]e}\ \textsc{ForallEval}$$

Summary

So far:

1. We have seen System F and its basic properties
2. Sketched a proof of type safety
3. Saw that a variety of datatypes were encodable in it
4. We saw that even data abstraction was representable in it
5. We asserted, but did not prove, termination

Termination for System F

- We proved termination for the STLC by defining a *logical relation*
  - This was a family of relations
  - Relations defined by recursion on the structure of the type
  - Enforced a "hereditary termination" property
- Can we define a logical relation for System F?
  - How do we handle free type variables? (i.e., what's the interpretation of α?)
  - How do we handle quantifiers? (i.e., what's the interpretation of $\forall\alpha.\, A$?)

Semantic Types

A *semantic type* is a set of closed terms X such that:

- (Halting) If $e \in X$, then e halts (i.e. $e \leadsto^* v$ for some v).
- (Closure) If $e \leadsto e'$, then $e' \in X$ iff $e \in X$.

Idea:

- Build generic properties of the logical relation into the definition of a type.
- Use this to interpret variables!

Semantic Type Interpretations

$$\frac{\alpha \in \Theta}{\Theta \vdash \alpha \text{ type}} \qquad \frac{\Theta \vdash A \text{ type} \qquad \Theta \vdash B \text{ type}}{\Theta \vdash A \to B \text{ type}} \qquad \frac{\Theta, \alpha \vdash A \text{ type}}{\Theta \vdash \forall \alpha.\, A \text{ type}}$$

- We can interpret *type well-formedness derivations*
- Given a type variable context Θ, we define will define a variable interpretation θ as a map from $\mathrm{dom}(\Theta)$ to semantic types.
- Given a variable interpretation θ, we write $(\theta, X/\alpha)$ to mean extending θ with an interpretation X for a variable α.

Interpretation of Types

$$[\![-]\!] \in \text{WellFormedType} \to \text{VarInterpretation} \to \text{SemanticType}$$

$$
\begin{array}{lcl}
[\![\Theta \vdash \alpha \text{ type}]\!]\ \theta & = & \theta(\alpha) \\
[\![\Theta \vdash A \to B \text{ type}]\!]\ \theta & = & \left\{ e \;\middle|\; \begin{array}{l} e \text{ halts} \wedge \\ \forall e' \in [\![\Theta \vdash A \text{ type}]\!]\ \theta. \\ \qquad (e\, e') \in [\![\Theta \vdash B \text{ type}]\!]\ \theta \end{array} \right\} \\
[\![\Theta \vdash \forall\alpha.\, B \text{ type}]\!]\ \theta & = & \left\{ e \;\middle|\; \begin{array}{l} e \text{ halts} \wedge \\ \forall A \in \text{type}, X \in \text{SemType}. \\ \qquad (e\, A) \in [\![\Theta, \alpha \vdash B \text{ type}]\!]\ (\theta, X/\alpha) \end{array} \right\}
\end{array}
$$

Note the *lack* of a link between A and X in the $\forall\alpha.\, B$ case

Properties of the Interpretation

- **Closure:** If θ is an interpretation for Θ, then $[\![\Theta \vdash A \text{ type}]\!]\ \theta$ is a semantic type.
- **Exchange:** $[\![\Theta, \alpha, \beta, \Theta' \vdash A \text{ type}]\!] = [\![\Theta, \beta, \alpha, \Theta' \vdash A \text{ type}]\!]$
- **Weakening:** If $\Theta \vdash A$ type, then $[\![\Theta, \alpha \vdash A \text{ type}]\!]\ (\theta, X/\alpha) = [\![\Theta \vdash A \text{ type}]\!]\ \theta$.
- **Substitution:** If $\Theta \vdash A$ type and $\Theta, \alpha \vdash B$ type then $[\![\Theta \vdash [A/\alpha]B \text{ type}]\!]\ \theta = [\![\Theta, \alpha \vdash B \text{ type}]\!]\ (\theta, [\![\Theta \vdash A \text{ type}]\!]\ \theta)$

Each property is proved by induction on a type well-formedness derivation.

Closure: (one half of the) $\forall$ Case

Closure: If θ interprets Θ, then $[\![\Theta \vdash \forall\alpha.\, A \text{ type}]\!]\ \theta$ is a type.

Suffices to show: if $e \leadsto e'$, then $e \in [\![\Theta \vdash \forall\alpha.\, A \text{ type}]\!]\ \theta$ iff $e' \in [\![\Theta \vdash \forall\alpha.\, A \text{ type}]\!]\ \theta$.

0	$e \leadsto e'$	Assumption
1	$e' \in [\![\Theta \vdash \forall\alpha.\, A \text{ type}]\!]\ \theta$	Assumption
2	$\forall(C, X).\ e'\ C \in [\![\Theta, \alpha \vdash A \text{ type}]\!]\ (\theta, X/\alpha)$	Def.
3	Fix arbitrary (C, X)	
4	$\quad e'\ C \in [\![\Theta, \alpha \vdash A \text{ type}]\!]\ (\theta, X/\alpha)$	By 2
5	$\quad e\ C \leadsto e'\ C$	CongForall on 0
6	$\quad e\ C \in [\![\Theta, \alpha \vdash A \text{ type}]\!]\ (\theta, X/\alpha)$	Induction on 4,5
7	$\forall(C, X).\ e\ C \in [\![\Theta, \alpha \vdash A \text{ type}]\!]\ (\theta, X/\alpha)$	
8	$e \in [\![\Theta \vdash \forall\alpha.\, A \text{ type}]\!]\ \theta$	From 7

Substitution: (one half of) the $\forall$ case

$$[\![\Theta, \alpha \vdash \forall\beta.\, B \text{ type}]\!]\ (\theta, [\![\Theta \vdash A \text{ type}]\!]\ \theta) = [\![\Theta \vdash [A/\alpha](\forall\beta.\, B) \text{ type}]\!]\ \theta$$

1. We assume $e \in [\![\Theta, \alpha \vdash \forall\beta.\, B \text{ type}]\!]\ (\theta, [\![\Theta \vdash A \text{ type}]\!]\ \theta)$
2. We want to show: $e \in [\![\Theta \vdash [A/\alpha](\forall\beta.\, B) \text{ type}]\!]\ \theta$.
3. Expanding the definition of 1:
   $\forall (C, X).\ e\, C \in [\![\Theta, \alpha, \beta \vdash B \text{ type}]\!]\ (\theta, [\![\Theta \vdash A \text{ type}]\!]\ \theta, X/\beta)$.
4. For 2, it suffices to show:
   $\forall (C, X).\ e\, C \in [\![\Theta, \beta \vdash [A/\alpha](B) \text{ type}]\!]\ (\theta, X/\beta)$.
   - Fix (C, X)
   - So $e\, C \in [\![\Theta, \alpha, \beta \vdash B \text{ type}]\!]\ (\theta, [\![\Theta \vdash A \text{ type}]\!]\ \theta, X/\beta)$
   - Exchange: $e\, C \in [\![\Theta, \beta, \alpha \vdash B \text{ type}]\!]\ (\theta, X/\beta, [\![\Theta \vdash A \text{ type}]\!]\ \theta)$
   - Weaken:
     $e\, C \in [\![\Theta, \beta, \alpha \vdash B \text{ type}]\!]\ (\theta, X/\beta, [\![\Theta, \beta \vdash A \text{ type}]\!]\ (\theta, X/\beta))$
   - Induction: $e\, C \in [\![\Theta, \beta \vdash [A/\alpha]B \text{ type}]\!]\ (\theta, X/\beta)$

The Fundamental Lemma

If we have that

- $\overbrace{\alpha_1, \ldots, \alpha_k}^{\Theta}; \overbrace{x_1 : A_1, \ldots, x_n : A_n}^{\Gamma} \vdash e : B$
- $\Theta \vdash \Gamma$ ctx
- θ interprets Θ
- For each $x_i : A_i \in \Gamma$, we have $e_i \in [\![\Theta \vdash A_i \text{ type}]\!]\ \theta$

Then it follows that:

- $[C_1/\alpha_1, \ldots, C_k/\alpha_k][e_1/x_1, \ldots, e_n/x_n]e \in [\![\Theta \vdash B \text{ type}]\!]\ \theta$

Questions

1. Prove the other direction of the closure property for the $\Theta \vdash \forall\alpha.\, A$ type case.
2. Prove the other direction of the substitution property for the $\Theta \vdash \forall\alpha.\, A$ type case.
3. Prove the fundamental lemma for the forall-introduction case $\Theta; \Gamma \vdash \Lambda\alpha.\, e : \forall\alpha.\, A$.