

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

Types $A ::= \dots \mid \exists\alpha. A$

Terms $e ::= \dots \mid \text{pack}_{\alpha.B}(A, e) \mid \text{let pack}(\alpha, x) = e \text{ in } e'$

Values $v ::= \text{pack}_{\alpha.B}(A, v)$

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

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

Operational Semantics for Abstract Types

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

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

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

Data Abstraction in System F

$\Theta, \alpha \vdash B \text{ type}$
 $\Theta \vdash A \text{ type}$
 $\Theta; \Gamma \vdash e : [A/\alpha]B$

 $\Theta; \Gamma \vdash \text{pack}_{\alpha.B}(A, e) : \exists \alpha. B$ $\exists I$

$\Theta; \Gamma \vdash e : \exists \alpha. A$
 $\Theta, \alpha; \Gamma, x : A \vdash e' : C$
 $\Theta \vdash C \text{ type}$

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

- 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

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

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

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

Reduction of the Encoding

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

System F, The Girard-Reynolds Polymorphic Lambda Calculus

Types $A ::= \alpha \mid A \rightarrow B \mid \forall \alpha. A$

Terms $e ::= x \mid \lambda x : A. e \mid e e \mid \Lambda \alpha. e \mid e A$

Values $v ::= \lambda x : A. e \mid \Lambda \alpha. e$

$$\frac{e_0 \rightsquigarrow e'_0}{e_0 e_1 \rightsquigarrow e'_0 e_1} \text{ CONGFUN}$$

$$\frac{e_1 \rightsquigarrow e'_1}{v_0 e_1 \rightsquigarrow v_0 e'_1} \text{ CONGFUNARG}$$

$$\frac{}{(\lambda x : A. e) v \rightsquigarrow [v/x]e} \text{ FUNEVAL}$$

$$\frac{e \rightsquigarrow e'}{e A \rightsquigarrow e' A} \text{ CONGFORALL}$$

$$\frac{}{(\Lambda \alpha. e) A \rightsquigarrow [A/\alpha]e} \text{ FORALLEVAL}$$

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. $e \rightsquigarrow^* v$ for some v).
- (Closure) If $e \rightsquigarrow e'$, 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

$$\frac{\alpha \in \Theta}{\Theta \vdash \alpha \text{ type}}$$

$$\frac{\Theta \vdash A \text{ type} \quad \Theta \vdash B \text{ type}}{\Theta \vdash A \rightarrow B \text{ type}}$$

$$\frac{\Theta, \alpha \vdash A \text{ type}}{\Theta \vdash \forall \alpha. A \text{ type}}$$

- 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 $\text{dom}(\Theta)$ 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

$\llbracket - \rrbracket \in \text{WellFormedType} \rightarrow \text{VarInterpretation} \rightarrow \text{SemanticType}$

$$\llbracket \Theta \vdash \alpha \text{ type} \rrbracket \theta = \theta(\alpha)$$

$$\llbracket \Theta \vdash A \rightarrow B \text{ type} \rrbracket \theta = \left\{ e \left| \begin{array}{l} e \text{ halts } \wedge \\ \forall e' \in \llbracket \Theta \vdash A \text{ type} \rrbracket \theta. \\ (e e') \in \llbracket \Theta \vdash B \text{ type} \rrbracket \theta \end{array} \right. \right\}$$

$$\llbracket \Theta \vdash \forall \alpha. B \text{ type} \rrbracket \theta = \left\{ e \left| \begin{array}{l} e \text{ halts } \wedge \\ \forall A \in \text{type}, X \in \text{SemType}. \\ (e A) \in \llbracket \Theta, \alpha \vdash B \text{ type} \rrbracket (\theta, X/\alpha) \end{array} \right. \right\}$$

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 $\llbracket \Theta \vdash A \text{ type} \rrbracket \theta$ is a semantic type.
- **Exchange:** $\llbracket \Theta, \alpha, \beta, \Theta' \vdash A \text{ type} \rrbracket = \llbracket \Theta, \beta, \alpha, \Theta' \vdash A \text{ type} \rrbracket$
- **Weakening:** If $\Theta \vdash A \text{ type}$, then $\llbracket \Theta, \alpha \vdash A \text{ type} \rrbracket (\theta, X/\alpha) = \llbracket \Theta \vdash A \text{ type} \rrbracket \theta$.
- **Substitution:** If $\Theta \vdash A \text{ type}$ and $\Theta, \alpha \vdash B \text{ type}$ then $\llbracket \Theta \vdash [A/\alpha]B \text{ type} \rrbracket \theta = \llbracket \Theta, \alpha \vdash B \text{ type} \rrbracket (\theta, \llbracket \Theta \vdash A \text{ type} \rrbracket \theta)$

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 $\llbracket \Theta \vdash \forall \alpha. A \text{ type} \rrbracket \theta$ is a type.

Suffices to show: if $e \rightsquigarrow e'$, then $e \in \llbracket \Theta \vdash \forall \alpha. A \text{ type} \rrbracket \theta$ iff $e' \in \llbracket \Theta \vdash \forall \alpha. A \text{ type} \rrbracket \theta$.

| | | |
|---|--|------------------|
| 0 | $e \rightsquigarrow e'$ | Assumption |
| 1 | $e' \in \llbracket \Theta \vdash \forall \alpha. A \text{ type} \rrbracket \theta$ | Assumption |
| 2 | $\forall (C, X). e' C \in \llbracket \Theta, \alpha \vdash A \text{ type} \rrbracket (\theta, X/\alpha)$ | Def. |
| 3 | Fix (C, X) | |
| 4 | $e' C \in \llbracket \Theta, \alpha \vdash A \text{ type} \rrbracket (\theta, X/\alpha)$ | By 2 |
| 5 | $e C \rightsquigarrow e' C$ | CONGFORALL on 0 |
| 6 | $e C \in \llbracket \Theta, \alpha \vdash A \text{ type} \rrbracket (\theta, X/\alpha)$ | Induction on 4,5 |
| 7 | $\forall (C, X). e C \in \llbracket \Theta, \alpha \vdash A \text{ type} \rrbracket (\theta, X/\alpha)$ | |
| 8 | $e \in \llbracket \Theta \vdash \forall \alpha. A \text{ type} \rrbracket \theta$ | From 7 |

Substitution: (one half of) the \forall case

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

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

The Fundamental Lemma

If we have that

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

Then it follows that:

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

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 $\Theta; \Gamma \vdash \Lambda\alpha. e : \forall\alpha. A$.