Type Systems

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

Neel Krishnaswami
University of Cambridge
• So far, we have used polymorphism to model datatypes and genericity
• Reynolds’s original motivation was to model *data abstraction*
We introduce an abstract type \( t \)

- There are two values, \( \text{yes} \) and \( \text{no} \) of type \( t \)
- There is an operation \( \text{choose} \), which takes a \( t \) and two values, and switches between them.
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  \( A ::= \ldots \mid \exists \alpha.A \)

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

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

\[
\begin{array}{c}
\Theta, \alpha \vdash B \text{ type} \quad \Theta \vdash A \text{ type} \quad \Theta; \Gamma \vdash e : [A/\alpha]B \\
\hline
\Theta; \Gamma \vdash \text{pack}_{\alpha.B}(A, e) : \exists \alpha. B
\end{array}
\]

\[
\begin{array}{c}
\Theta; \Gamma \vdash e : \exists \alpha.A \quad \Theta, \alpha; \Gamma, x : A \vdash e' : C \quad \Theta \vdash C \text{ type} \\
\hline
\Theta; \Gamma \vdash \text{let pack}(\alpha, x) = e \text{ in } e' : C
\end{array}
\]
Operational Semantics for Abstract Types

\[
\begin{align*}
  &e \leadsto e' \\
  \frac{\text{pack}_{\alpha.\beta}(A, e) \leadsto \text{pack}_{\alpha.\beta}(A, e')} \\
  &e \leadsto e' \\
  \frac{\text{let pack}(\alpha, x) = e \text{ in } t \leadsto \text{let pack}(\alpha, x) = e' \text{ in } t} \\
  \frac{\text{let pack}(\alpha, x) = \text{pack}_{\alpha.\beta}(A, v) \text{ in } e \leadsto [A/\alpha, v/x]e}
\end{align*}
\]
Data Abstraction in System F

- 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

<table>
<thead>
<tr>
<th>Original</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \exists \alpha. \ B )</td>
<td>( \forall \beta. (\forall \alpha. B \rightarrow \beta) \rightarrow \beta )</td>
</tr>
<tr>
<td>( \text{pack}_{\alpha. B}(A, e) )</td>
<td>( \Lambda \beta. \lambda k : \forall \alpha. B \rightarrow \beta. k A e )</td>
</tr>
<tr>
<td>( \text{let pack}(\alpha, x) = e \text{ in } e' : C )</td>
<td>( e \ C \ (\Lambda \alpha. \lambda x : B. \ e') )</td>
</tr>
</tbody>
</table>
let pack(\(\alpha, x\)) = pack_{\alpha,B}(A, e) in e' : C
= 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'
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
\[ v ::= \lambda x : A. e \mid \Lambda \alpha. e \]

\[
\frac{e_0 \rightsquigarrow e'_0}{e_0 e_1 \rightsquigarrow e'_0 e_1} \quad \text{CONGFUN} \quad \frac{e_1 \rightsquigarrow e'_1}{v_0 e_1 \rightsquigarrow v_0 e'_1} \quad \text{CONGFUNARG}
\]

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

\[
\frac{e \rightsquigarrow e'}{e A \rightsquigarrow e' A} \quad \text{CONGFORALL} \quad \frac{(\Lambda \alpha. e) A \rightsquigarrow [A/\alpha]e}{\text{FORALLEVAL}}
\]
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 $\alpha$?)
  • How do we handle quantifiers? (i.e., what’s the interpretation of $\forall\alpha. A$?)
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

\[ \alpha \in \Theta \quad \implies \quad \Theta \vdash \alpha \text{ type} \]

\[ \Theta \vdash A \text{ type} \quad \Theta \vdash B \text{ type} \quad \implies \quad \Theta \vdash A \rightarrow B \text{ type} \]

\[ \Theta, \alpha \vdash A \text{ type} \quad \implies \quad \Theta \vdash \forall \alpha. A \text{ type} \]

• We can interpret type well-formedness derivations
• Given a type variable context \( \Theta \), we define will define a variable interpretation \( \theta \) as a map from \( \text{dom}(\Theta) \) to semantic types.
• Given a variable interpretation \( \theta \), we write \((\theta, X/\alpha)\) to mean extending \( \theta \) with an interpretation \( X \) for a variable \( \alpha \).
Interpretation of Types

\[ \begin{align*}
[\neg] & \in \text{WellFormedType} \rightarrow \text{VarInterpretation} \rightarrow \text{SemanticType} \\
[\Theta \vdash \alpha \text{ type}] \; \theta & = \; \theta(\alpha) \\
[\Theta \vdash A \rightarrow B \text{ type}] \; \theta & = \begin{cases} 
  e & \text{\(e\) halts} \\
  \forall e' \in [\Theta \vdash A \text{ type}] \; \theta. \\
  (e \; e') \in [\Theta \vdash B \text{ type}] \; \theta & \text{\(e\) halts} \\
\end{cases} \\
[\Theta \vdash \forall \alpha. \; B \text{ type}] \; \theta & = \begin{cases} 
  e & \text{\(e\) halts} \\
  \forall A \in \text{type}, X \in \text{SemType}.
  (e \; A) \in [\Theta, \alpha \vdash B \text{ type}] \; (\theta, X/\alpha) & \text{\(e\) halts} \\
\end{cases}
\end{align*} \]

Note the lack of a link between \(A\) and \(X\) in the \(\forall \alpha. \; B\) case.
Properties of the Interpretation

- **Closure:** If $\theta$ is an interpretation for $\Theta$, 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 $\Theta \vdash A \text{ type}$, then $[\Theta, \alpha \vdash A \text{ type}] (\theta, X/\alpha) = [\Theta \vdash A \text{ type}] \theta$.
- **Substitution:** If $\Theta \vdash A \text{ type}$ and $\Theta, \alpha \vdash B \text{ type}$ then $[\Theta \vdash [A/\alpha]B \text{ type}] \theta = [\Theta, \alpha \vdash B \text{ type}] (\theta, [\Theta \vdash A \text{ type}] \theta/\alpha)$

Each property is proved by induction on a type well-formedness derivation.
Closure: (one half of the) ∀ Case

**Closure:** If \( \theta \) interprets \( \Theta \), 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 \( e' \in [\Theta \vdash \forall \alpha. A \text{ type}] \) \( \theta \).

1. \( e \leadsto e' \)  
   Assumption
2. \( e' \in [\Theta \vdash \forall \alpha. A \text{ type}] \) \( \theta \)  
   Assumption
3. \( \forall (C, X). \ e' C \in [\Theta, \alpha \vdash A \text{ type}] (\theta, X/\alpha) \)  
   Def.
4. \( \text{Fix arbitrary } (C, X) \)  
5. \( e C \leadsto e' C \)  
   CONGFORALL on 0
6. \( e C \in [\Theta, \alpha \vdash A \text{ type}] (\theta, X/\alpha) \)  
   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 ∀ 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

\[ \Theta \vdash \Gamma \]
\[ \alpha_1, \ldots, \alpha_k; x_1 : A_1, \ldots, x_n : A_n \vdash e : B \]
\[ \Theta \vdash \Gamma \text{ ctx} \]
\[ \theta \text{ interprets } \Theta \]
\[ \text{For each } x_i : A_i \in \Gamma, \text{ we have } e_i \in \llbracket \Theta \vdash A_i \text{ type} \rrbracket \theta \]

Then it follows that:

\[ [C_1/\alpha_1, \ldots, C_k/\alpha_k][e_1/x_1, \ldots, e_n/x_n]e \in \llbracket \Theta \vdash B \text{ type} \rrbracket \theta \]
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 $\Theta; \Gamma \vdash \Lambda \alpha. e : \forall \alpha. A$. 