Existential Types

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong
Peking University, Spring Term
About existential types

• System F: universal types
  • $\forall X. X \rightarrow T$

• Can we change the quantifier to form a new type?
  • $\exists X. X \rightarrow T$

• Existential types: 10 years ago
  • Almost only in theory
  • Used to understand encapsulation

• Existential types: now
  • Used in mainstream languages such as Java, Scala, Haskell
Existential Types in Java

• Designed by Martin Odersky
• How to print all elements in a generic collection in Java?
  
  ```java
  void printCollection(Collection<Object> c) {
    for (Object e : c) {
      System.out.println(e);
    }
  }
  ```
Existential Types in Java

• Designed by Martin Odersky

• How to print all elements in a generic collection in Java?
  
  ```java
  void printCollection(Collection<Object> c) {
    for (Object e : c) {
      System.out.println(e);
    }
  }
  ```

• Problem: Collection<Integer> cannot be passed.
Existential Types in Java

• Designed by Martin Odersky

• How to print all elements in a generic collection in Java?

```java
void printCollection(Collection<? ?> c) {
    for (Object e : c) {
        System.out.println(e);
    }
}
```

• ? stands for some unknown types
Existential Types in Java

• The previous example is used in almost every Java tutorial about wildcards
• Is there a problem?
Existential Types in Java

• The previous example is used in almost every Java tutorial about wildcards
• Is there a problem?
• This following code implements the same function in a more type-safe manner

```java
<T> void printCollection(Collection<T> c) {
    for (T e : c) {
        System.out.println(e);
    }
}
```
Existential Types in Java

• The use of wildcards is for encapsulation

• Will the following code compile?
  
  ```java
  public class A {
    private class B {...}
    public Collection<B> getInternalList() {...}
  }
  ```
Existential Types in Java

• The use of wildcards is for encapsulation

• Will the following code compile?
  
  ```java
  public class A {
    private class B {…}
    public Collection<B> getInternalList() {...}
  }
  ```

• Yes (weird Java design), but is not useful.
  
  ```java
  Collection<B> bs = new A().getInternalList();
  // Compilation error
  ```
Existential Types in Java

• The use of wildcards is for encapsulation

• Using Wildcards
  public class A {
    private class B {...
    public Collection<?> getInternalList() {...
  }
  Collection<?> bs = new A().getInternalList();
Existential Types

• Theoretical Intuition: Can we change the universal quantifier in $\forall X. T$ into existential quantifier $\exists X. T$?

• $\forall X. T$: for any type $X$, $T$ is a type

• $\exists X. T$: there exists some type $X$, $T$ is a type
  • Collection<?> is a type Collection<X> for some type $X$
  • You should not care about the value of $X$
A Problem in Java

• Rotate a list by one
  • List<?> l = getSomeList();
  • l.add(l.remove(0))  // compilation error

• Can we improve the design?
  • Give concrete name to “?”
Existential Type by Example

\[ p = \{ \text{Nat}, \{ a=0, f=\lambda x: \text{Nat}. \text{succ}(x) \} \} \text{ as } \{ \exists X, \{ a:X, f:X\rightarrow \text{Nat} \} \}; \]

\[ p : \{ \exists X, \{ a:X, f:X\rightarrow \text{Nat} \} \} \]

\[ \text{let } \{ X,x \}=p \text{ in } (x.f \ x.a); \]

\[ 1 : \text{Nat} \]

\[ \text{let } \{ X,x \}=p \text{ in } (\lambda y:X. \ x.f \ y) \ x.a; \]

\[ 1 : \text{Nat} \]

\[ \text{let } \{ X,x \}=p \text{ in } \text{succ}(x.a); \]

\[ \text{Error: argument of succ is not a number} \]

\[ \text{let } \{ X,x \}=p \text{ in } x.a; \]

\[ \text{Error: Scoping error!} \]
Exercise: are the following terms useful?

\[ p_6 = \{ \text{Nat}, \{ a=0, f=\lambda x:\text{Nat}. \text{succ}(x) \} \} \text{ as } \{ \exists X, \{ a:X, f:X \to X \} \}; \]

- \( p_6 : \{ \exists X, \{ a:X, f:X \to X \} \} \)
  
  Can never do anything with the result

\[ p_7 = \{ \text{Nat}, \{ a=0, f=\lambda x:\text{Nat}. \text{succ}(x) \} \} \text{ as } \{ \exists X, \{ a:X, f:\text{Nat} \to X \} \}; \]

- \( p_7 : \{ \exists X, \{ a:X, f:\text{Nat} \to X \} \} \)
  
  Same as above. Also cannot pass \( a \) to \( f \).

\[ p_8 = \{ \text{Nat}, \{ a=0, f=\lambda x:\text{Nat}. \text{succ}(x) \} \} \text{ as } \{ \exists X, \{ a:\text{Nat}, f:\text{Nat} \to \text{Nat} \} \}; \]

- \( p_8 : \{ \exists X, \{ a:\text{Nat}, f:\text{Nat} \to \text{Nat} \} \} \)
  
  Does not encapsulate anything
Defining Existential Type

New syntactic forms
\[ t ::= \ldots \]
\[ \{\ast T, t\} \text{ as } T \]
\[ \text{let } \{X, x\} = t \text{ in } t \]
\[ \nu ::= \ldots \]
\[ \{\ast T, \nu\} \text{ as } T \]
\[ T ::= \ldots \]
\[ \{\exists X, T\} \]

New typing rules
\[ \Gamma \vdash t : T \]
\[ \Gamma \vdash t_2 : [X \mapsto U]T_2 \]
\[ \Gamma \vdash \{\ast U, t_2\} \text{ as } \{\exists X, T_2\} \]
\[ \vdash \{\ast T_{11}, t_{12}\} \text{ as } T_1 \]
\[ \vdash \{\ast T_{11}, t'_{12}\} \text{ as } T_1 \]
\[ \Gamma \vdash \text{let } \{X, x\} = t_1 \text{ in } t_2 \]
\[ \vdash \text{let } \{X, x\} = t'_1 \text{ in } t_2 \]

New evaluation rules
\[ t \rightarrow t' \]
\[ \text{let } \{X, x\} = \{\ast T_{11}, \nu_{12}\} \text{ as } T_1 \text{ in } t_2 \]
\[ \rightarrow [X \mapsto T_{11}][x \mapsto \nu_{12}]t_2 \]

Extends System F (23-1)

(E-PACK)

(E-UNPACK)

(T-PACK)

(T-UNPACK)

(E-UNPACKPACK)

Figure 24-1: Existential types
Review: Abstract Data Type

• CounterRep = \{ x: \text{Ref Nat} \}
• newCounter =
  \lambda_: \text{Unit}. \text{let } r = \{ x = \text{ref 1} \} \text{ in }
  \{ \text{get} = \lambda_: \text{Unit}. ! (r.x),
  \text{inc} = \lambda_: \text{Unit}. r.x: = \text{succ}(! (r.x)) \} ;

Can we turn it into an immutable object?
Immutable Counter

• CounterRep = \{x: Nat\}
• newCounterImpl = \lambda r: CounterRep. 
  \{ get = \lambda _: Unit. r.x, 
    inc = \lambda _: CounterRep. \{x = r.x + 1\}\};
• newCounter = newCounterImpl \{x=1\}

But CounterRep is not encapsulated for the client.
Encoding Abstract Data Types

counterADT =
{\*{x: Nat},
 {new = \{x=1\},
  get = \lambda i:{x: Nat}. i.x,
  inc = \lambda i:{x: Nat}. \{x=succ(i.x)\}\}}
as {\exists Counter,
 {new: Counter, get: Counter→Nat, inc: Counter→Counter}};

> counterADT : {\exists Counter,
 {new: Counter, get: Counter→Nat, inc: Counter→Counter}}

let {Counter, counter} = counterADT in
counter.get (counter.inc counter.new);

> 2 : Nat
Encoding Objects

• Read the book
Encoding existential types in universal types

\[ p4 = \{ \forall \text{Nat}, \{ a=0, f=\lambda x:\text{Nat}. \text{succ}(x) \} \} \text{ as } \{ \exists X, \{ a:X, f:X \rightarrow \text{Nat} \} \}; \]

- \( p4 : \{ \exists X, \{ a:X, f:X \rightarrow \text{Nat} \} \} \)

let \( \{ X, x \} = p4 \) in \( (x.f \ x.a) \);

- \( 1 : \text{Nat} \)

\[ p4' = \lambda Y. \ \lambda g:(\forall X.\{ a:X, f:X \rightarrow \text{Nat} \} \rightarrow Y). \]

\[ g \ [\text{Nat}] \ \{ a=0, f=\lambda x:\text{Nat}. \text{succ}(x) \} \]

\[ p4' \ [\text{Nat}] \ (\lambda X. \ \lambda x: \{ a:X, f:X \rightarrow \text{Nat} \}. \ (x.f \ x.a)) \]
Encoding existential types in universal types

\[ \{\exists X, T\} \; \overset{\text{def}}{=} \; \forall Y. (\forall X. T \to Y) \to Y. \]

\[ \{\ast S, t\} \text{ as } \{\exists X, T\} \; \overset{\text{def}}{=} \; \lambda Y. \lambda f:(\forall X. T \to Y). f [{S}] t \]

\[
\Gamma \vdash t_1 : \{\exists X, T_{12}\}
\]
\[
\Gamma, X, x:T_{12} \vdash t_2 : T_2
\]
\[
\Gamma \vdash \text{let } \{X, x\}=t_1 \text{ in } t_2 : T_2
\]

\[
\text{let } \{X, x\}=t_1 \text{ in } t_2 \; \overset{\text{def}}{=} \; t_1 [T_2] (\lambda X. \lambda x:T_{12}. t_2).
\]