Existential Types

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong
Peking University, Spring Term, 2016
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?
  public class A {
      private class B {...}
      public Collection<B> getInternalList() {...}
  }

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

• The use of wildcards is for encapsulation

• Using Wildcards
  
  ```java
  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

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

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

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

\[ 1 : \text{Nat} \]

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

\[ 1 : \text{Nat} \]

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

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

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

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

\[\begin{align*}
p_6 &= \{\text{Nat}, \ a=0, \ f=\lambda x:\text{Nat}. \ \text{succ}(x)\}\ \text{as} \ \{\exists X, \ a:X, \ f:X\rightarrow X\}\; ; \\
&= \{\exists X, \ a:X, f:X\rightarrow X\} \\
p_7 &= \{\text{Nat}, \ a=0, \ f=\lambda x:\text{Nat}. \ \text{succ}(x)\}\ \text{as} \ \{\exists X, \ a:X, f:\text{Nat}\rightarrow X\}\; ; \\
&= \{\exists X, \ a:X, f:\text{Nat}\rightarrow X\} \\
p_8 &= \{\text{Nat}, \ a=0, \ f=\lambda x:\text{Nat}. \ \text{succ}(x)\}\ \text{as} \ \{\exists X, \ a:\text{Nat}, f:\text{Nat}\rightarrow \text{Nat}\}\; ; \\
&= \{\exists X, \ a:\text{Nat}, f:\text{Nat}\rightarrow \text{Nat}\}
\end{align*}\]

- Can never do anything with the result
- Same as above
- Does not encapsulate anything
Defining Existential Type

New syntactic forms

$t ::= \ldots$

- $\{\exists T, t\} as T$
- $let \{X, x\} = t in t$

$v ::= \ldots$

- $\{\exists T, v\} as T$

$T ::= \ldots$

- $\{\exists X, T\}$

New typing rules

1. $\Gamma \vdash t : T$
2. $\Gamma \vdash t_2 : [X \rightarrow U]T_2$
3. $\Gamma \vdash \{\exists U, t_2\} as \{\exists X, T_2\}$
4. $\vdash \{\exists X, T_{12}\}$
5. $\Gamma, X, x : T_{12} \vdash t_2 : T_2$
6. $\Gamma \vdash let \{X, x\} = t_1 in t_2 : T_2$

New evaluation rules

1. $t \rightarrow t'$
2. $\Gamma \vdash t_2 : [X \rightarrow U]T_2$
3. $\Gamma \vdash \{\exists U, t_2\} as \{\exists X, T_2\}$
4. $\vdash \{\exists X, T_{12}\}$
5. $\Gamma, X, x : T_{12} \vdash t_2 : T_2$
6. $\Gamma \vdash let \{X, x\} = t_1 in t_2 : T_2$

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

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

Can we turn it into a immutable object?
Immutable Counter

- `CounterRep = \{x: Nat\}`
- `newCounter =`
  \[
  \lambda_\_ : \text{Unit}. \quad \text{let } r = \{x = 1\} \quad \text{in}
  \quad \{ \text{get} = \lambda_\_ : \text{Unit}. \quad r.x, \quad
  \text{inc} = \lambda_\_: \text{CounterRep}. \quad r \};
  \]

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

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

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

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

- \[ \text{p4} : \{ \exists X, \{ a:X, f:X \to \text{Nat} \} \} \]

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

- \[ 1 : \text{Nat} \]

\[ \text{p4}' = \lambda Y. \ \lambda g:(\forall X.\{ a:X, f:X \to \text{Nat} \} \to Y). \]
\[
\text{g [Nat]} \ \{ a=0, f=\lambda x: \text{Nat}. \ \text{succ}(x) \} \]

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

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

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

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

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