Design Principles of Programming Languages

Existential Types

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong
Peking University, Spring Term, 2014
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

  \[
  \texttt{<T> void printCollection(Collection<T> c) \{ }
  \texttt{for (T e : c) \{ System.out.println(e); \}}
  \texttt{\}}
  \]
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

• Concrete name needs to be given to “?”
Existential Type by Example

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

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

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

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

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

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

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

- Error: argument of succ is not a number

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

- Error: Scoping error!
Exercise: are the following terms useful?

\[
p6 = \{*\text{Nat}, \{a=0, f=\lambda x: \text{Nat}. \text{succ}(x)\}\} \text{ as } \exists X, \{a:X, f:X\rightarrow X\};
\]

\[
p6 : \exists X, \{a:X, f:X\rightarrow X\}
\]

\[
p7 = \{*\text{Nat}, \{a=0, f=\lambda x: \text{Nat}. \text{succ}(x)\}\} \text{ as } \exists X, \{a:X, f: \text{Nat}\rightarrow X\};
\]

\[
p7 : \exists X, \{a:X, f: \text{Nat}\rightarrow X\}
\]

\[
p8 = \{*\text{Nat}, \{a=0, f=\lambda x: \text{Nat}. \text{succ}(x)\}\} \text{ as } \exists X, \{a:\text{Nat}, f: \text{Nat}\rightarrow \text{Nat}\};
\]

\[
p8 : \exists X, \{a:\text{Nat}, f: \text{Nat}\rightarrow \text{Nat}\}
\]
Defining Existential Type

New syntactic forms
- $t ::= ...$
  - $\{\exists T, t\}$ as $T$
  - let $\{X, x\} = t$ in $t$

- $v ::= ...$
  - $\{\exists T, v\}$ as $T$

- $T ::= ...$
  - $\{\exists X, T\}$

New typing rules
- $\Gamma \vdash t : T$
  - $\Gamma \vdash t_2 : [X \mapsto U] T_2$
  - $\Gamma \vdash \{\exists U, t_2\} = \{\exists X, T_2\}$

- $\Gamma \vdash \{\exists X, T_{12}\}$
  - $\Gamma, X, x : T_{12} \vdash t_2 : T_2$
  - $\Gamma \vdash$ let $\{X, x\} = t_1$ in $t_2 : T_2$

New evaluation rules
- $t \rightarrow t'$
  - let $\{X, x\} = (\{\exists T_{11}, v_{12}\}$ as $T_1$) in $t_2$
  - $\rightarrow [X \mapsto T_{11}][x \mapsto v_{12}] t_2$

(E-UNPACKPACK)

(E-PACK)

(E-UNPACK)

(Figure 24-1: Existential types)
Encoding Abstract Data Types

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

› counterADT : {∃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

\[ p_4 = \{ \forall \text{Nat}, \{a=0, f=\lambda x: \text{Nat}. \text{succ}(x)\}\} \text{ as } \{ \exists X, \{a:X, f:X \rightarrow \text{Nat}\}\}; \]

\[ p_4 : \{ \exists X, \{a:X, f:X \rightarrow \text{Nat}\}\} \]

\[ \text{let } \{X, x\} = p_4 \text{ in } (x.f \ x.a); \]

\[ 1 : \text{Nat} \]

\[ p_4' = \lambda Y. \lambda g : (\forall X. \{a:X, f:X \rightarrow \text{Nat}\} \rightarrow Y). \]

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

\[ p_4' [\text{Nat}] (\lambda X. \lambda x: \{a:X, f:X \rightarrow \text{Nat}\}. (x.f \ x.a)) \]
Encoding existential types in universal types

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

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

\begin{align*}
\Gamma \vdash t_1 : \{\exists X, T_{12}\} \\
\Gamma, X, x : T_{12} \vdash t_2 : T_2 \\
\frac{}{\Gamma \vdash \text{let } \{X, x\}=t_1 \text{ in } t_2 : T_2}
\end{align*}

\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).
Homework

• Encoding counterADT in universal types
  • Your code needs to be implemented in fullpoly
  • Your code should contain test cases that invoke “inc” several times and then invoke “get” to check its value
  • Please submit electronically