Design Principles of Programming Languages

Existential Types

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong
Peking University, Spring Term, 2016

About existential types

- System F: universal types
  - $\forall X. X \to T$
- Can we change the quantifier to form a new type?
  - $\exists X. X \to T$
- 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?

```
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?

```
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?

```
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

```
<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?

```
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

```
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  = {*Nat, {a=0, f=λx:Nat. succ(x)}} as {∃X, {a:X, f:X→Nat}};
▸ p  : {∃X, {a:X,f:X→Nat}}

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

▸ 1 : Nat

  let {X,x}=p  in (λy:X. x.f y) x.a;
▸ 1 : Nat

  let {X,x}=p  in succ(x.a);
▸ Error: argument of succ is not a number

   let {X,x}=p in x.a;
▸ Error: Scoping error!
```

Exercise: are the following terms useful?

```
p6 = {*Nat, {a=0, f=λx:Nat. succ(x)}} as {∃X, {a:X, f:X→X}};
```

▸ `p6 : {∃X, {a:X,f:X→X}}` Can never do anything with the result

```
p7 = {*Nat, {a=0, f=λx:Nat. succ(x)}} as {∃X, {a:X, f:Nat→X}};
```

▸ `p7 : {∃X, {a:X,f:Nat→X}}` Same as above

```
p8 = {*Nat, {a=0, f=λx:Nat. succ(x)}} as {∃X, {a:Nat, f:Nat→Nat}};
```

▸ `p8 : {∃X, {a:Nat,f:Nat→Nat}}` Does not encapsulate anything

Defining Existential Type

$\rightarrow$ $\forall$ $\exists$ *Extends System F (23-1)*

New syntactic forms

$\mathtt{t} ::= \ldots$		*terms:*
	$\{{*}\mathtt{T},\mathtt{t}\}\ \mathtt{as}\ \mathtt{T}$	*packing*
	$\mathtt{let}\ \{\mathtt{X},\mathtt{x}\}\mathtt{=t}\ \mathtt{in}\ \mathtt{t}$	*unpacking*
$\mathtt{v} ::= \ldots$		*values:*
	$\{{*}\mathtt{T},\mathtt{v}\}\ \mathtt{as}\ \mathtt{T}$	*package value*
$\mathtt{T} ::= \ldots$		*types:*
	$\{\exists\mathtt{X},\mathtt{T}\}$	*existential type*

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\mathtt{let}\ \{\mathtt{X},\mathtt{x}\}\mathtt{=}(\{{*}\mathtt{T}_{11},\mathtt{v}_{12}\}\ \mathtt{as}\ \mathtt{T}_1)\ \mathtt{in}\ \mathtt{t}_2 \longrightarrow [\mathtt{X} \mapsto \mathtt{T}_{11}][\mathtt{x} \mapsto \mathtt{v}_{12}]\mathtt{t}_2 \qquad \text{(E-UnpackPack)}$$

$$\frac{\mathtt{t}_{12} \longrightarrow \mathtt{t}'_{12}}{\{{*}\mathtt{T}_{11},\mathtt{t}_{12}\}\ \mathtt{as}\ \mathtt{T}_1 \longrightarrow \{{*}\mathtt{T}_{11},\mathtt{t}'_{12}\}\ \mathtt{as}\ \mathtt{T}_1} \qquad \text{(E-Pack)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}'_1}{\mathtt{let}\ \{\mathtt{X},\mathtt{x}\}\mathtt{=t}_1\ \mathtt{in}\ \mathtt{t}_2 \longrightarrow \mathtt{let}\ \{\mathtt{X},\mathtt{x}\}\mathtt{=t}'_1\ \mathtt{in}\ \mathtt{t}_2} \qquad \text{(E-Unpack)}$$

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\Gamma \vdash \mathtt{t}_2 : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{T}_2}{\Gamma \vdash \{{*}\mathtt{U},\mathtt{t}_2\}\ \mathtt{as}\ \{\exists\mathtt{X},\mathtt{T}_2\} : \{\exists\mathtt{X},\mathtt{T}_2\}} \qquad \text{(T-Pack)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \{\exists\mathtt{X},\mathtt{T}_{12}\} \qquad \Gamma,\mathtt{X},\mathtt{x}{:}\mathtt{T}_{12} \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \mathtt{let}\ \{\mathtt{X},\mathtt{x}\}\mathtt{=t}_1\ \mathtt{in}\ \mathtt{t}_2 : \mathtt{T}_2} \qquad \text{(T-Unpack)}$$

Figure 24-1: Existential types

Review: Abstract Data Type

- $\text{CounterRep} = \{x: \text{Ref Nat}\}$
- $\text{newCounter} =$

$$\lambda\_: \text{Unit}.\ \text{let } r = \{x = \text{ref } 1\} \text{ in}$$
$$\{\text{get} = \lambda\_: \text{Unit}.\ !(r.x),$$
$$\text{inc} = \lambda\_: \text{Unit}.\ r.x := \text{succ}(!(r.x))\};$$

Can we turn it into a immutable object?

Immutable Counter

- $\text{CounterRep} = \{x: \text{Nat}\}$
- $\text{newCounter} =$

$$
\begin{aligned}
&\lambda\_: \text{Unit}.\ \text{let } r = \{x = 1\} \text{ in} \\
&\quad \{\text{get} = \lambda\_: \text{Unit}.\ r.x, \\
&\quad\ \ \text{inc} = \lambda\_: \text{CounterRep}.\ r\};
\end{aligned}
$$

But ConterRep is not encapsulated for the client.

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

```
p4 = {*Nat, {a=0, f=λx:Nat. succ(x)}} as {∃X, {a:X, f:X→Nat}};
```

```
▸ p4 : {∃X, {a:X,f:X→Nat}}
```

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

```
▸ 1 : Nat
```

```
p4' = λY. λg:(∀X.{a:X, f:X→Nat}→Y).
        g [Nat] {a=0, f=λx:Nat. succ(x)}
p4' [Nat] (λX. λx: {a:X, f:X→Nat}. (x.f x.a))
```

Encoding existential types in universal types

$$\{\exists X, T\} \stackrel{\text{def}}{=} \forall Y.\ (\forall X.\ T \to Y) \to Y.$$

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

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

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