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
  - $\forall X. X \to T$
- 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
- Concrete name needs to be given 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}}

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

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

Defining Existential Type

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

New syntactic forms

t	::=	...	*terms:*
		{*T,t} as T	*packing*
		let {X,x}=t in t	*unpacking*
v	::=	...	*values:*
		{*T,v} as T	*package value*
T	::=	...	*types:*
		{∃X,T}	*existential type*

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

$$\texttt{let }\{X,x\}=(\{{*}T_{11},v_{12}\}\texttt{ as }T_1)\texttt{ in }t_2 \longrightarrow [X \mapsto T_{11}][x \mapsto v_{12}]t_2 \quad \text{(E-UnpackPack)}$$

$$\frac{t_{12} \longrightarrow t_{12}'}{\{{*}T_{11},t_{12}\}\texttt{ as }T_1 \longrightarrow \{{*}T_{11},t_{12}'\}\texttt{ as }T_1} \quad \text{(E-Pack)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{let }\{X,x\}=t_1\texttt{ in }t_2 \longrightarrow \texttt{let }\{X,x\}=t_1'\texttt{ in }t_2} \quad \text{(E-Unpack)}$$

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

$$\frac{\Gamma \vdash t_2 : [X \mapsto U]T_2}{\Gamma \vdash \{{*}U,t_2\}\texttt{ as }\{\exists X,T_2\} : \{\exists X,T_2\}} \quad \text{(T-Pack)}$$

$$\frac{\Gamma \vdash t_1 : \{\exists X,T_{12}\} \qquad \Gamma, X, x{:}T_{12} \vdash t_2 : T_2}{\Gamma \vdash \texttt{let }\{X,x\}=t_1\texttt{ in }t_2 : T_2} \quad \text{(T-Unpack)}$$

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

```
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

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

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

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

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

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