Lesson 12
Existential Types

2/29

Chapter 24

Existential Types

- Existential packages
- Existential types and abstract types
- Existential types and objects
- Encoding existentials with universals

Existential types

Existential types:

T ::= . . . | ∃X.T

Pierce uses the alternative, nonstandard notation {∃X, T} to suggest that the existential value is a mixed type-value tuple.

Existential types

A value of type ∃X.T is a package with a witness type T' for X and a value term t : [X => T']T.

pack X = T' with t as T : ∃X.T (conventional notation)
{*T', t} as {∃X, T} (Pierce's notation)

The *Intro* typing rule for existential types:

$$\frac{\Gamma \vdash t_1 : [X \Rightarrow U]T_1}{\Gamma \vdash \{*U, t_1\} \text{ as } \{\exists X, T_1\} : \{\exists X, T_1\}} \quad \text{(T- Pack)}$$

Examples of Existential types

{*Nat, 3} as {∃X, X} : {∃X, X}
{*Bool, true} as {∃X, X} : {∃X, X}

p = {*Nat, {a = 5, f = λx : Nat. succ x}} as {∃X, {a : X, f : X -> Nat}}
p : {∃X, {a : X, f : X -> Nat}}

q = {*Bool, {a = true, f = λx : Bool. 0}} as {∃X, {a : X, f : X -> Nat}}
q : {∃X, {a : X, f : X -> Nat}}

The type part is hidden (opaque, abstract), and the value part provides an interface for *interpreting* the hidden type.

Unpacking existential values

Unpacking an existential: let {X,x} = t_1 in t_2

$$\frac{\Gamma \vdash t_1 : \{\exists X, T_1\} \qquad \Gamma, X, x: T_1 \vdash t_2 : T_2}{\Gamma \vdash \text{let } \{X,x\} = t_1 \text{ in } t_2 \; : T_2} \quad \text{(T- Unpack)}$$

Type variable X cannot occur in T_2 -- it is not in scope (i.e. doesn't appear in the context Γ). This means that the name X of the existential witness type cannot "escape" from the let expression.

Also, within the body t_2, the type X is abstract and can only be used through the interface provided by x : T_1.

Abstract types as Existential types

```
Counter = {∃Counter, {new: Counter, get: Counter -> Nat,
                      inc: Counter -> Counter}}

counter0 = {*Nat,
            {new = 1,
             get = λx : Nat. x,
             inc = λx : Nat. succ x}}  as Counter
counter0 : Counter

let {C, cops} = counter0 in
  let add2 = λc : C. cops.inc(cops.inc c)
   in cops.get(add2(cops.new))
==> 4: Nat
```


Abstract types as Existential types

The idea:

let {X,x} = t_1 in t_2

The existential value packages the representation type of the abstract type (the existential witness), with the interface values through which to use the type.

The client code (t_2, the body of the let), does not have access to the representation type, and can only refer to it through the bound (and opaque) name X.

All code that interacts via the abstract type has to be included in t_2. This limitation can be overcome using the *dot notation*.

Object types as Existential types

```
Counter = {∃S, {state: S,
                methods: {get: S -> Nat, inc: S -> S}}}

c = {*Nat,
     {state = 1,
      methods = {get = λx : Nat. x,
                 inc = λx : Nat. succ x}}}  as Counter
c : Counter
sendget = λx : Counter.
            let {S, body} = x in
              body.methods.get(body.state)

sendget c ==> 1: Nat
```


Object types as Existential types

The idea:

The witness type of the existential value is the representation of the internal state of the object, and it is hidden by the existential package.

The methods operate on this internal state.

Encoding Existentials with Universals

Let

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

Then

$\{*S, t\} =_{def} \forall Y. \forall f : (\forall X. T \to Y). f\ [S]\ (t)$

and

$\text{let } \{X, x\} = t_1 \text{ in } t_2 =_{def} t_1\ [T_2]\ (\forall X. \forall f : T_1. t_2)$

where $t_1 : \{\exists X, T\}$ and $t_2 : T_2$.