Chapter 24: Existential Types

Existential Types

Power of Existential Types

Encoding Existential Types

Two Views of Existential Type {∃X,T}

- Logical Intuition: an element of {∃X,T} is a value of type [X → S]T, for some type S.

- Operational Intuition: an element of {∃X,T} is a pair, written {*S,t}, of a type S and a term t of type [X → S]T.
  - Like modules and abstract data types found in programming languages.

Example：
p = {*Nat, {a=5, f=λx:Nat. succ(x)}}
    as {∃X, {a:X, f:X→X}};

Existential Types

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 $\boxed{\Gamma \vdash t : T}$

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

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

Small Examples

- 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
- let {X,x}=p4 in (λy:X. x.f y) x.a;
  - 1 : Nat
- let {X,x}=p4 in succ(x.a);
  - Error: argument of succ is not a number
  - The only operations allowed on x are those warranted by its "abstract type" {a:X,f:X→Nat}

App1: Data Abstraction with Extentials

- Abstract Data Type

```
ADT counter =
  type Counter
  representation Nat
  signature
    new : Counter,
    get : Counter→Nat,
    inc : Counter→Counter;
  operations
    new = 1,
    get = λi:Nat. i,
    inc = λi:Nat.
```

For external use

Hidden Internal implementation

- Abstract Data Type in Existential Types

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

- Use Examples

let {Counter,counter} = counterADT

in counter.get (counter.inc counter.new);

➔ 2 : Nat

```
let {Counter,counter} = counterADT in

let {FlipFlop,flipflop} =
     {*Counter,
      {new    = counter.new,
       read   = λc:Counter. iseven (counter.get c),
       toggle = λc:Counter. counter.inc c,
       reset  = λc:Counter. counter.new}}
   as {∃FlipFlop,
       {new:    FlipFlop, read: FlipFlop→Bool,
        toggle: FlipFlop→FlipFlop, reset: FlipFlop→FlipFlop}} in

flipflop.read (flipflop.toggle (flipflop.toggle flipflop.new));
```

- Representation-Independent

```
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}}
```

App2: Existential Object

Internal state

Set of methods

```
c = {*Nat,
      {state = 5,
       methods = {get = λx:Nat. x,
                  inc = λx:Nat. succ(x)}}}
    as Counter;
```

where:

```
Counter = {∃X, {state:X, methods: {get:X→Nat, inc:X→X}}};
```

Example:

let {X,body} = c in body.methods.get(body.state);

Encoding Existentials

- Pair can be encoded in System F.

$\{U,V\} = \forall X.\ (U \to V \to X) \to X$

pair : $U \to V \to PairNat$

pair = $\lambda n1{:}U.\ \lambda n2{:}V.$
$\quad\quad \lambda X.\ \lambda f{:}U \to V \to X.\ f\ n1\ n2;$

fst: $\{U,V\} \to U$

fst = $\lambda p{:}\{U,V\}.\ p\ [U]\ (\lambda n1{:}U.\ \lambda n2{:}V.\ n1);$

snd : $\{U,V\} \to V$

snd = $\lambda p{:}\{U,V\}.\ p\ [V]\ (\lambda n1{:}U.\ \lambda n2{:}V.\ n2);$

- Existential Encoding

$\{\exists X, T\} = \forall Y. (\forall X. T \rightarrow Y) \rightarrow Y$

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

let $\{X,x\}$=t1 in t2 = t1 [T2] (λX. λx:T11.t2)

(if x :: T11, let ⋯ t2: T2)

Exercise: Show that

let {X,x}=({*T11,v12} as T1) in t2

→[X→T11][x→v12]t2