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

$$\begin{array}{llll} \mathtt{t} & ::= & \ldots & \textit{terms:} \\ & & \{{*}\mathtt{T},\mathtt{t}\}\ \mathtt{as}\ \mathtt{T} & \textit{packing} \\ & & \mathtt{let}\ \{\mathtt{X},\mathtt{x}\}\mathtt{=t}\ \mathtt{in}\ \mathtt{t} & \textit{unpacking} \end{array}$$

$$\begin{array}{llll} \mathtt{v} & ::= & \ldots & \textit{values:} \\ & & \{{*}\mathtt{T},\mathtt{v}\}\ \mathtt{as}\ \mathtt{T} & \textit{package value} \end{array}$$

$$\begin{array}{llll} \mathtt{T} & ::= & \ldots & \textit{types:} \\ & & \{\exists\mathtt{X},\mathtt{T}\} & \textit{existential type} \end{array}$$

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

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

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

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. i+1
```

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.

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

  pair : $U \rightarrow V \rightarrow \{U,V\}$

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

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

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

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

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

- Existential Encoding

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

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

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

  (if x :: T11, let … t2: T2)

  Exercise: Show that

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

  $\to [X \to T11][x \to v12]t2$