Chapter 24: Existential Types

Existential Types
Power of Existential Types
Encoding Existential Types
Two Views of Existential Type $\exists X, T$

- **Logical Intuition**: an element of $\exists X, T$ is a value of type $[X \to S]T$, for some type $S$.

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

Example:

\[ p = \{\text{Nat}, \{a=5, f=\lambda x:\text{Nat. } \text{succ}(x)\}\} \]

as $\exists X, \{a:X, f:X \to X\}$;
Existential Types

New syntactic forms
\[ t ::= \ldots \]
\[ \{^*T,t\} \text{ as } T \]
\[ \text{let } \{X,x\}=t \text{ in } t \]
\[ v ::= \ldots \]
\[ \{^*T,v\} \text{ as } T \]
\[ T ::= \ldots \]
\[ \{\exists X,T\} \]

New evaluation rules
\[ \text{let } \{X,x\}=\{^*T_{11},v_{12}\} \text{ as } T_1 \text{ in } t_2 \]
\[ \rightarrow [X \rightarrow T_{11}][x \rightarrow v_{12}] t_2 \]

New typing rules
\[ \Gamma \vdash t : T \]
\[ \Gamma \vdash t_2 : [X \rightarrow U]T_2 \]
\[ \Gamma \vdash \{^*U,t_2\} \text{ as } \{\exists X,T_2\} : \{\exists X,T_2\} \]
\[ \Gamma \vdash \text{let } \{X,x\}=t_1 \text{ in } t_2 : T_2 \]

(E-PACK)

(E-UNPACK)

(T-PACK)

(T-UNPACK)
Small Examples

• \( p4 = \{a=0, f=\lambda x: \text{Nat}. \text{succ}(x)\} \)
  as \( \exists X, \{a:X, f:X \rightarrow \text{Nat}\} \);
  - \( p4 : \{\exists X, \{a:X, f:X \rightarrow \text{Nat}\}\} \)

• let \( \{X,x\} = p4 \) in \((x.f \times a)\);
  - 1 : \text{Nat}

• let \( \{X,x\} = p4 \) in \((\lambda y:X. x.f \times y) \times a\);
  - 1 : \text{Nat}

• let \( \{X,x\} = p4 \) in \text{succ}(x.a);
  - Error: argument of \text{succ} is not a number
  - The only operations allowed on \( x \) are those warranted by its “abstract type” \( \{a:X, f:X \rightarrow \text{Nat}\} \)
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 = \lambda i: Nat. i,
    inc = \lambda i: Nat.
Abstract Data Type in Existential Types

counterADT =
{Nat,
  {new = 1,
   get = \lambda i:Nat. i,
   inc = \lambda i:Nat. succ(i)}
}
as
{\exists Counter,
  {new: Counter,
   get: Counter\rightarrow Nat,
   inc: Counter\rightarrow Counter}};
• Use Examples

```haskell
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: \text{Nat} \},
\{ \text{new} = \{ x = 1 \},
    \text{get} = \lambda i: \{ x: \text{Nat} \}. i \cdot x,
    \text{inc} = \lambda i: \{ x: \text{Nat} \}. \{ x = \text{succ}(i \cdot x) \}\}\}

as \{ \exists \text{Counter},
\{ \text{new: Counter, get: Counter} \rightarrow \text{Nat, inc: Counter} \rightarrow \text{Counter} \}\};

\cdot \text{counterADT} : \{ \exists \text{Counter},
\{ \text{new: Counter, get: Counter} \rightarrow \text{Nat, inc: Counter} \rightarrow \text{Counter} \}\}
App2: Existential Object

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.**

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

\[
\text{pair} : U \rightarrow V \rightarrow \text{PairNat}
\]
\[
\text{pair} = \lambda n1:U. \lambda n2:V.
\quad \lambda X. \lambda f:U \rightarrow V \rightarrow X. f \ n1 \ n2;
\]

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

\[
\text{snd} : \{U, V\} \rightarrow V
\]
\[
\text{snd} = \lambda p:{U, V}. p \ [V] (\lambda n1:U. \lambda n2:V. n2);
\]
• 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 \( \cdots t2: T2 \))

**Exercise:** Show that

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

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