Product Types / Sum Types

Types

nat
string
bool

Type constructors

$\tau_1 \rightarrow \tau_2$

Gödel's T

int $\rightarrow$ string

int $\rightarrow$ int $\rightarrow$ int

(int $\rightarrow$ int) $\rightarrow$ int

Products

- SML tuples (1, "a", true)
- C → structs
- Python → objects

Binary products

$\tau_1 * \tau_2$ for any types τ_1 and τ_2

values/
done
exps

(e_1, e_2)
done

eager: e_1 done
e_2 done

$$\frac{e_1 : \tau_1 \quad e_2 : \tau_2}{(e_1, e_2) : \tau_1 \times \tau_2}$$

Int x String	String x int
(4, "hi")	("a", 7)
(3, "a")	("b", 8)

how do you use an element of the product type?

→ getting out the left and right sides

#1 $\longrightarrow$ get the left

#2 $\longrightarrow$ get the right

$$\#1(4, \text{"a"}) \longmapsto 4$$

$$\#1(5, \text{"b"}) \longmapsto 5$$

$$\#2(4, \text{"a"}) \longmapsto \text{"a"}$$

$$\#1(\text{"a"}, 4) \longmapsto \text{"a"}$$

⋮

$$\tau ::= \cdots \mid \tau_1 \times \tau_2$$

$$e ::= \cdots \mid \langle e_1, e_2 \rangle \mid \text{projleft}(e) \mid \text{projright}(e)$$

SML $\quad (e_1, e_2) \mid \#1\ e \mid \#2\ e$

PFPL $\quad \langle e_1, e_2 \rangle \mid e \cdot l \mid e \cdot r$

Typing

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{projright}(e) : \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{projleft}(e) : \tau_1}$$

$$\frac{e_1 \text{ done} \quad e_2 \text{ done}}{\langle e_1, e_2 \rangle \text{ done}}$$

eager

$$\frac{e_1 \longmapsto e_1'}{\langle e_1, e_2 \rangle \longmapsto \langle e_1', e_2 \rangle} \qquad \frac{e_2 \longmapsto e_2' \quad e_1 \text{ done}}{\langle e_1, e_2 \rangle \longmapsto \langle e_1, e_2' \rangle}$$

$$\frac{e \longmapsto e'}{\begin{array}{l} \text{projleft}(e) \longmapsto \text{projleft}(e') \\ \text{projright}(e) \longmapsto \text{projright}(e') \end{array}}$$

search

$$\frac{\langle e_1, e_2 \rangle \text{ done}}{\begin{array}{l} \text{projleft}(\langle e_1, e_2 \rangle) \longmapsto e_1 \\ \text{projright}(\langle e_1, e_2 \rangle) \longmapsto e_2 \end{array}}$$

instr

$$\dfrac{\dfrac{1+1 \mapsto 2}{\langle 1+1, 3\rangle \mapsto \langle 2, 3\rangle}}{\text{projleft}\langle 1+1, 3\rangle \mapsto \text{projleft}\langle 2, 3\rangle} \quad \dfrac{}{\mapsto 2}$$

$$\text{projright}\langle 1+1, 3\rangle \mapsto \text{projright}\langle 2, 3\rangle \mapsto 3$$

Combining type constructors

① products and products

triples:

$(\text{int} \times \text{string}) \times \text{bool}$ — $\text{int} \times \text{string} \times \text{bool}$

$\langle \langle 4, \text{"a"} \rangle, \text{true} \rangle$

$\text{int} \times (\text{string} \times \text{bool})$ — $\text{int} \times \text{string} \times \text{bool}$

$\langle 4, \langle \text{"a"}, \text{true} \rangle \rangle$

(2) products and functions

(int × string) → bool

fun f(p: int × string) : bool =

...... #1 p ..... #2 p......

fun add(p: int * int) : int = (#1 p) + (#2 p)

→ **multi-argument function**

$$\text{fun add}(\underline{x : int,\ y : int}) : int = x + y$$

$$\hookrightarrow \text{add} : int \times int \rightarrow int$$

int → (int × int) multi-output functions

e.g. Def

fib(1) = 1
fib(0) = 1
fib(n) = fib(n - 1) + fib(n - 2)

```
fun fastfib(n: int) : int * int =      (fib(n-1), fib(n))
  case n of
    0 => (0, 1)
  | _ => let val p = fastfib(n-1)
         in
           (#2 p, (#1 p) + (#2 p))
         end
```

Nullary Product

→ 0 types

binary

$\tau_1 \times \tau_2 \mid \langle e_1, e_2 \rangle$

$\tau ::= \dots \mid \text{unit}$

$e ::= \dots \mid \langle\rangle$

$$\frac{}{\Gamma \vdash \langle\rangle : \text{unit}}$$

$$\frac{}{\langle\rangle \text{ done}}$$

Labeled products

→ you choose the labels!

$e : \tau_1 \times \tau_2$

labels are generic:

#1 e #2 e

binary

projleft(e) projright(e)

e.l e.r

$e : \{ x : int, y : int, z : bool \}$

$e = \{ x = 4, y = 7, z = true \}$

e.x

e.y

e.z

Struct / object / module

```
{ x: int,
  y: string,
  print: int -> string
}
```

fields

Products vs Lists

Products

fixed length

but

heterogeneous

int x string

#1 → int

#2 → string

Lists

arbitrary length

but

homogeneous

[1, 2, 3, 7, ...]

process w/ a loop

where x:int ranges over each

Sum Types

→ a type that is either one kind of thing **or** another

A natural number is either
- 0
- 1+n, where...

A list is either
- []
- x::xs where

A typOrError is either [HW2/3]

- WellTyped(t) where t : typ

or - IllTyped

---

typOrError := typ + unit

WellTyped(t) := inleft(t)

IllTyped := inright($\langle \rangle$)

A result is either

- Done, or
- Stepped(e') where e' : exp

---

result := unit + exp

Done := inleft <>

Stepped(e') := inright(e')

A typOrErr [HWI] is either

- WellTyped(t), where t : typ
- IllTyped(s), where s : String

typOrErr := typ + string

WellTyped(t) := inleft(t)

IllTyped(s) := inright(s)

case (e) of
{ inleft(x) => _ e_1 _
inright(y) => _ e_2 _ }

A boolean is either

- true,

or - false

---

bool := unit + unit

true := $\text{inleft}\langle\rangle$

false := $\text{inright}\langle\rangle$

if b
then $e_1[\langle\rangle/x]$
else $e_2[\langle\rangle/y]$

case (b : bool) of
{ inleft(x) => e_1
inright(y) => e_2 }

x, y : unit

<u>type</u> $\tau ::= \ldots \mid \tau_1 + \tau_2$

"either a τ_1 or a τ_2, but tagged with a constructor"

<u>exp</u> $e ::= \ldots \mid \text{inleft}(e) \mid \text{inright}(e)$

$\mid \text{case}(e)\{x.e_1,\ y.e_2\}$ abstract

or

case e of
{ inleft x ⇒ e_1
| inright y ⇒ e_2 }

concrete

Typing

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \underline{\text{inleft}}(e) : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \underline{\text{inright}}(e) : \tau_1 + \tau_2}$$

"generic constructor names"

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma, x : \tau_1 \vdash e_1 : \tau \qquad \Gamma, y : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \text{case}(e)\{x.e_1 \mid y.e_2\} : \tau}$$

case e of
  $\{$ inleft $x \Rightarrow e_1$
  inright $y \Rightarrow e_2 \}$

Op Sem

$$\frac{e \text{ done}}{\text{inleft}(e) \text{ done}} \qquad \frac{e \text{ done}}{\text{inright}(e) \text{ done}}$$

search

$$\frac{e \longmapsto e'}{\text{inleft}(e) \longmapsto \text{inleft}(e')} \qquad \frac{e \longmapsto e'}{\text{inright}(e) \longmapsto \text{inright}(e')}$$

$$\frac{e \longmapsto e'}{\text{case}(e)\{x.e_1, y.e_2\} \longmapsto \text{case}(e')\{x.e_1, y.e_2\}}$$

intgr

$$\frac{e \text{ done}}{\text{case}(\text{inleft } e)\{x.e_1, y.e_2\} \longmapsto e_1[e/x]}$$

$$\text{case}(\text{inright } e)\{x.e_1, y.e_2\} \longmapsto e_2[e/x]$$

Simulate sum types w/ products

int + string

want

case e of
  inleft(x : int) => e_1
| inright(y : string) => e_2

inl = true
inr = false

S: { tag : bool,
     x : int
     y : string }

e

if x
  then e_1[s.x] [s.y]
  else e_2[s.y] [s.x]

inleft(e) = { t = true, x = e, y = junk }
inright(e) = { t = false, x = junk, y = e }