

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

τ_1 * τ_2 for any types τ_1 and τ_2

(values/
done
exps)

(e_1, e_2)
done

eager: e_1 done
 e_2 done

$e_1 : \tau_1 \quad e_2 : \tau_2$

 $(e_1, e_2) : \tau_1 * \tau_2$

Int x String

(4, "hi")

(3, "a")

String x Int

("a", 7)

("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, "a") \longmapsto 4

#1 (5, "b") \longmapsto 5

#2 (4, "a") \longmapsto "a"

#1 ("a", 4) \longmapsto "a"

⋮

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

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

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

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

Typing

$\Gamma \vdash e_1 : \tau_1$

$\Gamma \vdash e_2 : \tau_2$

$\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2$

$\Gamma \vdash e : \tau_1 \times \tau_2$

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

$\Gamma \vdash e : \tau_1 \times \tau_2$

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

e_1 done e_2 done

 $\langle e_1, e_2 \rangle$ done

Pages

$e_1 \mapsto e_1'$

 $\langle e_1, e_2 \rangle \mapsto \langle e_1', e_2 \rangle$

$e_2 \mapsto e_2'$ e_1 done

 $\langle e_1, e_2 \rangle \mapsto \langle e_1, e_2' \rangle$

$e \mapsto e'$

projleft(e) \mapsto projleft(e')
projright(e) \mapsto projright(e')

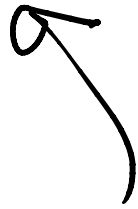
$\langle e_1, e_2 \rangle$ done

projleft($\langle e_1, e_2 \rangle$) \mapsto e_1
projright($\langle e_1, e_2 \rangle$) \mapsto e_2

Search

Instr

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

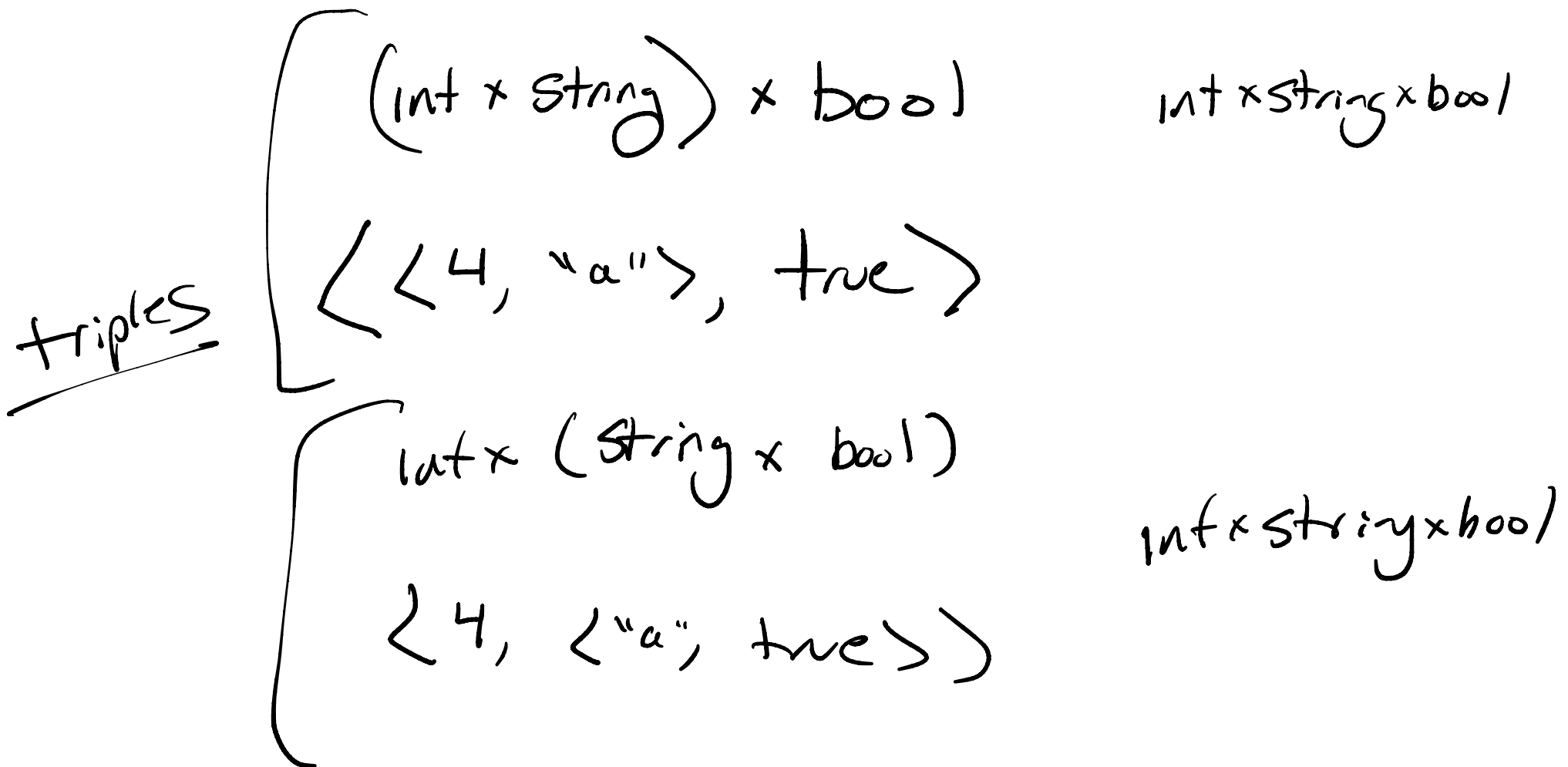


$$\text{projleft} \langle 1+1, 3 \rangle \mapsto \text{projleft} \langle 2, 3 \rangle \mapsto 2$$

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

Combining type constructors

① products and products



② products and functions

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

fun f(p: int * string): bool =

..... #1 p #2 p

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

→ multi-argument function

fun add(x:int, y:int) = x + y

↳ add: int x int → int

int \rightarrow (int * int)

multi-output
functions

e.g. Def

$\text{fib}(1) = 1$
 $\text{fib}(0) = 1$
 $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$

fun fastfib(n: int): int * int =

case n of

0 \Rightarrow (0, 1)

1 \Rightarrow let val p = fastfib(n-1)

in

(#2 p, (#1 p) + (#2 p))

end

Nullary product
→ 0 types

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

$\tau ::= \dots \mid \text{unit}$
 $e ::= \dots \mid \langle \rangle$

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

$\langle \rangle \text{ done}$

Labeled products \rightarrow you choose the labels!

$e: \tau_1 \times \tau_2$ labels are generic:
#1 e #2 e
projleft(e) projright(e)
 $e.l$ $e.r$

binary

$e: \{ x:int, y:int, z:bool \}$
 $e = \{ x=4, y=7, z=true \}$
 \hookrightarrow $\begin{cases} e.x \\ e.y \\ e.z \end{cases}$

Struct / object / module

{ x: int,

y: String

print: int → string

} fields

}

Products

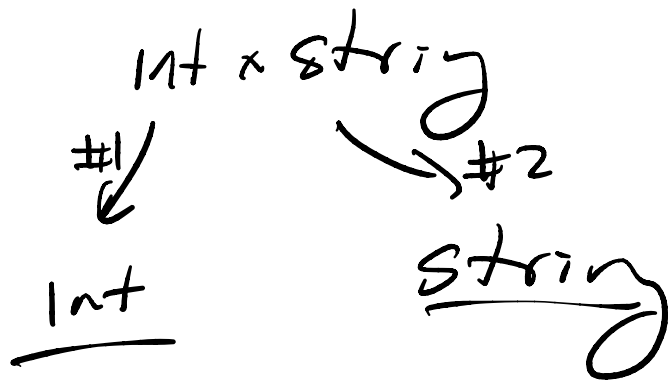
VS

Lists

fixed
length

but

heterogeneous



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 `TypeError` is either

[HW2/3]

- WellTyped(t) where $t: \text{typ}$

or - IllTyped

`TypeError` := typ + Unit

→ `WellTyped(t)` := `InLeft(t)`

→ `IllTyped` := `InRight(<>)`

A result is either

- Done, or

- Stopped(e') where $e' : \text{exp}$

result := unit + exp

→ Done := inleft $\langle \rangle$

→ Stopped(e') := inright(e')

A `typOrErr` [HW1] is either

- WellTyped (t), where $t: \text{typ}$

- IllTyped (s), where $s: \text{String}$

$\text{typOrErr} := \underline{\text{typ}} + \underline{\text{String}}$

$\text{WellTyped}(t) := \text{inleft}(t)$

$\text{IllTyped}(s) := \text{inright}(s)$

case (e) of
 $\{ \text{inleft}(x) \Rightarrow \dots, \text{inright}(y) \Rightarrow \dots \}$

A boolean is either

- true,

or - false

bool := unit + unit

→ true := inleft $\langle \rangle$

→ false := inright $\langle \rangle$

if b
then . e1 [$\langle \rangle$ / x]
else . e2 [$\langle \rangle$ / y]

case (b: bool) of
{ inleft x \Rightarrow . e1
inright y \Rightarrow . e2 }
x, y: unit

type $\tau ::= \dots \mid \tau_1 + \tau_2$

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

exp $e ::= \dots \mid \text{inleft}(e) \mid \text{inright}(e)$

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

or

case e of

{ inleft x \Rightarrow e₁

inright y \Rightarrow e₂ }

concrete

Typing

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

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

"generic constructor names"

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

case e of

{ inleft x \Rightarrow e₁

inright y \Rightarrow e₂ }

$$\boxed{\text{Op Sem}} \xrightarrow[\text{inleft}(e) \text{ done}]{e \text{ done}}$$

$$\xrightarrow[\text{inright}(e) \text{ done}]{e \text{ done}}$$

$$\frac{e \mapsto e'}{\text{inleft}(e) \mapsto \text{inleft}(e')}$$

$$\frac{e \mapsto e'}{\text{inright}(e) \mapsto \text{inright}(e')}$$

search

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

instr

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

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

Simulate sum types w/ products

int + string

want

case e of
 inleft(x: int) => e₁
 | inright(y: string) => e₂

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

inl = true
 inr = false

if x
 true e₁[s.x] (s.y)
 else e₂[s.y] [s.x]

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