

Foundations of Software Fall 2023

Week 7

1

Plan

PREVIOUSLY: unit, sequencing, let, pairs, tuples

TODAY:

1. options, variants
2. recursion
3. state

NEXT: exceptions?

NEXT: polymorphic (not so simple) typing

2

Records

$t ::= \dots$	<i>terms</i>
$\{l_i = t_i \mid i \in 1..n\}$	<i>record</i>
$t.l$	<i>projection</i>
$v ::= \dots$	<i>values</i>
$\{l_i = v_i \mid i \in 1..n\}$	<i>record value</i>
$T ::= \dots$	<i>types</i>
$\{l_i : T_i \mid i \in 1..n\}$	<i>type of records</i>

3

Evaluation rules for records

$$\{l_i = v_i \mid i \in 1..n\}.l_j \longrightarrow v_j \quad (\text{E-PROJRCD})$$

$$\frac{t_1 \longrightarrow t'_1}{t_1.l \longrightarrow t'_1.l} \quad (\text{E-PROJ})$$

$$\frac{t_j \longrightarrow t'_j}{\{l_i = v_i \mid i \in 1..j-1, l_j = t_j, l_k = t_k \mid k \in j+1..n\} \longrightarrow \{l_i = v_i \mid i \in 1..j-1, l_j = t'_j, l_k = t_k \mid k \in j+1..n\}} \quad (\text{E-RCD})$$

4

Typing rules for records

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i = t_i \mid i \in 1..n\} : \{l_i : T_i \mid i \in 1..n\}} \quad (\text{T-RCD})$$

$$\frac{\Gamma \vdash t_1 : \{l_i : T_i \mid i \in 1..n\}}{\Gamma \vdash t_1.l_j : T_j} \quad (\text{T-PROJ})$$

5

Sums and variants

6

Sums – motivating example

```
PhysicalAddr = {firstlast:String, addr:String}
VirtualAddr  = {name:String, email:String}
Addr         = PhysicalAddr + VirtualAddr
inl  : "PhysicalAddr → PhysicalAddr+VirtualAddr"
inr  : "VirtualAddr → PhysicalAddr+VirtualAddr"
```

```
getName = λa:Addr.
  case a of
    inl x ⇒ x.firstlast
  | inr y ⇒ y.name;
```

7

New syntactic forms

$t ::= \dots$	<i>terms</i>
inl t	tagging (left)
inr t	tagging (right)
case t of inl x⇒t inr x⇒t	case
$v ::= \dots$	<i>values</i>
inl v	tagged value (left)
inr v	tagged value (right)
$T ::= \dots$	<i>types</i>
T+T	sum type

T_1+T_2 is a *disjoint union* of T_1 and T_2 (the tags `inl` and `inr` ensure disjointness)

8

New evaluation rules

 $t \longrightarrow t'$

$$\frac{}{\text{case } (\text{inl } v_0) \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow [x_1 \mapsto v_0]t_1} \text{ (E-CASEINL)}$$

$$\frac{}{\text{case } (\text{inr } v_0) \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow [x_2 \mapsto v_0]t_2} \text{ (E-CASEINR)}$$

$$\frac{t_0 \longrightarrow t'_0}{\text{case } t_0 \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow \text{case } t'_0 \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2} \text{ (E-CASE)}$$

$$\frac{t_1 \longrightarrow t'_1}{\text{inl } t_1 \longrightarrow \text{inl } t'_1} \text{ (E-INL)}$$

$$\frac{t_1 \longrightarrow t'_1}{\text{inr } t_1 \longrightarrow \text{inr } t'_1} \text{ (E-INR)}$$

9

New typing rules

 $\Gamma \vdash t : T$

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \text{inl } t_1 : T_1 + T_2} \text{ (T-INL)}$$

$$\frac{\Gamma \vdash t_1 : T_2}{\Gamma \vdash \text{inr } t_1 : T_1 + T_2} \text{ (T-INR)}$$

$$\frac{\Gamma \vdash t_0 : T_1 + T_2 \quad \Gamma, x_1 : T_1 \vdash t_1 : T \quad \Gamma, x_2 : T_2 \vdash t_2 : T}{\Gamma \vdash \text{case } t_0 \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 : T} \text{ (T-CASE)}$$

10

Types of Sums

Consider the term

$$t = \text{inl } (\text{succ } 0)$$

Clicker question: What can we say about it? (multiple possible answers)

- A. $\vdash t : \text{Nat}$
- B. $\vdash t : \text{Nat} + \text{Bool}$
- C. $\vdash t : \text{Bool} + \text{Nat}$
- D. $\vdash t : \text{Nat} + \text{Nat}$

URL: tppoll.eu
Session ID: **cs452**

11

Sums and Uniqueness of Types

Problem:

If t has type T , then $\text{inl } t$ has type $T+U$ for every U .

I.e., we've lost uniqueness of types.

Possible solutions:

- ▶ "Infer" U as needed during typechecking
- ▶ Give constructors different names and only allow each name to appear in one sum type (requires generalization to "variants," which we'll see next) — OCaml's solution
- ▶ Annotate each **inl** and **inr** with the intended sum type
- ▶ (Add subtyping and a "bottom" (**Nothing**) type — Scala's solution)

For simplicity, let's choose the third.

12

New syntactic forms

$t ::= \dots$ $\text{inl } t \text{ as } T$ $\text{inr } t \text{ as } T$ $v ::= \dots$ $\text{inl } v \text{ as } T$ $\text{inr } v \text{ as } T$	<i>terms</i> <i>tagging (left)</i> <i>tagging (right)</i> <i>values</i> <i>tagged value (left)</i> <i>tagged value (right)</i>
--	---

Note that $\text{as } T$ here is not the ascription operator that we saw before — i.e., not a separate syntactic form: in essence, there is an ascription “built into” every use of inl or inr .

13

New typing rules

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

$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \text{inl } t_1 \text{ as } T_1 + T_2 : T_1 + T_2}$	(T-INL)
$\frac{\Gamma \vdash t_1 : T_2}{\Gamma \vdash \text{inr } t_1 \text{ as } T_1 + T_2 : T_1 + T_2}$	(T-INR)

14

Evaluation rules ignore annotations:

$\boxed{t \longrightarrow t'}$

$\begin{array}{l} \text{case } (\text{inl } v_0 \text{ as } T_0) \\ \text{of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \\ \quad \longrightarrow [x_1 \mapsto v_0]t_1 \end{array}$	(E-CASEINL)
$\begin{array}{l} \text{case } (\text{inr } v_0 \text{ as } T_0) \\ \text{of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \\ \quad \longrightarrow [x_2 \mapsto v_0]t_2 \end{array}$	(E-CASEINR)
$\frac{t_1 \longrightarrow t'_1}{\text{inl } t_1 \text{ as } T_2 \longrightarrow \text{inl } t'_1 \text{ as } T_2}$	(E-INL)
$\frac{t_1 \longrightarrow t'_1}{\text{inr } t_1 \text{ as } T_2 \longrightarrow \text{inr } t'_1 \text{ as } T_2}$	(E-INR)

15

Variants

Just as we generalized binary products to labeled records, we can generalize binary sums to labeled *variants*.

16

New syntactic forms

$t ::= \dots$
 $\langle l=t \rangle \text{ as } T$
 $\text{case } t \text{ of } \langle l_i=x_i \rangle \Rightarrow t_i \quad i \in 1..n$

terms
tagging
case

$T ::= \dots$
 $\langle l_i : T_i \rangle \quad i \in 1..n$

types
type of variants

17

New evaluation rules

$t \longrightarrow t'$

$\text{case } \langle l_j=v_j \rangle \text{ as } T \text{ of } \langle l_i=x_i \rangle \Rightarrow t_i \quad i \in 1..n \longrightarrow [x_j \mapsto v_j] t_j$ (E-CASEVARIANT)

$\frac{t_0 \longrightarrow t'_0}{\text{case } t_0 \text{ of } \langle l_i=x_i \rangle \Rightarrow t_i \quad i \in 1..n \longrightarrow \text{case } t'_0 \text{ of } \langle l_i=x_i \rangle \Rightarrow t_i \quad i \in 1..n}$ (E-CASE)

$\frac{t_i \longrightarrow t'_i}{\langle l_i=t_i \rangle \text{ as } T \longrightarrow \langle l_i=t'_i \rangle \text{ as } T}$ (E-VARIANT)

18

New typing rules

$\Gamma \vdash t : T$

$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \langle l_j=t_j \rangle \text{ as } \langle l_i : T_i \rangle \quad i \in 1..n : \langle l_i : T_i \rangle \quad i \in 1..n}$ (T-VARIANT)

$\frac{\begin{array}{c} \Gamma \vdash t_0 : \langle l_i : T_i \rangle \quad i \in 1..n \\ \text{for each } i \quad \Gamma, x_i : T_i \vdash t_i : T \end{array}}{\Gamma \vdash \text{case } t_0 \text{ of } \langle l_i=x_i \rangle \Rightarrow t_i \quad i \in 1..n : T}$ (T-CASE)

19

Example

```
Addr = <physical:PhysicalAddr, virtual:VirtualAddr>;

a = <physical=pa> as Addr;

getName =  $\lambda a : \text{Addr}.$ 
  case a of
    <physical=x>  $\Rightarrow$  x.firstlast
  | <virtual=y>  $\Rightarrow$  y.name;
```

20

Options

Just like in OCaml...

```
OptionalNat = <none:Unit, some:Nat>;

Table = Nat → OptionalNat;

emptyTable = λn:Nat. <none=unit> as OptionalNat;

extendTable =
  λt:Table. λm:Nat. λv:Nat.
    λn:Nat.
      if equal n m then <some=v> as OptionalNat
      else t n;

x = case t(5) of
  <none=u> ⇒ 999
  | <some=v> ⇒ v;
```

21

Enumerations

```
Weekday = <monday:Unit, tuesday:Unit, wednesday:Unit,
           thursday:Unit, friday:Unit>;

nextBusinessDay = λw:Weekday.
  case w of <monday=x>    ⇒ <tuesday=unit> as Weekday
  | <tuesday=x>           ⇒ <wednesday=unit> as Weekday
  | <wednesday=x>        ⇒ <thursday=unit> as Weekday
  | <thursday=x>         ⇒ <friday=unit> as Weekday
  | <friday=x>           ⇒ <monday=unit> as Weekday;
```

22