Foundations of Software
Fall 2022

Week 7

Plan

PREVIOUSLY: unit, sequencing, let, pairs, tuples

TODAY:
1. options, variants
2. recursion
3. state

NEXT: exceptions?
NEXT: polymorphic (not so simple) typing

Records

$$
\begin{array}{llll}
\texttt{t} & ::= & ... & \textit{terms} \\
& & \{\texttt{l}_i\texttt{=}\texttt{t}_i{}^{i \in 1..n}\} & \textit{record} \\
& & \texttt{t.l} & \textit{projection} \\
\\
\texttt{v} & ::= & ... & \textit{values} \\
& & \{\texttt{l}_i\texttt{=}\texttt{v}_i{}^{i \in 1..n}\} & \textit{record value} \\
\\
\texttt{T} & ::= & ... & \textit{types} \\
& & \{\texttt{l}_i\texttt{:}\texttt{T}_i{}^{i \in 1..n}\} & \textit{type of records}
\end{array}
$$

Evaluation rules for records

$$
\{\texttt{l}_i\texttt{=}\texttt{v}_i{}^{i \in 1..n}\}.\texttt{l}_j \longrightarrow \texttt{v}_j \qquad \text{(E-ProjRcd)}
$$

$$
\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1.\texttt{l} \longrightarrow \texttt{t}_1'.\texttt{l}} \qquad \text{(E-Proj)}
$$

$$
\frac{\texttt{t}_j \longrightarrow \texttt{t}_j'}{\begin{array}{l}\{\texttt{l}_i\texttt{=}\texttt{v}_i{}^{i \in 1..j-1},\texttt{l}_j\texttt{=}\texttt{t}_j,\texttt{l}_k\texttt{=}\texttt{t}_k{}^{k \in j+1..n}\} \\ \longrightarrow \{\texttt{l}_i\texttt{=}\texttt{v}_i{}^{i \in 1..j-1},\texttt{l}_j\texttt{=}\texttt{t}_j',\texttt{l}_k\texttt{=}\texttt{t}_k{}^{k \in j+1..n}\}\end{array}} \qquad \text{(E-Rcd)}
$$

Typing rules for records

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

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

Sums and variants

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;
```

New syntactic forms

t	::=	...		*terms*
		inl t		*tagging (left)*
		inr t		*tagging (right)*
		case t of inl x⇒t | inr x⇒t		*case*

v	::=	...		*values*
		inl v		*tagged value (left)*
		inr v		*tagged value (right)*

| T | ::= | ... | | *types* |
| | | 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)

Slide 9

New evaluation rules $\boxed{t \longrightarrow t'}$

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

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

$$\frac{t_0 \longrightarrow t_0'}{\begin{array}{c}\text{case } t_0 \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \\ \longrightarrow \text{case } t_0' \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2\end{array}} \quad \text{(E-Case)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{inl } t_1 \longrightarrow \text{inl } t_1'} \quad \text{(E-Inl)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{inr } t_1 \longrightarrow \text{inr } t_1'} \quad \text{(E-Inr)}$$

9

Slide 10

New typing rules $\boxed{\Gamma \vdash t : T}$

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

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

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

10

Slide 11

Sums and Uniqueness of Types

Problem:

If t has type T, then $inl\ t$ 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.

For simplicity, let's choose the third.

11

Slide 12

New syntactic forms

```
t ::= ...                          terms
      inl t as T                   tagging (left)
      inr t as T                   tagging (right)

v ::= ...                          values
      inl v as T                   tagged value (left)
      inr v as T                   tagged value (right)
```

Note that `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`.

12

New typing rules $\boxed{\Gamma \vdash t : T}$

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

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

Evaluation rules ignore annotations: $\boxed{t \longrightarrow t'}$

$$\begin{array}{c} \text{case (inl } v_0 \text{ as } T_0) \\ \text{of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \\ \longrightarrow [x_1 \mapsto v_0] t_1 \end{array} \quad \text{(E-CASEINL)}$$

$$\begin{array}{c} \text{case (inr } v_0 \text{ as } T_0) \\ \text{of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \\ \longrightarrow [x_2 \mapsto v_0] t_2 \end{array} \quad \text{(E-CASEINR)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{inl } t_1 \text{ as } T_2 \longrightarrow \text{inl } t_1' \text{ as } T_2} \quad \text{(E-INL)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{inr } t_1 \text{ as } T_2 \longrightarrow \text{inr } t_1' \text{ as } T_2} \quad \text{(E-INR)}$$

Variants

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

New syntactic forms

$$
\begin{array}{llr}
\text{t} & ::= \ldots & \textit{terms} \\
& \text{<l=t> as T} & \textit{tagging} \\
& \text{case t of <l}_i\text{=x}_i\text{>}\Rightarrow\text{t}_i{}^{i\in1..n} & \textit{case} \\
\\
\text{T} & ::= \ldots & \textit{types} \\
& \text{<l}_i\text{:T}_i{}^{i\in1..n}\text{>} & \textit{type of variants}
\end{array}
$$

$$\text{case } (\texttt{<}l_j\texttt{=}v_j\texttt{>} \text{ as } T) \text{ of } \texttt{<}l_i\texttt{=}x_i\texttt{>}{\Rightarrow}t_i{}^{i\in1..n} \quad \text{(E-CaseVariant)}$$
$$\longrightarrow [x_j \mapsto v_j]t_j$$

$$\frac{t_0 \longrightarrow t'_0}{\begin{array}{c}\text{case } t_0 \text{ of } \texttt{<}l_i\texttt{=}x_i\texttt{>}{\Rightarrow}t_i{}^{i\in1..n}\\ \longrightarrow \text{case } t'_0 \text{ of } \texttt{<}l_i\texttt{=}x_i\texttt{>}{\Rightarrow}t_i{}^{i\in1..n}\end{array}} \quad \text{(E-Case)}$$

$$\frac{t_i \longrightarrow t'_i}{\texttt{<}l_i\texttt{=}t_i\texttt{>} \text{ as } T \longrightarrow \texttt{<}l_i\texttt{=}t'_i\texttt{>} \text{ as } T} \quad \text{(E-Variant)}$$

17

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \texttt{<}l_j\texttt{=}t_j\texttt{>} \text{ as } \texttt{<}l_i\texttt{:}T_i{}^{i\in1..n}\texttt{>} : \texttt{<}l_i\texttt{:}T_i{}^{i\in1..n}\texttt{>}} \quad \text{(T-Variant)}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_0 : \texttt{<}l_i\texttt{:}T_i{}^{i\in1..n}\texttt{>}\\ \text{for each } i \quad \Gamma, x_i\texttt{:}T_i \vdash t_i : T\end{array}}{\Gamma \vdash \text{case } t_0 \text{ of } \texttt{<}l_i\texttt{=}x_i\texttt{>}{\Rightarrow}t_i{}^{i\in1..n} : T} \quad \text{(T-Case)}$$

18

Example

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

a = <physical=pa> as Addr;

getName = λa:Addr.
  case a of
    <physical=x> ⇒ x.firstlast
  | <virtual=y> ⇒ y.name;
```

19

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;
```

20

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;
```

21