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

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

Evaluation rules for records

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

$$\frac{t_1 \longrightarrow t_1'}{t_1.l \longrightarrow t_1'.l} \qquad \text{(E-PROJ)}$$

$$\frac{t_j \longrightarrow t_j'}{\{l_i = v_i \ ^{i \in 1..j-1}, l_j = t_j, l_k = t_k \ ^{k \in j+1..n}\} \atop \longrightarrow \{l_i = v_i \ ^{i \in 1..j-1}, l_j = t_j', l_k = t_k \ ^{k \in j+1..n}\}} \qquad \text{(E-RCD)}$$

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)

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

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

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

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

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

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

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

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

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

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.

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`.

New typing rules

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

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

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

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

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

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

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

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

Variants

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

New syntactic forms

```
t ::= ...                                    terms
      <l=t> as T                               tagging
      case t of <lᵢ=xᵢ>⇒tᵢ  i∈1..n             case

T ::= ...                                    types
      <lᵢ:Tᵢ i∈1..n>                           type of variants
```

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

$$\text{case } (\text{<}l_j\text{=}v_j\text{> as T}) \text{ of } \text{<}l_i\text{=}x_i\text{>} \Rightarrow t_i \;^{i \in 1..n} \atop \longrightarrow [x_j \mapsto v_j]t_j \quad (\text{E-CaseVariant})$$

$$\frac{t_0 \longrightarrow t_0'}{\begin{array}{c} \text{case } t_0 \text{ of } \text{<}l_i\text{=}x_i\text{>} \Rightarrow t_i \;^{i \in 1..n} \\ \longrightarrow \text{case } t_0' \text{ of } \text{<}l_i\text{=}x_i\text{>} \Rightarrow t_i \;^{i \in 1..n} \end{array}} \quad (\text{E-Case})$$

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

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

$$\frac{\Gamma \vdash \texttt{t}_j : \texttt{T}_j}{\Gamma \vdash \texttt{<l}_j\texttt{=t}_j\texttt{>} \texttt{ as } \texttt{<l}_i\texttt{:T}_i{}^{i \in 1..n}\texttt{>} : \texttt{<l}_i\texttt{:T}_i{}^{i \in 1..n}\texttt{>}} \text{ (T-VARIANT)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \texttt{t}_0 : \texttt{<l}_i\texttt{:T}_i{}^{i \in 1..n}\texttt{>} \\ \text{for each } i \quad \Gamma, \texttt{x}_i\texttt{:T}_i \vdash \texttt{t}_i : \texttt{T}\end{array}}{\Gamma \vdash \texttt{case } \texttt{t}_0 \texttt{ of } \texttt{<l}_i\texttt{=x}_i\texttt{>}{\Rightarrow}\texttt{t}_i{}^{i \in 1..n} : \texttt{T}} \text{ (T-CASE)}$$

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;
```

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;
```

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;
```