Foundations of Software
Fall 2015

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

```
t  ::=  ...                                    terms
        {l_i=t_i  i∈1..n}                        record
        t.l                                      projection


v  ::=  ...                                    values
        {l_i=v_i  i∈1..n}                        record value


T  ::=  ...                                    types
        {l_i:T_i  i∈1..n}                        type of records
```

Evaluation rules for records

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

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

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

Typing rules for records

$$\frac{\text{for each } i \quad \Gamma \vdash \text{t}_i : \text{T}_i}{\Gamma \vdash \{\text{l}_i = \text{t}_i{}^{i \in 1..n}\} : \{\text{l}_i : \text{T}_i{}^{i \in 1..n}\}} \qquad \text{(T-Rcd)}$$

$$\frac{\Gamma \vdash \text{t}_1 : \{\text{l}_i : \text{T}_i{}^{i \in 1..n}\}}{\Gamma \vdash \text{t}_1 . \text{l}_j : \text{T}_j} \qquad \text{(T-Proj)}$$

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)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 <l_i=x_i>⇒t_i ^{i∈1..n}          case

T ::= ...                                    types
    <l_i:T_i ^{i∈1..n}>                        type of variants
```

New evaluation rules
$$t \longrightarrow t'$$

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

$$\frac{t_0 \longrightarrow t'_0}{\begin{array}{l} \text{case } t_0 \text{ of } \texttt{<} l_i \texttt{=} x_i \texttt{>} \Rightarrow t_i \ ^{i \in 1..n} \\ \longrightarrow \text{case } t'_0 \text{ of } \texttt{<} l_i \texttt{=} x_i \texttt{>} \Rightarrow t_i \ ^{i \in 1..n} \end{array}} \qquad (\text{E-CASE})$$

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

New typing rules

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

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

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;
```