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

$\mathtt{t}$	::=	...	*terms*
		$\{\mathtt{l}_i\mathtt{=t}_i\ ^{i\in 1..n}\}$	*record*
		$\mathtt{t.l}$	*projection*
$\mathtt{v}$	::=	...	*values*
		$\{\mathtt{l}_i\mathtt{=v}_i\ ^{i\in 1..n}\}$	*record value*
$\mathtt{T}$	::=	...	*types*
		$\{\mathtt{l}_i\mathtt{:T}_i\ ^{i\in 1..n}\}$	*type of records*

Evaluation rules for records

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

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

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

Typing rules for records

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

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

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*

$\mathtt{T}_1\mathtt{+}\mathtt{T}_2$ is a *disjoint union* of $\mathtt{T}_1$ and $\mathtt{T}_2$ (the tags `inl` and `inr` ensure disjointness)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Variants

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

New syntactic forms

$$\begin{array}{lll} \mathtt{t} & ::= & \ldots & \textit{terms} \\ & & \langle \mathtt{l{=}t} \rangle\ \mathtt{as}\ \mathtt{T} & \quad \textit{tagging} \\ & & \mathtt{case}\ \mathtt{t}\ \mathtt{of}\ \langle \mathtt{l}_i{=}\mathtt{x}_i \rangle {\Rightarrow} \mathtt{t}_i\ ^{i \in 1..n} & \quad \textit{case} \\ \\ \mathtt{T} & ::= & \ldots & \textit{types} \\ & & \langle \mathtt{l}_i{:}\mathtt{T}_i\ ^{i \in 1..n} \rangle & \quad \textit{type of variants} \end{array}$$

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

$$\begin{array}{l}\mathtt{case}\ (\mathtt{<l}_j\mathtt{=v}_j\mathtt{>}\ \mathtt{as}\ \mathtt{T})\ \mathtt{of}\ \mathtt{<l}_i\mathtt{=x}_i\mathtt{>}\Rightarrow\mathtt{t}_i\ ^{i\in 1..n} \\ \qquad\longrightarrow [\mathtt{x}_j \mapsto \mathtt{v}_j]\mathtt{t}_j\end{array} \quad \text{(E-CaseVariant)}$$

$$\frac{\mathtt{t}_0 \longrightarrow \mathtt{t}_0'}{\begin{array}{l}\mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \mathtt{<l}_i\mathtt{=x}_i\mathtt{>}\Rightarrow\mathtt{t}_i\ ^{i\in 1..n} \\ \longrightarrow \mathtt{case}\ \mathtt{t}_0'\ \mathtt{of}\ \mathtt{<l}_i\mathtt{=x}_i\mathtt{>}\Rightarrow\mathtt{t}_i\ ^{i\in 1..n}\end{array}} \quad \text{(E-Case)}$$

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

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

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

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

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;
```