Foundations of Software
Fall 2023

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

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

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

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

Typing rules for records

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

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

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*

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

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

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

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

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

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

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

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

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

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

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

Types of Sums

Consider the term

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

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

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

URL: `ttpoll.eu`
Session ID: `cs452`

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
- (Add subtyping and a "bottom" (`Nothing`) type — Scala's solution)

For simplicity, let's choose the third.

New syntactic forms

$\mathtt{t}$	::=	...	*terms*
		$\mathtt{inl\ t\ as\ T}$	*tagging (left)*
		$\mathtt{inr\ t\ as\ T}$	*tagging (right)*
$\mathtt{v}$	::=	...	*values*
		$\mathtt{inl\ v\ as\ T}$	*tagged value (left)*
		$\mathtt{inr\ v\ as\ T}$	*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 $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

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

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

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

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

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

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

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

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` $\texttt{<l}_i\texttt{=x}_i\texttt{>}\Rightarrow\texttt{t}_i\ ^{i\in 1..n}$	*case*
`T`	::=	...	*types*
		$\texttt{<l}_i\texttt{:T}_i\ ^{i\in 1..n}\texttt{>}$	*type of variants*

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

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

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

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

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

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

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

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;
```