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

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

3

Evaluation rules for records

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

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

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

4

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*

T_1+T_2 is a *disjoint union* of T_1 and T_2 (the tags `inl` and `inr` ensure disjointness)

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

$$\begin{array}{ll}\texttt{case (inl v}_0\texttt{)} & \\ \texttt{of inl x}_1\Rightarrow\texttt{t}_1 \texttt{ | inr x}_2\Rightarrow\texttt{t}_2 & \longrightarrow [\texttt{x}_1 \mapsto \texttt{v}_0]\texttt{t}_1\end{array} \quad \text{(E-CaseInl)}$$

$$\begin{array}{ll}\texttt{case (inr v}_0\texttt{)} & \\ \texttt{of inl x}_1\Rightarrow\texttt{t}_1 \texttt{ | inr x}_2\Rightarrow\texttt{t}_2 & \longrightarrow [\texttt{x}_2 \mapsto \texttt{v}_0]\texttt{t}_2\end{array} \quad \text{(E-CaseInr)}$$

$$\frac{\texttt{t}_0 \longrightarrow \texttt{t}_0'}{\begin{array}{c}\texttt{case t}_0 \texttt{ of inl x}_1\Rightarrow\texttt{t}_1 \texttt{ | inr x}_2\Rightarrow\texttt{t}_2 \\ \longrightarrow \texttt{case t}_0' \texttt{ of inl x}_1\Rightarrow\texttt{t}_1 \texttt{ | inr x}_2\Rightarrow\texttt{t}_2\end{array}} \quad \text{(E-Case)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{inl t}_1 \longrightarrow \texttt{inl t}_1'} \quad \text{(E-Inl)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{inr t}_1 \longrightarrow \texttt{inr t}_1'} \quad \text{(E-Inr)}$$

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1}{\Gamma \vdash \texttt{inl t}_1 : \texttt{T}_1\texttt{+T}_2} \quad \text{(T-Inl)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_2}{\Gamma \vdash \texttt{inr t}_1 : \texttt{T}_1\texttt{+T}_2} \quad \text{(T-Inr)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \texttt{t}_0 : \texttt{T}_1\texttt{+T}_2 \\ \Gamma, \texttt{x}_1\texttt{:T}_1 \vdash \texttt{t}_1 : \texttt{T} \qquad \Gamma, \texttt{x}_2\texttt{:T}_2 \vdash \texttt{t}_2 : \texttt{T}\end{array}}{\Gamma \vdash \texttt{case t}_0 \texttt{ of inl x}_1\Rightarrow\texttt{t}_1 \texttt{ | inr x}_2\Rightarrow\texttt{t}_2 : \texttt{T}} \quad \text{(T-Case)}$$

Types of Sums

Consider the term

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

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

A. $\vdash$ `t : Nat`

B. $\vdash$ `t : Nat + Bool`

C. $\vdash$ `t : Bool + Nat`

D. $\vdash$ `t : Nat + Nat`

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

Sums and Uniqueness of Types

Problem:

If t has type T, then $\mathtt{inl}\ t$ 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

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1}{\Gamma \vdash \mathtt{inl}\ \mathtt{t}_1\ \mathtt{as}\ \mathtt{T}_1\mathtt{+}\mathtt{T}_2 : \mathtt{T}_1\mathtt{+}\mathtt{T}_2} \quad \text{(T-INL)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_2}{\Gamma \vdash \mathtt{inr}\ \mathtt{t}_1\ \mathtt{as}\ \mathtt{T}_1\mathtt{+}\mathtt{T}_2 : \mathtt{T}_1\mathtt{+}\mathtt{T}_2} \quad \text{(T-INR)}$$

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

$$\begin{array}{l}\texttt{case (inl } \texttt{v}_0 \texttt{ as } \texttt{T}_0\texttt{)} \\ \texttt{of inl } \texttt{x}_1 \Rightarrow \texttt{t}_1 \texttt{ | inr } \texttt{x}_2 \Rightarrow \texttt{t}_2 \\ \qquad \longrightarrow [\texttt{x}_1 \mapsto \texttt{v}_0]\texttt{t}_1\end{array} \qquad \text{(E-CaseInl)}$$

$$\begin{array}{l}\texttt{case (inr } \texttt{v}_0 \texttt{ as } \texttt{T}_0\texttt{)} \\ \texttt{of inl } \texttt{x}_1 \Rightarrow \texttt{t}_1 \texttt{ | inr } \texttt{x}_2 \Rightarrow \texttt{t}_2 \\ \qquad \longrightarrow [\texttt{x}_2 \mapsto \texttt{v}_0]\texttt{t}_2\end{array} \qquad \text{(E-CaseInr)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{inl } \texttt{t}_1 \texttt{ as } \texttt{T}_2 \longrightarrow \texttt{inl } \texttt{t}_1' \texttt{ as } \texttt{T}_2} \qquad \text{(E-Inl)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{inr } \texttt{t}_1 \texttt{ as } \texttt{T}_2 \longrightarrow \texttt{inr } \texttt{t}_1' \texttt{ as } \texttt{T}_2} \qquad \text{(E-Inr)}$$

Variants

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

New syntactic forms

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

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

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

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

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

New typing rules

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

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

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

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;
```