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

$\mathtt{t}$ ::=	...	*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

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

$$\frac{\Gamma \vdash t_1 : \{l_i{:}T_i\ ^{i \in 1..n}\}}{\Gamma \vdash t_1.l_j : T_j} \qquad \text{(T-Proj)}$$

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 $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Variants

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

New syntactic forms

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

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

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

$$\frac{t_0 \longrightarrow t_0'}{\mathtt{case}\ t_0\ \mathtt{of}\ <l_i=x_i>\Rightarrow t_i\ ^{i\in 1..n} \longrightarrow \mathtt{case}\ t_0'\ \mathtt{of}\ <l_i=x_i>\Rightarrow t_i\ ^{i\in 1..n}} \qquad \text{(E-Case)}$$

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

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

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash <l_j=t_j>\ \mathtt{as}\ <l_i:T_i\ ^{i\in 1..n}> : <l_i:T_i\ ^{i\in 1..n}>} \qquad \text{(T-Variant)}$$

$$\frac{\Gamma \vdash t_0 : <l_i:T_i\ ^{i\in 1..n}> \qquad \text{for each } i \quad \Gamma, x_i:T_i \vdash t_i : T}{\Gamma \vdash \mathtt{case}\ t_0\ \mathtt{of}\ <l_i=x_i>\Rightarrow t_i\ ^{i\in 1..n} : T} \qquad \text{(T-Case)}$$

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;
```