Foundations of Software
Spring 2025

Week 7

Plan

PREVIOUSLY: unit, sequencing, let, pairs, tuples

TODAY:

1. pairs
2. sums (Either)
3. tuples
4. records
5. options, variants
6. recursion

NEXT: state
NEXT: polymorphic (not so simple) typing

Pairs

`t`	::=	...	*terms*
		`{t,t}`	*pair*
		`t.1`	*first projection*
		`t.2`	*second projection*
`v`	::=	...	*values*
		`{v,v}`	*pair value*
`T`	::=	...	*types*
		$\mathtt{T}_1 \times \mathtt{T}_2$	*product type*

Evaluation rules for pairs

$$\{v_1, v_2\}.1 \longrightarrow v_1 \qquad \text{(E-PairBeta1)}$$

$$\{v_1, v_2\}.2 \longrightarrow v_2 \qquad \text{(E-PairBeta2)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.1 \longrightarrow t_1'.1} \qquad \text{(E-Proj1)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.2 \longrightarrow t_1'.2} \qquad \text{(E-Proj2)}$$

$$\frac{t_1 \longrightarrow t_1'}{\{t_1, t_2\} \longrightarrow \{t_1', t_2\}} \qquad \text{(E-Pair1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\{v_1, t_2\} \longrightarrow \{v_1, t_2'\}} \qquad \text{(E-Pair2)}$$

Typing rules for pairs

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \quad \text{(T-Pair)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.1 : T_{11}} \quad \text{(T-Proj1)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.2 : T_{12}} \quad \text{(T-Proj2)}$$

Sums

[On blackboard]

Curry Howard Revisited

[On blackboard]

Tuples

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

Evaluation rules for tuples

$$\{v_i \; ^{i \in 1..n}\}.j \longrightarrow v_j \qquad \text{(E-ProjTuple)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.i \longrightarrow t_1'.i} \qquad \text{(E-Proj)}$$

$$\frac{t_j \longrightarrow t_j'}{\begin{array}{l}\{v_i \; ^{i \in 1..j-1}, t_j, t_k \; ^{k \in j+1..n}\} \\ \longrightarrow \{v_i \; ^{i \in 1..j-1}, t_j', t_k \; ^{k \in j+1..n}\}\end{array}} \qquad \text{(E-Tuple)}$$

Typing rules for tuples

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

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

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

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

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

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

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*

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

New evaluation rules

$\boxed{t \longrightarrow t'}$

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

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

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

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

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

New typing rules

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

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

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

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

Types of Sums

Consider the term

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

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

A. $\vdash \texttt{t : Nat}$

B. $\vdash \texttt{t : Nat + Bool}$

C. $\vdash \texttt{t : Bool + Nat}$

D. $\vdash \texttt{t : Nat + Nat}$

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

`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

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

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

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

Evaluation rules ignore annotations:

$\boxed{t \longrightarrow t'}$

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

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

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

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

Variants

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

New syntactic forms

`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

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

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \langle l_j = t_j \rangle \text{ as } \langle l_i : T_i \,^{i \in 1..n} \rangle : \langle l_i : T_i \,^{i \in 1..n} \rangle} \quad \text{(T-Variant)}$$

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

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;
```

Recursion

Recursion in $\lambda_{\to}$

- In $\lambda_{\to}$, all programs terminate. (Cf. Chapter 12.)
- Hence, untyped terms like `omega` and `fix` are not typable.
- But we can *extend* the system with a (typed) fixed-point operator...

Example

```
ff = λie:Nat→Bool.
       λx:Nat.
         if iszero x then true
         else if iszero (pred x) then false
         else ie (pred (pred x));

iseven = fix ff;

iseven 7;
```

New syntactic forms

$\mathtt{t} ::= \ldots$ — *terms*

$\mathtt{fix\ t}$ — *fixed point of* $\mathtt{t}$

New evaluation rules

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

$$\mathtt{fix}\ (\lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2) \longrightarrow [\mathtt{x} \mapsto (\mathtt{fix}\ (\lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2))]\mathtt{t}_2 \qquad \text{(E-FixBeta)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{fix}\ \mathtt{t}_1 \longrightarrow \mathtt{fix}\ \mathtt{t}_1'} \qquad \text{(E-Fix)}$$

What would be the typing rule for terms of the form $\mathtt{fix\ t}$?

New typing rules

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \to \mathtt{T}_1}{\Gamma \vdash \mathtt{fix}\ \mathtt{t}_1 : \mathtt{T}_1} \qquad \text{(T-Fix)}$$

A more convenient form

$$\texttt{letrec } x{:}T_1{=}t_1 \texttt{ in } t_2 \stackrel{\text{def}}{=} \texttt{let } x = \texttt{fix } (\lambda x{:}T_1.t_1) \texttt{ in } t_2$$

```
letrec iseven : Nat→Bool =
  λx:Nat.
    if iszero x then true
    else if iszero (pred x) then false
    else iseven (pred (pred x))
in
  iseven 7;
```