On to real programming languages...

The Unit type

$$
\begin{array}{lll}
\mathtt{t} & ::= & \ldots \\
 & & \mathtt{unit}
\end{array}
$$

terms

constant unit

$$
\begin{array}{lll}
\mathtt{v} & ::= & \ldots \\
 & & \mathtt{unit}
\end{array}
$$

values

constant $unit$

$$
\begin{array}{lll}
\mathtt{T} & ::= & \ldots \\
 & & \mathtt{Unit}
\end{array}
$$

types

unit type

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

$$\Gamma \vdash \mathtt{unit} : \mathtt{Unit} \qquad \text{(T-Unit)}$$

Sequencing

$\texttt{t}$	::=	...	*terms*
		$\texttt{t}_1\texttt{;t}_2$	

Sequencing

$$\mathtt{t} \; ::= \; \ldots \qquad \textit{terms}$$
$$\mathtt{t_1;t_2}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1;t_2} \longrightarrow \mathtt{t_1';t_2}} \qquad \text{(E-Seq)}$$

$$\mathtt{unit;t_2} \longrightarrow \mathtt{t_2} \qquad \text{(E-SeqNext)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{Unit} \qquad \Gamma \vdash \mathtt{t_2} : \mathtt{T_2}}{\Gamma \vdash \mathtt{t_1;t_2} : \mathtt{T_2}} \qquad \text{(T-Seq)}$$

Derived forms

- Syntatic sugar
- Internal language vs. external (surface) language

Sequencing as a derived form

$$\mathtt{t}_1\mathtt{;t}_2 \quad \stackrel{\text{def}}{=} \quad (\lambda\mathtt{x}\mathtt{:Unit.t}_2)\ \mathtt{t}_1$$

where $\mathtt{x} \notin FV(\mathtt{t}_2)$

Ascription

New syntactic forms

$\mathtt{t} ::=$	$\ldots$	*terms*
	$\mathtt{t\ as\ T}$	*ascription*

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

$$\mathtt{v_1\ as\ T} \longrightarrow \mathtt{v_1} \qquad \text{(E-Ascribe)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1\ as\ T} \longrightarrow \mathtt{t_1'\ as\ T}} \qquad \text{(E-Ascribe1)}$$

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

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T}}{\Gamma \vdash \mathtt{t_1\ as\ T} : \mathtt{T}} \qquad \text{(T-Ascribe)}$$

Ascription as a derived form

$$\texttt{t as T} \stackrel{\text{def}}{=} (\lambda\texttt{x:T. x})\ \texttt{t}$$

Let-bindings

New syntactic forms

$$\begin{array}{lll} \texttt{t} ::= & \dots & \textit{terms} \\ & \texttt{let x=t in t} & \quad\textit{let binding} \end{array}$$

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

$$\texttt{let x=v}_1 \texttt{ in t}_2 \longrightarrow [\texttt{x} \mapsto \texttt{v}_1]\texttt{t}_2 \qquad \text{(E-LetV)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{let x=t}_1 \texttt{ in t}_2 \longrightarrow \texttt{let x=t}_1' \texttt{ in t}_2} \qquad \text{(E-Let)}$$

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1 \qquad \Gamma, \texttt{x}{:}\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \texttt{let x=t}_1 \texttt{ in t}_2 : \texttt{T}_2} \qquad \text{(T-Let)}$$

Pairs, tuples, and records

Pairs

$\mathtt{t}$	::=	...	*terms*
		$\mathtt{\{t,t\}}$	*pair*
		$\mathtt{t.1}$	*first projection*
		$\mathtt{t.2}$	*second projection*
$\mathtt{v}$	::=	...	*values*
		$\mathtt{\{v,v\}}$	*pair value*
$\mathtt{T}$	::=	...	*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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \{\mathtt{t}_1,\mathtt{t}_2\} : \mathtt{T}_1 \times \mathtt{T}_2} \tag{T-Pair}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \times \mathtt{T}_{12}}{\Gamma \vdash \mathtt{t}_1.\mathtt{1} : \mathtt{T}_{11}} \tag{T-Proj1}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \times \mathtt{T}_{12}}{\Gamma \vdash \mathtt{t}_1.\mathtt{2} : \mathtt{T}_{12}} \tag{T-Proj2}$$

Tuples

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

Evaluation rules for tuples

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

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

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

Typing rules for tuples

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

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

Records

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

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

New evaluation rules

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

$$\begin{array}{ll}\mathtt{case\ (inl\ v_0)} & \longrightarrow [\mathtt{x}_1 \mapsto \mathtt{v}_0]\mathtt{t}_1 \\ \mathtt{of\ inl\ x_1 \Rightarrow t_1\ |\ inr\ x_2 \Rightarrow t_2} & \end{array} \qquad \text{(E-CaseInl)}$$

$$\begin{array}{ll}\mathtt{case\ (inr\ v_0)} & \longrightarrow [\mathtt{x}_2 \mapsto \mathtt{v}_0]\mathtt{t}_2 \\ \mathtt{of\ inl\ x_1 \Rightarrow t_1\ |\ inr\ x_2 \Rightarrow t_2} & \end{array} \qquad \text{(E-CaseInr)}$$

$$\frac{\mathtt{t}_0 \longrightarrow \mathtt{t}_0'}{\begin{array}{l}\mathtt{case\ t_0\ of\ inl\ x_1 \Rightarrow t_1\ |\ inr\ x_2 \Rightarrow t_2} \\ \longrightarrow \mathtt{case\ t_0'\ of\ inl\ x_1 \Rightarrow t_1\ |\ inr\ x_2 \Rightarrow t_2}\end{array}} \qquad \text{(E-Case)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{inl\ t_1} \longrightarrow \mathtt{inl\ t_1'}} \qquad \text{(E-Inl)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{inr\ t_1} \longrightarrow \mathtt{inr\ t_1'}} \qquad \text{(E-Inr)}$$

New typing rules

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

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

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

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

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

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

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

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

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*.

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;
```

New syntactic forms

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

New evaluation rules

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

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

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

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

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

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

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

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;
```