On to real programming languages...

The `Unit` type

unit is like "void"

t	::=	...	*terms*
		$unit$	*constant* $unit$
v	::=	...	*values*
		$unit$	*constant* $unit$
T	::=	...	*types*
		$Unit$	*unit type*

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

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

Sequencing

$$
\begin{array}{lll}
\texttt{t} & ::= & \ldots \qquad\qquad \textit{terms} \\
 & & \texttt{t}_1\texttt{;}\texttt{t}_2
\end{array}
$$

Sequencing

unit is like "void"

$$\mathtt{t} \quad ::= \quad \ldots \qquad\qquad \textit{terms}$$
$$\qquad\qquad \mathtt{t}_1\mathtt{;}\mathtt{t}_2$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\mathtt{;}\mathtt{t}_2 \longrightarrow \mathtt{t}_1'\mathtt{;}\mathtt{t}_2} \qquad \text{(E-Seq)}$$

$$\mathtt{unit;t}_2 \longrightarrow \mathtt{t}_2 \qquad \text{(E-SeqNext)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Unit} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \mathtt{t}_1\mathtt{;}\mathtt{t}_2 : \mathtt{T}_2} \qquad \text{(T-Seq)}$$

Derived forms

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

Sequencing as a derived form

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

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

Ascription

documentation, enforcing types

New syntactic forms

$\texttt{t} ::= \ldots$ — *terms*

$\texttt{t as T}$ — *ascription*

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

$$\texttt{v}_1 \ \texttt{as T} \longrightarrow \texttt{v}_1 \qquad \text{(E-Ascribe)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1 \ \texttt{as T} \longrightarrow \texttt{t}_1' \ \texttt{as T}} \qquad \text{(E-Ascribe1)}$$

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}}{\Gamma \vdash \texttt{t}_1 \ \texttt{as T} : \texttt{T}} \qquad \text{(T-Ascribe)}$$

Ascription as a derived form

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

Let-bindings

```
Prelude> let y = let x=6 in x+2
Prelude> y
8
```

New syntactic forms

$\texttt{t} ::= \ldots$ *terms*

$\quad \texttt{let x=t in t}$ *let binding*

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

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

Derived Form for let binding?

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

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

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x\texttt{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \texttt{let } x\texttt{=}t_1 \texttt{ in } t_2 : T_2} \qquad \text{(T-LET)}$$

Pairs, tuples, and records

Pairs

`t`	::=	...	*terms*
		`{t,t}`	*pair*
		`t.1`	*first projection*
		`t.2`	*second projection*
`v`	::=	...	*values*
		`{v,v}`	*pair value*
`T`	::=	...	*types*
		$T_1 \times T_2$	*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

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

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

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

Tuples

t	::=	...	*terms*
		$\{t_i\ ^{i\in 1..n}\}$	*tuple*
		t.i	*projection*
v	::=	...	*values*
		$\{v_i\ ^{i\in 1..n}\}$	*tuple value*
T	::=	...	*types*
		$\{T_i\ ^{i\in 1..n}\}$	*tuple type*

Evaluation rules for tuples

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

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

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

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

tuples with "labelled" components

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

Evaluation rules for records

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

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

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

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

```
data Ast = TRUE
         | FALSE
         | IFTHENELSE Ast Ast Ast
         | PAIR Ast Ast
         | FST Ast
         | SND Ast
```

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;
```

inject left (tagging)

inject right (tagging)

```
data Ast = FST Ast
         | SND Ast
```

```
type2 t = case t of
     FST (PAIR t1 t2) -> type2 t1
     SND (PAIR t1 t2) -> type2 t2
```

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

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

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

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

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

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

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

New typing rules

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

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

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

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

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

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

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

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

Evaluation rules ignore annotations:

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

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

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

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

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

Variants

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

```
data Ast = TRUE
         | FALSE
         | IFTHENELSE Ast Ast Ast
         | PAIR Ast Ast
         | FST Ast
         | SND Ast
```

New syntactic forms

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

New evaluation rules

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

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

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

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

New typing rules

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

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

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

```
data Ast = IFTHENELSE Ast Ast Ast
         | PAIR Ast Ast
         | FST Ast
         | SND Ast

type2 t = case t of
     IFTHENELSE t t2 t3 -> if (type2 t == BOOL && t' == type2 t3)
                             then t' else error "Untypable"
             where t' = type2 t2
     PAIR t1 t2 -> P (type2 t1) (type2 t2)
     FST (PAIR t1 t2) -> type2 t1
     SND (PAIR t1 t2) -> type2 t2
     _ -> error "Untypable"
```