# CIS 500

Software Foundations

Fall 2004

20/27 October



# Sums



## 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* |

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

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

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

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



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

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

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

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



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

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



## 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)* |



*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:* $\boxed{t \longrightarrow t'}$

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

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

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

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



## Variants

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

*New syntactic forms*

| | | | |
|---|---|---|---|
| $t$ | ::= | ... | *terms* |
| | | $\langle l{=}t \rangle$ as $T$ | *tagging* |
| | | case $t$ of $\langle l_i{=}x_i \rangle \Rightarrow t_i \ ^{i \in 1..n}$ | *case* |
| $T$ | ::= | ... | *types* |
| | | $\langle l_i{:}T_i \ ^{i \in 1..n} \rangle$ | *type of variants* |



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

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

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

$$\frac{t_i \longrightarrow t_i'}{\langle l_i{=}t_i \rangle \text{ as } T \longrightarrow \langle l_i{=}t_i' \rangle \text{ as } T} \qquad \text{(E-Variant)}$$



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

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

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



## 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;
```



## Terminology: "Union Types"

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

(We could also consider a non-disjoint union $T_1 \vee T_2$, but its properties are substantially more complex, because it induces an interesting subtype relation. We'll come back to subtyping later.)



## 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*

| | | | |
|---|---|---|---|
| $t$ | ::= | ... | *terms* |
| | | $\texttt{fix}\ t$ | *fixed point of* $t$ |

*New evaluation rules*

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

$$\frac{}{\texttt{fix}\ (\lambda x{:}T_1.t_2) \longrightarrow [x \mapsto (\texttt{fix}\ (\lambda x{:}T_1.t_2))]t_2} \quad \text{(E-FixBeta)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{fix}\ t_1 \longrightarrow \texttt{fix}\ t_1'} \quad \text{(E-Fix)}$$



*New typing rules*

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

$$\frac{\Gamma \vdash t_1 : T_1 \to T_1}{\Gamma \vdash \texttt{fix}\ t_1 : T_1} \quad \text{(T-Fix)}$$

## A more convenient form

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

```
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;
```



# Lists



## Lists — syntax

| | | | |
|---|---|---|---|
| t | ::= | ... | *terms* |
| | | `nil[T]` | *empty list* |
| | | `cons[T] t t` | *list constructor* |
| | | `isnil[T] t` | *test for empty list* |
| | | `head[T] t` | *head of a list* |
| | | `tail[T] t` | *tail of a list* |
| v | ::= | ... | *values* |
| | | `nil[T]` | *empty list* |
| | | `cons[T] v v` | *list constructor* |
| T | ::= | ... | *types* |
| | | `List T` | *type of lists* |



## Lists — evaluation

$$\frac{t_1 \longrightarrow t_1'}{\texttt{cons[T] } t_1\ t_2 \longrightarrow \texttt{cons[T] } t_1'\ t_2} \quad \text{(E-Cons1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\texttt{cons[T] } v_1\ t_2 \longrightarrow \texttt{cons[T] } v_1\ t_2'} \quad \text{(E-Cons2)}$$

$$\texttt{isnil[S] (nil[T])} \longrightarrow \texttt{true} \quad \text{(E-IsnilNil)}$$

$$\texttt{isnil[S] (cons[T] } v_1\ v_2\texttt{)} \longrightarrow \texttt{false} \quad \text{(E-IsnilCons)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{isnil[T] } t_1 \longrightarrow \texttt{isnil[T] } t_1'} \quad \text{(E-Isnil)}$$

$$\texttt{head[S]}\ (\texttt{cons[T]}\ v_1\ v_2) \longrightarrow v_1 \qquad \text{(E-HeadCons)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{head[T]}\ t_1 \longrightarrow \texttt{head[T]}\ t_1'} \qquad \text{(E-Head)}$$

$$\texttt{tail[S]}\ (\texttt{cons[T]}\ v_1\ v_2) \longrightarrow v_2 \qquad \text{(E-TailCons)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{tail[T]}\ t_1 \longrightarrow \texttt{tail[T]}\ t_1'} \qquad \text{(E-Tail)}$$

Note that evaluation rules do not look at type annotations!



## Lists — typing

$$\Gamma \vdash \texttt{nil[}T_1\texttt{]} : \texttt{List}\ T_1 \qquad \text{(T-Nil)}$$

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : \texttt{List}\ T_1}{\Gamma \vdash \texttt{cons[}T_1\texttt{]}\ t_1\ t_2 : \texttt{List}\ T_1} \qquad \text{(T-Cons)}$$

$$\frac{\Gamma \vdash t_1 : \texttt{List}\ T_{11}}{\Gamma \vdash \texttt{isnil[}T_{11}\texttt{]}\ t_1 : \texttt{Bool}} \qquad \text{(T-Isnil)}$$

$$\frac{\Gamma \vdash t_1 : \texttt{List}\ T_{11}}{\Gamma \vdash \texttt{head[}T_{11}\texttt{]}\ t_1 : T_{11}} \qquad \text{(T-Head)}$$

$$\frac{\Gamma \vdash t_1 : \texttt{List}\ T_{11}}{\Gamma \vdash \texttt{tail[}T_{11}\texttt{]}\ t_1 : \texttt{List}\ T_{11}} \qquad \text{(T-Tail)}$$