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 $\boxed{t \longrightarrow t'}$

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

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

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

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

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

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

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

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

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

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

Variants

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

New syntactic forms

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

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

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

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

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

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

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

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

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 $\lambda_{\rightarrow}$

- In $\lambda_{\rightarrow}$, 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

$\mathtt{t}$	::=	...	*terms*
		$\mathtt{fix\ t}$	*fixed point of* $\mathtt{t}$

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

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

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

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

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

A more convenient form

$$\texttt{letrec x:T}_1\texttt{=t}_1\texttt{ in t}_2 \stackrel{\text{def}}{=} \texttt{let x = fix (}\lambda\texttt{x:T}_1\texttt{.t}_1\texttt{) in t}_2$$

```
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

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{cons[T] t}_1\texttt{ t}_2 \longrightarrow \texttt{cons[T] t}_1'\texttt{ t}_2} \quad \text{(E-Cons1)}$$

$$\frac{\texttt{t}_2 \longrightarrow \texttt{t}_2'}{\texttt{cons[T] v}_1\texttt{ t}_2 \longrightarrow \texttt{cons[T] v}_1\texttt{ t}_2'} \quad \text{(E-Cons2)}$$

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

$$\texttt{isnil[S] (cons[T] v}_1\texttt{ v}_2\texttt{)} \longrightarrow \texttt{false} \quad \text{(E-IsnilCons)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{isnil[T] t}_1 \longrightarrow \texttt{isnil[T] t}_1'} \quad \text{(E-Isnil)}$$

$$\texttt{head[S] (cons[T] v}_1\texttt{ v}_2\texttt{)} \longrightarrow \texttt{v}_1 \qquad \text{(E-HeadCons)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{head[T] t}_1 \longrightarrow \texttt{head[T] t}_1'} \qquad \text{(E-Head)}$$

$$\texttt{tail[S] (cons[T] v}_1\texttt{ v}_2\texttt{)} \longrightarrow \texttt{v}_2 \qquad \text{(E-TailCons)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{tail[T] t}_1 \longrightarrow \texttt{tail[T] t}_1'} \qquad \text{(E-Tail)}$$

Note that evaluation rules do not look at type annotations!

Lists — typing

$$\Gamma \vdash \texttt{nil[T}_1\texttt{] : List T}_1 \qquad \text{(T-Nil)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1 \qquad \Gamma \vdash \texttt{t}_2 : \texttt{List T}_1}{\Gamma \vdash \texttt{cons[T}_1\texttt{] t}_1\texttt{ t}_2 : \texttt{List T}_1} \qquad \text{(T-Cons)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{List T}_{11}}{\Gamma \vdash \texttt{isnil[T}_{11}\texttt{] t}_1 : \texttt{Bool}} \qquad \text{(T-Isnil)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{List T}_{11}}{\Gamma \vdash \texttt{head[T}_{11}\texttt{] t}_1 : \texttt{T}_{11}} \qquad \text{(T-Head)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{List T}_{11}}{\Gamma \vdash \texttt{tail[T}_{11}\texttt{] t}_1 : \texttt{List T}_{11}} \qquad \text{(T-Tail)}$$