CIS 500

Software Foundations

Fall 2003

20-22 October

Sums

Sums - motivating example

```
PhysicalAddr = {firstlast:String, addr:String}
VirtualAddr  = {name:String, email:String}
Addr         = PhysicalAddr + VirtualAddr
```

inl : "PhysicalAddr $\longrightarrow$ PhysicalAddr+VirtualAddr"

inr : "VirtualAddr $\longrightarrow$ 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

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

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

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

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

New evaluation rules

$\boxed{t \longrightarrow t'}$

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1}}{\Gamma \vdash \mathtt{inl}\ \mathtt{t_1}\ \mathtt{as}\ \mathtt{T_1}+\mathtt{T_2} : \mathtt{T_1}+\mathtt{T_2}} \qquad \text{(T-INL)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_2}}{\Gamma \vdash \mathtt{inr}\ \mathtt{t_1}\ \mathtt{as}\ \mathtt{T_1}+\mathtt{T_2} : \mathtt{T_1}+\mathtt{T_2}} \qquad \text{(T-INR)}$$

Evaluation rules ignore annotations:

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

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

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

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

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

Variants

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

New syntactic forms

$\texttt{t}$	::=	...	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

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

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

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

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

New typing rules

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

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

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

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 t

New evaluation rules

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

$$\begin{array}{l}\mathtt{fix\ (\lambda x\!:\!T_1.t_2)} \\ \longrightarrow [\mathtt{x} \mapsto \mathtt{(fix\ (\lambda x\!:\!T_1.t_2))}]\mathtt{t_2}\end{array} \qquad \text{(E-FixBeta)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{fix\ t_1} \longrightarrow \mathtt{fix\ t_1'}} \qquad \text{(E-Fix)}$$

New typing rules

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

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

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

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

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

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

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

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

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

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

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

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

Note that evaluation rules do not look at type annotations!

Lists — typing

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

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

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

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

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