**Sums**

New recursive forms

\[ T \equiv T_1 ]

\[ \cdots \equiv \cdots \]

\[ 1 \]

\[ \text{tagging(left)} \]

\[ \text{tagging(right)} \]

\[ \text{taggedvalue(left)} \]

\[ \text{taggedvalue(right)} \]

\[ \text{sum type} \]

\[ \text{cases} \]

\[ \text{values} \]

\[ \text{recursive forms} \]

**Sums - motivating example**

```plaintext
getName = \text{a} : \text{Addr}. \\
\text{case} \text{a} \text{of} \text{inlx}) \text{x} . \text{firstlast} | \text{inry}) \text{y} . \text{name};
```

```plaintext
\text{name} \leftrightarrow \text{VirtualAddr + PhysicalAddr} \\
\text{addr} = \text{PhysicalAddr + VirtualAddr} \\
\text{virtualAddr} = \{ \text{name} : \text{String}, \text{email} : \text{String}, \text{addr} : \text{String}, \text{firstLast} : \text{String} \};
```

2/27 October

Fall 2004

Software Foundations

CIS 500
For simplicity, let's choose the third.
- Announce each \texttt{inl} and \texttt{inr} with the intended sum type.

\textbf{OCaml's solution:}

\textbf{Possible solutions:}

\begin{itemize}
  \item Either, we've lost uniqueness of types.
  \item If \texttt{t} has type \texttt{T}, then \texttt{inl \ t} has type \texttt{T+U} for every \texttt{U}.
\end{itemize}

\begin{itemize}
  \item \texttt{inl} and \texttt{inr} with the intended sum type.
\end{itemize}

\textbf{Sums and Uniqueness of Types:}

Problem:

\textbf{I.e., we've lost uniqueness of types.}

Possibilities:

\textbf{Infer} \texttt{U} as needed during typechecking

\begin{itemize}
  \item Give constructors different names and only allow each name to appear in
  \begin{itemize}
    \item \texttt{inl} and \texttt{inr} as needed during typechecking
  \end{itemize}
\end{itemize}

\textbf{OCaml's solution:}

\begin{itemize}
  \item Annotate each \texttt{inl} and \texttt{inr} with the intended sum type.
\end{itemize}

\textbf{Forsimplicity, let's choose the third.}
Newsyntactic forms

\[ t ::= \ldots \text{terms} \]

inlt\(\text{left}\) tagging

\[ v ::= \ldots \text{values} \]

inl\(\text{left}\) taggedvalue

\[ \text{Evaluation rules ignore annotations:} \]

\[ \begin{align*}
\text{E-Inl} & : t_1 : T_1, t_2 : T_2 \quad \text{inlt}\(\text{left}\)_\{\text{left}\} \\
& \quad \text{inlt}\(\text{left}\)_\{\text{right}\} \\
& \quad t_1 \vdash t_2 \\
\end{align*} \]

\[ \begin{align*}
\text{E-Inr} & : t_1 : T_1, t_2 : T_2 \quad \text{inrt}\(\text{left}\)_\{\text{left}\} \\
& \quad \text{inrt}\(\text{left}\)_\{\text{right}\} \\
& \quad t_1 \vdash t_2 \\
\end{align*} \]

\[ \begin{align*}
\text{E-CaseInl} & : \text{case (inlv}_{0} \text{ as T}_{0}) \\
& \quad \text{case (inlx}_{1} \text{ as T}_{1}) \\
& \quad t_1 \mid \text{inrx}_{2} \text{ as T}_{2} \\
& \quad t_2 \\
& \quad \text{E-CaseInr} \\
\end{align*} \]

\[ \begin{align*}
\text{E-CaseInl} & : \text{case (inrv}_{0} \text{ as T}_{0}) \\
& \quad \text{case (inlx}_{1} \text{ as T}_{1}) \\
& \quad t_1 \mid \text{inrx}_{2} \text{ as T}_{2} \\
& \quad t_2 \\
& \quad \text{E-CaseInr} \\
\end{align*} \]

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

\begin{align*}
\text{variants}
\end{align*}
Newsyntacticforms
\[ \text{terms} <l=t> \text{asT} \]
\[ \text{case a of} \]
\[ \text{getname} = \text{addr} \]
\[ a = \text{opbytes-addr} \text{ as addr} \]
\[ \text{addr} = \text{opbytes-addr} \text{ of name:} \]

Example

\[ \text{Addr} = \text{physical:PhysicalAddr, virtual:VirtualAddr}; \]
\[ a = \text{physical=pa} \text{ as Addr}; \]
\[ \text{name} = a : \text{Addr}. \]
\[ \text{case a of} \]
\[ \text{physical=x} \text{ as Addr}; \]
\[ \text{addr}. \]

New evaluation rules

\[ t : T \]
\[ \forall \ \text{case } t \text{ of } x : T \text{ } \]
\[ \text{E-Case} \]
\[ \text{E-CaseVariant} \]

New typing rules

\[ t : T \]
\[ \forall \ \text{foreach } i; x_i : T_i \text{ } \]
\[ \text{caset} 0 \text{ of } i = x_i \]
\[ \text{T-Case} \]
\[ \text{T-Variant} \]

Example

\[ \text{Addr} = \text{physical:PhysicalAddr, virtual:VirtualAddr}; \]
\[ a = \text{physical=pa} \text{ as Addr}; \]
\[ \text{name} = a : \text{Addr}. \]
\[ \text{case a of} \]
\[ \text{physical=x} \text{ as Addr}; \]
\[ \text{addr}. \]
Reursion

OptionalNat = <none:Unit, some:Nat>
Table = Nat

extendTable =
  t: Table.
  m: Nat.
  v: Nat.
  n: Nat.
  if equalnm then <some=v> asOptionalNat else t

x = case (5) of
  <none=u> 999
  | <some=v> v

Terminology: "Union Types"

\[ T_1 \cup T_2 \]

\( T_1 \cup T_2 \) is a disjoint union of \( T_1 \) and \( T_2 \) (the laws \( \cup \) and \( \cap \) give

Enumerations

Weekday = <monday: Unit, tuesday: Unit, wednesday: Unit, thursday: Unit, friday: Unit>

nextBusinessDay =
  w: Weekday.
  case w of <monday=x>
    <tuesday=unit> asWeekday
  | <tuesday=x>
    <wednesday=unit> asWeekday
  | <thursday=x>
    <friday=unit> asWeekday
  | <friday=x>
    <monday=unit> asWeekday

Recursion
Recursion in \( \lambda \)

Example

Recursion in \( \lambda \)
Lists | Syntax

- nil[T] ::=
- cons[T] v t ::=
- isnil[T] t ::=
- head[T] t ::=
- tail[T] t ::=

Lists | Evaluation

- E-IsnilNil
- E-IsnilCons
- E-Isnil

A more convenient form
Note that evaluation rules do not look at type annotations!