COMPSCI 3MI3 - Principles of Programming Languages

Data-Structures

J. Carette

McMaster University

Fall 2023

Adapted from “Types and Programming Languages” by Benjamin C. Pierce

Adding data-structures: pairs

$$
\begin{aligned}
\langle t \rangle ::= \ & \ldots \\
| \quad & \{\mathrm{t},\ \mathrm{t}\} \\
| \quad & \{\mathrm{t},\ \mathrm{t}\}.1 \\
| \quad & \{\mathrm{t},\ \mathrm{t}\}.2
\end{aligned}
\qquad\qquad
\begin{aligned}
\langle v \rangle ::= \ & \ldots \\
| \quad & \{\mathrm{v},\ \mathrm{v}\}
\end{aligned}
$$

$$\frac{t \to t'}{t.1 \to t'.1} \quad \text{(E-Proj1)}$$

$$\frac{t \to t'}{t.2 \to t'.2} \quad \text{(E-Proj2)}$$

$$\frac{t_1 \to t_1'}{\{t_1, t_2\} \to \{t_1', t_2\}} \quad \text{(E-Pair1)}$$

$$\frac{t_2 \to t_2'}{\{v_1, t_2\} \to \{v_1, t_2'\}} \quad \text{(E-Pair2)}$$

$$\{v_1, v_2\}.1 \to v_1 \quad \text{(E-PairBeta1)}$$

$$\{v_1, v_2\}.2 \to v_2 \quad \text{(E-PairBeta2)}$$

Typing Pairs

$\langle T \rangle$ ::= ...
$\quad | \quad \langle T \rangle \times \langle T \rangle$

This is known as the **product** or the **Cartesian Product** type constructor.

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \qquad \text{(T-Pair)}$$

$$\frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash t.1 : T_1} \qquad \text{(T-Proj1)}$$

$$\frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash t.2 : T_2} \qquad \text{(T-Proj2)}$$

Tuples: from 2 to n

$$\begin{array}{lll} \langle t \rangle & ::= & \ldots \\ & | & \{\langle t \rangle, \langle t \rangle, \ldots, \langle t \rangle\} \\ & | & \text{t.i} \end{array}$$

where there are n terms in the first case, and $1 \leq i \leq n$ in the second.

$$\begin{array}{lll} \langle v \rangle & ::= & \ldots \\ & | & \{\langle v \rangle, \langle v \rangle, \ldots, \langle v \rangle\} \end{array}$$

$$\begin{array}{lll} \langle T \rangle & ::= & \ldots \\ & | & \{\langle T \rangle \times \langle T \rangle \times \ldots \times \langle T \rangle \} \end{array}$$

As this $\ldots$ notation can get tiresome, we use $\vec{t}$, $\vec{v}$ and $\vec{T}$.

Evaluation Rules

$$\frac{j \in 1..n}{\{\vec{v}\}.j \to v_j} \qquad \text{(E-ProjTuple)}$$

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

$$\frac{t_j \to t_j'}{\{v_1, v_2, \ldots, v_{j-1}, t_j, \ldots t_n\} \to \{v_1, v_2, \ldots, v_{j-1}, t_j', \ldots t_n\}} \qquad \text{(E-Tuple)}$$

Tuping Typles

$$\frac{\Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2 \quad \ldots \Gamma \vdash t_n : T_n}{\Gamma \vdash \{\vec{t}\} : \{\vec{T}\}} \tag{T-Tuple}$$

$$\frac{j \in 1..n \qquad \Gamma \vdash t : \{\vec{T}\}}{\Gamma \vdash t.j : T_j} \tag{T-Proj}$$

Record

Numbers are silly labels, let's use names as **labels**. $l \in \mathcal{L}$.

$$\begin{array}{lll}\langle t \rangle & ::= & \ldots \\ & \mid & \{\langle l \rangle = \langle t \rangle, \langle l \rangle = \langle t \rangle, \ldots, \langle l \rangle = \langle t \rangle\ \} \\ & \mid & \langle t \rangle . \langle l \rangle \end{array}$$

$$\begin{array}{lll}\langle v \rangle & ::= & \ldots \\ & \mid & \{\langle l \rangle = \langle v \rangle, \langle l \rangle = \langle v \rangle, \ldots, \langle l \rangle = \langle v \rangle\ \} \end{array}$$

$$\begin{array}{lll}\langle T \rangle & ::= & \ldots \\ & \mid & \{\langle l \rangle : \langle T \rangle, \langle l \rangle : \langle T \rangle, \ldots, \langle l \rangle : \langle T \rangle\ \} \end{array}$$

structs in C, object with only fields in Java, dictionaries (sort of) in Python

Evaluation Rules

$$\frac{j \in 1..n}{\{\overrightarrow{l = v}\}.l_j \to v_j} \qquad \text{(E-ProjRcd)}$$

$$\frac{t \to t'}{t.l_i \to t'.l_i} \qquad \text{(E-Proj)}$$

$$\frac{t_j \to t_j'}{\begin{array}{c}\{l_1 = v_1, \ldots, l_{j-1} = v_{j-1}, l_j = t_j, \ldots l_n = t_n\} \to \\ \{l_1 = v_1, \ldots, l_{j-1} = v_{j-1}, l_j = t_j', \ldots l_n = t_n\}\end{array}} \qquad \text{(E-Rcd)}$$

Note: order of labels is induced by the language somehow. Usually at type declaration time.

Typing

$$\frac{\Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2 \quad \ldots \Gamma \vdash t_n : T_n}{\Gamma \vdash \{\overrightarrow{f = t}\} : \{\overrightarrow{f : T}\}} \qquad \text{(T-Tuple)}$$

$$\frac{j \in 1..n \qquad \Gamma \vdash t : \{\overrightarrow{T}\}}{\Gamma \vdash t.j : T_j} \qquad \text{(T-Proj)}$$

Pattern Matching (for records)

The often forgotten programming language "on the left":

```
snd :: (a, b) -> b
snd (x, y) = y
```

Pattern Matching (for records)

The often forgotten programming language "on the left":

```
snd :: (a, b) -> b
snd (x, y) = y
```

Do it as a generalization of **let bindings**.

$\langle t \rangle ::= \ldots$
$\quad | \quad \text{let } \langle p \rangle = \langle t \rangle \text{ in } \langle t \rangle$

New syntactic category: patterns.

$\langle p \rangle ::= \langle x \rangle$
$\quad | \quad \{\langle f \rangle = \langle p \rangle, \langle f \rangle = \langle p \rangle, \ldots, \langle f \rangle = \langle p \rangle\}$

Record Patterns

Looking at the rules in more detail:

$$\texttt{let}\ \ p = v\ \ \texttt{in}\ \ t \to \mathit{match}(p, v)t \qquad \text{(E-LetV)}$$

$$\frac{t_1 \to t_1'}{\texttt{let}\ \ p = t_1\ \ \texttt{in}\ \ t_2 \to \texttt{let}\ \ p = t_1'\ \ \texttt{in}\ \ t_2} \qquad \text{(E-Let)}$$

The *match* function **creates substitutions**.

Specifying match

$$match(x, v) = [x \mapsto v] \tag{M-Var}$$

$$\frac{\forall i \in 1..n \mid match(p_i, v_i) = \sigma_i}{\{match(\overrightarrow{f = p}\}, \{\overrightarrow{f = v}\} = \sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_n} \tag{M-Rcd}$$

Where $\circ$ is function composition.

Examples I

$$
\begin{aligned}
& \texttt{let}\ \{x, y, z\} = \{2, 4, 6\}\ \texttt{in}\ (x \cdot z) - y \\
\xrightarrow{\texttt{E-LetV}}\ & match(\{x, y, z\}, \{2, 4, 6\})((x \cdot z) - y) \\
\xrightarrow{\texttt{M-Rcd}}\ & match(x, 2) \circ match(y, 4) \circ match(z, 6)((x \cdot z) - y) \\
\xrightarrow{\texttt{M-Var}}\xrightarrow{\texttt{M-Var}}\xrightarrow{\texttt{M-Var}}\ & [x \mapsto 2][y \mapsto 4][z \mapsto 6]((x \cdot z) - y) \\
\xrightarrow{\texttt{Subst}}\xrightarrow{\texttt{Subst}}\xrightarrow{\texttt{Subst}}\ & (2 \cdot 6) - 4 \\
\xrightarrow{\texttt{Arith}}\xrightarrow{\texttt{Arith}}\ & 8 \\
\not\to\ &
\end{aligned}
$$

Examples II

$$
\begin{aligned}
& \texttt{let}\ \{x, y\} = \{2, \{4, 6\}\}\ \texttt{in}\ ((\lambda t.\lambda f.f)\, x\, y) \\
\xrightarrow{\texttt{E-LetV}}\ & \mathit{match}(\{x, y\}, \{2, \{4, 6\}\})((\lambda t.\lambda f.f)\, x\, y) \\
\xrightarrow{\texttt{M-Rcd}}\ & \mathit{match}(x, 2) \circ \mathit{match}(y, \{4, 6\})((\lambda t.\lambda f.f)\, x\, y) \\
\xrightarrow{\texttt{M-Var}}\xrightarrow{\texttt{M-Var}}\ & [x \mapsto 2][y \mapsto \{4, 6\}]((\lambda t.\lambda f.f)\, x\, y) \\
\xrightarrow{\texttt{Subst}}\xrightarrow{\texttt{Subst}}\ & (\lambda t.\lambda f.f)\, 2\, \{4, 6\} \\
\xrightarrow{\texttt{E-AppAbs}}\ & (\lambda f.f)\, \{4, 6\} \\
\xrightarrow{\texttt{E-AppAbs}}\ & \{4, 6\} \\
\not\rightarrow\ &
\end{aligned}
$$

Lists

More precisely: uniformly typed (linked) lists.

Like λ abstractions, each of our list terms will require **type annotation**.

$\langle t \rangle$ :: ...
  | nil[$\langle T \rangle$]
  | cons[$\langle T \rangle$] $\langle t \rangle$ $\langle t \rangle$
  | isnil[$\langle T \rangle$] $\langle t \rangle$

Both empty lists and lists containing only values will be values themselves.

$\langle v \rangle$ ::= ...
  | nil[$\langle T \rangle$]
  | cons[$\langle T \rangle$] $\langle v \rangle$ $\langle v \rangle$

Evaluating Cons

- In the same way that `true` has no evaluation rules, `nil` has no evaluation rules.
- Since `cons` is a *constructor*, we only have two congruence rules for it:

$$\frac{t_1 \to t_1'}{cons[T]\ t_1\ t_2 \to cons[T]\ t_1'\ t_2} \qquad \text{(E-Cons1)}$$

$$\frac{t_2 \to t_2'}{cons[T]\ v_1\ t_2 \to cons[T]\ v_1\ t_2'} \qquad \text{(E-Cons2)}$$

Evaluating `isnil`

`isnil` is very much like `iszero`:

$$\mathit{isnil}[S](\mathit{nil}[T]) \to \mathit{true} \qquad \text{(E-IsNilNil)}$$

$$\mathit{isnil}[S](\mathit{cons}[T]\,v_1\,v_2) \to \mathit{false} \qquad \text{(E-IsNilCons)}$$

$$\frac{t_1 \to t_1'}{\mathit{isnil}[T]\,t_1 \to \mathit{isnil}[T]\,t_1'} \qquad \text{(E-IsNil)}$$

Typing Lists

$$\begin{array}{lcl} \langle T \rangle & ::= & \dots \\ & | & \text{List } \langle T \rangle \end{array}$$

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

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

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