Overview

Modern programming languages include more interesting, and more complex, data types than 

num and str.

To analyze such features, we've started with these types:

- Product, or tuple, types.
- Sum, or disjoint union, types.

Today we consider $L\{\text{pat}\}$, a simple language of pattern matching over eager product and sum types.

Pattern Matching

As developed thus far, product and sum types are not easily used for computing. For instance, to add two components of a pair of num values requires:

```plaintext
let x be e in pr1(x) + pr2(x)
```

Using pattern matching, we could instead use:

```plaintext
match e {x, y. ⟨x, y⟩ ⇒ x + y}
```

Expression $e$ is matched against pattern $⟨x, y⟩$ binding variables $x$ and $y$, which are then used in expression $x + y$.

Pattern Matching

A more interesting example, involving both products and sums:

```plaintext
match e {x. ⟨in[l](⟨⟩), x⟩ ⇒ x + x |
y. ⟨in[r](⟨⟩), y⟩ ⇒ y * y}
```

(How would this be expressed without matching?)

Comparing a match value with a (finite) sequence of rules, each of which consists of a pattern and an expression.

Matching a rule’s pattern binds some set of variables, and those bindings may be used in evaluating the rule’s expression.

$L\{\text{pat}\}$: Abstract and Concrete Syntax

<table>
<thead>
<tr>
<th>Category</th>
<th>Item</th>
<th>Abstract</th>
<th>Concrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expr</td>
<td>$e$</td>
<td>$\text{match}(e; rs)$</td>
<td>$\text{match}(rs)$</td>
</tr>
<tr>
<td>Rules</td>
<td>$rs$</td>
<td>$\text{rules}(x; r_1, \ldots, r_n)$</td>
<td>$r_1, \ldots, r_n$</td>
</tr>
<tr>
<td>Rule</td>
<td>$r$</td>
<td>$x_1, \ldots, x_k \ \text{rule}(p; e)$</td>
<td>$x_1, \ldots, x_k \ p \Rightarrow e$</td>
</tr>
<tr>
<td>Pattern</td>
<td>$p$</td>
<td>wild</td>
<td>$x$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{pair}(p_1; p_2)$</td>
<td>$p_1, p_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{in}<a href="p">l</a>$</td>
<td>$\text{in}<a href="p">l</a>$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{in}<a href="p">r</a>$</td>
<td>$\text{in}<a href="p">r</a>$</td>
</tr>
</tbody>
</table>

Operator $\text{rules}(n; r_1, \ldots, r_n)$ has arity $(k_1, \ldots, k_0)$ where each rule $r_i$ has valence $k_i \geq 0$.

$L\{\text{pat}\}$: Static Semantics

Using linear hypothetical judgements, of form:

$x_1 : \tau_1, \ldots, x_k : \tau_k \vdash p : \sigma$

Similar to ordinary hypothetical judgement:

$x_1 : \tau_1, \ldots, x_k : \tau_k \vdash p : \sigma$

but forces variables to be used exactly once in any pattern.

Usual rules of weakening and contraction are dropped, and combinations of sets of assumptions $\Lambda_1$ and $\Lambda_2$ require that they be disjoint, written $\Lambda_1 \# \Lambda_2$. 
$\mathcal{L}[\text{pat}]: \text{Static Semantics}$

\[ \begin{align*}
\Gamma \vdash x : \tau & \quad \text{if} \quad \Gamma \vdash x
\end{align*} \]

\[ \begin{align*}
\Gamma \vdash \bot : \tau & \quad \text{if} \quad \Gamma \vdash \bot
\end{align*} \]

\[ \begin{align*}
\Gamma \vdash \text{unit} : \tau & \quad \text{if} \quad \Gamma \vdash \text{unit}
\end{align*} \]

\[ \begin{align*}
\vdash p : \tau \quad \text{if} \quad \Gamma \vdash p \quad \text{for} \quad \Gamma \vdash \tau
\end{align*} \]

\[ \begin{align*}
A_1, A_2 \vdash p : \tau_1 \quad A_3 \vdash p : \tau_2
\end{align*} \]

\[ \begin{align*}
A_1 \parallel A_2 \parallel A_3 \vdash \tau_1 \times \tau_2
\end{align*} \]

\[ \begin{align*}
\Gamma \vdash \text{match} \langle \tau \rangle : \tau
\end{align*} \]

\[ \begin{align*}
\vdash \text{match} \langle \tau \rangle : \tau
\end{align*} \]

\[ \begin{align*}
\vdash \text{match} \langle \tau \rangle : \tau
\end{align*} \]

$\mathcal{L}[\text{pat}]: \text{Dynamic Semantics}$

\[ \begin{align*}
\text{match}(\tau[\alpha]) & \rightarrow \text{match}(\tau'[\alpha])
\end{align*} \]

Checked condition of pattern match failure.

\[ \begin{align*}
e \dagger \text{val} \quad e \dagger \text{val} \rightarrow [e_1, \ldots, e_k]p_0 = e
\end{align*} \]

\[ \begin{align*}
\text{match}(e_1, \ldots, e_k, p_0) = e_0 \rightarrow [e_1, \ldots, e_k]p_0 = e_0
\end{align*} \]

Appropriate substitution is “guessed”.

\[ \begin{align*}
\vdash \text{match} \langle \tau \rangle : \tau
\end{align*} \]

Rules checked left to right until all rules exhausted.

$\mathcal{L}[\text{pat}]: \text{Pattern Matching Judgements}$

As defined thus far, semantics doesn’t say how to find a pattern match, or to determine that none exists.

Rectifying this involves two additional judgements. The first:

\[ \begin{align*}
x_k < e_1, \ldots, e_k \dagger \vdash p < e
\end{align*} \]

A linear hypothetical judgement where $e$ val and $e_i$ val for $1 \leq i \leq k$ says $[e_1, \ldots, e_k, x_1, \ldots, x_k]p = e$.

The second:

\[ \begin{align*}
e \perp p
\end{align*} \]

where $e$ val says that $e$ fails to match pattern $p$.

$\mathcal{L}[\text{pat}]: \text{Preservation}$

Theorem 1

If $e : \tau$ and $e \rightarrow e'$, then $e' : \tau$.

By induction on evaluation.

A progress theorem could be proved, but not very interesting since it couldn’t rule out failure. More on this later.

$\mathcal{L}[\text{pat}]: \text{Pattern Matching Judgements}$

Writing $\Theta$ for assumptions governing variables:

\[ \begin{align*}
\vdash \Theta \vdash \emptyset \rightarrow \emptyset
\end{align*} \]

\[ \begin{align*}
\vdash \Theta \vdash \emptyset
\end{align*} \]

\[ \begin{align*}
\Theta_1, \Theta_2 \vdash p_1 < e_1, \ldots, p_k < e_k
\end{align*} \]

\[ \begin{align*}
\Theta_1 \parallel \Theta_2 \vdash p_1 < e_1, \ldots, p_k < e_k
\end{align*} \]

\[ \begin{align*}
\vdash \Theta \vdash p < e
\end{align*} \]

\[ \begin{align*}
\vdash \Theta \vdash p < e
\end{align*} \]

\[ \begin{align*}
\vdash \Theta \vdash \text{in}(\tau)(p) < \text{in}(\tau)(e)
\end{align*} \]

\[ \begin{align*}
\vdash \Theta \vdash p < e
\end{align*} \]

\[ \begin{align*}
\vdash \Theta \vdash \text{in}(\tau)(p) < \text{in}(\tau)(e)
\end{align*} \]
L[pat]: Pattern Mismatching Judgements

\[
\begin{align*}
\text{\textit{e}} & \perp \textit{p} \\
\textit{p} & \perp \textit{e} \\
\textit{in}[l](\textit{e}) & \perp \textit{in}[r](\textit{p}) \\
\textit{e} & \perp \textit{p} \\
\textit{in}[l](\textit{e}) & \perp \textit{in}[r](\textit{p}) \nonumber
\end{align*}
\]

\[
\begin{align*}
\text{\textit{e}} & \perp \textit{p} \\
\textit{in}[l](\textit{e}) & \perp \textit{in}[r](\textit{p}) \\
\textit{e} & \perp \textit{p} \\
\textit{in}[l](\textit{e}) & \perp \textit{in}[r](\textit{p}) \nonumber
\end{align*}
\]

\[\text{Variable wildcards and null tuples cannot mismatch any appropriately typed value} \]

\[\text{A pair can only mismatch due to a mismatch in one of its components} \]

\[\text{An injection can mismatch the opposite injection} \]

\[\text{An injection can mismatch the same injection if its argument mismatches the argument pattern} \]

\[\text{Importantly, the pattern matching and pattern mismatching judgements are complementary:} \]

\[\text{Theorem 2} \]

\[\text{Suppose that } e : \tau, e : \tau \text{ and } x_1 : \tau_1, \ldots, x_k : \tau_k \vdash p : \tau. \text{ Then either there exists } e_1, \ldots, e_k \text{ such that } x_1 < e_1, \ldots, x_k < e_k \vdash p < e \text{ or } e \perp p. \]

By induction on typing.

\[\text{Exhaustiveness and Redundancy} \]

\[\text{More realistic treatment of matching, but still not in position to prove an interesting progress theorem for } L[\text{pat}]. \text{ We need:} \]

\[\text{• Exhaustiveness of a sequence of rules: Every value of domain type must match some rule in sequence.} \]

\[\text{• Irredundancy of rules: Every rule in sequence matches at least one domain value matched by no previous rule in the sequence.} \]

We introduce a language of \textit{match conditions} that identify a subset of the closed values of a type.

### Match Conditions: Abstract and Concrete Syntax

<table>
<thead>
<tr>
<th>Category</th>
<th>Item</th>
<th>Abstract</th>
<th>Concrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cond ζ</td>
<td>::=</td>
<td>any[τ]</td>
<td>⊤[τ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\textit{in}[l](\text{sum}(\textit{r}_1, \textit{r}_2)))(\text{\textit{q}}_1)</td>
<td>(\textit{in}<a href="%5Ctext%7B%5Ctextit%7Bq%7D%7D_1">l</a>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\textit{in}[r](\text{sum}(\textit{r}_1, \textit{r}_2)))(\text{\textit{q}}_2)</td>
<td>(\textit{in}<a href="%5Ctext%7B%5Ctextit%7Bq%7D%7D_2">r</a>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\textit{triv}</td>
<td>()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\textit{pair}(\text{\textit{q}}_1);(\text{\textit{q}}_2)</td>
<td>(\text{\textit{q}}_1, \text{\textit{q}}_2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\textit{nil}(\text{\textit{q}}_1)</td>
<td>(\textit{r})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\textit{alt}(\textit{q}_1;\textit{q}_2)</td>
<td>(\textit{q}_1 \lor \textit{q}_2)</td>
</tr>
</tbody>
</table>

\[\text{Match Conditions} \]

The judgement $\textit{ζ} : \tau$, meaning that $\textit{ζ}$ constrains values of type $\tau$ is defined by the following rules:

\[\begin{align*}
\tau & : \tau \\
\textit{in}[l](\textit{q}_1) & : \tau \vdash \textit{ζ} \\
\textit{in}[r](\text{\textit{q}}_2) & : \tau \vdash \textit{ζ} \\
\textit{ζ} & : \tau
\end{align*}\]
Match Conditions

The judgement $\xi : \tau$, meaning that $\xi$ constrains values of type $\tau$ is (further) defined by the following rules:

\[
\begin{align*}
\frac{}{\emptyset : \text{unit}} & \\
\frac{\xi_1 : \tau_1 \quad \xi_2 : \tau_2}{(\xi_1, \xi_2) : \tau_1 \times \tau_2} & \quad \xi \vdash \tau
\end{align*}
\]

Satisfaction Judgement

For $\xi : \tau$, $\epsilon : \tau$ and $\epsilon$ val, the satisfaction judgement $\epsilon \models \xi$ is (further) defined by the following rules:

\[
\begin{align*}
\frac{\epsilon \models \xi_1 \quad \epsilon \models \xi_2}{\epsilon \models (\xi_1, \xi_2)} & \\
\frac{\epsilon \models \xi_1}{\epsilon \models \tau_1 \times \xi_2} & \quad \epsilon \vdash \tau
\end{align*}
\]

The entailment judgement $\xi_1 \models \xi_2$, where $\xi_1 : \tau$ and $\xi_2 : \tau$ is defined to hold iff $\epsilon \models \xi_1$ implies $\epsilon \models \xi_2$.

\[
\begin{align*}
\xi \vdash \tau & \\
\frac{}{\emptyset : \text{unit}} & \\
\frac{\xi_1 : \tau_1 \quad \xi_2 : \tau_2}{(\xi_1, \xi_2) : \tau_1 \times \tau_2} &
\end{align*}
\]

$\mathcal{L}($pat$)$: Augmented Static Semantics

Instrumenting the static semantics of $\mathcal{L}$(pat) to associate match conditions with patterns and rules allows specifying values that they may match.

This in turn enables us to ensure that the patterns and rules are both exhaustive and irredundant.

The judgement $\Gamma \vdash p : \tau(\xi)$ augments the judgement $\Gamma \vdash p : \tau$ with a match constraint characterizing the set of values of type $\tau$ matched by pattern $p$.

$\mathcal{L}($pat$)$: Augmented Static Semantics

Match constraint characterizes the rule’s pattern component

\[
\begin{align*}
\Gamma \vdash r_1 : \tau \supset \tau(\xi_1) \quad \ldots \quad \Gamma \vdash r_n : \tau \supset \tau(\xi_n)
\end{align*}
\]

Match constraint characterizes the values matched by some rule in the rule sequence and ensures irredundancy: each successive rule must match at least one value not matched by any preceding rule.
Guarantees exhaustiveness: third premise ensures every value of type \( \tau \) satisfies \( \zeta \) which characterizes the values matched by some rule in the sequence.

The constraints of the augmented static semantics are sufficient to ensure progress: no well-formed match expression can fail to match a value of the specified type.

**Theorem 3**

If \( e : \tau \), then either \( e \) \text{ val} is a value, or there exists \( e' \) such that 
\[
\begin{align*}
  e &\rightarrow e'.
\end{align*}
\]

Exhaustiveness can always be forced by adding a "default" rule like \( x.x \Rightarrow e_x \) where \( e_x \) takes some "graceful" action, e.g., raises an exception.