

CMPSCI 630: Programming Languages
**Toward More Realistic Languages –
 Pattern Matching**

Spring 2009
 (with thanks to Robert Harper)

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

1

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:

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

Using **pattern matching**, we could instead use:

```
match e{x, y, <x, y> => x+y}
```

Expression e is matched against pattern $\langle x, y \rangle$ binding variables x and y , which are then used in expression $x+y$

2

Pattern Matching

A more interesting example, involving both products and sums:

```
match e{x.in[1](<>), x} => x+x | y.<in[r](<>), y> => y*y}
```

(How would this be expressed without matching?)

Compares 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.

3

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

Category	Item	Abstract	Concrete
Expr	e	$::= \text{match}(e; rs)$	$\text{match } e\{rs\}$
Rules	rs	$::= \text{rules}[n](r_1; \dots; r_n)$	$r_1 \dots r_n$
Rule	r	$::= x_1, \dots, x_k.\text{rule}(p; e)$	$x_1, \dots, x_k.p \Rightarrow e$
Pattern	p	$::= \text{wild}$	$-$
		x	x
		triv	$\langle \rangle$
		$\text{pair}(p_1; p_2)$	$\langle p_1, p_2 \rangle$
		$\text{in}[1][\tau](p)$	$\text{in}[1](p)$
		$\text{in}[\tau][\tau](p)$	$\text{in}[\tau](p)$

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

4

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

Using **linear hypothetical judgements**, of form:

$$x_1 : \tau_1, \dots, x_k : \tau_k \Vdash p : \tau$$

Similar to ordinary hypothetical judgement:

$$x_1 : \tau_1, \dots, x_k : \tau_k \vdash p : \tau$$

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 Λ_1 and Λ_2 require that they be disjoint, written $\Lambda_1 \# \Lambda_2$.

5

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

$$\frac{x : \tau \Vdash x : \tau}{\emptyset \Vdash - : \tau}$$

$$\frac{}{\emptyset \Vdash \langle \rangle : \text{unit}}$$

$$\frac{\Lambda_1 \Vdash p_1 : \tau_1 \quad \Lambda_2 \Vdash p_2 : \tau_2 \quad \Lambda_1 \# \Lambda_2}{\Lambda_1 \Lambda_2 \Vdash \langle p_1, p_2 \rangle : \tau_1 \times \tau_2}$$

$$\frac{\Lambda_1 \Vdash p : \tau_1}{\Lambda_1 \Vdash \text{in}[1](p) : \tau_1 + \tau_2}$$

$$\frac{\Lambda_2 \Vdash p : \tau_2}{\Lambda_2 \Vdash \text{in}[\bar{1}](p) : \tau_1 + \tau_2}$$

6

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

$$\frac{\Lambda \Vdash p : \tau \quad \Gamma \vdash e : \tau' \quad \Lambda \equiv x_1 : \tau_1, \dots, x_k : \tau_k \quad \Gamma \# \Lambda}{\Gamma \vdash x_1, \dots, x_k.p \Rightarrow e : \tau > \tau'}$$

Rule with pattern of type τ binds variables x_i of types τ_i and yields result of type τ' .

$$\frac{\Gamma \vdash r_1 : \tau > \tau' \quad \dots \quad \Gamma \vdash r_n : \tau > \tau'}{\Gamma \vdash r_1 \mid \dots \mid r_n : \tau > \tau'}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash rs : \tau > \tau'}{\Gamma \vdash \text{match } e\{rs\} : \tau'}$$

7

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

$$\frac{e \mapsto e'}{\text{match } e\{rs\} \mapsto \text{match } e'\{rs\}}$$

$$\frac{}{\text{match } e\{\} \text{ err}}$$

Checked condition of pattern match failure.

$$\frac{e_1 \text{ val } \dots \quad e_k \text{ val } \quad [e_1, \dots, e_k/x_1, \dots, x_k]p_0 = e}{\text{match } e\{x_1, \dots, x_k.p_0 \Rightarrow e_0\} \mapsto [e_1, \dots, e_k/x_1, \dots, x_k]e_0}$$

Appropriate substitution is "guessed".

$$\frac{\neg \exists e_1, \dots, e_k. [e_1, \dots, e_k/x_1, \dots, x_k]p_0 = e \quad e \text{ val } \quad \text{match } e\{rs\} \mapsto e'}{\text{match } e\{x_1, \dots, x_k.p_0 \Rightarrow e_0\} \mapsto e'}$$

Rules checked left to right until all rules exhausted.

8

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

Theorem 1

If $e : \tau$ and $e \mapsto e'$, 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.

9

$\mathcal{L}\{\text{pat}\}$: 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:

$$x_1 \triangleleft e_1, \dots, x_k \triangleleft e_k \Vdash p \triangleleft e$$

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

The second:

$$e \perp p$$

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

10

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

Writing Θ for assumptions governing variables:

$$\frac{x \triangleleft e \Vdash x \triangleleft e}{\emptyset \Vdash - \triangleleft e}$$

$$\frac{}{\emptyset \Vdash \langle \rangle \triangleleft \langle \rangle}$$

$$\frac{\Theta_1 \Vdash p_1 \triangleleft e_1 \quad \Theta_2 \Vdash p_2 \triangleleft e_2 \quad \Theta_1 \# \Theta_2}{\Theta_1 \Theta_2 \Vdash \langle p_1, p_2 \rangle \triangleleft \langle e_1, e_2 \rangle}$$

$$\frac{\Theta \Vdash p \triangleleft e}{\Theta \Vdash \text{in}[1](p) \triangleleft \text{in}[1](e)}$$

$$\frac{\Theta \Vdash p \triangleleft e}{\Theta \Vdash \text{in}[\bar{1}](p) \triangleleft \text{in}[\bar{1}](e)}$$

11

$\mathcal{L}\{\text{pat}\}$: Pattern Mismatching Judgements

$$\frac{e_1 \perp p_1}{\langle e_1, e_2 \rangle \perp \langle p_1, p_2 \rangle}$$

$$\frac{e_2 \perp p_2}{\langle e_1, e_2 \rangle \perp \langle p_1, p_2 \rangle}$$

$$\frac{}{\text{in}[l](e) \perp \text{in}[r](p)}$$

$$\frac{e \perp p}{\text{in}[l](e) \perp \text{in}[l](p)}$$

$$\frac{}{\text{in}[r](e) \perp \text{in}[l](p)}$$

$$\frac{e \perp p}{\text{in}[r](e) \perp \text{in}[r](p)}$$

12

$\mathcal{L}\{\text{pat}\}$: Pattern Mismatching Judgements

- Variables, wildcards and null tuples cannot mismatch any appropriately typed value
- A pair can only mismatch due to a mismatch in one of its components
- An injection can mismatch the opposite injection
- An injection can mismatch the same injection if its argument mismatches the argument pattern

13

$\mathcal{L}\{\text{pat}\}$: Matching and Mismatching Judgements

Importantly, the pattern matching and pattern mismatching judgements are complementary:

Theorem 2

Suppose that $e : \tau$, e val and $x_1 : \tau_1, \dots, x_k : \tau_k \Vdash p : \tau$. Then either there exists e_1, \dots, e_k such that $x_1 \triangleleft e_1, \dots, x_k \triangleleft e_k \Vdash p \triangleleft e$ or $e \perp p$.

By induction on typing.

14

Exhaustiveness and Redundancy

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

- **Exhaustiveness** of a sequence of rules: Every value of domain type must match some rule in sequence.
- **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 **match conditions** that identify a subset of the closed values of a type.

15

Match Conditions: Abstract and Concrete Syntax

Category	Item	Abstract	Concrete
Cond	ζ	$\text{any}[\tau]$	\top_τ
		$\text{in}[l][\text{sum}(\tau_1; \tau_2)](\zeta_1)$	$\text{in}[l](\zeta_1)$
		$\text{in}[r][\text{sum}(\tau_1; \tau_2)](\zeta_2)$	$\text{in}[r](\zeta_2)$
		triv	$\langle \rangle$
		$\text{pair}(\zeta_1; \zeta_2)$	$\langle \zeta_1, \zeta_2 \rangle$
		$\text{nil}[\tau]$	\perp_τ
		$\text{alt}(\zeta_1; \zeta_2)$	$\zeta_1 \vee \zeta_2$

16

Match Conditions

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

$$\frac{}{\top_\tau : \tau}$$

$$\frac{\zeta_1 : \tau_1}{\text{in}[l](\zeta_1) : \tau_1 + \tau_2}$$

$$\frac{\zeta_1 : \tau_2}{\text{in}[r](\zeta_1) : \tau_1 + \tau_2}$$

17

Match Conditions

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

$$\frac{}{\langle \rangle : \text{unit}}$$

$$\frac{\zeta_1 : \tau_1 \quad \zeta_2 : \tau_2}{\langle \zeta_1, \zeta_2 \rangle : \tau_1 \times \tau_2}$$

$$\frac{}{\perp_\tau : \tau}$$

$$\frac{\zeta_1 : \tau_1 \quad \zeta_2 : \tau_2}{\zeta_1 \vee \zeta_2 : \tau}$$

18

Satisfaction Judgement

For $\zeta : \tau$, $e : \tau$ and e val, the **satisfaction** judgement $e \models \zeta$ is defined by the following rules:

$$\frac{}{e \models \perp_\tau}$$

$$\frac{e_1 \models \zeta_1}{\text{in}[1](e_1) \models \text{in}[1](\zeta_1)}$$

$$\frac{e_2 \models \zeta_2}{\text{in}[x](e_2) \models \text{in}[x](\zeta_2)}$$

$$\frac{}{\langle \rangle \models \langle \rangle}$$

19

Satisfaction Judgement

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

$$\frac{e_1 \models \zeta_1 \quad e_2 \models \zeta_2}{\langle e_1, e_2 \rangle \models \langle \zeta_1, \zeta_2 \rangle}$$

$$\frac{e \models \zeta_1}{e \models \zeta_1 \vee \zeta_2}$$

$$\frac{e \models \zeta_2}{e \models \zeta_1 \vee \zeta_2}$$

The **entailment** judgement $\zeta_1 \models \zeta_2$, where $\zeta_1 : \tau$ and $\zeta_2 : \tau$ is defined to hold iff $e \models \zeta_1$ implies $e \models \zeta_2$.

20

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

Instrumenting the static semantics of $\mathcal{L}\{\text{pat}\}$ 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 $\Lambda \Vdash p : \tau[\zeta]$ augments the judgement $\Lambda \Vdash p : \tau$ with a match constraint characterizing the set of values of type τ matched by pattern p .

21

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

$$\frac{}{x : \tau \Vdash x : \tau[\perp_\tau]}$$

$$\frac{}{\emptyset \Vdash _ : \tau[\perp_\tau]}$$

$$\frac{}{\emptyset \Vdash \langle \rangle : \text{unit}[\langle \rangle]}$$

$$\frac{\Lambda_1 \Vdash p : \tau_1[\zeta_1]}{\Lambda_1 \Vdash \text{in}[1](p) : \tau_1 + \tau_2[\text{in}[1](\zeta_1)]}$$

$$\frac{\Lambda_2 \Vdash p : \tau_2[\zeta_2]}{\Lambda_2 \Vdash \text{in}[x](p) : \tau_1 + \tau_2[\text{in}[x](\zeta_2)]}$$

$$\frac{\Lambda_1 \Vdash p_1 : \tau_1[\zeta_1] \quad \Lambda_2 \Vdash p_2 : \tau_2[\zeta_2] \quad \Lambda_1 \# \Lambda_2}{\Lambda_1 \Lambda_2 \Vdash (p_1, p_2) : \tau_1 \times \tau_2[\langle \zeta_1, \zeta_2 \rangle]}$$

22

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

$$\frac{\Lambda \Vdash p : \tau[\zeta] \quad \Gamma \Vdash e : \tau'}{\Gamma \Vdash x_1, \dots, x_k.p \Rightarrow e : \tau > \tau'[\zeta]}$$

Match constraint characterizes the rule's pattern component

$$\frac{\Gamma \Vdash r_1 : \tau > \tau'[\zeta_1] \quad \dots \quad \Gamma \Vdash r_n : \tau > \tau'[\zeta_n]}{\Gamma \Vdash r_1 \dots | r_n : \tau > \tau'[\zeta_1 \vee \dots \vee \zeta_n]}$$

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.

23

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

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash rs : \tau > \tau'[\zeta] \quad \top_{\tau} \models \zeta}{\Gamma \vdash \text{match } e\{rs\} : \tau'}$$

Guarantees exhaustiveness: third premise ensures every value of type τ satisfies ζ which characterizes the values matched by some rule in the sequence.

24

Progress for $\mathcal{L}\{\text{pat}\}$

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 val is a value, or there exists e' such that $e \mapsto e'$.

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.

25