The Algebra of Patterns

ECOOP '25, Bergen

Slides available: binderdavid.github.io/talks

David Binder, University of Kent and Lean Ermantraut, Radboud University Nijmegen

```haskell
data Color = Red | Green | Blue

isRed :: Color -> Bool
isRed Red = True
isRed _ = False
```

The semantics of the pattern match depends on the order of clauses

```haskell
data Color = Red | Green | Blue

isRed :: Color -> Bool
isRed _ = False
isRed Red = True
```

The meaning of each individual clause depends on its position

```haskell
data Color = Red | Green | Blue

isRed :: Color -> Bool
isRed Red = True
isRed Green = False
isRed Blue = False
```

Making clauses non-overlapping by expanding all constructors is impractical

Make Order-Independent Pattern Matching Practical

Why Order-Independence Matters

Why Order-Independence Matters

Equational Reasoning!

$$\mathrm{id}(x) \coloneqq x \tag{1}$$

$$\mathrm{map}(f, []) \coloneqq [] \tag{2}$$

$$\mathrm{map}(f, x :: xs) \coloneqq f(x) :: \mathrm{map}(f, xs) \tag{3}$$

$$\mathrm{map}(\mathrm{id}, []) =_{(2)} []$$

$$\mathrm{map}(\mathrm{id}, x :: xs) =_{(3)} \mathrm{id}(x) :: \mathrm{map}(\mathrm{id}, xs) =_{(1)} x :: \mathrm{map}(\mathrm{id}, xs) =_{(IH)} x :: xs$$

Equations for equational reasoning come from pattern matching clauses

Why Order-Independence Matters

Equational Reasoning!

$$\mathrm{isRed}(\texttt{Red}) := \texttt{True} \tag{4}$$

$$\mathrm{isRed}(\_) := \texttt{False} \tag{5}$$

$$\texttt{True} =_{(4)} \mathrm{isRed}(\texttt{Red}) =_{(5)} \texttt{False}$$

What judgemental equalities are available for automatic rewriting?

Why Order-Independence Matters

Reasoning about Change over Time

- Software engineering is programming integrated over time.
- PL design should be judged for how it accommodates change over time
- How does the meaning of a program change when you add clauses to a pattern match, or constructors to a data type.

More details in the paper!

"Participants felt similarly about the use of wildcards in pattern matching, which silently assign pre-existing behavior to new variants of an enumeration. P7 mentioned that, in their main codebase, wildcards are completely disallowed for this reason, even though they make programming more convenient.

Justin Lubin and Sarah E. Chasins: How statically-typed functional programmers write code.

A Boolean Algebra of Patterns

A Boolean Algebra of Patterns

The cast of patterns

- Variable patterns: x
- Constructor patterns: C(p1,...,pn)
- Wildcard patterns: _
- Absurd patterns: #
- And patterns: p & p ← Generalize x @ p Patterns
- Or patterns: p || p
- Negation patterns: ¬p ← Allow to express complement without mentioning all constructors

The Semantics of Pattern Matching, I

When does a pattern match a value?

Pattern matches: $p \triangleright v \rightsquigarrow \sigma$

$$\frac{}{x \triangleright v \rightsquigarrow [x \mapsto v]}\ \textsc{Var} \qquad \frac{}{\_ \triangleright v \rightsquigarrow []}\ \textsc{Wild} \qquad \frac{p \not\triangleright v \rightsquigarrow \sigma}{\neg p \triangleright v \rightsquigarrow \sigma}\ \textsc{Neg}_1$$

$$\frac{p_1 \triangleright v \rightsquigarrow \sigma}{p_1 \parallel p_2 \triangleright v \rightsquigarrow \sigma}\ \textsc{Or}_1 \qquad \frac{p_2 \triangleright v \rightsquigarrow \sigma}{p_1 \parallel p_2 \triangleright v \rightsquigarrow \sigma}\ \textsc{Or}_2 \qquad \frac{p_1 \triangleright v \rightsquigarrow \sigma_1 \qquad p_2 \triangleright v \rightsquigarrow \sigma_2}{p_1 \,\&\, p_2 \triangleright v \rightsquigarrow \sigma_1 +\!\!+ \sigma_2}\ \textsc{And}$$

$$\frac{p_1 \triangleright v_1 \rightsquigarrow \sigma_1 \qquad \ldots \qquad p_n \triangleright v_n \rightsquigarrow \sigma_n}{\mathcal{C}^n(p_1, \ldots, p_n) \triangleright \mathcal{C}^n(v_1, \ldots, v_n) \rightsquigarrow \sigma_1 +\!\!+ \ldots +\!\!+ \sigma_n}\ \textsc{Ctor}$$

The Semantics of Pattern Matching, II

When does a pattern NOT match a value?

Pattern doesn't match: $p \not\triangleright v \rightsquigarrow \sigma$

$$\frac{}{\# \not\triangleright v \rightsquigarrow []}\ \textsc{Absurd}$$

$$\frac{\exists i : p_i \not\triangleright v_i \rightsquigarrow \sigma}{\mathcal{C}^n(p_1, \ldots, p_n) \not\triangleright \mathcal{C}^n(v_1, \ldots, v_n) \rightsquigarrow \sigma}\ \textsc{Ctor}_1$$

$$\frac{p \triangleright v \rightsquigarrow \sigma}{\neg p \not\triangleright v \rightsquigarrow \sigma}\ \textsc{Neg}_2$$

$$\frac{\mathcal{C}^n \neq \mathcal{C}'^m}{\mathcal{C}^n(p_1, \ldots, p_n) \not\triangleright \mathcal{C}'^m(v_1, \ldots, v_m) \rightsquigarrow []}\ \textsc{Ctor}_2$$

$$\frac{p_1 \not\triangleright v \rightsquigarrow \sigma}{p_1 \,\&\, p_2 \not\triangleright v \rightsquigarrow \sigma}\ \textsc{And}_1 \qquad \frac{p_2 \not\triangleright v \rightsquigarrow \sigma}{p_1 \,\&\, p_2 \not\triangleright v \rightsquigarrow \sigma}\ \textsc{And}_2$$

$$\frac{p_1 \not\triangleright v \rightsquigarrow \sigma_1 \qquad p_2 \not\triangleright v \rightsquigarrow \sigma_2}{p_1 \parallel p_2 \not\triangleright v \rightsquigarrow \sigma_1 +\!\!+ \sigma_2}\ \textsc{Or}$$

Equivalence of Patterns

$$\llbracket p \rrbracket \equiv \llbracket q \rrbracket := \forall v, \forall \sigma,\ p \triangleright v \rightsquigarrow \sigma \Rightarrow \exists \sigma', q \triangleright v \rightsquigarrow \sigma' \wedge \llbracket \sigma \rrbracket \equiv \llbracket \sigma' \rrbracket$$
$$\wedge\ q \triangleright v \rightsquigarrow \sigma \Rightarrow \exists \sigma', p \triangleright v \rightsquigarrow \sigma' \wedge \llbracket \sigma \rrbracket \equiv \llbracket \sigma' \rrbracket$$
$$\wedge\ p \not\triangleright v \rightsquigarrow \sigma \Rightarrow \exists \sigma', q \not\triangleright v \rightsquigarrow \sigma' \wedge \llbracket \sigma \rrbracket \equiv \llbracket \sigma' \rrbracket$$
$$\wedge\ q \not\triangleright v \rightsquigarrow \sigma \Rightarrow \exists \sigma', p \not\triangleright v \rightsquigarrow \sigma' \wedge \llbracket \sigma \rrbracket \equiv \llbracket \sigma' \rrbracket$$

Two patterns are equivalent if they match against the same values with equivalent substitutions, and don't match against the same values with equivalent substitutions.

Reasoning about Patterns, I

▶ **Theorem 8** (Algebraic Equivalences of Patterns, I, 🐦). *For all patterns p, q, r, the following equivalences hold:*

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}l}
[\![p \,\&\, q]\!] & \equiv & [\![q \,\&\, p]\!] & [\![p \parallel q]\!] & \equiv & [\![q \parallel p]\!] & \text{Commutativity} \\
[\![p \,\&\, (q \,\&\, r)]\!] & \equiv & [\![(p \,\&\, q) \,\&\, r]\!] & [\![p \parallel (q \parallel r)]\!] & \equiv & [\![(p \parallel q) \parallel r]\!] & \text{Associativity} \\
[\![p \,\&\, \_]\!] & \equiv & [\![p]\!] & [\![p \parallel \#]\!] & \equiv & [\![p]\!] & \text{Neutral Elements} \\
[\![\neg \_]\!] & \equiv & [\![\#]\!] & [\![\neg \#]\!] & \equiv & [\![\_]\!] & \text{Duality} \\
[\![\neg (p \parallel q)]\!] & \equiv & [\![(\neg p) \,\&\, (\neg q)]\!] & [\![\neg (p \,\&\, q)]\!] & \equiv & [\![(\neg p) \parallel (\neg q)]\!] & \text{De Morgan} \\
\multicolumn{7}{c}{[\![\neg\neg p]\!] \equiv [\![p]\!] \qquad \text{Double Negation}}
\end{array}
$$

Requires semantics to be non-deterministic!

Reasoning about Patterns, II

▶ **Theorem 9** (Equivalences of Constructor Patterns, I, 🐦). *For all patterns p_1 to p_n and p'_1 to p'_n, the following equivalences hold:*

$$[\![\mathcal{C}(p_1, \ldots, p_n) \,\&\, \mathcal{C}(p'_1, \ldots, p'_n)]\!] \equiv [\![\mathcal{C}(p_1 \,\&\, p'_1, \ldots, p_n \,\&\, p'_n)]\!]$$

$$[\![\mathcal{C}(p_1, \ldots, p_i \parallel p'_i, \ldots, p_n)]\!] \equiv [\![\mathcal{C}(p_1, \ldots, p_i, \ldots, p_n) \parallel \mathcal{C}(p_1, \ldots, p'_i, \ldots, p_n)]\!]$$

Essential for correctness of compilation algorithm.

Reasoning About Patterns, III

▶ **Theorem 12** (Algebraic equivalences of Patterns, II, 🦈). *For all patterns p, q and r, the following equivalences hold if the patterns on both sides are linear (i.e. $\mathbf{lin}^{\pm}$).* ← Details in the paper

$$[\![p \,\&\, (q \parallel r)]\!] \equiv [\![(p \,\&\, q) \parallel (p \,\&\, r)]\!] \qquad [\![p \parallel (q \,\&\, r)]\!] \equiv [\![(p \parallel q) \,\&\, (p \parallel r)]\!] \qquad \text{Distributivity}$$

$$[\![p \,\&\, p]\!] \equiv [\![p]\!] \qquad [\![p \parallel p]\!] \equiv [\![p]\!] \qquad \text{Idempotence}$$

$$[\![p \,\&\, \#]\!] \equiv [\![\#]\!] \qquad [\![p \parallel \_]\!] \equiv [\![\_]\!] \qquad \text{Zeros}$$

↑ Patterns on both sides might have different sets of free variables.

Reasoning about Patterns, IV

▶ **Theorem 13** (Equivalences of Constructor Patterns, II, 🦈). *For all patterns p_1 to p_n and p'_1 to p'_n and constructors $\mathcal{C} \neq \mathcal{C}'$, the following equivalences hold if the patterns on both sides are linear (i.e. $\mathbf{lin}^{\pm}$).*

$$[\![\mathcal{C}(p_1, \ldots, \#, \ldots, p_n)]\!] \equiv [\![\#]\!]$$
$$[\![\mathcal{C}(p_1, \ldots, p_n) \,\&\, \mathcal{C}'(p'_1, \ldots, p'_m)]\!] \equiv [\![\#]\!]$$
$$[\![\mathcal{C}(p_1, \ldots, p_n) \,\&\, \neg\mathcal{C}'(p'_1, \ldots, p'_m)]\!] \equiv [\![\mathcal{C}(p_1, \ldots, p_n)]\!]$$
$$[\![\neg\mathcal{C}(p_1, \ldots, p_n)]\!] \equiv [\![\neg\mathcal{C}(\_, \ldots, \_) \parallel \mathcal{C}(\neg p_1, \ldots, p_n) \parallel \ldots \parallel \mathcal{C}(p_1, \ldots, \neg p_n)]\!]$$

Example: ¬Inl(True) = ¬Inl(_) || Inl(¬True)

Open World Assumption: Other constructors of the data type are irrelevant

¬Inl(_) can be compiled to simple check of constructor tag

Compilation of Algebraic Patterns

Negation Normal Form

$$N \quad ::= \quad x \mid \neg x \mid \neg\mathcal{C}^n \mid \mathcal{C}^n(N_1, \ldots, N_n) \mid N \,\&\, N \mid N \parallel N \mid \_ \mid \#$$

Repeatedly rewrite with the following equivalence (+ boolean laws)

$$[\![\, \neg\mathcal{C}(p_1, \ldots, p_n) \,]\!] \equiv [\![\, \neg\mathcal{C}(\_, \ldots, \_) \parallel \mathcal{C}(\neg p_1, \ldots, p_n) \parallel \ldots \parallel \mathcal{C}(p_1, \ldots, \neg p_n) \,]\!]$$

Disjunctive Normal Form

$$N \quad ::= \quad x \mid \neg x \mid \neg \mathcal{C}^n \mid \mathcal{C}^n(N_1, \ldots, N_n) \mid N \,\&\, N \mid N \parallel N \mid \_ \mid \#$$

$$\begin{array}{lll} D & ::= & \parallel \{K_1, \ldots, K_n\} \qquad \text{Disjunctive Normal Form} \\ K & ::= & x \mid \neg x \mid \neg \mathcal{C}^n \mid \mathcal{C}^n(K_1, \ldots, K_n) \mid K \,\&\, K \mid \_ \mid \# \qquad \text{Elementary Conjunct} \end{array}$$

Repeatedly rewrite with the following equivalence (+ boolean laws)

$$[\![\mathcal{C}(p_1, \ldots, p_i \parallel p_i', \ldots, p_n)]\!] \equiv [\![\mathcal{C}(p_1, \ldots, p_i, \ldots, p_n) \parallel \mathcal{C}(p_1, \ldots, p_i', \ldots, p_n)]\!]$$

Normalized Disjunctive Normal Form

$$
\begin{array}{lcll}
D & ::= & \| \{K_1, \ldots, K_n\} & \text{Disjunctive Normal Form} \\
K & ::= & x \mid \neg x \mid \neg \mathcal{C}^n \mid \mathcal{C}^n(K_1, \ldots, K_n) \mid K \,\&\, K \mid \_ \mid \# & \text{Elementary Conjunct}
\end{array}
$$

$$
\begin{array}{lcl}
\overline{D} & ::= & \| \{\overline{K}_1, \ldots, \overline{K}_n\} \\
\overline{K} & ::= & \{x_1, \ldots, x_n\} \,\&\, \mathcal{C}(\overline{K}_1, \ldots, \overline{K}_m) \mid \{x_1, \ldots, x_n\} \,\&\, \neg\{\mathcal{C}_1, \ldots, \mathcal{C}_m\} \mid \{x_1, \ldots, x_n\} \,\&\, \#
\end{array}
$$

Positive Information Negative Information Absurd

Repeatedly rewrite with the following equivalences (+ boolean laws)

$$[\![\mathcal{C}(p_1, \ldots, p_n) \,\&\, \mathcal{C}(p'_1, \ldots, p'_n)]\!] \equiv [\![\mathcal{C}(p_1 \,\&\, p'_1, \ldots, p_n \,\&\, p'_n)]\!]$$

$$[\![\mathcal{C}(p_1, \ldots, p_n) \,\&\, \mathcal{C}'(p'_1, \ldots, p'_m)]\!] \equiv [\![\#]\!]$$

$$[\![\mathcal{C}(p_1, \ldots, p_n) \,\&\, \neg\mathcal{C}'(p'_1, \ldots, p'_m)]\!] \equiv [\![\mathcal{C}(p_1, \ldots, p_n)]\!]$$

Compiled to Decision Trees

Modification of algorithm described by Maranget compiles to decision trees.

$$\textbf{case } v_1, \ldots, v_n \textbf{ of } \begin{bmatrix} \overline{D}_1^1 & \ldots & \overline{D}_n^1 & \Rightarrow e^1 \\ & \vdots & & \\ \overline{D}_1^m & \ldots & \overline{D}_n^m & \Rightarrow e^m \\ & \textbf{default} & & \Rightarrow e_d \end{bmatrix}$$

More infos about these default clauses in paper :)

Summary

Summary

- Provide a language of patterns that makes it **easy to express complements**
- Complements make it **practical to enforce order-independence**
- **Many valid laws** make it easy to reason about patterns (Verified in Rocq)
- Not too expressive: Interesting **properties of patterns are decidable**
- Can be **compiled to efficient code** via decision trees