Algorithmic Redundancy Checking for Pattern Matching

We present a *syntax-directed*, algorithmic procedure for detecting redundant (unreachable) branches in a single-scrutinee `match` expression.

The key idea is to maintain a symbolic description of the set of *uncovered* values, and to test each new pattern against this uncovered space.

Symbolic Value Shapes

We introduce a notion of *symbolic value shape*, or simply *shape*, which is a pattern-like term with no variables except wildcards. These serve as canonical representatives of sets of values.

$$q \quad ::= \quad \_ \mid c \mid C(q_1, \ldots, q_k)$$

We will use finite sets of shapes $S = \{q_1, \ldots, q_r\}$ to represent an over-approximation of the set of values of a given type that are not yet covered by a set of patterns.

Uncovered-Space Judgment

We define a judgment

$$\Delta \vdash \mathsf{Uncov}(T, \vec{p}) \Rightarrow S$$

read: *under type context* Δ*, the sequence of patterns* $\vec{p} = (p_1, \ldots, p_n)$ *for type* T *leaves uncovered the symbolic space* S.

Intuitively:

- $S = \emptyset$ means the patterns are exhaustive.
- $S \neq \emptyset$ gives symbolic examples of values not matched.

We define Uncov inductively using an auxiliary judgment

$$\Delta \vdash \mathsf{Diff}(T, S, p) \Rightarrow S'$$

which removes from S all shapes covered by the pattern p.

Base Case With no patterns, the entire type T is initially uncovered, represented by a single wildcard shape:

$$[\text{U-Empty}]\Delta \vdash \mathsf{Uncov}(T, ()) \Rightarrow \{\_\}$$

Inductive Step To extend the uncovered space with a new pattern p:

$$[\text{U-Cons}]\Delta \vdash \mathsf{Uncov}(T, (p_1, \ldots, p_{k-1})) \Rightarrow S\Delta \vdash \mathsf{Diff}(T, S, p_k) \Rightarrow S'\Delta \vdash \mathsf{Uncov}(T, (p_1, \ldots, p_k)) \Rightarrow S'$$

Difference Judgment DiffDiff

The judgment

$$\Delta \vdash \mathsf{Diff}(T, S, p) \Rightarrow S'$$

means: *from the symbolic uncovered space S of type T, remove all shapes that are covered by pattern p, yielding S'.*

We define Diff syntax-directly on the structure of p and shapes $q \in S$.

Wildcard and Variable Patterns A wildcard or variable pattern matches *all* values, so it removes everything from the uncovered space:

$$[\text{D-W}\textsc{ild}]\Delta \vdash \mathsf{Diff}(T, S, \_) \Rightarrow \emptyset$$

$$[\text{D-V}\textsc{ar}]\Delta \vdash \mathsf{Diff}(T, S, x) \Rightarrow \emptyset$$

Constant Patterns A constant pattern c removes only shapes compatible with c. For simplicity we say: any wildcard shape $\_$ is split into *one* shape equal to c and *one* residual wildcard representing "all values except c". This is schematic; in an implementation one would refine the residual space more precisely.

We write $S = \{q_1, \ldots, q_r\}$ and define:

$$[\text{D-C}\textsc{onst}]\Delta \vdash \mathsf{Diff}(T, \{c\}, c) \Rightarrow \emptyset$$

$$[\text{D-C}\textsc{onst-}\text{W}\textsc{ild}]\Delta \vdash \mathsf{Diff}(T, \{\_\}, c) \Rightarrow \{\_\}'$$

Here $\{\_\}'$ is understood informally as a wildcard shape over the residual values of T distinct from c. For non-matching shapes, we propagate them unchanged:

$$[\text{D-C}\textsc{onst-}\text{O}\textsc{ther}]\Delta \vdash \mathsf{Diff}(T, \{q\}, c) \Rightarrow \{q\} \quad \text{if } q \text{ is incompatible with } c$$

For a general finite set S, we apply Diff pointwise and union the results:

$$[\text{D-S}\textsc{et}]\forall i.\ \Delta \vdash \mathsf{Diff}(T, \{q_i\}, p) \Rightarrow S_i \Delta \vdash \mathsf{Diff}(T, \{q_1, \ldots, q_r\}, p) \Rightarrow \bigcup_{i=1}^{r} S_i$$

Constructor Patterns Assume T is an algebraic data type

$$\texttt{data}\ \ T = C_1(\vec{\tau}_1) \mid \cdots \mid C_n(\vec{\tau}_n).$$

A constructor pattern $C(\vec{p})$ only affects shapes whose outer constructor is C or wildcard $\_$. Wildcards are refined into constructor-specific shapes.

For a single shape:

$$[\text{D-C}\textsc{onstr-}\text{M}\textsc{atch}]q = C(q_1, \ldots, q_k)\Delta \vdash \mathsf{DiffArgs}((\tau_1, \ldots, \tau_k), (q_1, \ldots, q_k), (\vec{p})) \Rightarrow \{(q'_1, \ldots, q'_k)_1, \ldots, (q'_1, \ldots$$

Here $\mathsf{DiffArgs}$ is an auxiliary, syntax-directed procedure that subtracts the argument-pattern space $(\vec{p})$ from the argument-shape tuple $(q_1, \ldots, q_k)$ componentwise; details depend on the desired precision of the checker.

A wildcard shape $\_$ is expanded into constructor-specific shapes before applying D-Constr-Match:

$$[\text{D-Constr-Wild}]\text{Constructors of } T \text{ are exactly } C_1, \ldots, C_n \forall i.\ q_i = C_i(\underbrace{\_, \ldots, \_}_{k_i})\Delta \vdash \mathsf{Diff}(T, \{q_i\}, C(\vec{p})) \Rightarrow$$

Shapes whose outer constructor is different from C are unaffected:

$$[\text{D-Constr-Other}]q = C'(q_1, \ldots, q_\ell),\ C' \neq C\Delta \vdash \mathsf{Diff}(T, \{q\}, C(\vec{p})) \Rightarrow \{q\}$$

Algorithmic Usefulness and Redundancy

Given the uncovered-space analysis, we define a purely syntactic *usefulness* judgment for patterns:

$$\Delta \vdash \mathsf{Useful}(T, \vec{p}, p)$$

read: *for type T, pattern p is useful (non-redundant) relative to the preceding patterns $\vec{p}$.* Algorithmically:

$$[\text{U-Alg}]\Delta \vdash \mathsf{Uncov}(T, \vec{p}) \Rightarrow S\exists q \in S,\ \exists\theta.\ \mathsf{match}(p, q) = \theta\Delta \vdash \mathsf{Useful}(T, \vec{p}, p)$$

That is, p is useful if there is some symbolic shape q in the current uncovered space S that it still matches.

Dually, we define an algorithmic redundancy judgment:

$$\Delta \vdash \mathsf{Redundant}(T, \vec{p}, p)$$

when *no uncovered shape* is matched by p:

$$[\text{R-Alg}]\Delta \vdash \mathsf{Uncov}(T, \vec{p}) \Rightarrow S\forall q \in S.\ \mathsf{match}(p, q) \text{ is undefined}\Delta \vdash \mathsf{Redundant}(T, \vec{p}, p)$$

Algorithmic Non–Redundancy for a Full Branch List

Given a full branch list $\vec{p} = (p_1, \ldots, p_m)$, we say it is algorithmically non–redundant for T if:

$$\Delta \vdash \mathsf{NR}^{\mathsf{alg}}(T, (p_1, \ldots, p_m))$$

holds, where:

$$[\text{NR-Alg-Empty}]\Delta \vdash \mathsf{NR}^{\mathsf{alg}}(T, ())$$

$$[\text{NR-ALG-CONS}]\Delta \vdash \mathsf{NR}^{\mathsf{alg}}(T, (p_1, \ldots, p_{k-1}))\Delta \vdash \mathsf{Useful}(T, (p_1, \ldots, p_{k-1}), p_k)\Delta \vdash \mathsf{NR}^{\mathsf{alg}}(T, (p_1, \ldots, p_k))$$

Thus each new pattern p_k must be *useful* with respect to the uncovered space after the previous patterns.

Meta–Properties (Informal)

- **Soundness**: If $\Delta \vdash \mathsf{Redundant}(T, \vec{p}, p)$ then p is semantically redundant in the sense of the semantic non–redundancy judgment NR (no value of type T is matched first by p).
- **(Possible) Completeness**: With a sufficiently precise definition of Diff and $\mathsf{DiffArgs}$, the algorithmic notion Useful can be made complete with respect to the semantic notion of non–redundancy for first-order algebraic data types, as in classical coverage algorithms for ML/Haskell pattern matching.

In practice, Kleis can implement Uncov and Useful as a coverage-checking pass on pattern matrices, following techniques à la Maranget, using the rules above as a specification.