Pattern-Matching Exhaustiveness Lemma

We work with a fixed algebraic data type declaration

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

Values of type $D(\vec{\sigma})$ are of the form

$$C_i(v_1, \ldots, v_k)$$

with v_j values and C_i one of the constructors above.

Exhaustiveness Judgment

We introduce a judgment

$$\Delta \vdash \mathsf{Exh}(D, \mathcal{P})$$

read: *under type context Δ, the pattern multiset $\mathcal{P}$ is exhaustive for type D.*

We write a branch list as

$$\mathcal{P} = \{\, p_1, \ldots, p_m \,\}.$$

We define exhaustiveness inductively by the following inference rules.

Wildcard and Variable Patterns A single wildcard (or variable) pattern is exhaustive for any type:

$$[\textsc{Ex-Wild}]\Delta \vdash \mathsf{Exh}(T, \{\_\})$$

$$[\textsc{Ex-Var}]\Delta \vdash \mathsf{Exh}(T, \{x\})$$

Constructor Families Assume D has constructors $C_1, \ldots, C_n$ as above. For each i, let $\vec{\tau}_i = (\tau_{i1}, \ldots, \tau_{ik_i})$ denote the argument types.

We consider pattern families of the form

$$\mathcal{P} = \{\, C_1(\vec{p}_1), \ldots, C_n(\vec{p}_n) \,\}$$

where $\vec{p}_i$ is a tuple $(p_{i1}, \ldots, p_{ik_i})$.

We require that for each argument position j of each constructor C_i, the collection of patterns at that position is itself exhaustive for the corresponding argument type τ_{ij}.

Formally, for each i and j, let

$$\mathcal{P}_{i,j} = \{\, p_{ij} \mid C_i(\ldots, p_{ij}, \ldots) \in \mathcal{P} \,\}.$$

Then we define:

$$[\textsc{Ex-Constrs}]\text{Constructors of } D \text{ are exactly } C_1, \ldots, C_n \forall i, j.\ \Delta \vdash \mathsf{Exh}(\tau_{ij}, \mathcal{P}_{i,j})\Delta \vdash \mathsf{Exh}(D, \{\, C_1(\vec{p}_1), \ldots, C_i$$

Adding Redundant Patterns If a set of patterns is exhaustive, adding more patterns preserves exhaustiveness:

$$[\textsc{Ex-Weakening}]\Delta \vdash \mathsf{Exh}(T, \mathcal{P})\Delta \vdash \mathsf{Exh}(T, \mathcal{P} \cup \{p\})$$

(Here $\cup$ denotes multiset union.)

Typing Rule for `match`

We extend the expression typing judgment with a rule for pattern matching:

$$\Gamma \vdash e : T \qquad \Gamma, \Gamma_{p_k} \vdash e_k : U \quad (1 \le k \le m) \qquad \Delta \vdash \mathsf{Exh}(T, \{p_1, \ldots, p_m\})$$

where Γ_{p_k} is the context of bindings introduced by pattern p_k, e.g. for $C(x, y)$ we have $\Gamma_{C(x,y)} = \{x : \tau_1, y : \tau_2\}$.

Then:

$$[\text{T-}\textsc{Match}]\Gamma \vdash e : T\forall k.\ \Gamma, \Gamma_{p_k} \vdash e_k : U\Delta \vdash \mathsf{Exh}(T, \{p_1, \ldots, p_m\})\Gamma \vdash \texttt{match}\ e\{\, p_1 \Rightarrow e_1 \mid \cdots \mid p_m \Rightarrow e_m \,\} : U$$

The third premise enforces *exhaustiveness* of the branch patterns.

Pattern-Matching Exhaustiveness Lemma

We now state the central meta-theoretic property.

Lemma (Exhaustiveness). Let $\Gamma \vdash e : D(\vec{\sigma})$ and

$$\Gamma \vdash \texttt{match}\ e\{\, p_1 \Rightarrow e_1 \mid \cdots \mid p_m \Rightarrow e_m \,\} : U$$

be derived using rule T-Match above, so that in particular

$$\Delta \vdash \mathsf{Exh}(D(\vec{\sigma}), \{p_1, \ldots, p_m\}).$$

Assume a big-step evaluation semantics $\rho \vdash e \Downarrow v$ for values v of type $D(\vec{\sigma})$.

Then for every environment ρ consistent with Γ and for every v with $\rho \vdash e \Downarrow v$, there exists an index k and a pattern environment θ such that:

$$\mathsf{match}(p_k, v) = \theta$$

is defined (i.e. branch k matches), and hence the operational semantics of the whole `match` expression is defined and cannot get stuck due to an unmatched value.

Sketch of Proof. The proof proceeds by induction on the derivation of $\Delta \vdash \mathsf{Exh}(D, \{p_1, \ldots, p_m\})$.

- In cases Ex-Wild and Ex-Var, any value v trivially matches the unique pattern.
- In case Ex-Constrs, any value of type D is of the form $C_i(v_1, \ldots, v_{k_i})$ for some i. The premise ensures that for each argument position j, the patterns $\mathcal{P}_{i,j}$ are exhaustive for the corresponding type τ_{ij}, so the induction hypothesis yields matching subpatterns for each argument, which combine to a matching constructor pattern.
- In case Ex-Weakening, adding extra patterns cannot destroy the existence of a matching branch.

Thus, well-typed `match` expressions with an Exh-justified pattern set are *exhaustive*: evaluation cannot get stuck due to an unmatched constructor.