AutoMan: Mode Analysis

Christa Jenkins

July 16, 2024

1 Conventions

The meta-variable conventions are as follows.

- x ranges over term variables, m over member name segments, and M over module qualified names.
- $\Gamma_{\text{in}}, \Gamma_{\text{out}}$ range over contexts, that is, partial maps from variable x to types T (and possibly x's definition, if it is let-bound)
- Σ ranges over signatures, that is, Σ is a partial map from qualified names M to M's information (module, class, datatype, function/predicate). If M is the name of a predicate declared in Σ, then $\Sigma(M)$ contains the *predicate mode description* for M.
- e ranges over *expressions*.

2 Judgments

2.1 Evaluatable judgment

Read $\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e$ evl as *"under signature Σ, with input-moded formal parameters Γ_{in} and output-moded formal parameters Γ_{out}, expression e is evaluatable."* This judgment is invoked on certain subexpressions of the bodies of predicates marked as an action type (describing either initialization or a state transition), and it maintains the following invariants.

$$\boxed{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e \text{ evl}}$$

Figure 1: Mode analysis (judgments)

- $\{\Gamma_{\text{in}}, \Gamma_{\text{out}}\}$ is a partition of the predicate's formal parameters and previously encountered let bindings. These sets are initially determined by mode annotations supplied by the programmer, but may vary according to the judgment for generatable code.
- Formal type parameters are always considered to be input.
- Σ is a partial map from qualified identifiers to their *predicate mode descriptions*, that is, a mapping (by position) of formal parameters to their input/output mode.

  More formally, if defined then $\Sigma(M) = (I, O)$ where $\{I, O\}$ is a partition on $\{1 \ldots \mathit{arity}(M)\}$. When $O = \emptyset$, M is considered a *proper function* (that is, one that does not need translation); otherwise, M is considered to be the relational specification of a function which we will generate.

3 Rules

$$\frac{x \notin \Gamma_{\text{out}}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash x \text{ evl}} \qquad \frac{m \text{ an id or nonneg int} \quad \Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e \text{ evl}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e.m \text{ evl}}$$

$$\frac{(\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e_i)_{i \in \{0 \ldots n\}}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e_0.\,(m_i := e_i)_{i \in \{1 \ldots n\}} \text{ evl}} \qquad \frac{(\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e_i \text{ evl})_{i \in \{0 \ldots 2\}}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e_0[e_1..e_2] \text{ evl}}$$

$$\frac{\Sigma(M) = (\{1 \ldots n\}, \emptyset) \quad (\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e_i)_{i \in \{1 \ldots n\}}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash M(e_1, \ldots, e_n) \text{ evl}}$$

$$\frac{\Sigma, \Gamma_{\text{in}}\,(x_i : T_i)_{i \in \{1 \ldots n\}}, \Gamma_{\text{out}} \vdash e \text{ evl}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash (x_i : T_i)_{i \in \{1 \ldots n\}} \Rightarrow e \text{ evl}} \qquad \frac{(\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e_i \text{ evl})_{i \in \{1,2,3\}}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash \mathit{if}\ e_1\ \mathit{then}\ e_2\ \mathit{else}\ e_3 \text{ evl}}$$

$$\frac{(\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e_i)_{i \in \{1 \ldots n\}}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash [e_1, \ldots, e_n] \text{ evl}}$$

Figure 2: Mode analysis rules (evaluatable judgment)

$$\boxed{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash \mathcal{Q}_{\text{dom}}\ \text{evl}} \qquad \frac{(\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash e_i\ \text{evl})_{i \in \{1,2\}}}{\Sigma, \Gamma_{\text{in}}, \Gamma_{\text{out}} \vdash x : T \leftarrow e_1 \mid e_2\ \text{evl}}$$

Mode analysis rules (evaluatable judgment, auxiliary rules)

3.1 Identifiers

Identifiers can range over many different sorts of entities.

- Formal (term) parameters of a predicate.

  These must not appear in the set of output-moded variables.

- Module or class names.

  These are always considered input.

- Types (datatypes, primitives, formal type parameters)

  These are always considered input.

3.2 Member Access

- As long as e is evaluatable, so is $e.m$.
- In Dafny, m can be an identifier (or identifier segment, i.e., with type instantiation) *or* a nonnegative integer, such as in tuple component access

3.3 Member update

- As long as e_0 is evaluatable, and each expression e_i that updates a field is evaluatable, then the field update expression $e_0.(m_i =: e_i)_{i \in \{1 \ldots n\}}$ is evaluatable.

3.4 Sequence (slices, access, update)

- As long as e_0, e_1, e_2 are evaluatable, so is $e_0[e_1..e_2]$.
- Similarly for length subsequences ($e_0[e_1 : e_2 : \ldots]$), accesses ($e_0[e_1]$), and updates ($e_0[e_{1,1} := e_{1,2}]$).

3.5 Function calls

- The rule restricts called functions to be qualified identifiers defined within signature Σ.
  - Formal parameters with function types that are marked as input could be permitted here, but we do not currently support function types
  - Lambda expressions could be permitted here, but rarely does one see a redex like this in practice
- M must be a *proper function*.
  - This is not a hard requirement, just a simplification. In theory, M could be a relational specification that assigns values to output parameters, which would be acceptable to use in the evaluatable position. However, this would make the evaluatable judgment mutually inductively defined with the generatable judgment, and require the former to track the assignments generated.