Mode System in Mercury

Kate Verbitskaia

JetBrains Programming Languages and Tools Lab

30.08.2022

Mercury

- Purely declarative: predicates and functions do not have non-logical side effects
- Uses powerful static analyses for compilation
  - Strong type system
  - Strong mode system
  - Strong determinism system
- Is intended for real-world use
  - Has a module system
  - Supports higher-order programming, with closures, currying, and lambda expressions
  - Very efficient

`https://www.mercurylang.org/about.html`

Example Program

- Code
- See also: `https://github.com/Mercury-Language/mercury/tree/master/samples`

Modes

- Describe how instantiatedness of variables changes during execution
- Basic modes
  - `in == ground » ground`
  - `out == free » ground`
- Modes with extra uniqueness information
  - `di == unique » clobbered`
  - `uo == free » unique`

```
:- pred append(in, in, out) is det.
```

Benefits of Mode Analysis

- Predicates are specialized according to the mode
  - Specialized unifications
  - Conjunction reordering
- No need for occurs check
- Early bug detection
- Documentation

Discriminated Union Types

```
:- type employee
        ---> employee( name        :: string,
                       age         :: int,
                       department  :: string).

:- type tree
        ---> empty
        ;    leaf(int)
        ;    branch(tree, tree).

:- type list(T)
        ---> []
        ;    [T | list(T)].
```

Instantiatedness Declarations

Assigning either `free` or `bound` to nodes of a type tree

```
:- type list(T) ---> []; [T | list(T)].

:- inst listskel == bound([] ; [free | listskel]).

:- inst listskel(Inst) for list/1
   ---> []
   ;    [Inst | listskel(Inst)]
```

Terms approximated with instantiatedness `listskel`

- ✓ `[A, B]`
- ✗ `[A, 2]`
- ✗ `[H | T]`
- ✗ `[A, A]`

Mode Correctness: High Level Idea

A variable cannot become more free after predicate execution

Precise and expressive mode systems for typed logic programming languages. David Overton (PhD thesis): https://mercurylang.org/documentation/papers.html#dmo-thesis

Instantiation State

$$\iota ::= \textit{free} \mid \textit{bound}(\mathcal{P} f(\bar{\iota}))$$

Mode Annotations for `append(in, in, out)`

$$
\begin{array}{ll}
\texttt{append}(\texttt{Xs}, \texttt{Ys}, \texttt{Zs}) \leftarrow & \\
\quad \bigvee\langle & \\
\quad\quad \bigwedge\langle & \\
\quad\quad\quad \texttt{Xs} = \texttt{[]}, & \{\, \texttt{Xs} \mapsto \mathsf{bound}(\{\, \texttt{[]} \,\}) \,\} \\
\quad\quad\quad \texttt{Ys} = \texttt{Zs} & \{\, \texttt{Zs} \mapsto \mathsf{ground} \,\} \\
\quad\quad \rangle, & \{\, \texttt{Xs} \mapsto \mathsf{bound}(\{\, \texttt{[]} \,\}), \texttt{Zs} \mapsto \mathsf{ground} \,\} \\
\quad\quad \exists \{\, \texttt{Xs0}, \texttt{Zs0}, \texttt{X} \,\}.(& \\
\quad\quad\quad \bigwedge\langle & \\
\quad\quad\quad\quad \texttt{Xs} = \texttt{[X | Xs0]}, & \left\{ \begin{array}{l} \texttt{Xs} \mapsto \mathsf{bound}(\{\, \texttt{[}\mathsf{ground}\,|\,\mathsf{ground}\texttt{]} \,\}), \texttt{X} \mapsto \mathsf{ground}, \\ \texttt{Xs0} \mapsto \mathsf{ground} \end{array} \right\} \\
\quad\quad\quad\quad \texttt{append}(\texttt{Xs0}, \texttt{Ys}, \texttt{Zs0}), & \{\, \texttt{Zs0} \mapsto \mathsf{ground} \,\} \\
\quad\quad\quad\quad \texttt{Zs} = \texttt{[X | Zs0]} & \{\, \texttt{Zs} \mapsto \mathsf{bound}(\{\, \texttt{[}\mathsf{ground}\,|\,\mathsf{ground}\texttt{]} \,\}) \,\} \\
\quad\quad\quad \rangle & \left\{ \begin{array}{l} \texttt{Xs} \mapsto \mathsf{bound}(\{\, \texttt{[}\mathsf{ground}\,|\,\mathsf{ground}\texttt{]} \,\}), \texttt{X} \mapsto \mathsf{ground}, \\ \texttt{Xs0} \mapsto \mathsf{ground}, \texttt{Zs0} \mapsto \mathsf{ground}, \\ \texttt{Zs} \mapsto \mathsf{bound}(\{\, \texttt{[}\mathsf{ground}\,|\,\mathsf{ground}\texttt{]} \,\}) \end{array} \right\} \\
\quad\quad) & \left\{ \begin{array}{l} \texttt{Xs} \mapsto \mathsf{bound}(\{\, \texttt{[}\mathsf{ground}\,|\,\mathsf{ground}\texttt{]} \,\}), \\ \texttt{Zs} \mapsto \mathsf{bound}(\{\, \texttt{[}\mathsf{ground}\,|\,\mathsf{ground}\texttt{]} \,\}) \end{array} \right\} \\
\quad \rangle & \{\, \texttt{Xs} \mapsto \mathsf{bound}(\{\, \texttt{[]}, \texttt{[}\mathsf{ground}\,|\,\mathsf{ground}\texttt{]} \,\}), \texttt{Zs} \mapsto \mathsf{ground} \,\}
\end{array}
$$

Figure 3.10: Abstract syntax for predicate ‘`append/3`’ with mode annotations

Partial Order on Insts

$$\iota \preceq \mathit{free}$$

$$\mathit{bound}(B) \preceq \mathit{bound}(B') \text{ iff}$$
$$\forall \beta \in B. \exists \beta' \in B'. \beta = f(\iota_1, \ldots, \iota_n), \beta' = f(\iota'_1, \ldots, \iota'_n), \forall i. \iota_i \preceq \iota'_i$$

$$\mathit{not\_reached} = \mathit{bound}(\varnothing)$$

Partial Order Example

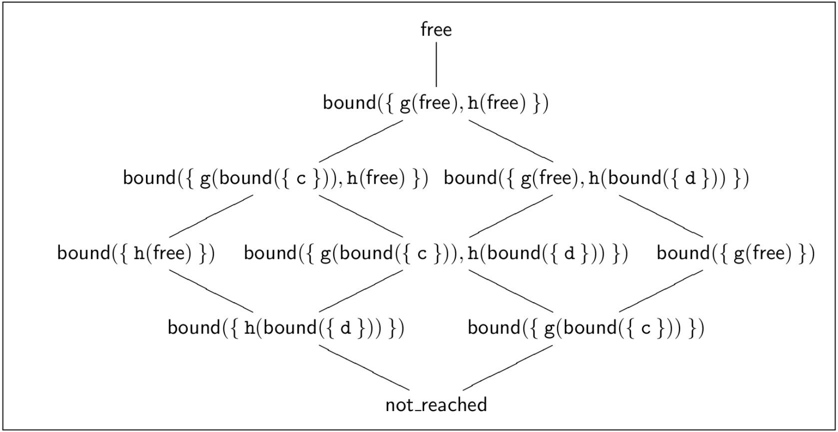

Figure 3.5: Hasse diagram for $\langle \mathsf{Inst}, \preceq \rangle$ and Herbrand universe $\{$ `g(c), h(d)` $\}$ (see Example 3.1.2)

Insts Concretization

$$\gamma(\mathit{free}) = \{\_\}$$

$$\gamma(\mathit{bound}(B)) = \bigcup_{f(\iota_1,\ldots,\iota_n)\in B} \{f(t_1,\ldots,t_n), \forall j. t_j \in \gamma(\iota_j)\}$$

Matches Partial Order

$$free \sqsubseteq free$$

$$not\_reached \sqsubseteq free$$

$$bound(B) \sqsubseteq bound(B')\ iff$$
$$\forall \beta \in B. \exists \beta' \in B'. \beta = f(\iota_1, \ldots, \iota_n), \beta' = f(\iota'_1, \ldots, \iota'_n), \forall i. \iota_i \sqsubseteq \iota'_i$$

$$not\_reached = bound(\varnothing)$$

Matches Partial Order Example

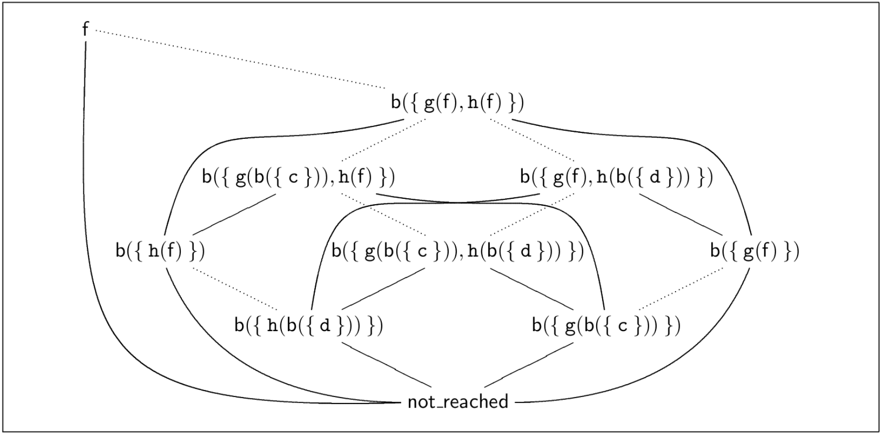

Figure 3.6: Hasse diagram for $\langle \mathsf{Inst}, \sqsubseteq \rangle$ and Herbrand universe { `g(c), h(d)` } (see Example 3.1.3)

Terms Abstraction

$$\alpha(T) = \bigsqcup_{t \in T} \{\alpha'(t)\}$$

$$\alpha'(\_) = \mathit{free}$$

$$\alpha'(f(t_1, \ldots, t_n)) = \mathit{bound}(\{f(\alpha'(t_1), \ldots, \alpha'(t_n))\})$$

Mode Checking Algorithm

- To mode check the program, mode check all predicates
- To mode check a predicate:
  - Initialize insts of head vars with initial insts
  - Mode check the body goal
  - If no mode errors, check that final insts match the final insts declaration

Mode Checking: Disjunction

- Mode check each disjunct
- Merge resulting insts

$$merge(\langle I, I' \rangle, \langle I, I'' \rangle) = \langle I, \{v \rightarrow \iota \mid v \in dom(I) \wedge \iota = I'(v) \sqcup I''(v)\}\rangle$$

Mode Checking: Conjunction

- Schedule a conjunct if possible
  - Scheduling: attempt to mode check
    - If success, then scheduling succeeds and we commit to the current order
    - If fails with not sufficiently instantiated local variable, check other order
  - Delay conjunct if its not sufficiently instantiated
  - Every time a var is bound, check if a delayed conjunct should be awakened
- Mode check conjuncts, combine the result

$$combine(\langle I, I' \rangle, \langle I', I'' \rangle) = \langle I, I'' \rangle$$

Reordering of Conjuncts

- Conjunction may be not mode-correct, but some permutation of conjuncts may be
- Mode checker may pick any mode-correct permutation
- Efficiency not guaranteed

Mode Checking: Unification

- Check that unification does not attempt to unify two free vars
- Split unifications to avoid complex unifications
- The result inst is the initial inst, but the affected vars are associated with the result of abstract unification:

$$\mathit{abstract\_unify\_inst}(\iota_1, \iota_2, \iota) \text{ iff } \iota = \iota_1 \curlywedge \iota_2 \wedge \iota \curlywedge \mathit{ground}$$

Mode Checking: Call

- Check that there is a mode which matches the insts of args
- If there is none, attempt to infer mode
- !NB Modes may be implied
  - ✓ append(in, in, in) is an implied mode of append(in, in, out)
  - ✗ append(out, in, in) is not an implied mode of append(in, in, out)

Mode Inference

- Set init insts
- Set final insts to be `not_reached`
- Until fix point is reached (final insts)
  - Mode check the body goal
  - Normalize final insts

What is Not in this Talk

- Determinism Analysis
- Uniqueness and liveness
- Polymorphism
- Constraint-based mode analysis