Soundness of MetaML

Runtime Verification Inc.

January 8, 2024

In the following, we semi-formally analyze the soundness problem for matching logic extended with metavariables and discuss the problem with *void proof schemas*. We assume a fixed set of variables $\textsc{Var} = \mathrm{EV} \cup \mathrm{SV}$ and symbols Σ, given by the data types of our implementation. We also assume an implicit ordering on the set of variables (so that we can define some functions conveniently).

1 Preliminaries

Let us first go over basic definitions, where we extend basic syntax with metavariables.

Definition 1.1 (Constraints)**.** *A constraint is a tuple* (χ, τ) *where* $\chi \in \textsc{Var}$ *and* $\tau \in \{\mathsf{ef}, \mathsf{sf}, \mathsf{pos}, \mathsf{neg}, \mathsf{hol}\}$ *is a* constraint type. ■

Definition 1.2 (Meta-variable)**.** *A pair* $M = (\mathsf{id}, R)$ *is called a* meta-variable *if* $\mathsf{id} \in \mathbb{N}$ *and* R *is a finite set of constraints. The set of all metavariables is denoted* $\textsc{Metavar}$. ■

Note that the set of constraints R is finite as any instantiation is meant to represent a concrete pattern, which contains finitely many variables.

Definition 1.3 (Patterns)**.** *We extend the syntax of patterns with*

$$\begin{aligned}\varphi ::= {} & M \mid M[\varphi/x] \text{ for every } x \in \mathrm{EV} \mid M[\varphi/X] \text{ for every } X \in \mathrm{SV} \\ & \mid \varphi \in \textsc{Pattern}\end{aligned}$$

for every metavariable $M \in \textsc{Metavar}$. ■

If we want to explicitly say that a pattern contains a metavariable, we use the following terms:

Definition 1.4 (Meta-patterns)**.** *A pattern containing at least one meta-variable is called a* meta-pattern. ■

Definition 1.5 (Concrete patterns)**.** *A pattern is called* concrete *if it is not a meta-pattern.* ■

We do not make any assumptions about meta-variables with the same ids to have the same constraints. On the contrary, this is a special property of patterns that we will be considering:

Definition 1.6 (Metavariable-consistency)**.** *We call a pattern ψ* metavariable-consistent*, if for every M, M' in ψ we have that $M.id = M'.id$ implies $M = M'$.* ■

We naturally extend the notion of constraints to all patterns:

Definition 1.7 (Pattern constraints)**.** *Let φ be a pattern. Its set of constraints R is defined such that for every $\chi \in \textsc{Var}$, we set by definition $(\chi, \tau) \in R$ iff $\varphi.\tau(\chi)$ returns true (as given by the specification of the given attributes).*[1]

For convenience, we also define the sets $\chi \in R.\tau$ iff $(\chi, \tau) \in R$ for each constraint type, i.e., for each $\tau \in \{\mathsf{ef}, \mathsf{sf}, \mathsf{pos}, \mathsf{neg}, \mathsf{hol}\}$. ■

Note that constraints are *determined* for every given pattern and can be computed. However, we can "change" the constraints of contained meta-variables, yielding a structurally equivalent but different pattern:

Definition 1.8 (Structural equivalence)**.** *Given two patterns φ, ψ, we write $\varphi \approx \psi$ iff φ and ψ are equal up to metavariable constraints.* ■

Put together, constraints are solely determined by contained metavariables and pattern structure. The following lemma puts this intuition together and says that if we preserve some kind of constraint in all contained metavariables of structurally equivalent patterns, then the computed constraint of this type also remains the same for every contained subpattern:

Lemma 1.1. *Let $\psi \approx \psi'$ and τ be a constraint type. If for all corresponding metavariables contained in ψ and ψ' we have constraints R, R' with $R.\tau = R'.\tau$, then for every subpattern φ of ψ and φ' of ψ' we have $\varphi.\tau = \varphi'.\tau$.*

Proof. By a simple structural induction.[2] □

We care heavily whether a pattern under given constraints represents some concrete matching logic pattern, in other words, if it is *instantiable*:

Definition 1.9. *A pattern ψ is called* instantiable *if there exists a (possibly partial)* instantiation function $\delta : \mathbb{N} \to \textsc{Pattern}$ *such that $\psi\delta$ is concrete*[3] *and δ satisfies every constraint of every meta-variable with the given id in ψ.*

In this case, we call such $\psi\delta$ an instantiation *of ψ and ψ* instantiable by *δ.*

If δ is defined only for metavariables occurring in ψ, we say δ is tight *for ψ.* ■

Proposition 1. *ψ is instantiable by δ iff every metavariable in ψ is instantiable by δ.* ■

[1] TODO: Add those definitions here.

[2] Add proof.

[3] $\psi\delta$ is the pattern resulting from replacing every meta-variable M in ψ with $\delta(M.id)$

This definition makes good sense. There are patterns that are not instantiable. The simplest example is a metavariable M such that $M.\mathsf{ef} \cap M.\mathsf{hol} = \{x\}$. An instantiation would need to contain x both free (as a hole variable) and not free (as a fresh variable), which is a contradiction. Note that a conflict does not need to occur in a single metavariable. Consider $M_1 \vee M'_1$, where $M_1 \approx M'_1$ share the same id "1" but constraints are conflicting in the same manner as what we just sketched: $M_1.\mathsf{hol} = \{x\}$ and $M'_1.\mathsf{ef} = \{x\}$. Then there cannot be a single function δ instantiating $M_1 \vee M'_1$, as $\delta(1)$ will be in conflict with constraints of M_1 or M'_1 for the reasons explained above.

Note that the condition "$\psi\delta$ is concrete" in Definition 1.9 is necessary. Otherwise we could call a meta-pattern instantiable even when we could "instantiate" it with another uninstantiable meta-pattern, which is very straightforward to do (given how constraints in meta-patterns work). On the other hand, concrete patterns are instantiable trivially as expected.

We can extend these concepts to proofs. As pattern syntax was extended, proofs can now contain meta-patterns. Recall that meta-patterns are patterns representing a set of concrete patterns consisting of all admissible instantiations. A *proof schema* is similar to a meta-pattern, in a way that it represents a whole set of proofs in the original matching logic sense that contain the respective instantiations:

Definition 1.10 (Proof schema)**.** *A* proof schema *is any proof containing at least one meta-pattern.* ■

We do not really care to differentiate proof schemas from "concrete" proofs as long as every contained pattern is instantiable. If a proof schema contains an uninstantiable pattern, this proof schema does not correspond to any proof in the original matching logic sense and such proof schemas are called void:

Definition 1.11 (Void proof schemas)**.** *A proof schema $\varphi_1, \ldots, \varphi_n$ is called* void *if there exists i such that φ_i is not instantiable.* ■

2 Problem

Void proof schemas represent no values in the original ML syntax, i.e., no ML proofs in the original sense inhabit them. The following shows that it is possible to derive an instantiable meta-pattern with a void proof schema, which means there is an illusory ML derivation of a concrete theorem, as the proof schema does not yield a ML proof in the original sense (at least not directly):

Proposition 2. *There exists a void proof schema of an instantiable pattern.*

Proof. Consider the meta-axiom $A_1 := M_1 \to (M_2 \to M_1)$ where M_1 is a meta-variable that is not instantiable. Secondly, let $\top := x \vee \neg x$. As we have complete propositional reasoning, $A_1 \to \top$ is derivable with some propositional proof schema π.[4] But then $\ldots \pi \ldots, A_1 \to \top, A_1, \top$ consitutes a void proof schema, where the conclusion $\top$ is instantiable trivially (being concrete). □

[4] this would not hold for μ-based top

This does not show that extending patterns with meta-variables make the proof system unsound. However, we see now that void proof schemas with concrete conclusions are admissible. We have to *prove* that this does not open a possibility of a void proof schema ending with a false conclusion.

Let us emphasize first that we only care about *structure*. We do not mind replacing structurally equivalent patterns in proof schemas, insofar as the proof schema remains correct. We will prove that for any proof schema deriving a pattern ψ (concrete or not), there is a (correct) proof schema deriving $\psi' \approx \psi$ that is not void.

In other words, for every proof schema, there exists a schema with a structurally equivalent conclusion in which all of the proof steps are instantiable w.r.t. some constraints. This will mean that every proof schema indirectly corresponds to some proof in the original ML sense. By soundness of our proof system, this will show that there is no void proof schema of a false conclusion, and so the proof system is sound even when we extend patterns with meta-variables.

Main idea. Such a soundness proof is based on a simple observation that all of constraint types are conditional (if a variable appears, then it meets the constraint) except for hol. If hol is disjoint from the other constraints, we are always instantiable with a simple application sequence of variables to variables in hol:

Definition 2.1 (Variable application sequences)**.** *Let* $\mathsf{Fin}(A)$ *denote the set of all finite subsets of a set* A*. We define a function* $I : \mathsf{Fin}(\textsc{Var}) \times \textsc{Var} \to \textsc{Pattern}$ *as follows:*

- $I(\emptyset, \square) = \square,$
- $I(\{\chi\} \cup \Xi', \square) = \mathsf{app}\ \chi\ I(\Xi')$

■

To make things simpler, we will w.l.o.g. assume some special variable $\square$ not being used in any one of our constraints in the following text.

Definition 2.2. *If for every* n*,* $\delta(n)$ *is undefined or there exists a set* Ξ *such that* $\delta(n) = I(\Xi, \square)$*, then* δ *is called a* trivial instantiation function*.* ■

Definition 2.3. *Let* M *be a metavariable. Any trivial instantiation function* $\delta : \mathbb{N} \to \textsc{Pattern}$ *with* $\delta(M.id) = I(M.\mathsf{hol}, \square)$*, we call a trivial instantiation function* for M*.* ■

Proposition 3. *Let* M *be a metavariable with* $M.\mathsf{hol}$ *disjoint from other constraints. If* δ *is the trivial instantiation function for* M*, then* $M\delta$ *is an instantiation of* M*.*

Proof. The function δ satisfies all of the constraints for M:

1. $M\delta$ is an application context w.r.t. all variables in $M.\mathsf{hol}$, and
2. $M\delta$ trivially satifies all of the remaining constraint types by our disjointness assumption.

□

Corollary 1. *If $M.\mathsf{hol}$ is disjoint from other constraint types, then M is instantiable over any signature Σ.* ■

We want to extend this concept to patterns. Recall that we can have two metavariables with same id but different constraints, in which case we have a *conflict*. However, this cannot happen when we work with metavariable-consistent patterns. There it is only a matter of merging instantiations together:

Definition 2.4 (Products)**.** *If δ_1, δ_2 are instantiation functions corresponding on their domains, we define their product $\delta_1\delta_2 : \mathbb{N} \to \textsc{Pattern}$ as follows:*

$$\delta_1\delta_2(n) := \begin{cases} \delta_1(n) & \delta_1(n) = \delta_2(n) \text{ is defined} \\ \delta_1(n) & \delta_1(n) \text{ is defined} \\ \delta_2(n) & \delta_2(n) \text{ is defined} \end{cases}$$

■

Proposition 4. *If δ_1, δ_2 are trivial instantiation functions corresponding on their domains, then $\delta_1\delta_2$ is a trivial instantiation function.* ■

Proposition 5. *Let M_1, M_2 be metavariables with $M_1.id \neq M_2.id$. Assume $M_1.\mathsf{hol}$ is disjoint from other constraints of M_1, and $M_2.\mathsf{hol}$ is disjoint from other constraints of M_2. If M_1 (M_2) is instantiable by a* tight *trivial instantiation function δ_1 (δ_2), then both M_1, M_2 are instantiable by $\delta := \delta_1\delta_2$.*

Theorem 1. *Let every metavariable M in ψ have $M.\mathsf{hol}$ disjoint from other constraints of M. If ψ is metavariable-consistent, then ψ is instantiable.*

Proof. We will prove a stronger statement:

> For every ψ holds the following. If ψ is metavariable-consistent and every metavariable in ψ has hole constraints disjoint from its other constraints, then ψ is instantiable by a *tight* instantiation function that is trivial for every metavariable in ψ.

By a simple structural induction. The main trick is to use products of trivial instantiation functions to merge them together.

- Concrete base cases are trivial.
- $\psi \equiv M$: Let δ be a tight instantiation function that is trivial for M. Then $M\delta$ is an instantiation of M by Proposition 3.

- $\psi \equiv M[\varphi/\chi]$. Our assumptions about ψ distribute to their subpatterns, and so M, φ have all hole constraints disjoint and are metavariable-consistent. We have by IH that M is instantiable by some tight instantiation function δ_1 and φ is instantiable by some tight instantiation function δ_2. Consider the product $\delta := \delta_1\delta_2$

  - δ is well-defined. By IH, δ_1 is tight for M and δ_2 is tight for φ, meaning if both $\delta_1(n), \delta_2(n)$ are defined, then metavariables M_1, M_1' with $M_1.id = M_1'.id = n$ occurs in both M and φ. As ψ is metavariable-consistent, $M_1 = M_1'$. As both δ_1, δ_2 are *trivial* for $M_1 = M_1'$, by construction, necessarily $\delta_1(n) = \delta_2(n)$.
  - δ is tight for ψ. By product construction, as δ_1 (δ_2) is tight by IH for M (φ).
  - δ is trivial for every metavariable in ψ. By IH and product construction.

  By Proposition 5, $M\delta$ is an instantiation of M and also $\varphi\delta$ is an instantiation of φ: φ is instantiable by δ iff every M' in φ is instantiable by δ (Proposition 1), which it is by IH and Proposition 5 (pick $M_1 := M$ and $M_2 := M'$). Thus the expression

  $$M\delta[\varphi\delta/\chi] = (M[\varphi/\chi])\delta$$

  is an instantiation of $M[\varphi/\chi]$.

- Other cases are analogous, as it is all just constructors where metavariables cannot be bound.

□

Not every pattern is metavariable-consistent. However, recall that our concern here is only to make sure that no proof schema can derive a structurally false conclusion. We can change constraints insofar as we do not mess up with the correctness of the schema. First, we extend metavariable-consistency to proofs:

Definition 2.5. *A proof schema π is called* metavariable-consistent*, if for every metavariable M, M' occurring in π we have $M.id = M'.id$ implies $M = M'$.* ■

Theorem 2. *If a proof schema is metavariable-consistent such that for every contained metavariable M, $M.\mathsf{hol}$ is disjoint from other constraints of M, then every pattern in the proof schema is instantiable by a single instantiation function.*

Proof. For every metavariable M in the proof schema, use the tight trivial instantiation for M. Every pattern in the proof schema is instantiable by the same instantiation function: the product of these tight trivial instantiation functions. □

Corollary 2. *If metavariables are only allowed with holes disjoint from other constraints and proofs are metavariable-consistent, the proof system extended with metavariables is sound.*

Proof. With our assumption, every proof schema represents a concrete proof in the original proof system by Theorem 2. This is because there is a single instantiation function, and thus every metavariable is instantiated consistently. The conclusion follows from the soundness of the matching logic proof system (over concrete patterns). □