The Semantics of K

Formal Systems Laboratory
University of Illinois

July 26, 2017

Please feel free to contribute to this report in all ways. You could add new contents, remove redundant ones, refactor and organize the texts, and correct typos.

Definition 1 (Matching Logic Theory)**.** A matching logic theory (S, Σ, A) is a triple that contains a nonempty finite set of sorts, a finite set of symbols, and a recursive set of axioms. Two theories are *equal* if they have the same set of sorts and symbols, and they deduce the same set of theorems.

Example 2. We use serif fonts to denote matching logic theories. Some of the commonly used ones are the theory (theories) of definedness DEF, the theory of Presburger arithmetic PA, the theory of sequences of natural numbers SEQ, the theory of memory heaps HEAP, the theory of IMP programs IMP, the theory (theories) of fixed-points FIX, and the theory (theories) of contexts CTXT.

Definition 3 (The Kore Language)**. We haven't come to an agreement on the syntax of the Kore language yet. One could, though, refer to the Kore text representation at the wiki page at kframework repos on Github, whose link I cannot find any more, which is considered as the first step towards that direction.**

The Kore language is a language to write matching logic theories. The outcomes are called Kore definitions. Kore definitions are mainly served as the interface between a K frontend and a K backend, but a human should be able to read and write Kore definitions of simple theories, too. The Kore language is designed in a way that:

- Every Kore definition defines exactly one matching logic theory;
- Every matching logic theory can be defined as a Kore definition;
- There is no parsing ambiguity.
- The least amount of inferring is needed;
- Symbols are decorated with their argument sorts and result sort;

- There is no polymorphic or overloaded symbols, so every symbol has a unique name (reference).

  The above two points will lead to some redundancy in Kore definitions, though. For example, there will be n^2 definedness symbol if there are n sorts in the theory. Similarly, n^2 equalities are need, too.

- User-defined (matching logic) variables are allowed as in theory LAMBDA. We shouldn't fix a syntax for variables. Please refer to later sections about theory LAMBDA for more discussions.

- And more ...

Definition 4 (Frontend)**.** A K frontend is an artifact that generates Kore definitions.

Definition 5 (Backend)**.** A K backend is an artifact that consumes a Kore definition of a theory T and does some work. Whatever it does can and should be algorithmically reduced to the task of proving $\mathsf{T} \vdash \varphi$ where φ "encodes" that work. A K backend should justify its results by generating formal proofs that can be proof-checked by the oracle matching logic proof checker.

1 Object-level and meta-level

It is an aspect of life in mathematical logics to distinguish the *object-level* and *meta-level* concepts. The basic principle is to use serif fonts letters (x) for object-level concepts and normal math fonts letters for meta-level concepts (x).

Variables and metavariables for variables For any matching logic theory $\mathsf{T} = (S, \Sigma, A)$, it comes for each sort $s \in S$ a countably infinite set V_s of variables. We use $\mathsf{x} : \mathsf{s}, \mathsf{y} : \mathsf{s}, \mathsf{z} : \mathsf{s}, \ldots$ for variables in V_s, and omit their sorts when that is clear from the contexts. Different sorts have disjoint sets of variables, so $\mathrm{Var}_s \cap \mathrm{Var}_{s'} = \emptyset$ if $s \neq s'$.

Proposition 6. *Let A be a set of axioms and $\bar{A} = \forall A$ be the universal quantification closure of A, then for any pattern φ, $A \vdash \varphi$ iff $\bar{A} \vdash \varphi$.*

Remark 7 (Free variables in axioms)*.* The free variables appearing in the axioms of a theory can be regarded as implicitly universal quantified, because a theory and its universal quantification closure are equal.

Example 8.

$$\begin{aligned}
A_1 &= \{\mathsf{mult}(\mathsf{x}, 0) = 0\}\\
A_2 &= \{\forall \mathsf{x}.\mathsf{mult}(\mathsf{x}, 0) = 0\}\\
A_3 &= \{\forall \mathsf{y}.\mathsf{mult}(\mathsf{y}, 0) = 0\}\\
A_4 &= \{\mathsf{mult}(x, 0) = 0\}\\
A_5 &= \{\forall x.\mathsf{mult}(x, 0) = 0\}\\
A_6 &= \{\forall y.\mathsf{mult}(y, 0) = 0\}\\
A_7 &= \{\mathsf{mult}(\mathsf{x}, 0) = 0, \mathsf{mult}(\mathsf{y}, 0) = 0, \mathsf{mult}(\mathsf{z}, 0) = 0, \ldots\}\\
A_8 &= \{\forall \mathsf{x}.\mathsf{mult}(\mathsf{x}, 0) = 0, \forall \mathsf{y}.\mathsf{mult}(\mathsf{y}, 0) = 0, \forall \mathsf{z}.\mathsf{mult}(\mathsf{z}, 0) = 0, \ldots\}
\end{aligned}$$

All the eight theories are equal. Theories A_4, A_5, A_6 are finite representations of theories A_7, A_8, A_8 respectively.

Remark 9. There is no need to have metavariables for variables in the Kore language, because (1) if they are used as bound variables, then replacing them with any (matching logic) variables will result in the same theories, thanks to alpha-renaming; and (2) if they are used as free variables, then it makes no difference to consider the universal quantification closure of them and we get to the case (1).

~~Given said that, there are cases when metavariables for variables make sense. In those cases we often want our metavariables to range over all variables of all sorts, in order to make our Kore definitions compact.~~ No, we do not need metavariables over variables. I was thinking of the definedness symbols. We might want to write only one axiom schema of $\lceil x \rceil$ instead many $\lceil x{:}s \rceil_s^{s'}$'s, but we cannot do that unless we allow polymorphic and overloaded symbols in Kore definitions.

Patterns and metavariables for patterns It is in practice more common to use metavariables that range over all patterns. One typical example is axiom schemata. For example, $\vdash \varphi \to \varphi$ in which φ is the metavariable that ranges all well-formed patterns.

There has been an argument on whether metavariables for patterns should be sorted or not. Here are some observations. Firstly, since all symbols are decorated and not overloaded, in most cases, the sort of a metavariable for patterns can be inferred from its context. Secondly, the only counterexample against the first point that I can think of is when they appear alone, which is not an interesting case anyway. Thirdly, we do want the least amount of reasoning and inferring in using Kore definitions, so it breaks nothing if not helping things to have metavariables for patterns carrying their sorts.

Example 10.

$$A_1 = \{\mathsf{merge}(\mathsf{h1}, \mathsf{h2}) = \mathsf{merge}(\mathsf{h2}, \mathsf{h1})\}$$
$$A_2 = \{\forall \mathsf{h1} \forall \mathsf{h2}. \mathsf{merge}(\mathsf{h1}, \mathsf{h2}) = \mathsf{merge}(\mathsf{h2}, \mathsf{h1})\}$$
$$A_3 = \{\mathsf{merge}(\varphi, \psi) = \mathsf{merge}(\psi, \varphi)\}$$

All three theories are equal. It is easier to see that fact from a model theoretic point of view, since all theories require that the interpretation of merge is commutative and nothing more. On the other hand, it is not straightforward to obtain that conclusion from a proof theoretic point of view. For example, to deduce $\mathsf{merge}(\mathsf{list}(\mathsf{one}, \mathsf{cons}(\mathsf{two}, \mathsf{epsilon})), \mathsf{top}) = \mathsf{merge}(\mathsf{top}, \mathsf{list}(\mathsf{one}, \mathsf{cons}(\mathsf{two}, \mathsf{epsilon})))$ needs only one step in A_3, but will need a lot more in either A_1 or A_2, because one cannot simply substitute any patterns for universal quantified variables in matching logic.

This is a nice example that shows the power of metavariables and how they greatly shorten formal proofs, but this is not a good example to convince people that we need metavariables for patterns in order to embrace more expressiveness. There is an example in the theory LAMBDA that clearly shows metavariables give us more expressiveness power, and we will cover that example in later sections.

2 Binders

In matching logic there is a unified representation of binders. We will be using the theory of lambda calculus LAMBDA as an example in this section. Recall that the syntax for untyped lambda calculus is

$$\Lambda ::= V \mid \lambda V.\Lambda \mid \Lambda\ \Lambda$$

where V is a countably infinite set of atomic λ-terms. The set of all λ-terms is the smallest set satisfying the above grammar.

The theory LAMBDA in matching logic has one sort Exp for lambda expressions. It also has in its signature a binary symbol #lambda that builds a λ-terms, and a binary symbol app for lambda applications. To mimic the binding behavior of λ in lambda calculus, we define syntactic sugar $\mathsf{lambda} x.e = \exists x. \#\mathsf{lambda}(x, e)$ and $e_1 e_2 = \mathsf{app}(e_1, e_2)$ in theory LAMBDA. Notice that by defining lambda as a syntactic sugar using existential quantifier, we get alpha-renaming for free. The β-reduction is captured in LAMBDA by the next (β) axiom:

$$(\mathsf{lambda} x.e)e' = e[e'/x] \quad \text{, where } e \text{ and } e' \text{ are metavariables for } \lambda\text{-terms.}$$

We have two important observations about the (β) axiom. Firstly, metavaraibles e and e' cannot be replaced by matching logic variables, because λ-terms in matching logic are (often) not functional patterns. Secondly, metavariables e and e' cannot range over all patterns of sort Exp, but only those which are (syntactic sugar) of λ-terms. Allowing e and e' range over all patterns of Exp in the (β) axiom will quickly lead to an inconsistent theory, because the next contradiction becomes an instance of the (β) axiom.

$$\bot \overset{(\mathrm{N})}{=} \mathsf{app}((\mathsf{lambda}x.\top), \bot) \overset{(\beta)}{=} \top.$$

The above example is a strong evidence that we need metavariables that range over only a restricted set of patterns instead of all of them. Such a restricted set of patterns is called the *range* (or maybe *domain*?) of metavariables, which we will formally define later in this section. However, before we get into that, here are some observations that against having such restricted metavariables.

1. **We found no example that has such inconsistency issue except theories that contain binders and applications. Therefore, restricted metavariables, even if we do introduce them in the Kore language, will be used in a quite limited way in defining theories that have binders, such as the lambda calculus and the theory of contexts, and will hardly be used in theories that don't have binders;**

2. **The reason why we need metavariables is because binding structures, such as λ-terms, are defined in matching logic as syntactic sugar with existential quantification. This makes those binding structures, such as λ-terms, no longer matching logic terms, and thus cannot substitute for matching logic variables, while they can (and should) be able to substitute for variables in their original calculus.**

3. **Introducing restrictive metavariables does solve the inconsistency issue, but it is apparently not the only solution. An alternative solution requires us to reexamine binders and binding structures in matching logic. If we could find a way to encode, say λ-terms, as *functional patterns* in matching logic, then using variables is sufficient, and we can get rid of restrictive metavariables.**

4. **Having a unified theory of binders in matching logic is a good idea. Splitting binding structures into the process of constructing a term and the process of create a binding is also a promising way to go. We know how to construct a term in matching logic by simply using symbols. The issue is how to create a binding between variables and terms (or patterns in general).**

5. **A quantifier, given it a universal or existential one, creates a binding between a variable and a pattern *and does something more*. According the semantics of matching logic, A universal quantifier creates the binding and calculates the big conjunction**

of all instances while an existential quantifier creates the binding and calculates the big disjunction of them.

6. **On the other hand, most binders that we use simply construct a term and create a binding between a variable and the term, and we should not pour any semantics upon them. However, when we use the existential quantifier to create a binding relation, we do put extra semantics upon those user-defined binders. That is the fundamental reason why λ-terms, even though they should be terms, become nonfunctional patterns in matching logic as we have seen, and thus we need to design the Kore language to support restricted metavariables.**

7. **Why not introduce a "pure" binder that creates a binding between variables and patterns and does nothing else at all? All other binders such as the λ in the lambda calculus, the μ and ν in theories of fixed point, the γ in theory of contexts, and even the universal quantifier $\forall$ and existential quantifier $\exists$, can all be defined using that "pure" binder.**

Definition 11 (Restricted metavariables)**.** Let φ be a metavariable of sort $s \in S$. The range of φ is a subset of all patterns of the sort s. We write $\varphi :: R$ if the range of φ is $R \subseteq \text{Pattern}_s$.

Definition 12 (Common ranges of metavariables)**.**

- The full range Pattern_s;
- The logic-free range (variables plus symbols without logic connectives);
- The ground logic-free range (symbols only);
- The variable range Var_s;

Remark 13. We need to design a syntax for specifying ranges of metavariables in the Kore language.

Remark 14. We have not proved that matching logic is a conservative extension of untyped lambda calculus, which bothers me a lot. I will remain skeptical about everything we do in this section until we prove that conservative extension result.

Rewriting logic

3 Contexts

Fix a signature (S, Σ). For each sort $s \in S$, introduce an infinite number of *hole* variables, written as $\square_1, \square_2, \cdots$. Think of hole variables as normal matching logic variables, but lie in a disjoint namespace. Extend the grammar by adding the following.

$$\begin{aligned} P ::= {} & \cdots \\ & \mid \Box \\ & \mid \gamma\Box.P \\ & \mid P[P']. \end{aligned}$$

The sort of $\gamma\Box.P$ is the sort of P. The sort of $P[P']$ is the sort of P, too. Think of $\gamma\Box$ as a binder. Alpha-renaming is always assumed. Patterns of the form $\gamma\Box.P$ are often called *contexts*, denoted by $C, C_0, C_1, \ldots$. The pattern $P[P']$ is often called an *application*. The $\_[\_]$ operator is left associative.

Definition 15. The context $\gamma\Box.\Box$ is called the identity context, denoted as I. Identity context has the axiom schema $\mathsf{I}[P] = P$ where P is any pattern.

Example 16. $\mathsf{I}[\mathsf{I}] = \mathsf{I}$.

Example 17. Consider a signature (S, Σ) of a simple imperative programming language, with $S = \{BExp, Pgm\}$, and $\Sigma = \{skip, ite, seq, true, false\}$. Add axiom schemata

$$ite(C_1[B], P, Q) = (\gamma\Box.ite(C_1[\Box], P, Q))[B]$$

and

$$seq(C_2[P], Q) = (\gamma\Box.seq(C_2[\Box], Q))[P],$$

where P, Q are Pgm patterns, B is a $BExp$ pattern, C_1 is a $BExp$ context, and C_2 is a Pgm context.

Suppose we have the rewrite rules (schemata):

- $C[ite(true, P, Q)] \Rightarrow C[P]$,
- $C[ite(false, P, Q)] \Rightarrow C[Q]$,
- $C[seq(skip, Q)] \Rightarrow C[Q]$.

Example (a). Rewrite $seq(skip, skip)$.

$$\begin{aligned} seq(skip, skip) &= \mathsf{I}[seq(skip, skip)] \\ &\Rightarrow \mathsf{I}[skip] \\ &= skip. \end{aligned}$$

Example (b). Rewrite $ite(true, P, Q); R$.

$$\begin{aligned} seq(ite(true, P, Q), R) &= seq(\mathsf{I}[ite(true, P, Q)], R) \\ &= (\gamma\Box.seq(\mathsf{I}[\Box], R))[ite(true, P, Q)] \\ &\Rightarrow (\gamma\Box.seq(\mathsf{I}[\Box], R))[P] \\ &= seq(\mathsf{I}[P], R) \\ &= seq(P, R). \end{aligned}$$

Definition 18. Let $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ is an n-arity symbol. We say σ is *active* on its ith argument ($1 \leq i \leq n$), if $\sigma(P_1, \ldots, C[P_i], \ldots, P_n) = (\gamma\square.\sigma(P_1, \ldots, C[\square], \ldots, P_n))[P_i]$. Orienting the equation from the left to the right is often called *heating*, while orienting from the right to the left is called *cooling*.

Example 19. Suppose f and g are binary symbols who are active on their first argument. Suppose a, b are constants, and x is a variable. Let $\square_1$ and $\square_2$ be two hole variables. Define two contexts $C_1 = \gamma\square_1.f(\square_1, a)$ and $C_2 = \gamma\square_2.g(\square_2, b)$.

Because f is active on the first argument,

$$\begin{aligned} C_1[\varphi] &= (\gamma\square_1.f(\square_1, a))[\varphi] \\ &= (\gamma\square_1.f(\mathsf{I}[\square_1], a))[\varphi] \\ &= f(\mathsf{I}[\varphi], a) \\ &= f(\varphi, a), \text{ for any pattern } \varphi. \end{aligned}$$

And for the same reason, $C_2[\varphi] = g(\varphi, b)$. Then we have

$$\begin{aligned} C_1[C_2[x]] &= C_1[f(x, a)] \\ &= g(f(x, a), b). \end{aligned}$$

On the other hand,

$$\begin{aligned} g(f(x, a), b) &= g(C_1[x], b) \\ &= (\gamma\square.g(C_1[\square], b))[x] \\ &= (\gamma\square.g(f(\square, a), b))[x]. \end{aligned}$$

Therefore, the context $\gamma\square.g(f(\square, a), b))$ is often called the *composition* of C_1 and C_2, denoted as $C_1 \circ C_2$.

Example 20. Suppose f is a binary symbol with all its two arguments active. Suppose C_1 and C_2 are two contexts and a, b are constants. Then easily we get

$$\begin{aligned} f(C_1[a], C_2[b]) &= (\gamma\square_2.f(C_1[a], C_2[\square_2]))[b] \\ &= (\gamma\square_2.((\gamma\square_1.f(C_1[\square_1], C_2[\square_2]))[a]))[b]. \end{aligned}$$

What happens above is similar to *curring* a function that takes two arguments. It says that there exists a context C_a, related with C_1, C_2, f and a of course, such that $C_a[b]$ returns $f(C_1[a], C_2[b])$. The context C_a has a binding hole $\square_2$, and a body that itself is another context C'_a applied to a. In other words, there exists C_a and C'_a such that

- $f(C_1[a], C_2[b]) = C_a[b]$,
- $C_a = \gamma\square_2.(C'_a[a])$,
- $C'_a = \gamma\square_1.f(C_1[\square_1], C_2[\square_2])$.

A natural question is whether there is a context C such that $C[a][b] = f(C_1[a], C_2[b])$.

Proposition 21. *$C_1[C_2[\varphi]] = C[\varphi]$, where $C = \gamma\square.C_1[C_2[\square]]$.*

3.0.1 Normal forms

In this section, we consider *decomposition* of patterns. A decomposition of a pattern P is a pair $\langle C, R \rangle$ such that $C[R] = P$. Let us now consider patterns that do not have logical connectives.

Fixed points