Foundation of Language-Independent Verification

Xiaohong Chen

University of Illinois at Urbana-Champaign, USA
Email address: xc3@illinois.edu

Abstract. content...

Contents

Todo list

CHAPTER 1

Preliminaries

CHAPTER 2

Matching logic

2.1. The need for another logic

2.2. Syntax and semantics

A signature is a triple $\mathbb{\Sigma} = (\textsc{Var}, S, \Sigma)$ where $\textsc{Var}$ is a countably infinite set of variable names, S is an nonempty countable (finite or infinite) sort set, and $\Sigma = \{\Sigma_{s_1 \ldots s_n, s}\}_{s_1, \ldots, s_n, s \in S}$ is an $(S^* \times S)$-indexed countable (finite or infinite) symbol set. We write $\Sigma_{s_1 \ldots s_n, s}$ as $\Sigma_{\lambda, s}$ if $n = 0$. When $\textsc{Var}$ is clear from the context, we often omit it and just write the signature $\mathbb{\Sigma} = (S, \Sigma)$. If S is also clear, we refer to a signature by just Σ. The set of all $\mathbb{\Sigma}$-patterns of sort s, denoted as $\textsc{Pattern}_s(\mathbb{\Sigma})$ or just $\textsc{Pattern}_s$ when the signature is clear from the context, is defined by the following grammar:

$$
\begin{array}{rcll}
\varphi_s & ::= & x{:}s \text{ with } x \in \textsc{Var} & \text{// Variable} \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \text{ with } \sigma \in \Sigma_{s_1 \ldots s_n, s} & \text{// Structure} \\
& | & \neg \varphi_s & \text{// Complement} \\
& | & \varphi_s \wedge \varphi_s & \text{// Intersection} \\
& | & \exists x{:}s'.\varphi_s \text{ with } s' \in S \text{ (no need be the same as } s) & \text{// Binding}
\end{array}
$$

The set of all variables of sort s is denoted as $\textsc{Var}_s$. Let $\textsc{Var} = \{\textsc{Var}_s\}_{s \in S}$ and $\textsc{Pattern} = \{\textsc{Pattern}_s\}_{s \in S}$ be the S-indexed family set of all variable and patterns. For simpler notation, we often blur the distinction between a family of sets and their union, and use $\textsc{Var}$ and $\textsc{Pattern}$ to denote the set of all variables and patterns respectively. We write $\varphi \in \textsc{Pattern}$ to mean that φ is a pattern, and $\varphi_s \in \textsc{Pattern}$ or $\varphi \in \textsc{Pattern}_s$ to mean that it has sort s. Similarly, $\sigma \in \Sigma$ means σ is a symbol. If $\sigma \in \Sigma_{\lambda, s}$, we say σ is a constant symbol of sort s, and we write σ instead of $\sigma()$. We often drop the sort when writing variables, so instead of $x{:}s$, we just write x. We can define the conventional notions of free variables and alpha-renaming as in first-order logic. We write $\varphi[\psi/x]$ for variable-capture-free substitution, in which alpha-renaming happens implicitly to prevent free variable capturing.

A $\mathbb{\Sigma}$-model (or simply a model) is a pair $\mathcal{M} = (M, \_{\mathcal{M}})$ where $M = \{M_s\}_{s \in S}$ is an S-indexed family of nonempty carrier sets and $\_{\mathcal{M}} = \{\sigma_{\mathcal{M}}\}_{\sigma \in \Sigma}$ maps every symbol $\sigma \in \Sigma_{s_1, \ldots, s_n, s}$ to a function $\sigma_{\mathcal{M}} \colon M_{s_1} \times \cdots \times M_{s_n} \to 2^{M_s}$, where 2^{M_s} means the set of all subsets of M_s. In particular, each constant symbol $\sigma \in \Sigma_{\lambda, s}$ maps to a subset $\sigma_{\mathcal{M}} \subseteq M_s$. An $\mathcal{M}$-valuation (or simply a valuation) is a mapping $\rho \colon \textsc{Var} \to M$ such that $\rho(x{:}s) \in M_s$ for every sort $s \in S$. Two valuations ρ_1 and ρ_2 are x-equivalent for some variable x, denoted as $\rho_1 \overset{x}{\sim} \rho_2$, if $\rho_1(y) = \rho_2(y)$ for every y distinct from x. A valuation ρ can be extended to a mapping $\bar{\rho} \colon \textsc{Pattern} \to 2^M$ such that $\bar{\rho}(\varphi_s) \subseteq M_s$, in the following inductive way:

- $\bar{\rho}(x) = \{\rho(x)\}$, for every $x \in \textsc{Var}_s$;
- $\bar{\rho}(\sigma(\varphi_1, \ldots, \varphi_n)) = \sigma_{\mathcal{M}}(\bar{\rho}(\varphi_1), \ldots, \bar{\rho}(\varphi_n))$, for every $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ and appropriate $\varphi_1, \ldots, \varphi_n$
- $\bar{\rho}(\neg \varphi) = M_s \setminus \bar{\rho}(\varphi)$, for every $\varphi \in \textsc{Pattern}_s$;
- $\bar{\rho}(\varphi_1 \wedge \varphi_2) = \bar{\rho}(\varphi_1) \cap \bar{\rho}(\varphi_2)$, for every φ_1, φ_2 of the same sort;

- $\bar{\rho}(\exists x.\varphi) = \bigcup\{\overline{\rho'}(\varphi) \mid \text{for every } \rho' \overset{x}{\sim} \rho\}$.

We also need to give some intuition for "matching" (the patterns are "matched" by the values in their interpretation, which becomes literal when the model contains terms. Intuitively, $\bar{\rho}(\varphi)$ is the set of elements that match the pattern φ. Derived constructs are defined as follows for convenience:

$$\begin{array}{rcl@{\qquad}rcl} \top_s & \equiv & \exists x{:}s.x{:}s & \bot_s & \equiv & \neg\top_s \\ \varphi_1 \vee \varphi_2 & \equiv & \neg(\neg\varphi_1 \wedge \neg\varphi_2) & \varphi_1 \rightarrow \varphi_2 & \equiv & \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \leftrightarrow \varphi_2 & \equiv & (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) & \forall x.\varphi & \equiv & \neg(\exists x.\neg\varphi) \end{array}$$

Interested readers are encouraged to prove these derived constructs have the intended semantics, or refer to [**?**] for details. We often drop the sort subscripts when there is no confusion.

Given a model $\mathcal{M}$ and a valuation ρ, we say $\mathcal{M}$ and ρ satisfy a pattern φ_s, denoted as $\mathcal{M}, \rho \vDash \varphi_s$, if $\bar{\rho}(\varphi) = M_s$. We say $\mathcal{M}$ satisfies φ_s or φ_s holds in $\mathcal{M}$, denoted as $\mathcal{M} \vDash \varphi_s$, if $\mathcal{M}, \rho \vDash \varphi_s$ for every valuation ρ. We say φ_s is valid if it holds in every model. Let Γ be a pattern set. We say $\mathcal{M}$ satisfies Γ, if $\mathcal{M} \vDash \varphi$ for every $\varphi \in \Gamma$. We say Γ semantically entails φ, denoted as $\Gamma \vDash \varphi$, if for every model $\mathcal{M}$ such that $\mathcal{M} \vDash \Gamma$, $\mathcal{M} \vDash \varphi$. When Γ is the empty set, we abbreviate $\emptyset \vDash \varphi$ as just $\vDash \varphi$, which is equivalent to say φ is valid.

Say that we use mathcal fonts $\mathcal{M}, \mathcal{I}, \ldots$ to denote models, and the corresponding M, $I, \ldots$ to denote carrier sets. Use capital Greek letters $\Gamma, \Delta, T, \ldots$ to denote pattern set. In particular, use $\mathcal{I}$ to denote intended models and $\mathcal{S}$ to denote standard models.

Given a signature $\mathbb{\Sigma}$, matching logic gives us all $\mathbb{\Sigma}$-models. Sometimes, we are only interested in some models, instead of all models. There are typically two ways to restrict models. One way is to define a syntactic matching logic theory $(\mathbb{\Sigma}, H)$ where H is a set of patterns called axioms. A model $\mathcal{M}$ belongs to the theory $(\mathbb{\Sigma}, H)$ if $M \vDash H$. Syntactic theories are preferred if the models of interest can be axiomatized by a recursively enumerable set H. Most syntactic theories defined in this paper have finite axiom sets. Alternatively, we can define a semantic matching logic theory $(\mathbb{\Sigma}, C)$ where C is a collection of $\mathbb{\Sigma}$-models. A model $\mathcal{M}$ belongs to the theory $(\mathbb{\Sigma}, C)$ if $\mathcal{M} \vDash \varphi$ for all φ that holds in all models in C. Usually, the collection C is a singleton set containing exactly one model $\mathcal{I}$, often referred as the intended model or the standard model, and we abbreviate $(\mathbb{\Sigma}, \{\mathcal{I}\})$ as $(\mathbb{\Sigma}, \mathcal{I})$. Semantic theories are preferred if the intended model is not axiomatized by any recursively enumerable set of axioms; this is often the case for initial algebra semantics or domains which are defined by induction, such as natural numbers (with multiplication) and finite maps. We will see an example of semantic theories in Section **??**.

Syntactic theories support a notion of proofs (see Section **??**) as they have an axiom set, but semantic theories offer arbitrary expressiveness. They both are means to restricting models, and we simply say "theories" when their distinction is not the emphasis.

2.2.1. Known results about matching logic expressiveness power. I think it is also important to state that when all models are assumed, then ML has been shown to have the same expressiveness as FOL=, but like it is the case with capturing SL, or in FOL with induction, or in initial algebra semantics, one can reduce the set of models and thus get arbitrary expressiveness.

A few important logics and calculus are shown to be definable in matching logic, including propositional calculus, predicate logic, algebraic specification, first-order logic with equality, modal logic S5, and separation logic. In this section, we only discuss a few of them and refer interested readers to [**?**] for more details.

2.3. Sound and complete deduction

CHAPTER 3

Define Other Logics and Calculi

The purpose of this section is to show that many important logics and calculi can be naturally defined/captured/subsumed in matching logic as theories.

Find a better word.

This chapter is organized in sections, in each of which we discuss one logic or calculus. In every section, we start by an overview of the target logic, and then show how it is captured in matching logic, followed by discussions and important theorems about the definition.

3.1. Untyped lambda calculus

3.1.1. An overview. Untyped lambda calculus, initially proposed by Alonzo Church in the 1930s, captures computation using function abstraction and application. The syntax of untyped lambda calculus is simple, except the binding behavior and variable-capture-free substitution, which we already saw when introducing the matching logic syntax, so we assume readers are familiar with it. In short, alpha-conversion is assumed, and two lambda terms (defined below) are regarded as *the same term* if they are alpha-equivalent. Given and fix a countably infinite set of variables. A lambda term is either a variable, or an application ee' where e and e' are lambda terms, or a function abstraction $\lambda x.e$ where x is a variable and e is a lambda term. We abbreviate $\lambda x_1 \dots \lambda x_n.e$ as $\lambda x_1 \dots x_n.e$. The scope of λ goes as far as possible to the right, so $\lambda xy.xy$, for example, means $\lambda xy.(xy)$. The set of all lambda terms is Λ.

The famous beta-reduction axiom, as shown below, gives the intuitive meaning of functions.

$$(\beta) \qquad (\lambda x.e)e' = e[e'/x]. \tag{β}$$

We leave two remarks. Firstly, (β) is an axiom schema where x is any lambda calculus variable and e, e' are any lambda terms, so there are infinitely many instances of it. Secondly, equality (=) is not part of the syntax of the calculus, and one should think of (β) defining a binary relation on Λ, the set of all lambda terms.

We are interested in those binary relations on Λ that are congruence relations with respect to function abstraction and application. We call them *lambda theories*, if these congruence relations contain all instances of (β). The set of all lambda theories is $\lambda\mathcal{T}$, which is a nonempty set, as the total relation Λ^2, a trivial lambda theory, is in it. We call Λ^2 an inconsistent lambda theory, because it makes any two lambda terms equal. Any lambda theories that are not Λ^2 are called consistent theories. One can show that the intersection of arbitrarily many lambda theories is also a lambda theory. Denote the intersection of all lambda theories, i.e. the smallest lambda theory, as λ_β.

The set of all lambda theories $\lambda\mathcal{T}$ forms a complete lattice.

3.2. One-path reachability logic

We consider one-path reachability logic as introduced in [**SPY**$^+$**16**]. This section is a very short but comprehensive presentation about how to define reachability logic in matching logic.

Reachability logic is parametric on three components: (1) a many-sorted signature (S, Σ) which contains a distinguished sort Cfg, (2) a sort-wise countably infinite set VAR of variables, and (3) a partial Σ-algebra $\mathcal{T}$. Validity and provability of reachability logic are denoted as $\models_{\mathsf{RL}}$ and $\vdash_{\mathsf{RL}}$ respectively.

3.2.1. Reachability logic as a matching logic theory. We define a matching logic theory RL that faithfully captures reachability logic.

Sorts in RL contains all that in S, including the distinguished sort Cfg.

Symbols in RL contains all functions and partial functions in Σ. In addition, it contains a unary symbol $\bullet \in \Sigma_{Cfg,Cfg}$.

Axioms in RL contains all equations $t_1 = t_2$ that are valid in the algebra $\mathcal{T}$.

NOTATION 3.1. We adopt the following notations in RL.

$$\bullet\varphi \equiv \bullet(\varphi)$$
$$\circ\varphi \equiv \neg\bullet\neg\varphi$$
$$\varphi \Rightarrow^{\exists} \psi \equiv$$

Given these three components, we can define a matching logic theory RL. RL has the same variable set VAR and the sort set S. It contains all symbols in Σ plus a unary symbol $\bullet \in \Sigma_{Cfg}$

The matching logic theory contains The matching logic theory of one-path reachability logic contains a unary symbol $\bullet$.

Bibliography

[SPY+16] Andrei Ştefănescu, Daejun Park, Shijiao Yuwen, Yilong Li, and Grigore Roşu, *Semantics-based program verifiers for all languages*, ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016), ACM, November 2016, pp. 74–91.