CALCO'19, Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik 139, pp 1-4. 2019

Matching μ-Logic: Foundation of $\mathbb{K}$ Framework

Xiaohong Chen
University of Illinois at Urbana-Champaign, USA
http://fsl.cs.illinois.edu/~xchen
xc3@illinois.edu

Grigore Roşu
University of Illinois at Urbana-Champaign, USA
http://fsl.cs.illinois.edu/~grosu
grosu@illinois.edu

Abstract

$\mathbb{K}$ framework is an effort in realizing the ideal language framework where programming languages must have formal semantics and all languages tools are automatically generated from the formal semantics in a correct-by-construction manner at no additional costs. In this extended abstract, we present matching μ-logic as the foundation of $\mathbb{K}$ and discuss some of its applications in defining constructors, transition systems, modal μ-logic and temporal logic variants, and reachability logic.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Matching μ-logic, Program verification, Reachability logic

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.1

Category Invited Paper

1 Introduction

In an ideal language framework, all programming languages must have formal semantics and all language tools are automatically generated from the formal semantics in a correct-by-construction manner at no additional costs. $\mathbb{K}$ framework (`www.kframework.org`) is an almost 20-year continuous effort in realizing the ideal language framework. Many real-world languages such as C [5], Java [1], JavaScript [9] as well as the emerging blockchain languages such as EVM [6], have had their formal semantics successfully defined in $\mathbb{K}$ and language tools such as parsers, interpreters, and deductive verifiers have been automatically generated by $\mathbb{K}$.

In terms of program verification, $\mathbb{K}$ adopts a language-independent approach that is different from the classic language-specific approaches such as Hoare-style verification [7], where different languages have different program logics and thus different verifiers. Instead, the current $\mathbb{K}$ implementation uses *matching logic* [10] to specify static structures of programs and *reachability logic* [11] to reason about dynamic reachability properties for all languages. Formal semantics are given as *theories* in these logics, so their fixed and thus language-independent proof systems achieve semantic-based program verification for all languages [4].

As its name suggests, reachability logic can only express reachability properties, which limits $\mathbb{K}$ to verifying, for instance, liveness properties, which are beyond reachability logic but can naturally be expressed in temporal logics such as linear temporal logic (LTL) or computation tree logic (CTL). To overcome this limitation, we recently proposed *matching μ-logic* [2], which is a powerful logic that subsumes not only matching logic and reachability logic, but also first-order logic with least fixpoints, modal μ-logic, many variants of temporal logics, dynamic logic, and others (see Fig. 1). This demonstrates that matching μ-logic can serve as the uniform foundation of an ideal language framework.

Here we only present matching μ-logic by examples and show its application in specifying and reasoning about constructors, transition systems, and reachability. For more details see [2, 3].

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).
Editors: Markus Roggenbach and Ana Sokolova; Article No. 1; pp. 1:1–1:4
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

CALCO'19, Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik 139, pp 1-4. 2019

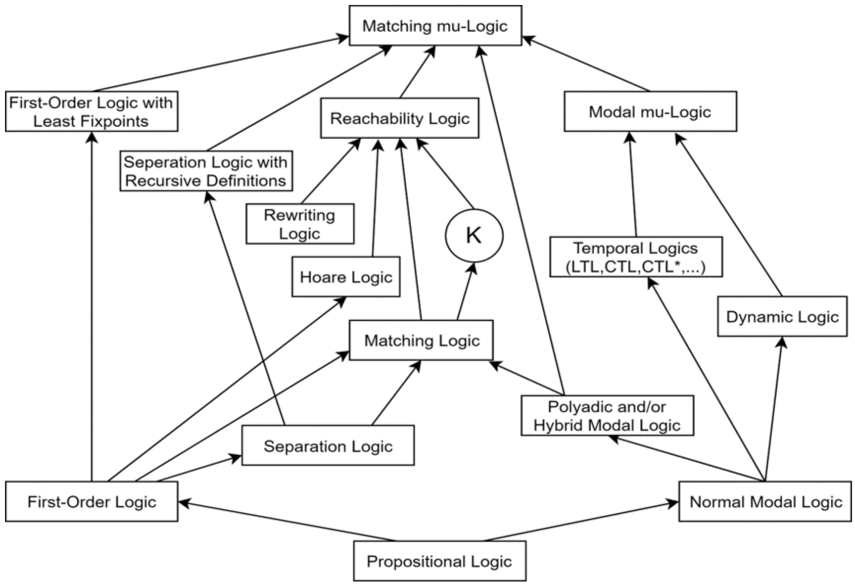

Figure 1 Many popular logics can be defined in matching μ-logic as theories and notations [2]; the current $\mathbb{K}$ implementation (denoted as the node labeled "K") is so far the best effort in implementing reachability logic reasoning and will eventually be lifted to the same level as matching μ-logic.

2 Matching μ-Logic Examples

Preliminaries and basic examples

Matching logic (the version without μ) is a variant of many-sorted first-order logic (FOL) which makes no distinction between functions and predicates but uses *symbols* to uniformly build *patterns* that can represent static structures, dynamic properties, and logic constraints. Matching μ-logic extends matching logic with the least fixpoint μ-binder as in modal μ-logic [8], which can build *inductive patterns* to represent inductive and co-inductive data structures and recursive properties and logical constraints.

Intuitively speaking, a pattern evaluates to the set of elements *matching* it. For example:

- x, called an *element variable*, is matched by exactly one element x;
- X, called a *set variable*, is matched by any set X of elements;
- $\mathit{succ}(x)$ is matched by the successor(s) of x; here *succ* is a *symbol* that builds structures;
- $\exists x\,.\,\mathit{succ}(x)$ is matched by the successor of *some* x, i.e., all successors;
- $\mathit{zero} \vee \exists x\,.\,\mathit{succ}(x)$ is matched by either zero or successors;
- $\top \equiv \exists x\,.\,x$ is matched by x for some x, i.e., everything; $\bot \equiv \neg\top$ is matched by nothing;
- $\mathit{list}(x)$ is matched by all linked lists in the heap starting at pointer x; *list* is also a symbol;
- $\mathit{list}(x) \wedge \mathit{prime}(x)$, same as above but with prime x;
- $\mu N\,.\,\mathit{zero} \vee \mathit{succ}(N)$ is matched by all natural numbers zero, $\mathit{succ}(\mathit{zero})$, $\mathit{succ}(\mathit{succ}(\mathit{zero}))$, $\ldots$; this is because the μ-binder denotes the least set N w.r.t. set containment such that $N = \mathit{zero} \vee \mathit{succ}(N)$; in other words, N is the least set closed under *zero* and *succ*.

Constructors

The last example above $\mu N\,.\,\mathit{zero} \vee \mathit{succ}(N)$ that is matched by all natural numbers can be easily generalized to deal with any constructor set $C = \{c_i \mid c_i \text{ is a constructor of arity } n_i\}$, where the pattern $\mu D\,.\,\bigvee_{c_i \in C} c_i(\underbrace{D,\ldots,D}_{n_i \text{ times}})$ evaluates to the least set that is closed under all constructors in C, yielding the set of all terms generated by C.

CALCO'19, Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik 139, pp 1-4. 2019

Transition systems and temporal logics

A transition system (S, R) is a pair of a state set S and a transition relation $R \subseteq S \times S$. In matching μ-logic, transition systems can be captured by one unary symbol $\bullet$ called *one-path next* (we write $\bullet\varphi$ instead of $\bullet(\varphi)$) with the intended interpretation that $\bullet\varphi$ is matched by all *predecessors* of those matching φ:

$$\begin{array}{cccccccl} \cdots & s & \xrightarrow{R} & s' & \xrightarrow{R} & s'' & \cdots & \text{// states} \\ & \bullet\bullet\varphi & & \bullet\varphi & & \varphi & & \text{// patterns} \end{array}$$

In other words, a state matches $\bullet\varphi$ iff it has *one next* state that matches φ. Its dual *all-path next* $\circ\varphi \equiv \neg\bullet\neg\varphi$ is matched by those states whose *next states all* match φ (see Fig. 2).

We can define patterns that represent more complex dynamic properties. For example,

- $\bullet\top$ is matched by all non-terminal states;
- $\circ\bot$ is matched by all terminal states;
- $\Diamond\varphi \equiv \mu X \,.\, \varphi \vee \bullet X$ is matched by all states that eventually reach φ on some path;
- $\Box\varphi \equiv \nu X \,.\, \varphi \wedge \circ X$ is matched by all states that always stay in φ on all paths; ν-binder is the dual of μ-binder that builds greatest fixpoints instead of least fixpoints, defined as usual: $\nu X \,.\, \varphi \equiv \neg\mu X \,.\, \neg\varphi[\neg X/X]$ where $\_[\_/\_]$ is the standard capture-avoiding substitution;
- $\mathsf{WF} \equiv \mu X \,.\, \circ X$ is matched by all states that are well-founded, i.e., have no infinite paths.

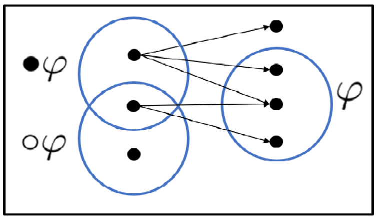

Figure 2 One/All-path next.

We point out that the above definitions are standard definitions in modal μ-logic. Since, as is well known, modal μ-logic subsumes many variants of temporal logic such as LTL and CTL and that matching μ-logic subsumes modal μ-logic (see [2, Section VII]), there is no surprise that matching μ-logic also subsumes LTL and CTL. What is interesting is that it only requires a few natural and intuitive axioms to faithfully capture LTL and CTL in matching μ-logic, as summarized below:

Target logic	Assumption on traces	Axioms required in matching μ-logic
Modal μ-logic	Any traces, no assumptions	No axioms
Infinite-trace LTL	Infinite and linear traces	(Inf) + (Lin)
Finite-trace LTL	Finite and linear traces	(Fin) + (Lin)
CTL	Infinite traces	(Inf)

where (Inf) is the pattern/axiom $\bullet\top$ stating that all states are non-terminal states, (Fin) is the pattern/axiom $\mathsf{WF} \equiv \mu X \,.\, \circ X$ stating that all states are well-founded, and (Lin) is the pattern/axiom $\bullet X \to \circ X$ enforcing the linear paths: X holds on one next state implies X holds on all next states.

In conclusion, modal μ-logic is the empty theory over a unary symbol $\bullet$ that contains no axioms. Adding (Inf) yields precisely CTL. Adding (Inf) yields precisely infinite-trace LTL and replacing (Inf) with (Fin) yields finite-trace LTL. Therefore, matching μ-logic over the one-path next symbol $\bullet$ gives a playground for defining variants of temporal logics.

CALCO'19, Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik 139, pp 1-4. 2019

Reachability logic

Our last example is to define reachability properties $\varphi \Rightarrow \varphi'$, called *reachability rules* [11], in matching μ-logic using the one-path next symbol. Here, φ and φ' are matching logic patterns not containing μ that are matched by program configurations. The semantics of $\varphi \Rightarrow \varphi'$ is that for every configuration γ that matches φ, either it reaches some configuration γ' that matches φ' in finitely many steps, or it is not well-founded. In other words, reachability is like a "weak" eventuality statement that applies to only well-founded states. This suggests to define the derived construct "weak eventually" $\Diamond_w \psi \equiv \nu X \,.\, \psi \vee \bullet X$, which is like the definition of the normal eventually $\Diamond \psi$ but replacing μ by ν, and define $\varphi \Rightarrow \varphi' \equiv \varphi \to \Diamond_w \varphi'$. We can prove that $\Diamond_w \psi = \Diamond \psi \vee \neg \mathsf{WF}$, i.e., it indeed captures the semantics of (partial correctness) reachability, and thus our definition of reachability logic is faithful.

3 Conclusion

In this extended abstract, we presented matching μ-logic as the foundation of $\mathbb{K}$ and discussed some of its applications to defining constructors, transition systems, modal μ-logic and temporal logic variants, and finally reachability logic.

References

1 Denis Bogdănaş and Grigore Roşu. K-Java: A complete semantics of Java. In *Proceedings of the 42^{nd} Symposium on Principles of Programming Languages (POPL'15)*, pages 445–456. ACM, 2015.

2 Xiaohong Chen and Grigore Roşu. Matching μ-logic. In *Proceedings of the 34^{th} Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'19)*, 2019.

3 Xiaohong Chen and Grigore Roşu. Matching μ-logic. Technical report, University of Illinois at Urbana-Champaign, 2019. URL: `http://hdl.handle.net/2142/102281`.

4 Andrei Ştefănescu, Daejun Park, Shijiao Yuwen, Yilong Li, and Grigore Roşu. Semantics-based program verifiers for all languages. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'16)*, pages 74–91. ACM, 2016.

5 Chris Hathhorn, Chucky Ellison, and Grigore Roşu. Defining the undefinedness of C. In *Proceedings of the 36^{th} annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, pages 336–345. ACM, 2015.

6 Everett Hildenbrandt, Manasvi Saxena, Xiaoran Zhu, Nishant Rodrigues, Philip Daian, Dwight Guth, Brandon Moore, Yi Zhang, Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KEVM: A complete semantics of the Ethereum virtual machine. In *Proceedings of the 2018 IEEE Computer Security Foundations Symposium (CSF'18)*. IEEE, 2018. URL: `http://jellopaper.org`.

7 C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

8 Dexter Kozen. Results on the propositional μ-calculus. In *Proceedings of the 9^{th} International Colloquium on Automata, Languages and Programming (ICALP'82)*, pages 348–359. Springer, 1982.

9 Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KJS: A complete formal semantics of JavaScript. In *Proceedings of the 36^{th} annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, pages 346–356. ACM, 2015.

10 Grigore Roşu. Matching logic. *Logical Methods in Computer Science*, 13(4):1–61, 2017.

11 Grigore Roşu, Andrei Ştefănescu, Ştefan Ciobâcă, and Brandon M. Moore. One-path reachability logic. In *Proceedings of the 28^{th} Symposium on Logic in Computer Science (LICS'13)*, pages 358–367. IEEE, 2013.