Verified Equivalence Checker for Omega-Regular Expressions

IONA KUHN, Saarland University, Germany

Additional Key Words and Phrases: Formal Verification, Coq, Omega-Regular Expressions, Languages over Infinite Words, Equivalence Checking

1 PROBLEM AND MOTIVATION

Formal language theory is an elegant tool used in many fields of computer science. In particular, languages over infinite words are a convenient tool to model reactive systems and hence find applications to verification and synthesis. In these applications, the correctness of algorithms is crucial. Thus, it is important to have trustworthy algorithms on languages.

Often, the most important problem is to decide language equality or inclusion, because many other problems can be reduced to it. For example, in model checking, a system and a given specification can both be represented as languages. These languages are then tested for language inclusion to check if the system matches the specification. Similarly, system equivalence can be reduced to language equivalence.

Languages can be represented in many ways, including, for example, automata, logics and expressions [1]. Automata and logics have already been studied extensively, including in proof assistants, resulting in a variety of trustworthy algorithms to decide equivalence on them. Omega-regular expressions, however, have been considered much less and are to our knowledge not yet formalized in a proof assistant. In contrast to automata, omega-regular expressions can be modified and manipulated syntactically. They are also more compact and easier to reason about, especially in a proof assistant. In particular, it is possible to use structural induction, a powerful reasoning tool, on expressions. Despite these advantages, there are so far no algorithms to decide equivalence for omega-regular expressions without translating them to some other representation.

In this work, we study the semantics of omega-regular expressions in the Coq proof assistant [8] as well as develop a new (and first to our knowledge) algorithm for deciding equivalence directly on omega-regular expressions. Furthermore, we formalize the algorithm in Coq and verify it. The equivalence checker is proven sound, but not complete, and extracted to an executable OCaml program. More concretely, we provide an algorithm such that the following theorem holds:

Theorem 1.1 (Soundness of Equivalence Checker). *For all omega-regular expressions e_1, e_2 holds: If the equivalence checker outputs "yes", then $\mathcal{L}_\omega(e_1) \equiv \mathcal{L}_\omega(e_2)$ (i.e. the languages are the same). If it outputs "no", then $\mathcal{L}_\omega(e_1) \not\equiv \mathcal{L}_\omega(e_2)$ (i.e. the languages are not the same).*

2 BACKGROUND AND RELATED WORK

Languages over infinite words have already been formalized in proof assistants. For example, Esparza et al. have written a fully verified executable LTL model checker in the proof assistant Isabelle/HOL [4]. They focused on the formalization of Büchi automata and the translation of LTL to Büchi automata. As part of their model checker, they have written an inclusion checker for Büchi automata.

Omega-regular expressions, which are also a representation for languages over infinite words, are not yet studied as much as logics and automata. However, a lot of work has been done on regular expressions, the corresponding representation for languages over finite words. In 1964, Brzozowski defined an algorithm to translate regular expressions into finite automata based on a notion of "derivatives" [3]. Independent from Brzozowski, McNaughton and Yamada also defined

Author's address: Iona Kuhn, ioku00001@stud.uni-saarland.de, Saarland University, Saarbrücken, Germany.

an algorithm to translate regular expressions into finite automata using a different approach [6]. This approach was formalized in Coq by Braibant and Pous [2]. They wrote an efficient Coq proof tactic to decide the equational theory of Kleene algebra, which subsumes the problem of deciding equivalence of regular expressions.

Rutten recognized in 1998 that one can also use derivatives to check equivalence on two regular expressions without translating them to automata [7]. Nipkow and Krauss then formalized the algorithm from Rutten in the proof assistant Isabelle/HOL. They obtained a fully verified executable algorithm to check equivalence of two regular expressions [5].

Thiemann and Sulzmann adapted the approach of Brzozowski for omega-regular expression and defined an algorithm using derivatives to translate an omega-regular expression to a Büchi automaton [9]. However, they do not provide an equivalence checker and their work is not formalized in a proof assistant.

3 APPROACH AND UNIQUENESS

3.1 Syntax and Semantics of Omega-Regular Expressions in Coq

The first step is to formalize the notion of omega-regular expressions and languages over infinite words in Coq. Omega-regular expressions are defined as follows:

$$e ::= \emptyset \mid e_1 + e_2 \mid r \cdot e \mid r^{\omega}$$

where r is a regular expression. Furthermore, the empty word ε is not allowed to be in the language of r in the last case. The main difference to regular expressions is the omega-operator, which describes infinite words constructed by an infinite succession of words of the language of r. Omega-regular expressions are defined as an inductive type in Coq.

The semantics decides given an infinite word and an omega-regular expression whether the word is in the language. The main concern when formalizing it is to define the semantics of the omega operator, which strongly depends on how we formalize infinite words. We tried two approaches to define words: functions and coinductive lists. From each definition of infinite words, a definition of the semantics for omega-regular expressions is derived, because the structure of the word influences the formalization of the semantics. After evaluating both approaches and proving their equivalence, we decided on the coinductive one. The main reason is that the coinductive approach leads to simpler definitions and simpler proofs. In particular, it enables proofs by coinduction. In contrast, in the functional approach, words are represented as natural numbered indexed sequences and proofs require explicit tracking of indices, which rapidly becomes overwhelming.

3.2 Equivalence Checker

The final goal is to develop an equivalence checker which given two omega-regular expressions decides whether the languages of the expressions are equivalent or not. The equivalence checker on regular expressions from [7] uses a concept of derivatives to incrementally build a bisimulation between two expressions (a bisimulation implies equivalence). This approach does not immediately translate to omega-regular expressions. We propose an alternative approach that extends the notion of derivatives to omega-regular expressions and relies on the following theorem:

Theorem 3.1. *The languages L_1 and L_2 are equivalent if and only if all derivatives of the two languages are equivalent.*

Our algorithm maintains a set of pairs of omega-regular expressions (called *equations*). At the beginning, the set only contains the equation with the two initial expressions. The algorithm

iterates over the set. At every iteration, one of the following transition rules is applied:

$$\textsc{delete}\ \frac{\mathcal{L}_\omega(e_1) \equiv \mathcal{L}_\omega(e_2)}{E \cup \{(e_1, e_2)\} \rightarrow_\Delta E} \qquad \textsc{clash}\ \frac{\mathcal{L}_\omega(e_1) \not\equiv \mathcal{L}_\omega(e_2)}{E \cup \{(e_1, e_2)\} \rightarrow_\Delta \bot}$$

$$\textsc{deriv}\ \frac{\forall a, \mathcal{L}_\omega(e_1^a) \equiv \Delta_a \mathcal{L}_\omega(e_1) \qquad \forall a, \mathcal{L}_\omega(e_2^a) \equiv \Delta_a \mathcal{L}_\omega(e_2)}{E \cup \{(e_1, e_2)\} \rightarrow_\Delta E \cup \bigcup_{a \in \Sigma} (e_1^a, e_2^a)}$$

The symbol $\rightarrow_\Delta$ denotes one step of the algorithm. E stands for a set of equations.

The DELETE-rule removes an equation from the set if the languages of the expressions are equivalent. The idea is to have some easy syntactic criteria to check equivalence, which are of course not complete as this would subsume the equivalence checker.

The DERIV-rule says that every equation in the set can be replaced with all its derivatives (see Theorem 3.1). The operator Δ_a takes a language and a symbol and returns the derived language with respect to the given symbol a. Note that there always exists a derivative and e_1^a/e_2^a (i.e. the derivatives of the expressions) can always be computed. The idea behind this rule is to replace the equation with easier equations such that the other two rules can be used.

The CLASH-rule is for detecting counterexamples. The idea is to have a collection of simple criteria which are sufficient to show that the two expressions do not describe the same language. This collection is of course also not complete, else this would subsume the equivalence checker.

In Coq, the equivalence checker is first formalized as a declarative version with the three given transition rules. We prove that the rules are sound in the following sense: If $\{(e_1, e_2)\} \rightarrow_\Delta^* \emptyset$, then $\mathcal{L}_\omega(e_1) \equiv \mathcal{L}_\omega(e_2)$, i.e. if the algorithm reaches the empty set, the two omega-regular expressions describe the same language. Similarly, if $\{(e_1, e_2)\} \rightarrow_\Delta^* \bot$ then $\mathcal{L}_\omega(e_1) \not\equiv \mathcal{L}_\omega(e_2)$. After that, an executable variant is written in Coq that deterministically applies one rule at every step of the algorithm. We prove that it corresponds to the declarative version. The executable variant implements the DELETE-rule using a syntactic equivalence check of the expressions as an easy criterion for language equivalence. While this algorithm is not complete, it is proven sound.

3.3 Executable Interface

We use the extraction mechanism of Coq to extract the equivalence checker into an executable OCaml program. On top of that, we wrote a small parser so that a user can use the equivalence checker as a standalone program by just giving the two omega-regular expressions that should be checked.

4 RESULTS AND CONTRIBUTIONS

In this work, we formalized the theory of omega-regular expressions in Coq. We developed a novel algorithm that checks equivalence of two omega-regular expressions and also formalized it in Coq. It is implemented as a verified executable algorithm and proven sound. Furthermore, the equivalence checker is extracted into OCaml as a verified standalone program.

At the moment, our equivalence checker is not complete. It could be improved by extending the syntactic check for equivalence and adding more criteria that allow to determine whether two expressions describe different languages. Another approach is to use the notion of partial derivatives described in [9], which, we hope, would allow us to recover completeness. This is left as future work.

REFERENCES

[1] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press, Cambridge, Mass. [u.a.]. http://www.ulb.tu-darmstadt.de/tocs/19825007X.pdf

[2] Thomas Braibant and Damien Pous. 2010. An Efficient Coq Tactic for Deciding Kleene Algebras. In *Proceedings of the First International Conference on Interactive Theorem Proving* (Edinburgh, UK) *(ITP'10)*. Springer-Verlag, Berlin, Heidelberg, 163–178. https://doi.org/10.1007/978-3-642-14052-5_13

[3] Janusz A. Brzozowski. 1964. Derivatives of Regular Expressions. *J. ACM* 11, 4 (oct 1964), 481–494. https://doi.org/10.1145/321239.321249

[4] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. 2013. A Fully Verified Executable LTL Model Checker. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 463–478.

[5] Alexander Krauss and Tobias Nipkow. 2012. Proof Pearl: Regular Expression Equivalence and Relation Algebra. *J. Autom. Reason.* 49, 1 (jun 2012), 95–106. https://doi.org/10.1007/s10817-011-9223-4

[6] R. McNaughton and H. Yamada. 1960. Regular Expressions and State Graphs for Automata. *IRE Transactions on Electronic Computers* EC-9, 1 (1960), 39–47. https://doi.org/10.1109/TEC.1960.5221603

[7] J. J.M.M. Rutten. 1998. *Automata and Coinduction (an Exercise in Coalgebra)*. Technical Report. NLD.

[8] The Coq Development Team. 2022. *The Coq Proof Assistant*. https://doi.org/10.5281/zenodo.7313584

[9] Peter Thiemann and Martin Sulzmann. 2015. From ω-Regular Expressions to Büchi Automata via Partial Derivatives. In *Language and Automata Theory and Applications*, Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe (Eds.). Springer International Publishing, Cham, 287–298.