Crafting Provers in Racket

Adrian Manea
Coordinator: Assoc. Prof. Traian Şerbănuţă

March 13, 2020

Contents

TO ADD/CLARIFY

INTRODUCTION AND MOTIVATION

The main motivation for starting the work on this project is my interest in the programming language Racket. This grew from me getting acquainted with Emacs and Lisps, from a user standpoint at first, but then my interest grew and I started reading more about lambda calculus and various other related subjects. Before long, I discovered John McCarthy's pioneering work in trying to make lambda calculus suitable for programming languages, but what really captured my interest were the theoretical foundations that McCarthy's articles [McC60, McC61, McC62] set for this task, in a time when programming was either electrical engineering or pure mathematics. His work was said to have introduced a paradigm shift in computer science that is comparable to the non-Euclidean geometry revolution. The next step was discovering [AS96], which showed me how an apparently simple programming language such as Scheme can be used for wonderful constructions and various degrees of abstraction. This is further supported by the thorough specification and revisions that were published by the Scheme Community, the latest being [S+13].

Further reading in the world of Lisp dialects, I have discovered Racket, whose appeal was instantaneous to me, since it is so often described not as a general purpose programming language derived from Scheme,[1] but rather as a toolbox for constructing languages to solve various problems. In fact, as I see it, it offers the necessary items for one to properly understand, craft and teach languages that exhibit particular behaviour.

This is precisely the aim of the current work. Having a background in mathematics, I quickly became interested on the one hand in proof assistants (or so-called "theorem provers") and type theory, on the other. The appeal of category theory mixed with (or, by some sources, morphed into) topos theory, type theory and the background of intuitionistic logic is very strong for me and I have been trying to approach the subject from many of its sides. Of equal interest for me is the teaching aspect. I believe in strong foundations, a thorough understanding of fundamentals before getting into more complex structures and I so often search for methods, tools and examples that I can use to showcase particular aspects of the subject I'm learning or teaching. Racket, for

[1] Racket used to be called PLT Scheme, where PLT is the name of the research group that has been working on this project since the very beginning, scattered throughout the world, as showcased at [plt20].

me, provides the perfect environment to fulfil most of such intentions. On the one hand, the language is big and flexible enough to be expressive for concepts of type theory, logic and proofs and on the other, it can be used in a piecemeal setup to serve as a great teaching toolbox for my purposes.

The plan of the work is as follows. We will start with some basic elements of type theory and intuitionistic logic. While the subject is hugely developed and used in many parts of (theoretical) computer science, we will try to make the work as self-contained as possible and as such, include in the preliminaries strictly the topics that will be developed further. For simplicity, but also for historical reasons, we will be focusing on the main work by one of the creators of the field, at least in terms of making its presentation appropriate for computer science applications, Per Martin-Löf, [ML80]. While Martin-Löf's lectures focus more on the theoretical side of things, the excellent [NP90] provides enough details to see how the concepts can be used in actual programming.

The second preliminary part of the dissertation will focus on Racket itself. We will try to provide a so-called "crash-course" to introduce some elements of syntax. Again, we will be focusing on items that will be used throughout the dissertation.

A first, excellent example will follow, along the lines of P. Ragde's article, [Rag16]. There, the author showcases a toy proof assistant, called *Proust*, that they have written in Racket, the purpose being twofold. On the one hand, it shows the power that the language has for such tasks and on the other, as the author mentions, it helps them and their students to understand at least a part of the inner workings of well established proof assistants, such as Coq and Agda. That is, they will be implementing a small portion of the tools that are available in Coq and Agda in an intentionally verbose style so that their functions are transparent.

Finally, further examples are provided, which will exhibit features of interest from type theory and intuitionistic logic.

change bib style to go well with websites

BIBLIOGRAPHY

[AS96] Harold Abelson and Gerald Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1996.

[McC60] John McCarthy. Recursive functions of symbolic expressions and their computation by machine. *Commun. ACM*, 3(4):184–195, April 1960.

[McC61] John McCarthy. A basis for a mathematical theory of computation. *Western Joint Computer Conference*, 1961.

[McC62] John McCarthy. Towards a mathematical science of computation. *IFIP-62*, 1962.

[ML80] Per Martin-Löf. Intuitionistic type theory. Lectures in Padua, 1980. Notes by Giovanni Sambin.

[NP90] Bengt Nordström and Kent Petersson. *Programming in Martin-Löf Type Theory*. 1990. Freely available at http://www.cse.chalmers.se/research/group/logic/book/.

[plt20] The PLT group. https://racket-lang.org/people.html, 2020. Accessed: March 2020.

[Rag16] Prabhakar Ragde. Proust: A nano proof assistant. In Johan Jeuring and Jay McCarthy, editors, *Trends in Functional Programming in Education*, pages 63–75. arXiv/cs.PL, 2016.

[S+13] Gerald Sussman et al. Revised7 Report on the Algorithmic Language Scheme. Technical report, Scheme Working Group, 7 2013.