Andrés Ricardo Garza Vela

The Lonnrot Language

SIGNATURE

APRIL 16, 2021

Copyright © 2021

Andrés Ricardo Garza Vela

ALL RIGHTS RESERVED

A report written for the course of Compiler Design at ITESM

Contents

About the Lonnrot Language 5

1 Tetragrammaton 7

1.1 Basic Elements 7

1.2 Syntax 8

1.3 Syntax Diagrams 9

2 Red Scharlach 13

2.1 Operational Semantics 13

3 Rue de Toulon, or The Intermediate Code 15

4 Triste-le-Van-Roy, or The Target Language 17

Conclusion 19

Appendix 21

Bibliography 22

About the Lonnrot Language

The Lonnrot Language is intended to be an implementation of a compiler for a subset of Scheme extended with the core of miniKanren. As such, it will belong to the functional and relational paradigms of programming. This project is intended to be, above all things, *exploratory*. That implies several things, chief amongst which are that it is not the be-all end-all of implementations, nor is it a breakthrough or anything like that. It is quite simply a first approximation to compiler building (for me) and in that sense it synthesizes my learning from resources I've found and have yet to find along the way.

That being said, there *is* some sort of value proposition to this project. Mainly, most if not all implementations of miniKanren to date are interpreters or are DSL's for their respective host languages. This is understandable, especially when it comes to Lisps: extending the Lisp language via macros is a well documented tradition, and the resulting product is undeniably beautiful. Not only that, but given Lisp's syntax, it is indistinguishable from a DSL. However, there are things to be gained and interesting insights to be had: is it worth compiling miniKanren? Can we transfer our knowledge of methods used for other languages to this project? Can this shed light on how to optimize some of the workings that deal with the relational semantics?

For example, Will Byrd has talked about thread safety and the potential of parallelization in miniKanren [Byr15]

I do not claim to have answers to these questions, and frankly I know better than to expect to have them four months from now. What I do know is that it is an interesting premise for a first incursion in this domain (again, for me): **if not know, when would I start?** In the same vein, I hope that this report embodies the philosophy of growing a body of knowledge, of allowing oneself to err and to be playful. That is the reason why tenses will seem to be mixed up, because I'm writing this as I go. Speaking of which ...

This report will be divided into an arbitrary number of chapters, which cover the steps involved in writing the compiler (s). This

might seem weird at first, since a Preface is meant to outline the lay of the land. However, the construction of Lonnrot will be done following the ideas presented by Abdulaziz Ghuloum in *An Incremental Approach to Compiler Construction*. The main idea is to write multiple compilers, with each one covering a progressively larger language. This is opposed to the idea of writing one compiler, front to back, for the originally intended language. Due to this, it would be awkward to decide *a priori* how chapters would be laid out. Instead, each chapter will be a documentation of progress for a given time frame (about a week or so).

Lindsey Kuper [Kup19] talks about this approach and Kent Dybvig's "cousin" approach of writing the compiler back-to-front, making a language progressively more high-level, for those interested.

The details and implications of this approach are very interesting and I would do them a disservice by trying to explain them in a paragraph; I would encourage anyone to read up on them. For now it's enough to say that this project is a sort of mishmash of ideas on compiler construction: from Ghuloum's approach, to what makes the WAM be the WAM, passing through what makes miniKanren different from Prolog. In any case, that means that each chapter covers the progression of each part of the compiler. For example, Chapter **??** will cover my first deliverable, Chapter **??** covers the second deliverable and so on. It could be thought of as a sort of more fleshed-out log.

Lastly, the last part of this report comprises conclusions and some interesting miscellanea. The conclusion will naturally discuss the state of affairs by the time the course is over. The Appendix will have tidbits of information that might not fit the natural flow of the report. As of the time of this writing I'm not entirely sure what I want to include there, but stuff like the language's namesake will be explained there, for those who may be curious.

1

Tetragrammaton

The s-expression syntax dates back to 1960. This syntax is often controversial amongst programmers. Observe, however, something deeply valuable that it gives us. While parsing traditional languages can be very complex, parsing this syntax is virtually trivial.

Shriram Krishnamurthi [Fis20]

1.1 Basic Elements

⟨*datum*⟩ ::= ⟨*boolean*⟩
| ⟨*character*⟩
| ⟨*variable*⟩
| ⟨*string*⟩
| ⟨*number*⟩
| ⟨*list*⟩

⟨*variable*⟩ ::= ⟨*initial*⟩ ⟨*subsequent*⟩*

⟨*initial*⟩ ::= ⟨*letter*⟩ | '`!`' | '`$`' | '`&`' | '`*`' | '`/`' | '`:`' | '`<`' | '`=`' | '`>`' | '`?`' | '`~`' | '`_`' | '`^`'

⟨*letter*⟩ ::= a | ... | z | A | ... | Z

⟨*boolean*⟩ ::= `#t` | `#f`

⟨*number*⟩ ::= ⟨*integer*⟩ | ⟨*decimal*⟩

⟨*integer*⟩ ::= ⟨*digit*⟩+

⟨*decimal*⟩ ::= ⟨*integer*⟩ '`.`' ⟨*digit*⟩+

⟨*char*⟩ ::= '`#`' ⟨*any-character*⟩

⟨*list*⟩ ::= (⟨*datum*⟩*)

| (⟨*datum*⟩+ . ⟨*datum*⟩)

1.2 Syntax

⟨*program*⟩ ::= ⟨*form*⟩*

⟨*form*⟩ ::= ⟨*definition*⟩ | ⟨*expression*⟩

⟨*definition*⟩ ::= (`define` ⟨*variable*⟩ ⟨*expression*⟩)

⟨*expression*⟩ ::= ⟨*constant*⟩

| ⟨*variable*⟩

| (`quote` ⟨*datum*⟩)

| (`fresh` ⟨*formals*⟩ ⟨*expression*⟩ ⟨*expression*⟩*)

| (`==` ⟨*expression*⟩ ⟨*expression*⟩)

| (`run` ⟨*digit*⟩ ⟨*formals*⟩ ⟨*expression*⟩)

| (`lambda` ⟨*formals*⟩ ⟨*expression*⟩ ⟨*expression*⟩*)

| (`if` ⟨*expression*⟩ ⟨*expression*⟩ ⟨*expression*⟩)

| (`set!` ⟨*variable*⟩ ⟨*expression*⟩)

| ⟨*application*⟩

⟨*constant*⟩ ::= ⟨*boolean*⟩ | ⟨*number*⟩ | ⟨*char*⟩ | ⟨*string*⟩

⟨*formals*⟩ ::= ⟨*variable*⟩

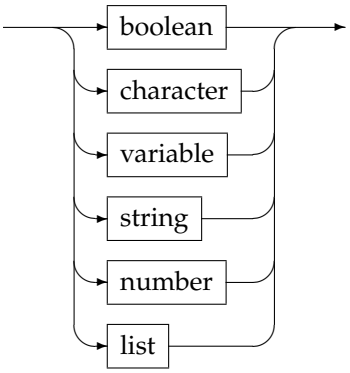
| (⟨*variable*⟩*)

| (⟨*variable*⟩ ⟨*variable*⟩* . ⟨*variable*⟩)

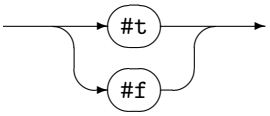
⟨*application*⟩ ::= (⟨*expression*⟩ ⟨*expression*⟩*)

1.3 Syntax Diagrams

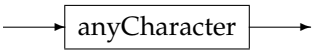
datum



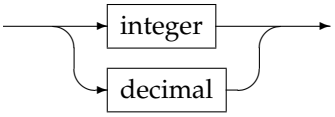
boolean



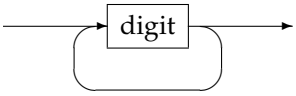
char



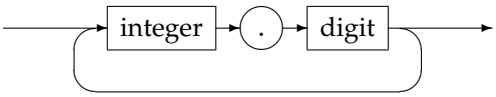
number



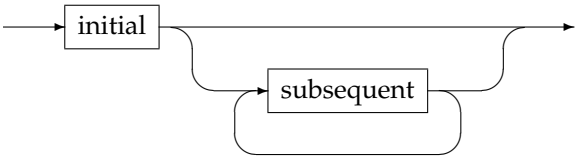
integer



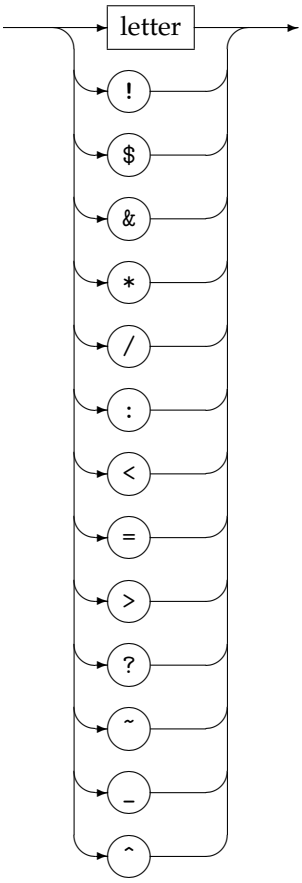
decimal



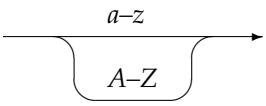
variable



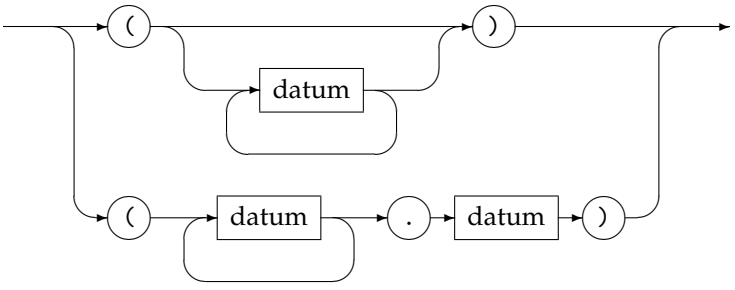
initial



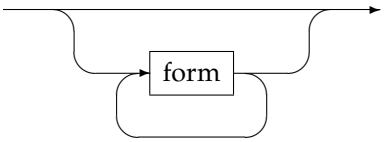
letter



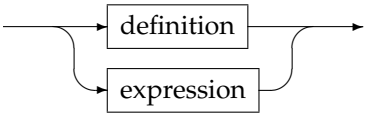
list



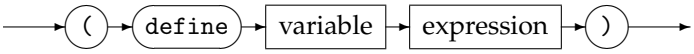
program



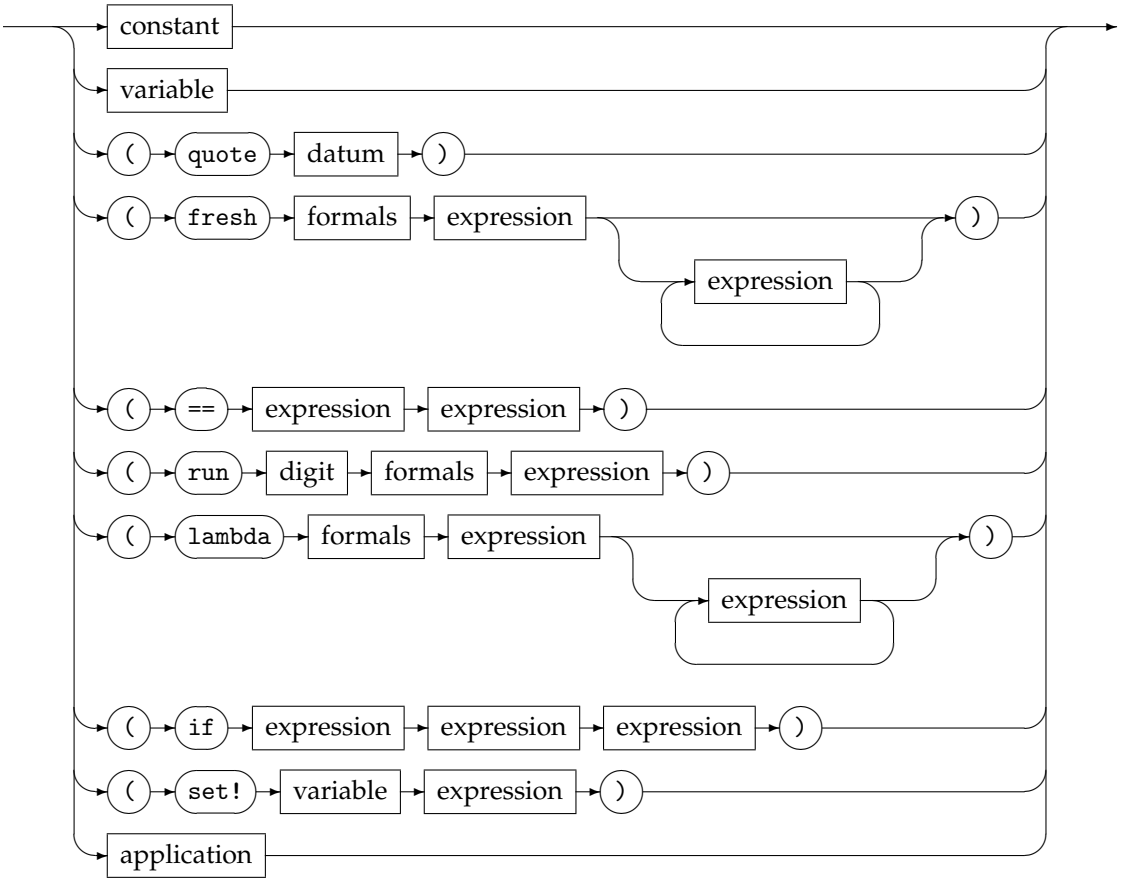
form



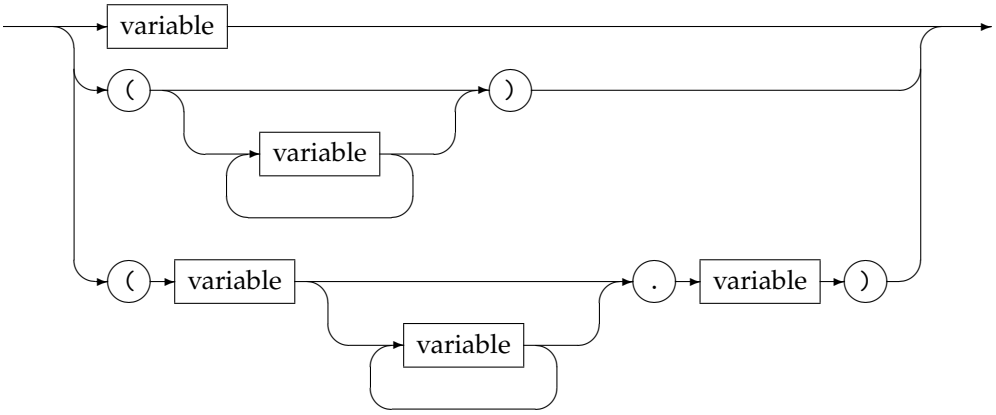
definition



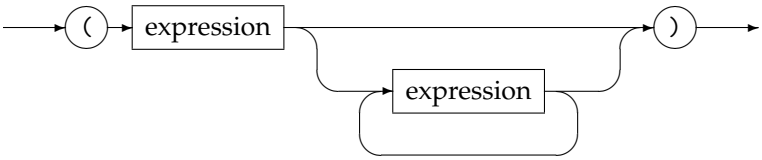
expression



formals



application



clearpage

2

Red Scharlach

2.1 Operational Semantics

3
Rue de Toulon, or The Intermediate Code

4
Triste-le-Van-Roy, or The Target Language

Conclusion

Appendix

Lonnrot's namesake

The are only two hard problems in Computer Science: cache invalidation and naming things

---

Popular Wisdom

Bibliography

[AK91] Hassan Aït-Kaci. *Warren's abstract machine. A tutorial reconstruction*. Logic Programming. The MIT Press, 1991.

[Byr15] William E. Byrd. Differences between minikanren and prolog. http://minikanren.org/minikanren-and-prolog.html, 2015. Accessed April 3, 2021.

[Fis20] Shriram Krishnamurthi; Benjamin S. Lerner; Joe Gibbs Politz; Kathi Fisler. Programming and programming languages. https://papl.cs.brown.edu/2019/, 2020. Accessed April 3, 2021.

[Hem18] Daniel P. Friedman; William E. Byrd; Oleg Kiselyov; Jason Hemann. *The Reasoned Schemer*. The MIT Press, 2nd edition, 2018.

[Kup19] Lindsey Kuper. My first fifteen compilers. http://minikanren.org/minikanren-and-prolog.html, 2019. Accessed April 3, 2021.