Relational Programming in miniKanren

Relational Programming in miniKanren

William E. Byrd

Typeset by the author in XƎLaTeX, using Tufte-Style Book from http://www.LaTeXTemplates.com.
June 27, 2013 version.

For my H211 students: Indiana University, Fall 2010 & 2011,

and Team pw0ni3.

Learning with always trumps learning from.

—Woodie Flowers

Contents

Preface

This is a book about *relational programming*. Just as functional programs model mathematical functions, relational programs model mathematical relations. Relational programming is intimately related to both logic programming and relational algebra, the theory behind relational databases.

If you know nothing about logic programming, relational algebra, or databases, *don't panic!* This book only assumes that you can write simple recursive programs in Scheme. See the *Audience* section below for details.

Relational programs generalize functional programs, in that they do not distinguish between the "input" arguments passed to a function and the "output" result returned by that function. For example, consider a two-argument variant of Scheme's addition function, restricted to natural numbers: $(+\ 3\ 4) \Rightarrow 7$. A relational version of addition, which we'll call $+^o$, takes three arguments: $(+^o\ 3\ 4\ z)$, where z is a *logic variable* that represents the result of adding the first two arguments of $+^o$. In this case z is associated with 7. More interestingly, we can write $(+^o\ 3\ y\ 7)$, which associates y with 4; our addition relation also performs subtraction. We can also write $(+^o\ x\ y\ 7)$, which associates x and y with all pairs of natural numbers that sum to 7; $+^o$ then produces multiple answers, including $x = 3$ and $y = 4$, and $x = 0$ and $y = 7$. Finally, we can write $(+^o\ x\ y\ z)$, which enumerates all triples of natural numbers (x, y, z) such that $x + y = z$; here our $+^o$ relation produces infinitely many answers. Informally, we say that the call $(+^o\ 3\ 4\ z)$ runs the $+^o$ relation "forwards," while the calls $(+^o\ 3\ y\ 7)$, $(+^o\ x\ y\ 7)$, and $(+^o\ x\ y\ z)$ run $+^o$ "backwards."

The *natural numbers* are the non-negative integers: 0, 1, . . .

Here we are taking a notational liberty, as $+^o$ expects 3 and 4 to be represented as *binary*, *little-endian* lists: (1 1) and (0 0 1), respectively. Zero is uniquely represented as the empty list, (). To ensure a unique representation of each number, lists may not end with the digit 0. This numeric representation is extremely flexible, since the lists can contain logic variables—for example, the list '(1 . ,x) represents any odd natural number, while '(0 . ,x) represents any positive even natural.

We can also perform relational arithmetic on built-in Scheme numbers, using *Constraint Logic Programming over Finite Domains*, or *CLP(FD)*; CLP(FD) is faster, but less general, than $+^o$ and friends ($*^o$, $/^o$, etc.).

This book will teach you how to write relations that produce interesting answers when running both forwards and backwards. For example, you will learn to write a relational interpreter for a subset of Scheme: ($eval^o$ '((λ (x) x) 5) val) associates val with 5. Of course, we can play more tricks with $eval^o$: ($eval^o$ exp '6) generates legal Scheme expressions that *evaluate* to 6, while ($eval^o$ exp exp) generates Scheme expressions that evaluate to themselves.

Douglas Hofstadter coined the term *quine* to describe a program that evaluates to itself, in honor of logician Willard Van Orman Quine (1908–2000). Writing quines has long been a favorite hacker activity, and quines are often featured in the The International Obfuscated C Code Contest (`http://www.ioccc.org/`). A delightful introduction to quines can be found in Doug's classic, *GEB*:

D. R. Hofstadter. *Gödel, Escher, Bach : an Eternal Golden Braid*. Basic, 1979

Audience

This book is written for intermediate-to-advanced programmers, computer science students, and researchers. For this book, *intermediate* means that you are comfortable writing simple recursive procedures in a functional programming language, such as Scheme, Racket, Clojure, Lisp, ML, or Haskell. I also assume you have a reading knowledge of Scheme. No knowledge of relational programming, logic programming, or programming language theory is required.

I have attempted to deliver [these lectures] in a spirit that should be recommended to all students embarking on the writing of their PhD theses: imagine that you are explaining your ideas to your former smart, but ignorant, self, at the beginning of your studies!

—Richard P. Feynman
The Feynman Lectures on Computation

If you want to learn about relational programming, but are new to programming, Dan Friedman, Oleg Kiselyov, and I have written a book just for you, called *The Reasoned Schemer*[1]. In that book we assume you are familiar with the material in *The Little Schemer*[2], which is a very gentle introduction to recursion and functional programming.

If you are an experienced programmer, but weak on recursion, you, too, might benefit from *The Little Schemer*. If you are comfortable with recursion, but not functional programming, good introductions include *Scheme and the Art of Programming*[3] and the classic *Structure and Interpretation of Computer Programs*[4].

If you are an experienced functional programmer, but do not know Scheme, the beginning of *Structure and Interpretation of Computer Programs* should get you up to speed, while *The Scheme Programming Language, 4th Edition*[5] describes the language in detail.

[1] D. P. Friedman, W. E. Byrd, and O. Kiselyov. *The Reasoned Schemer*. MIT Press, Cambridge, MA, 2005

[2] D. P. Friedman and M. Felleisen. *The Little Schemer (4th ed.)*. MIT Press, Cambridge, MA, 1996

[3] G. Springer and D. P. Friedman. *Scheme and the Art of Programming*. MIT Press, Cambridge, MA, 1989

[4] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, 2nd edition, 1996
(full text at http://mitpress.mit.edu/sicp/full-text/book/book.html)

[5] R. K. Dybvig. *The Scheme Programming Language, 4th Edition*. The MIT Press, 4th edition, 2009
(full text at http://www.scheme.com/tspl4/)

The Language

Goals

The two high-level goals for this book are to **make relational programming accessible** to a broader audience, and to **describe the state-of-the-art** in the design, implementation, and use of the miniKanren language. An important side-effect of these goals is to **provide the background** needed to understand academic papers and talks on miniKanren.

To some extent, this book can be thought of as an updated, greatly expanded, and much more accessible version of my PhD dissertation (full text at http://gradworks.umi.com/3380156.pdf):
W. E. Byrd. *Relational Programming in miniKanren: Techniques, Applications, and Implementations*. PhD thesis, Indiana University, 2009

One sub-goal of this book is to **present a variety of non-trivial miniKanren relations**, written in what I consider to be **idiomatic style**, and to show **how these relations were derived**. Many of these examples draw from academic papers on miniKanren, and focus on programming language theory (interpreters, type inferencers, etc.). Other examples, such as finite state machines, should be immediately understandable by a wider audience of programmers. To make the book as accessible as possible, all of these concepts are explained either in the main text or in appendices.

Margin Notes

This book is typeset in the style of Edward Tufte's magnificent and beautiful *The Visual Display of Quantitative Information*[6]. I share Tufte's love of margin notes, and use them in this book to help solve the problem of addressing readers with widely varying knowledge of computer science and programming. To make the book accessible as possible, in the main text I assume the reader is the hypothetical *intermediate-level* programmer or student described in the *Audience* section above. In the margin notes, however, anything goes.

This book is set using the "Tufte-Style Book" LaTeX style, freely available from `http://www.LaTeXTemplates.com`

[6] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, 1986

Another great lover of marginalia was David Foster Wallace (1962–2008). The *Harry Ransom Center*'s DFW collection includes heavily annotated books from Wallace's personal library: `http://www.hrc.utexas.edu/press/releases/2010/dfw/books/`. Wallace's love of margin notes is best demonstrated by his essay, "Host," in:

D. F. Wallace. *Consider the Lobster and Other Essays*. Little, Brown and Co., 2005

Typographic Conventions

Acknowledgements

William E. Byrd
Salt Lake City, Utah
June 2013

Introduction

gl hf!

(Traditional greeting in the Koprulu Sector)

Conclusion

G.G.

—Sean “Day[9]” Plott

Bibliography

H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, 2nd edition, 1996.

W. E. Byrd. *Relational Programming in miniKanren: Techniques, Applications, and Implementations*. PhD thesis, Indiana University, 2009.

R. K. Dybvig. *The Scheme Programming Language, 4th Edition*. The MIT Press, 4th edition, 2009.

D. P. Friedman and M. Felleisen. *The Little Schemer (4th ed.)*. MIT Press, Cambridge, MA, 1996.

D. P. Friedman, W. E. Byrd, and O. Kiselyov. *The Reasoned Schemer*. MIT Press, Cambridge, MA, 2005.

D. R. Hofstadter. *Gödel, Escher, Bach : an Eternal Golden Braid*. Basic, 1979.

G. Springer and D. P. Friedman. *Scheme and the Art of Programming*. MIT Press, Cambridge, MA, 1989.

E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, 1986.

D. F. Wallace. *Consider the Lobster and Other Essays*. Little, Brown and Co., 2005.