Relational Programming in miniKanren

Relational Programming in miniKanren

William E. Byrd

Typeset by the author in XeLaTeX, using Tufte-Style Book from http://www.LaTeXTemplates.com.
June 25, 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.

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: (+ 3 4) ⇒ 7. A relational version of addition, which we'll call $+^o$, takes three arguments: ($+^o$ 3 4 z). Here 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 $+^o$ relation performs subtraction as well as addition. We can also write ($+^o$ x y 7), which associates x and y with all pairs of natural numbers that sum to 7; here our $+^o$ relation produces multiple answers, such as $x = 3$ 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 +o relation can "run backwards," in contrast to Scheme's + function, which only can be used in the "forwards" direction.

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.

Here we are taking a notational liberty: $+^o$ actually expects 3 and 4 to be represented as binary, little-endian lists: (1 1) and (0 0 1), respectively. Zero is represented as the empty list, (). This numeric representation is extremely flexible, since the lists can contain logic variables—for example, the list '(1 . ,x) represents all odd natural numbers, while '(0 . ,x) represents all positive even naturals. 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.).

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/`). Much more on quines can be found in Doug's delightful book, *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.

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.

> 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*

[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/`)

Goals

The high-level goals for this book are to make relational programming accessible to a broader audience, and to explain the state-of-the-art in the design, implementation, and use of the miniKanren language.

Specifically, this book aims to:

1. describe the current state of the miniKanren language, and its Constraint Logic Programming (CLP) extensions;
2. present a variety of relational programs that exemplify the relational style, and that introduce important relational idioms;
3. explain how to derive miniKanren relations from pure Scheme functions in a systematic manner;
4. explain in detail the canonical implementation of core miniKanren, and show how to hack the implementation to change or extend the language;
5. explain how constraint solving works in the cKanren Constraint Logic Programming framework, and show how to design and implement new constraint domains;

6. demonstrate how to debug miniKanren relations;

7. explain how miniKanren differs from related languages, including Prolog, Curry, Mercury, and Oz;

8. describe important open problems in relational programming, and limitations of miniKanren;

9. explain the rationale behind key design decisions for miniKanren;

10. use relational programming to explain important and interesting concepts from programming languages, such as interpreters, type inferencers, and Continuation-Passing Style.

An important side-effect of these goals is to provide the background needed to understand academic papers and talks on miniKanren.

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.

William E. Byrd
Salt Lake City, Utah
June 2013

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

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.

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.