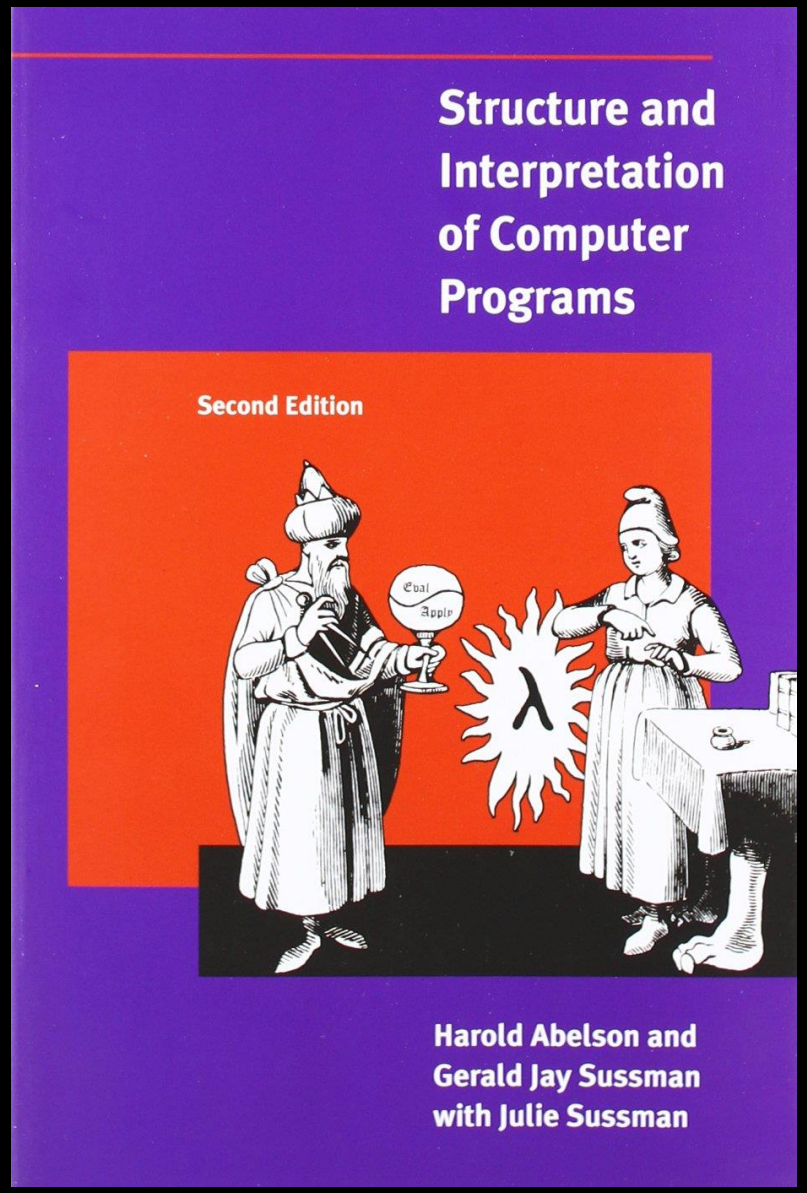


Structure and Interpretation of Computer Programs: SICP

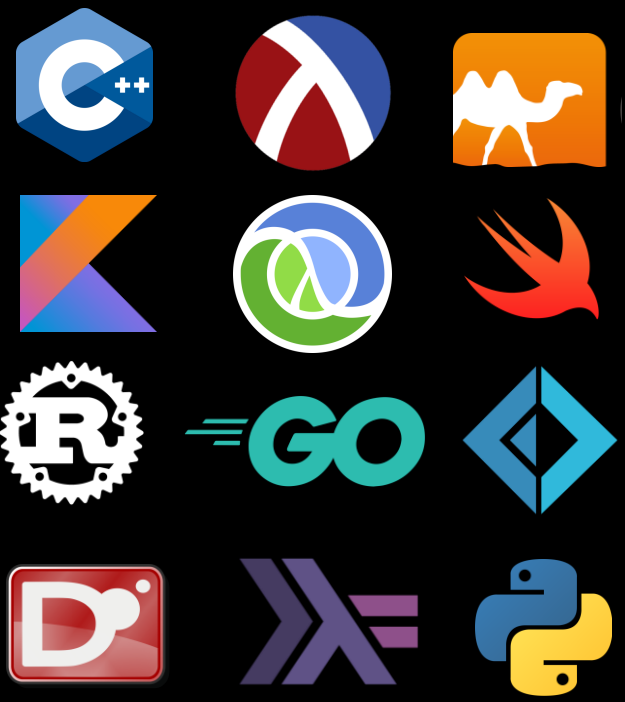
Conor Hoekstra code_report

github.com/codereport/Talks

2020-09-CppCon/SICP

About Me

- I'm a Sr Library Software Engineer for 
  - Working on the RAPIDS AI team (http://rapids.ai)
- I am a programming language enthusiast
- I love algorithms and beautiful code
- Organizer of Programming Languages Virtual Meetup
- I have a YouTube channel



code_report

21.1K subscribers

SUBSCRIBE

HOME VIDEOS PLAYLISTS COMMUNITY CHANNELS ABOUT

Uploads PLAY ALL

SORT BY

Structure and Interpretation of Computer Programs - ...

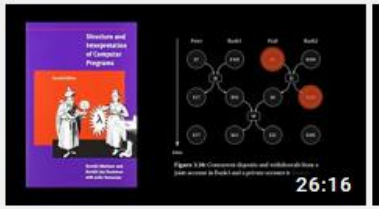
116 views • 23 hours ago

Structure and Interpretation of Computer Programs - ...

215 views • 1 week ago

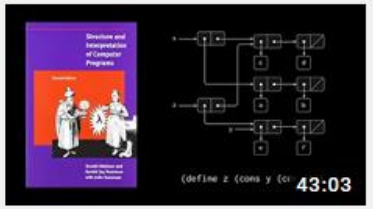
Structure and Interpretation of Computer Programs - ...

314 views • 2 weeks ago



Structure and Interpretation of Computer Programs - ...

232 views • 2 weeks ago

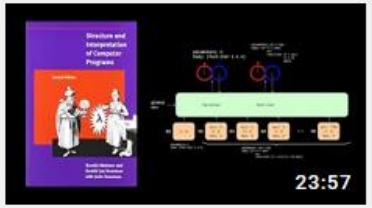


Structure and Interpretation of Computer Programs - ...

292 views • 4 weeks ago

Structure and Interpretation of Computer Programs - ...

248 views • 4 weeks ago



Structure and Interpretation of Computer Programs - ...

347 views • 1 month ago

Structure and Interpretation of Computer Programs - ...

506 views • 1 month ago

Structure and Interpretation of Computer Programs - ...

265 views • 1 month ago

Structure and Interpretation of Computer Programs - ...

327 views • 1 month ago

Structure and Interpretation of Computer Programs - ...

537 views • 2 months ago

Structure and Interpretation of Computer Programs - ...

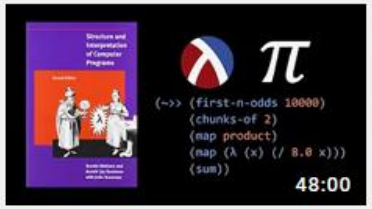
398 views • 2 months ago

Structure and Interpretation of Computer Programs - ...

384 views • 2 months ago

Structure and Interpretation of Computer Programs - ...

602 views • 3 months ago



Structure and Interpretation of Computer Programs - ...

705 views • 3 months ago

Structure and Interpretation of Computer Programs - ...

1.8K views • 3 months ago

Clojure!

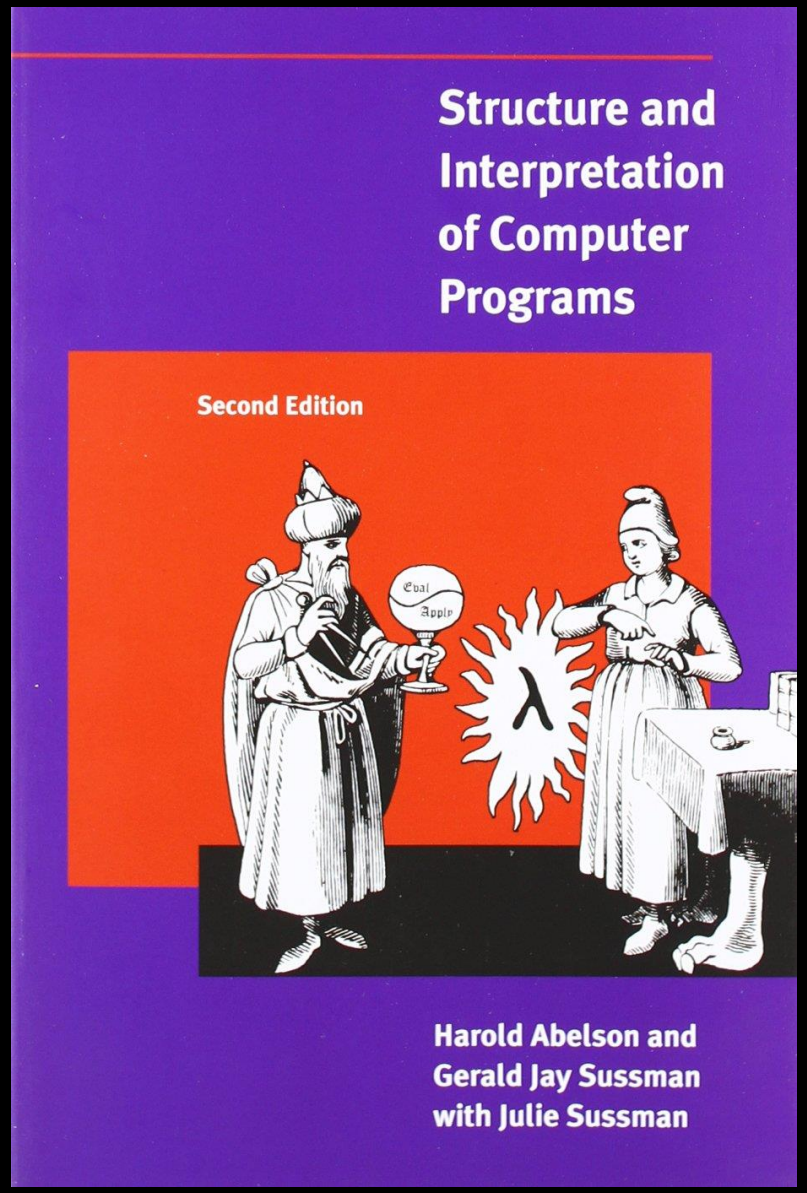
1.5K views • 3 months ago

Structure and Interpretation of Computer Programs - ...

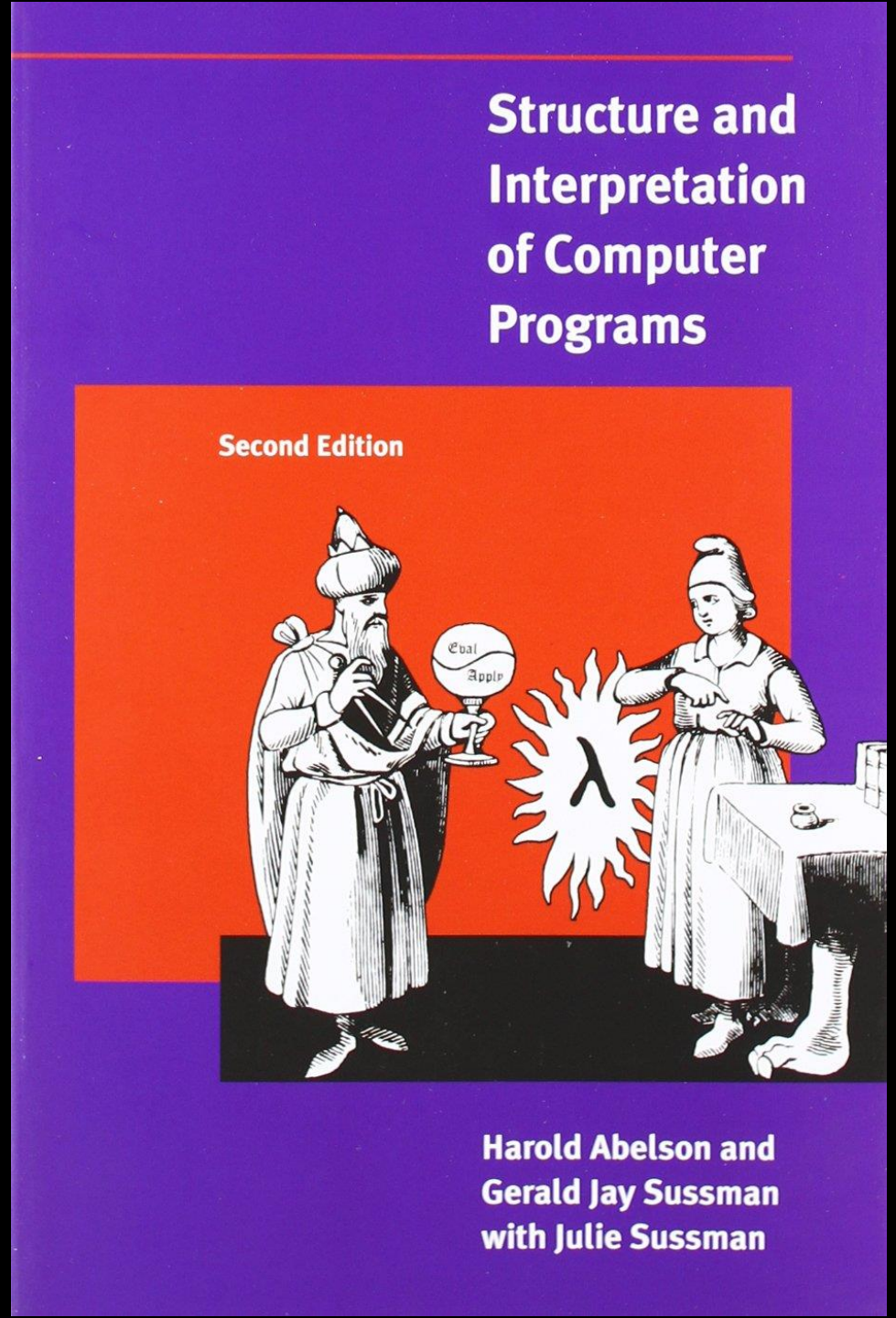
3.3K views • 3 months ago

Agenda

1. Reviews
2. Highlights
3. Problems (in )
4. Conclusion


Structure and Interpretation of Computer Programs
Second Edition
Eval
Apply
λ
Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

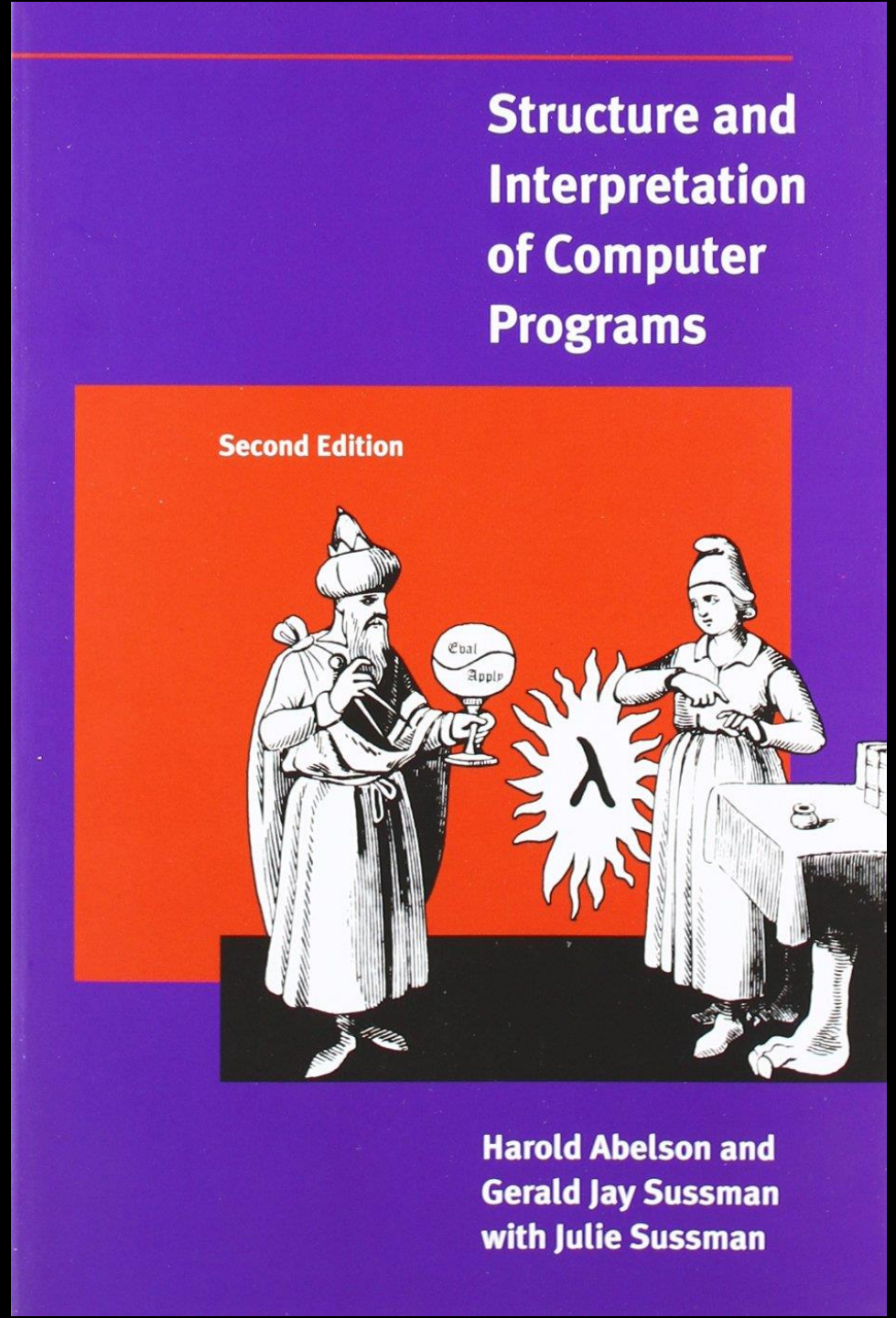
Reviews



"the best computer science book in the world"

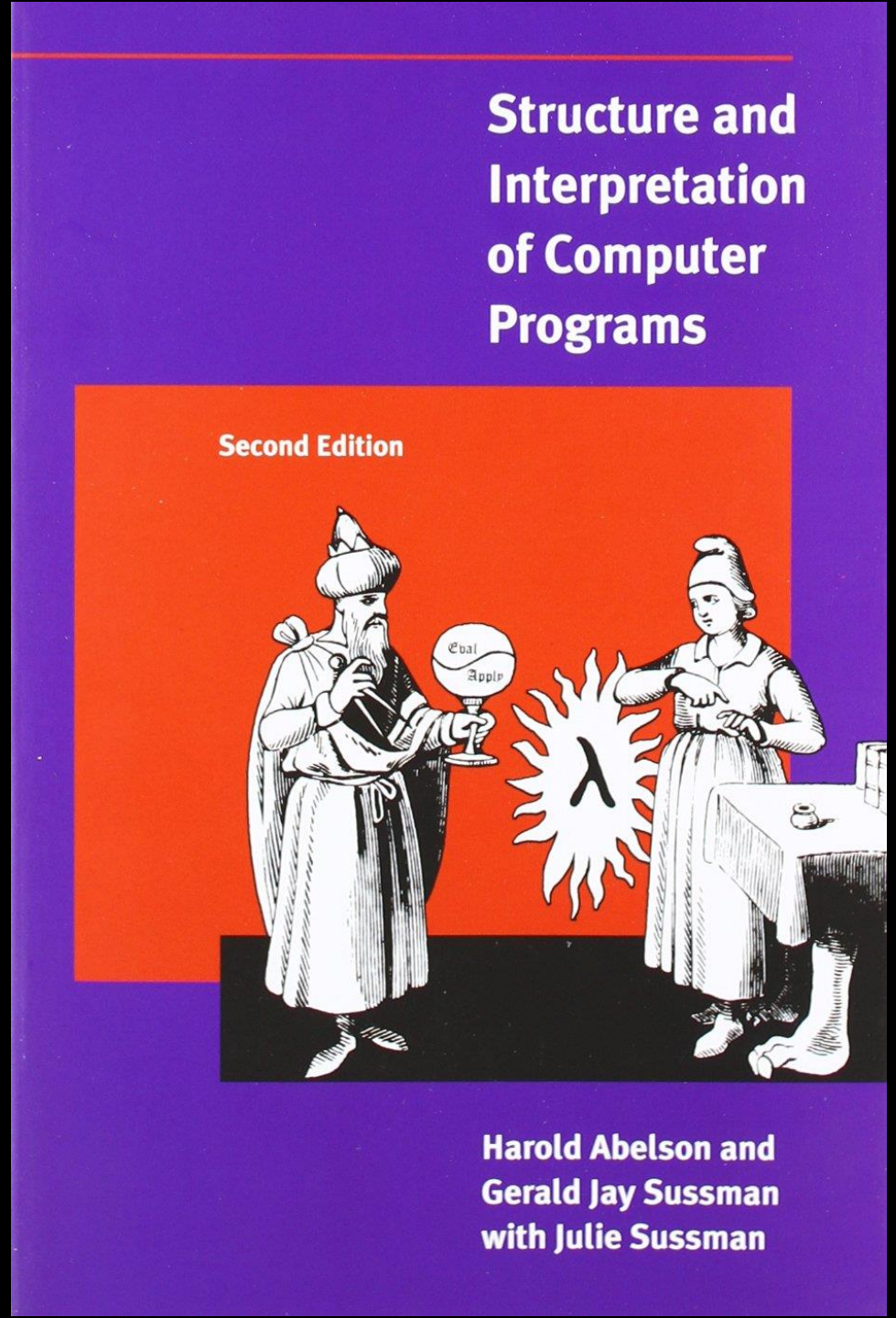
Brian Harvey

UC Berkeley Professor of 61A for 25+ Years



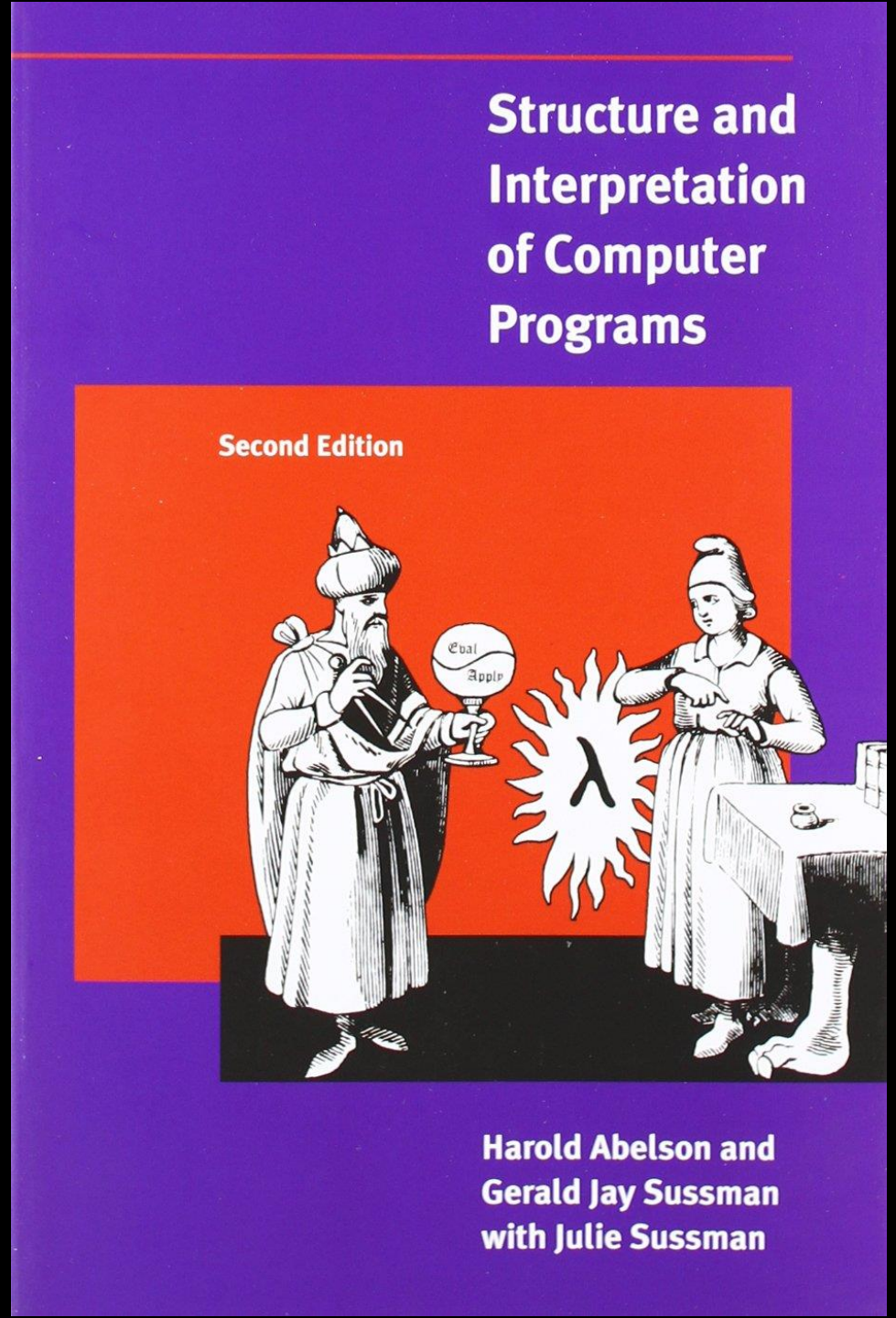
"probably the best introduction to computer science"

Peter Norvig
Author of
Artificial Intelligence:
A Modern Approach



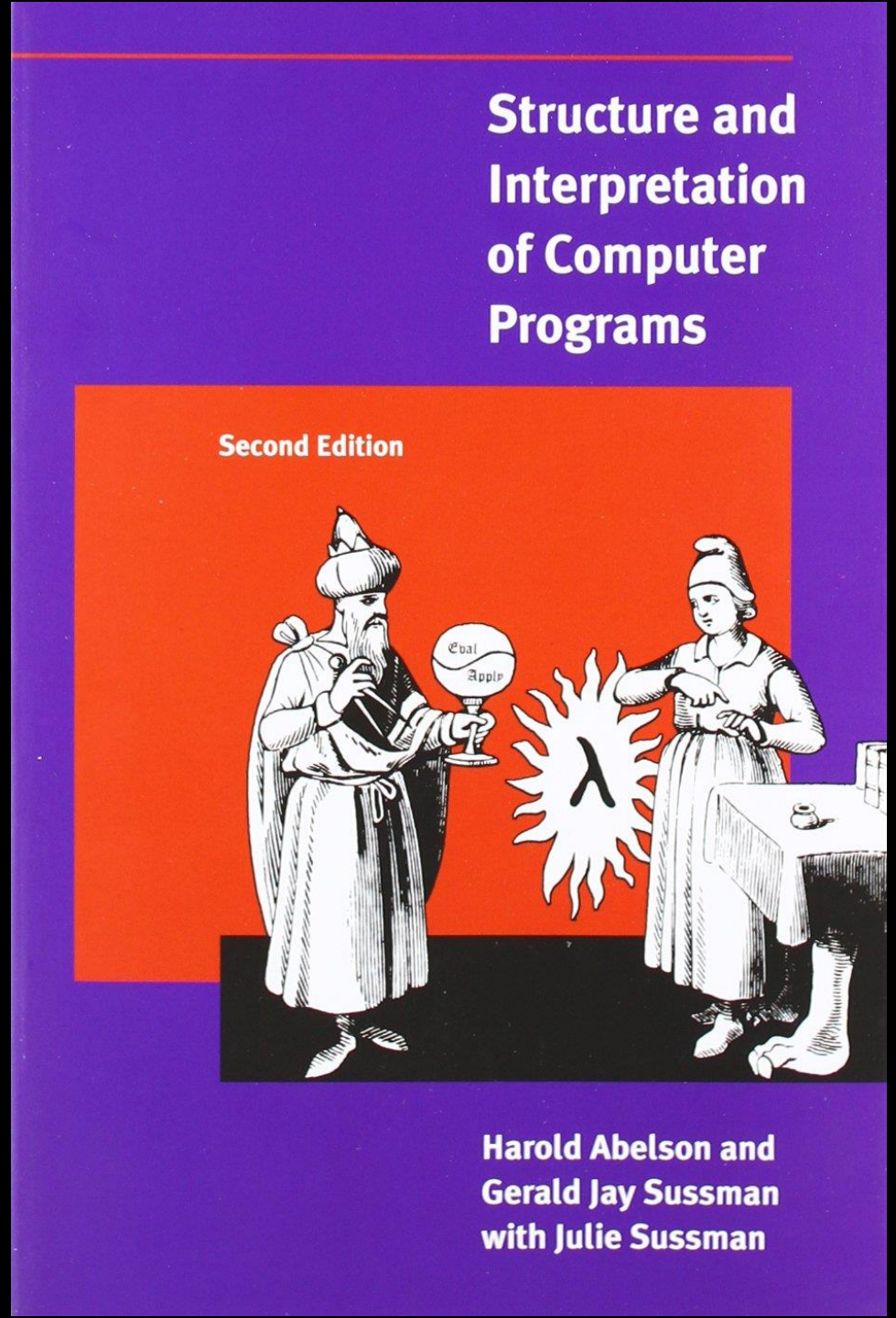
“one of the great classics of computer science”

Paul Graham
Founder of HackerNews and Co-Founder of Y-Combinator



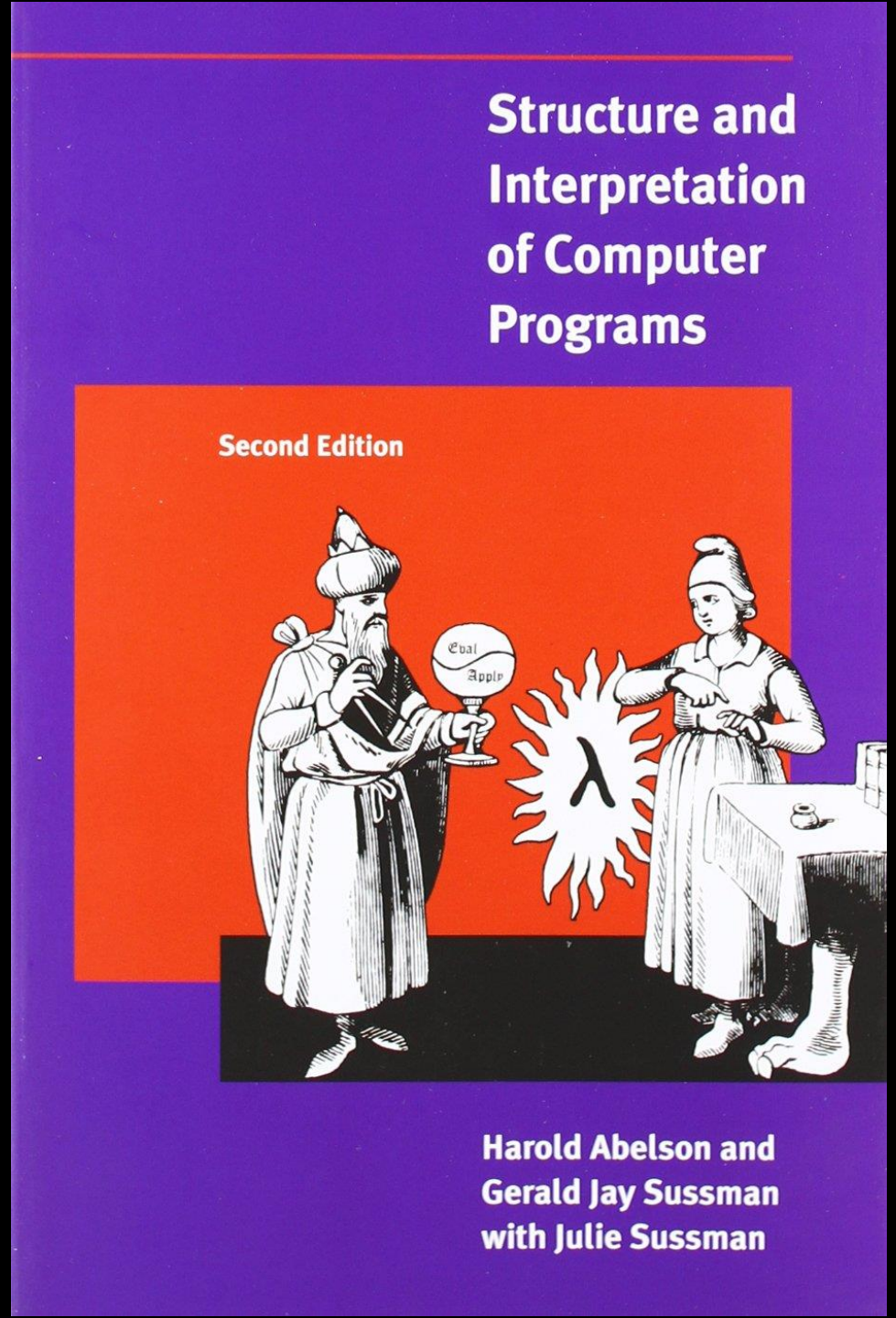
“simply the best programming book I’ve ever read”

Adam Tornhill
Founder of Empear,
Author & (LISP)
Programmer



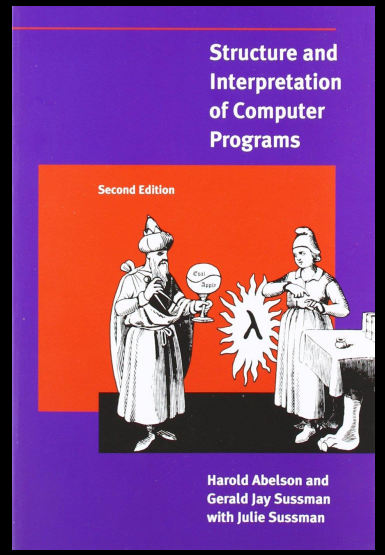
“changed the way I approach programming”

Adam Tornhill
Founder of Empear, Author & (LISP) Programmer



“amazing … startling … it has become a kind of bible for me”

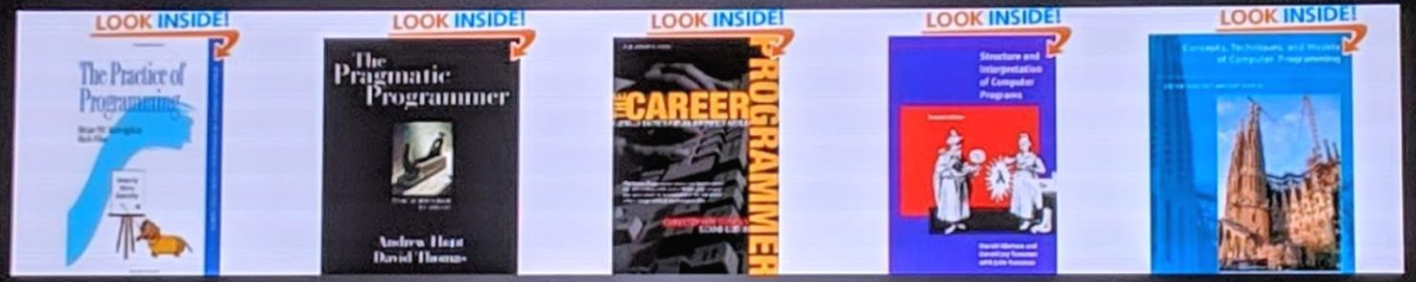
First Episode of Functional Geekery



“amazing … startling … it has become a kind of bible for me”

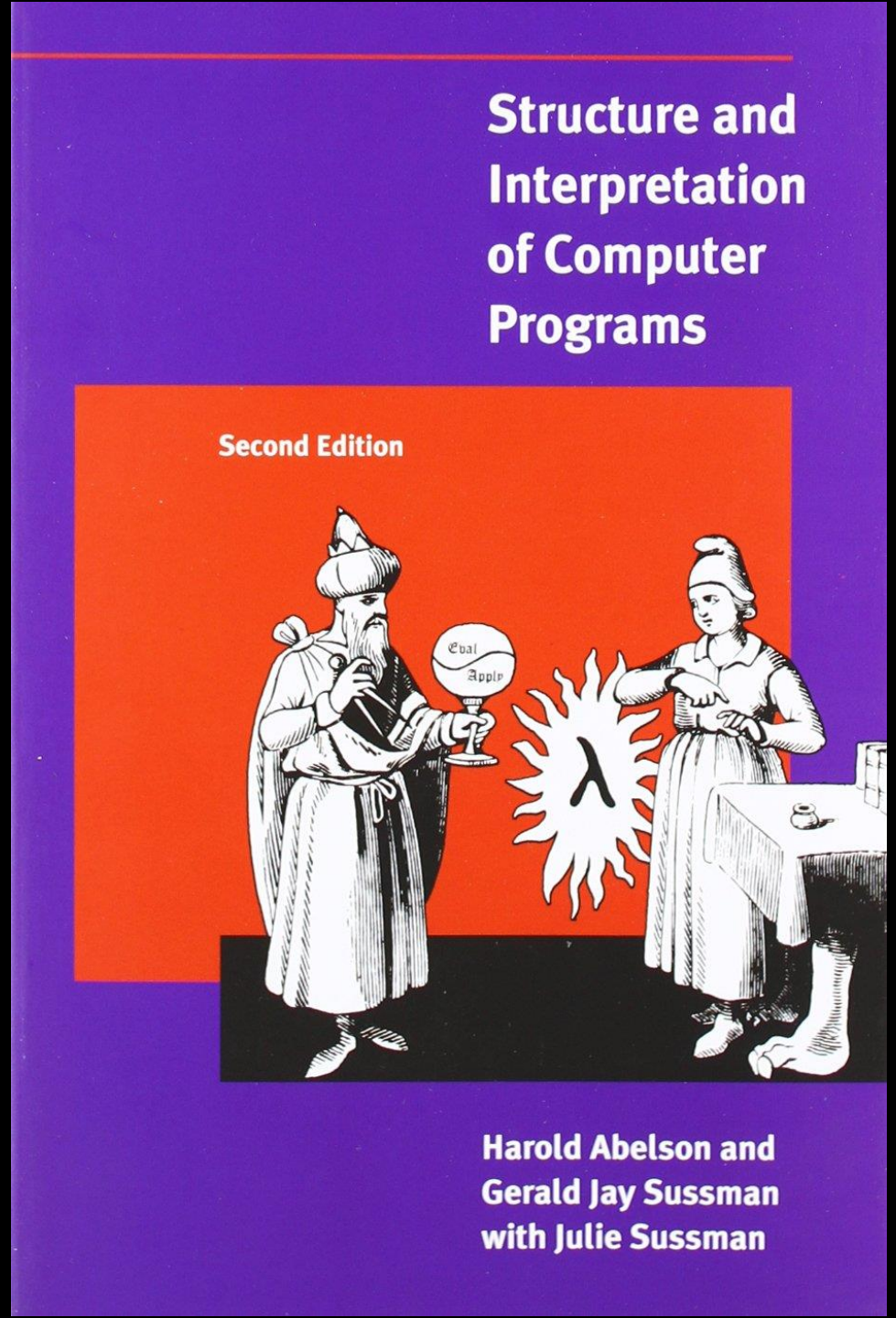
First Episode of Functional Geekery

Useful Sources of Input



- The Practice of Programming
- The Pragmatic Programmer
- The Career Programmer: Guerilla Tactics for an Imperfect World
- Structure and Interpretation of Computer Programs
- Concepts, Techniques, and Models of Computer Programming

python™



Why SICP Matters?

Brian Harvey
UC Berkeley Professor
of 61A for 25+ Years

Why SICP Matters?

SICP was revolutionary in many different ways…
SICP is about standing back from the details to learn big-picture ways to think about the programming process. It focused attention on the central idea of abstraction… the idea of functions as data… [and] three different programming paradigms: functional, object oriented, and declarative.

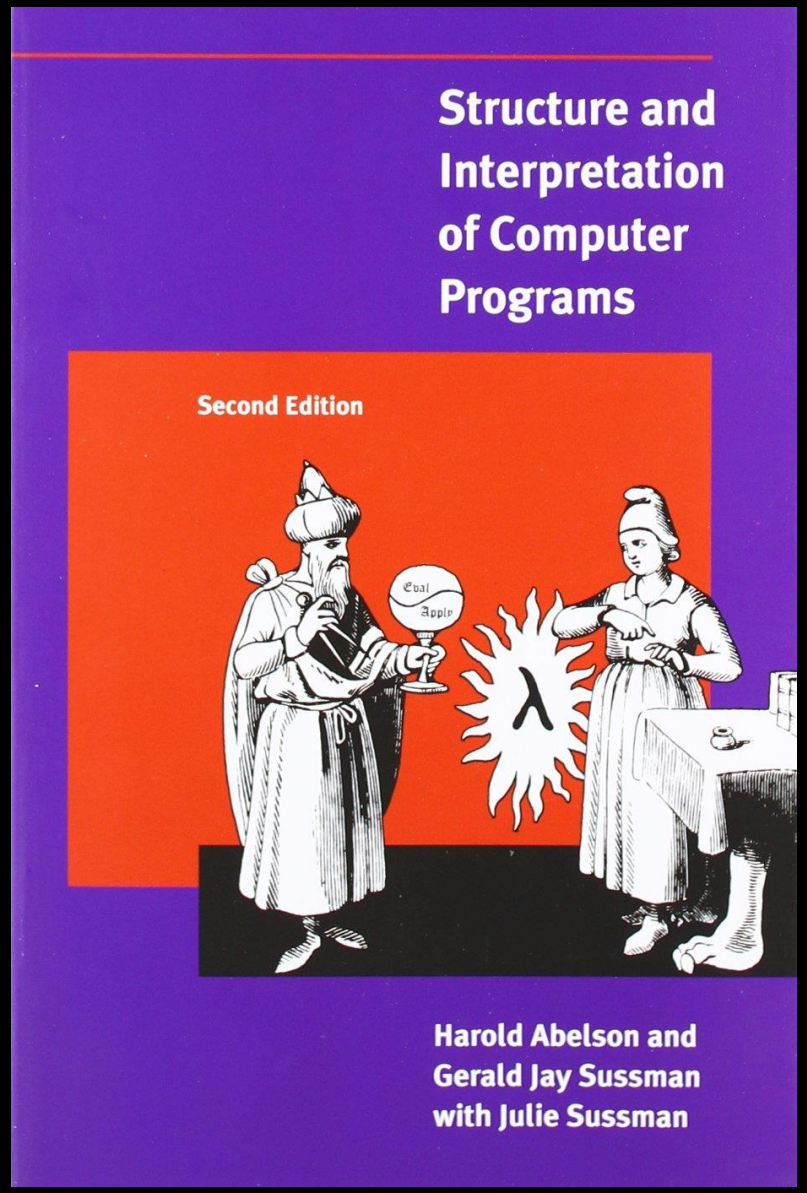
https://people.eecs.berkeley.edu/~bh/sicp.html

Programming Paradigms for Dummies: What Every Programmer Should Know

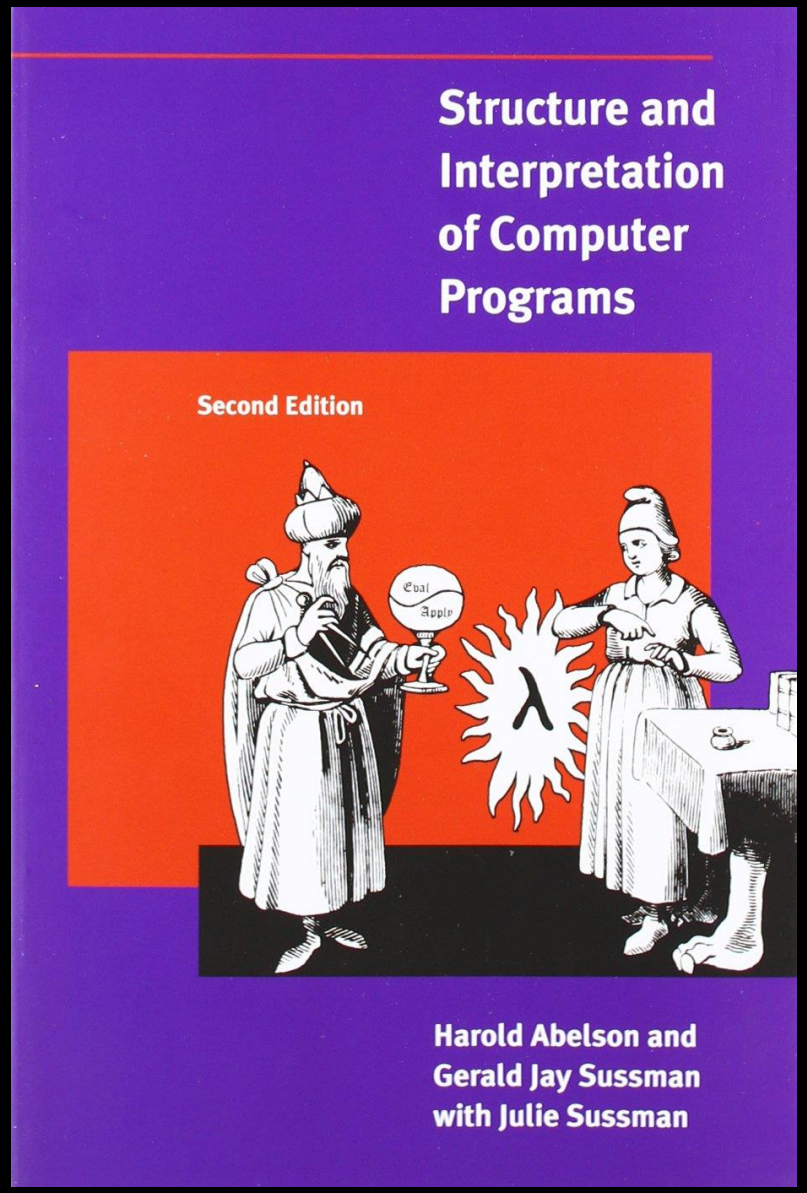
Peter Van Roy

This chapter gives an introduction to all the main programming paradigms, their underlying concepts, and the relationships between them. We give a broad view to help programmers choose the right concepts they need to solve the problems at hand. We give a taxonomy of almost 30 useful programming paradigms and how they are related. Most of them differ only in one or a few concepts, but this can make a world of difference in programming. We explain briefly how programming paradigms influence language design, and we show two sweet spots: dual-paradigm languages and a definitive language. We introduce the main concepts of programming languages: records, closures, independence (concurrency), and named state. We explain the main principles of data abstraction and how it lets us organize large programs. Finally, we conclude by focusing on concurrency, which is widely considered the hardest concept to program with. We present four little-known but important paradigms that greatly simplify concurrent programming with respect to mainstream languages: declarative concurrency (both eager and lazy), functional reactive programming, discrete synchronous programming, and constraint programming. These paradigms have no race conditions and can be used in cases where no other paradigm works. We explain why for multi-core processors and we give several examples from computer music, which often uses these paradigms.

More is not better (or worse) than less, just different.
– The paradigm paradox.


Structure and Interpretation of Computer Programs
Second Edition
Eval
Apply
λ
Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

Highlights


Structure and Interpretation of Computer Programs
Second Edition
Eval
Apply
λ
Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

Quotes

“I think that it’s extraordinarily important that we in computer science keep fun in computing.”

Alan J. Perlis

Dedication, SICP

“A programmer should acquire good algorithms and idioms.”

Alan J. Perlis

Forward, SICP

“The discretionary exportable functionality entrusted to the individual Lisp programmer is more than an order of magnitude greater than that to be found within Pascal enterprises.”

Alan J. Perlis

Forward, SICP

"Pascal is for building pyramids – imposing, breathtaking, static structures built by armies pushing heavy blocks into place. Lisp is for building organisms - imposing, breathtaking, dynamic structures built by squads fitting fluctuating myriads of simpler organisms into place."

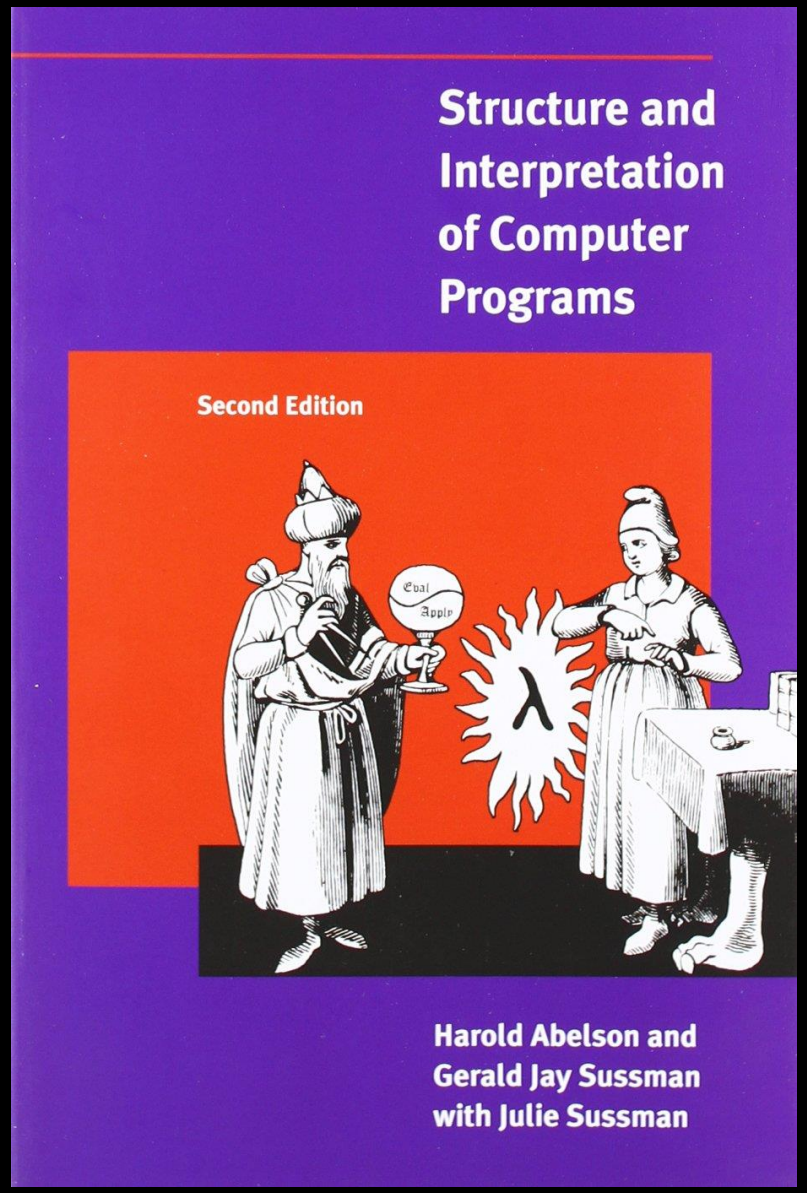
Alan J. Perlis

Forward, SICP

“Scheme, the dialect of Lisp that we use, is an attempt to bring together the power and elegance of Lisp and Algol.”

Preface to the First Edition, SICP

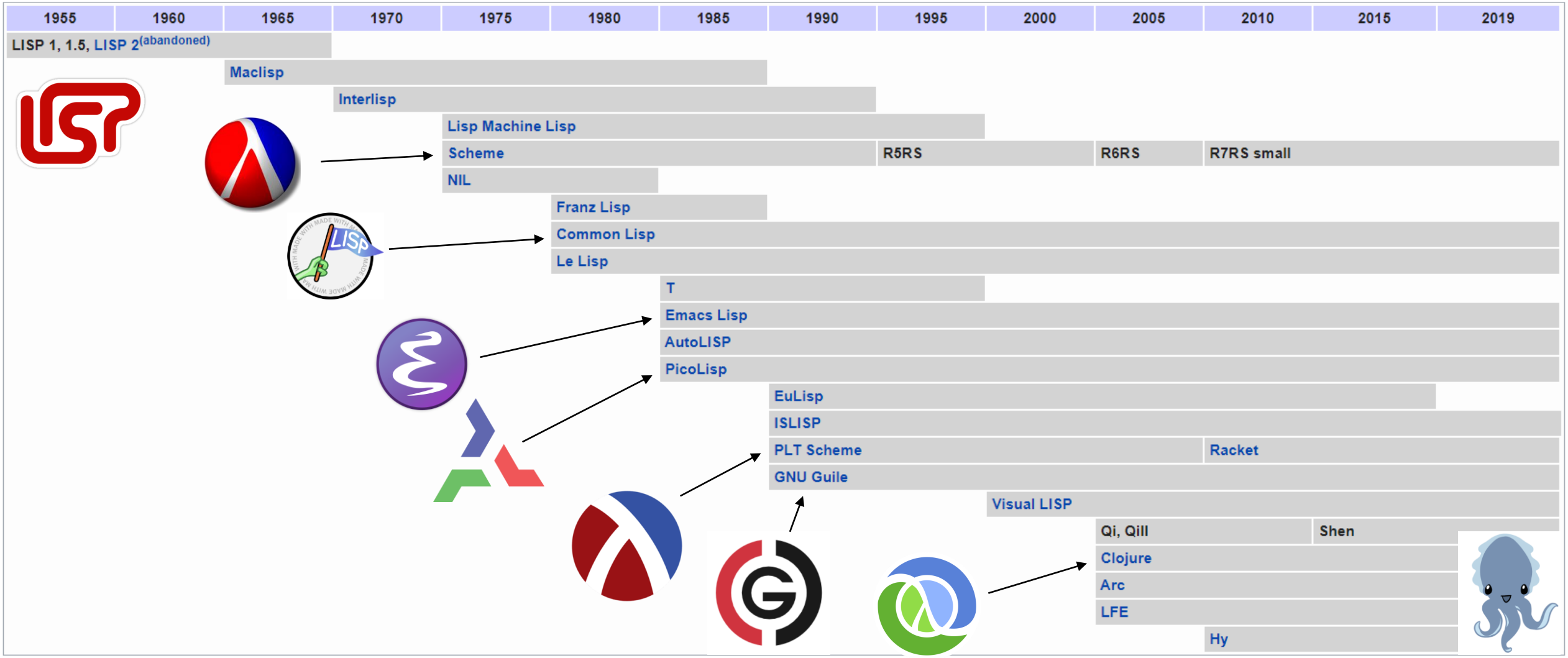
Highlights



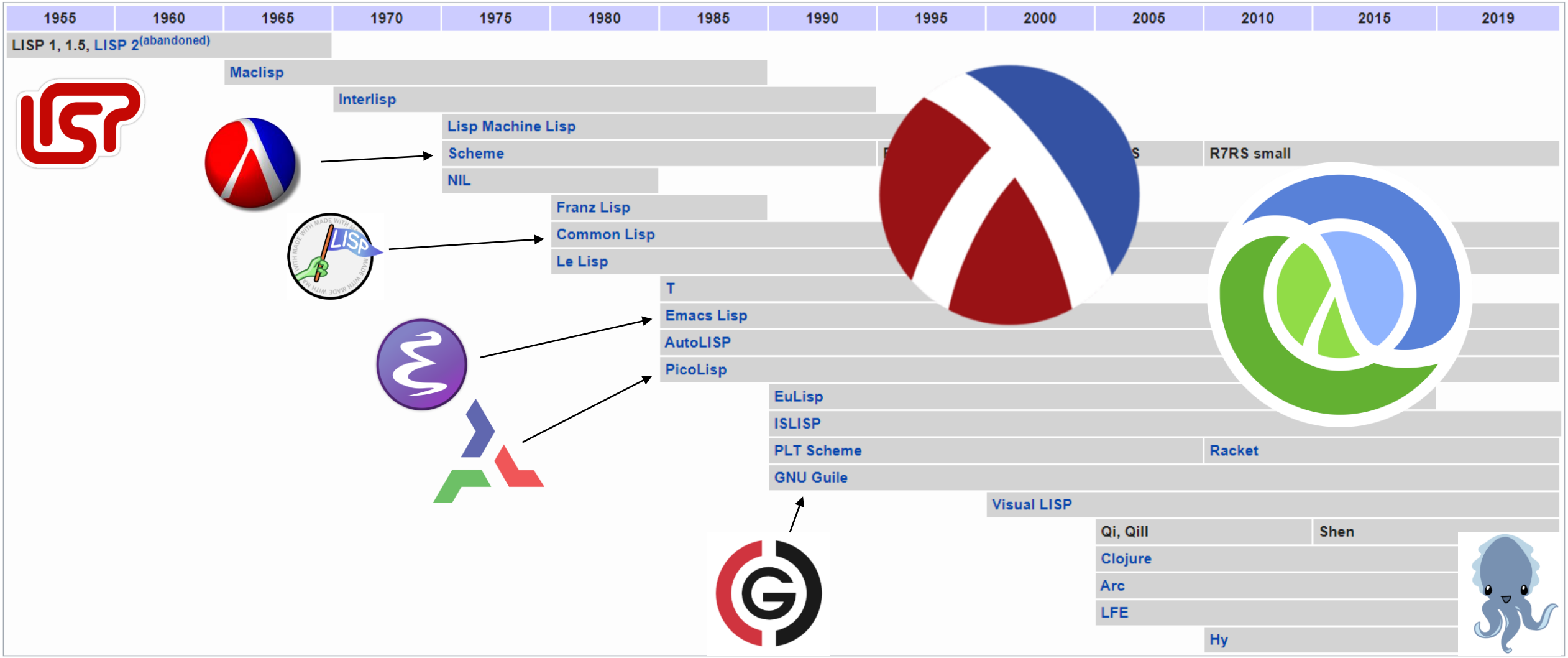
Mini

History

Timeline of Lisp dialects[edit]



Timeline of Lisp dialects[edit]



Late 50s

Mid 70s

90s

McCarthy

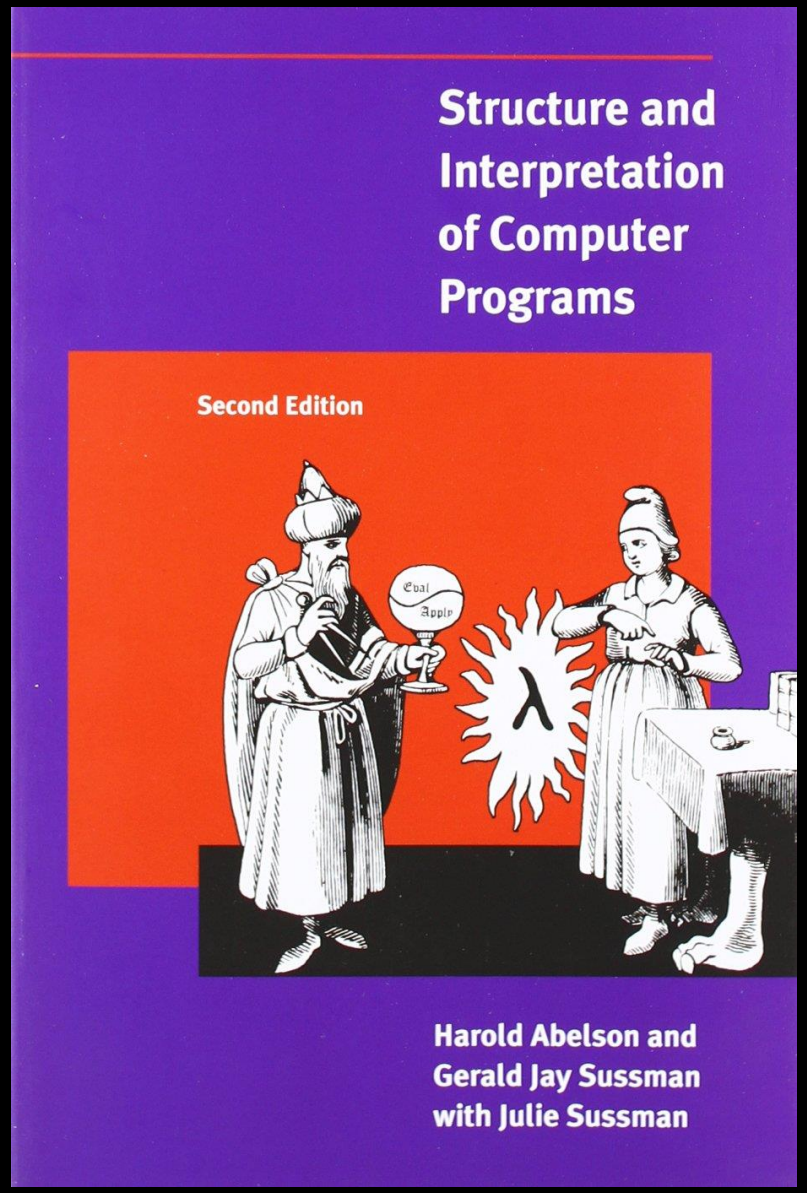
Scheme

Sussman & Steele

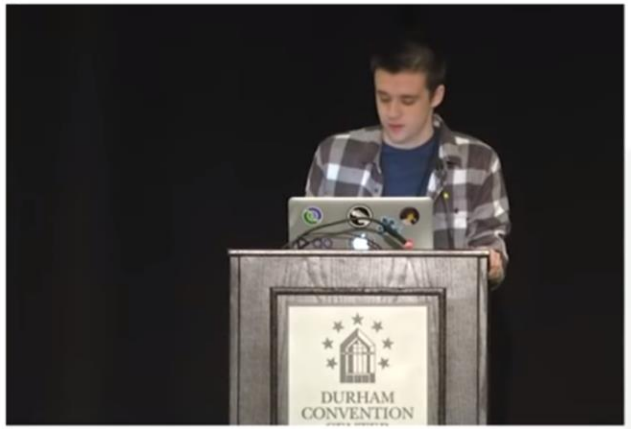
Racket
(originally PLT Scheme)

PLT Research Group

Algol Committee (incl. Perlis)



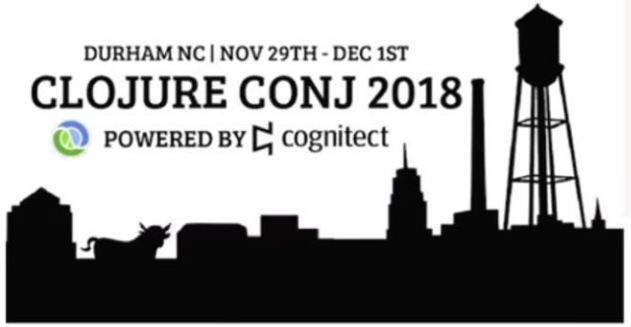
It's Just Data



Every Clojure Talk Ever

How Clojure saved {{company}} and transformed our {{noun}}!

Clojure/conj 2018



Clojure!

code_report • 1.5K views • 4 months ago

Original Talk: https://www.youtube.com/watch?v=jlPaby7suOc **Clojure** Website: https://**clojure**.org/ As mentioned at the end of the ...


:data {:data
[:data "data"]}
its-not-code.clj
< 2L, 39C written

In Pascal the plethora of declarable data structures induces a specialization within functions that inhibits and penalizes casual cooperation. It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures. As a result the pyramid must stand unchanged for a millennium; the organism must evolve or perish.

http://pu.inf.uni-tuebingen.de/users/klaeren/epigrams.html

```
(define p (cons 42 1729))
(car p) ; 42
(cdr p) ; 1729
```

	Language	First	Rest
	Scheme	car	cdr
	Racket	first	rest
	Clojure	first	rest
	Haskell	head	tail
	C++	rv::take(1)	rv::tail
	Elixir	hd	tl
	OCaml	hd	tl
	APL	1↑	1↓

```
(define p (cons 42 1729))
(car p) ; 42
(cdr p) ; 1729
```

```lisp
(define p (cons 42 1729))
(car p) ; 42
(cdr p) ; 1729
```

```cpp
auto p = std::pair{42, 1729};
p.first;  // 42
p.second; // 1729
```

```
(define (cons x y)
  (define (dispatch m)
    (cond ((eq? m 'car) x)
          ((eq? m 'cdr) y)))
  dispatch)
```

```
enum class msg : bool { FIRST, SECOND };

auto pair(auto x, auto y) {
    return [x, y](auto m) {
        if      (m == msg::FIRST)     return x;
        else /* (m == msg::SECOND) */ return y;
    };
}
```

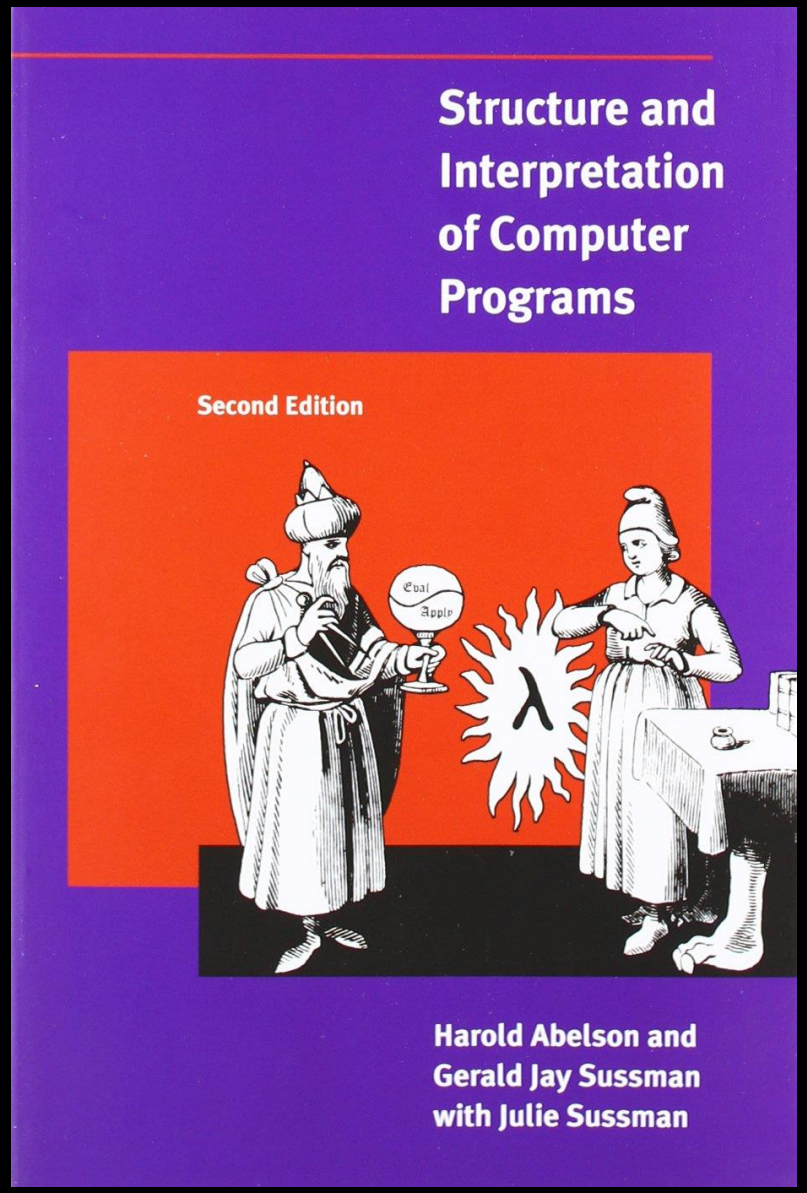
```cpp
enum class msg : bool { FIRST, SECOND };

auto pair(auto x, auto y) {
    return [x, y](auto m) {
        if      (m == msg::FIRST)     return x;
        else /* (m == msg::SECOND) */ return y;
    };
}


auto p = pair(42, 1729);

fmt::print("{}\n", p(msg::FIRST));  // 42
fmt::print("{}\n", p(msg::SECOND)); // 1729
```

Highlights



Data Abstraction

The general technique of isolating the parts of a program that deal with how data objects are represented from the parts of a program that deal with how data objects are used is a powerful design methodology called *data abstraction*. We will see how data abstraction makes programs much easier to design, maintain, and modify.

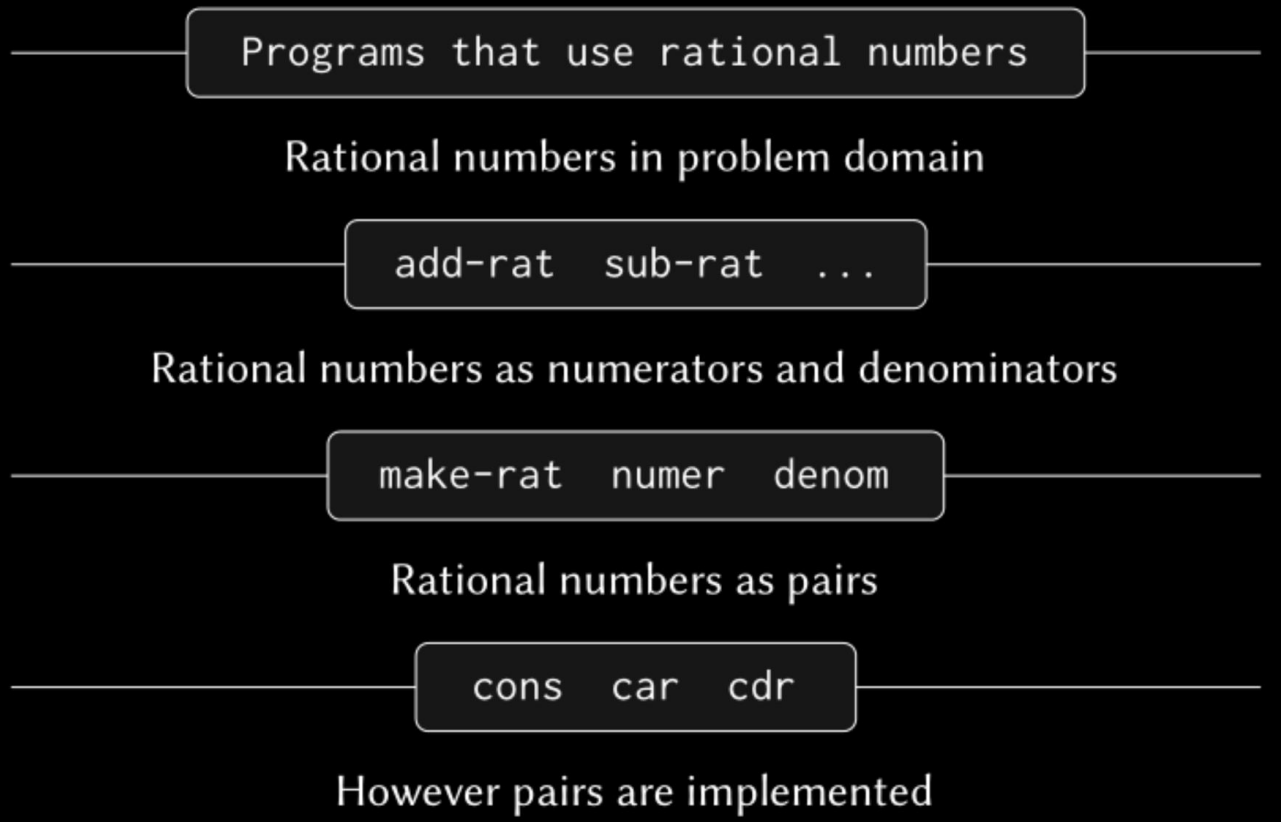
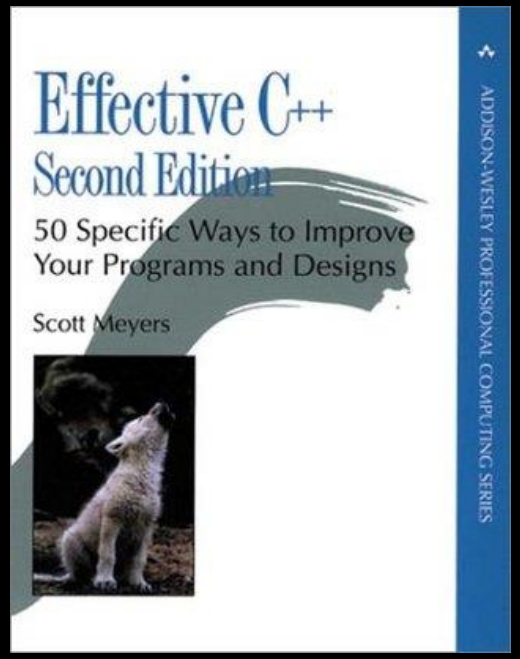
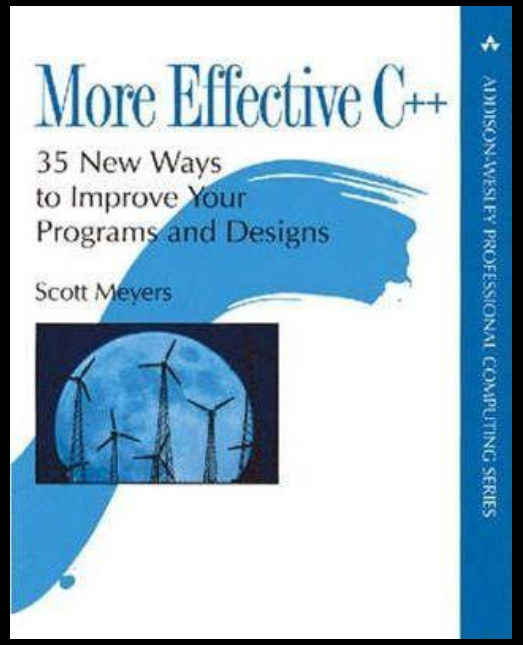
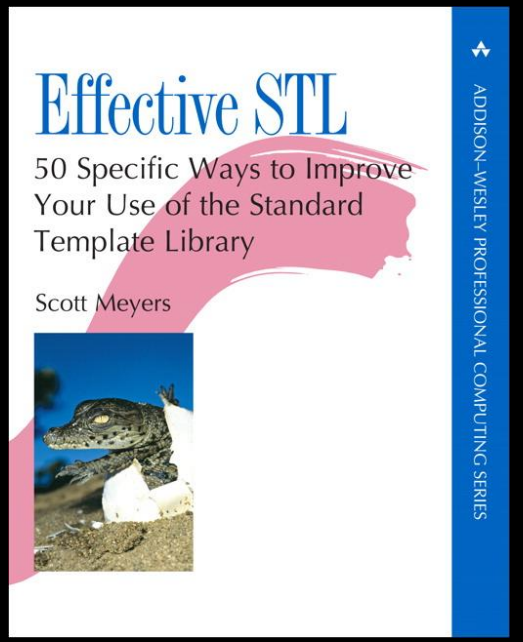
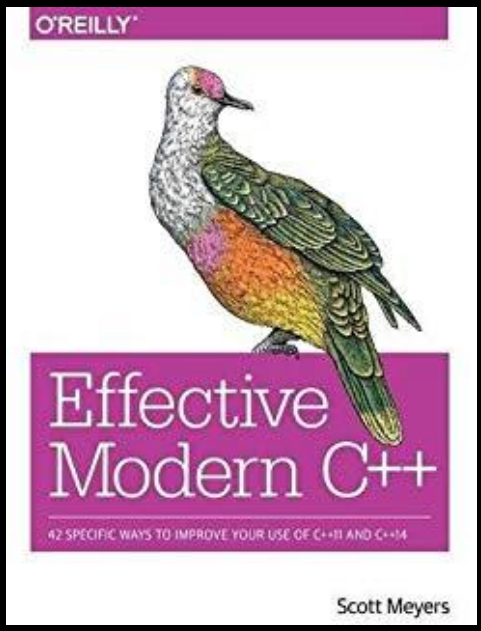


Figure 2.1: Data-abstraction barriers in the rational-number package.

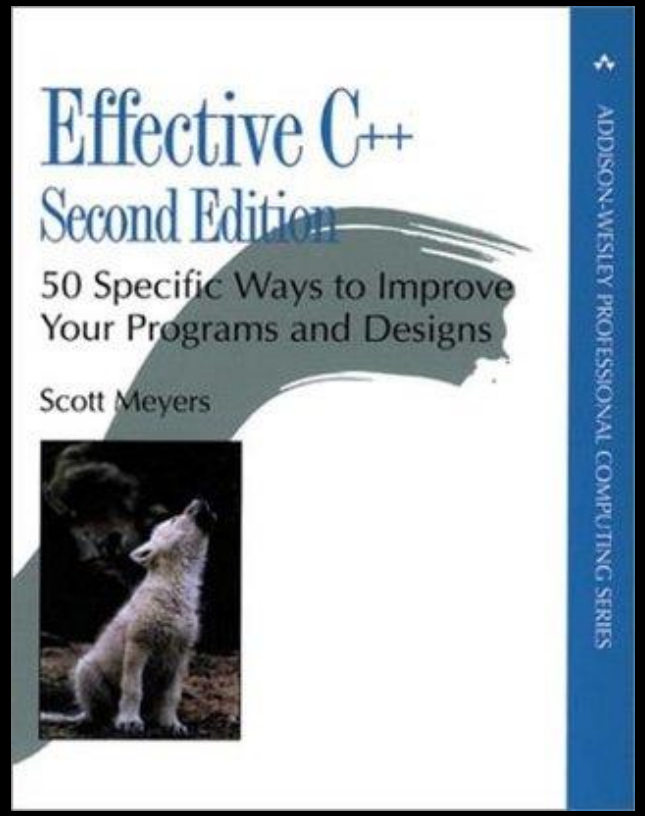

Effective C++
Second Edition
50 Specific Ways to Improve Your Programs and Designs
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


More Effective C++
35 New Ways to Improve Your Programs and Designs
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

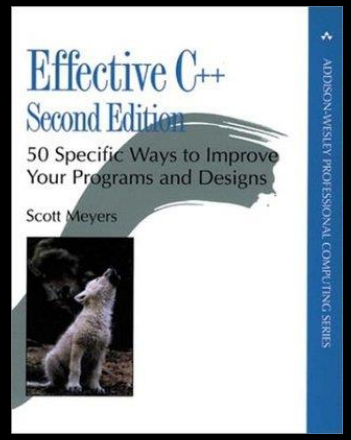

Effective STL
50 Specific Ways to Improve Your Use of the Standard Template Library
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


O'REILLY
Effective Modern C++
42 SPECIFIC WAYS TO IMPROVE YOUR USE OF C++11 AND C++14
Scott Meyers

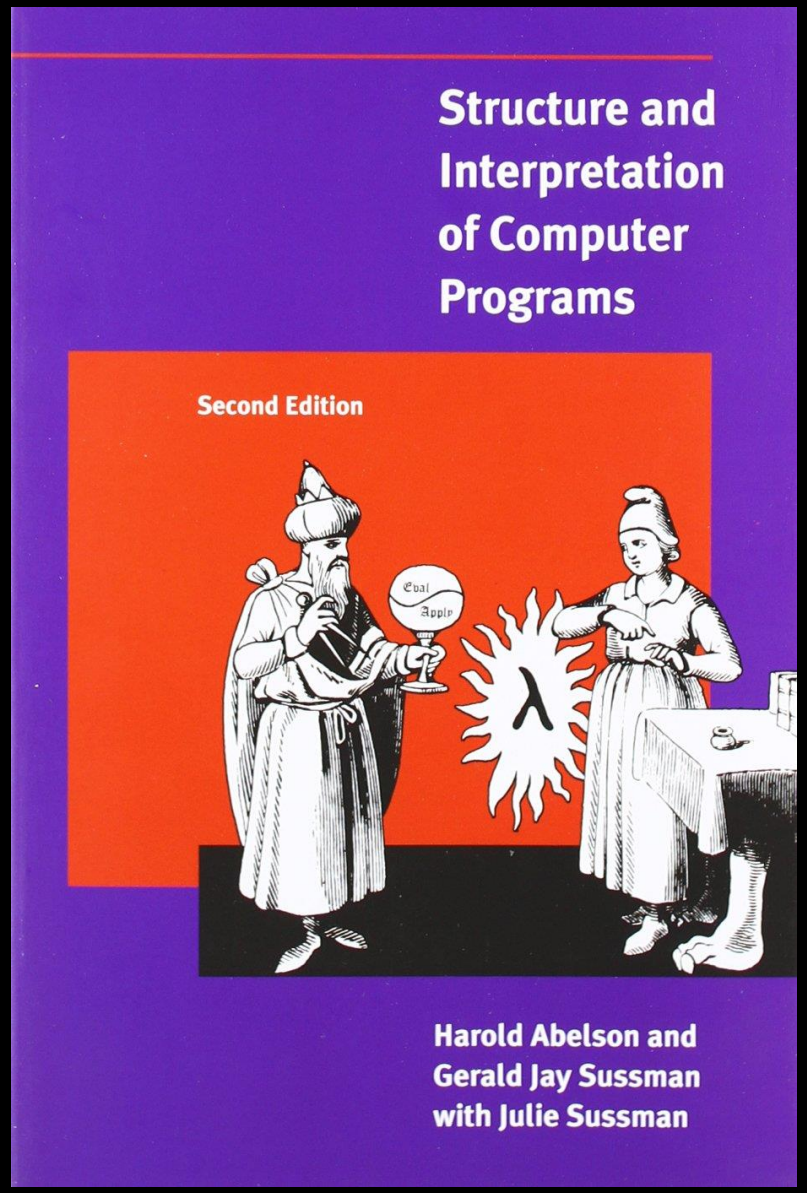
Item 23: Prefer non-member non-friend functions to member functions.



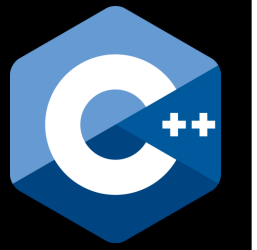
We'll begin with encapsulation. If something is encapsulated, it's hidden from view. The more something is encapsulated, the fewer things can see it. The fewer things can see it, the greater flexibility we have to change it, because our changes directly affect only those things that can see what we change. The greater something is encapsulated, then, the greater our ability to change it. That's the reason we value encapsulation in the first place: it affords us the flexibility to change things in a way that affects only a limited number of clients.



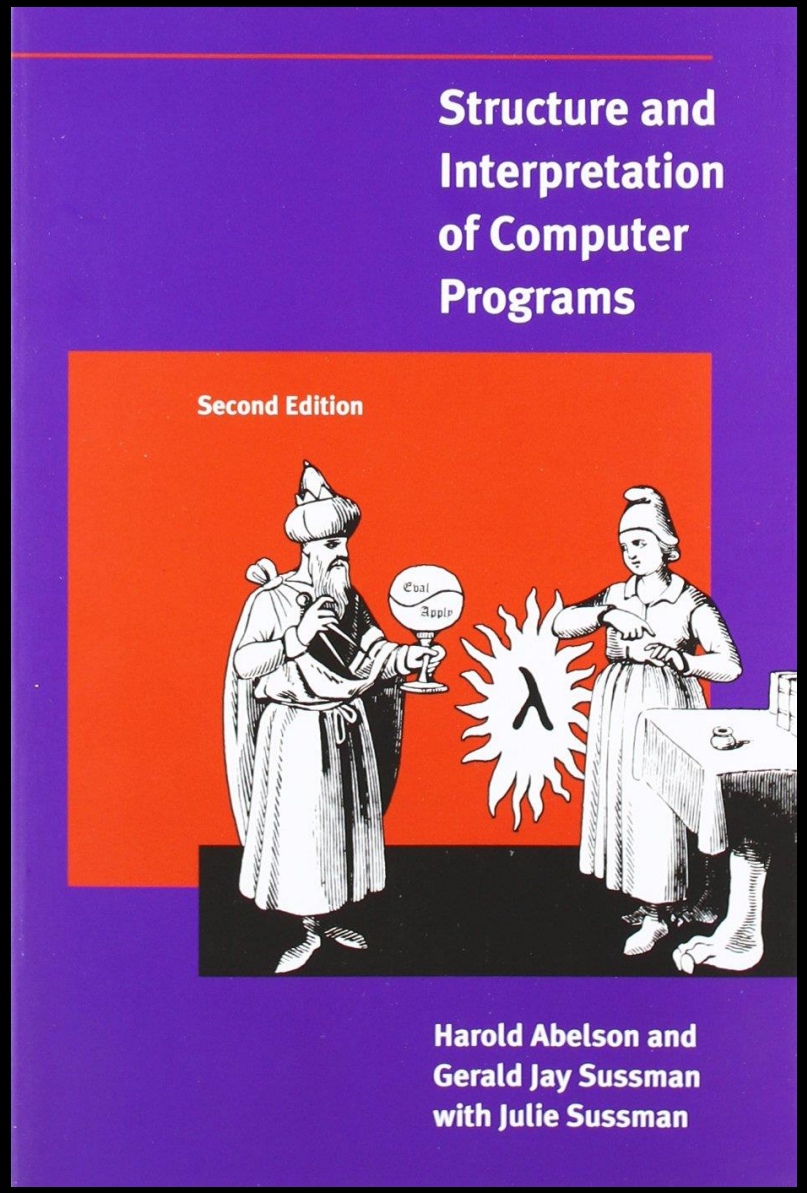
Highlights



OOP?

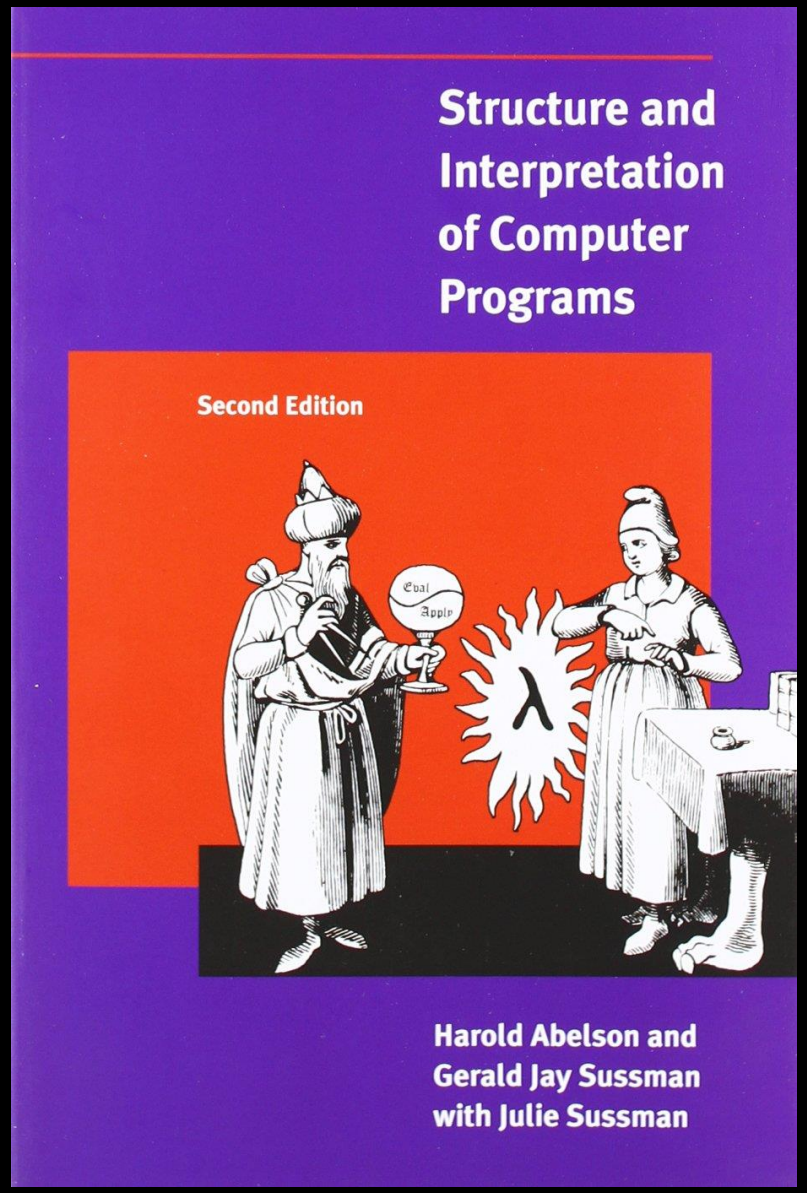
						
Closures (λ)	NO (planned)	kind of	YES	YES	YES	YES
Everything is an object (uniform reference)	NO	NO	NO	YES	kind of	YES
Garbage Collection	NO	YES	YES	YES	YES	YES

	C++	Java	Erlang	Smalltalk	JS	Ruby
Closures (λ)	YES	YES	YES	YES	YES	YES
Everything is an object (uniform reference)	NO	NO	NO	YES	kind of	YES
Garbage Collection	NO	YES	YES	YES	YES	YES


Structure and Interpretation of Computer Programs
Second Edition
Eval
Apply
λ
Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

Problems in


++



Warmup Problem: Sum Square of Two Largest

Warmup Problem:
Sum Square of Two Largest

Given a list of n ($n > 1$) positive integers, return the sum of the squares of the two largest.

Example: [1, 2, 3] -> $2^2 + 3^2 = 13$

[1, 8, 2, 6, 3] -> $8^2 + 6^2 = 100$

x86-64 gcc 10.2 -std=c++20 -O2

```cpp
auto solution(auto const& v) {
    auto const [a, b] = std::accumulate(
        std::cbegin(v),
        std::cend(v),
        std::pair{0, 0},
        [](auto acc, auto e) {
            auto [a, b] = acc;
            if      (e > a) { std::swap(a, b); a = e; }
            else if (e > b) {                  b = e; }
            return std::pair{a, b};
        });
    return a * a + b * b;
}
```

[[digression]]

“We write awful code.”

“We write awful code.
I think that the starting
point for all of us is to just
admit to yourself that
you write awful code.”


Adobe

“We write awful code.
I think that the starting
point for all of us is to just
admit to yourself that
you write awful code.”

Alexander Stepanov, Author of STL

Programming Conversations Lecture 5 Part 2, A9 Videos

stepanovpapers.com

Alexander A. Stepanov

stepanov@gmail.com
June 27, 2019

This site is a collection of Alex Stepanov's books, papers, class notes, and source code, covering generic programming and other topics. Technical contact: paul@mcjones.org.

Books

- *From Mathematics to Generic Programming*
  - Alexander A. Stepanov and Daniel E. Rose: *From Mathematics to Generic Programming*. Addison-Wesley Professional, November 7, 2014. See www.fm2gp.com for more information, including sample chapter, source code, and errata. See below for the videos and slides of the *Three Algorithmic Journeys* course from which this book evolved.
- *Elements of Programming*
  - Alexander Stepanov and Paul McJones: *Elements of Programming*. Semigroup Press, June 2019 and Addison-Wesley Professional, June 2009. See www.elementsofprogramming.com for a free PDF version, a link to purchase a paperback copy, source code, and errata.
  - Alexander Stepanov: Transformations and their orbits. Lecture based on Chapter 2 of *Elements of Programming* presented at Yandex, April 20, 2010.
    - Video. Part 1 Part 2
    - Slides. PDF (See also Chapter 2.)
  - Alexander Stepanov and Paul McJones: Elements of Programming. Lecture presented at Stanford University Department of Electrical Engineering Computer Systems Colloquium (EE380), November 3, 2010.
    - Abstract
    - Slides. PDF
    - Video: Stanford / YouTube / iTunes
- *C++ Standard Template Library*
  - P.J. Plauger, Meng Lee, David Musser, and Alexander A. Stepanov: *C++ Standard Template Library*. Prentice Hall PTR, December 2000, 498 pages.
- *The Ada Generic Library Linear List Processing Packages*
  - David R. Musser and Alexander A. Stepanov: *The Ada Generic Library Linear List Processing Packages*. Springer-Verlag, 1989, 289 pages.

Generic programming

- D. Kapur, D. R. Musser, and A. A. Stepanov: Operators and Algebraic Structures. *Proceedings of the 1981 conference on Functional programming languages and computer architecture*, pages 59-63. PDF (online at acm.org)
  - Also appears as General Electric Company, Corporate Research and Development, Report 81CRD114. PDF
- Deepak Kapur, David R. Musser, and Alexander A. Stepanov: Tecton: A Language for Manipulating Generic Objects. In *Program Specification, Proceedings of a Workshop, Aarhus, Denmark, August 1981,* J. Staunstrup, editor, *Lecture Notes In Computer Science*, volume 134, Springer-Verlag, London, pages 402-414. PDF
- Alexander Stepanov and Aaron Kershenbaum: Using Tournament Trees to Sort, Center for Advanced Technology in Telecommunications Technical Report 86-13, Polytechnic University of New York, 1986. PDF
- Alexander A. Stepanov, Aaron Kershenbaum, and David R. Musser: Higher Order Programming. March 5, 1987. PDF
- David R. Musser and Alexander A. Stepanov: A library of generic algorithms in Ada. *Proceedings of the 1987 annual ACM SIGAda international conference on Ada*, pages 216-225. PDF (online at acm.org)
  - Also: slides from conference presentation. PDF
- Aaron Kershenbaum, David Musser, and Alexander Stepanov: Higher Order Imperative Programming. Rensselaer Polytechnic Institute Computer Science Department, Technical Report 88-10. April 1988. PDF PostScript
- David R. Musser and Alexander A. Stepanov: Generic Programming. ISSAC 1988, pages 13-25. PDF PostScript
- D. R. Musser and A. A. Stepanov: Ada Generic Library Linear Data Structure Packages, Release 1.1, General Electric Company, Corporate Research and Development.
  - Volume One, Report 88CRD112, March 4, 1988. PDF
  - Volume Two, Report 88CRD113, March 3, 1988. PDF
- D. Kapur, D. Musser, W. Olthoff, A. Snyder, A. Stepanov, and A. Szymanski. A Prototyping Language for Rapid Reuse : Technical Proposal. Software Technology Laboratory Report STL-89-10, Hewlett-Packard Laboratories, 1989. PDF
- Alexander A. Stepanov: Design of Generic Libraries. Lecture given at SRI, 1991. PDF
- David R. Musser and Alexander A. Stepanov: Algorithm-oriented Generic Libraries.
  - *Software—Practice and Experience*, Vol. 24(7), July 1994, pages 623-642. PDF PostScript
  - HP Laboratories Technical Report 94-13, February 1994. PDF (online at hp.com)
  - Preprint. September 1993. PDF PostScript
  - Algorithm-Oriented Generic Libraries. Rensselaer Polytechnic Institute Computer Science Department Technical Report 92-23, 1993.
  - Algorithm-Oriented Generic Software Library Development. HPL-92-65(R.1), November 1993 PDF (online at hp.com)
  - Algorithm-Oriented Generic Software Library Development. Rensselaer Polytechnic Institute Computer Science Department Technical Report 92-13, April 1992.
- Mehdi Jazayeri, Meng Lee, and Alex Stepanov: Generic C++ Components. Hewlett-Packard Laboratories/CSL/CCD/PAP/CLL talk, October 4, 1993, 17 slides. PDF
- Alexander A. Stepanov: Science of C++ Programming. Invited presentation to C++ standards committee, San Jose, California, November 11, 1993, 25 slides. PDF
- Meng Lee & Alexander Stepanov: Science of C++ Programming. Hewlett-Packard Laboratories, January 1994, 45 slides. PDF
- Alexander Stepanov & Meng Lee: The Standard Template Library. Presentation to the C++ standards committee, March 7, 1994, 19 slides. PDF
- Alexander Stepanov and Meng Lee: The Standard Template Library. HP Laboratories Technical Report 95-11(R.1), November 14, 1995. PDF PostScript FrameMaker
  - Revised version of A. A. Stepanov and M. Lee: The Standard Template Library, Technical Report X3J16/94-0095, WG21/N0482, ISO Programming Language C++ Project, May 1994.
  - Supercedes Alexander Stepanov and Meng Lee: The Standard Template Library, HP Laboratories Technical Report 94-34(R.1), April 13, 1994.

- Al Stevens Interviews Alex Stepanov: *Dr. Dobb's Journal,* March 1995. PDF HTML (online at sgi.com via Internet Archive)
- Graziano Lo Russo: An Interview with A. Stepanov: *Edizioni Infomedia S.r.l.* PDF HTML (online at stlport.org) Original Italian version. PDF
- Yuyong Zhao: An interview with Alex Stepanov. *Chinese Popular Computer Week,* 28 February 2005. PDF Original Chinese version. PDF HTML
- Andrew Binstock: An interview with Alexander Stepanov and Paul McJones on *Elements of Programming*. *informIT,* August 2009. (online at informit.com)
- Robert Rozeboom: Alexander Stepanov and Daniel E. Rose Answer Your Questions. *Slashdot,* January 2015. (online at slashdot.org)
- John Lakos: *From Mathematics to Generic Programming*: An Interview with Alexander Stepanov and Daniel Rose. *informIT,* February 2015. (online at informit.com)

Bibliographies

- DBLP Bibliography Server entry for Alexander A. Stepanov (local copy)

Source code

- Alexander Stepanov: Scheme higher order programming library, August 1986. .tar unpacked notes
  Array_alg within the USL C++ Standard Components. C documentation
- David R. Musser and Alexander A. Stepanov: Ada Generic Library (source files described in *The Ada Generic Library: Linear List Processing Packages*, Compass Series, Springe
- Alexander Stepanov: Algorithmic simulation and measurement codes, 1993. .zip unpacked
- Alexander Stepanov and Meng Lee: The Standard Template Library, HP Laboratories, release of October 31, 1995.
  - Release directory. Contains STL, FAQ, and related materials. unpacked (originally from ftp://butler.hpl.hp.com/stl/)
  - stl.zip unpacked (originally from ftp://butler.hpl.hp.com/stl/stl.zip)
- Alex Stepanov: Abstraction Penalty Benchmark, version 1.2 (KAI). Silicon Graphics, Incorporated, 199?. C++
  - Also appears as Appendix D.3 of Technical Report on C++ Performance, ISO/IEC PDTR 18015, 11 August 2003 PDF
- Matthew Austern and Alexander Stepanov: jal (Java Algorithm Library), Silicon Graphics, Incorporated, 1996.
  - Source (generic) version: .tar.gz unpacked
  - Source (instantiated) version: .jar documentation
- Matt Austern with Hans Boehm (managed by Alexander Stepanov). SGI Standard Template Library. Source code and documentation for the version of the Standard Template Libr
- Bjarne Stroustrup and Alex Stepanov: Standard Container Benchmark, version 0.9. May 2003 C++

Class notes and videos

Note: The documents in this section have been superseded by From Mathematics to Generic Programming and Elements of Programming (see above).

- Alexander Stepanov: Notes for the Higher Order Programming in Scheme course taught at the General Electric Corporate R&D Center. Schenectady, New York, July 1986. PDF te
- A. A. Stepanov: CS 603 Notes. Department of Electrical Engineering and Computer Science, Polytechnic University of New York, 1986(?). PDF
- Alexander A. Stepanov: Handouts for the Advanced Programming class taught by David R. Musser at Rensselaer Polytechnic Institute. 1990. PDF

it would also be nice to implement reduction (reduction operator was introduced by Kenneth Iverson in APL)

```
(define (reduce iterator)
  (lambda (function . initial-value)
    (define (add-to x)
      (set! initial-value (function initial-value x)))
    (cond (initial-value
           (set! initial-value (car initial-value))
           (iterator add-to)
           initial-value)
          (else
           (let ((marker #!false))
             (define (first-time x)
               (set! initial-value x)
               (set! marker #!true)
               (set! first-time add-to))
             (iterator (lambda (x) (first-time x)))
             (if marker initial-value (function)))))))
```

[[end of digression]]

```cpp
auto solution(auto const& v) {
    auto const [a, b] = std::accumulate(
        std::cbegin(v),
        std::cend(v),
        std::pair{0, 0},
        [](auto acc, auto e) {
            auto [a, b] = acc;
            if      (e > a) { std::swap(a, b); a = e; }
            else if (e > b) {                  b = e; }
            return std::pair{a, b};
        });
    return a * a + b * b;
}
```

NDC { Oslo }

9

There's Treasure Everywhere

Prepared for NDC Oslo 2016

Andrei Alexandrescu, Ph.D.

2016-06-08

```cpp
auto solution(auto const& v) {
    auto const [a, b] = std::accumulate(
        std::cbegin(v),
        std::cend(v),
        std::pair{0, 0},
        [](auto acc, auto e) {
            auto [a, b] = acc;
            if      (e > a) { std::swap(a, b); a = e; }
            else if (e > b) {                  b = e; }
            return std::pair{a, b};
        });
    return a * a + b * b;
}
```

C++20 Concepts

C++17 Structured Bindings

C++14 Function Deduced Return Type

C++14 Generic Lambdas

C++17 CTAD

```
auto solution(auto const& v) {
    auto const [a, b] = std::accumulate(
        std::cbegin(v),
        std::cend(v),
        std::pair{0, 0},
        [](auto acc, auto e) {
            auto [a, b] = acc;
            if      (e > a) { b = a; a = e; }
            else if (e > b) { b = e; }
            return std::pair{a, b};
        });
    return a * a + b * b;
}
```

```cpp
auto solution2(auto v) {
    using namespace std::ranges;
    nth_element(v, std::begin(v) + 2, std::greater{});
    transform(std::cbegin(v), std::cbegin(v) + 2, std::begin(v),
        [](auto e) { return e * e; });
    return std::accumulate(std::begin(v), std::begin(v) + 2,
        0, std::plus{});
}
```

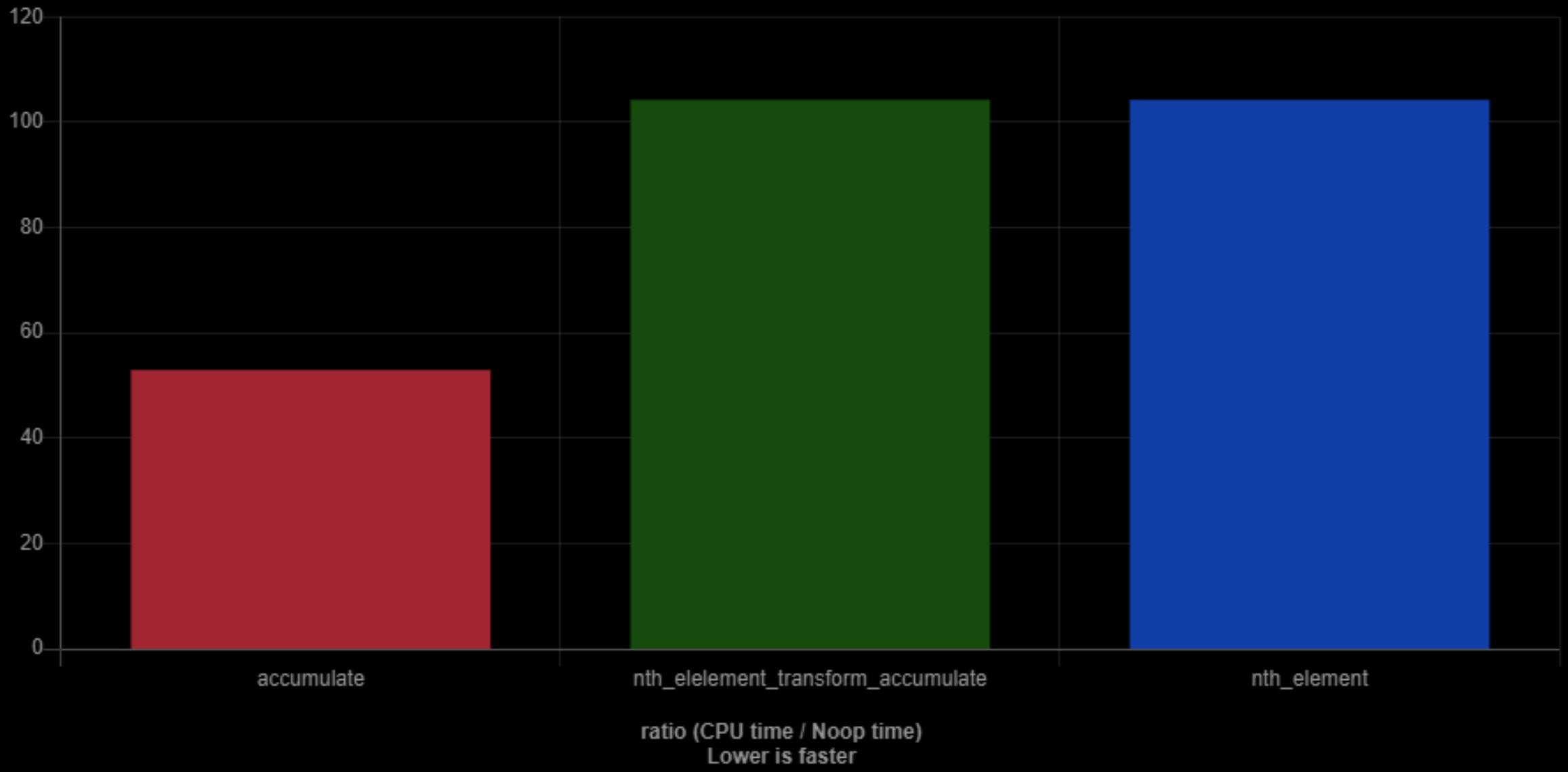
C++14 Function Deduced Return Type

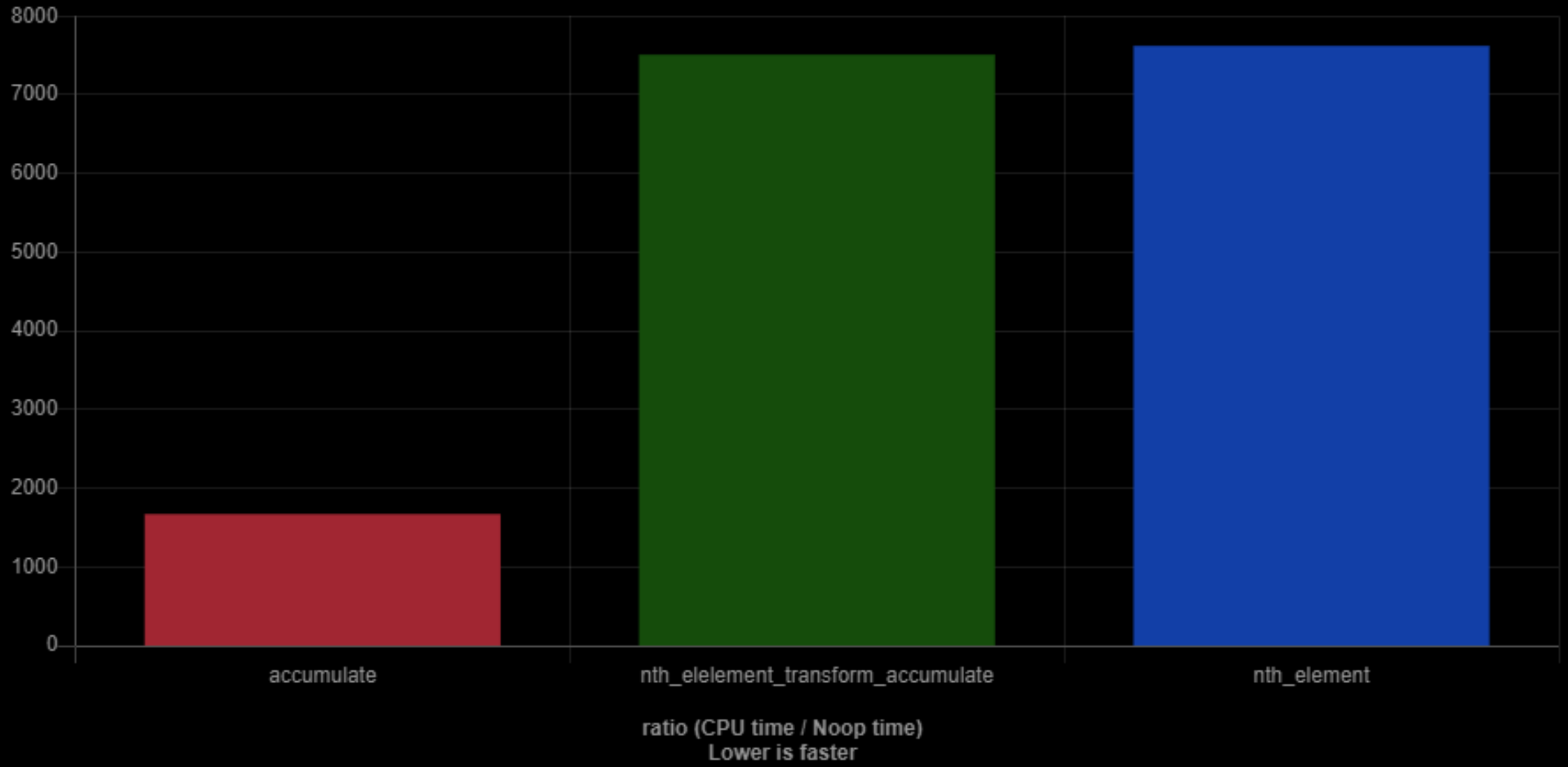
C++20 Concepts

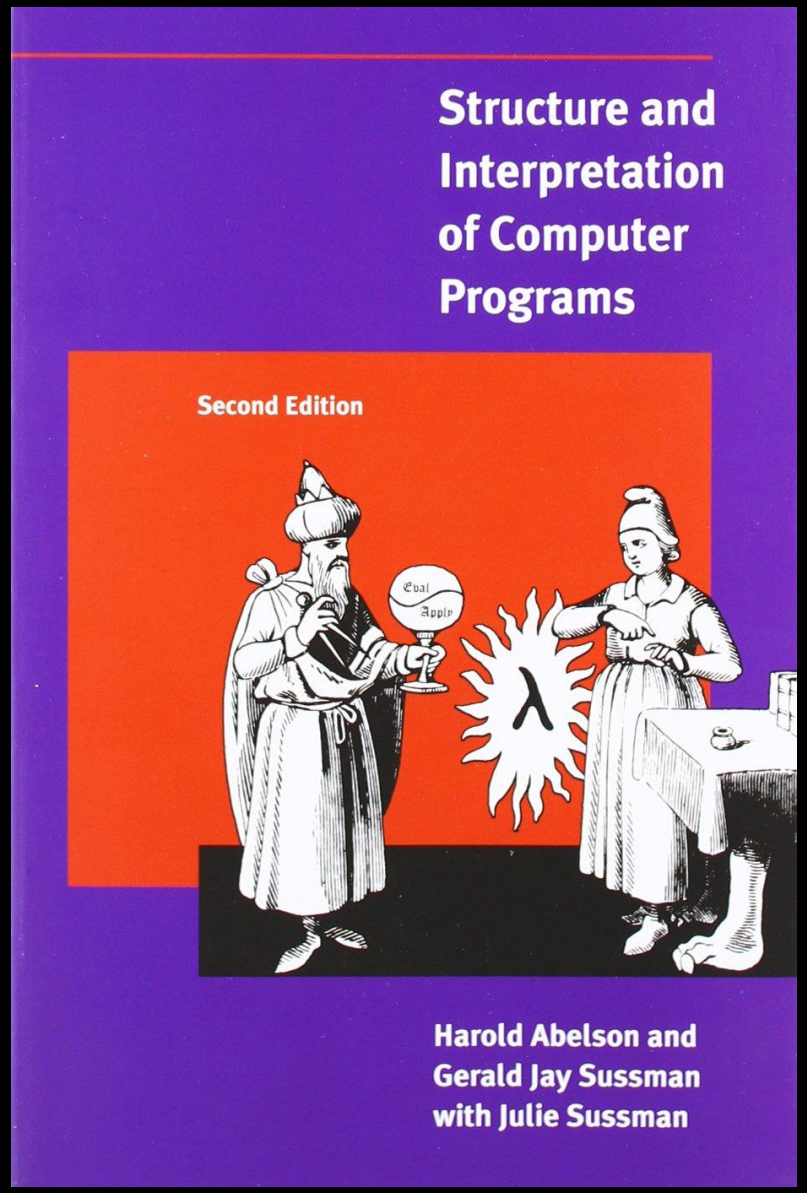
C++17 CTAD

C++20 Ranges

```
auto solution3(auto v) {
    using namespace std::ranges;
    nth_element(v, std::begin(v) + 2, std::greater{});
    return v[0] * v[0] + v[1] * v[1];
}
```


120
100
80
60
40
20
0
accumulate
nth_elelement_transform_accumulate
nth_element
ratio (CPU time / Noop time)
Lower is faster


8000
7000
6000
5000
4000
3000
2000
1000
0
accumulate
nth_elelement_transform_accumulate
nth_element
ratio (CPU time / Noop time)
Lower is faster



Problem 1:
Even Fibonacci
(vs Sum Odd Squares)

```
(define (sum-odd-squares tree)
  (cond ((null? tree) 0)
        ((not (pair? tree))
         (if (odd? tree) (square tree) 0))
        (else (+ (sum-odd-squares (car tree))
                 (sum-odd-squares (cdr tree))))))
```

```
(define (even-fibs n)
  (define (next k)
    (if (> k n)
        nil
        (let ((f (fib k)))
          (if (even? f)
              (cons f (next (+ k 1)))
              (next (+ k 1))))))
  (next 0))
```

```
(define (fib n)
  (fib-iter 1 0 n))
(define (fib-iter a b count)
  (if (= count 0)
      b
      (fib-iter (+ a b) a (- count 1))))

(define (even-fibs n)
  (accumulate
   cons
   nil
   (filter even? (map fib (enumerate-interval 0 n)))))
```

```
(define (fib n)
  (define (fib-iter a b count)
    (if (= count 0)
        b
        (fib-iter (+ a b) a (- count 1))))
  (fib-iter 1 0 n))

(define (even-fibs n)
  (accumulate cons nil
    (filter even?
      (map fib
        (enumerate-interval 0 n)))))
```

[15]Richard Waters (1979) developed a program that automatically analyzes traditional Fortran programs, viewing them in terms of maps, filters, and accumulations. He found that fully 90 percent of the code in the Fortran Scientific Subroutine Package fits neatly into this paradigm. One of the reasons for the success of Lisp as a programming language is that lists provide a standard medium for expressing ordered collections so that they can be manipulated using higher-order operations. The programming language APL owes much of its power and appeal to a similar choice. In APL all data are represented as arrays, and there is a universal and convenient set of generic operators for all sorts of array operations.

A Method for Analyzing Loop Programs

RICHARD C. WATERS, MEMBER IEEE

Abstract—**This paper presents a method for automatically analyzing loops, and discusses why it is a useful way to look at loops. The method is based on the idea that there are four basic ways in which the logical structure of a loop is built up. An experiment is presented which shows that this accounts for the structure of a large class of loops. The paper discusses how the method can be used to automatically analyze the structure of a loop, and how the resulting analysis can be used to guide a proof of correctness for the loop. An automatic system is described which performs this type of analysis. The paper discusses the relationship between the structure building methods presented and programming language constructs. A system is described which is designed to assist a person who is writing a program. The intent is that the system will cooperate with a programmer throughout all phases of work on a program and be able to communicate with the programmer about it.**

Index Terms—**Loops, plans, program analysis, program verification, program understanding.**

I. Introduction

THIS paper presents one part of a general method (described in full in [28]) for analyzing programs. The shows why the resulting proof is more useful than a proof based on a single loop invariant.

An experiment is discussed in Section V which shows that the PBM's can be used to analyze the loops in a representative sample of programs from the IBM Scientific Subroutine Package (SSP) [12]. The SSP was chosen as an object of study because it is a large group of clearly written programs which is an actual commercial product. The experiment also shows that the pieces which result from the analysis are largely simple and easy to understand. A system (described in Section VI) has been implemented which performs this type of analysis automatically. Section VII describes the relationship between the PBM's and current programming language constructs. It also discusses how a language could be extended in order to include constructs based on the PBM's.

The method for analyzing the logical structure of loops presented in this paper was developed as part of a larger research project. The goal of this project is to develop a system which can assist a person who is writing a program. Research on this system [20]–[23], [25], [27], [28] is being carried out by a

It turned out that nearly 90 percent of the time, the pieces which resulted from PBM analysis were very simple. Of the 187 underlying loops, 166 (88 percent) were semantically of the form "DO I=L TO M BY N;" though some of them were not syntactically of that form. Of the 273 augmentations, 243 (89 percent) were easy to recognize as either a product, a sum, a max, a min, a count, or a trivial augmentation whose recurrence relation did not reference prior values of the variable being defined (such as "$X_i=2*Z_i$"). All three filters were simple comparisons with zero.

[15]Richard Waters (1979) developed a program that automatically analyzes traditional Fortran programs, viewing them in terms of maps, filters, and accumulations. He found that fully 90 percent of the code in the Fortran Scientific Subroutine Package fits neatly into this paradigm. One of the reasons for the success of Lisp as a programming language is that lists provide a standard medium for expressing ordered collections so that they can be manipulated using higher-order operations. The programming language APL owes much of its power and appeal to a similar choice. In APL all data are represented as arrays, and there is a universal and convenient set of generic operators for all sorts of array operations.

[15]Richard Waters (1979) developed a program that automatically analyzes traditional Fortran programs, viewing them in terms of maps, filters, and accumulations. He found that fully 90 percent of the code in the Fortran Scientific Subroutine Package fits neatly into this paradigm. One of the reasons for the success of Lisp as a programming language is that lists provide a standard medium for expressing ordered collections so that they can be manipulated using higher-order operations. The programming language APL owes much of its power and appeal to a similar choice. In APL all data are represented as arrays, and there is a universal and convenient set of generic operators for all sorts of array operations.

```cpp
namespace rv = std::ranges::views;

auto fib(auto n) {
    using fun = std::function<int(int,int,int)>;
    fun fib_iter = [&](auto a, auto b, auto count) {
        return count == 0 ? b
                          : fib_iter(a + b, a, count - 1);
    };
    return fib_iter(1, 0, n);
}

auto even_fibs(auto n) {
    return rv::iota(0, n+1)
         | rv::transform([](auto e) { return fib(e); })
         | rv::filter([](auto e) { return e % 2 == 0; });
}
```

```cpp
auto fib(auto n) {
    using fun = std::function<int(int,int,int)>;
    fun fib_iter = [&](auto a, auto b, auto count) {
        return count == 0 ? b
                          : fib_iter(a + b, a, count - 1);
    };
    return fib_iter(1, 0, n);
}
```

```cpp
namespace rv = std::ranges::views;

auto fib(auto n) {
    using fun = std::function<int(int,int,int)>;
    fun fib_iter = [&](auto a, auto b, auto count) {
        return count == 0 ? b
                          : fib_iter(a + b, a, count - 1);
    };
    return fib_iter(1, 0, n);
}

auto even_fibs(auto n) {
    return rv::iota(0, n+1)
         | rv::transform([](auto e) { return fib(e); })
         | rv::filter([](auto e) { return e % 2 == 0; });
}
```

```
auto even_fibs(auto n) {
    return rv::iota(0, n+1)
         | rv::transform([](auto e) { return fib(e); })
         | rv::filter([](auto e) { return e % 2 == 0; });
}
```

```cpp
namespace rv = std::ranges::views;

auto fib(auto n) {
    using fun = std::function<int(int,int,int)>;
    fun fib_iter = [&](auto a, auto b, auto count) {
        return count == 0 ? b
                          : fib_iter(a + b, a, count - 1);
    };
    return fib_iter(1, 0, n);
}

auto even_fibs(auto n) {
    return rv::iota(0, n+1)
         | rv::transform([](auto e) { return fib(e); })
         | rv::filter([](auto e) { return e % 2 == 0; });
}
```

C++20 Concepts

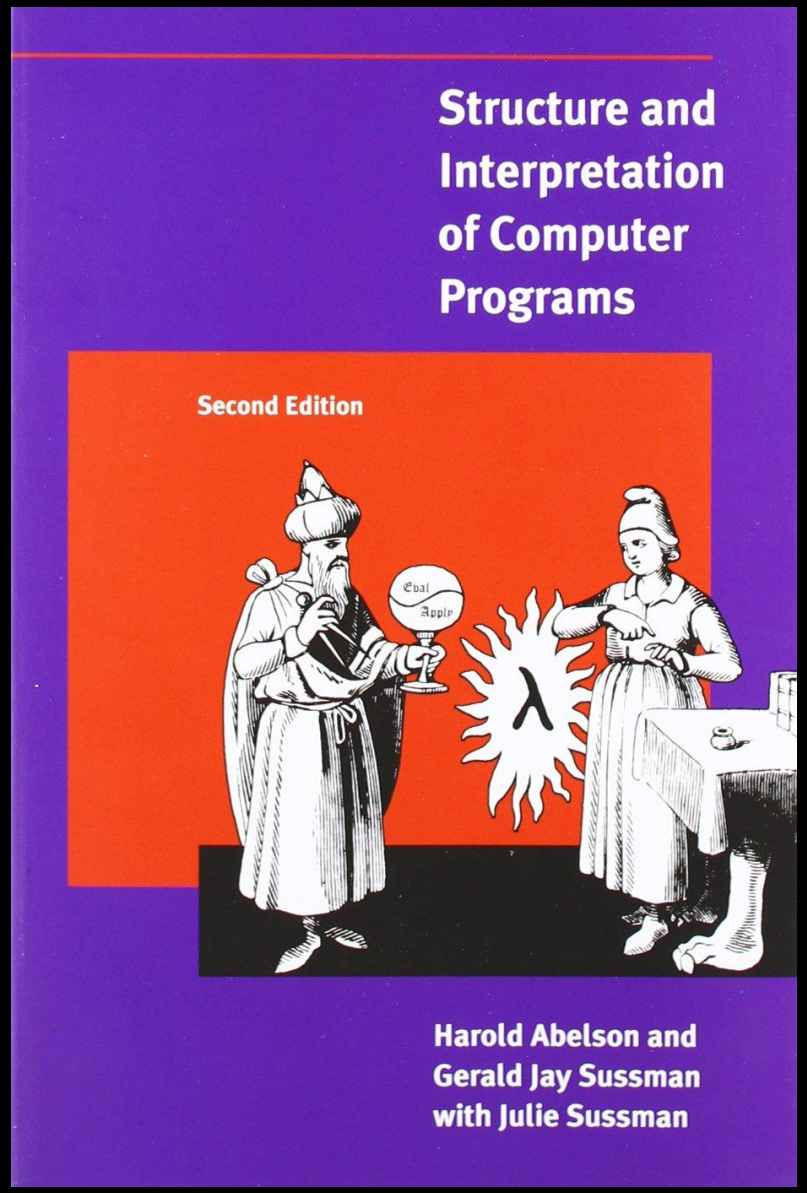
C++14 Generic Lambdas

C++14 Function Deduced Return Type

C++20 Concepts

C++14 Generic Lambdas

C++20 Ranges



Problem 2: Pi Approximation (Leibniz)

$$\frac{\pi}{4} = \frac{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdots}{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdots}.$$

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots.$$

$$\frac{\pi}{8} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \cdots,$$

$$\frac{\pi}{8} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \cdots,$$

$$\frac{\pi}{8} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \cdots,$$

1. Generate odd numbers
2. Group into chunks of two
3. Transform each chunk by multiplying and inverting
4. Add them up
5. Multiply by 8

$$\frac{\pi}{8} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \cdots,$$

1. Generate odd numbers
2. Group into chunks of two
3. transform each chunk by multiplying and inverting
4. Add them up
5. Multiply by 8

```cpp
auto leibniz_pi_approximation(int n) {
    return (rv::iota(0, n)
          | rv::transform([](auto e){ return 1 + 2 * e;})
          | rv::chunk(2)
          | rv::transform([](auto rng) { return 1.0 / (rng[0] * rng[1]); })
          | hs::accumulate(0.0, std::plus{})) * 8;
}
```

```cpp
auto leibniz_pi_approximation(int n) {
    return (rv::iota(0, n)
          | rv::transform([](auto e){ return 1 + 2 * e;})
          | rv::chunk(2)
          | rv::transform([](auto rng) { return 1.0 / (rng[0] * rng[1]); })
          | hs::accumulate(0.0, std::plus{})) * 8;
}


fmt::print("{}\n", leibniz_pi_approximation(10000)); // 3.1414926535900367
```

$$\frac{\pi}{4} = \frac{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdots}{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdots}.$$

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots.$$

$$\frac{\pi}{8} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \cdots,$$

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots.$$

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots$$

1. Generate odd numbers
2. Zip with repeating 1, -1
3. Transform odd numbers by inverting & multiplying with 1/-1s
4. Add them up
5. Multiply by 4

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots$$

1. Generate odd numbers
2. zip with repeating 1, -1
3. transform odd numbers by inverting & multiplying with 1/-1s
4. Add them up
5. Multiply by 4

```
auto leibniz_pi_approximation2(int n) {
    auto const signs = {1,-1};
    return (rv::zip_with(
                [](auto e, auto sign) { return sign * 1.0 / e; },
                rv::iota(0, n) | rv::transform([](auto e){ return 1 + 2 * e;}),
                rv::cycle(signs))
            | hs::accumulate(0.0, std::plus{})) * 4;
}
```

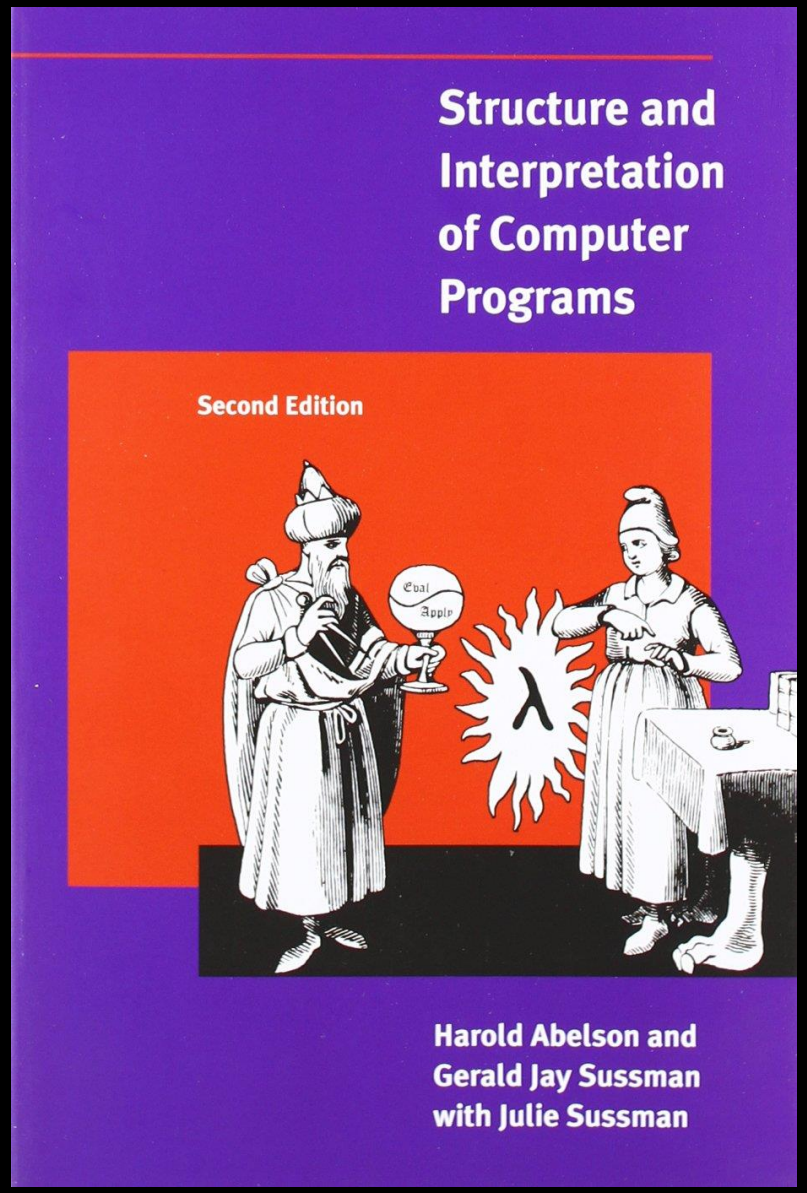
```
auto leibniz_pi_approximation2_alt(int n) {
    return (rv::iota(0, n)
          | rv::transform([s = -1](auto e) mutable {
              s = s > 0 ? -1 : 1;
              return s * 1.0 / (1 + 2 * e); })
          | hs::accumulate(0.0, std::plus{})) * 4;
}
```

```
auto leibniz_pi_approximation2_alt(int n) {
    return (rv::iota(0, n)
            | rv::transform([s = -1.0](auto e) mutable {
                s *= -1;
                return s / (1 + 2 * e); })
            | hs::accumulate(0.0, std::plus{})) * 4;
}
```

$$\frac{\pi}{4} = \frac{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdots}{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdots}.$$

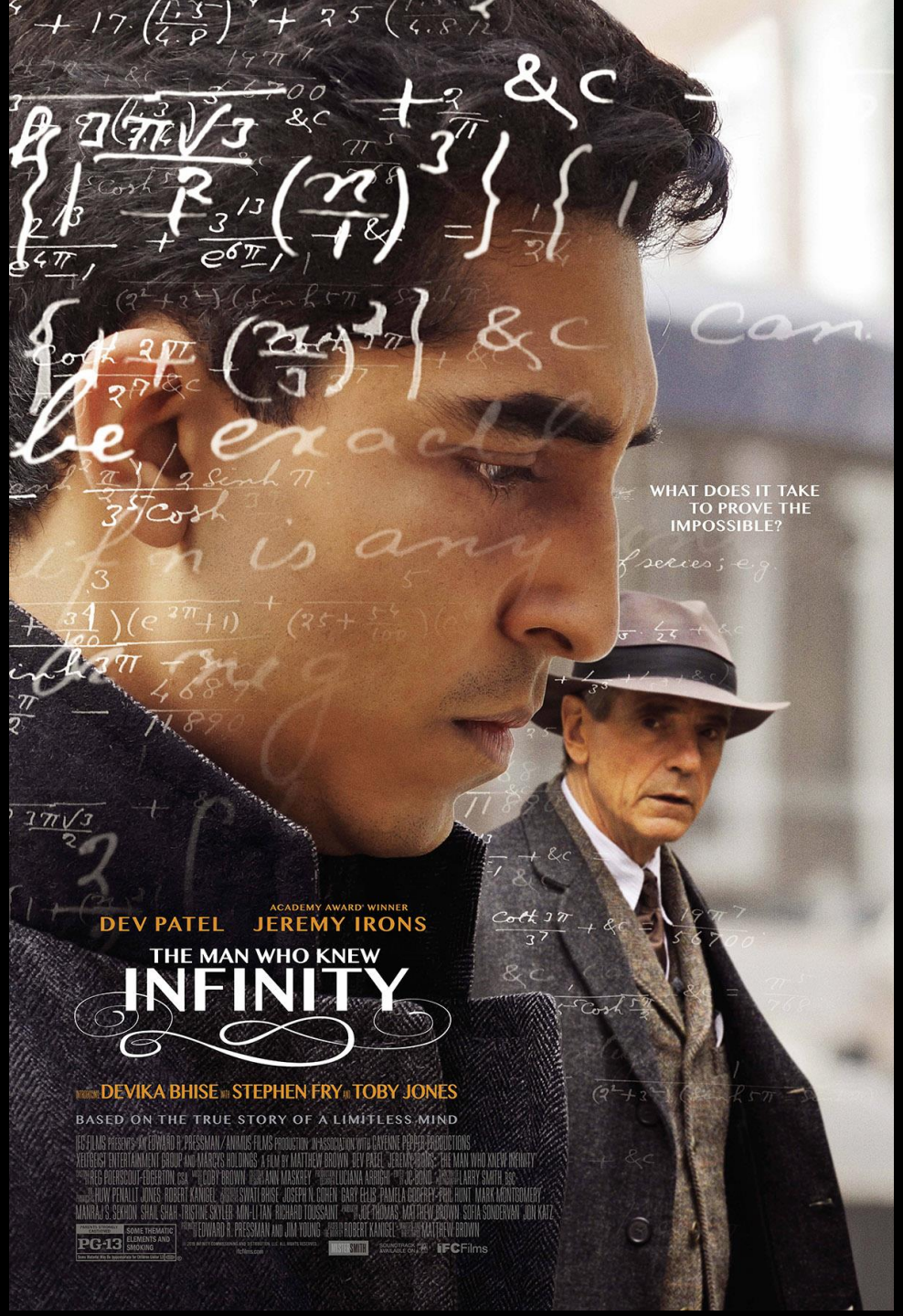
$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots.$$

$$\frac{\pi}{8} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \cdots,$$



Problem 3: Ramanujan Numbers

[70]To quote from G. H. Hardy's obituary of Ramanujan (Hardy 1921): "It was Mr. Littlewood (I believe) who remarked that 'every positive integer was one of his friends.' I remember once going to see him when he was lying ill at Putney. I had ridden in taxi-cab No. 1729, and remarked that the number seemed to me a rather dull one, and that I hoped it was not an unfavorable omen. 'No,' he replied, 'it is a very interesting number; it is the smallest number expressible as the sum of two cubes in two different ways.' "


WHAT DOES IT TAKE TO PROVE THE IMPOSSIBLE?
ACADEMY AWARD® WINNER
DEV PATEL
JEREMY IRONS
THE MAN WHO KNEW
INFINITY
DEVIKA BHISE
STEPHEN FRY
TOBY JONES
BASED ON THE TRUE STORY OF A LIMITLESS MIND
PG-13
SOME THEMATIC ELEMENTS AND SMOKING
IFCFilms


DN 933

[70]To quote from G. H. Hardy's obituary of Ramanujan (Hardy 1921): "It was Mr. Littlewood (I believe) who remarked that 'every positive integer was one of his friends.' I remember once going to see him when he was lying ill at Putney. I had ridden in taxi-cab No. 1729, and remarked that the number seemed to me a rather dull one, and that I hoped it was not an unfavorable omen. 'No,' he replied, 'it is a very interesting number; it is the smallest number expressible as the sum of two cubes in two different ways.' "

Usual Arithmetic Conversions

```
#include <iostream>
using namespace std;
int main() {
    cout << boolalpha;
    short s          = -1;
    unsigned int ui = 1729;
    cout << (s < ui) << endl;
}
```

- What does this print?

Stephan T. Lavavej

C++20 STL Features: One Year of Development on GitHub

Highly Artificial Example 

- GH-843 implemented by AlexGuteniev

```
#include <atomic>
#include <iostream>
#include <thread>
using namespace std;
int main() {
  int i{500};
  i += 500; // ordinary read/write
  { atomic_ref atom{i};
    thread t1{[&atom] { for (int val{0}, x{0}; x < 70;) {
      if (atom.compare_exchange_weak(val, val + 10)) { ++x; }}}};
    thread t2{[&atom] { for (int val{0}, y{0}; y < 29;) {
      if (atom.compare_exchange_weak(val, val + 1)) { ++y; }}}};
    t1.join(); t2.join(); }
  cout << i << endl; // ordinary read, 1729
}
```

Stephan T. Lavavej

C++20 STL Features: One Year of Development on GitHub

$$a^3 + b^3$$

$$a^3 + b^3$$

$$1729$$

$$a^3 + b^3$$

$$1729$$

$$1^3 + 12^3 \qquad 9^3 + 10^3$$

```cpp
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int {
        auto const [a, b] = t;
        return (a * a * a) + (b * b * b);
    };

    auto wpairs = rv::cartesian_product(rv::iota(1, 33), rv::iota(1, 33))
        | rv::filter([](auto r) { return std::get<0>(r) < std::get<1>(r); })
        | rv::transform(sum_cubes)
        | ranges::to<std::vector<int>>
        | ra::sort;

    auto ramanujan = wpairs
        | rv::adjacent_filter(std::equal_to{})
        | rv::drop(1);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

```cpp
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int { ... };

    auto wpairs = rv::cartesian_product(rv::iota(1, 33), rv::iota(1, 33))
        | rv::filter([](auto r) { return std::get<0>(r) < std::get<1>(r); })
        | rv::transform(sum_cubes)
        | ranges::to<std::vector<int>>
        | ra::sort;

    auto ramanujan = wpairs
        | rv::adjacent_filter(std::equal_to{})
        | rv::drop(1);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

Hoogle Translate

cartesianproduct

	Rust	**cartesian_product**	trait.itertools	Doc
	D	**cartesianProduct**	std.algorithm.setops	Doc
	Racket	**cartesian-product**	base	Doc
	C++	**cartesian_product**	range-v3	Doc
	Python	**product**	itertools	Doc
	Ruby	**product**	Array	Doc
	q	**cross**	-	Doc

Hoogle Translate

innerproduct

	APL	**. (inner product)**	-	Doc
	C++	**inner_product**	<numeric>	Doc
	J	**. (Matrix Product)**	-	Doc
	D	**dotProduct**	std.numeric	Doc

Hoogle Translate

outer

	Clojure	**outer-product**	core.matrix	Doc
	APL	**∘. (outer product)**	-	Doc
	Haskell	**outerProduct**	Data.List.HT	Doc
	R	**outer**	-	Doc

1	1	1	2	1	3	1	4	1	5
2	1	2	2	2	3	2	4	2	5
3	1	3	2	3	3	3	4	3	5
4	1	4	2	4	3	4	4	4	5
5	1	5	2	5	3	5	4	5	5

```cpp
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int { ... };

    auto wpairs = rv::cartesian_product(rv::iota(1, 33), rv::iota(1, 33))
        | rv::filter([](auto r) { return std::get<0>(r) < std::get<1>(r); })
        | rv::transform(sum_cubes)
        | ranges::to<std::vector<int>>
        | ra::sort;

    auto ramanujan = wpairs
        | rv::adjacent_filter(std::equal_to{})
        | rv::drop(1);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

1	1	1	2	1	3	1	4	1	5
2	1	2	2	2	3	2	4	2	5
3	1	3	2	3	3	3	4	3	5
4	1	4	2	4	3	4	4	4	5
5	1	5	2	5	3	5	4	5	5

	1 2	1 3	1 4	1 5
		2 3	2 4	2 5
			3 4	3 5
				4 5

```cpp
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int { ... };

    auto wpairs = rv::cartesian_product(rv::iota(1, 33), rv::iota(1, 33))
        | rv::filter([](auto r) { return std::get<0>(r) < std::get<1>(r); })
        | rv::transform(sum_cubes)
        | ranges::to<std::vector<int>>
        | ra::sort;

    auto ramanujan = wpairs
        | rv::adjacent_filter(std::equal_to{})
        | rv::drop(1);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

```cpp
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int { ... };

    auto wpairs = rv::triangle_product(rv::iota(1, 33), rv::iota(1, 33))
        | rv::transform(sum_cubes)
        | ranges::to<std::vector<int>>
        | ra::sort;

    auto ramanujan = wpairs
        | rv::adjacent_filter(std::equal_to{})
        | rv::drop(1);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

```cpp
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int { ... };

    auto is = rv::iota(1, 33);
    auto ramanujan = rv::ordered_triangle_product(is, is, sum_cubes)
        | rv::transform(sum_cubes)
        | rv::adjacent_filter(std::equal_to{})
        | rv::drop(1);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

```cpp
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int { ... };

    auto is = rv::iota(1);
    auto ramanujan = rv::ordered_triangle_product(is, is, sum_cubes)
        | rv::transform(sum_cubes)
        | rv::adjacent_filter(std::equal_to{})
        | rv::drop(1)
        | rv::take(5);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

```cpp
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int { ... };

    auto ramanujan = rv::iota(1)
        |> rv::ordered_triangle_product(_, _, sum_cubes)
        |> rv::transform(_, sum_cubes)
        |> rv::adjacent_filter(_, std::equal_to{})
        |> rv::drop(_, 1)
        |> rv::take(_, 5);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

C++14 Generic Lambdas

C++17 Structured Bindings

C++23 Pipeline Operator

C++20/23/xx Ranges

```
void ramanujan_numbers() {

    auto sum_cubes = [](auto t) -> int {
        auto const [a, b] = t;
        return (a * a * a) + (b * b * b);
    };

    auto ramanujan = rv::iota(1)
        |> rv::ordered_triangle_product(_, _, sum_cubes)
        |> rv::transform(_, sum_cubes)
        |> rv::adjacent_filter(_, std::equal_to{})
        |> rv::drop(_, 1)
        |> rv::take(_, 5);

    std::cout << ramanujan; // [1729, 4104, 13832, 20683, 32832]
}
```

Conclusion

- SICP is an absolutely brilliant & timeless textbook
- You will learn about the fundamentals of CS in not just one but three different programming paradigms
- Many of the examples / exercises in the text are challenging – but it is incredibly rewarding and fun to solve them (in whichever language you choose)
- IT IS FREE! (as well as the MIT & Berkeley Lectures)

Thank You

To all the passionate teachers that inspire

- Gerald Sussman & Hal Abelson
- Alexander Stepanov & Sean Parent
- All of the speakers at

github.com/codereport/Talks

github.com/codereport/Talks

YouTube Video Links:

Speaker	Conference/Meetup	Year	Talk
Sean Parent	Going Native	2013	C++ Seasoning
Kate Gregory	Meeting C++	2017	It's Complicated
Alex Engelberg & Derek Slager	Clojure Conj	2018	Every Clojure Talk Ever
Sean Parent	A9 Programming Conversations	2014	3 Goals for Better Code - Part 1
Sean Parent	A9 Programming Conversations	2014	3 Goals for Better Code - Part 2
Andrei Alexandrescu	NDC Oslo	2016	There's Treasure Everywhere

github.com/codereport/SICP-2020

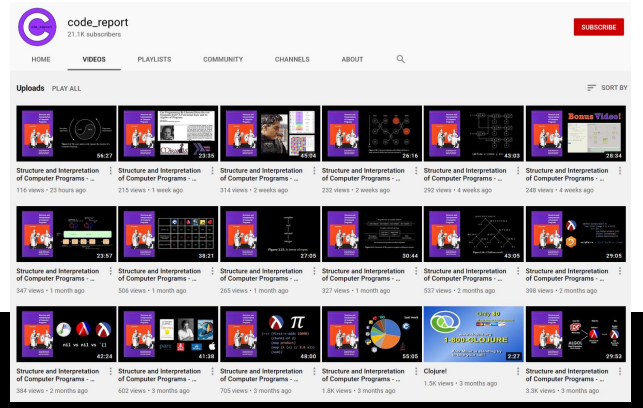
github.com/codereport/SICP-2020

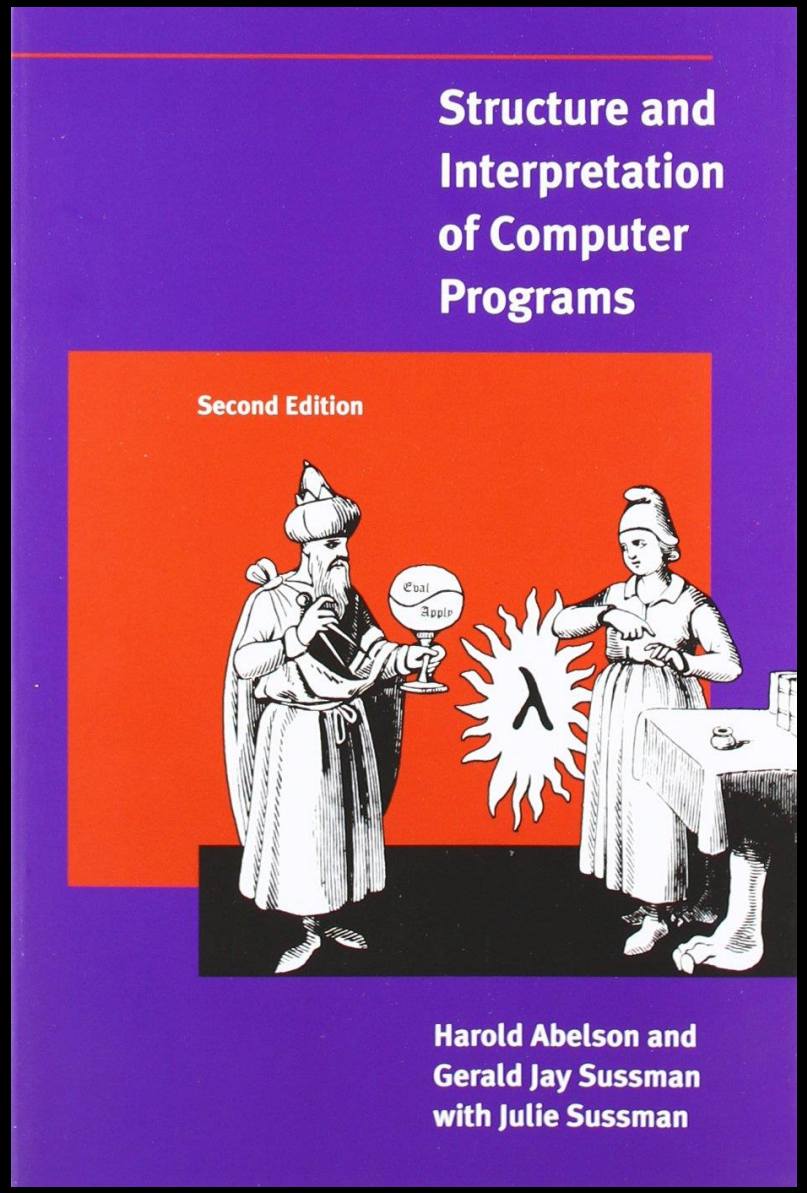
SICP-2020

This is the material (code and presentation slide decks) that correspond to the Programming Languages Virtual Meetup course that covered the **Structure and Interpretation of Computer Programs** textbook.

Read a short article by Brian Harvey on Why SICP Matters?

- **Textbook:** Structure and Iterpretation of Computer Programs
- **MIT Lectures** (by Prof Gerald Sussman and Hal Abelson): YoutTube Playlist | MIT Open CourseWare
- **MIT Lectures** (by Prof Eric Grimson): YouTube Playlist
- **UC Berkeley Lectures** (by Prof Brian Harvey): YouTube Playlist
- **Meetup Pre-recordings:** YouTube Playlist
- **Online STk Interpreter:** https://code.cs61a.org/
- **Logo Interpreter:** https://www.calormen.com/jslogo/





Thank You!

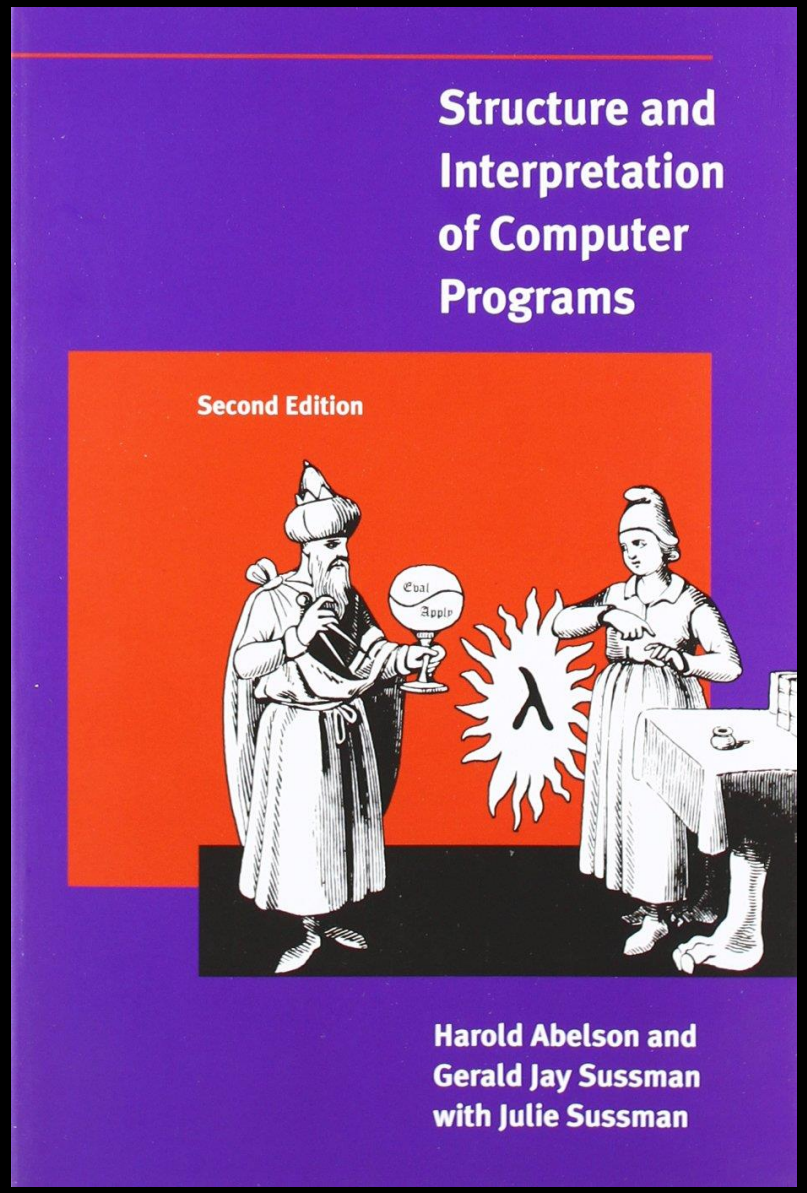
Conor Hoekstra

code_report

NVIDIA

RAPIDS

#include <C++>



Questions?

Conor Hoekstra

code_report

NVIDIA RAPIDS

#include <C++>