Book review

Systematic Program Design: From Clarity to Efficiency, by Yanhong Annie Liu, Cambridge University Press, 2013, ISBN: 978-1-107-03660-4.
doi:10.1017/S0956796813000269

1 Context

The book describes techniques for deriving sophisticated optimized algorithms from simpler but inefficient formulations using systematic techniques. The techniques apply to various classes of algorithm formulations – loops, recursive functions, set comprehensions, Datalog programs, and object-oriented programs.

The title may be misleading to some readers. I expected a work on systematically producing complete programs from problem statements, along the lines of *How to Design Programs* by Felleisen *et al.* (2002, MIT Press). Instead, the present book focusses on improving existing programs, or rather isolated parts of programs – and thus it is fundamentally a book on algorithms.

As such, it is different from the plethora of existing books on algorithms. Most of them present known algorithms, but rarely illustrate systematic techniques that the reader could use to arrive at those algorithms herself. (Hoare famously claimed that Quicksort was the second most obvious sorting algorithm he could come up with after bubble sort. This proves his genius, but is frustrating to many lesser computer scientists; H. Klaeren, personal communication.) Liu's book focusses on techniques the reader can apply herself. Many functional programming papers on clever derivations of algorithms would benefit from such an approach – functional programming prides itself on its principled approach, yet many such derivations feature 'eureka moments' that spring from the author's intuition rather than the straightforward application of an explicitly formulated systematic technique.

Liu draws from a long list of her own previous publications, most of which share this common theme. The techniques presented in the book all follow the same blueprint, described in the Introduction as follows.

> *The method has three steps: Iterate, Incrementalize, and Implement [...]*
>
> 1. *Step Iterate determines a minimum increment operation to take repeatedly, iteratively, to arrive at the desired program output.*
> 2. *Step Incrementalize makes expensive computations incremental in each iteration by using and maintaining appropriate values from the previous iteration.*
> 3. *Step Implement designs appropriate data structures for efficiently storing and accessing the values maintained for incremental computation.*

While these steps are probably instinctively familiar to experienced designer of algorithms, their explicit formulation was eye-opening for me. Throughout the book, Liu makes good on her promise to present systematic techniques. This requires careful presentation and explicit description, with sometimes extreme amounts of detail. This will seem overly pedantic to a casual reader, but makes a closer reading quite rewarding.

While many of the original algorithm formulations in the book are functional or declarative, her techniques mostly produce imperative code. Hence, it is not primarily a 'functional-programming book', but still makes for relevant and useful reading for functional programmers.

2 Overview

As mentioned above, the book's all techniques start with an existing program, and then produce efficient algorithms using the 'III' steps. Each of the book's central chapters treats existing programs written in a particular form: loops, set comprehensions, recursive functions, Datalog-style rule sets, and object-oriented stateful programs.

In each chapter, the book shows how to apply the 'III' steps to a particular program form. Each chapter features a running example that illustrates what makes the original program inefficient, what needs to be done to improve it, and how those improvements result from systematic transformations or translations. The discussion is accompanied by a definition of the source language, and the cost model used to gauge the original formulation as well as the resulting program. Each chapter then features additional examples and applies the presented techniques to them afresh. When necessary, the chapters describe techniques for constructing the data structures needed. The resulting programs rarely rely on lookup tables, but rather store needed information directly with the objects manipulated by the algorithms using *based representations*. (A technique I had often seen in algorithm code, but never known a name for: Essentially, objects contain references to the collections of which they are members.)

'Step Incrementalize', the most important 'III' step, produces programs that compute results incrementally as they traverse the input, caching intermediate results along the way. This makes some of the resulting algorithms, particularly those representing Datalog programs, incremental. They will be able to efficiently reflect in the output any small changes made in the input.

The chapter on object-oriented programs focusses not so much on algorithms but instead on the propagation of data dependencies through object graphs. The observer pattern can make this propagation modular, but is often non-incremental and therefore inefficient. Liu shows how to propagate incremental information through the program once the observer pattern is in place, thus optimizing it.

The central chapters of the book are bracketed by an introduction that introduces the 'III' steps and a conclusion chapter that takes a deeper look at incrementalization, and reviews some aspects of the material in the book and the future work.

3 Audience

The book is written with many of Liu's papers as source material, and hence is visibly written-up research. It does not explicitly identify a particular target audience, but instead starts with a list of program families where the book may help produce efficient algorithms: database queries, hardware design, image processing, string processing, graph analysis, and querying complex relationships. Anyone writing programs in one of these areas, and, more generally, anyone who occasionally finds herself in the situation of having written a straightforward program that runs too slowly, will find the book accessible and useful. Readers should have basic knowledge of algorithms and big-O analysis. Familiarity with set comprehensions and Datalog helps, but is not required.

4 Comparison

The natural objects of comparison with Liu's books are books on algorithm design, such as Edmonds's *How to Think About Algorithms* (2008, Cambridge University Press). I find

Liu's book more constructive in its approach, and thus more empowering. Okasaki's *Purely Functional Data Structures* (1999, Cambridge University Press) does describe more general techniques for designing functional data structures, but in more general terms than Liu's book. On the other hand, Liu's book does not provide even remotely encyclopedic coverage of algorithms that may be useful to a programmer. (No Quicksort.) Books like *Introduction to Algorithms* by Cormen *et al.* (2009, third edition, MIT Press) do provide encyclopedic coverage, but describe constructive approaches to algorithm design only in general terms.

I cannot claim extensive knowledge of the field, but I would have appreciated some references in the field of adaptive computation, which seems closely related.

5 Appraisal of ideas

As mentioned above, the book is essentially a compilation of Liu's own research, which was published in a wide variety of venues and journals, including TOPLAS, HOSC, POPL, and PEPM. Although JFP and ICFP are absent, the disciplined approach that Liu takes, as well as functional source programs, should have great appeal for JFP readers. The ideas presented are mostly Liu's own, and those of her co-workers.

Yet I see the strength of the book not in its ideas but in its insistence on *systematic* techniques and presentation. The book does give all necessary details for the reader to apply its techniques to her own programs. While often pedantic (and, in my experience excruciating to write), it is exactly what makes the book useful and unique.

Generally the program transformations are given in prose. I sometimes wished for more formal presentation, as the prose form is not always completely unambiguous. Given how systematic the book's techniques are, they scream for mechanization of some form. (Liu's publication list seems to feature some abandoned attempts.) This would probably be more evident from formal notation rather than the prose presented in the book. I hope mechanization will be the subject of Liu's future work.

6 Production

The book has mostly been well adapted from the research papers from which it originates. Still some sections seem poorly integrated. This is particularly evident in the Conclusion, which presents some new ideas in addition to sections that seem randomly pasted, and which seem to go nowhere.

The book could have benefited from more thorough copy editing: the language is occasionally stilted, and sometimes a little too unconsciously self-congratulatory. The book's associated website is disappointing – there are headings for executable code and slides, but no links to any content.

7 Conclusion

Liu's book is an excellent read for working programmers who occasionally struggle with producing efficient algorithms. It also serves as a valuable introduction to the use of systematic techniques in software development. It deserves a linear read-through, and can then serve as a reference to be kept next to the keyboard when applying its techniques. It does not replace but rather complements a good selection of books on algorithms on a programmer's bookshelf.

MICHAEL SPERBER
Active Group GmbH
Hornbergstraße 49
70794 Filderstadt, Germany