Circular Programming Across Paradigms

Proposal for Post-doc Research Project

João Fernandes

September 25, 2008

Contents

1 Summary

We propose to explore the technique of Circular Programming across several programming paradigms. Circular lazy programs, as introduced by Richard Bird, are a famous example that demonstrates the power of a lazy evaluation mechanism. Bird's work showed that any multiple traversal algorithm can be expressed in a lazy language as a single traversal circular function. Using the style of Circular Programming, the programmer does not have to concern him/herself with the definition and the scheduling of the different traversal functions, since a single (traversal) function has to be defined. Moreover, because there is a single traversal function, the programmer does not have to define intermediate gluing data structures to convey values computed in one traversal and needed in following ones, either.

In our work, we intend to systematically explore the nice of circular programs in programming paradigms such as bidirectional transformations, incremental computing, program calculation, attribute grammars or spreadsheets.

2 The Team

2.1 Post-doc Researcher

Dr. João Fernandes
Departamento de Informática
Universidade do Minho
Braga
Portugal

João Fernandes will obtain his PhD. title in Computer Science from the University of Minho in ? ?. The title of his PhD. thesis is *Design, Implementation and Calculation of Circular Programs.* His promotor and co-promotor are Prof. Dr. João Saraiva (Universidade do Minho), and Prof. Dr. Oege de Moor (Oxford University).

2.2 Supervisors

Prof. Dr. Alberto Pardo
Instituto de Computación
Universidad de la República
Montevideo
Uruguay

Alberto Pardo is Associate Professor of the Instituto de Computación (INCO) at Universidad de la Repblica (Montevideo, Uruguay), which he joined in 1986, and the head of the Formal Methods Group of INCO. He also taught Computer Science courses at Universidade Federal de Pernambuco (Brazil) and Technische Universität Darmstadt (Germany). Alberto Pardo holds an M.Sc. degree from the Universidade Federal de Pernambuco and a PhD from the Technische Universität Darmstadt. His research interests are focused on program transformation techniques for functional programs, generic programming and formal semantics of programming languages.

Alberto Pardo has been responsible for several research projects on program transformation tecniques in subjects close to this project. He is member of the IFIP TC2 Working Group 2.1 on "Algorithmic Languages and Calculi". He has also been involved in the organization of various international events like, for example, the International Summer School on Language Engineering and Rigurous Software Development (LERNET 2008), the 11th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2004), and the International Winter School on Semantics and Applications (WSSA 2003).

Prof. Dr. João Saraiva
Departamento de Informática
Universidade do Minho
Braga
Portugal

João Saraiva is an university lecturer of Computer Science at University of Minho and the research coordinator of the Department of Informatics in the Algoritmi Research Institute. His research is focused on programming language design and implementation, and functional programming. João Saraiva completed a Ph.D. program at Utrecht University, The Netherlands, in December 1999 where he worked on purely functional implementation of attribute grammars.

João Saraiva has been involved in the organization of various international events in the context of this project, for example, the 10^{th} European Joint Conferences on Theory and Practice of Software (ETAPS 2007), the International Summer Schools on Advanced Functional Programming AFP'98 (Swierstra et al., 1999b), Applied Semantics APPSEM'00 (Barthe et al., 2002) and Generative and Transformation Techniques on Sofware Enginnering 2005, 2007 and 2009 (Lämmel et al., 2006), and the International Workshop on Attribute Grammars and their Applications (WAGA) and International Workshop on Language Descriptions Tools and Application (LDTA).

2.3 Host Institutions

This Pos-doc research project will be carried out in the Department of Informatics at Minho University. Several members of the department (José Nuno Oliveira, Luís Barbosa, José Barros, Jorge Sousa Pinto, João Saraiva) are investigating the use of formal methods and generic programming in the context of Program Understanding and Re-engineering.

Programming languages, design and implementation is also an intensive area of research in the department (Pedro Henriques, José Almeida, José Ramalho, João Saraiva). Work in this area has been done around the Lrc project in collaboration with Ordina Research, Holland (Dr. Matthijs Kuiper) and the group on Software Technology of the Department of Computer Science at Utrecht University (headed by Prof. Dr. Doaitse Swierstra), and we keep in regular contact with our industrial and academic collaborators.

vou tentar escrever alguma coisa sobre minhas mutiples visitas ao Minho e o projeto ALFA Uruguay...

3 Introduction/Motivation

Circular lazy programs, as introduced by Bird (1984), are a famous example that demonstrates the power of a lazy evaluation mechanism. Bird's work showed that any multiple traversal algorithm can be expressed in a lazy language as a single traversal *circular function*. Such a (virtual) circular function may contain a *circular definition*, that is, an argument of a function call that is also a result of that same call. Although circular definitions induce non-termination under a strict evaluation mechanism, they can be immediately evaluated using a lazy evaluation strategy. The lazy engine is able to compute the right evaluation order, if that order exists. Indeed, using this style of circular programming, the programmer does not have to concern him/herself with the definition and the scheduling of the different traversal functions, since a single (traversal) function has to be defined. Moreover, because there is a single traversal function, the programmer does not have to define intermediate gluing data structures to convey values computed in one traversal and needed in following ones, either.

Bird's work showed the power of circular programming, not only as an optimization technique to eliminate multiple traversal of data, but also as a powerful, elegant and concise technique to express multiple traversal algorithms. Circular programs are also used in the construction of Haskell compilers (Marlow and Jones, 1999; Hinze and Jeuring, 2002), to express pretty printing algorithms (Swierstra et al., 1999a), breadth-first traversal strategies (Okasaki, 2000), type systems (Dijkstra and Swierstra, 2004) and aspect-oriented compilers (de Moor et al., 2000). As an optimization technique, circular programs are used, for example, in the deforestation of accumulating parameters (Voigtländer, 2004). Circular programs can also be obtained through partial evaluation (Lawall, 2001) and continuations (Danvy and Goldberg, 2002).

4 Tasks

In this section we present the different working directions that compose our reserach proposal. All of them have circular programming as the common component.

4.1 Strictification of Circular Programs in Calculational Form

Functional programs often combine separate parts of the program using intermediate structures for communicating results. Programs such as $prog = cons \circ prod$, where *prod* is called the producer function and *cons* is called the consumer function, are modular and have many benefits, such as clarity and maintainability, but suffer from inefficiencies caused by the generation of the intermediate data structures that glue functions *cons* and *prod* together.

Indeed, the elimination of such intermediate structures, also called program fusion, is a key aspect taken into account in the implementation of functional compilers (Jones et al., 1993; Leroy, 1997). Program fusion has, therefore, been thoroughly studied in the context of functional languages: for pure programs (Wadler, 1990; Gill et al., 1993; Onoue et al., 1997; Ohori and Sasano, 2007; Fernandes et al., 2007) but also in the context of monadic programs (Ghani and Johann, 2008; Manzino and Pardo, 2008; Pardo et al., 2009).

Another key optimization to implement in compilers for lazy languages such as *Haskell* is the static detection of functions that can be evaluated strictly. Indeed, an important area of research is strictness analysis (**?**). Thus, we plan to study how to transform circular programs into strict ones, using calculational techniques and proving the correctness of the transformation. In this research, we will closely follow the approach studied in (Fernandes and Saraiva, 2007), where attribute grammar techniques have been adapted to transform circular programs into compositions of strict programs. The correctness of such techniques, however, remains to be formally proved.

In order to be able to calculate strict programs from circular ones, we initially will try to invert the transformation presented in (Fernandes et al., 2007): there, a calculation rule was proposed to calculate circular programs from strict ones.

4.2 Bidirectional Transformations

There are many situations in which one data structure, called *source*, is transformed to another, called *view*, in such a way that changes on the view can be transformed back to those on the original data structure. This is called Bidirectional Transformation (BT), and pratical examples include synchronization of replicated data in different formats (Foster et al., 2005), presentation-oriented structured document development (Hu et al., 2004; Michiel, 2004), interactive user interface design (Meertens, 1998), and the well-known view updating mechanism which has been intensively studied in the database community (Bancilhon and Spyratos, 1981; Dayal and Bernstein, 1982; Gottlob et al., 1988; Lechtenbörger and Vossen, 2003). The simple diagram presented below illustrates a Bidirectional Transformation.

$$\begin{tikzcd}
\mathit{source} \arrow[r, "\text{forward transformation}"] \arrow[d, bend left=60, "\text{backward transformation}"'] & \mathit{view} \arrow[d] \\
\mathit{source}' & \mathit{view}' \arrow[l]
\end{tikzcd}$$

Figure 1: Bidirectional Transformation

It was during a visit of the candidate to one of the leading research groups in the area of Bidirectional Transformation, at the University of Tokyo, that it was preliminary discussed how such transformations could benefit from the properties of circular programs. Indeed, circular programs may provide an ideal setting to compute a new *source*, given the original one and it's *view*, but submitted to a particular change.

In this task, we intend to fully explore this promising research direction. Our plan is to formally establish how circular programs can be integrated within Bidirectional Transformations, namely how circular programs can be used in the backward transformation of BTs.

4.3 Incremental Computation

Incremental computation is about maintaining the input-output relationship of a program, as the input undergoes changes. The changes in the input may be such that one cannot avoid a complete recomputation of the output. However, in many cases, one can reuse results of the previous computation to update the output more efficiently than by performing a complete recomputation from scratch. Obviously, incremental computation is more efficient for cases where changes in the input cause small changes in the output.

The investigation field of incremental computation has proven to be an exciting one, as, over the years, several researchers have studied and proposed techniques to reuse previously computed results, in order to improve efficiency of computer programs. Incremental Computation is, indeed, essential to the implementation of programming environments (Reps and Teitelbaum, 1989; Michiel, 2004) or spreadsheets, for example.

Change Propagation (Reps, 1982), Adaptive Programming (Acar et al., 2002, 2006b,a) and Function Memoization (Pugh and Teitelbaum, 1989; Hughes, 1985) are among the techniques proposed to achieve Incremental Computation.

However, the Change Propagation and Function Memoization approaches do not handle circular programs. Furthermore, Reps' techniques do not handle circular attribute grammars. Thus, the incremental functional implementations derived from incremental attribute grammars will never be circular programs.

Acar's ingenious Adaptive Programming technique is proposed in the strict functional setting ML and its implementation in Haskell (Carlsson, 2002) does not support *lazyness*. Thus, his technique also does not allow to combine incrementality with the circular definitions that may occur in a *lazy* setting.

As for Memoization, Hughes' *lazy* memo-functions are specially suitable to manipulate circular (infinite) structures. However, these circularities are not of the same kind as the ones we want to be able to deal with: we exploit the use of function call results as some of the same call arguments with the purpose of eliminating multiple traversals over data structures. It is still not clear how to memoize such circular function calls.

Our plan in this subject is the development of techniques that make it possible to combine incrementality with circular programming.

4.4 Attribute Grammars

Circular programs and Attribute Grammars are closely related. Indeed, as Johnsson (1987) and Swierstra and Kuiper (Kuiper and Swierstra, 1987) originally showed, circular programs are the natural representation of attribute grammars in a lazy setting (Swierstra and Azero, 1998; de Moor et al., 2000; Saraiva, 1999; Dijkstra, 2005). Furthermore, several *Haskell* and circular based Attriubte Grammar Systems have been developed (Swierstra et al.; Wyk et al., 2006).

In our research, we plan to use circular program strictification techniques, in the sense described in Section 4.1, to derive/design a correct by construction Attribute Grammar System. Furthermore, we plan to incorporate in such a system our developments on Incremental Computation, as described in Section 4.3, in order to obtain incremental attribute evaluators. Indeed, incremental evaluators are an important application of Attribute Grammars after the seminal work of Tom Reps on the Synthesizer Generator system(Reps and Teitelbaum, 1989).

4.5 Spreadsheets

Spreadsheet tools can be viewed as programming environments for non-professional programmers. These so-called "end-user" programmers vastly outnumber professional programmers. In fact, spreadsheets, when viewed as a programming language (PL), are one of the largest PLs and can be characterized as a particularly low-level one: there is no support for abstraction, testing, encapsulation, or structured programming. As a result, numerous studies have shown that existing spreadsheets contain errors at an alarmingly high rate.

Surprisingly, there is little work by the programming language community on the foundations of spreadsheets, being the works (Burnett, 2004; Abraham et al., 2005; Erwig et al., 2005) an exception.

Spreadsheets are the motivating example for incremental evaluation. Indeed, a spreadsheet defines dependencies between different cells. When the user changes the value of one cell, only the ones that are affected by the change are re-evaluated.

Building a spreadsheet system is a complex and hard-working task: the developer has to define/maintain not only a complex dependency graph to determine the re-evaluation order and propagate changes, but also a powerful incremental evaluation engine. Within the style of circular programming the user does not have to handle such dependencies: the lazy machinery infers the dependencies and evaluation order at runtime. Moreover, if we combine circular programming with incremental evaluation, then we have a perfect setting to develop spreadsheets.

4.6 Research Questions

With our work, we plan to answer the following questions:

- how can circular programs be transformed, in a calculational setting, into strict programs, whose execution is not restricted to a lazy execution model? A calculational setting is essential since we want to be able to prove the correctness of the transformation.
- develop a proved correct Attribute Grammar System, once we have the answer to the previous research question. Indeed, by then we will be able to prove that the transformations implemented in such system are correct.
- how to fully explore circular definitions in the implementation of Bidirectional Transformations? In particular, up to which extent can circular programs be used in the implementation of simple and concise backward transformations?
- how can the nice properties of circular programming be combined with incremental programming? Ideally, we intend to develop a framework where we can write circular programs that can be executed incrementally.
- implement a spreadsheet system, using our circular and incremental programming setting. Such a programming environment is indeed the ideal one to avoid the complex tasks of maintaining all the dependencies between all the cells of a spreadsheet and of determining a re-evaluation order to propagate the changes resulting from a change in a cell.

4.6.1 Significance

This proposal is geared towards realizing the full potential of circular programming, by exploring its best properties in the interest of a wide range of programming/research areas. We expect benefits of performance, conciseness, and robustness in areas such as program calculation, bidirectional transformations or spreadsheets, for example.

References

Robin Abraham, Martin Erwig, Steve Kollmansberger, and Ethan Seifert. Visual specifications of correct spreadsheets. In *VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 189–196, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2443-5. doi: http://dx.doi.org/10.1109/VLHCC.2005.70.

Umut A. Acar, Guy E. Blelloch, and Robert Harper. Adaptive functional programming. In *POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 247–259, New York, NY, USA, 2002. ACM.

Umut A. Acar, Guy E. Blelloch, Matthias Blume, and Kanat Tangwongsan. An experimental analysis of self-adjusting computation. In *PLDI '06: Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*, pages 96–107, New York, NY, USA, 2006a. ACM.

Umut A. Acar, Guy E. Blelloch, and Robert Harper. Adaptive functional programming. *ACM Trans. Program. Lang. Syst.*, 28(6):990–1034, 2006b.

F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, 1981. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/319628.319634.

Gilles Barthe, Peter Dybjer, Luis Pinto, and João Saraiva, editors. *Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures*, volume 2395 of *Lecture Notes in Computer Science*, 2002. Springer. ISBN 3-540-44044-5.

Richard S. Bird. Using circular programs to eliminate multiple traversals of data. *Acta Inf*, 21:239–250, 1984.

Margaret Burnett. Spreadsheet Quality. In *Int. workshop on Foundations of Spreadsheet (inivted talk)*, 2004.

Magnus Carlsson. Monads for incremental computing. In *ICFP '02: Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, pages 26–35, New York, NY, USA, 2002. ACM.

Olivier Danvy and Mayer Goldberg. There and back again. In *ICFP '02: Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, pages 230–234, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-487-8. doi: http://doi.acm.org/10.1145/581478.581500.

Umeshwar Dayal and Philip A. Bernstein. On the correct translation of update operations on relational views. *ACM Trans. Database Syst.*, 7(3):381–416, 1982. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/319732.319740.

Oege de Moor, Kevin Backhouse, and S. Doaitse Swierstra. First-class attribute grammars. *Informatica (Slovenia)*, 24(3), 2000. URL `citeseer.ist.psu.edu/demoor00firstclass.html`.

Oege de Moor, Simon Peyton-Jones, and Eric Van Wyk. Aspect-oriented compilers. *Lecture Notes in Computer Science*, 1799, 2000. URL `citeseer.ist.psu.edu/demoor99aspectoriented.html`.

Atze Dijkstra. *Stepping through Haskell.* PhD thesis, Department of Computer Science, Utrecht University, The Netherlands, November 2005.

Atze Dijkstra and Doaitse Swierstra. Typing haskell with an attribute grammar (part i). Technical Report UU-CS-2004-037, Institute of Information and Computing Sciences, Utrecht University, 2004.

Martin Erwig, Robin Abraham, Irene Cooperstein, and Steve Kollmansberger. Automatic generation and maintenance of correct spreadsheets. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 136–145, New York, NY, USA, 2005. ACM. ISBN 1-59593-963-2. doi: http://doi.acm.org/10.1145/1062455.1062494.

João Paulo Fernandes and João Saraiva. Tools and Libraries to Model and Manipulate Circular Programs. In *Proc. of the ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation (PEPM'07)*, pages 102–111. ACM Press, 2007.

João Paulo Fernandes, Alberto Pardo, and João Saraiva. A shortcut fusion rule for circular program calculation. In *Haskell '07: Proceedings of the ACM SIGPLAN Haskell workshop*, pages 95–106, New York, NY, USA, 2007. ACM.

J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In *POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 233–246, New York, NY, USA, 2005. ACM. ISBN 1-58113-830-X. doi: http://doi.acm.org/10.1145/1040305.1040325.

N. Ghani and P. Johann. Short Cut Fusion of Recursive Programs with Computational Effects. In *Symposium on Trends in Functional Programming (TFP 2008)*, 2008.

Andrew Gill, John Launchbury, and Simon L. Peyton Jones. A short cut to deforestation. In *Conference on Functional Programming Languages and Computer Architecture*, pages 223–232, June 1993.

Georg Gottlob, Paolo Paolini, Roberto Zicari, and Roberto Zicari. Properties and update semantics of consistent views. *ACM Trans. Database Syst.*, 13(4):486–524, 1988. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/49346.50068.

Ralf Hinze and Johan Jeuring. Generic Haskell: Practice and theory. In *Summer School on Generic Programming*, 2002. URL `http://www.cs.uu.nl/ johanj/publications/GH.pdf`.

Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. In *PEPM '04: Proceedings of the 2004 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages

178–189, New York, NY, USA, 2004. ACM. ISBN 1-58113-835-0. doi: http://doi.acm.org/10.1145/1014007.1014025.

John Hughes. Lazy memo-functions. In Jean-Pierre Jouannaud, editor, *Functional Programming Languages and Computer Architecture*, volume 201 of *LNCS*, pages 129–146. Springer-Verlag, September 1985.

Thomas Johnsson. Attribute grammars as a functional programming paradigm. In *Functional Programming Languages and Computer Architecture*, pages 154–173, 1987.

Simon L. Peyton Jones, Cordelia V. Hall, Kevin Hammond, Will Partain, and Philip Wadler. The glasgow haskell compiler: a technical overview. In *Proc. UK Joint Framework for Information Technology (JFIT) Technical Conference*, 1993. URL `http://citeseer.ist.psu.edu/jones92glasgow.html`.

Matthijs Kuiper and Doaitse Swierstra. Using attribute grammars to derive efficient functional programs. In *Computing Science in the Netherlands CSN'87*, November 1987.

Ralf Lämmel, João Saraiva, and Joost Visser, editors. *Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. Revised Papers*, volume 4143 of *Lecture Notes in Computer Science*, 2006. Springer. ISBN 3-540-45778-X.

Julia L. Lawall. Implementing Circularity Using Partial Evaluation. In *Proceedings of the Second Symposium on Programs as Data Objects PADO II*, volume 2053 of *LNCS*. Springer-Verlag, May 2001.

Jens Lechtenbörger and Gottfried Vossen. On the computation of relational view complements. *ACM Trans. Database Syst.*, 28(2):175–208, 2003. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/777943.777946.

Xavier Leroy. The Objective Caml System - Documentation and User's Manual, 1997.

C. Manzino and A. Pardo. Short Cut Fusion of Monadic Programs. In *Brazilian Symposium on Programming Languages (SBLP 2008)*, 2008.

Simon Marlow and Simon Peyton Jones. The new GHC/Hugs Runtime System. URL `http://research.microsoft.com/Users/simonpj/Papers/new-rts.ps.gz`. 1999.

Lambert Meertens. Designing constraint maintainers for user interaction. http://www.cwi.nl/∼lambert, 1998.

Martijn Michiel. *Proxima : a presentation-oriented editor for structured documents*. PhD thesis, Department of Computer Science, Utrecht University, The Netherlands, 2004.

Atsushi Ohori and Isao Sasano. Lightweight fusion by fixed point promotion. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 143–154, New York, NY, USA, 2007. ACM Press.

Chris Okasaki. Breadth-first numbering: lessons from a small exercise in algorithm design. *ACM SIGPLAN Notices*, 35(9):131–136, 2000.

Y. Onoue, Z. Hu, H. Iwasaki, and M. Takeichi. A Calculational Fusion System HYLO. In *IFIP TC 2 Working Conference on Algorithmic Languages and Calculi, Le Bischenberg, France*, pages 76–106. Chapman & Hall, February 1997.

Alberto Pardo, João Paulo Fernandes, and João Saraiva. Calculating Monadic Circular Programs, 2009. Submitted to ACM Symposium on Applied Computing.

W. Pugh and T. Teitelbaum. Incremental computation via function caching. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 315–328, New York, NY, USA, 1989. ACM.

T. Reps and T. Teitelbaum. *The Synthesizer Generator*. Springer, 1989.

Thomas Reps. Optimal-time incremental semantic analysis for syntax-directed editors. In *POPL '82: Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 169–176, New York, NY, USA, 1982. ACM.

João Saraiva. *Purely Functional Implementation of Attribute Grammars*. PhD thesis, Department of Computer Science, Utrecht University, The Netherlands, December 1999.

Doaitse Swierstra, Pablo Azero, and João Saraiva. Designing and Implementing Combinator Languages. In Doaitse Swierstra, Pedro Henriques, and José Oliveira, editors, *Third Summer School on Advanced Functional Programming*, volume 1608 of *LNCS Tutorial*, pages 150–206. Springer-Verlag, September 1999a.

S. D. Swierstra, Arthur Baars, and Andres Löh. The UU-AG attribute grammar system. `http://www.cs.uu.nl/groups/ST`.

S. Doaitse Swierstra and Pablo Azero. Attribute grammars in a functional style. In *Systems Implementation 2000*, Berlin, 1998. Chapman & Hall.

S. Doaitse Swierstra, Pedro Rangel Henriques, and José N. Oliveira, editors. *Advanced Functional Programming, Third International School, Braga, Portugal, September 12-19, 1998, Revised Lectures*, volume 1608 of *Lecture Notes in Computer Science*, 1999b. Springer. ISBN 3-540-66241-3.

Janis Voigtländer. Using circular programs to deforest in accumulating parameters. *Higher-Order and Symbolic Computation*, 17:129–163, 2004. Previous version appeared in *ASIA-PEPM 2002*, Proceedings, pages 126–137, ACM Press, 2002.

P. Wadler. Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.

Eric Van Wyk, Lijesh Krishnan, Derek Bodin, Eric Johnson, August Schwerdfeger, and Phil Russell. Tool demonstration: Silver extensible compiler frameworks and modular language extensions for java and c. In *SCAM*, page 161, 2006.