Pointer Analysis

Yannis Smaragdakis
University of Athens
smaragd@di.uoa.gr

George Balatsouras
University of Athens
gbalats@di.uoa.gr

now
the essence of knowledge
Boston — Delft

Foundations and Trends® in Programming Languages

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

Y. Smaragdakis and G. Balatsouras. *Pointer Analysis*. Foundations and Trends® in Programming Languages, vol. 2, no. 1, pp. 1–69, 2015.

This Foundations and Trends® issue was typeset in LaTeX using a class file designed by Neal Parikh. Printed on acid-free paper.

ISBN: 978-1-68083-020-0

Foundations and Trends® in Programming Languages
Volume 2, Issue 1, 2015

Editorial Scope

Topics

Foundations and Trends® in Programming Languages publishes survey and tutorial articles in the following topics:

- Abstract interpretation
- Compilation and interpretation techniques
- Domain specific languages
- Formal semantics, including lambda calculi, process calculi, and process algebra
- Language paradigms
- Mechanical proof checking
- Memory management
- Partial evaluation
- Program logic
- Programming language implementation
- Programming language security
- Programming languages for concurrency
- Programming languages for parallelism
- Program synthesis
- Program transformations and optimizations
- Program verification
- Runtime techniques for programming languages
- Software model checking
- Static and dynamic program analysis
- Type theory and type systems

Information for Librarians

Foundations and Trends® in Programming Languages, 2015, Volume 2, 4 issues. ISSN paper version 2325-1107. ISSN online version 2325-1131. Also available as a combined paper and online subscription.

Foundations and Trends® in Programming Languages
Vol. 2, No. 1 (2015) 1–69

DOI: 10.1561/2500000014

Pointer Analysis

Yannis Smaragdakis
University of Athens
smaragd@di.uoa.gr

George Balatsouras
University of Athens
gbalats@di.uoa.gr

Contents

Abstract

Pointer analysis is a fundamental static program analysis, with a rich literature and wide applications. The goal of pointer analysis is to compute an approximation of the set of program objects that a pointer variable or expression can refer to.

We present an introduction and survey of pointer analysis techniques, with an emphasis on distilling the essence of common analysis algorithms. To this end, we focus on a declarative presentation of a common core of pointer analyses: algorithms are modeled as configurable, yet easy-to-follow, logical specifications. The specifications serve as a starting point for a broader discussion of the literature, as independent threads spun from the declarative model.

Y. Smaragdakis and G. Balatsouras. *Pointer Analysis*. Foundations and Trends® in Programming Languages, vol. 2, no. 1, pp. 1–69, 2015.

DOI: 10.1561/2500000014.

1

Introduction

Pointer analysis or *points-to analysis* is a static program analysis that determines information on the values of pointer variables or expressions. Such information offers a static model of a program's heap. Since the heap is the primary structure for global program data, pointer analysis forms the substrate of most inter-procedural static analyses. Virtually all interesting questions one may want to ask of a program will eventually need to query the possible values of a pointer expression, or its relationship to other pointer expressions. The exact representation of such information, i.e., the static abstraction used to model the heap, often serves as a classifier of the analysis algorithm. Although the literature is not entirely consistent on high-level terminology, pointer analysis is a near-synonym of *alias analysis*. Whereas, however, pointer/points-to analysis typically tries to model heap objects and asks "what objects can a variable point to?", alias analysis algorithms focus on the closely related question of "can a pair of variables or expressions be aliases, i.e., point to the same object?" [Landi and Ryder, 1992, Emami et al., 1994]

In this monograph, we attempt to survey the most common modern approaches to pointer analysis, with an eye towards ease of exposition and concreteness. Our presentation aspires to be rather more tutorial and hands-on than other surveys of the pointer analysis area. For a thorough view of the literature, with an emphasis on coverage, readers can consult Hind [2001], Ryder [2003], Sridharan et al. [2013], and Kanvar and Khedker [2014].

Our tutorial will develop, in significant detail, a modular, configurable model of a standard points-to analysis. This analysis model is a skeleton on which we progressively add more flesh, to reflect several realistic features and analysis enhancements from the recent literature. The analysis model will also serve as a firm basis for high-level tangential discussions on topics that deviate from the model: alias analysis, complexity theory results, algorithms not captured well by the formal model, and more.

Importantly, our analysis model is executable: The specification of our algorithms is given in Datalog, which is simultaneously a logic and a realistic programming language. The use of Datalog allows us to express the *precision* aspects of pointer analyses concisely, at almost the same high level as a mathematical formalism, yet with no need to separately treat the topic of how to implement the algorithms so that they perform *efficiently*.

The axes of *precision* and *performance/efficiency* characterize every approach to program reasoning. All interesting questions about universal (i.e., all-inputs) program behavior are undecidable—see Landi [1992], Ramalingam [1994], Reps [2000] specifically for pointer analysis problems. Thus, every technique is evaluated both on its precision, i.e., the degree to which the result approximates the uncomputable mathematical ideal, and on its performance, i.e., the asymptotic complexity or practical speed of computation.

We can use these axes to guide a more general overview of the landscape of techniques for reasoning about program memory, of which pointer analysis is only one part. Further along the precision axis lie several approaches such as *shape analysis* [Sagiv et al., 2002] and *separation logic* [Reynolds, 2002, O'Hearn et al., 2001]. Separation logic is a full-fledged logic, typically deep in the forests of undecidability, where reasoning requires close human guidance. Shape analysis is a computable, automated program analysis, yet with performance complexity in the territory of intractability: complexity bounds for the best-known shape analyses are super-exponential.

In contrast, the term "pointer analysis" is typically reserved for techniques of modest performance cost, scaling to realistic, automated whole-program analysis efforts. This bias in favor of automation and scalability to full realistic programs is also reflected throughout our discussion and analysis formulations. Datalog (with the standard enhancement of an order relation) is a language that captures the $\mathcal{PTIME}$ complexity class [Immerman, 1999, Ch.14]: every Datalog program runs in polynomial time, and every polynomial algorithm can be written in Datalog.

Although a polynomial complexity bound is hardly a guarantee of practical scalability, in broad mathematical strokes it is a reliable distingushing feature. Indeed, it is tempting to consider "pointer analysis" to refer precisely to heap analysis algorithms of polynomial complexity. In our formulation of analyses, we strive to maintain this complexity boundary. This is reflected in our effort to stay within standard Datalog (with stratified negation) and only use extensions as syntactic sugar.

We begin with essential background and an illustration of the best known pointer analysis algorithms.

References

Ole Agesen. The cartesian product algorithm: Simple and precise type inference of parametric polymorphism. In *Proc. of the 9th European Conf. on Object-Oriented Programming*, ECOOP '95, pages 2–26. Springer, 1995. ISBN 3-540-60160-0.

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321486811.

Lars O. Andersen. *Program Analysis and Specialization for the C Programming Language*. PhD thesis, DIKU, University of Copenhagen, May 1994.

David F. Bacon and Peter F. Sweeney. Fast static analysis of C++ virtual function calls. In *Proc. of the 11th Annual ACM SIGPLAN Conf. on Object Oriented Programming, Systems, Languages, and Applications*, OOPSLA '96, pages 324–341, New York, NY, USA, 1996. ACM. ISBN 0-89791-788-X.

Gogul Balakrishnan and Thomas W. Reps. Recency-abstraction for heap-allocated storage. In *Proc. of the 14th International Symp. on Static Analysis*, SAS '06, pages 221–239. Springer, 2006.

Marc Berndl, Ondrej Lhoták, Feng Qian, Laurie J. Hendren, and Navindra Umanee. Points-to analysis using BDDs. In *Proc. of the 2003 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '03, pages 103–114, New York, NY, USA, 2003. ACM. ISBN 1-58113-662-5.

Martin Bravenboer and Yannis Smaragdakis. Strictly declarative specification of sophisticated points-to analyses. In *Proc. of the 24th Annual ACM SIGPLAN Conf. on Object Oriented Programming, Systems, Languages, and Applications*, OOPSLA '09, New York, NY, USA, 2009a. ACM. ISBN 978-1-60558-766-0.

Martin Bravenboer and Yannis Smaragdakis. Exception analysis and points-to analysis: Better together. In *Proc. of the 18th International Symp. on Software Testing and Analysis*, ISSTA '09, pages 1–12, New York, NY, USA, 2009b. ACM. ISBN 978-1-60558-338-9.

Ramkrishna Chatterjee, Barbara G. Ryder, and William A. Landi. Relevant context inference. In *Proc. of the 26th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, POPL '99, pages 133–146. ACM, 1999. ISBN 1-58113-095-3.

Ben-Chung Cheng and Wen-Mei W. Hwu. Modular interprocedural pointer analysis using access paths: Design, implementation, and evaluation. In *Proc. of the 2000 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '00, pages 57–69, New York, NY, USA, 2000. ACM. ISBN 1-58113-199-2.

Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 269–282, 1979.

Jeffrey Dean, David Grove, and Craig Chambers. Optimization of object-oriented programs using static class hierarchy analysis. In *Proc. of the 9th European Conf. on Object-Oriented Programming*, ECOOP '95, pages 77–101. Springer, 1995. ISBN 3-540-60160-0.

Isil Dillig, Thomas Dillig, and Alex Aiken. Sound, complete and scalable path-sensitive analysis. In *Proc. of the 2008 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '08, pages 270–280, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-860-2.

Michael Eichberg, Sven Kloppenburg, Karl Klose, and Mira Mezini. Defining and continuous checking of structural program dependencies. In *Proc. of the 30th International Conf. on Software Engineering*, ICSE '08, pages 391–400, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-079-1.

Maryam Emami, Rakesh Ghiya, and Laurie J. Hendren. Context-sensitive interprocedural points-to analysis in the presence of function pointers. In *Proc. of the 1994 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '94, pages 242–256, New York, NY, USA, 1994. ACM. ISBN 0-89791-662-X.

Manuel Fähndrich, Jeffrey S. Foster, Zhendong Su, and Alexander Aiken. Partial online cycle elimination in inclusion constraint graphs. In *Proc. of the 1998 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '98, pages 85–96, New York, NY, USA, 1998. ACM. ISBN 0-89791-987-4.

Chen Fu and Barbara G. Ryder. Exception-chain analysis: Revealing exception handling architecture in Java server applications. In *Proc. of the 29th International Conf. on Software Engineering*, ICSE '07, pages 230–239, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2828-7.

Samuel Z. Guyer and Calvin Lin. Client-driven pointer analysis. In *Proc. of the 10th International Symp. on Static Analysis*, SAS '03, pages 214–236. Springer, 2003. ISBN 3-540-40325-6.

Elnar Hajiyev, Mathieu Verbaere, and Oege de Moor. CodeQuest: Scalable source code queries with Datalog. In *Proc. of the 20th European Conf. on Object-Oriented Programming*, ECOOP '06, pages 2–27. Springer, 2006. ISBN 978-3-540-35726-2.

Ben Hardekopf and Calvin Lin. The ant and the grasshopper: fast and accurate pointer analysis for millions of lines of code. In *Proc. of the 2007 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '07, pages 290–299, New York, NY, USA, 2007a. ACM. ISBN 978-1-59593-633-2.

Ben Hardekopf and Calvin Lin. Exploiting pointer and location equivalence to optimize pointer analysis. In *Proc. of the 14th International Symp. on Static Analysis*, SAS '07, pages 265–280. Springer, 2007b. ISBN 978-3-540-74060-5.

Nevin Heintze and Olivier Tardieu. Ultra-fast aliasing analysis using CLA: A million lines of C code in a second. In *Proc. of the 2001 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '01, pages 254–263, New York, NY, USA, 2001a. ACM. ISBN 1-58113-414-2.

Nevin Heintze and Olivier Tardieu. Demand-driven pointer analysis. In *Proc. of the 2001 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '01, pages 24–34, New York, NY, USA, 2001b. ACM. ISBN 1-58113-414-2.

Michael Hind. Pointer analysis: haven't we solved this problem yet? In *Proc. of the 3rd ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '01, pages 54–61, New York, NY, USA, 2001. ACM. ISBN 1-58113-413-4.

Susan Horwitz. Precise flow-insensitive may-alias analysis is NP-hard. *ACM Trans. Program. Lang. Syst.*, 19(1):1–6, January 1997. ISSN 0164-0925.

Neil Immerman. *Descriptive Complexity*. Graduate texts in computer science. Springer, 1999. ISBN 978-1-4612-6809-3.

Vini Kanvar and Uday P. Khedker. Heap abstractions for static analysis. *CoRR*, abs/1403.4910, 2014. URL `http://arxiv.org/abs/1403.4910`.

George Kastrinis and Yannis Smaragdakis. Efficient and effective handling of exceptions in Java points-to analysis. In *Proc. of the 22nd International Conf. on Compiler Construction*, CC '13, pages 41–60. Springer, 2013a. ISBN 978-3-642-37050-2.

George Kastrinis and Yannis Smaragdakis. Hybrid context-sensitivity for points-to analysis. In *Proc. of the 2013 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '13, New York, NY, USA, 2013b. ACM. ISBN 978-1-4503-2014-6.

Monica S. Lam, John Whaley, V. Benjamin Livshits, Michael C. Martin, Dzintars Avots, Michael Carbin, and Christopher Unkel. Context-sensitive program analysis as database queries. In *Proc. of the 24th Symp. on Principles of Database Systems*, PODS '05, pages 1–12, New York, NY, USA, 2005. ACM. ISBN 1-59593-062-0.

William Landi. Undecidability of static analysis. *LOPLAS*, 1(4):323–337, 1992.

William Landi and Barbara G. Ryder. A safe approximate algorithm for interprocedural aliasing. In *Proc. of the 1992 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '92, pages 235–248, New York, NY, USA, 1992. ACM. ISBN 0-89791-475-9.

Ondřej Lhoták. *Program Analysis using Binary Decision Diagrams*. PhD thesis, McGill University, January 2006.

Ondřej Lhoták and Laurie J. Hendren. Evaluating the benefits of context-sensitive points-to analysis using a BDD-based implementation. *ACM Trans. Softw. Eng. Methodol.*, 18(1):1–53, 2008. ISSN 1049-331X.

Yue Li, Tian Tan, Yulei Sui, and Jingling Xue. Self-inferencing reflection resolution for Java. In *Proc. of the 28th European Conf. on Object-Oriented Programming*, ECOOP '14, pages 27–53. Springer, 2014. ISBN 978-3-662-44201-2.

Percy Liang and Mayur Naik. Scaling abstraction refinement via pruning. In *Proc. of the 2011 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '11, pages 590–601, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0663-8.

Benjamin Livshits. *Improving Software Security with Precise Static and Runtime Analysis*. PhD thesis, Stanford University, December 2006.

Benjamin Livshits, John Whaley, and Monica S. Lam. Reflection analysis for Java. In *Proc. of the 3rd Asian Symp. on Programming Languages and Systems*, APLAS '05, pages 139–160. Springer, 2005. ISBN 3-540-29735-9.

Benjamin Livshits, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J. Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z. Guyer, Uday P. Khedker, Anders Møller, and Dimitrios Vardoulakis. In defense of soundiness: A manifesto. *Commun. ACM*, 58(2):44–46, January 2015. ISSN 0001-0782. . URL `http://doi.acm.org/10.1145/2644805`.

Magnus Madsen, Benjamin Livshits, and Michael Fanning. Practical static analysis of JavaScript applications in the presence of frameworks and libraries. In *Proc. of the ACM SIGSOFT International Symp. on the Foundations of Software Engineering*, FSE '13, pages 499–509. ACM, 2013. ISBN 978-1-4503-2237-9.

Matthew Might, Yannis Smaragdakis, and David Van Horn. Resolving and exploiting the k-CFA paradox: Illuminating functional vs. object-oriented program analysis. In *Proc. of the 2010 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '10, pages 305–315, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0019-3.

Ana Milanova, Atanas Rountev, and Barbara G. Ryder. Parameterized object sensitivity for points-to and side-effect analyses for Java. In *Proc. of the 2002 International Symp. on Software Testing and Analysis*, ISSTA '02, pages 1–11, New York, NY, USA, 2002. ACM. ISBN 1-58113-562-9.

Ana Milanova, Atanas Rountev, and Barbara G. Ryder. Parameterized object sensitivity for points-to analysis for Java. *ACM Trans. Softw. Eng. Methodol.*, 14(1):1–41, 2005. ISSN 1049-331X.

Mayur Naik, Alex Aiken, and John Whaley. Effective static race detection for Java. In *Proc. of the 2006 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '06, pages 308–319, New York, NY, USA, 2006. ACM. ISBN 1-59593-320-4.

Mayur Naik, Chang-Seo Park, Koushik Sen, and David Gay. Effective static deadlock detection. In *Proc. of the 31st International Conf. on Software Engineering*, ICSE '09, pages 386–396, New York, NY, USA, 2009. ACM. ISBN 978-1-4244-3452-7.

Rupesh Nasre. Exploiting the structure of the constraint graph for efficient points-to analysis. In *Proc. of the 2012 International Symp. on Memory Management*, ISMM '12, pages 121–132. ACM, 2012. ISBN 978-1-4503-1350-6.

Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proc. of the 15th International Workshop on Computer Science Logic*, volume 2142 of *CSL '01*, pages 1–19. Springer, 2001. ISBN 3-540-42554-3.

Ganesan Ramalingam. The undecidability of aliasing. *ACM Trans. Program. Lang. Syst.*, 16(5):1467–1471, 1994.

Thomas W. Reps. Demand interprocedural program analysis using logic databases. In R. Ramakrishnan, editor, *Applications of Logic Databases*, pages 163–196. Kluwer Academic Publishers, 1994.

Thomas W. Reps. Program analysis via graph reachability. *Information & Software Technology*, 40:701–726, 1998.

Thomas W. Reps. Undecidability of context-sensitive data-independence analysis. *ACM Trans. Program. Lang. Syst.*, 22(1):162–186, 2000.

Thomas W. Reps, Susan Horwitz, and Shmuel Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proc. of the 22nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, POPL '95, pages 49–61, New York, NY, USA, 1995. ACM. ISBN 0-89791-692-1.

John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of the 17th IEEE Symp. on Logic in Computer Science*, LICS '02, pages 55–74. IEEE Computer Society, 2002. ISBN 0-7695-1483-9.

Noam Rinetzky, Ganesan Ramalingam, Shmuel Sagiv, and Eran Yahav. On the complexity of partially-flow-sensitive alias analysis. *ACM Trans. Program. Lang. Syst.*, 30(3), 2008. .

Atanas Rountev and Satish Chandra. Off-line variable substitution for scaling points-to analysis. In *Proc. of the 2000 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '00, pages 47–56, New York, NY, USA, 2000. ACM. ISBN 1-58113-199-2.

Barbara G. Ryder. Dimensions of precision in reference analysis of object-oriented programming languages. In *Proc. of the 12th International Conf. on Compiler Construction*, CC '03, pages 126–137. Springer, 2003. ISBN 3-540-00904-3.

Mooly Sagiv, Thomas W. Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, May 2002. ISSN 0164-0925.

Micha Sharir and Amir Pnueli. Two approaches to interprocedural data flow analysis. In Steven S. Muchnick and Neil D. Jones, editors, *Program flow analysis: theory and applications*, chapter 7, pages 189–233. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981. ISBN 0137296819.

Olin Shivers. *Control-Flow Analysis of Higher-Order Languages*. PhD thesis, Carnegie Mellon University, may 1991.

Yannis Smaragdakis, Martin Bravenboer, and Ondřej Lhoták. Pick your contexts well: Understanding object-sensitivity. In *Proc. of the 38th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, POPL '11, pages 17–30, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0490-0.

Yannis Smaragdakis, George Balatsouras, and George Kastrinis. Set-based pre-processing for points-to analysis. In *Proc. of the 28th Annual ACM SIGPLAN Conf. on Object Oriented Programming, Systems, Languages, and Applications*, OOPSLA '13, pages 253–270, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2374-1.

Yannis Smaragdakis, George Kastrinis, and George Balatsouras. Introspective analysis: Context-sensitivity, across the board. In *Proc. of the 2014 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '14, pages 485–495, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2784-8.

Manu Sridharan and Rastislav Bodík. Refinement-based context-sensitive points-to analysis for Java. In *Proc. of the 2006 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '06, pages 387–400, New York, NY, USA, 2006. ACM. ISBN 1-59593-320-4.

Manu Sridharan, Denis Gopan, Lexin Shan, and Rastislav Bodík. Demand-driven points-to analysis for Java. In *Proc. of the 20th Annual ACM SIGPLAN Conf. on Object Oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 59–76, New York, NY, USA, 2005. ACM. ISBN 1-59593-031-0.

Manu Sridharan, Satish Chandra, Julian Dolby, Stephen J. Fink, and Eran Yahav. Alias analysis for object-oriented programs. In Dave Clarke, James Noble, and Tobias Wrigstad, editors, *Aliasing in Object-Oriented Programming. Types, Analysis and Verification*, volume 7850 of *Lecture Notes in Computer Science*, pages 196–232. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36945-2.

Bjarne Steensgaard. Points-to analysis in almost linear time. In *Proc. of the 23rd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, POPL '96, pages 32–41, New York, NY, USA, 1996. ACM. ISBN 0-89791-769-3.

Frank Tip and Jens Palsberg. Scalable propagation-based call graph construction algorithms. In *Proc. of the 15th Annual ACM SIGPLAN Conf. on Object Oriented Programming, Systems, Languages, and Applications*, OOPSLA '00, pages 281–293, New York, NY, USA, 2000. ACM. ISBN 1-58113-200-X.

David Van Horn and Harry G. Mairson. Deciding *k*CFA is complete for EXPTIME. In *Proc. of the 13th ACM SIGPLAN International Conference on Functional programming*, ICFP '08, pages 275–282. ACM, 2008. ISBN 978-1-59593-919-7.

John Whaley and Monica S. Lam. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams. In *Proc. of the 2004 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '04, pages 131–144, New York, NY, USA, 2004. ACM. ISBN 1-58113-807-5.

John Whaley, Dzintars Avots, Michael Carbin, and Monica S. Lam. Using Datalog with binary decision diagrams for program analysis. In *Proc. of the 3rd Asian Symp. on Programming Languages and Systems*, APLAS '05, pages 97–118. Springer, 2005. ISBN 3-540-29735-9.

Robert P. Wilson and Monica S. Lam. Efficient context-sensitive pointer analysis for C programs. In *Proc. of the 1995 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '95, pages 1–12, New York, NY, USA, 1995. ACM. ISBN 0-8989791-697-2.

Qirun Zhang, Michael R. Lyu, Hao Yuan, and Zhendong Su. Fast algorithms for Dyck-CFL-reachability with applications to alias analysis. In *Proc. of the 2013 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '13, pages 435–446, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2014-6.

Xin Zheng and Radu Rugina. Demand-driven alias analysis for C. In *Proc. of the 35th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, POPL '08, pages 197–208, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-689-9.