References

[1] O. Agesen, D. Detlefs, and J. E. Moss. Garbage collection and local variable type-precision and liveness in Java virtual machines. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 269–279. ACM, 1998.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[3] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2/e)*. Addison-Wesley Longman Publishing Co., Inc., 2006.

[4] F. E. Allen. Control flow analysis. *ACM SIGPLAN Notices*, 5(7):1–19, 1970.

[5] F. E. Allen. A basis for program optimization. In *IFIP Congress (1)*, pages 385–390. North Holland Publishing Company, 1971.

[6] F. E. Allen. Interprocedural data flow analysis. In *Proceedings of IFIP Congress 74*, pages 398–408. North Holland Publishing Company, 1974.

[7] F. E. Allen and J. Cocke. A program data flow analysis procedure. *Communications of the ACM*, 19(3):137–147, 1976.

[8] B. Alpern, M. N. Wegman, and F. K. Zadeck. Detecting equality of variables in programs. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 1–11. ACM, 1988.

[9] L. O. Andersen. *Program Analysis and Specialization for the C Programming Language*. PhD thesis, DIKU, University of Copenhagen, 1994.

[10] A. W. Appel and M. Ginsburg. *Modern Compiler Implementation in C*. Cambridge University Press, 1998.

[11] Andrew W. Appel. SSA is functional programming. *ACM SIGPLAN Notices*, 33(4):17–20, 1998.

[12] J. Banning. An efficient way to find the side effects of procedure calls and aliases of variables. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 29–41. ACM, 1979.

[13] J. M. Barth. An interprocedural data flow analysis algorithm. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 119–131. ACM, 1977.

[14] Jeffrey M. Barth. A practical interprocedural data flow analysis algorithm. *Communications of the ACM*, 21(9):724–736, 1978.

[15] B. Blanchet. Escape analysis for object-oriented languages: application to Java. In *Proceedings of the ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, pages 20–34. ACM, 1999.

[16] B. Blanchet. Escape analysis for Java^{TM}: Theory and practice. *ACM Transactions on Programming Languages and Systems*, 25(6):713–775, 2003.

[17] R. Bodik, R. Gupta, and M. L. Soffa. Complete removal of redundant computations. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1–14. ACM, 1998.

[18] P. Briggs, K. D. Cooper, T. J. Harvey, and L. T. Simpson. Practical improvements to the construction and destruction of static single assignment form. *Software—Practice and Experience*, 28(8):859–881, 1998.

[19] D. Callahan. The program summary graph and flow-sensitive interprocedural data flow analysis. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 47–56. ACM, 1988.

[20] D. Callahan, A. Carle, M. W. Hall, and K. Kennedy. Constructing the procedure call multigraph. *IEEE Transactions on Software Engineering*, 16(4):483–487, 1990.

[21] J. Choi, M. Burke, and P. Carini. Efficient flow-sensitive interprocedural computation of pointer-induced aliases and side effects. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 232–245. ACM, 1993.

[22] J. Choi, R. Cytron, and J. Ferrante. Automatic construction of sparse data flow evaluation graphs. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 55–66. ACM, 1991.

[23] J. Choi, M. Gupta, M. Serrano, V. C. Sreedhar, and S. Midkiff. Escape analysis for Java. In *Proceedings of the ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, pages 1–19. ACM, 1999.

[24] J. Cocke. Global common subexpression elimination. *ACM SIGPLAN Notices*, 5(7):20–24, 1970.

[25] K. Cooper. Analyzing aliases of reference formal parameters. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 281–290. ACM, 1985.

[26] K. D. Cooper and K. Kennedy. Fast interprocedural alias analysis. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 49–59. ACM, 1989.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, (2/e)*. The MIT Press and McGraw-Hill Book Company, 2001.

[28] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, 1991.

[29] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order (2/e)*. Cambridge University Press, 2002.

[30] D. M. Dhamdhere and U. P. Khedker. Complexity of bidirectional data flow analysis. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 397–408. ACM, 1993.

[31] D. M. Dhamdhere, B. K. Rosen, and F. K. Zadeck. How to analyze large programs efficiently and informatively. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 212–223. ACM, 1992.

[32] E. Duesterwald, R. Gupta, and M. L. Soffa. Demand-driven computation of interprocedural data flow. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 37–48. ACM, 1995.

[33] E. Duesterwald, R. Gupta, and M. L. Soffa. A practical framework for demand-driven interprocedural data flow analysis. *ACM Transactions on Programming Languages and Systems*, 19(6):992–1030, 1997.

[34] M. Emami, R. Ghiya, and L. J. Hendren. Context-sensitive interprocedural points-to analysis in the presence of function pointers. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 242–256. ACM, 1994.

[35] M. Fähndrich, J. S. Foster, Z. Su, and A. Aiken. Partial online cycle elimination in inclusion constraint graphs. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 85–96. ACM, 1998.

[36] M. Fähndrich, J. Rehof, and M. Das. Scalable context-sensitive flow analysis using instantiation constraints. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 253–263. ACM, 2000.

[37] S. Graham and M. Wegman. A fast and usually linear algorithm for global data flow analysis. *Journal of ACM*, 23(1):172–202, 1976.

[38] D. Grove and C. Chambers. A framework for call graph construction algorithms. *ACM Transactions on Programming Languages and Systems*, 23(6):685–746, 2001.

[39] D. Grove and L. Torczon. Interprocedural constant propagation: a study of jump function implementation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementations*, pages 90–99. ACM, 1993.

[40] D. Grune, H. E. Bal, C. J. H. Jacobs, and K. G. Langendoen. *Modern Compiler Design*. John Wiley & Sons, 2000.

[41] S. Hack. *Register Allocation for Programs in SSA Form*. PhD thesis, Universität Karlsruhe, 2007.

[42] S. Hack, D. Grund, and G. Goos. Register allocation for programs in SSA-form. In *Proceedings of the International Conference on Compiler Construction*, pages 247–262. Springer-Verlag, 2006.

[43] M. W. Hall and K. Kennedy. Efficient call graph analysis. *ACM Letters on Programming Languages and Systems*, 1(3):227–242, 1992.

[44] M. S. Hecht. *Flow Analysis of Computer Programs*. Elsevier North-Holland Inc., 1977.

[45] M. S. Hecht and J. D. Ullman. Flow graph reducibility. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 238–250. ACM, 1972.

[46] M. S. Hecht and J. D. Ullman. Characterization of reducible flow graphs. *Journal of ACM*, 21(3):367–375, 1974.

[47] M. Hind. Pointer analysis: haven't we solved this problem yet? In *Proceedings of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, pages 54–61. ACM, 2001.

[48] M. Hind, M. Burke, P. Carini, and J. Choi. Interprocedural pointer alias analysis. *ACM Transactions on Programming Languages and Systems*, 21(4):848–894, 1999.

[49] J. B. Kam and J. D. Ullman. Global data flow analysis and iterative algorithms. *Journal of ACM*, 23(1):158–171, 1976.

[50] J. B. Kam and J. D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7(3):305–317, 1977.

[51] A. Kanade, U. P. Khedker, and A. Sanyal. Heterogeneous fixed points with application to points-to analysis. In *Proceedings of the Asian Symposium on Programming Languages and Systems*, pages 298–314. Springer-Verlag, 2005.

[52] A. Karkare, A. Sanyal, and U. P. Khedker. Effectiveness of garbage collection in MIT/GNU Scheme. *CoRR*, abs/cs/0611093, 2006.

[53] B. Karkare. *Complexity and Efficiency Issues in Data Flow Analysis*. PhD thesis, Indian Institute of Technology, Bombay, 2007.

[54] B. Karkare and U. P. Khedker. An improved bound for call-strings based interprocedural analysis of bit vector frameworks. *ACM Transactions on Programming Languages and Systems*, 29(6):38, 2007.

[55] K. Kennedy. A global flow analysis algorithm. *International Journal of Computer Mathematic*, 3(1):5–15, 1971.

[56] K. Kennedy. Node listings applied to data flow analysis. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 10–21. ACM, 1975.

[57] K. Kennedy. A survey of data flow analysis techniques, 1981. In [77].

[58] R. Kennedy, S. Chan, S. Liu, R. Lo, P. Tu, and F. Chow. Partial redundancy elimination in SSA form. *ACM Transactions on Programming Languages and Systems*, 21(3):627–676, 1999.

[59] U. P. Khedker. Data flow analysis. In Y. N. Srikant and Priti Shankar, editors, *The Compiler Design Handbook: Optimizations & Machine Code Generation*. CRC Press, 2002.

[60] U. P. Khedker and D. M. Dhamdhere. A generalized theory of bit vector data flow analysis. *ACM Transactions on Programming Languages and Systems*, 16(5):1472–1511, 1994.

[61] U. P. Khedker and B. Karkare. Efficiency, precision, simplicity, and generality in interprocedural data flow analysis: Resurrecting the classical call strings method. In *Proceedings of the International Conference on Compiler Construction*, pages 213–228. Springer-Verlag, 2008.

[62] U. P. Khedker, A. Sanyal, and A. Karkare. Heap reference analysis using access graphs. *ACM Transactions on Programming Languages and Systems*, 30(1):1, 2007.

[63] G. Kildall. A unified approach to global program optimization. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 194–206. ACM, 1973.

[64] K. Knobe and V. Sarkar. Array SSA form and its use in parallelization. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 107–120. ACM, 1998.

[65] J. Knoop, O. Rüthing, and B. Steffen. Lazy code motion. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 224–234. ACM, 1992.

[66] W. Landi and B. G. Ryder. A safe approximate algorithm for interprocedural pointer aliasing. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 235–248. ACM, 1992.

[67] W. Landi, B. G. Ryder, and S. Zhang. Interprocedural side effect analysis with pointer aliasing. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 56–67. ACM, 1993.

[68] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–141, 1979.

[69] O. Lhoták and L. Hendren. Context-sensitive points-to analysis: is it worth it? In *Proceedings of the International Conference on Compiler Construction*, pages 47–64. Springer-Verlag, 2006.

[70] E. S. Lowry and C. W. Medlock. Object code optimization. *Communications of the ACM*, 12(1):13–22, 1969.

[71] T. J. Marlowe and B. G. Ryder. Properties of data flow frameworks. *Acta Informatica*, 28(2):121–163, 1990.

[72] F. Martin. Experimental comparison of call string and functional approaches to interprocedural analysis. In *Proceedings of the International Conference on Compiler Construction*, pages 63–75. Springer-Verlag, 1999.

[73] C. E. McDowell. Reducing garbage in java. *ACM SIGPLAN Notices*, 33(9):84–86, 1998.

[74] E. Morel and C. Renvoise. Global optimization by suppression of partial redundancies. *Communications of the ACM*, 22(2):96–103, 1979.

[75] R. Morgan. *Building an Optimizing Compiler*. Digital Press, 1998.

[76] S. S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishing Co., 1997.

[77] S. S. Muchnick and N. D. Jones. *Program Flow Analysis : Theory and Applications*. Prentice-Hall Inc., 1981.

[78] M. Müller-Olm and O. Rüthing. On the complexity of constant propagation. In *Proceedings of the European Symposium on Programming Languages and Systems*, pages 190–205. Springer-Verlag, 2001.

[79] E. M. Myers. A precise inter-procedural data flow algorithm. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 219–230. ACM, 1981.

[80] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1998.

[81] A. Reid, J. McCorquodale, J. Baker, W. Hsieh, and J. Zachary. The need for predictable garbage collection. In *Proceedings of the ACM SIGPLAN Workshop on Compiler Support for System Software*. ACM, 1999.

[82] T. W. Reps, S. Horwitz, and S. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the ACM SIGPLAN-SIGACT*

Symposium on Principles of Programming Languages, pages 49–61. ACM, 1995.

[83] S. E. Richardson and M. Ganapathi. Interprocedural optimizations : Experimental results. *Software Practice and Experience*, 19(2):149–169, 1989.

[84] B. K. Rosen. Monoids for rapid data flow analysis. *SIAM Journal of Computing*, 9(1):159–196, 1980.

[85] B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Global value numbers and redundant computations. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–27. ACM, 1988.

[86] B. G. Ryder and M. C. Paull. Elimination algorithms for data flow analysis. *ACM Computing Surveys*, 18(3):277–316, 1986.

[87] M. Sagiv, T. Reps, and S. Horwitz. Precise interprocedural dataflow analysis with applications to constant propagation. *Theoretical Computer Science*, 167(1–2):131–170, 1996.

[88] S. Sagiv, T. W. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Transactions on Programming Languages and Systems*, 24(3):217–298, 2002.

[89] R. Shaham, E. K. Kolodner, and M. Sagiv. On effectiveness of GC in Java. In *International Symposium on Memory Management*, pages 12–17. ACM, 2000.

[90] R. Shaham, E. K. Kolodner, and M. Sagiv. Heap profiling for space-efficient java. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 104–113. ACM, 2001.

[91] R. Shaham, E. K. Kolodner, and M. Sagiv. Estimating the impact of heap liveness information on space consumption in Java. In *Proceedings of the International Symposium on Memory Management*, pages 64–75. ACM, 2002.

[92] R. Shaham, E. Yahav, E. K. Kolodner, and S. Sagiv. Establishing local temporal heap safety properties with applications to compile-time memory management. In *Proceedings of the International Static Analysis Symposium*, pages 483–503. Springer-Verlag, 2003.

[93] M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In S. S. Muchnick and N. D. Jones, editors, *Program Flow Analysis : Theory and Applications*. Prentice-Hall Inc., 1981.

[94] T. C. Spillman. Exposing side effects in a PL/I optimizing compiler. In *Proceedings of IFIP Congress 71*, pages 376–381. North Holland Publishing Company, 1971.

[95] V. C. Sreedhar and G. R. Gao. A linear time algorithm for placing ϕ-nodes. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 62–73. ACM, 1995.

[96] V. C. Sreedhar, R. Dz-Ching Ju, D. M. Gillies, and V. Santhanam. Translating out of static single assignment form. In *Proceedings of the International Symposium on Static Analysis*, pages 194–210, 1999.

[97] B. Steensgaard. Points-to analysis in almost linear time. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 32–41. ACM, 1996.

[98] R. E. Tarjan. Fast algorithms for solving path problems. *Journal of ACM*, 28(3):594–614, 1981.

[99] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[100] J. D. Ullman. Fast algorithms for the elimination of common subexpressions. *Acta Informatica*, 2(3):191–213, 1973.

[101] V. Vyssotsky and P. Wegner. A graph theoretical FORTRAN source language analyzer. AT & T Bell Laboratories, Murray Hill, N. J., 1963. (Manuscript).

[102] M. N. Wegman and F. K. Zadeck. Constant propagation with conditional branches. *ACM Transactions on Programming Languages and Systems*, 13(2):181–210, 1991.

[103] M. N. Wegman and F. K. Zadeck. Constant propagation with conditional branches. *ACM Transactions on Programming Languages and Systems*, 13(2):181–210, 1991.

[104] J. Whaley and M. S. Lam. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2004.

[105] R. Wilhelm and D. Maurer. *Compiler Design*. Addison-Wesley, 1995.

[106] R. Wilhelm, T. W. Reps, and S. Sagiv. Shape analysis and applications. In Y. N. Srikant and Priti Shankar, editors, *The Compiler Design Handbook: Optimizations & Machine Code Generation*, pages 175–218. CRC Press, 2002.

[107] R. P. Wilson and M. S. Lam. Efficient context-sensitive pointer analysis for C programs. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1–12. ACM, 1995.