

# References

- P. Anderson, D. Binkley, G. Rosay, T. Teitelbaum, Flow insensitive points-to sets. *Inf. Softw. Technol.* **44**(13), 743–754 (2002)
- W.A. Appel, M. Ginsburg, *Modern Compiler Implementation in C* (Cambridge University Press, Cambridge, 2004)
- S. Abramsky, C. Hankin (Hrsg.), *Abstract Interpretation of Declarative Languages* (Ellis Horwood, Chichester, 1987)
- A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman, *Compilers: Principles, Techniques, & Tools*, 2nd revised edn. (Addison-Wesley, New York, 2007)
- T. Amtoft, F. Nielson, H.R. Nielson, Type and behaviour reconstruction for higher-order concurrent programs. *J. Funct. Program.* **7**(3), 321–347 (1997)
- W.A. Appel, *Compiling with Continuations* (Cambridge University Press, Cambridge, 2007)
- D.F. Bacon, Fast and effective optimization of statically typed object-oriented languages. Ph.D. thesis, Berkeley, 1997
- H.P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, Volume 103 of Studies in Logic and the Foundations of Mathematics, revised edition (North Holland, Amsterdam, 1984)
- R.M. Burstall, J. Darlington, A transformation system for developing recursive programs. *J. ACM* **24**(1), 44–67 (1977)
- G.L. Burn, C. Hankin, S. Abramsky, Strictness analysis for higher-order functions. *Sci. Comput. Program.* **7**(3), 249–278 (1986)
- P. Cousot, R. Cousot, Static determination of dynamic properties of programs, in *2nd International Symposium on Programming*, pp. 106–130. Dunod, Paris, France, 1976
- P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in *4th ACM Symposium on Principles of Programming Languages (POPL)*, pp. 238–252, 1977a
- P. Cousot, R. Cousot, Static determination of dynamic properties of recursive procedures, ed. by E.J. Neuhold (Hrsg.), in *IFIP Conference on Formal Description of Programming Concepts*, pp. 237–277. (North Holland, Amsterdam, 1977b)
- P. Cousot, R. Cousot, Systematic design of program transformation frameworks by abstract interpretation, in *29th ACM Symposium on Principles of Programming Languages (POPL)*, pp. 178–190, 2002
- J.-D. Choi, M. Gupta, M. Serrano, V.C. Sreedhar, S. Midkiff, Escape analysis for Java. *SIGPLAN Not.* **34**(10), 1–19 (1999)
- P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in *5th ACM Symposium on Principles of Programming Languages (POPL)*, pp. 84–97, 1978

- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, 3rd edn. (MIT Press, Cambridge, 2009)
- K.D. Cooper, L. Torczon, *Engineering a Compiler* (Morgan Kaufmann, Massachusetts, 2004)
- M. Fähndrich, J. Rehof, M. Das, Scalable context-sensitive flow analysis using instantiation constraints. *SIGPLAN Not.* **35**(5), 253–263 (2000)
- C. Fecht, H. Seidl, A faster solver for general systems of equations. *Sci. Comput. Program. (SCP)* **35**(2), 137–161 (1999)
- A.J. Gill, J. Launchbury, S.L.P. Jones, A short cut to deforestation, in *Functional Programming and Computer Architecture (FPCA)*, pp. 223–232, 1993
- R. Giegerich, U. Möncke, R. Wilhelm, Invariance of approximate semantics with respect to program transformations. In *GI Jahrestagung*, pp. 1–10, 1981
- P. Granger, Static analysis of linear congruence equalities among variables of a program, in *International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, pp. 169–192. LNCS 493 (Springer, Heidelberg, 1991)
- T. Gawlitza, H. Seidl, Precise fixpoint computation through strategy iteration, in *European Symposium on Programming (ESOP)*, pp. 300–315. LNCS 4421 (Springer, Heidelberg, 2007)
- M.S. Hecht, *Flow Analysis of Computer Programs* (North Holland, Amsterdam, 1977)
- N. Heintze, Set-based analysis of ML programs. *SIGPLAN Lisp Pointers* **VII**(3), 306–317 (1994)
- M. Hofmann, A. Karbyshev, H. Seidl, in *Verifying a Local Generic Solver in Coq*, ed. by R. Cousot, M. Martel. Static Analysis, Volume 6337 of Lecture Notes in Computer Science (Springer, Heidelberg, 2010), pp. 340–355
- M. Hofmann, A. Karbyshev, H. Seidl, *What is a Pure Functional?* ed. by S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, P.G. Spirakis (Hrsg.), ICALP (2), Volume 6199 of Lecture Notes in Computer Science (Springer, Heidelberg, 2010), pp. 199–210
- S.L.P. Jones, W. Partain, A. Santos, Let-floating: moving bindings to give faster programs, in *International Conference on Functional Programming (ICFP)*, pp. 1–12, 1996
- L. Simon, P. Jones, A.L.M. Santos, A Transformation-based optimiser for Haskell. *Sci. Comput. Program.* **32**(1–3), 3–47 (1998)
- M. Karr, Affine relationships among variables of a program. *Acta Inf.* **6**, 133–151 (1976)
- G.A. Kildall, A unified approach to global program optimization, in *ACM Symposium on Principles of Programming Languages (POPL)*, pp. 194–206, 1973
- J. Knoop, *Optimal Interprocedural Program Optimization, A New Framework and Its Application*, LNCS 1428. (Springer, Berlin, 1998)
- J. Knoop, O. Rüthing, B. Steffen, Optimal code motion: theory and practice. *ACM Trans. Program. Lang. Syst.* **16**(4), 1117–1155 (1994)
- J. Knoop, O. Rüthing, B. Steffen, Partial dead code elimination, in *ACM Conference on Programming Languages Design and Implementation (PLDI)*, pp. 147–158, 1994
- J. Knoop, B. Steffen, The interprocedural coincidence theorem, in *4th International Conference on Compiler Construction (CC)*, pp. 125–140. LNCS 541 (Springer, Heidelberg, 1992)
- S. Kundu, Z. Tatlock, S. Lerner, Proving optimizations correct using parameterized program equivalence, in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2009
- J.B. Kam, J.D. Ullman, Global data flow analysis and iterative algorithms. *J. ACM* **23**(1), 158–171 (1976)
- J.B. Kam, J.D. Ullman, Monotone data flow analysis frameworks. *Acta Inf.* **7**, 305–317 (1977)
- X. Leroy, Formal verification of a realistic compiler. *Commun. ACM* **52**(7), 107–115 (2009)
- S. Lerner, T.D. Millstein, C. Chambers, Automatically proving the correctness of compiler optimizations, in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 220–231, 2003
- S. Lerner, T. Millstein, E. Rice, C. Chambers, Automated soundness proofs for dataflow analyses and transformations via local rules, in *32nd ACM Symp. on Principles of Programming Languages (POPL)*, pp. 364–377, 2005

- D. Liang, M. Pennings, M.J. Harrold, Extending and evaluating flow-insensitive and context-insensitive points-to analyses for Java, in *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering (PASTE)*, pp. 73–79, 2001
- F. Martin, M. Alt, R. Wilhelm, C. Ferdinand, Analysis of loops, in *7th International Conference on Compiler Construction (CC)*, pp. 80–94. LNCS 1383, Springer, 1998.
- S.S. Muchnick, N.D. Jones (Hrsg.), *Program Flow Analysis: Theory and Application* (Prentice Hall, Englewood Cliffs, 1981)
- M. Müller-Olm, H. Seidl, Precise interprocedural analysis through linear algebra, in *31st ACM Symposium on Principles of Programming Languages (POPL)*, pp. 330–341, 2004.
- M. Müller-Olm, H. Seidl, A generic framework for interprocedural analysis of numerical properties, in *Static Analysis, 12th International Symposium (SAS)*, pp. 235–250. LNCS 3672 (Springer, Heidelberg, 2005)
- M. Müller-Olm, H. Seidl, Analysis of modular arithmetic. *ACM Trans. Program. Lang. Syst.* **29**(5), 2007
- S.S. Muchnick, *Advanced Compiler Design and Implementation* (Morgan Kaufmann, Massachusetts, 1997)
- A. Mycroft, The theory and practice of transforming call-by-need into call-by-value, in *Symposium on Programming: Fourth 'Colloque International sur la Programmation'*, pp. 269–281. LNCS 83 (Springer, Heidelberg, 1980)
- F. Nielson, H.R. Nielson, C. Hankin, *Principles of Program Analysis* (Springer, Heidelberg, 1999)
- R. Paige, Symbolic finite differencing—Part I, in *3rd European Symposium on Programming (ESOP)*, pp. 36–56. LNCS 432 (Springer, Heidelberg, 1990)
- R. Paige, J.T. Schwartz, Reduction in strength of high level operations, in *4th ACM Symposium on Principles of Programming Languages (POPL)*, pp. 58–71, 1977
- G. Ramalingam, On loops, dominators, and dominance frontiers. *ACM Trans. Program. Lang. Syst. (TOPLAS)* **24**(5), 455–490 (2002)
- V. Sundareshan, L. Hendren, C. Razafimahefa, R. Vallée-Rai, P. Lam, E. Gagnon, C. Godin, Practical virtual method call resolution for Java, in *15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pp. 264–280, 2000
- A. Simon, *Value-Range Analysis of C Programs: Towards Proving the Absence of Buffer Overflow Vulnerabilities* (Springer, Heidelberg, 2008)
- J. Sheldon, W. Lee, B. Greenwald, S.P. Amarasinghe, Strength reduction of integer division and modulo operations, in *Languages and Compilers for Parallel Computing, 14th International Workshop (LCPC). Revised Papers*, pp. 254–273. LNCS 2624 (Springer, Heidelberg, 2003)
- M. Sharir, A. Pnueli, Two approaches to interprocedural data flow analysis, ed. by S.S. Muchnick, N.D. Jones (Hrsg.). *Program Flow Analysis: Theory and Application*, pp. 189–234 (Prentice Hall, Englewood Cliffs, 1981)
- R.C. Sekar, S. Pawagi, I.V. Ramakrishnan, Small domains spell fast strictness analysis, in *ACM Symposium on Principles of Programming Languages (POPL)*, pp. 169–183, 1990
- M. Sagiv, T.W. Reps, R. Wilhelm, Parametric shape analysis via 3-valued logic, in *26th ACM Symposium on Principles of Programming Languages (POPL)*, pp. 105–118, 1999
- M.Sagiv, T.W. Reps, R. Wilhelm, Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst. (TOPLAS)* **24**(3), 217–298 (2002)
- H. Seidl, M.H. Sørensen, Constraints to stop deforestation. *Sci. Comput. Program.* **32**(1–3), 73–107 (1998)
- J.P. Secher, M.H. Sørensen, On perfect supercompilation, in *3rd Int. Andrei Ershov Memorial Conference: Perspectives of System Informatics (PSI)*, pp. 113–127. LNCS 1755 (Springer, Heidelberg, 1999)
- Y.N. Srikant, P. Shankar (Hrsg.), *The Compiler Design Handbook: Optimizations and Machine Code Generation* (CRC Press, Boca Raton, 2003)

- B. Steensgaard, Points-to analysis in almost linear time, in *23rd ACM Symposium on Principles of Programming Languages (POPL)*, pp. 32–41, 1996
- J.-B. Tristan, X. Leroy, Verified validation of lazy code motion, in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 316–326, 2009
- A. Takano, E. Meijer, Shortcut deforestation in calculational form, in *SIGPLAN-SIGARCH-WG2.8 Conference on Functional Programming Languages and Computer Architecture (FPCA)*, pp. 306–313, 1995
- P. Wadler, Deforestation: transforming programs to eliminate trees. *Theor. Comput. Sci.* **73**(2), 231–248 (1990)

# Index

## A

- Abstract interpretation, 47
- Algorithm, 23, 83, 87
  - recursive fixed-point, 88, 89, 114
  - round-robin, 23, 25, 27
  - worklist
- Alias, 67
  - may, 67
  - must, 67
- Alias analysis, 66
- $\alpha$  conversion, 4
- Analysis, 72
  - distributive framework, 30
  - flow-insensitive, 75
  - interprocedural, 124
  - monotonic framework, 26
  - points-to, 72
- Antisymmetry, 17
- Application
  - partial, 1
- Approach
  - call string, 135
  - functional, 133
- Array-bounds check, 4
- Assignment
  - available, 138
  - between variables, 40
  - dead, 32
  - delayable, 103
  - partially dead, 102
  - partially redundant, 108
  - very busy, 132

## B

- Back edge, 99
- Backward analysis, 34

- $\beta$  reduction, 143
- Bottom, 18
- Bound
  - greatest lower, 16, 18
  - least upper, 16, 17
  - upper, 17

## C

- C, 141, 144
- Call, 117, 122, 135
  - last, 123
- Call graph, 122
- Call stack, 117, 124
- Call string, 135
- Chain
  - ascending, 166
  - descending, 65
  - stable ascending, 88
- Closure, 159
- Code
  - loop invariant, 97
- Compilation
  - separate, 124
- Computation
  - reaching, 120
  - redundant, 8
  - same level, 119
  - step, 8
- Concretization, 47
- Constant folding, 42
- Constant propagation, 44, 42
  - interprocedural, 133
- Control-flow graph, 8
  - interprocedural, 116
- Copy propagation
  - interprocedural, 40, 124, 126

**C** (*cont.*)

## Correctness

- of constant propagation, 47
- of interprocedural analysis, 127
- of interval analysis, 59
- of points-to analysis, 74
- of the transformation PRE, 97
- of transformation RE, 42
- of transformation DE, 37

**D**

## Data structure

- intermediate, 155
- union-find-, 79

## Data-flow analysis, vi, 35

## Dead-code elimination, 35

## Deforestation, 15

## Dependence, 68

## Dependence analysis, 68

## Description relation, 47

## .NET instructions, 1

**E**

## Edge effect

- abstract, 11
- concrete, 9

## Element

- atomic, 29
- greatest, 18
- least, 18

## Equivalence class, 77

## Evaluation

- eager, 5, 19
- lazy, 141, 144, 159
- multiple, 7
- partial, 44

## Expression

- abstract evaluation of, 45
- available, 11
- concrete evaluation of, 10

**F**

## Feedback vertex set, 62

## Fixed point

- greatest, 23
- least, 22
- post-, 22

## Fixed-point iteration

- accumulating, 24
- local, 89
- naive, 23

recursive, 87

round-robin, 23

worklist, 83

## Fortran, 3

## Forward analysis, 34

## F#, 1

## Function

- distributive, 28
- folding, 8
- inlining, 6
- monotonic, 20
- specialization, 7
- strict, 28
- totally distributive, 28

## Functional abstraction, 2

**H**

## Haskell, 141, 144, 147

**I**

## Inequalities, 15

system of

## Inlining

- of functions, 6
- procedure, 121

## Interval analysis, 53

## Interval arithmetic, 56

**J**

## Java, 4

**L** $\lambda$  calculus, 143

## Lattice

- atomic, 29
- complete, 16, 17
- flat, 18
- height of, 25
- powerset-, 18

## Lisp, 7

## List constructor, 8

## Loop inversion, 98

**M**

## Memoization, 7

## Memory

dynamically allocated, 67

## Memory cell

uninitialized, 74

**N**

Narrowing, 63

**O**

Operator, 65

Ocaml, 141, 145

Order

dual, 23

partial, 16

**P**

Partial order, 16

Partition

refinement of, 77

Path

infeasible, 14

Pattern, 10

Pattern matching, 141

Pointer, 67

Pointer arithmetic, 67

Polymorphism, 141

Predominator, 98

Program

well-typed, 3

Program optimization semantics, 3

Program point, 8

Program state, 8

**R**

Recursion

tail, 123

Redundancy

partial, 89

Redundancy elimination, 13

Reflexivity, 17

Register

virtual, 4

Register allocation, 4

Root-strictness, 23

Round-robin iteration

correctness, 25

**S**

Scala, 141

Semantics

denotational, 162

instrumented, 74

operational, 3, 154

small-step operational, 8

Side effects, 5

Solution, 149

merge-over-all-paths, 27

Stack frame, 117

Strictness

root, 23

total, 24

Substitution, 3

Supergraph

interprocedural, 135

System of inequalities

size of, 85

**T**

Termination, 145

Top, 18

Total strictness, 164

Transitivity, 17

Tree grammar

regular, 151

type inference, 141

type system, 67

**V**

Value analysis, 149

expression, 150

Variable

binding, 9, 10

abstract, 44

dead, 32

definition of, 32

global, 32

live, 33

order, 25

partially dead, 102

renaming of a, 144

true use, 37

truly live, 37

use of, 32

Verification, 4

**W**

Widening, 60

-operator, 61