An Appreciation of the Work of Reinhard Wilhelm

Thomas Reps[1], Mooly Sagiv[2], and Jörg Bauer[3]

[1] Comp. Sci. Dept., University of Wisconsin; reps@cs.wisc.edu
[2] School of Comp. Sci., Tel-Aviv University; msagiv@post.tau.ac.il
[3] Fachrichtung Informatik, Univ. des Saarlandes;joba@cs.uni-sb.de

Reinhard Wilhelm's career in Computer Science spans more than a third of a century. During this time, he has made numerous research contributions in the areas of programming languages, compilers and compiler generators, static program analysis, program transformation, algorithm animation, and real-time systems; co-founded a company to transfer some of these ideas to industry; held the Chair for Programming Languages and Compiler Construction at Saarland University; and served since its inception as the Scientific Director of the International Conference and Research Center for Computer Science at Schloß Dagstuhl.

1 Research Activities

1.1 Foundations of Programming and Programming Languages

Reinhard's work in the area of programming languages is unusual in that he has had an interest in, and made contributions in, *all* styles of programming languages (imperative, functional, logic, parallel, object-oriented, and constraint-oriented) [14–17, 20, 19, 21, 22, 24–26, 31, 77].

1.2 Compilers, Compiler Generators, and Compilation Algorithms

Compilers convert a program or specification written in some language into a form that allows it to be executed on a computer or network of computers. *Compiler generators* are tools for creating compilers themselves (or components of compilers) from specifications. From the number of his publications on these subjects, and their distribution in time, it is easy to see that Reinhard's longest-held interest and deepest attachment in Computer Science has been to the area of compilers [32], including compiler generators [27–30] and the algorithms needed to accomplish the tasks required in various phases of compilation (see below).

Attribute Grammars. Attribute grammars are a language-description formalism that was introduced by Donald Knuth in 1968.[4] In an attribute grammar, a language and its properties are specified by giving

– a context-free grammar for the language,
– sets of "attributes" (annotations) to be attached to nodes of the grammar's derivation trees, and

[4] D.E. Knuth: Semantics of context-free languages. Math. Syst. Theory 2(2): 127-145 (1968).

– rules that describe how information from one node's attributes affects the attributes of other nodes.

Reinhard's research in this area concerned analysis and applications of attribute grammars [33–41, 60, 61]. Reinhard also spearheaded the implementation of three of the most influential compiler-generation systems that were based on attribute grammars: MUG1 [28], MUG2 [29, 30], and OPTRAN [38, 77, 40, 41, 72].

Code Generation via Transformational Grammars/Tree Automata. The code-generation phase of a compiler turns an intermediate representation of a program into a sequence of machine-code instructions. One code-generation subtask is code selection—the actual selection of the instructions to be emitted. In a sequence of papers [44, 46, 48], Reinhard together with Helmut Seidl developed the connections between the code-selection problem and the theories of regular-tree grammars and finite tree automata. This work provided a sound theoretical basis for code selection, generalized and improved existing methods, and furthered the understanding of how to generate code-selection components of compilers from machine descriptions.

Other Compilation Algorithms. Other compilation issues with which Reinhard has been concerned include table compression [47], graph reduction [42], code optimization [43, 49], and virtual machines [50, 51].

1.3 Static Program Analysis

One of the areas in which Reinhard has worked for multiple decades, and made many important contributions, is *static program analysis* (also known as "dataflow analysis" or "abstract interpretation"). Static analysis is the basic technique used in optimizing compilers and other programming tools for obtaining information about the possible states that a program can reach during execution, but without actually running the program on specific inputs. Instead, static-analysis techniques explore a program's behavior for *all* possible inputs to determine (an approximation of) *all* possible states that the program can reach. To make this feasible, the program is "run in the aggregate"—i.e., on descriptors that represent collections of many states. The fundamental theory that provides the foundation for such techniques is that of *abstract interpretation*, enunciated in 1977 by Patrick and Radhia Cousot.[5]

Reinhard's work in the area of static analysis includes [52–58, 61, 59], as well as many other papers mentioned below.

Grammar Flow Analysis. In two papers with U. Möncke [60, 61], Reinhard proposed the technique of "grammar flow analysis". This work generalized the concepts and algorithms of (intraprocedural) dataflow analysis—which generally is applied to graph data structures—to a method for obtaining information about properties of derivation trees

[5] P. Cousot, R. Cousot: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. POPL 1977: 238-252.

of the nonterminals in context-free grammars (but without actually building the grammar's derivation trees). Not only does this generalization have important applications in interprocedural dataflow analysis, but Möncke and Wilhelm also showed how grammar flow analysis has applications in many other areas of compilers—ranging from static-semantic analysis to code generation.

Shape Analysis. In a series of papers with M. Sagiv, T. Reps, and others, (e.g., [62–71]), Reinhard has addressed one of the major remaining challenges in static analysis of program behavior—sometimes called *shape analysis*—namely, how to check properties of programs that manipulate linked data structures (i.e., programs that use dynamic allocation and freeing of storage cells, and destructive updating of structure fields—as happens pervasively in languages such as C, C++, and Java). In such programs, data structures can grow and shrink dynamically, with no fixed upper bound on their size or number. The analysis problem is complicated even more by the fact that such languages also permit fields of dynamically allocated objects to be destructively updated. In the case of thread-based languages, such as Java, the number of threads can also grow and shrink dynamically—again with no fixed upper bound on size or number. These features create considerable difficulties for any method that tries to check program properties, and this subject is considered to be one of the most challenging areas of program analysis.

A key issue when analyzing programs that use such features is how to create finite-sized descriptors of memory configurations, such that the descriptors

– abstract away certain details, but
– retain enough key information so that the analysis can identify interesting properties that hold.

One of the crucial obstacles is that dynamically-allocated storage cells have no static names, and the number of such objects is, in general, unbounded.

The Sagiv-Reps-Wilhelm approach [67] provides a solution to the problem of devising finite-sized descriptors of the infinite number of storage configurations that can arise at a point in the program. Moreover, it also provides a way to tune the precision of the descriptors in use, which is important both for reducing the time required to run an analysis, and for reducing the number of false positives that an analysis may report. From a static-analysis perspective, the work is novel because of the way that it uses 3-valued logic to address these issues.

1.4 Program Transformation/Rewriting

The heart of many compilation steps (such as optimization and code generation) is *tree transformation* or *rewriting*. Reinhard's work in this area has concerned systems for program transformation and optimization—originally formalized using attribute-grammar-based notations and implemented as rewriting of attributed abstract-syntax trees [72–75, 77]. Later work addressed these problems using a specialized functional notation for specifying patterns and transformations [78].

1.5 Algorithm Animation and Visualization

Algorithm animation concerns how to create visual presentations of computations. This area has been another of Reinhard's long-term interests [79–83]. He has worked on such varied problems as the depiction of compilation steps [83], finite-state automata [82], and abstract interpretation [79, 80].

1.6 Timing Analysis for Real-Time Systems

In a series of papers with various authors (beginning with [84]), Reinhard spearheaded a new approach to predicting the timing behavior of processors that are equipped with cache memory. Because of the substantial differences exhibited by modern processors between the latency for an access that goes to main memory versus an access that can be resolved in the cache, cache behavior has to be taken into account to predict a program's execution time. Classical methods based on experimental measurement provide only soft guarantees: any measurement-based approach, via either hardware or software, only determines the execution time for the specific inputs of the test suite. Moreover, software-based monitoring alters the code, thereby influencing the cache behavior—and hence the performance—of the program whose performance one is trying to measure.

The behavior-prediction techniques that Reinhard has helped to devise make use of static program analysis, which (as noted above) provides a way to obtain information about the possible states that a program reaches during execution, but without actually running the program on specific inputs. Using this approach, Reinhard and his colleagues have shown that it is possible to determine the worst-case cache behavior (i.e., to identify what could be in the cache at each point in the program) [84–89], as well as to bound the worst-case running time of real-time programs and to check the program for the possibility of cache conflicts that could affect time-critical parts [90–98]. The advantage of their approach is that the predictions obtained using their techniques are valid for all inputs. From a static-analysis perspective, the work often combines "may" and "must" information in an unusual way.

An additional area of concern has been with processor models—e.g., with such issues as the semantics of processors and how to specify them—as well as with gaining an understanding of which real-time systems are time-predictable [99–103].

2 Technology Transfer

Reinhard has been successful in making the work on timing analysis for real-time systems accessible to the embedded-systems and real-time communities, thereby demonstrating how static program analysis can be applied to the problems that arise in these domains. This work also forms the basis for the products of a company that Reinhard co-founded (AbsInt Angewandte Informatik GmbH, Saarbruecken, Germany [104]). Their tool has been successfully used to certify time-critical subsystems of the Airbus A380 plane.

3 Pedagogical Activities

Reinhard Wilhelm holds the Chair for Programming Languages and Compiler Construction at Saarland University, where he has supervised twenty-five Ph.D. theses and nearly one hundred fifty Masters/Diploma theses.

Reinhard's textbook ("Compiler Design"), co-authored with D. Maurer and published in German [10], French [11], and English [12], is successfully used in many graduate and undergraduate compiler courses. The book is noteworthy for the way that it presents the problems of compiling imperative languages, functional languages, object-oriented languages, and logic-programming languages in a way that draws out their commonalities. The book covers in depth many difficult aspects of compilation, such as bottom-up parsing with error recovery, attribute grammars, and abstract interpretation. Compared with other textbooks, Reinhard's book provides the most theoretically well-grounded treatment of the problems that arise in writing compilers.

Recently, Reinhard also participated in an effort to design a graduate curriculum on embedded software and systems [13].

4 Schloß Dagstuhl

Reinhard has also performed a notable service to the international Computer Science community by having served since its inception as the Scientific Director of the International Conference and Research Center for Computer Science at Schloß Dagstuhl. Dagstuhl was set up in 1990 along the lines of the famous conference center for Mathematics at Oberwolfach. Dagstuhl hosts several kinds of activities, but predominantly "Dagstuhl Seminars", which are week-long intensive seminars involving lectures, group discussions, software demonstrations, etc. By now, Dagstuhl Seminars have been held in a large number of different subject areas of Computer Science. There are about forty of these every year, each with somewhere between twenty and sixty participants from all over the world.

Reinhard has been involved with Dagstuhl from the start, and as the founding Scientific Director, his imprint is to be found on all aspects of its operation, including not just the choice of seminar topics and attention to maintaining the highest scientific standards, but also the original renovation of Schloß Dagstuhl, the design and construction of a major second building, the solicitation of donations from industry and charitable foundations, the arrangement of a special funding program for junior researchers (young faculty and graduate students), and the list could go on and on. Many weeks, he even leads the traditional Wednesday mid-week hike.

Reinhard has made Dagstuhl a very special place for Computer Scientists by his quiet, but skillful, way of using his "bully pulpit" to nudge the Computer Science community into fruitful and interesting interactions:

- He has made seminars that bring together groups that have common interests, but that for one reason or another have had relatively limited contact, a Dagstuhl specialty.
- He attends at least one day of each Dagstuhl seminar to monitor progress and offer advice to that week's organizers about possible mid-course corrections.

– He actively solicits Seminars on new and promising topics.
– During the planning stages of Seminars, he keeps an eye out for potential attendees who might have been overlooked, but who would especially profit from and/or contribute to the activities of a Seminar.

Dagstuhl and its attendees have also benefited from Reinhard's careful—and tasteful—attention to detail, which goes far beyond the scientific aspects of the establishment: the choice of music scores and musical instruments in Dagstuhl's music room; the organizing of displays of modern art in the Dagstuhl buildings and on the Dagstuhl grounds; even the choice of wine in the famous Dagstuhl Wine Cellar, where participants are encouraged to repair for the evening for both technical and non-technical conversation.

5 A Partial List of Reinhard Wilhelm's Collaborators

L. Almeida, M. Alt, H.-J. Bach, M. Baston, J. Bauer, G. Becker, Y. Ben-Asher, A. Benveniste, C. Berg, J. Börstler, P. G. Bouillon, B. Bouyssounouse, B. Braune, F. Warren Burton, G. C. Buttazzo, P. Caspi, J. Ciesinger, V. Claus, I. Crnkovic, W. Damm, S. Diehl, J. Engblom, A. A. Evstiougov-Babaev, C. Fecht, C. Ferdinand, G. Fohler, F. Fontaine, N. Francez, N. Fritz, H. Ganzinger, M. García-Valls, R. Giegerich, I. Glasner, H. Hagen, R. Heckmann, E. Hoffmann, D. Johannes, D. Kästner, A. Kerren, H. Kopetz, B. Kuhn, W. Lahner, Y. Lakhnech, M. Langenbach, F. Laroussinie, L. Lavagno, T. Lev-Ami, G. Lipari, P. Lipps, J. Loeckx, P. Lucas, A. Lucks-Baus, F. Maraninchi, F. Martin, D. Maurer, K. Mehlhorn, J. Messerschmidt, U. Möncke, T. Müldner, F. Müller, R. Nollmann, H.-G. Oberhauser, M. Olk, O. Parshin, P. Peti, J. Antonio de la Puente, M. Raber, A. Rakib, F. Randimbivololona, T. Rauber, T. Remmel, T. Reps, N. Rinetzky, K. Ripken, B. Robinet, M. Rodeh, G. Rünger, M. Sagiv, G. Sander, A. L. Sangiovanni-Vincentelli, N. Scaife, M. Schmidt, J. Schneider, A. Schuster, R. Seidel, H. Seidl, M. Sicks, J. Sifakis, R. de Simone, J. Souyris, O. Spaniol, H. Theiling, S. Thesing, L. Thiele, W. Thome, M. Törngren, P. Veríssimo, B. Weisgerber, A.J. Wellings, D.B. Whalley, S. Wilhelm, T.A.C. Willemse, E. Yahav, W. Yi, G. Yorsh.

Acknowledgments

We would like to thank Angelika Müller, Rosi Bardohl, Annette Beyer, and all of the Dagstuhl staff for the splendid job they did organizing and hosting the Reinhard Wilhelm 60th-Birthday Celebratory Symposium. We would also like to thank Patrick Cousot for furnishing us with the list of references to Reinhard's publications that he assembled for the presentation of Reinhard's work at the Saarland University celebration of Reinhard's birthday.

References

1. R. Wilhelm. *Informatics: 10 Years Back. 10 Years Ahead*. Lecture Notes in Computer Science 2000. Springer, Berlin, Germany, 2001.
2. R. Wilhelm: Compiler Construction, 10th International Conference, CC 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings Springer 2001

3. R. Wilhelm: Informatik: Grundlagen - Amwendungen - Perspektiven [Forum "Perspektiven der Informatik", Dagstuhl, November 1993] Verlag C. H. Beck 1996
4. R. Wilhelm: Generische und generative Methoden. Perspektiven der Informatik 1993: 84-85
5. R. Wilhelm, H. Hagen: Programmiersprachen. Perspektiven der Informatik 1993: 86-90
6. V. Claus, R. Wilhelm: Einleitung. Perspektiven der Informatik 1993: 9-12
7. R. Wilhelm, O. Spaniol: Parallele und verteilte Systeme. Perspektiven der Informatik 1993: 90-94
8. B. Robinet, R. Wilhelm: ESOP 86, European Symposium on Programming, Saarbrücken, Federal Republic of Germany, March 17-19, 1986, Proceedings, LNCS 213, Springer 1986
9. R. Wilhelm: GI - 10. Jahrestagung, Saarbrücken, 30. September - 2. Oktober 1980, Proceedings Springer 1980
10. R. Wilhelm and D. Maurer. *Übersetzerbau - Theorie, Konstruktion, Generierung.* Springer, Berlin, Germany, 1992, 2. Auflage Springer 1997
11. R. Wilhelm and D. Maurer. *Les Compilateurs, théorie, construction, génération*. Masson, Paris, France, 1994.
12. R. Wilhelm and D. Maurer. *Compiler Design: Theory, Construction, Generation*. Addison-Wesley, Reading, MA, 1996.
13. P. Caspi, A. L. Sangiovanni-Vincentelli, Luís Almeida, A. Benveniste, B. Bouyssounouse, G. C. Buttazzo, I. Crnkovic, W. Damm, J. Engblom, G. Fohler, M. García-Valls, H. Kopetz, Y. Lakhnech, François Laroussinie, L. Lavagno, G. Lipari, F. Maraninchi, P. Peti, J. Antonio de la Puente, N. Scaife, J. Sifakis, R. de Simone, M. Törngren, P. Veríssimo, A.J. Wellings, R. Wilhelm, T.A.C. Willemse, W. Yi: Guidelines for a Graduate Curriculum on Embedded Software and Systems. ACM Trans. Embedded Comput. Syst. 4(3): 587-611 (2005)
14. R. Wilhelm: Imperative, prädikative und funktionale Programmierung (Kurzfassung). GI Jahrestagung 1982: 188-193
15. J. Messerschmidt, R. Wilhelm: Constructors for Composed Objects. Comput. Lang. 7(2): 53-59 (1982)
16. R. Wilhelm: Symbolische Programmausführung - Das aktuelle Schlagwort. Informatik Spektrum 6(3): 170 (1983)
17. J. Loeckx, K. Mehlhorn, R. Wilhelm: Grundlagen der Programmiersprachen. Teubner, 1986
18. G. Becker, B. Kuhn, D. Maurer, R. Wilhelm: SiATEX - eine interaktive Arbeitsumgeubng für TEX. Innovative Informations-Infrastrukturen 1988: 162-169
19. J. Loeckx, K. Mehlhorn, R. Wilhelm: Foundations of Programming Languages. John Wiley, 1989
20. M. Baston, H.-J. Bach, A. Lucks-Baus, F. Müller, R. Wilhelm: Implementierung der funktionalen Programmiersprache HOPE mit Hilfe von Kombinatoren. Innovative Informations-Infrastrukturen 1988: 114-131
21. R. Wilhelm: Übersetzer für imperative, funktionale und logische Programmiersprachen: Ein Vergleich (eingeladener Vortrag). Software-Entwicklung 1989: 156-165
22. Y. Ben-Asher, G. Rünger, A. Schuster, R. Wilhelm: 2DT-FP: An FP Based Programming Language for Efficient Parallel Programming of Multiprocessor Networks. PARLE 1993: 42-55
23. Y. Ben-Asher, G. Rünger, R. Wilhelm, A. Schuster: Implementing 2DT on a Multiprocessor. CC 1994: 113-127
24. T. Rauber, G. Rünger, R. Wilhelm: An application specific parallel programming paradigm. HPCN Europe 1995: 735-740
25. R. Heckmann, R. Wilhelm: A Functional Description of TEX's Formula Layout. J. Funct. Program. 7(5): 451-485 (1997)
26. P. Lucas, N. Fritz, R. Wilhelm: The Development of the Data-Parallel GPU Programming Language CGiS. Int. Conf. on Computational Science, 2006: 200-203

27. H. Ganzinger, R. Wilhelm: Verschränkung von Compiler-Moduln. GI Jahrestagung 1975: 654-665
28. R. Wilhelm, K. Ripken, J. Ciesinger, H. Ganzinger, Walter Lahner, R. Nollmann: Design Evaluation of the Compiler Generating System MUGI. ICSE 1976: 571-576
29. H. Ganzinger, K. Ripken, R. Wilhelm: Automatic Generation of Optimizing Multipass Compilers. IFIP Congress 1977: 535-540
30. H. Ganzinger, R. Giegerich, U. Möncke, R. Wilhelm: A Truly Generative Semantics-Directed Compiler Generator. SIGPLAN Symposium on Compiler Construction 1982: 172-184
31. R. Wilhelm, M. Alt, F. Martin, M. Raber: Parallel Implementation of Functional Languages. LOMAPS 1996: 279-295
32. P. Lucas, N. Fritz, R. Wilhelm: The CGiS Compiler-A Tool Demonstration. CC 2006: 105-108
33. R. Giegerich, R. Wilhelm: Implementierbarkeit attributierter Grammatiken. GI Jahrestagung 1977: 17-36
34. R. Giegerich, R. Wilhelm: Counter-One-Pass Features in One-Pass Compilation: A Formalization Using Attribute Grammars. Inf. Process. Lett. 7(6): 279-284 (1978)
35. R. Wilhelm: Attributierte Grammatiken. Informatik Spektrum 2(3): 123-130 (1979)
36. R. Wilhelm: LL- and LR-Attributed Grammars. Fachtagung über Programmiersprachen 1982: 151-164
37. U. Möncke, B. Weisgerber, R. Wilhelm: How to Implement a System for Manipulation of Attributed Trees. Fachtagung über Programmiersprachen 1984: 112-127
38. P. Lipps, U. Möncke, M. Olk, R. Wilhelm: Attribute (Re)evaluation in OPTRAN. Acta Inf. 26(3): 213-239 (1988)
39. Winfried Thome, R. Wilhelm: Simulating Circular Attribute Grammars Through Attribute Reevaluation. Inf. Process. Lett. 33(2): 79-81 (1989)
40. R. Wilhelm: Attribute Reevaluation in OPTRAN. Attribute Grammars, Applications and Systems 1991: 507
41. P. Lipps, U. Möncke, R. Wilhelm: An Overview of the OPTRAN System. Attribute Grammars, Applications and Systems 1991: 505-506
42. M. Raber, T. Remmel, E. Hoffmann, D. Maurer, F. Müller, H.-G. Oberhauser, R. Wilhelm: Complied Graph Reduction on a Processor Network. ARCS 1988: 198-212
43. R. Wilhelm: Code-Optimierung Mittels Attributierter Transformationsgrammatiken. GI Jahrestagung 1974: 257-266
44. B. Weisgerber, R. Wilhelm: Two Tree Pattern Matchers for Code Selection. CC 1988: 215-229
45. C. Ferdinand, H. Seidl, R. Wilhelm: Tree Automata for Code Selection. Code Generation 1991: 30-50
46. R. Wilhelm: Tree Tranformations, Functional Languages, and Attribute Grammars. WAGA 1990: 116-129
47. J. Börstler, U. Möncke, R. Wilhelm: Table Compression for Tree Automata. ACM Trans. Program. Lang. Syst. 13(3): 295-314 (1991)
48. C. Ferdinand, H. Seidl, R. Wilhelm: Tree Automata for Code Selection. Acta Inf. 31(8): 741-760 (1994)
49. P. G. Bouillon, G. Sander, R. Wilhelm: Lokale Optimierung ausnahmebehafteter Programme durch Spuroptimierung. Inform., Forsch. Entwickl. 9(2): 72-81 (1994)
50. D. Maurer, R. Wilhelm: MaMa - eine abstrakte Maschine zur Implementierung funktionaler Programmiersprachen. Inform., Forsch. Entwickl. 4(2): 67-88 (1989)
51. M. Alt, G. Sander, R. Wilhelm: Generation of Synchronization Code for Parallel Compilers. PLILP 1993: 420-421

52. N. Rinetzky, J. Bauer, T. Reps, M. Sagiv, R. Wilhelm: A Semantics for Procedure Local Heaps and its Abstractions. POPL 2005: 296-309
53. T. Lev-Ami, T. Reps, M. Sagiv, R. Wilhelm: Putting Static Analysis to Work for Verification: A Case Study. ISSTA 2000: 26-38
54. F. Martin, M. Alt, R. Wilhelm, C. Ferdinand: Analysis of Loops. CC 1998: 80-94
55. M. Sagiv, N. Francez, M. Rodeh, R. Wilhelm: A Logic-Based Approach to Program Flow Analysis. Acta Inf. 35(6): 457-504 (1998) 1997
56. R. Wilhelm: Program Analysis: A Toolmaker's Perspective. SIGPLAN Notices 32(1): 120-121 (1997) 1996
57. R. Wilhelm: Program Analysis - A Toolmaker's Perspective. ACM Comput. Surv. 28(4es): 177 (1996)
58. M. Sagiv, N. Francez, M. Rodeh, R. Wilhelm: A Logic-Based Approach to Data Flow Analysis Problem. PLILP 1990: 277-292
59. R. Wilhelm: Computation and Use of Data Flow Information in Optimizing Compilers. Acta Inf. 12: 209-225 (1979)
60. U. Möncke, R. Wilhelm: Grammar Flow Analysis. Attribute Grammars, Applications and Systems 1991: 151-186
61. U. Möncke, R. Wilhelm: Iterative Algorithms on Grammar Graphs, 8th Conf. on Graphtheoretic Concepts in Comp. Sci. 1982: 177-194
62. M. Sagiv, T. Reps, R. Wilhelm: Solving Shape-Analysis Problems in Languages with Destructive Updating. POPL 1996: 16-31
63. M. Sagiv, T. Reps, R. Wilhelm: Solving Shape-Analysis Problems in Languages with Destructive Updating. ACM Trans. Program. Lang. Syst. 20(1): 1-50 (1998)
64. M. Sagiv, T. Reps, R. Wilhelm: Parametric Shape Analysis via 3-Valued Logic. POPL 1999: 105-118
65. R. Wilhelm, M. Sagiv, T. Reps: Shape Analysis. CC 2000: 1-17
66. R. Wilhelm, T. Reps, M. Sagiv: Shape Analysis and Applications. The Compiler Design Handbook 2002: 175-218
67. M. Sagiv, T. Reps, R. Wilhelm: Parametric Shape Analysis via 3-Valued Logic. ACM Trans. Program. Lang. Syst. 24(3): 217-298 (2002)
68. E. Yahav, T. Reps, M. Sagiv, R. Wilhelm: Verifying Temporal Heap Properties Specified via Evolution Logic. ESOP 2003: 204-222
69. T. Reps, M. Sagiv, R. Wilhelm: Static Program Analysis via 3-Valued Logic. CAV 2004: 15-30
70. E. Yahav, T. Reps, M. Sagiv, R. Wilhelm: Verifying Temporal Heap Properties Specified Via Evolution Logic. Logic Journal of the IGPL 14, 5 (Oct. 2006): 755-784
71. G. Yorsh, T. Reps, M. Sagiv, R. Wilhelm: Logical Characterizations of Heap Abstractions. ACM Trans. Comp. Logic 8, 1 (Jan. 2007)
72. I. Glasner, U. Möncke, R. Wilhelm: OPTRAN, a Language for the Specification of Program Transformations. Fachtagung über Programmiersprachen 1980: 125-142
73. R. Giegerich, U. Möncke, R. Wilhelm: Invariance of Approximate Semantics with Respect to Program Transformations. GI Jahrestagung 1981: 1-10
74. R. Wilhelm: A Modified Tree-to-Tree Correction Problem. Inf. Process. Lett. 12(3): 127-132 (1981)
75. R. Wilhelm: Inverse Currying Transformation on Attribute Grammars. POPL 1984: 140-147
76. F. Warren Burton, D. Maurer, H.-G. Oberhauser, R. Wilhelm: A Space-Efficient Optimization of Call-by-Need. IEEE Trans. Software Eng. 13(6): 636-642 (1987)
77. P. Lipps, U. Möncke, R. Wilhelm: OPTRAN - A Language/System for the Specification of Program Transformations: System Overview and Experiences. CC 1988: 52-65
78. M. Alt, C. Fecht, C. Ferdinand, R. Wilhelm: Transformation Development: TrafoLa-H Subsystem. PROSPECTRA Book 1993: 539-576

79. D. Johannes, R. Seidel, R. Wilhelm: Algorithm Animation Using Shape Analysis: Visualising Abstract Executions. SOFTVIS 2005: 17-26
80. R. Wilhelm, T. Müldner, R. Seidel: Algorithm Explanation: Visualizing Abstract States and Invariants. Software Visualization 2001: 381-394
81. B. Braune, R. Wilhelm: Focusing in Algorithm Explanation. IEEE Trans. Vis. Comput. Graph. 6(1): 1-7 (2000)
82. B. Braune, S. Diehl, A. Kerren, R. Wilhelm: Animation of the Generation and Computation of Finite Automata for Learning Software. WIA 1999: 39-47
83. G. Sander, M. Alt, C. Ferdinand, R. Wilhelm: CLaX - A Visualized Compiler. Graph Drawing 1995: 459-462
84. M. Alt, C. Ferdinand, F. Martin, R. Wilhelm: Cache Behavior Prediction by Abstract Interpretation. SAS 1996: 52-66
85. C. Ferdinand, R. Wilhelm: On Predicting Data Cache Behavior for Real-Time Systems. LCTES 1998: 16-30
86. C. Ferdinand, F. Martin, R. Wilhelm, M. Alt: Cache Behavior Prediction by Abstract Interpretation. Sci. Comput. Program. 35(2): 163-189 (1999)
87. C. Ferdinand, R. Wilhelm: Efficient and Precise Cache Behavior Prediction for Real-Time Systems. Real-Time Systems 17(2-3): 131-181 (1999)
88. H. Theiling, C. Ferdinand, R. Wilhelm: Fast and Precise WCET Prediction by Separated Cache and Path Analyses. Real-Time Systems 18(2/3): 157-179 (2000)
89. A. Rakib, O. Parshin, S. Thesing, R. Wilhelm: Component-Wise Instruction-Cache Behavior Prediction. ATVA 2004: 211-229
90. R. Wilhelm: Why AI + ILP Is Good for WCET, but MC Is Not, Nor ILP Alone. VMCAI 2004: 309-322
91. S. Thesing, J. Souyris, R. Heckmann, F. Randimbivololona, M. Langenbach, R. Wilhelm, C. Ferdinand: An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software. Proc. Int. Performance and Dependability Symp.: 625-632
92. C. Ferdinand, D. Kästner, F. Martin, M. Langenbach, M. Sicks, S. Wilhelm, R. Heckmann, Nico Fritz, S. Thesing, F. Fontaine, H. Theiling, M. Schmidt, A. A. Evstiougov-Babaev, R. Wilhelm: Validierung des Zeitverhaltens von kritischer Echtzeit-Software. GI Jahrestagung (1) 2003: 335-339
93. C. Ferdinand, R. Heckmann, H. Theiling, R. Wilhelm: Convenient User Annotations for a WCET Tool. WCET 2003: 17-20
94. R. Wilhelm, J. Engblom, S. Thesing, D.B. Whalley: Industrial Requirements for WCET Tools - Answers to the ARTIST Questionnaire. WCET 2003: 39-43
95. R. Heckmann, M. Langenbach, S. Thesing, R. Wilhelm: The Influence of Processor Architecture on the Design and the Results of WCET Tools. Proc. of the IEEE 91(7): 1038-1054 (2003)
96. C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, R. Wilhelm: Reliable and Precise WCET Determination for a Real-Life Processor. EMSOFT 2001: 469-485
97. C. Ferdinand, D. Kästner, M. Langenbach, F. Martin, M. Schmidt, J. Schneider, H. Theiling, S. Thesing, R. Wilhelm: Run-Time Guarantees for Real-Time Systems - The USES Approach. GI Jahrestagung 1999: 410-419
98. R. Wilhelm: Timing Analysis and Validation for Real-Time Systems - Guest Editor's Introduction. Real-Time Systems 17(2-3): 127-129 (1999)
99. R. Wilhelm: Timing Analysis and Timing Predictability. FMCO 2004: 317-323
100. R. Wilhelm: Formal Analysis of Processor Timing Models. SPIN 2004: 1-4
101. L. Thiele, R. Wilhelm: Design for Timing Predictability. Real-Time Systems 28(2-3): 157-177 (2004)

102. R. Wilhelm: Run-Time Guarantees for Real-Time Systems. FORMATS 2003: 166-167
103. R. Wilhelm: Determining Bounds on Execution Times. In R. Zurawski, editor, Handbook on Embedded Systems, pages 14-1,14-23. CRC Press, 2005.
104. http://www.absint.com/