Annotated Bibliography
Harpoon Project

C. Scott Ananian

June 10, 1999

References

[1] Sungdo Moon, Mary W. Hall, and Brian R. Murphy. Predicated array data-flow analysis for run-time parallelization. In *ICS '98. Conference proceedings of the 1998 international conference on Supercomputing*, pages 204–211, Melbourne, Australia, July 1998. Predication in an array analysis context.

[2] Adam Brooks Webber. Program analysis using binary relations. In *Proceedings of the ACM SIGPLAN '97 Conference on Programming Language Design and Implementation (PLDI)*, pages 249–259, Las Vegas, Nevada, May 1997. Very powerful analysis/optimization technique. Originally presented for LISP basic blocks only. With phi and sigma functions we can extend the analysis to the whole program and incorporate elements of reachability analysis. Analysis extracts relations between pairs of variables, and is very type-flexible. Fairly complicated implementation, however.

[3] Jean-Raymond Gagné and John Plaice. The non-standard semantics of Esterel. In *Advances in Computing Science — ASIAN '97. Third Asian Computing Conference. Proceedings*, pages 381–382, Kathmandu, Nepal, 1997. Presents rationale for using a (time+delta) timeline instead of Esterel's integer-valued instantaneous time model. Claims that Esterel really uses the (time+delta) timeline without realizing it.

[4] Gérard Berry. Esterel on hardware. In *Mechanized Reasoning and Hardware Design*, pages 87–103. Prentice Hall, 1992. a simple hardware implementation of a subset of esterel, using 4 type of fundamental logic blocks. Optimizers based on BDDs. Very useful bibliography.

[5] Gérard Berry. The Esterel v5 language primer. Lots of dry technical detail on the language, but also some enlightening discussion of the rationale behind the design. See especially sections on 'Transformational, Interactive, and Reactive Systems,' 'Write Things Once,' and preemption., March 1998.

[6] Gérard Berry. Hardware and software synthesis, optimization, and verification from Esterel programs. In *Tools and Algorithms for the Construction and Analysis of Systems. Third International Workshop, TACAS '97. Proceedings*, pages 1–3, Enschede, The Netherlands, April 1997. Two-page overview of the Esterel system. At least it has a decent bibliography.

[7] Raj S. Mitra, Bishnupriya Bhattacharya, and Luciano Lavagno. Asynchronous implementation of synchronous Esterel specifications. In *Proceedings. Tenth International Conference on VLSI Design*, pages 348–353, Hyderabad, India, January 1997. In which we learn that Esterel is inherently a synchronous language (surprise!). Semantics have to change to support asynchronous implementation; this paper shows how. Not useful to us.

[8] Pascalin Amagbégnon, Loïc Besnard, and Paul Le Guernic. Implementation of the data-flow synchronous language SIGNAL. In *Proceedings of the ACM SIGPLAN '95 Conference on Programming Language Design and Implementation (PLDI)*, pages 163–173, La Jolla, California, June 1995. Not terribly useful. Intriguing use of the word 'arborescent' for 'tree-structured'.

[9] Tai M. Chung and Hank G. Dietz. Language constructs and transformation for hard real-time systems. *ACM SIGPLAN Notices*, 30(11):41–49, November 1995. Presents a simple set of language-independent extensions to specify real-time constraints. Also contains the interesting claim that the real-time compilation problem has been solved, and references an unpublished paper to prove it.

[10] Daniel Weise, Roger F. Crew, Michael Ernst, and Bjarne Steensgaard. Value dependence graphs: Representation without taxation. In *Proceedings of the 21st ACM Symposium on Principles of Programming Languages (POPL)*, pages 297–310, Portland, Oregon, January 1994. Dataflow representation useful; also recursive/loop structure useful.

[11] Seongsoo Hong and Richard Gerber. Compiling real-time programs into schedulable code. In *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation (PLDI)*, pages 166–176, Albuquerque, New Mexico, June 1993. Code motion for timing optimization. Are all real-time schedulers iterative?

[12] Evelyn Duesterwald, Rajiv Gupta, and Mary Lou Soffa. A practical data flow framework for array reference analysis and its use in optimizations. In *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation (PLDI)*, pages 68–77, Albuquerque, New Mexico, June 1993. Not terribly useful. REREAD LATER.

[13] Cliff Click and Michael Paleczny. A simple graph-based intermediate representation. In *The First ACM SIGPLAN Workshop on Intermediate Representations*, San Francisco, California, January 1995. Interesting examples

of C++ code, but light on optimization results. We really want to peek at Cliff Click's PhD thesis. [Ed note: I found it!].

[14] Vugranam C. Sreedhar and Guang R. Gao. A linear time algorithm for placing ϕ-nodes. In *Proceedings of the 22nd ACM Symposium on Principles of Programming Languages (POPL)*, pages 62–73, San Francisco, California, January 1995. Just what the title says. I need to implement this eventually.

[15] Vugranam C. Sreedhar, Guang R. Gao, and Yong-fong Lee. A new framework for exhaustive and incremental data flow analysis using DJ graphs. In *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation (PLDI)*, pages 278–290, Philadelphia, Pennsylvania, May 1996. Emphasis on Incremental. If we decide to use DJ graphs, this paper could be useful for its examples. Not much new, though, compared to the N other DJ graph papers these authors have cranked out.

[16] Vugranam C. Sreedhar, Guang R. Gao, and Yong-fong Lee. Incremental computation of dominator trees. *ACM Transactions on Programming Languages and Systems*, 19(2):239–252, March 1997. Seems to work well, but I don't think we need incremental updates for Harpoon. Maybe I'm wrong. Very similar to their other papers on this topic. ¡wry grin¿.

[17] Torbjörn Granlund and Peter L. Montgomery. Division by invariant integers using multiplication. In *Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation (PLDI)*, pages 61–72, Orlando, Florida, June 1994. *Very* useful for eliminating logic complexity. Also includes lots of references for deconstructing multiplies. In conjunction with induction variable analysis, some very powerful optimizations are possible. The optimization of the radix conversion code is nothing short of brilliant.

[18] Greg DeFouw, David Grove, and Craig Chambers. Fast interprocedural class analysis. In *Proceedings of the 25th ACM Symposium on Principles of Programming Languages (POPL)*, pages 222–246, San Diego, California, January 1998. This basically tells us that class/type analysis will be very slow. We can make it faster, but we lose much precision. In hardware compilation we don't care so much about speed, perhaps, so maybe we just suck up and deal. Return to this paper before implementing our type inference engine. [theory practice conflict: the idea I've got in my head doesn't seem as slow as the fastest algorithm presented here. Reconcile this.] [Further thought: these algorithms look like they should be integrated with an (SSA?) constant-propagation/dead-code analysis to come up with the most precise class information possible. Will be slow, but worth it. This paper's optimizations result in less class/type precision, which is unacceptable to us.].

[19] Cliff Click. Global code motion / global value numbering. In *Proceedings of the ACM SIGPLAN '95 Conference on Programming Language Design and*

Implementation (PLDI), pages 246–257, La Jolla, California, June 1995. Simple and works, but relies on heuristic to place, and seems overly wedded to basic blocks (which we want to eliminate). I like SSAPRE better, even though it is *much* more complicated. Reassessment: Maybe basic blocks aren't so bad.

[20] Fred Chow, Sun Chan, Robert Kennedy, Shin-Ming Liu, Raymond Lo, and Peng Tu. A new algorithm for partial redundancy elimination based on SSA form. In *Proceedings of the ACM SIGPLAN '97 Conference on Programming Language Design and Implementation (PLDI)*, pages 273–286, Las Vegas, Nevada, May 1997. Powerful but very, very complicated. Probably worth implementing if we stay in SSA form, although dataflow-based methods (see Click's IR) may be simpler. Also non-linear complexity for simple implementations.

[21] Cliff Click and Keith D. Cooper. Combining analyses, combining optimizations. *ACM Transactions on Programming Languages and Systems*, 17(2):181–196, March 1995. Very theoretical==not terribly useful.

[22] Cliff Click. *Combining Analyses, Combining Optimizations*. PhD thesis, Rice University, February 1995. Lots of very useful optimization information, plus a very powerful optimistic analysis algorithm, which I plan to extend into our 'extended type analysis' pass.

[23] Micah Beck, Richard Johnson, and Keshav Pingali. From control flow to dataflow. *Journal of Parallel and Distributed Computing*, 12(2):118–129, June 1991. The paper that started it all. Uses an over-restrictive control model, though, and funky store stuff.

[24] Richard Johnson and Keshav Pingali. Dependence-based program analysis. In *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation (PLDI)*, pages 78–89, Albuquerque, New Mexico, June 1993. An $O(EV)$ algorithm for merge and switch placement.

[25] Richard Johnson, David Pearson, and Keshav Pingali. The program structure tree: Computing control regions in linear time. In *Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation (PLDI)*, pages 171–185, Orlando, Florida, June 1994. An $O(EV)$ algorithm for computing SESE regions.

[26] Richard Johnson, David Pearson, and Keshav Pingali. Finding regions fast: Single entry single exit and control regions in linear time. Technical Report TR 93-1365, Cornell University, Ithaca, NY 14853-7501, July 1993. SESE regions through cycle equivalence. Most complete algorithm description.

[27] Keshav Pingali, Micah Beck, Richard C. Johnson, Mayan Moudgill, and Paul Stodghill. Dependence flow graphs: An algebraic approach to program dependencies. Technical Report TR 90-1152, Cornell University, Ithaca,

NY 14853-7501, September 1990. interesting variation on 'dependence-based program analysis'; defines a formal executable semantics for a DFG-like program representation, with special loop constructs.

[28] Martin C. Rinard and Pedro C. Diniz. Commutivity analysis: A new analysis technique for parallelizing compilers. *ACM Transactions on Programming Languages and Systems*, 19(6):942–991, November 1997. Not terribly relevant to the current project, but interesting nonetheless. Perhaps we can perform some commutivity analysis in Harpoon to free up parallelism?

[29] Johan Janssen and Henk Corporaal. Making graphs reducible with controlled node splitting. *ACM Transactions on Programming Languages and Systems*, 19(6):1031–1052, November 1997. How to make any control flow graph reducible using a minimum number of splits/duplicates. VERY USEFUL if we need reducible graphs. (I suspect that the properties of Java give us this for free.) Good results.

[30] Michael P. Gerlek, Eric Stoltz, and Michael Wolfe. Beyond induction variables: Detecting and classifying sequences using a demand-driven SSA form. *ACM Transactions on Programming Languages and Systems*, 17(1):85–122, January 1995. Very powerful technique for induction variable analysis using a variant of the SSA form. Requires a symbolic math package for full power, but can accomplish classification easily. mu and nu nodes are used in addition to phi nodes; they might be worthwhile subclasses. Min/max bounds can often be computed. Useful useful stuff. Reference to paper on 'loop distribution' and 'loop interchanging' that might be worthwhile to track down. Implementable algorithms.

[31] Frank Mueller and David B. Whalley. Avoiding conditional branches by code replication. In *Proceedings of the ACM SIGPLAN '95 Conference on Programming Language Design and Implementation (PLDI)*, pages 56–66, La Jolla, California, June 1995. Just what the title says. Probably not useful for a hardware compiler. Only mildly useful in its original domain, for that matter.

[32] David Harel and Chaim-arie Kahana. On statecharts with overlapping. *ACM Transactions on Software Engineering and Methodology*, 1(4):399–421, October 1992. Here we basically learn that overlapping statecharts are a Bad Thing. Do not attempt.

[33] Doron Drusinsky and David Harel. Using statecharts for hardware description and synthesis. *IEEE Transactions on Computer-Aided Design*, 8(7):798–807, July 1989. A good description of what statecharts are, and a variety of implementation methodologies using various types of PLAs and FSMs. The infamous traffic light example. Basic idea: implement statecharts as multi-level Mealy/Moore machines. [See Esterel 'latch optimization' paper for how to best create hardware from state machines.].

[34] David Harel and Eran Gery. Executable object modeling with statecharts. *Computer*, 30(7):31–42, July 1997. Not useful at all. A waste of time. Although some of the ideas of mapping methods to events are similar to our approach.

[35] Rajat Aggarwal, Rajeev Murgai, and Masahiro Fujita. Speeding up technology-independent timing optimization by network partitioning. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 83–90, San Jose, California, November 1997. Useful, but presupposes a basic optimizer. Some good references to basic optimizers, which may prove useful. Presents a timing-driven partitioning technique.

[36] Valeria Bertacco and Maurizio Damiani. The disjunctive decomposition of logic functions. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 78–82, San Jose, California, November 1997. Computationally very expensive. Enables creation of a compact, canonical multi-level circuit directly from a BDD representation, which is a Good Thing. "We found the final netlist to be often close to the output of more complex dedicated optimization tools." Probably worth implementing, or at least researching further. BIG DRAWBACK: "For the largest benchmarks, the limited set of BDD transformations... do not compensate for the exceptional growth of the BDD representation with respect to the original representation." I wish I knew what the "original representation" was.

[37] Yuji Kukimoto, Wilsin Gosti, Alexander Saldanha, and Robert K. Brayton. Approximate timing analysis of combinational circuits under the XBD0 model. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 176–181, San Jose, California, November 1997. Good overview of timing analysis. Uses a SAT solver to compute approximate timing. May be useful.

[38] Majid Sarrafzadeh and Maogang Wang. NRG: Global and detailed placement. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 532–537, San Jose, California, November 1997. Not enough detail given to be useful.

[39] Srinivasa R. Arikati and Ravi Varadarajan. A signature based approach to regularity extraction. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 542–545, San Jose, California, November 1997. The signature idea is a good one. Not enough info for an implementation, though. And our regularity information should come from a much higher level. Might be worthwhile referencing to prove that logic synthesis folk spend a lot of effort reconstructing the high-level patterns thrown away by the compiler. Not otherwise useful.

[40] Morgan Enos, Scott Hauck, and Majid Sarrafzadeh. Replication for logic bipartitioning. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 342–349, San Jose, California, November 1997. We can create a better logic partition if we are allowed to duplicate some logic. Good idea and good overview of partitioning methods. VERY USEFUL if we need to partition. We want to use the Strawman algorithm, apparently; we need to look up papers on it.

[41] Charles E. Leiserson, Flavio M. Rose, and James B. Saxe. Optimizing synchronous circuitry by retiming. In R. Bryant, editor, *Third Caltech Conference on Very Large Scale Integration*, pages 87–116, Pasadena, California, March 1983. Three good references on other optimization types. This paper describes how to optimize register placement to obtain the smallest possible clock cycle time. *DON'T HAVE COMPLETE PAPER*. Useful. Includes algorithms to compute the clock period of a circuit, among other things.

[42] Sharad Malik, Ellen M. Sentovich, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. *IEEE Transactions on Computer-Aided Design*, 10(1):74–84, January 1991. Useful extension of Leiserson's Retiming paper. Perhaps more information can be found in malik93? Basic idea is that we can push registers to the periphery and disconnect cycles, optimize the resulting circuit, and then push all the registers back in and reconnect the feedback loops. Makes sense. We still have to find a valid combinational optimization technique to plug in the middle, though.

[43] Jianzhong Shi, Akash Randhar, and Dinesh Bhatia. Macro block based FPGA floorplanning. In *Proceedings. Tenth International Conference on VLSI Design*, pages 21–26, Hyderabad, India, January 1997. Interesting paper when we get to floorplanning our hardware. Authors note that it is difficult to find hierarchically-structured designs/testcases; hopefully our compiler will change this somewhat.

[44] Madhav Y. Chikodikar, Shridhar Laddha, and Ashish Sirasao. A technology mapper for Xilinx FPGAs. In *Proceedings. Tenth International Conference on VLSI Design*, pages 57–61, Hyderabad, India, January 1997. "This paper presents a method for area optimal technology mapping for Xilinx FPGAs." Maps logic onto CLBs; 10% better than the Xilinx method. If we ever need to do this, this is the method to use. [Hopefully we can use their code!].

[45] C.P. Ravikumar and R. Aggarwal. A graph-theoretic approach for register file based synthesis. In *Proceedings. Tenth International Conference on VLSI Design*, pages 118–123, Hyderabad, India, January 1997. In-house know-how! Interesting paper; replace registers with register *files*... complete algorithms. Meant for datapath synthesis. "[A]ssumes as input a scheduled data flow graph.".

[46] Kimberly D. Emerson. Asynchronous design—an interesting alternative. In *Proceedings. Tenth International Conference on VLSI Design*, pages 318–320, Hyderabad, India, January 1997. Good overview of asynch. design and the rationale behind it. Briefly: clock skew, power consumption, handling of metastability, and elimination of critical path delay. Circuits run in 'average case' time instead of 'worst case' time.

[47] Fu-Chiung Cheng, Stephen H. Unger, Michael Theobald, and Wen-Chung Cho. Delay-insensitive carry-lookahead adders. In *Proceedings. Tenth International Conference on VLSI Design*, pages 322–328, Hyderabad, India, January 1997. Pretty typical example of async design, using dual-rail signalling. Proposes DICLASP adder, compares to DIRCA adder. O(n) logic complexity, O(log log n) average case time complexity. About twice as transistor-expensive as the standard O(n) * O(n) synchronous ripple-carry adder.

[48] K. Nanda, S. K. Desai, and S. K. Roy. A new methodology for the design of asynchronous digital circuits. In *Proceedings. Tenth International Conference on VLSI Design*, pages 342–347, Hyderabad, India, January 1997. Proposes a way-out revamp of logic design for async systems. Replaces the conventional dual-rail system with a "non-return-to-zero event driven" scheme, and invents a "universal gate" which acts as and/or/not for this system. Completely delay-insensitive; *however* very area expensive. Might be useful for automated async design, though.

[49] Scott Hauck. Asynchronous design methodologies: An overview. Technical Report TR-93-05-07, University of Washington, May 1993. Very good overview of state of the art in asynchronous design. As of 1993, there were significant problems with each async design methodology presented. Is there recent work overcoming some of these difficulties?

[50] Bernd Becker and Rolf Drechsler. Decision diagrams in synthesis: Algorithms, applications, and extensions. In *Proceedings. Tenth International Conference on VLSI Design*, pages 46–50, Hyderabad, India, January 1997. Excellent overview of BDDs and its related cousins. Lengthy bibliography. Also overview the extensions to BDDs to handle (for example) multipliers.

[51] Rolf Drechsler. Pseudo kronecker expressions for symmetric functions. In *Proceedings. Tenth International Conference on VLSI Design*, pages 511–513, Hyderabad, India, January 1997. An optimizer based of PSDKROs for symmetric functions is described; seems to work pretty well on non-symmetric functions, too. Tenth-of-a-second execution times; 1000x faster than comparable methods. Go Go Go!

[52] Rakefet Kol and Ran Ginosar. Kin: A high performance asynchronous processor architecture. In *ICS '98. Conference proceedings of the 1998 international conference on Supercomputing*, pages 433–440, Melbourne, Australia, July 1998. Why asynchronous design will take over the world.

[53] David I. August, Wen mei W. Hwu, and Scott A. Mahlke. A framework for balancing control flow and predication. In *MICRO 30. Proceedings of the thirtieth annual IEEE/ACM international symposium on Microarchitecture*, pages 92–103, Research Triangle Park, North Carolina, December 1997. Interesting IR with predication bits.

[54] Jaejin Lee, David A. Padua, and Samuel P. Midkiff. Basic compiler algorithms for parallel programs. In *Proceedings of the 7th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, pages 1–12, Atlanta, Georgia, May 1999. SSA form for explicitly parallel programs.

[55] Shih-Wei Liao, Amer Diwan, Robert P. Bosch Jr., Anwar Ghuloum, and Monica S. Lam. SUIF explorer: An interactive and interprocedural parallelizer. In *Proceedings of the 7th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, pages 37–48, Atlanta, Georgia, May 1999. Interprocedural SSA form.

[56] Kenneth R. Traub. A compiler for the MIT tagged-token dataflow architecture. Technical Report MIT/LCS/TR-370, Massachusetts Institute of Technology, Cambridge, MA 02139, August 1986. Very interesting, historically. Intraloop, not interloop, parallelism.

[57] Paul Havlak. *Interprocedural Symbolic Analysis*. PhD thesis, Rice University, Houston, Texas, May 1994. TGSSA form.

[58] Jong-Deok Choi, Ron Cytron, and Jeanne Ferrante. Automatic construction of sparse data flow evaluation graphs. In *Proceedings of the 18th ACM Symposium on Principles of Programming Languages (POPL)*, pages 55–66, Orlando, Florida, January 1991. Definition and algorithm for "pruned" SSA form. Incorrectly referenced by [66].

[59] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE: Another look at real time programming*, 79(9):1270–1282, September 1991. Referenced by [4] as source of the *perfect synchrony hypothesis*.

[60] Jean-Raymond Gagné and John Plaice. A non-standard temporal deductive database system. *Journal of Symbolic Computation*, 22(5-6):649–664, November–December 1996. Referenced by [3] to support the idea of "non-standard numbers" in ennumerating time.

[61] David Galloway. The Transmogrifier C hardware description language and compiler for FPGAs. In *IEEE Symposium on FPGAs for Custom Computing Machines. Proceedings*, pages 136–144, April 1995.

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

[63] Nabeel Shirazi, Al Walters, and Peter Athanas. Quantitative analysis of floating point arithmetic on FPGA based Custom Computing Machines. In *IEEE Symposium on FPGAs for Custom Computing Machines. Proceedings*, pages 155–162, Napa Valley, California, April 1995.

[64] Andrew W. Appel. *Modern Compiler Implementation in Java.* Cambridge University Press, 1998. The standard practical compilers text.

[65] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification.* Addison-Wesley, September 1996.

[66] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, October 1991. One of the canonical SSA form papers.

[67] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. An efficient method of computing static single assignment form. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages (POPL)*, pages 25–35, Austin, Texas, January 1989. The other canonical SSA form paper.

[68] Ron K. Cytron and Jeanne Ferrante. Efficiently computing ϕ-nodes on-the-fly. *ACM Transactions on Programming Languages and Systems*, 17(3):487–506, May 1995. Contains: "Since one reason for introducing ϕ-functions is to eliminate potentially quadratic behavior when solving actual data flow problems, such worst-case behavior during SEG or SSA construction could be problematic. Clearly, avoiding such behavior necessitates placing ϕ-functions without computing or using dominance frontiers.".

[69] B. Alpern, Mark N. Wegman, and F. Kenneth Zadeck. Detecting equality of variables in programs. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages (POPL)*, pages 1–11, San Diego, California, January 1988. Earliest SSA reference?

[70] Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Global value numbers and redundant computations. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages (POPL)*, pages 12–27, San Diego, California, January 1988.

[71] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981. Referenced in [27].

[72] Robert M. Shapiro and H. Saint. The representation of algorithms. Technical Report CA-7002-1432, Massachusetts Computer Associates, February 1970. Referenced in [24] as the original SSA form paper.

[73] Fred Chow, Sun Chan, Shin-Ming Liu, Raymond Lo, and Mark Streich. Effective representation of aliases and indirect memory operations in SSA form. In *Proceedings of the Sixth International Conference on Compiler Construction*, pages 253–267, April 1996. Referenced in [20] for HSSA form; also documentation of SGI's use of SSA-variant in their production MIPSpro compiler (release 7.2).

[74] Robert A. Ballance, Arthur B. Maccabe, and Karl J. Ottenstein. The Program Dependence Web: A representation supporting control-, data-, and demand-driven interpretation of imperative languages. In *Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation (PLDI)*, pages 257–271, White Plains, New York, June 1990. PDW canonical reference.

[75] Peng Tu and David Padua. Efficient building and placing of gating functions. In *Proceedings of the ACM SIGPLAN '95 Conference on Programming Language Design and Implementation (PLDI)*, pages 47–55, La Jolla, California, June 1995. Efficiently constructing GSSA form.

[76] J. H. Reif and R. E. Tarjan. Symbolic program analysis in almost-linear time. *SIAM Journal on Computing*, 11(1):81–93, February 1981. Introduction of "birthpoints," which Havlek [57] claims was the direct ancestor of SSA form.