Automated Development of Software for Program Analysis and Transformation

Deepak Goyal
Department of Computer Science
New York University
251 Mercer Street, New York, NY 10012
e-mail: deepak@cs.nyu.edu

Y. Annie Liu
Computer Science Department
Indiana University
150 S. Woodlawn Ave., Bloomington, IN 47405
e-mail: liu@cs.indiana.edu

Bob Paige was the Principle Investigator of this project. Due to his recent tragic death, we are reporting on it.

The objectives of Paige's research were to improve software productivity for, and reliability and performance of, complex systems through transformational programming. Over the past twenty years, contributions were made to the area of program transformation along two distinct tracks: (1) general methods for transforming high-level problem specifications into efficient implementations, and (2) special methods and systems for implementing the program analyses and transformations used in Track (1). The long-term goal was to combine both tracks by specifying the complex systems in Track (2) at a high level and transforming them into efficient implementations using the methods in Track (1).

To address both correctness and efficiency of systems, a specification language must be *wide-spectrum*, i.e., capable of expressing both abstract specifications and low-level efficient implementations, and *computationally transparent*, i.e., able to facilitate accurate estimation of time and space requirements. For example, C is computationally transparent but not wide-spectrum, while SETL [13] is wide-spectrum but not computationally transparent. A wide-spectrum language provides a uniform notation for both specifications and their implementations, which is helpful for proving that the transformations preserve semantics. Computational transparency allows the selection of transformations to be guided by complexity considerations and, thus, helps in getting more efficient implementations.

Contributions to Track (1) include the formal definition of a wide-spectrum set-theoretic specification language called Typed SETL, which is made computationally transparent by associating asymptotic complexities with syntactic constructs, and the development of a three-stage transformational methodology: *dominated convergence*, *finite differencing*, and *real-time simulation*. Dominated convergence transforms very-high-level fixed-point specifications into iterative programs. Finite differencing replaces repeated costly computations on sets and maps by cheaper incremental counterparts involving restricted primitive operations. Real-time simulation selects appropriate data representations so that each restricted primitive operation, such as set membership testing, is performed in guaranteed constant time.

The specification language Typed SETL and the three-stage transformational methodology have been shown to be effective for explaining existing algorithms [8, 11, 2, 12, 5] and even discovering new improved algorithms [10, 1, 7, 4, 6, 5]. Perhaps the most compelling evidence that this transformational methodology can provide a dramatic improvement in the productivity of complex high-performance systems may be found in the experiments by Cai and Paige [3]. In that work, a simple but conservative model of productivity was developed, and within the model, a five-fold improvement in productivity of high-performance implementation in C in comparison to hand-coding in C was demonstrated. These experiments were, however, limited to small-scale examples.

Contributions to Track (2) include the development of a prototype for implementing the methodology of Track (1). This includes a complex translation of a statically typed variant of SETL into C that was used to conduct the productivity experiments mentioned above. This translation was implemented using the APTS program transformation system [9] in a high-level language called RSL (Rule Specification Language). The efficiency of the generated C code was comparable to that of hand-coded C. However, the translation process was inefficient, for two reasons: firstly, APTS is implemented in SETL2 [14] which runs about thirty times slower than C at best and, secondly, APTS provides only an interpretive implementation for RSL.

Paige's future research plans focused on rectifying these two inefficiencies. This included rewriting the RSL code for translating SETL to C to accept the full Typed SETL, and rewriting the APTS system in Typed SETL. This would help obtain an efficient implementation of APTS in C, thus eliminating the first inefficiency. The second inefficiency may be addressed using partial evaluation. This would require a partial evaluator for Typed SETL to be written in Typed SETL. The partial evaluation of APTS with respect to the RSL code for the translator would produce a compiled implementation of the translator in Typed SETL, which could itself be translated into an efficient implementation in C.

References

[1] B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24(3):189–220, 1995.

[2] J. Cai and R. Paige. Program derivation by fixed point computation. *Science of Computer Programming*, 11:197–261, 1988/89.

[3] J. Cai and R. Paige. Towards increased productivity of algorithm implementation. In *Proc. ACM SIGSOFT'93 Symp. on Foundations of Soft. Eng.*, pages 71–78, Dec. 1993.

[4] C.-H. Chang and R. Paige. From regular expressions to DFA's using compressed NFA's. *Theoretical Computer Science*, 178(1–2):1–36, May 1997.

[5] D. Goyal. *A Language-Theoretic Approach to Algorithms.* PhD thesis, New York University, Jan. 2000. Advisor: Bob Paige.

[6] D. Goyal and R. Paige. The formal reconstruction and speedup of the linear time fragment of Willard's relational calculus subset. In Bird and Meertens, editors, *Algorithmic Languages and Calculi*, pages 382–414. Chapman and Hall, 1997.

[7] J. Keller and R. Paige. Program derivation with verified transformations – a case study. *Communications on Pure and Applied Mathematics*, 48(9–10):1053-1113, 1995.

[8] R. Paige. *Formal Differentiation: A Program Synthesis Technique.* UMI Research Press, 1981. Revision of Ph.D. thesis, New York University, Jun. 1979.

[9] R. Paige. Viewing a program transformation system at work. In *Proc. Joint 6th Intl. Conf. on Prog. Lang. Impl. and Logic Prog. and 4th Intl. Conf. on Algebraic and Logic Prog.*, volume 844 of *Lecture Notes in Computer Science*, pages 5–24, Sep. 1994.

[10] R. Paige and F. Henglein. Mechanical translation of set theoretic problem specifications into efficient RAM code - a case study. *Journal of Symbolic Computation*, 4(2):207–232, Aug. 1987.

[11] R. Paige, R. Tarjan, and R. Bonic. A linear time solution to the single function coarsest partition problem. *Theoretical Computer Science*, 40(1):67–84, Sep. 1985.

[12] R. Paige and Z. Yang. High level reading and data structure compilation. In *Proc. 24th ACM SIGPLAN-SIGACT Symp. on Principles of Prog. Lang.*, pages 456–469, Paris, France, 15–17 Jan. 1997.

[13] J. Schwartz, R. Dewar, E. Dubinsky, and E. Schonberg. *Programming with Sets: An Introduction to SETL.* Springer-Verlag, New York, 1986.

[14] K. Snyder. The SETL2 programming language. Technical Report 490, Courant Institute, New York University, 1990.