DFA&OPT-MetaFrame: A Tool Kit for Program Analysis and Optimization

Marion Klein* Jens Knoop† Dirk Koschützki† Bernhard Steffen†

ABSTRACT Whereas the construction process of a compiler for the early and late phases like syntactic analysis and code generation is well-supported by powerful tools, the *optimizer*, the key component for achieving highly efficient code is usually still hand-coded. The tool kit presented here supports this essential step in the construction of a compiler. The two key features making it exceptional are (1) that it automatically generates global program analyses for intraprocedural, interprocedural and parallel data flow problems, and (2) that it supports the combination of the results obtained to program optimizations.

1 Overview

Compilers are expected to produce *highly efficient* code. Thus, *optimizers* are integrated in order to detect and remove inefficiencies in application programs. Typically, optimization proceeds in two steps: First, a program analysis, usually a *data flow analysis (DFA)*, which detects the side conditions under which an optimizing program transformation is applicable, and second, the concrete transformation based on the data flow facts computed by the preceding analysis. The algorithms realizing these two steps are usually still hand-coded. As the construction process for essentially every other phase of compilation is well-supported by powerful tools, the construction of the optimizer still belongs to the most expensive, time consuming, and error prone steps in the construction of a compiler.

The DFA&OPT-MetaFrame tool kit supports this essential step of compiler construction. It *automatically* generates efficient DFA-algorithms from concise specifications given in a *modal logic* (cf. [St]). In essence, the DFA-generator of the tool kit works by *partially evaluating* an appropiate *model checker* with respect to the modal formula specifying the data flow property

*Lehrstuhl für Informatik II, Rheinisch-Westfälische Technische Hochschule Aachen, Ahornstraße 55, D-52056 Aachen, Germany E-mail: marion@informatik.rwth-aachen.de

†Fakultät für Mathematik und Informatik, Universität Passau, Innstrasse 33, D-94032 Passau, Germany. E-mail: {knoop | koschuetzki | steffen}@fmi.uni-passau.de

of interest (cf. [SCKKM]). The result is a usual iterative DFA-algorithm, which runs on the machine the model checker is implemented on, and which can immediately be integrated into the compiler under construction. A *high level programming language* allows to combine the results of different analyses to optimizing program transformations. It serves as the connecting link for combining program analysis and optimization, such that the tool kit supports the complete process of the optimizer construction.

The benefits of this approach are as follows: The DFA-algorithms required are directly specified in terms of the data flow properties of interest. All the details about the corresponding computation procedures are hidden in the tool kit. This yields concise high-level specifications, simplifies and structures the specification development, and supports the reasoning about features such as correctness and optimality of the DFAs. In fact, the DFA-algorithms required by the program optimizations considered below result from two to five line specifications in a modal logic. They are not only significantly shorter, but also more intuitive than their traditionally specified counterparts. Moreover, our practical experience shows that the generated DFA-algorithms are as efficient as their hand-coded counterparts. Summarizing, we profit from:

- *Concise specifications* directly in terms of the data flow properties
- *Combining* global program analysis (DFA) and optimization
- *Simple reasoning* about DFA and optimization on a very high level
- *Hiding* of all details of the computation procedure
- *No efficiency penalty* in comparison to hand-coded algorithms
- *High flexibility* supporting *rapid prototyping*

2 Screen Shots from a Sample Session

We illustrate the usage of the tool kit by means of two screen shots from a sample session. The optimization considered is to remove all *partially redundant computations* in a program (in the example of '$a+b$') by means of the *busy code motion (BCM)* transformation of [KRS1]. This transformation requires the computation of all program points being *down-safe* and *up-safe* for a computation, here '$a + b$'. The results of the corresponding DFA-algorithms, which are automatically generated from the specifications shown in the lower left window, are displayed in the right window of Figure 1, which shows the argument program in an automatically generated and layouted transition system like representation. The states represent program points, and the transitions the control flow and the basic blocks of the underlying procedure. The analysis and optimization process is controlled by means of the high level language, whose commands are executed by an interpreter running in the upper left window.

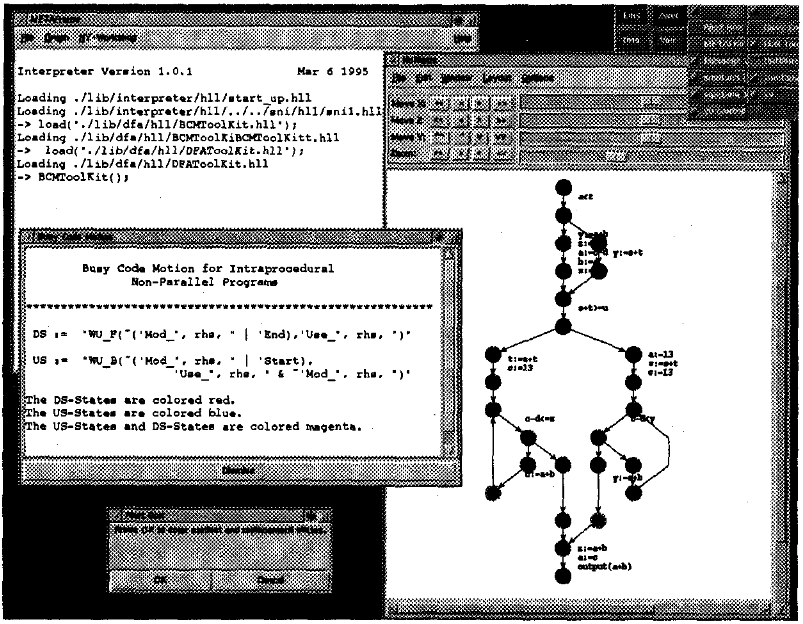

FIGURE 1. First Screen Shot

After computing the set of up-safe and down-safe program points, the *BCM*-transformation can be specified within the high level language: (1) Initializing a new temporary **h** by $a + b$ at all down-safe program points which have an incoming edge modifying a or b, or an unsafe predecessor. (2) Replacing all original occurrences of $a + b$ by **h**. The result of the complete transformation is displayed in the right window of Figure 2.

3 Scope and Current State

The DFA&OPT-METAFrame tool kit supports data flow analysis and optimization of intraprocedural, interprocedural and parallel programs (cf. [KS, KSV1, KSV2, St]). It is particularly well-suited for optimizations based on bitvector analyses, which are most relevant in practice due to their broad scope of powerful and practically relevant optimizations ranging from *code motion* over *assignment motion* and *partial dead code elimination* to *strength reduction* and (via definition-use chains) to *constant propagation* and *constant folding*. Moreover, all these techniques can be performed in the parallel and interprocedural program setting as efficiently as in the sequential intraprocedural one.

The intraprocedural sequential case is fully implemented in the current prototype of our tool kit, i.e., the program analyses are directly generated from their modal logic specifications (cf. [SCKKM]). Interprocedural and parallel DFA-problems are specified in terms of local semantic functionals

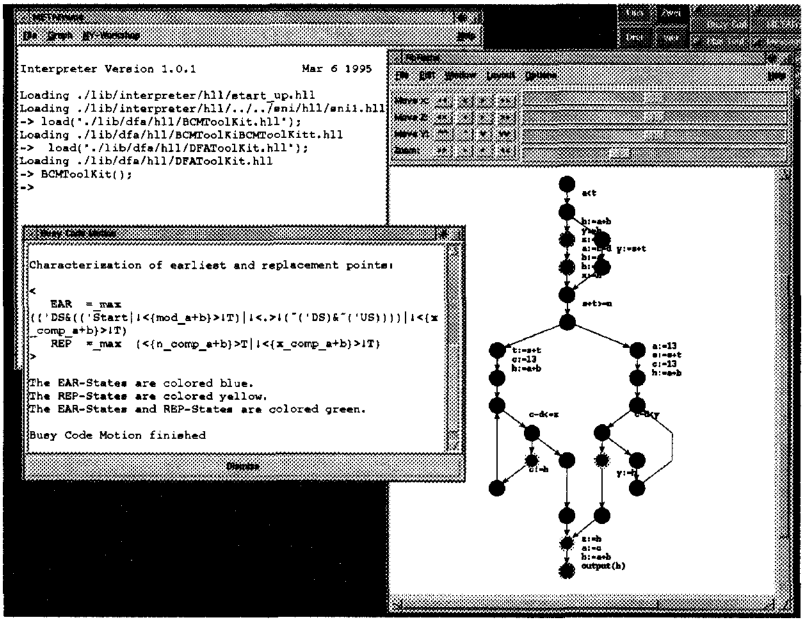

FIGURE 2. Second Screen Shot

giving abstract semantics to statements, the direction of data flow, and the kind of fixpoint desired as it is common in classical data flow analysis. For the interprocedural setting the current version supports programs composed of procedures with global variables. An extension to local variables, and value and reference parameters is in progress. The tool kit has successfully been tested on the program optimizations mentioned above.

4 Related Work

Early approaches to the automatic generation of optimizers concentrated on peephole optimizations, which do not require global program analyses (cf. [DF, Ke]). These approaches are contrasted by others which address the generation of global analyses, but do not support the construction of program transformations (cf. [AM, YH]). Both steps, i.e., global analyses and program transformations, are supported by the systems of [VF, WS], which support intraprocedural optimization. Whereas the system of [VF] concentrates on 'classical' intraprocedural optimizations, the one of [WS] is particularly well-suited for local transformations based on data dependency information which are important for the automatic parallelization of sequential programs. The generality of the DFA&OPT-MetaFrame tool kit that it works for intraprocedural, interprocedural, and parallel programs, and supports the generation of global program analyses and their combination to optimizations, is in fact exceptional.

5 References

[AM] Alt, M., and Martin, F. Generation of efficient interprocedural analyzers with PAG. In Proc. 2^{nd} Internat. Static Analysis Symposium (SAS'95), Glasgow, UK, Springer-Verlag, LNCS 983 (1995), 33 - 50.

[DF] Davidson, J. W., and Fraser, C. W. Automatic generation of peephole transformations. In *Proc. ACM SIGPLAN'84 Symp. on Comp. Construct.*, Montreal, Canada, *SIGPLAN Notices 19*, 6 (1984), 111 - 115.

[Ke] Kessler, R. R. Peep – An architectural description driven peephole transformer. In *Proc. ACM SIGPLAN'84 Symp. on Comp. Construct.*, Montreal, Canada, *SIGPLAN Notices 19*, 6 (1984), 106 - 110.

[KRS1] Knoop, J., Rüthing, O., and Steffen, B. Optimal code motion: Theory and practice. *Transactions on Programming Languages and Systems 16*, 4 (1994), 1117 - 1155.

[KRS2] Knoop, J., Rüthing, O., and Steffen, B. The power of assignment motion. In *Proc. ACM SIGPLAN'95 Conf. on Programming Language Design and Implementation (PLDI'95)*, La Jolla, California, *SIGPLAN Notices 30*, 6 (1995), 233 - 245.

[KS] Knoop, J., and Steffen, B. The interprocedural coincidence theorem. In *Proc. 4^{th} Internat. Conference on Compiler Construction (CC'92)*, Paderborn, Germany, Springer-Verlag, LNCS 641 (1992), 125 - 140.

[KSV1] Knoop, J., Steffen, B., and Vollmer, J. Parallelism for free: Efficient and optimal bitvector analyses for parallel programs. Accepted for *Transactions on Programming Languages and Systems.*

[KSV2] Knoop, J., Steffen, B., and Vollmer, J. Parallelism for free: Bitvector analyses $\Rightarrow$ No state explosion! In *Proc. 1^{st} Internat. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, Springer-Verlag, LNCS 1019 (1995), 264 - 289.

[St] Steffen, B. Generating data flow analysis algorithms from modal specifications. *Science of Computer Programming 21*, (1993), 115 - 139.

[SCKKM] Steffen, B., Claßen, A., Klein, M., Knoop, J., and Margaria, T. The fixpoint-analysis machine. In *Proc. 6^{th} Internat. Conference on Concurrency Theory (CONCUR'95)*, Philadelphia, Pennsylvania, Springer-Verlag, LNCS 962 (1995), 72 - 87.

[VF] Venkatesh, G. V., and Fischer, C. N. Spare: A development evironment for program analysis algorithms. In *IEEE Transactions on Software Engineering 18*, 4 (1992), 304 - 318.

[WS] Whitfield, D., and Soffa, M. L. Automatic generation of global optimizers. In *Proc. ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI'91)*, Toronto, Ontario, Canada, *SIGPLAN Notices 26*, 6 (1991), 120 - 129.

[YH] Yi, K., and Harrison III, W. L. Automatic generation and management of interprocedural program analyses. In ACM SIGPLAN-SIGACT, (Jan. 1993).