Refinement of Graph Programs: From Specifications to Fast Code

PhD Project Proposal

Detlef Plump

This project will contribute to the departmental research theme Critical Systems in that it develops a refinement-based methodology for the design, verification and implementation of critical programs working on graph-like structures. Examples for such programs are classical graph algorithms, pointer manipulating programs in languages such as C [4, 5], and security-critical systems with graph-based access policies [9, 1].

Critical programs will be developed within the rule-based graph programming language GP 2 [10], starting with abstract non-deterministic programs which can be seen as specifications. These will be refined to fast deterministic programs by correctness-preserving refinement steps which gradually reduce the degree of non-determinism. GP 2 has a simple syntax and semantics to facilitate formal reasoning (see [11] for a Hoare logic approach to verifying graph programs) and comes with a compiler which generates efficient C code [2, 3].

Graph programs can be non-deterministic for two reasons: to apply a set of rules, a rule from the set is chosen, and to apply a rule, a match for the rule is chosen. Non-determinism allows concise programs because rule selection and host graph traversal are left to GP 2's implementation. Such programs are typically easier to verify than programs whose non-determinism has been restricted by encoding strategies for rule selection and graph traversal. In the setting of imperative programs, the idea of hiding implementation detail by non-determinism goes back to Floyd who introduced a choose operator to concisely express backtracking algorithms [8]. Similarly, [6] extends the Scala language with a choose operator and refines non-deterministic programs to efficient deterministic programs. In contrast, GP 2 can directly be used as a wide-spectrum language which encompasses both non-deterministic and deterministic programs.

The advantages of highly non-deterministic graph programs come at the cost of slow running times: in general, the time to match a rule is polynomial in the size of the host graph. To overcome this problem, *rooted* GP 2 programs rely on distinguished root nodes in rules and host graphs which can be accessed in constant time. For example, [3] uses a rooted program to check in linear time whether input graphs (of bounded node degree) are 2-colourable. Timing experiments demonstrate that on grid graphs of up to 100,000 nodes, the GP 2 program matches the run time of Sedgewick's tailor-made C program for 2-colourability [12].

The goal of this project is to develop a methodology for refining non-deterministic unrooted graph programs to deterministic rooted programs. Initially, the development process of the 2-colouring program in [3] will be followed by introducing root nodes and marked edges to derive an efficient depth-first graph traversal program. Case studies on linear-time and polynomial-time graph algorithms will be conducted, based on algorithms found in textbooks such as [12, 7, 13]. In each case, the algorithm will be specified by an unrooted graph program which subsequently is refined to a fast rooted program. The running times of the resulting programs will be experimentally compared with published imperative programs for the algorithms. More complicated algorithms may require to encode data structures such as priority queues within host graphs. Subsequently, the methodology will be evaluated in case studies on pointer programs and on programs controlling graph-based access policies in security-critical systems.

References

[1] S. Alves and M. Fernández. A graph-based framework for the analysis of access control policies. *Theoretical Computer Science*, 685:3–22, 2017.

[2] C. Bak. *GP 2: Efficient Implementation of a Graph Programming Language.* PhD thesis, Department of Computer Science, University of York, 2015.

[3] C. Bak and D. Plump. Compiling graph programs to C. In *Proc. International Conference on Graph Transformation (ICGT 2016)*, volume 9761 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2016.

[4] A. Bakewell, D. Plump, and C. Runciman. Checking the shape safety of pointer manipulations. In *Int. Seminar on Relational Methods in Computer Science (RelMiCS 7), Revised Selected Papers*, volume 3051 of *Lecture Notes in Computer Science*, pages 48–61. Springer-Verlag, 2004.

[5] A. Bakewell, D. Plump, and C. Runciman. Specifying pointer structures by graph reduction. In *Applications of Graph Transformations With Industrial Relevance (AGTIVE 2003), Revised Selected and Invited Papers*, volume 3062 of *Lecture Notes in Computer Science*, pages 30–44. Springer-Verlag, 2004.

[6] S. Barman, R. Bodík, S. Chandra, J. Galenson, D. Kimelman, C. Rodarmor, and N. Tung. Programming with angelic nondeterminism. In *Proc. Symposium on Principles of Programming Languages (POPL 2010)*, pages 339–352. ACM, 2010.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, third edition, 2009.

[8] R. W. Floyd. Nondeterministic algorithms. *Journal of the ACM*, 14(4):636–644, 1967.

[9] M. Koch, L. Mancini, and F. Parisi-Presicce. Graph-based specification of access control policies. *Journal of Computer and System Sciences*, 71(1):1 – 33, 2005.

[10] D. Plump. The design of GP 2. In *Proc. Workshop on Reduction Strategies in Rewriting and Programming (WRS 2011)*, volume 82 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–16, 2012.

[11] C. M. Poskitt. *Verification of Graph Programs.* PhD thesis, University of York, 2013.

[12] R. Sedgewick. *Algorithms in C. Part 5: Graph Algorithms.* Addison-Wesley, third edition, 2002.

[13] S. S. Skiena. *The Algorithm Design Manual.* Springer, second edition, 2008.