Formalizing Static Analysis Techniques with Kleene Algebra

Therrezinha Fernandes

Département d'informatique et de génie logiciel
Université Laval
Québec, QC, G1K 7P4 Canada
Therrezinha.Fernandes@ift.ulaval.ca

1 Research thesis and motivations

Static program analysis consists of compile-time techniques for determining properties of programs without actually running them [1–3]. Among the applications of static program analysis are the optimization by compilers of object code [4] and the detection of malicious code or code that might be maliciously exploited [5, 6]. The obvious relevance and (sometimes critical) importance of such applications explain the many attempts to try to understand the full picture of static program analysis and constitute our main motivation to concentrate our research effort in the area of the development of general frameworks for static program analysis. Because another goal we have is the development of the theory and applications of Kleene algebra (KA) [7–9], we were naturally led to the subject of our research: the development of an algebraic framework for static program analysis based on Kleene algebra.

1.1 Related work

Other authors have used Kleene algebra for the purpose of conducting static analysis of programs. In particular, in [10], Kot and Kozen present an approach to abstract interpretation [1] of programs based on KA. In contrast with their approach, which is a second-order one where the object of prime interest is not the information about the execution state, but the transfer functions that describe how the state is transformed by an instruction, our approach focuses on the dataflow information in order to provide concise and readable equations characterizing the considered analyses.

2 Kleene algebra

Definition 1. *A* Kleene algebra *(KA) [8] is a structure* $\mathcal{K} = (K, +, \cdot, ^*, 0, 1)$ *such that* $(K, +, 0)$ *is an idempotent commutative monoid,* $(K, \cdot, 1)$ *is a monoid and the unary operator* * *acts like the Kleene closure of the algebra of regular expressions.*

Models of KA include (min,+) algebras, Boolean algebras, algebras of languages over an alphabet and algebras of relations over a set.

One interesting property of KA is that the set of matrices of size $n \times n$ over a KA is itself a KA. We use this property in order to develop our algebraic framework: the static analyses of interest will be characterized by means of an algebraic calculus of matrices of size $n \times n$ over a KA.

3 Partial results

In recent papers [11, 12], we have presented an approach based on KA for the static dataflow analysis of programs. With this approach, it is possible to compute the precise "meet-over-all-paths" (MOP) [2, 3] solutions to a general class of intraprocedural dataflow analysis problems. This class consists of all problems which are instances of the bit vector framework. This class has a broad scope of application in program code optimization like for instance code motion and partial dead-code elimination. In [11], we illustrated our approach by formalizing four "gen/kill" analyses. The result of this exercise is a very concise and very readable set of equations characterizing the analyses and exhibiting the dualities between them in a clear manner. We provided the sets of equations for two different representations of programs, one in which the statements label the nodes of a control flow graph (CFG) and one in which the statements label the transitions. We then formally described how the data flow equations for the two representations are related. As an illustration, the equations for the classical *reaching definitions analysis* [4], conducted on a CFG representation of a program, are the following:

$$\begin{aligned} O &= S^* \cdot g \cdot (S \cdot \neg k)^* \\ I &= O \cdot S \end{aligned}$$

where S, g and k, the input to the analysis, are matrices representing respectively the structure of the program, what is generated and what is killed at each label, and the matrices O and I, the output of the analysis, are such that $O[i, j]$ (resp. $I[i, j]$) is the set of data flow information outputted from (resp. inputted at) node j by the sequences of instructions occurring on the paths between i and j.

In [12], we proved the soundness of our KA-based approach with respect to the standard iterative approach and mentioned how it was possible to implement an algorithm as efficient as the classical one.

Since then, we have been working on the generalization of this approach to a greater class of analysis problems, such as non-separable and non-bit-vector problems, while trying to stay at least as efficient as their classical counterparts.

4 Expected contributions

Using Kleene algebra we seek to come up with a general framework for the formalization of static program analysis problems. We wish to give a full formal development for those problems, including proofs of their correctness, in a cookbook style. The framework will permit to represent both the programs and the relevant properties in an homogeneous, compact and readable way and traditional algorithms used to compute the result of an analysis will be replaced by algebraic manipulations of elements of a Kleene algebra. This will thus allow an elegant, yet rigorous and non-ambiguous treatment of the analyses while benefiting from a deductive system for the discovery and formal proof of interesting properties. We also seek to present several case studies.

We hope that Kleene algebra will be expressive enough to allow the formalization of a large class of relevant static program analysis problems and that computation within our framework will be efficient. Thus, using our method, all the well-known algorithms for the problems of that class could be formalized in a Kleene algebra framework at almost no additional cost on the runtime side while providing concise specifications of the analyses.

References

1. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis by construction or approximation of fixpoints. In: Proceedings of POPL'77, Los Angeles, California, ACM Press (1977) 238–252
2. Kam, J.B., Ullman, J.D.: Monotone data flow analysis frameworks. Acta Informatica **7** (1977) 309–317
3. Kildall, G.: A unified approach to global program optimization. In: Proceedings of the 1st Annual ACM Symposium on Principles of Programming Languages. (1973) 194–206
4. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques and Tools. Addison-Wesley (1986)
5. Lo, R.W., Levitt, K.N., Olsson, R.A.: MCF: A malicious code filter. Computers and Security **14**(6) (1995) 541–566
6. Bergeron, J., Debbabi, M., Desharnais, J., Erhioui, M.M., Lavoie, Y., Tawbi, N.: Static detection of malicious code in executable programs. In: 1st Symposium on Requirements Engineering for Information Security, Indianapolis, IN (2001)
7. Desharnais, J., Möller, B., Struth, G.: Modal Kleene algebra and applications — A survey. Technical Report DIUL-RR-0401, Département d'informatique et de génie logiciel, Université Laval, D-86135 Augsburg (2004)
8. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation **110**(2) (1994) 366–390
9. Kozen, D.: Kleene algebra with tests. ACM Transactions on Programming Languages and Systems **19**(3) (1997) 427–443
10. Kot, L., Kozen, D.: Second-order abstract interpretation via Kleene algebra. Technical Report 2004–1971, Computer Science Department, Cornell University (2004)
11. Fernandes, T., Desharnais, J.: Describing gen/kill static analysis techniques with Kleene algebra. In Kozen, D., ed.: Mathematics of Program Construction, 7th International Conference (MPC 2004). Volume 3125 of Lecture Notes in Computer Science., Stirling, Scotland, UK, Springer (2004) 110–128
12. Fernandes, T., Desharnais, J.: Describing data flow analysis techniques with Kleene algebra. Accepted for publication in a special issue of Science of Computer Programming (SCP) (2006)