$\mathbb{K}$ SEMANTICS FOR ABSTRACTIONS

IRINA MĂRIUCA ASĂVOAE

A THESIS SUBMITTED FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE
UNIVERSITY ALEXANDRU IOAN CUZA, IAŞI

2012

This dissertation shows how abstract interpretation's view of program analysis and verification can be instilled in the $\mathbb{K}$ framework in a generic fashion as reflective semantics.

$\mathbb{K}$ is a rewriting-based framework dedicated to defining executable specifications for programming languages semantics. The definitional style proposed by the $\mathbb{K}$ framework is an amalgamation of features from different semantics (e.g., features from operational semantics, continuation based semantics, denotational semantics, a.s.o.). The current $\mathbb{K}$'s desideratum is to demonstrate that its formal specification environment can be effectively used for various classes of programming languages paradigms.

Abstract interpretation provides a well known, standardized and extensively used framework for program analysis and verification. The main idea in abstract interpretation is that program analysis and verification can be achieved by applying fix-point iterators over sound approximations of program semantics. Solely from a semantics perspective, abstract interpretation is a *reflective* semantics environment where the reflexion of the semantics is called *abstraction*.

The program analysis and verification approaches tackled in $\mathbb{K}$ are either model checking, achieved via Maude's LTL model checker, or deductive verification, achieved via matching logic. However, the third major verification technique, namely abstract interpretation, was not systematically approached in $\mathbb{K}$. Our thesis addresses this omission and covers it. As such, we design a generic method for defining in $\mathbb{K}$ abstract specifications of pushdown systems and fixpoint iterators over these specifications. We demonstrate the efficiency of this design by instantiating it with three case studies of abstractions for data analysis, alias analysis, and shape analysis.

0.1 Objective

The main objective of the current work is to study how abstract interpretation can be used in $\mathbb{K}$ for program analysis and verification.

Abstract interpretation is a very prolific research area which debuted in [13] as a unified framework for static program analysis. Hence, in the beginnings abstract interpretation was a framework dedicated to defining static semantics. However, along the time, abstract interpretation proved to be applied in many other areas of programming languages. For a comprehensive description of these applications we refer to [12].

As stated before, we aim to obtain a combination of static semantics with dynamic semantics. More to the point, we intend to employ model checking over dynamic semantics of abstractions in order to reproduce results obtained by static analysis via static semantics. In doing this, we rely on the abstract interpretation view given in [33] where a meta-algorithm for representing the analysis methods from abstract interpretation into abstract model checking problems is described.

The meta-algorithm from [33] is transforms the concrete program into the abstract one into two steps of syntax-based, control flow-preserving abstraction and a final step which applies collecting semantics via model checking of the abstract model. Note that our main concern for the current work is this final step, while the syntax-based abstractions are collapsed into a single step and covered only for a simple example.

Our view about the meta-algorithm in [33] coordinates with the view available in the tool Astrée [35, 11]. Namely, Astrée is *a comprehensive collection of abstractions used to derive static analysis results for structured C programs, with complex memory usages, but without dynamic memory allocation and recursion*. However, our main concern is the analysis method itself. Hence, we adopt the decoupling provided by the meta-algorithm in

[33], and assume that the static analysis methods are parametric in the source programming language. As such, we provide *a collection of abstract programming languages used to derive static analysis results for real programming languages containing static and dynamic memory allocation in the presence of recursion.*

By comparison with the meta-algorithm in [33], we replace conceptually the program with the specification of the programming language in which the program is defined. Moreover, the syntax-based, control flow preserving abstracting steps are seen as one (compositional) step. In any case, our main concern in the current work is the step which uses the abstract model, in our case abstract programming language specification, to obtain the static analysis results. This step is based on the collecting semantics, a semantics standardized in abstract interpretation as a means of producing fixpoint iterations. Collecting semantics relies on the operational semantics of the abstraction and collects executions of the targeted program via a forward or backward fixpoint iteration.

Consequently, we provide a generic method for defining various analysis and verification methods using abstract programming languages specifications and some instantiation of collecting semantics. This is the objective of the current work and, out of the two steps, we focus on the second one.

Brief related work: Among the projects supporting formal executable semantics specification we mention AsmL [34], Coq [36, 10], Isabelle [37, 26, 25], or Maude [38, 19]. $\mathbb{K}$ is a dedicated framework for specification of formal executable semantics [14, 23]. Among the achievements reported in $\mathbb{K}$ we mention the C specification [18, 16] and the specification of a subset of Verilog [24]. The verification methods employed by these definitions are model checking, via Maude LTL Model Checking tool [15], and deductive verification, via matching logic [14]. There are also ad-hoc approaches to analysis methods as type checking [17, 16] and a more standardized assertion-based analysis framework [22, 21]. However, abstract interpretation remains unexplored methodically in the $\mathbb{K}$ framework.

0.2 Contributions

Towards achieving our objective, the key contributions of this dissertation are:

1. **Generic collecting semantics for pushdown systems abstractions in $\mathbb{K}$** We describe a methodology for defining in $\mathbb{K}$ finite abstractions for pushdown systems. We choose pushdown systems because of their relative generality wrt programming languages. Furthermore, we give a generic algorithm for defining collecting semantics over the $\mathbb{K}$ specification of finite pushdown systems. This methodology is instantiated in all subsequent contributions. This contribution is introduced in [1, 2] and presented in Chapter 3.

2. **Data analysis in $\mathbb{K}$** We instantiate with

predicate abstraction [20], a flow sensitive abstraction for static memory which can be used to verify data invariants. We associate predicate abstraction with a collecting semantics which produces the fixpoint iteration. This contribution is published in [3, 4] and presented in Chapter 4.

3. **Alias analysis in** $\mathbb{K}$ We give the $\mathbb{K}$ specification of an abstract programming language defined in [30, 32]. We use this language with a collecting semantics which defines the fixpoint iteration for producing alias analysis results. This contribution is introduced in [1] and presented in Chapter 5.

4. **Shape analysis in** $\mathbb{K}$ We give the $\mathbb{K}$ specification of an abstract programming language defined in [31]. We use this language with a col-

lecting semantics which defines a generic invariant model checking algorithm for pushdown system specifications. The state properties employed for the abstract language define shape invariants. We employ these properties to produce demand driven shape analysis. This contribution is briefly described in [7, 5, 31, 6] and detailed in Chapter 6.

Earlier work on program analysis in $\mathbb{K}$ is presented in [22, 21] where the analysis is given as an abstract semantics for a language of program assertions. That work evolved into the deductive verification tool proposed by matching logic [29, 28, 14]. The main difference in the approach presented in the current work is that we propose an abstract semantics which is decoupled from the actual code, in the style of abstract interpretation.

Other publications co-authored and co-related with the current work, but not directly reflected here, are [8, 9, 27].

References

[1] I. M. Asăvoae. Abstract semantics for alias analysis in K. *K 2011*, 2012. To appear.

[2] I. M. Asăvoae. Systematic design of abstractions in K. *WADT 2012, Technical Report TR-08/12, Universidad Complutense de Madrid, Departamento de Sistemas Informáticos y Computación*, pages 9–11, 2012. Accepted for presentation at WADT'12, `http://maude.sip.ucm.es/wadt2012/docs/WADT2012-preproceedings.pdf`.

[3] I. M. Asăvoae and M. Asăvoae. Collecting semantics under predicate abstraction in the K framework. P. C. Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held*

as a Satellite Event of ETAPS 2010, Paphos, Cyprus, March 20-21, 2010, Revised Selected Papers, volume 6381 of *Lecture Notes in Computer Science*, pages 123–139. Springer, 2010.

[4] I. M. Asăvoae, M. Asăvoae, and D. Lucanu. Path directed symbolic execution in the K framework. T. Ida, V. Negru, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie, editors, *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2010, Timişoara, Romania, 23-26 September 2010*, pages 133–141. IEEE Computer Society, 2010.

[5] I. M. Asăvoae, F. de Boer, M. Bonsangue, D. Lucanu, and J. Rot. Bounded model checking of recursive programs with pointers in K. *WRLA 2012*, 2012. Accepted for presentation as work in progress at WRLA'12.

[6] I. M. Asăvoae, F. de Boer, M. Bonsangue, D. Lucanu, and J. Rot. Bounded model checking of recursive programs with pointers in K. *WADT 2012, Technical Report TR-08/12, Universidad Complutense de Madrid, Departamento de Sistemas Informáticos*

y Computación, pages 12–15, 2012. Accepted for presentation at WADT'12, http://maude.sip.ucm.es/wadt2012/docs/WADT2012-preproceedings.pdf.

[7] I. M. Asăvoae, F. de Boer, M. Bonsangue, D. Lucanu, and J. Rot. Model checking programs with dynamic linked structures. Technical Report LIACS Technical Report 2012-02, LIACS, Universiteit Leiden, 2012.

[8] M. Asăvoae and I. M. Asăvoae. Using the executable semantics for CFG extraction and unfolding. D. Wang, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie, editors, *13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2011, Timişoara, Romania, September 26-29, 2011*, pages 123–127. IEEE Computer Society, 2011.

[9] M. Asăvoae, I. M. Asăvoae, and D. Lucanu. On abstractions for timing analysis in the K framework. R. P. na, M. van Eekelen, and O. ShkaravskaDongming, editors, *Second International Workshop on Foundational and Practical Aspects of Resource Analysis,*

FOPARA 2011, Madrid, Spain, May 2011, Revised selected papers, volume 7177 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2012.

[10] G. Barthe, G. Dufay, L. Jakubiec, B. P. Serpette, and S. M. de Sousa. A formal executable semantics of the JavaCard platform. D. Sands, editor, *Programming Languages and Systems, 10th European Symposium on Programming, ESOP 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, volume 2028 of *Lecture Notes in Computer Science*, pages 302–319. Springer, 2001.

[11] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. R. Cytron and R. Gupta, editors, *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003, San Diego, California, USA, June 9-11, 2003*, pages 196–207. ACM, 2003.

[12] P. Cousot. Abstract interpretation. `http://www.di.ens.fr/~cousot/AI/`.

[13] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *Symposium on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.

[14] T. F. Şerbănuţă, G. Roşu, and A. Ştefănescu. An overview of K and matching logic. M. Hills, editor, *K'11*, Electronic Notes in Theoretical Computer Science, to appear.

[15] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. F. Gaducci and U. Montanari, editors, *Workshop on Rewriting Logic and Its Applications*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 2002.

[16] C. Ellison. *An Executable Formal Semantics of C with Applications*. PhD thesis, University of Illinois at Urbana-Champaign, July 2012. `http://fsl.cs.uiuc.edu/index.php/A_Formal_Semantics_of_C_with_Applications`.

[17] C. Ellison, T.-F. Şerbănuţă, and G. Roşu. A rewriting logic approach to type inference. A. Corradini and U. Montanari, editors, *Recent Trends in Algebraic Development Techniques, 19th International Workshop, WADT 2008, Pisa, Italy, June 13-16, 2008, Revised Selected Papers*, volume 5486 of *Lecture Notes in Computer Science*, pages 135–151. Springer, 2009.

[18] C. Ellison and G. Roşu. An executable formal semantics of C with applications. J. Field and M. Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 533–544. ACM, 2012.

[19] A. Farzan, F. Chen, J. Meseguer, and G. Roşu. Formal analysis of Java programs in JavaFAN. R. Alur and D. Peled, editors, *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*, pages 501–505. Springer, 2004.

[20] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. *Proceedings of the 9th Conference on Computer-Aided Verification*, pages 72–83. Springer-Verlag, 1997.

[21] M. Hills. *A Modular Rewriting Approach to Language Design, Evolution, and Analysis*. PhD thesis, University of Illinois at Urbana-Champaign, 2009. `http://fsl.cs.uiuc.edu/~mhills/thesis/thesis.pdf`.

[22] M. Hills and G. Roşu. A rewriting logic semantics approach to modular program analysis. C. Lynch, editor, *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 151–160, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[23] D. Lucanu, T.-F. Şerbănuţă, and G. Roşu. K framework distilled. F. Duran, editor, *WRLA 2012*, volume ? of *Lecture Notes in Computer Science*, pages ?–?? Springer, 2012. To appear.

[24] P. O. Meredith, M. Katelman, J. Meseguer, and G. Roşu. A formal executable seman-

tics of Verilog. *8th ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010), Grenoble, France, 26-28 July 2010*, pages 179–188. IEEE Computer Society, 2010.

[25] S. Owens. A sound semantics for OCamllight. *In: Programming Languages and Systems, 17th European Symposium on Programming, ESOP 2008, Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[26] J. F. Ranson, H. J. Hamilton, P. W. L. Fong, H. J. Hamilton, and P. W. L. Fong. A semantics of Python in Isabelle/HOL, 2008.

[27] A. Riesco, I. M. Asăvoae, and M. Asăvoae. A generic program slicing technique based on language definitions. *WADT 2012, Technical Report TR-08/12, Universidad Complutense de Madrid, Departamento de Sistemas Informáticos y Computación*, pages 91–92, 2012. Accepted for presentation at WADT'12, `http://maude.sip.ucm.es/wadt2012/docs/WADT2012-preproceedings.pdf`.

[28] G. Roşu and A. Ştefănescu. Matching logic: a new program verification approach. R. N. Taylor, H. Gall, and N. Medvidovic, editors, *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 868–871. ACM, 2011.

[29] G. Roşu, C. Ellison, and W. Schulte. Matching logic: An alternative to Hoare/Floyd logic. M. Johnson and D. Pavlovic, editors, *Algebraic Methodology and Software Technology - 13th International Conference, AMAST 2010, Lac-Beauport, QC, Canada, June 23-25, 2010. Revised Selected Papers*, volume 6486 of *Lecture Notes in Computer Science*, pages 142–162. Springer, 2011.

[30] J. Rot. *A Pushdown System Representation for Unbounded Object Creation*. PhD thesis, Universiteit Leiden Opleiding Informatica, June 2010. `http://www.liacs.nl/assets/Masterscripties/10-06-JurriaanRot.pdf`.

[31] J. Rot, I. M. Asăvoae, F. de Boer, M. Bonsangue, and D. Lucanu. Interacting via the

heap in the presence of recursion. *ICE 2012*, 2012. To appear.

[32] J. Rot, M. Bonsangue, and F. de Boer. A pushdown automaton for unbounded object creation. *FOVEOOS 2010*, 2010. Accepted for presentation as position paper/ work in progress at FOVEOOS.

[33] D. A. Schmidt and B. Steffen. Program analysis *as* model checking of abstract interpretations. G. Levi, editor, *Static Analysis, 5th International Symposium, SAS '98, Pisa, Italy, September 14-16, 1998, Proceedings*, volume 1503 of *Lecture Notes in Computer Science*, pages 351–380. Springer, 1998.

[34] WWW. AsmL. `http://research.microsoft.com/en-us/projects/asml/`.

[35] WWW. Astrée. `http://www.astree.ens.fr/`.

[36] WWW. Coq. `http://coq.inria.fr/`.

[37] WWW. Isabelle. `http://www.cl.cam.ac.uk/research/hvg/isabelle/`.

[38] WWW. Maude. `http://maude.cs.uiuc.edu/`.