TLP **18** (2): 122–125, 2018. © Cambridge University Press 2018
doi:10.1017/S1471068418000029

Introduction to the special issue on computational logic for verification

GERMÁN VIDAL
MiST, DSIC, Universitat Politècnica de València
Camino de Vera, S/N, 46022 Valencia, Spain
(e-mail: gvidal@dsic.upv.es)

submitted 22 March 2018; revised 23 March 2018; accepted 23 March 2018

Logic underlies many fundamental techniques in computer science. It helps us to rigorously formalize these techniques and prove them correct. The last decade has witnessed a growing interest in the use of computational logic methods for program verification. It has attracted researchers from both computational logic and program verification communities, giving rise to a fruitful exchange of ideas and experiences.

One of the most successful approaches in this area considers (some form of) Horn clauses as an intermediate representation, where different analysis and verification methods can be defined; see, e.g., the papers by (Bjørner *et al.* 2015) and (Gange *et al.* 2015). For instance, programs written in different programming languages (e.g., functional, imperative, object oriented, concurrent) can be translated into Constrained Horn Clauses (CHCs) so that the same tools for CHC verification can be used for all these programs. Furthermore, the extensive literature on analysis and transformation techniques for Constraint Logic Programs (CLPs) (Jaffar and Maher 1994), a particular case of CHCs, can now be also applied to improve CHC verification tools (Gallagher and Kafle 2014).

For this special issue, we have selected four papers that contain cutting-edge research results on the use of computational logic for program verification.

The paper "Predicate Pairing for Program Verification" by E. De Angelis, F. Fioravanti, A. Pettorossi and M. Proietti considers a general setting in which the verification of partial correctness properties for imperative programs is reduced to the satisfiability problem for sets of CHCs. In this context, some CHC solvers may fail to verify the satisfiability of a set of CHCs because they cannot find an $\mathscr{A}$-definable model (i.e., a model that is definable in a given class $\mathscr{A}$ of constraints). The paper thus introduces a transformation based on the well-known unfold/fold transformation framework (Tamaki and Sato 1984; Pettorossi and Proietti 1994), called predicate pairing, that may increase the number of sets of CHCs that can be proved satisfiable by finding an $\mathscr{A}$-definable model. In particular, the authors prove that the predicate pairing strategy cannot worsen the effectiveness of the CHC solver. They have also implemented an algorithm for predicate pairing on the VeriMAP transformation and verification system (De Angelis *et al.* 2014) and have evaluated its effectiveness on a benchmark of over 160 problems encoding relational properties of small, yet nontrivial C-like programs. The results show that the use

of predicate pairing as a preprocessor greatly improves the ability of the Z3 CHC solver (de Moura and Bjørner 2008) to prove satisfiability.

The paper "Interval-based Resource Usage Verification by Translation into Horn Clauses and an Application to Energy Consumption" by P. López-García, L. Darmawan, M. Klemen, U. Liqat, F. Bueno and M. V. Hermenegildo also considers a setting in which different programming languages can be translated into an intermediate language based on Horn clauses (Méndez-Lojo *et al.* 2008). It presents a configurable framework for static resource usage verification where specifications can include data size-dependent resource usage functions, expressing both lower and upper bounds. In particular, the framework allows the definition of preconditions expressing intervals within which the input data size of a program is supposed to lie. Both the analysis and the specifications may include different kinds of functions (e.g., polynomial, exponential, summatory or logarithmic functions) and the authors propose sound methods to compare them during the verification process. Experimental results with a prototype implementation of the general framework suggest that the proposed techniques are feasible and accurate in practice. Finally, the applicability of the framework is shown by considering programs written in the XC language and running on the XMOS XS1-L architecture (Watt 2009) w.r.t. some energy consumption specifications. The example illustrates how the verification system can prove whether energy consumption specifications are met or not, or infer particular conditions under which the specifications hold.

The paper "Tree dimension in verification of constrained Horn clauses" by B. Kafle, J. P. Gallagher and P. Ganty shows how the notion of tree dimension can be used in the verification of CHCs. The dimension of a tree is a numerical measure of the tree's branching complexity. This notion, originally introduced to analyse flows in rivers and other tree structures found in nature (Esparza *et al.* 2014), has recently found several applications in program analysis and verification (Esparza *et al.* 2010; Reps *et al.* 2016). Here, the authors apply the notion of tree dimension to measure the complexity of Horn clause derivation trees. In this context, they introduce two algorithms that are based on decomposing a set of CHCs into sets whose derivations have dimension at most k and at least $k + 1$ for some given k. Experimental results on a set of non-linear Horn clause verification problems show its feasibility and its usefulness, both for proving safety as well as for finding bugs in programs.

The paper "A Concurrent Constraint Programming Interpretation of Access Permissions" by C. Olarte, E. Pimentel and C. Rueda proposes the use of linear concurrent constraint (lcc) programming (Fages *et al.* 2001) for the verification of programs annotated with Access Permissions (APs). Concurrent constraint programming (Saraswat 1993) subsumes and extends both concurrent logic programming (Shapiro 1989) and CLP (Jaffar and Maher 1994). Essentially, APs are abstractions about the aliased access to an object content (Boyland *et al.* 2001); they allow a direct control of the access to the mutable state of an object, thus facilitating verification and enabling code parallelization. The authors have implemented a tool called Alcove, which allows both to animate the AP behavior of a given program and to formally verify properties such as the absence of deadlocks or the correctness

w.r.t. a given specification. The technique is based on a declarative interpretation of APs as lcc programs. The authors exploit the underlying constraint system of lcc as well as the logical nature of the language to represent in a natural way the semantics of AP specifications. Furthermore, the declarative reading of lcc agents as formulas in intuitionistic linear logic makes it possible to define effective verification techniques.

Finally, I would like to express my gratitude to M. Truszczynski, Editor-in-Chief of *Theory and Practice of Logic Programming*, as well as to Cambridge University Press for their support in editing the special issue. I would also like to thank the reviewers for their feedback and suggestions for improving the papers, as well as the authors of the selected papers for their high-quality submissions.

References

Bjørner, N., Gurfinkel, A., McMillan, K. L. and Rybalchenko, A. 2015. Horn clause solvers for program verification. In *Fields of Logic and Computation II – Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, L. D. Beklemishev, A. Blass, N. Dershowitz, B. Finkbeiner and W. Schulte, Eds. Lecture Notes in Computer Science, vol. 9300. Springer, Springer International Publishing Switzerland, 24–51.

Boyland, J., Noble, J. and Retert, W. 2001. Capabilities for sharing: A generalisation of uniqueness and read-only. In *Proc. of 15th European Conference on Object-Oriented Programming (ECOOP 2001)*, J. L. Knudsen, Ed. Lecture Notes in Computer Science, vol. 2072. Springer, Verlag Berlin Heidelberg, 2–27.

De Angelis, E., Fioravanti, F., Pettorossi, A. and Proietti, M. 2014. VeriMAP: A tool for verifying programs through transformations. In *Proc. of 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2014)*, E. Ábrahám and K. Havelund, Eds. Lecture Notes in Computer Science, vol. 8413. Springer, 568–574.

de Moura, L. M. and Bjørner, N. 2008. Z3: an efficient SMT solver. In *Proc. of 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, C. R. Ramakrishnan and J. Rehof, Eds. Lecture Notes in Computer Science, vol. 4963. Springer, Verlag Berlin Heidelberg, 337–340.

Esparza, J., Kiefer, S. and Luttenberger, M. 2010. Newtonian program analysis. *Journal of the ACM 57*, 6, 33:1–33:47.

Esparza, J., Luttenberger, M. and Schlund, M. 2014. A brief history of Strahler numbers. In *Proc. of 8th International Conference Language and Automata Theory and Applications (LATA 2014)*, A. Dediu, C. Martín-Vide, J. L. Sierra-Rodríguez, and B. Truthe, Eds. Lecture Notes in Computer Science, vol. 8370. Springer, International Publishing Switzerland, 1–13.

Fages, F., Ruet, P. and Soliman, S. 2001. Linear concurrent constraint programming: Operational and phase semantics. *Information and Computation 165,* 1, 14–41.

Gallagher, J. P. and Kafle, B. 2014. Analysis and transformation tools for constrained Horn clause verification. In *Proc. of 30th International Conference on Logic Programming (Technical Communications), ICLP 2014*. Available from `http://arxiv.org/abs/1405.3883`

Gange, G., Navas, J. A., Schachte, P., Søndergaard, H. and Stuckey, P. J. 2015. Horn clauses as an intermediate representation for program analysis and transformation. *Theory and Practice of Logic Programming 15*, 4–5, 526–542.

Jaffar, J. and Maher, M. J. 1994. Constraint logic programming: A survey. *Journal of Logic Programming 19/20*, 503–581.

Méndez-Lojo, M., Navas, J. A. and Hermenegildo, M. V. 2008. A flexible, (C)LP-based approach to the analysis of object-oriented programs. In *Proc. of 17th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2007). Revised Selected Papers*, A. King, Ed. Lecture Notes in Computer Science, vol. 4915. Springer, Verlag Berlin Heidelberg, 154–168.

Pettorossi, A. and Proietti, M. 1994. Transformation of logic programs: Foundations and techniques. *Journal of Logic Programming 19/20*, 261–320.

Reps, T. W., Turetsky, E. and Prabhu, P. 2016. Newtonian program analysis via tensor product. In *Proc. of 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2016)*, R. Bodík and R. Majumdar, Eds. ACM, New York, NY, USA, 663–677.

Saraswat, V. A. 1993. *Concurrent Constraint Programming*. ACM Doctoral dissertation awards. MIT Press, Cambridge, MA, USA.

Shapiro, E. Y. 1989. The family of concurrent logic programming languages. *ACM Computing Surveys 21*, 3, 413–510.

Tamaki, H. and Sato, T. 1984. Unfold/fold transformation of logic programs. In *Proc. of 2nd International Logic Programming Conference (ICLP'84)*, S. Tärnlund, Ed. Uppsala University, Uppsala, Sweden, 127–138.

Watt, D. 2009. *Programming XC on XMOS Devices*. XMOS Limited. Antony Row, Chippenham, UK.