9-2012

Verification of graph programs

Christopher M. POSKITT
Singapore Management University, cposkitt@smu.edu.sg

Citation

POSKITT, Christopher M.. Verification of graph programs. (2012). *Graph transformations: 6th International Conference, ICGT 2012, Bremen, Germany, September 24-29: Proceedings*. 7562, 420-422.
Available at: https://ink.library.smu.edu.sg/sis_research/4917

Verification of Graph Programs

Christopher M. Poskitt

Department of Computer Science, The University of York, UK
cposkitt@cs.york.ac.uk

1 Introduction

GP (for Graph Programs) is an experimental nondeterministic programming language which allows for the manipulation of graphs at a high level of abstraction [11]. The program states of GP are directed labelled graphs. These are manipulated directly via the application of (conditional) rule schemata, which generalise double-pushout rules with expressions over labels and relabelling. In contrast with graph grammars, the application of these rule schemata is directed by a number of simple control constructs including sequential composition, conditionals, and as-long-as-possible iteration. GP shields programmers at all times from low-level implementation issues (e.g. graph representation), and with its nondeterministic semantics, allows one to solve graph-like problems in a declarative and natural way.

An important question to ask of any program is whether it is correct with respect to its specification. For more traditional programming languages, verification techniques to help answer this have been studied for many years [1]. But a number of issues prevent these techniques being used for graph programs "out of the box" (e.g. the state we must reason about is a graph, not a mapping from variables to values). Fortunately, research into verifying graph transformations is gaining momentum, with numerous verification approaches emerging in recent years [15,2,9,3,8] (though typically focusing on sets of rules or graph grammars). Recent work by Habel, Pennemann, and Rensink [5,6] contributed a weakest precondition based verification framework for a language similar to GP, although this language lacks important features like expressions as graph labels in rules.

2 Research Aims and Progress

Our research programme is concerned with the challenge of verifying graph programs using a Hoare-style approach, especially from a theoretical viewpoint so as to provide the groundwork for later development of e.g. tool support, and formalisations in theorem provers. The particular contributions we aim to make in our thesis are discussed below.

Nested conditions with expressions. In [5,6], nested conditions are studied as an appropriate graphical formalism for expressing and reasoning about structural properties of graphs. However, in the context of GP, where graphs are labelled over an infinite label alphabet and graph labels in rules contain expressions, nested conditions are insufficient. For example, to express that a graph contains an integer-labelled node, one would need the infinite condition $\exists(\,(0)\,) \vee \exists(\,(1)\,) \vee \exists(\,(\text{-}1)\,) \vee \exists(\,(2)\,) \vee \exists(\,(\text{-}2)\,) \vee \cdots$.

In [13,12], we added expressions and assignment constraints to yield nested conditions with expressions (short E-conditions). E-conditions can be thought of as finite representations of (usually) infinite nested conditions, and are shown to be appropriate for reasoning about first-order properties of structure and labels in the graphs of GP. For example, an E-condition equivalent to the infinite nested condition earlier is $\exists(\,(\mathtt{x}) \mid \mathtt{type(x) = int})$, expressing that the variable `x` must be instantiated with integer values. A similar approach was used earlier by Orejas [10] for attributed graph constraints, but without e.g. the nesting allowed in E-conditions. Despite the graphical nature of E-conditions, they are precise (the formal definition is based on graph morphisms), and thus suitable for use as an assertion language for GP.

Many-sorted predicate logic. In [14] we defined a many-sorted first-order predicate logic for graphs, as an alternative assertion language to E-conditions. This formalism avoids the need for graph morphisms and nesting, and is more familiar to classical logic users. It is similar to Courcelle's two-sorted graph logic [4] in having sorts (types) for nodes and edges, but additionally has sorts for labels (the semantic domain of which is infinite): these are organised into a hierarchy of sorts corresponding to GP's label subtypes. This hierarchy is used, for example, to allow predicates such as equality to compare labels of any subtype, while restricting operations such as addition to expressions that are of type integer. We have shown that this logic is equivalent in power to E-conditions, and have constructed translations from E-conditions to many-sorted formulae and vice versa.

Hoare Logic. In [13,12] we proposed a Hoare-style calculus for partial correctness proofs of graph programs, using E-conditions as the assertion language. We demonstrated its use by proving properties of graph programs computing colourings. In proving $\vdash \{c\}\ P\ \{d\}$ where P is a program and c, d are E-conditions, from our soundness result, if P is executed on a graph satisfying c, then if a graph results, it will satisfy d. Currently we are extending the proof rules to allow one to reason about both termination and freedom of failure. We require the termination of loops to be shown outside of the calculus, by defining termination functions # mapping graphs to naturals, and showing that executing loop bodies (rule schemata sets) yields graphs for which # returns strictly smaller numbers.

Case studies and further work. We will demonstrate our techniques on larger graph programs in potential application areas, e.g. in modelling pointer manipulations as graph programs and verifying properties of them. Also, the challenges involved in formalising our Hoare logic in an interactive theorem prover like Isabelle will be explored. Finally, we will discuss how our calculus could be extended to integrate a stronger assertion language such as the HR conditions of [7], which can express non-local properties.

Acknowledgements. The author is grateful to Detlef Plump and the anonymous referees for their helpful comments, which helped to improve this paper.

References

1. Krzysztof R. Apt, Frank S. de Boer, and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Springer-Verlag, third edition, 2009.
2. Paolo Baldan, Andrea Corradini, and Barbara König. A framework for the verification of infinite-state graph transformation systems. *Information and Computation*, 206(7):869–907, 2008.
3. Dénes Bisztray, Reiko Heckel, and Hartmut Ehrig. Compositional verification of architectural refactorings. In *Proc. Architecting Dependable Systems VI (WADS 2008)*, volume 5835, pages 308–333. Springer-Verlag, 2009.
4. Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 5. Elsevier, 1990.
5. Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009.
6. Annegret Habel, Karl-Heinz Pennemann, and Arend Rensink. Weakest preconditions for high-level programs. In *Proc. International Conference on Graph Transformation (ICGT 2006)*, pages 445–460. Springer-Verlag, 2006.
7. Annegret Habel and Hendrik Radke. Expressiveness of graph conditions with variables. In *Proc. Colloquium on Graph and Model Transformation on the Occasion of the 65th Birthday of Hartmut Ehrig*, volume 30 of *Electronic Communications of the EASST*, 2010.
8. Barbara König and Javier Esparza. Verification of graph transformation systems with context-free specifications. In *Proc. Graph Transformations (ICGT 2010)*, volume 6372, pages 107–122. Springer-Verlag, 2010.
9. Barbara König and Vitali Kozioura. Towards the verification of attributed graph transformation systems. In *Proc. Graph Transformations (ICGT 2008)*, volume 5214, pages 305–320. Springer-Verlag, 2008.
10. Fernando Orejas. Attributed graph constraints. In *Proc. International Conference on Graph Transformation (ICGT 2008)*, volume 5214, pages 274–288. Springer-Verlag, 2008.
11. Detlef Plump. The graph programming language GP. In *Proc. Algebraic Informatics (CAI 2009)*, volume 5725, pages 99–122. Springer-Verlag, 2009.
12. Christopher M. Poskitt and Detlef Plump. A Hoare calculus for graph programs. In *Proc. International Conference on Graph Transformation (ICGT 2010)*, volume 6372, pages 139–154. Springer-Verlag, 2010.
13. Christopher M. Poskitt and Detlef Plump. Hoare-style verification of graph programs. *Fundamenta Informaticae*, 118(1-2):135–175, 2012.
14. Christopher M. Poskitt, Detlef Plump, and Annegret Habel. A many-sorted logic for graph programs, 2012. Submitted for publication.
15. Arend Rensink, Ákos Schmidt, and Dániel Varró. Model checking graph transformations: A comparison of two approaches. In *Proc. Graph Transformations (ICGT 2004)*, volume 3256, pages 226–241. Springer-Verlag, 2004.