Development of correct graph transformation systems

Preliminary abstract

Karl-Heinz Pennemann

University of Oldenburg, Germany**
pennemann@informatik.uni-oldenburg.de

Abstract. A major goal of this thesis is the ability to determine the correctness of graphical specifications consisting of a graph precondition, a graph program and graph postcondition. According to Dijkstra, the correctness of program specifications can be shown by constructing a weakest precondition of the program relative to the postcondition and checking whether the precondition implies the weakest precondition. With the intention of tool support, we investigate the construction of weakest graph preconditions, consider fragments of graph conditions, for which the implication problem is decidable, and investigate an approximative solution of said problem in the general case. All research is done within the framework of adhesive high-level replacement categories. Therefore, the results will be applicable to different kinds of transformation systems and petri nets.

Graph transformation has many application areas in computer science, such as software engineering or the design of concurrent and distributed systems. It is a visual modeling technique and expected to play a decisive role in the development of growingly larger and complex systems. However, the use of visual modeling techniques alone does not guarantee the correctness of a design. In context of rising standards for trustworthy systems, there is a growing need for the verification of graph transformation systems and programs. The research of appropriate methods for this purpose is the topic of this thesis. More precisely, a major goal of this thesis is the ability to determine the correctness of graph program specifications consisting of a graph precondition, a graph program and a graph postcondition. For an overview, see Figure 1.

Graph conditions. As language for the specification of state properties, graph conditions [5–12] are investigated. Graphs conditions are a graphical and intuitive, yet precise formalism, well-suited to describe structural properties of first-order.

** This work is supported by the German Research Foundation (DFG), grants GRK 1076/1 (Graduate School on Trustworthy Software Systems) and HA 2936/2 (Development of Correct Graph Transformation Systems).

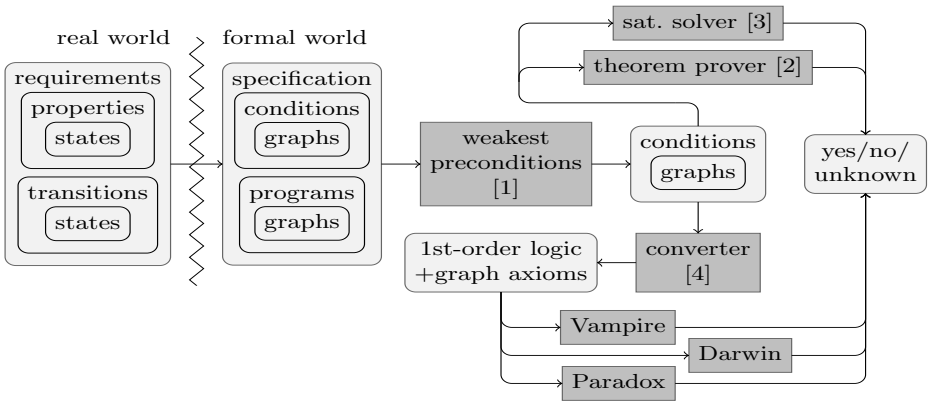

Fig. 1. The big picture

Graph programs. Graph transformations rules with application conditions form the elementary steps of the considered computation model. Additional program constructs are nondeterministic choice, sequential composition, conditional execution, and various iterative constructs. Graph programs [13, 1] are able to model transactions that deal with an unbounded number of elements and are computationally complete.

Weakest preconditions. According to Dijkstra, the correctness of program specifications can be shown in a classical way by constructing a weakest precondition of the program relative to the postcondition and checking whether the precondition implies the weakest precondition. The construction of weakest preconditions and strongest postconditions of graph programs and graph postconditions is described in [1] and [4], respectively.

Implication problem. To determine, whether or not a graph precondition implies a weakest precondition, either a proof or a counterexample must be found. In [2], a correct calculus for proving graph conditions was proposed. In [3], a satisfiability algorithm was presented, proven to be correct and complete, and a decidable fragment of conditions was identified.

Implementation. Following the outlines of [14], the components sketched in Figure 1 are implemented. The weakest precondition transformer, the theorem prover and the satisfiability solver rely on a small number of structure-specific operations, provided by [15] for directed, labeled graphs.

Evaluation. A number of case studies are conducted, e.g. a railroad control system [9, 11], an access control system [1, 2], and a car platooning case study. The prover and solver components are evaluated against existing tools such as Vampire, Darwin and Paradox applied onto straightforward translations of graph conditions into first-order graph formulas [10, 16, 4], using the axiomatization of Courcelle, which is extended to labeled graphs. *Related concepts.* An alternative application of theorem proving to graph transformation is the translation of graph transformation rules into logical formulas [17]. Following

this idea, Strecker [18] models graph programs in the proof assistant Isabelle. His approach supports the manual verification of formulas of "a fragment of first-order logic enriched by transitive closure". There exists a number of model checking approaches such as CheckVML [19], Groove [20], Augur [21], and Hiralisys [22]. Only Augur and Hiralisys apply abstraction, thus are able to handle infinite transition systems. Still, some of the properties of the access control example cannot be treated, due to the fact that both tools consider only properties in the certain fragments of their logics. On the other hand, Augur can check monadic properties such as "Never there exists a path of arbitrary length between two elements", which are not expressible by graph conditions.

Preliminary conclusion and further work. Graphs conditions are convenient to describe system requirements as well as suited to infer knowledge about system behavior. The developed methods and their implementations are able to automatically prove or refuse all test specifications of the access control case study. Surprisingly, existing tools such as Vampire, Darwin and Paradox fail to determine the correctness of some of these specifications. Reasons for this may include the axiomatization of direct, labeled graphs that tends to become a part of the problem to be solved. In contrast, condition-based algorithms are constructively restricted to the considered structure, e.g. directed labeled graphs.

References

1. Habel, A., Pennemann, K.H., Rensink, A.: Weakest preconditions for high-level programs. In: Proc. ICGT'06. Volume 4178 of LNCS., Springer (2006) 445–460
2. Pennemann, K.H.: Resolution-like theorem proving for high-level conditions. In: Proc. ICGT'08. LNCS, Springer (2008) This volume.
3. Pennemann, K.H.: An algorithm for approximating the satisfiability problem of high-level conditions. In: Proc. GT-VC'07. Volume 213 of ENTCS., Elsevier (2008) 75–94
4. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. MSCS (2008) Accepted for publication.
5. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae **26**(3/4) (1996) 287–313
6. Heckel, R., Wagner, A.: Ensuring consistency of conditional graph grammars. In: SEGRAGRA'95. Volume 2 of ENTCS. (1995) 95–104
7. Koch, M., Mancini, L., Parisi-Presicce, F.: Graph-based specification of access control policies. JCSS **71** (2005) 1–33
8. Ehrig, H., Ehrig, K., Habel, A., Pennemann, K.H.: Constraints and Application Conditions: From Graphs to High-Level Structures. In: Proc. ICGT'04. Volume 3256 of LNCS., Springer (2004) 287–303
9. Pennemann, K.H.: Generalized constraints and application conditions for graph transformation systems. Master's thesis, Department of Computing Science, University of Oldenburg, Oldenburg (2004)
10. Rensink, A.: Representing first-order logic by graphs. In: Proc. ICGT'04. Volume 3256 of LNCS., Springer (2004) 319–335
11. Habel, A., Pennemann, K.H.: Nested constraints and application conditions for high-level structures. In: Formal Methods in Software and System Modeling. Volume 3393 of LNCS., Springer (2005) 293–308

12. Ehrig, H., Ehrig, K., Habel, A., Pennemann, K.H.: Theory of constraints and application conditions: From graphs to high-level structures. Fundamenta Informaticae **74** (2006) 135–166
13. Habel, A., Plump, D.: Computational completeness of programming languages based on graph transformation. In: Proc. FoSSaCS'01. Volume 2030 of LNCS., Springer (2001) 230–245
14. Azab, K., Habel, A., Pennemann, K.H., Zuckschwerdt, C.: ENFORCe: A system for ensuring formal correctness of high-level programs. In: Proc. GraBaTs'06. Volume 1., ECEASST (2007) 82–93
15. Zuckschwerdt, C.: Ein System zur Transformation von Konsistenz in Anwendungsbedingungen. Berichte aus dem Department für Informatik 11/06, 114 pages, Universität Oldenburg (2006)
16. Habel, A., Pennemann, K.H.: Satisfiability of high-level conditions. In: Proc. ICGT'06. Volume 4178 of LNCS., Springer (2006) 430–444
17. Courcelle, B.: Graph rewriting: An algebraic and logical approach. In: Handbook of Theoretical Computer Science. Volume B of volume. Elsevier, Amsterdam (1990) 193–242
18. Strecker, M.: Modeling and verifying graph transformations in proof assistants. In: Proc. Termgraph'07. Volume 203 of ENTCS., Elsevier (2008) 135–148
19. Varró, D.: Automated formal verification of visual modeling languages by model checking. Journal of Software and Systems Modelling **3(2)** (2004) 85–113
20. Rensink, A.: The GROOVE simulator: A tool for state space generation. In: AGTIVE'03. Volume 3062 of LNCS., Springer (2004) 485
21. König, B., Kozioura, V.: Augur 2 — a new version of a tool for the analysis of graph transformation systems. In: Proc. GT-VMT'06. Volume 211 of ENTCS., Elsevier (2008) 201?210
22. Bauer, J.: Analysis of Communication Topologies by Partner Abstraction. PhD thesis, Universität des Saarlandes (2006)