Multi-Granular Conflict and Dependency Analysis in Software Engineering based on Graph Transformation (Summary)

Leen Lambers[1] Daniel Strüber[2] Gabriele Taentzer[3] Kristopher Born[4] Jevgenij Huebert[5]

We present a novel multi-granular static conflict and dependency analysis (CDA) of graph transformation as proposed in our ICSE 2018 (International Conference of Software Engineering) contribution [La18b].

Conflicts and dependencies are fundamental phenomena in software engineering. For example, when a software system is developed collaboratively, a change operation can facilitate or prohibit other change operations. In concurrent programming, conflicts may arise from data races when a thread writes to a memory location accessed by another thread. From unrecognized conflicts and dependencies, severe consequences may arise, ranging from productivity obstacles to fatal safety hazards. Therefore, there is a need for techniques to detect conflicts and dependencies automatically.

Graph transformation [Ro97] has been shown to be a versatile foundation for supporting conflict and dependency detection in software engineering, based on the following three principles: First, graphs are used for representing structures of interest, such as states of computation or versions of the system structure. Second, certain changes, such as state or structure modifications, are described using graph transformation rules. Third, the provided transformation specification is fed to the static *conflict and dependency analysis* (CDA) of graph transformations [Pl94, HKT02]: Given a set of transformation rules, all conflicts and dependencies arising from a given pair of rules are identified. A conflict arises, for example, if the first rule application deletes an element required by the second rule application. A key benefit of graph transformation is its mature formal foundation, which supports CDA techniques that are correct by design: all conflicts and dependencies can be detected.

Based on these principles, the CDA of graph transformations has enabled a large number of *use-cases in software engineering*, including analysis and design, model-driven engineering, and testing. For example, graph transformations can be used to model the execution behavior

[1] Universität Potsdam, Hasso-Plattner-Institut, Germany Leen.Lambers@hpi.de
[2] Chalmers University | University of Gothenburg, Sweden danstru@chalmers.se
[3] Philipps-Universität Marburg, Germany taentzer@mathematik.uni-marburg.de
[4] born@mathematik.uni-marburg.de
[5] huebert@mathematik.uni-marburg.de

of Java programs in terms of preconditions and effects on the object structure; identified conflicts and dependencies are then used for generating tests covering them. In model-based refactoring, graph transformations and CDA are used to find a suitable order of refactoring steps. In software product line engineering, feature interactions can be detected by specifying features as graph transformations and identifying conflicts and dependencies with CDA. We present a literature survey of 25 papers describing such use-cases and identify three key requirements for an improved CDA technique for software engineering: it shall be (i) *domain-independent* to be applicable to a large variety of software engineering domains, (ii) *usable* in the sense that it should display a reasonable amount of information to support understandability, and (iii) *efficient* when applied to software projects of realistic size.

To address these requirements, we present a novel static CDA technique for software engineering based on graph transformation. It builds on the notion of *granularity* of conflicts and dependencies introduced in [Bo17][6]. In particular, we provide an *efficient algorithm suite* for computing *binary, coarse-grained, and fine-grained* conflicts and dependencies: Binary granularity indicates the presence or absence of conflicts and dependencies, coarse focuses on root causes for conflicts and dependencies, and fine shows each conflict and dependency in full detail. In an experimental evaluation, our algorithm suite computes conflicts and dependencies rapidly. Finally, we present a *user study*, in which the participants found our coarse-grained results more understandable than the fine-grained ones reported in a state-of-the-art tool. In summary, we present a *multi-granular CDA technique based on graph transformation* achieving the same level of (i) domain-independence as the state of the art, while providing major (ii) understandability and (iii) performance improvements.

References

[Bo17] Born, Kristopher; Lambers, Leen; Strüber, Daniel; Taentzer, Gabriele: Granularity of Conflicts and Dependencies in Graph Transformation Systems. In: Graph Transformation, ICGT. pp. 125–141, 2017.

[HKT02] Heckel, Reiko; Küster, Jochen Malte; Taentzer, Gabriele: Confluence of Typed Attributed Graph Transformation Systems. In: ICGT. pp. 161–176, 2002.

[La18a] Lambers, Leen; Born, Kristopher; Kosiol, Jens; Strüber, Daniel; Taentzer, Gabriele: Granularity of conflicts and dependencies in graph transformation systems: A two-dimensional approach. Journal of Logical and Algebraic Methods in Programming, 103:105 – 129, 2018.

[La18b] Lambers, Leen; Strüber, Daniel; Taentzer, Gabriele; Born, Kristopher; Huebert, Jevgenij: Multi-granular Conflict and Dependency Analysis in Software Engineering Based on Graph Transformation. In: Proceedings of the 40th International Conference on Software Engineering. ICSE '18, ACM, New York, NY, USA, pp. 716–727, 2018.

[Pl94] Plump, Detlef: Critical Pairs in Term Graph Rewriting. In: Mathematical Foundations of Computer Science. volume 841, pp. 556–566, 1994.

[Ro97] Rozenberg, Grzegorz, ed. Handbook of Graph Grammars and Computing by Graph Transformations, Vol. 1: Foundations. World Scientific, 1997.

[6] The granularity notion has been refined in [La18a], partly based on the experiences gathered in [La18b].