On Modularizing Triple Graph Grammars with Rule Refinement

Anthony Anjorin, Karsten Saller, Malte Lochau, Andy Schürr

Real Time Systems Lab, Technische Universität Darmstadt, Germany
{surname}@es.tu-darmstadt.de

Abstract: A *Triple Graph Grammar* (TGG) is a set of declarative rules describing how consistent triples of graph structures in a source, target, and correspondence domain are to be generated. This generative and high-level specification of consistency can be automatically operationalized and used to derive model generators for test case generation, forward and backward translators, and incremental model synchronizers. As with any domain specific language, a means to modularize TGG specifications is an important requirement, especially for practical applications. We formalized an extended concept of *rule refinement* for TGGs in previous work and now report on our experience from using rule refinements intensively in a recent industrial project.

1 Rule Refinement for Triple Graph Grammars in a Nutshell

Restoring and maintaining consistency amongst multiple artefacts is an important challenge towards supporting *concurrent engineering*. This is a primary application of *bidirectional languages*, which support incremental change propagation with a precise semantics.

Triple Graph Grammars (TGGs) [Sch94] are a prominent example for a bidirectional language, with various implementations and success stories. In large TGG specifications, a means of avoiding redundancy and enabling a reuse of rule fragments becomes crucial for maintainability. In previous work [ASLS14], we formalized a flexible concept of *rule refinement* for TGGs, extending existing ideas from various authors. In this paper, we report on our experience using rule refinement in practice. We describe usage patterns that have evolved, and suggest possible improvements and extensions. As we can only provide a high-level intuition in this paper, the reader is referred to [ASLS14] for further details.

TGG rules are essentially graph patterns describing how consistent triples of graph structures in a source, target, and correspondence domain are to be generated. The basic idea of enabling a *refinement* of TGG rules is to (1) provide a *merge operator* with which multiple basis rules can be combined, and (2) allow an *overriding* of certain parts of the merged basis rules in a sub-rule. In our implementation of rule refinement, the labels of the nodes and edges in each rule serve as identifiers. A merge of rules is realized as a union followed by a gluing of elements with the same name. Repeating elements with the same name in a sub-rule equates to overriding the corresponding element in the merged basis rule. All restrictions required to guarantee a well-defined process are presented in [ASLS14].

2 Emerging Usage Patterns and Ideas for Future Work

We used rule refinement intensively in a recent industrial project with the goal of implementing a synchronization tool for two textual languages. The reader is referred to `www.emoflon.org` for a screencast providing an overview of the project.

Rule Refinement as Design or Refactoring? An unexpected process-oriented usage pattern we have observed is that developers use rule refinement not only as an a posteriori means to refactor TGG rules, but also increasingly as an a priori means of *designing* the specification. The latter is, however, only possible with sufficient experience with TGGs, a good understanding of the problem, and adequate training with using rule refinements.

Rule Composition via Multi-Refinement vs. Overriding in Sub-Rules: Supporting multiple refinement enables a *horizontal composition* of often fairly independent patterns, while sub-rules enable a *vertical overriding* of elements from merged basis rules. In our case study, the *refinement network* consisted of 46 TGG rules with 42 refinement relations, a maximum vertical depth of 3, and a maximum horizontal fanout of 3, implying that the network was more flat than deep, i.e., that rule composition is preferred to overriding. This is probably because understanding how sub-patterns are overridden becomes a substantial cognitive challenge with increasing depth of the network, while rule composition is more akin to a separation of concerns and can even aid understanding if applied suitably.

Ideas for Improvements and Extensions: Deleting nodes via overriding in sub-rules leads to confusing refinements and was already forbidden in [ASLS14]. Based on experience and usage patterns, further restrictions can be imposed to ensure “good” refinement networks. Maximum depth and fanout, as well as forbidding deletion/adding of links that were already previously added/deleted in a basis rule are all ideas for such “good practice”. A current limitation of rule refinement is re-using patterns that remain exactly the same structurally, but for which multiple types are to be replaced in a sub-rule. Although this is possible with refinements, the sub-rule would be almost as large as the basis rule as every node with a new type must be repeated explicitly. Possible extensions include *type replacement* refinements or a complementary template concept. Finally, lifting rule refinement to the level of grammars would enable a reuse and composition of entire TGGs.

References

[ASLS14] Anthony Anjorin, Karsten Saller, Malte Lochau, and Andy Schürr. Modularizing Triple Graph Grammars Using Rule Refinement. In Stefania Gnesi and Arend Rensink, editors, *FASE 2014*, volume 8411 of *LNCS*, pages 340–354. Springer, 2014.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer, editors, *WG 1994*, volume 903 of *LNCS*, pages 151–163. Springer, 1994.