

# AUTOMATIC REFINEMENT OF TRANSFORMATIONS BY EXAMPLES

Gábor GUTA

Advisors: Wolfgang SCHREINER, Dániel VARRÓ

## I. Introduction

Model driven software development (MDS) is a promising method to deliver software more efficiently. In MDS software is generated from models with the help of transformations. Current industrial MDS technologies focus on the production of the software with the help of out-of-the-box transformations, but provide no efficient support for the development of new transformations [2]. This problem has been recently recognized by the research community leading to the paradigm of model transformation by example (MTBE). In MTBE the transformation is inferred from examples by various methods, but there are still several open questions [4].

Our goal is to provide automatic or semi-automatic inference of transformations according to an approximate solution of the transformation problem provided by the developer. This approach is extending the traditional MTBE approaches by having additional information from the approximate solution to guide the inference process. In this paper we describe the overall problem we are going to address in the near future. We also present our preliminary results on a simplified model of transformation, namely on finite string transducers.

## II. Problem statement

In a model transformation system we usually generate code  $g$  from a specification  $s$  by a given transformation  $t$ . The generated code is determined by the specification and the transformation  $g=t(s)$ . We may define an expected result of the transformation  $g'$ , which is produced from  $g$  by modification  $m$ , i.e.:  $g'=m(g)$ . It seems an interesting research topic to investigate whether it is possible to modify the approximate transformation  $t$  to the expected transformation  $t'$  automatically to generate the modified code  $g'$  from the original specification, i.e.:  $g'=t'(s)$ . With fixed  $g$  and no constraints on the  $t'$  the task is trivial, e.g. the algorithm may just learn to give the right answer to the specified example. In order to get a non-trivial solution, constraints (metrics of the generalisation properties of the algorithm) need to be introduced on  $t'$ .

Therefore we reformulate this problem in the following way:  $S$  is a set of possible specifications.  $G$  is a set of generated codes where each element  $g$  of  $G$  has a corresponding element  $s$  of  $S$  such that  $g=t(s)$ . Take a small subset of  $S$  and call it  $S_k$ . Call  $G_k$  the subset of  $G$ , containing the elements of  $G$  which correspond to the elements of  $S_k$ . Then we may introduce modification  $m$  over the elements of  $G_k$ , the result of which will be the members of the set  $G_k'$ . Thus the reformulated question is, whether it is possible to infer algorithmically an optimal transformation  $t'$ , which transforms the elements of  $S_k$  to the corresponding  $G_k'$  elements in such a way that a certain “optimality” criterion is satisfied. To express the optimality of the transformation we have to introduce metrics. These metrics cannot be computed by  $t$  alone, but are also influenced by other characteristics related to the transformation, e.g. similarity of  $t$  to  $t'$ .

## III. Our approach

The main difference of our approach from other model transformation by example approaches is that we require extra information in form of an approximate solution. Model transformation by example can be viewed as a transducer inference problem. (A transducer is an automaton with input

and output.) According to the theoretical results of the grammatical inference field we are aware of the negative results of algorithmic learning of even relatively simple computational models, e.g. inferring simple regular grammars [3]. We start our investigation with studying the properties of learning finite state string transducer modifications.

Given a finite state string transducer and an example string e.g.  $(\{a,b\}, \{x,y,w,z\}, \{A,B\}, \{A\}, \{(A,a) \rightarrow (A,x), (A,b) \rightarrow (A,y)\})$  and "aabaab", our experimental setting to evaluate inference algorithms is the following:

- The transducer is executed using the example string as an input. The result of an execution is an output string and sequence of the executed transitions (trace), e.g.: "xxyxxy" and  $[(A,a) \rightarrow (A,x), (A,a) \rightarrow (A,x), (A,b) \rightarrow (A,y), (A,a) \rightarrow (A,x), (A,a) \rightarrow (A,x), (A,b) \rightarrow (A,y)]$ .
- Modifications are inserted into the output by us, e.g.: "xxyzxxyz" and a difference string is generated from the original and the modified output string (the difference string contains information from both versions of the output, as well as their relation) by a selected "diff" algorithm. Example difference string:  $[x, x, y, +z, x, x, y, +z]$ .
- The inference algorithm is executed with the given transducer, trace, and difference string generated during the previous step. The algorithm generates the new transducer, which is evaluated by us. Transitions of the inferred automaton:  $\{(A,a) \rightarrow (A,x), (A,b) \rightarrow (New_1,y), (New_1, \lambda) \rightarrow (A, z)\}$ , where  $\lambda$  denotes the empty symbol.

The basic idea behind the transducer learning algorithm is that it tries to modify the transducer transitions as little as possible at the same time making it capable to produce the expected modified output. The expected solution according to the human intuition is usually one of the many valid inferred solutions, to obtain this intuitive solution extensive tuning of the algorithm implementation is needed. This automatically raises the question how to formalize the intuitive expectation of the inferred solution in form of objective quality criteria. We can use simple product metrics [1] e.g.: number of new states, number of modified transition rules; or we can use constraints on the expected output e.g.: "If we introduce two equivalent changes in the output positions in which the same unique state transition occurs before the insertion positions, then the corresponding modifications of the transducer have to be equivalent." According to our experience such constraints are more important than single metrics to specify the quality of the inferred transducer. We also noted that the information from the preceding steps, like the selection of the diff algorithm, also heavily affects the quality of the solution.

#### IV. Conclusion

Finite state string transducers appear as a promising computational model to study properties of transducer modification learning algorithms. We are going to investigate computational models to which realistic examples can be mapped as well, e.g. tree transformations.

#### References

- [1] N. E. Fenton and S. L. Pfleeger: *Software Metrics: a Rigorous and Practical Approach 2nd*, PWS Publishing, 1998.
- [2] G. Guta, W. Schreiner, and D. Draheim: "A Lightweight MDS Process Applied in Small Projects," in *Proc. of the 35th Euromicro Conference Software Engineering and Advanced Applications*, pp. 255–258, 2009.
- [3] C. Higuera: "A bibliographical study of grammatical inference," *Pattern Recognition*, 38(9): 13322–1348, 2005.
- [4] D. Varró and Z. Balogh: "Automating model transformation by example using inductive logic programming," in *Proc. of ACM Symposium on Applied Computing*, pp. 978–984, 2007.