Guided State Space Exploration using Back-annotation of Occurrence Vectors*

Ábel Hegedüs and Dániel Varró

Budapest University of Technology and Economics, Hungary
{hegedusa,varro}@mit.bme.hu

Introduction Model transformation is a common technique in Model Driven Engineering to design, analyze and simulate various kinds of models. In case of model analysis, forward transformations usually carry out an abstraction to enable efficient formal validation. However, mapping the information gathered from validation back to the original models (i.e. *back-annotation*) is a challenge due to the abstraction gap between the source and target languages.

Graph transformation systems (GTS) are highly relevant at many application areas (e.g. creating models or modeling the behavior of systems) and their abstraction, e.g. as Place-Transition (P/T) nets, are often used for termination analysis [1], optimization [2], verification [3, 4] or finding errors in the implementation (debugging) [5]. However, the results of these analysis methods are usually not *execution paths* (ordered sequence of rule applications or transition firings) for the GTS but a more abstracted information, such as an *occurrence vector* ($\overline{v}_o$) containing only the number of transition executions instead of their exact order.

In order to successfully retrieve the rule application sequence (execution path or trajectory) on the GTS-level, the state space of the GTS is explored using the information collected by back-annotating the analysis results. Our goal is to efficiently identify a feasible execution path of a GTS corresponding to a given occurrence vector (if such path exists). During the exploration of the state space, possible execution paths are examined to check their compliancy with the analysis information.

Occurrence vector-based search strategy approach In [2], the computation of an optimal rule application sequence is performed by encoding the P/T net abstraction (detailed in [1]) of the GTS into an integer linear programming (ILP) problem. The solution of this problem is a candidate transition occurrence vector, which counts the number of rule applications. Since the abstraction does not guarantee that this vector corresponds to an executable execution, its feasibility should be checked on the GTS-level. However, in the original approach, $\overline{v}_o$ was used in the GTS state space exploration by only allowing occurrence vector compliant execution paths to be explored. Therefore, it did not help in selecting the most promising execution path or cutting the search on a given path when it is guaranteed to be infeasible.

* This work was partially supported by the ICT FP7 SecureChange (ICT-FET-231101) European Project and CertiMoT.

Selection and cut-off criteria for search strategy In this paper, we propose additional techniques, which use the occurrence vector as a hint, to guide the state space exploration to further increase the performance of the algorithm. The main features of these new techniques are (*a*) using the rule (or transition) *dependency graph* (G_d) computed from the GTS (or P/T net) [6] to have a global view on the effects of rule applications; (*b*) defining *selection criteria* Cr^{sel} on the applicable rules (transitions) at a given state; and (*c*) defining *cut-off criteria* Cr^{cut} on the paths. Criteria defined in both (*b*) and (*c*) depend on G_d and the application numbers for the rules (transitions) in $\overline{v}_o$.

Definitions Let us assume that we have a GTS including a set of GT rules (r_i) and an *initial graph* (G_I). Furthermore, as a result of critical pair analysis [7] we have the dependency graph (G_d, illustrated in Fig. 1) of the rules, where each r_i is a node (n_i) and there is a directed arc from n_i to n_j if r_j has casual dependency on r_i (i.e. the application of r_i may affect the match set of r_j). Note that there may be arcs in both direction between two nodes.

In this paper, $n_i\bullet$ refers to the set of nodes which have casual dependency on n_i, while $\bullet n_i$ refers to nodes on which n_i has casual dependency (both sets illustrated for n_c in Fig. 1).

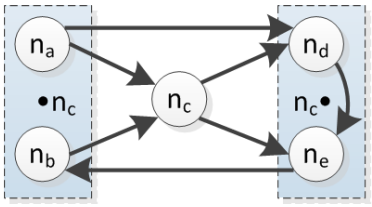

Fig. 1. Example dependency graph

Finally, we have a candidate transition occurrence vector ($\overline{v}_o$) as a solution of the state equation in the P/T net, where $\overline{v}_o$[i] is the number of times that r_i is applied during the execution. During the state space exploration, the number of times r_i has been applied in a given path is stored in the *application vector* ($\overline{v}_a$) as $\overline{v}_a$[i]. Obviously, an execution path of the state space exploration is *compliant* with $\overline{v}_o$ if $\overline{v}_a \leq \overline{v}_o$.

Using these definitions, we can define the following cut-off and selection criteria for the search strategy:

Cr^{cut}_{Ncp} **Non-compliant path (Look-ahead)** When the application of any GT rule would make the current execution path non-compliant with the occurrence vector of its corresponding P/T net, it can be cut.

Cr^{cut}_{Pdr} **Permanently disabled rule** When there is a disabled rule r_i which still has to be applied based on $\overline{v}_o$, but the application of any rule in $\bullet n_i$ would violate Cr^{cut}_{Ncp}, the current path can be cut.

Cr^{sel}_{Mfd} **Maximum forward-dependant application path** Among the applicable GT rules at any given state of the exploration, the one with the most (transitively) dependant rule applications ($n_i\bullet$) should be executed first. The selection is based on calculating the effect of each applicable rule using G_d.

Cr^{sel}_{mbd} **Minimum backward-dependant application path** In order to guide the exploration towards a state where one of the cut-off criteria may be applicable, the rule selection is based on calculating the remaining rule applications for backward-dependant rules ($\bullet(n_i\bullet)$).

Additional notes on the criteria The applicability of these criteria highly depends on the structural properties of the dependency graph. First, if most dependencies between rules are bidirectional or if the graph is almost strongly connected, the selection criteria will be less effective. Furthermore, we suggest using transitive closure for path computation as a first approximation, but we believe that more sophisticated algorithms may be defined by handling cycles in the graph differently from simple paths. Although only the dependency graph was used when defining the criteria, we suggest that a graph created from critical pairs could be used as well.

The application of the presented techniques can increase the efficiency of the state space exploration, while extensions may help to further refine the approach.

References

1. Varró, D., Varró-Gyapay, S., Ehrig, H., Prange, U., Taentzer, G.: Termination Analysis of Model Transformations by Petri Nets. In Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G., eds.: Proc. Third International Conference on Graph Transformation (ICGT 2006). Volume 4178 of LNCS., Natal, Brazil, Springer (2006)
2. Varró-Gyapay, S., Varró, D.: Optimization in Graph Transformation Systems Using Petri Net Based Techniques. Electronic Communications of the EASST (ECEASST) **2** (2006) Selected papers of Workshop on Petri Nets and Graph Transformations.
3. König, B., Kozioura, V.: Counterexample-Guided Abstraction Refinement for the Analysis of Graph Transformation Systems. In Hermanns, H., Palsberg, J., eds.: Tools and Algorithms for the Construction and Analysis of Systems. Volume 3920 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2006) 197–211
4. Baresi, L., Spoletini, P.: On the Use of Alloy to Analyze Graph Transformation Systems. In Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G., eds.: Graph Transformations. Volume 4178 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2006) 306–320
5. Wimmer, M., Kappel, G., Schoenboeck, J., Kusel, A., Retschitzegger, W., Schwinger, W.: A Petri Net Based Debugging Environment for QVT Relations. In: Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on. (2009) 3 –14
6. Mens, T., Kniesel, G., Runge, O.: Transformation dependency analysis - a comparison of two approaches. In Rousseau, R., Urtado, C., Vauttier, S., eds.: LMO, Hermès Lavoisier (2006) 167–184
7. Heckel, R., Küster, J.M., Taentzer, G.: Confluence of Typed Attributed Graph Transformation Systems. In: In: Proc. ICGT 2002. LNCS, Springer (2002)