Criteria Evaluation-driven State Space Exploration of Graph Transformation Systems

Ábel HEGEDÜS
Advisor: Dániel VARRÓ

I. 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 (GT) [1] is a mathematical formalism which provides a rule-based manipulation of graph models. GT rules are defined as two graph patterns, where application of the rule is replacing an occurrence of the first pattern in the graph with the second pattern. *Graph transformation system* (GTS) is defined as graph, where nodes are instance graphs and edges are the applications of GT rules. Thus, it describes the reachable instance graphs from a given host graph. GTS are frequently used at many application areas (e.g. creating models or modeling the behavior of systems). For example, in [2] graph transformations are used for software architecture reconfigurations and the reconfiguration actions of services will be captured by a graph transformation system. Fig. 1 shows a simple reconfiguration where a faulty service (*Do*) having a backup (*St*) fails over to it's standby service, that becomes active (*Ac*).

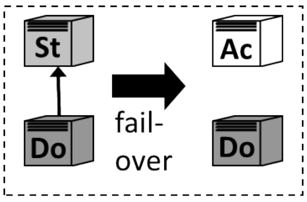

Figure 1: Service reconfiguration example

Place-Transition (P/T) net is a formalism often used to describe the dynamic behavior of systems, P/T nets are defined as bipartite graphs, where nodes can be either places or transitions. Tokens placed in places represent the state of the system, while transitions represent state changes by removing and adding tokens to connected places. The abstraction of GTS, e.g. as Place-Transition (P/T) nets, are often used for termination analysis [3], optimization [4], verification [5, 6] or finding errors in the implementation (debugging) [7].

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.

II. State Space Exploration of Graph Transformation Systems

In the service reconfiguration example, the initial (or host) graph of the GTS represents the current configuration of services. Furthermore, there are certain Quality of Service (QoS) requirements about the desired configuration (e.g. the number of available services) and *global constraints* which must be satisfied at all times (e.g. the maximum amount of services) that can be described as graph patterns. The state space exploration of the GTS is performed in order to find a sequence of manipulations which leads to a state satisfying QoS requirements (*goals*) and each intermediate state satisfies the constraints.

Occurrence vector-based search strategy. In [4], the computation of an optimal rule application sequence is performed by encoding the P/T net abstraction (detailed in [3]) of the GTS (along with the desired *goals* and *global constraints*) into an integer linear programming (ILP) problem (illustrated in Fig. 2). The *solution* of this problem is a candidate *transition occurrence vector* ($\overline{v}_o$), which counts the number of rule applications. Since the abstraction does not guarantee that this vector corresponds to an executable *trajectory*, 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 strategy* 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. Note that an alternative approach could use backward searching with planning techniques from artificial intelligence if the final state is precisely known, but in our case the goals hold only partial information about the state.

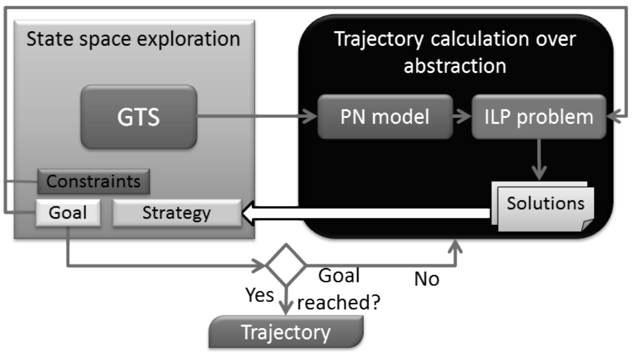

Figure 2: Overview of the approach

Selection and cut-off criteria for search strategy. In this paper, I 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) [8] to have a global view on the effects of rule applications; (b) defining *selection criteria* on the applicable rules (transitions) at a given state; and (c) defining *cut-off criteria* on the paths. Criteria defined in both (b) and (c) depend on G_d and the application numbers for the rules (transitions) in $\overline{v}_o$. A more detailed description of criteria types is given in the following:

- *Cut-off criteria (Cr^{cut})* inspect the current state of the dependency graph and return a boolean result which is true if further exploration of the current branch cannot lead to a goal state with a compliant trajectory. In this case, the exploration continues from an other state instead of executing a GT rule in the current state.
- *Selection criteria (Cr^{sel})* take the dependency graph and define an ordering of the enabled GT rules. A given GT rule r_i is placed before an other rule r_j, if the execution of r_i is more promising, based on the criteria and the current state, than the execution of r_j. The ordering is used instead of selecting the most promising rule since the exploration may lead to a cut-off on the most promising branch. In this case the next GT rule in the ordering is executed.

III. Criteria Definition for Dependency Graphs

In this section the main concepts regarding graph transformation rule dependency and transition occurrence vectors are described which are relevant for the definition of selection and cut-off criteria for the state space exploration. Next, the building blocks which are used to construct arbitrary criteria are introduced, followed by criteria examples usable for the search strategy.

Definitions. Assume that there is a GTS including a set of GT rules (r_i) and an *initial graph* (G_I). Furthermore, as a result of critical pair analysis [9] there is a dependency graph (G_d, illustrated in Fig. 3) 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 sequential 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▸ refers to the set of nodes which have sequential dependency on n_i, while ◂n_i refers to nodes on which n_i has sequential dependency (both sets illustrated for n_c in Fig. 3).

Finally, the candidate transition occurrence vector is 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 $\overline{v}_a[i]$. Furthermore, an execution path of the state space exploration is *compliant* with $\overline{v}_o$ if $\overline{v}_a \leq \overline{v}_o$. Throughout the paper I use the difference between $\overline{v}_o[i]$ and $\overline{v}_a[i]$ ($\overline{v}_o[i] - \overline{v}_a[i]$) as the *remaining application number* of r_i ($\#_i$, illustrated in Fig. 3 at the bottom-left of each node).

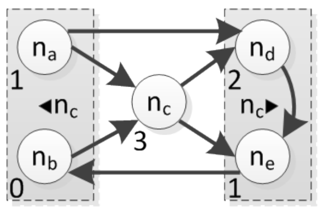

Figure 3: Example dependency graph

Building blocks for criteria. In our approach, the criteria are constructed using starting point identifiers and a well-defined set of operators (i.e. *building blocks*), which can represent navigation over the graph edges, numerical and logical functions between subcriteria, ordering of results and quantifiers. Starting points behave as *operands* and create a criteria together with an operator. The resulting criteria (called a *subcriteria*) can be also an operand to create more complex criteria. Throughout the paper I refer to an operator as *enclosing* for the operands which it is combined with.

Example criteria for guided search. Using these definitions, the following cut-off and selection criteria examples can be defined for the search strategy:

Permanently disabled rule (Cr^{cut}_{Pdr}): 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 ◂n_i would violate result in a non-compliant path, the current path can be cut.

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

IV. Evaluation of Criteria

The definition of the criteria and the evaluation algorithm is separated in the approach since different algorithms are suitable for different classes of dependency graphs. For example a graph without bidirectional edges should be handled differently than one including many of those. Similarly, loops and strongly connected components in the dependency graph call for more sophisticated algorithms.

A simple algorithm which may serve as a basis for advanced ones is described in the following:

First, one of the starting points (S) is selected from the criteria to decide upon which rules will the evaluation iterate through. Next, each node, that satisfies S, is evaluated after each other by iteratively applying the enclosing operators.

Navigation operators return reachable nodes in the graph that are not included in the already visited nodes and the $\#_i$ of each n_i is summed and stored as a partial result (R_p). Numerical and logical operators are applied as implied by their definition and result in R_p and boolean values (R_b), respectively.

Finally, ordering operator places the current node in the appropriate position in the list of evaluated nodes while quantifier operators decide whether Cr^{cut} is satisfied based on R_b (i.e. if Cr^{cut} is satisfied, the branch is cut). The final result of Cr^{sel} therefore is an ordered list of nodes (corresponding to enabled GT rules), where the first node in the list is deemed the most promising for execution. It is important to note that the exploration strategy is often constructed using several criteria, where the combination of Cr^{cut} is trivial (i.e. the branch is cut if any of Cr^{cut} is satisfied), while the combination of Cr^{sel} requires careful consideration.

Evaluation example. The evaluation of cut-off and selection criteria is illustrated on the dependency graph in Fig. 3. Assume that r_a is enabled and r_c is disabled in the current state. The Cr^{cut}_{Pdr} is evaluated on n_c, first the algorithm navigates backward to nodes n_a and n_b and finds that $\#_a$ is not zero, therefore the criteria is not satisfied and the branch is not cut.

Then the Cr^{sel}_{Mfd} is evaluated on n_a, as a result of forward navigation, n_c and n_d are visited and $\#_c$ is summed up with $\#_d$ in R_p (5). Since the navigation is transitive, n_e is also visited in the next step, and $\#_e$ is added to R_p (6). Next n_b is visited, followed by n_c, but that is already visited, so the transitive navigation stops, and $R_p = 6$. If n_e is also enabled, the same criteria is evaluated to $R_p = 5$ (after visiting n_b, n_c and n_d), therefore n_e is placed after n_a in the result list.

Once r_a is applied ($\#_a$ becomes 0) and r_c is still disabled, the evaluation of Cr^{cut}_{Pdr} will return true, as both $\#_a$ and $\#_b$ are zero. Thus the branch is cut and a different branch should be explored.

Conclusion. The state space exploration of GTS appears as a relevant issue in several application areas, however it remains a challenging problem. In this paper the back-annotation of analysis results in the P/T net abstraction of GTS and the inherent dependencies of GT rules are utilized for driving the exploration by evaluating arbitrarily defined cut-off and selection criteria. The application of the presented techniques can increase the efficiency of the state space exploration. Note that the complexity and scalability of the approach are not discussed in the paper, although early results indicate that the approach is usable for meaningful cases.

Ongoing implementation efforts extend upon the CSP(M) framework [10] to provide criteria evaluation-driven state space exploration, while future work includes investigating the usage of graphs created from critical pairs, incorporating goals and constraints into the graphs and the definition of sophisticated criteria and evaluation algorithms.

References

[1] G. Rozenberg, Ed., *Handbook of Graph Grammars and Computing by Graph Transformations: Foundations*, World Scientific, 1997.

[2] L. Baresi, R. Heckel, S. Thöne, and D. Varró, "Style-based modeling and refinement of service-oriented architectures," *Journal of Software and Systems Modelling*, 5(2):187–207, 2006.

[3] D. Varró, S. Varró-Gyapay, H. Ehrig, U. Prange, and G. Taentzer, "Termination Analysis of Model Transformations by Petri Nets," in *Proc. Third International Conference on Graph Transformation (ICGT 2006)*, A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, Eds., vol. 4178 of *LNCS*, Natal, Brazil, 2006. Springer.

[4] S. Varró-Gyapay and D. Varró, "Optimization in Graph Transformation Systems Using Petri Net Based Techniques," *ECEASST*, 2, 2006, Selected papers of Workshop on Petri Nets and Graph Transformations.

[5] B. König and V. Kozioura, "Counterexample-Guided Abstraction Refinement for the Analysis of Graph Transformation Systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, H. Hermanns and J. Palsberg, Eds., vol. 3920 of *LNCS*, pp. 197–211. Springer Berlin / Heidelberg, 2006.

[6] L. Baresi and P. Spoletini, "On the Use of Alloy to Analyze Graph Transformation Systems," in *Graph Transformations*, A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, Eds., vol. 4178 of *LNCS*, pp. 306–320. Springer Berlin / Heidelberg, 2006.

[7] M. Wimmer, G. Kappel, J. Schoenboeck, A. Kusel, W. Retschitzegger, and W. Schwinger, "A Petri Net Based Debugging Environment for QVT Relations," in *Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on*, pp. 3 –14, 16-20 2009.

[8] T. Mens, G. Kniesel, and O. Runge, "Transformation dependency analysis - a comparison of two approaches," in *LMO*, R. Rousseau, C. Urtado, and S. Vauttier, Eds., pp. 167–184. Hermès Lavoisier, 2006.

[9] R. Heckel, J. M. Küster, and G. Taentzer, "Confluence of Typed Attributed Graph Transformation Systems," in *In: Proc. ICGT 2002. LNCS*. Springer, 2002.

[10] Á. Horváth and D. Varró, "CSP(M): Constraint Satisfaction Problem over Models," in *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009. Proceedings*, A. Schürr and B. Selic, Eds., vol. 5795 of *LNCS*. Springer, 2009.