Uncover: Using Coverability Analysis for Verifying Graph Transformation Systems

Jan Stückrath(✉)

Universität Duisburg-Essen, Essen, Germany
jan.stueckrath@uni-due.de

Abstract. Uncover is a tool for high level verification of distributed or concurrent systems. It uses graphs and graph transformation rules to model these systems in a natural way. Errors in such a system are modelled by upward-closed sets for which two orders are provided, the subgraph and the minor ordering. We can then exploit the theory of well-structured transition systems to obtain exact or approximating decidability results (depending on the order and system) for the question whether an error can occur or not. For this framework we also introduced an extension of classical graph transformation which is capable of modelling broadcast protocols.

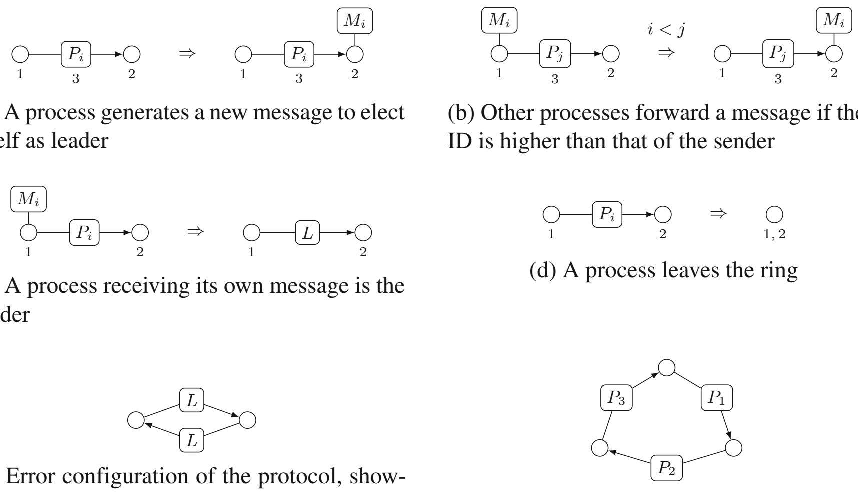
1 Introduction

Verification is a very broad area of computer science and Uncover aims at the highest abstraction level, i.e. the verification of protocols or dynamic systems in general. For modelling systems we use graphs and graph transformation rules [17], called graph transformation systems (GTS). Graphs are here used to model the current state of a system, and graph transformation rules are used to model state changes. More precisely we use hypergraphs, a generalization of directed graphs, where each edge need not connect only two nodes, but can be connected to an arbitrarily long, but finite sequence of nodes. Graph transformation systems are effectively a transformation schema which can be applied to possibly infinitely many graphs and can therefore finitely represent infinitely large transition systems. The transformation approach we use is the single pushout approach (SPO) based on category theoretical constructions using partial morphisms, i.e. partial mappings from graphs to graphs.

Not many tools for verifying GTS exist, examples being Groove [10] for finite state systems or Augur2 [3] and GBT [8,18] for infinite state systems. Since most problems are undecidable in the infinite case, the latter two tools use approximations via Petri nets (Augur2) and abstraction with graph patterns (GBT). With Uncover we also target infinite state systems and use the theory of well-structured transition systems [2,7] to achieve decidability results, which gave rise to the framework we presented in [14]. In this paper we will present Uncover including an introduction to the framework it implements.

Research partially funded by DFG project GaReV.

© Springer International Publishing Switzerland 2015



(a) A process generates a new message to elect itself as leader

(b) Other processes forward a message if their ID is higher than that of the sender

(c) A process receiving its own message is the leader

(d) A process leaves the ring

(e) Error configuration of the protocol, showing two leaders

(f) Initial ring structure of the protocol

Fig. 1. Modelling of a leader election protocol by graph transformation rules [12]

To obtain a well-structured transition system we need to equip the GTS with a well-quasi-order which is a simulation relation for the transition relation, i.e. if a graph G can be transformed to a G', then any graph larger than G can be transformed to a graph larger than G'. Using this order we can now model errors in a GTS by a set of minimal error graphs, i.e. every graph which is larger or equal to a minimal error graph contains the error. We can see this for instance in Fig. 1 where we model a leader election protocol for a ring structure. Initially the protocol starts on a directed ring of processes, each with a unique ID (Fig. 1f) where processes can propose their leadership (Fig. 1a), forward other processes proposals (Fig. 1b), get elected (Fig. 1c) or simply leave the ring (Fig. 1d). The system is erroneous if two processes can both elect themselves to be leader. This error is exhaustively described (for rings) by the minimal error graph in Fig. 1e when using the minor ordering. A graph G is a minor (i.e. smaller or equal) of a graph G' if we can obtain G by deleting nodes or contracting edges of G'. A contraction deletes the edge and merges its incident nodes according to any partition on them, which includes edge deletion. This means that the graph in Fig. 1e is a minor of any directed ring (among others) of length larger or equal to two where there are at least two leaders. Thus, the protocol is correct if and only if we can *not* reach such a ring from the initial ring. Note that contraction is essential in this case and the given error graph would not be sufficient wrt. subgraph ordering, which only allows node and edge deletions. In fact, we would need infinitely many subgraphs to describe the same error.

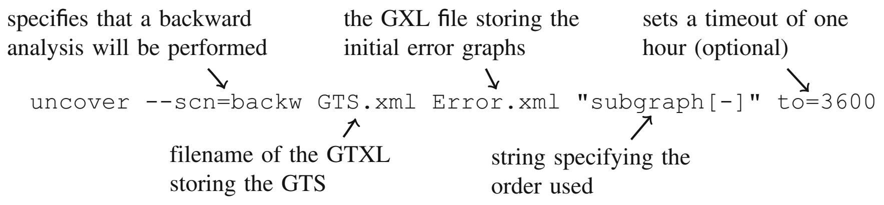
In this setting, checking whether an error is reachable is equivalent to checking whether a minimal error graph is coverable, which is decidable for well-structured transition systems if a so-called effective pred-basis exists. An effective pred-basis is a algorithm which takes a graph G and computes the minimal graphs which can be rewritten – in one step – to a graph larger than G. When called, Uncover will use the initial error graphs as working set and compute in each step the pred-basis of the current working set, add it to the set and keeps only the minimal graphs, until eventually the working set stabilizes. All graphs which are larger or equal to one of the graphs in the final working set can reach an error, i.e. a graph larger or equal to an initial error graph. For the example in Fig. 1 using three processes this will be 38 graphs in total, representing mostly graphs with a "broken" ring structure or graphs where two processes have the same ID. Since the initial graph (Fig. 1f) is not larger or equal to any of those graphs, the protocol is correct. The simulation property of the order ensures the correctness of the result set and being a well-quasi-order ensures termination. This theory has also been successfully applied to related formalisms such as the π-calculus [15].

So far both the subgraph ordering and the coarser minor ordering are implemented in Uncover. Both orders impose different restrictions on the graphs and graph transformation systems and we will illustrate the resulting trade-off in Sect. 3 in more detail. The sources and documentation of the Uncover tool, as well as some example case studies (including Fig. 1) can be found on its main website [19].

2 Design and Usage

Uncover is a command line tool, written in C++ and licensed under GPLv2. Since run times may be long for larger systems, it is designed to run autonomously on a server once it has received its input, logging the performed computations up to the desired verbosity and storing the final set of error graphs. Figure 2 shows how an invocation of the tool may look like.

To perform an analysis Uncover requires three parameters: the system model, the initial error description and the order used. The first two parameters may be any GTS (not requiring initial graphs) and any set of graphs up to certain restrictions depending on the order (see Sect. 3). The order may be chosen from a set of predefined orders provided by Uncover, currently the minor ordering and the subgraph ordering. Beyond the required parameters, there are a few optional parameters e.g. for setting a timeout or the log file verbosity, which are described in the documentation [19].



```
uncover --scn=backw GTS.xml Error.xml "subgraph[-]" to=3600
```

specifies that a backward analysis will be performed
the GXL file storing the initial error graphs
sets a timeout of one hour (optional)
filename of the GTXL storing the GTS
string specifying the order used

Fig. 2. Shows an exemplary use of the Uncover tool

System Model. The system to be analysed must be modelled as a graph transformation system using SPO rules [17], i.e. partial morphisms, as shown in Fig. 1a – d (the set of initial graphs may also be empty). Injective or conflict-free matches can be used, which result in a slightly different induced transition system. In this context a match is conflict-free wrt. some rule if every two elements with the same image are either both deleted or both preserved by the rule. Note that the transition system induced by conflict-free matches contains the transition system using injective matches, since every injective match is also conflict-free. In recent work we extended the standard SPO approach with so-called universally quantified rules, i.e. rules capable of matching the entire neighbourhood of a node, to model broadcast operations [6] and also implemented this extension in Uncover. The input format for the GTS is based on the GTXL format (i.e. XML-based) and a definition file is available with the source code [19].

Initial Error Description. The initial error description is a finite set of graphs and is interpreted as the minimal elements of an upward-closed class of graphs all containing an error. This means that an error can be described in this way if it is invariant wrt. to the order used, i.e. if a graph contains the error, any larger graph must contain the error as well. For instance the error graph in Fig. 1e represents – wrt. the minor ordering – all rings (among others) containing two leaders, which are all erroneous states of the system. As input format for the initial error description we use the XML-based GXL format [11].

Predefined Orders. For an analysis the used order must be specified. It influences the interpretation of the initial error configuration and may impose restrictions on the analyzable GTS (see Sect. 3). Uncover currently supports the minor ordering and the subgraph ordering, although the implemented framework is not limited to these orders. In fact, in [14] we stated necessary conditions for an order to be compatible and have also shown that the induced subgraph ordering[1] satisfies these conditions. Note that different orders also lead to different notions of coverability and may impose different restrictions on the system model. As indicated in Fig.2, the third parameter may either be '`minor`' for the minor ordering or '`subgraph[x]`' for the subgraph ordering, where `x` may either be a natural number specifying a path bound or '`-`' for no bound (we define and discuss path bounds in Sect. 3.2). Furthermore, Uncover is specifically implemented to be easily extendible with further orders.

Results. The analysis procedure returns a finite set of graphs. It contains all graphs that can reach a graph larger than one of the initial error graph. Obviously this also includes the initial error description.

Additional Scenarios. In addition to the backward analysis scenario, Uncover also provides auxiliary scenarios, the important being '`gtxl2latex`' and

[1] G is an induced subgraph of G' if we can obtain G by deleting a subset of the nodes of G' including their incident edges.

‘`gxl2pic`’, which use Graphviz [9] and Latex to draw GXL and GTXL files, and ‘`leq`’, which checks if a graph is in the upward closure of a given set of graphs. All auxiliary scenarios are described in the documentation.

3 Decidability Results

Normally, given a (finite) set of initial error graphs $\mathcal{I}$ and a GTS $\mathcal{T}$, Uncover will return a (finite) set of final graphs $\mathcal{E}$, which characterize by their upward closure all graphs from which an error can be reached. More precisely, a graph G can reach a graph larger than a graph of $\mathcal{I}$ if and only if there is an $G' \in \mathcal{E}$ such that $G' \sqsubseteq G$ (wrt. the order used). However, Uncover is not always guaranteed to terminate and in the following subsections we will examine this separately for the minor and subgraph orderings. We will also see that there is a trade-off between these orders: the minor ordering guarantees termination for all classes of graphs whereas the subgraph ordering can analyse any GTS. Which order is best suited depends on the concrete case study. If the GTS is suitable, the minor ordering is often a good choice. However, the minor ordering is too coarse for some properties to be described as its upward closure, in which case the subgraph ordering is better.

If the GTS has initial graphs for which coverability should be checked, Uncover can also prematurely terminate as soon as a graph was found that is smaller or equal to one of the initial graphs. Moreover, we are not limited to checking coverability for individual graphs. If $\mathcal{T}$ models for instance a distributed algorithm, the final graphs $\mathcal{E}$ represent all network topologies for which the algorithm is *not* correct. This effect can be seen in the leader election protocol in Fig. 1, where final graphs (see [12]) represent networks with duplicate process identifiers as well as non-ring structures.

3.1 Minor Ordering

The minor ordering for hypergraphs was first used in [12] and a similar idea was presented in [1] to abstractly represent heaps of programs, a more restricted class of graphs. Since the minor ordering is a well-quasi-order on all graphs [16], all upward-closed sets are finitely representable. This also guarantees that Uncover will terminate when using minors. However, the minor ordering is not a simulation relation wrt. all GTS, but only for GTS containing edge contraction rules for each label, i.e. rules deleting an edge and merging an arbitrary partition on its incident nodes. A class of systems which naturally satisfy this restriction are lossy systems, where communication is assumed to be unreliable, i.e. messages may be lost at any time. In the example shown in Fig. 1a process leaving the ring and the loss of messages (not shown explicitly) constitute edge contraction rules. In [13] we have shown that this restriction may even hold in the presence of negative application conditions, although this is not yet implemented in Uncover.

If the input GTS does not satisfy the previously mentioned restriction, then Uncover will analyse the GTS as if it would contain edge contraction rules, i.e. implicitly add these rules. Obviously this GTS is an over-approximation of the original GTS and $\mathcal{E}$ will be an over-approximation as well. Note that the precision of this approximation strongly depends on the GTS and that Uncover is still guaranteed to terminate, regardless of approximation.

Although it is technically not a problem, injective matches can currently not be used with the minor ordering in Uncover.

3.2 Subgraph Ordering

We first proposed to use the subgraph ordering for the backwards analysis in [5] and integrated it into our framework in [14]. However, there have also been other approaches to use the subgraph ordering backwards [18] or forwards [4] in the context of well-structured transition systems, often introducing approximations. Uncover implements the subgraph ordering with conflict-free and injective matches and additionally allows so-called universally quantified rules, capable of matching entire neighbourhoods of nodes, in the injective case.

A nice property of the subgraph ordering is that it is a simulation relation wrt. all GTS. However, not every upward closed set is finitely representable, since it is not a well-quasi-order on all graphs, but only on the class of graphs where every (undirected) path is bounded by a constant k. This also means that termination is not guaranteed when we call Uncover without a path bound, although we obtain a precise result for every terminating instance. Note that in the case of non-termination we can still semi-decide coverability for a graph G by letting Uncover check if G was found after each backward step.

To guarantee termination, we need to set a path bound, but this will affect the expressiveness of the computed $\mathcal{E}$. It still holds, that any G in the upward closure of $\mathcal{E}$ can cover a initial error graph. However, for any G *not* in the upward closure of $\mathcal{E}$ we only know that G cannot cover an initial error without exceeding the path bound. In the latter case we simply do not know whether G can or cannot cover an error if the paths were not bounded.

When using the subgraph ordering with injective matches, we can also use universally quantified rules as introduced in [6]. Regardless of the use of bounded or unbounded paths, $\mathcal{E}$ will usually be an over-approximation when using universally quantified rules, since these rules impose negative application conditions.

4 Case Studies

To demonstrate the effectiveness of our analysis procedure we verified several case studies of which some are published in several papers [5,6,12–14]. Table 1 shows for each case study the order used, the class of graphs for which the system was verified, the runtime and the number of graphs in the final graphs $\mathcal{E}$. The runtime results where computed on an Intel® Xeon® CPU E5-2637 v2 with 64 GB RAM using only one core (parallelisation is not yet implemented). All case studies are available on the Uncover website [19].

Table 1. Runtime result for different case studies

Case study	Order	Graph class	Runtime	#(EG)
Leader election (IDs $\leq$ 10)	minor	all graphs	1m 1.6s	451
Leader election (IDs $\leq$ 20)	minor	all graphs	28m 17.5s	2401
Termination det. (faulty)	minor	all graphs	803ms	69
Termination det. (correct)	minor	all graphs	330ms	101
Rights management	subg	all graphs	37ms	4
Dining Philosophers	subg	all graphs	466ms	12
Public-private server	subg	path $\leq$ 50	13.8s	104
Public-private server	subg	path $\leq$ 100	3m 28.6s	204

Leader Election (see [12]**).** This is the leader election protocol modelled in Fig. 1. We could verify that no two processes are elected as leader if the protocol is used on a ring. However, the number of processes needs to be fixed beforehand, since it affects the GTS.

Termination Detection (see [5,13]**).** Here we modelled a termination detection protocol for a ring structure, where processes can be generated by other processes, leave the ring and can be passive or active. We modelled two variants, a faulty and a correct version, where in the former case our analysis found the error and in the latter case we could prove the protocol correct. In [13] we extended this protocol with negative conditions.

Rights management (see [14]**).** We modelled a rights management protocol with users and objects where users can have read or write access rights for objects. We could show that no two users may obtain write access to the same object. For this case study the analysis terminates without setting a path bound (which is not guaranteed in general).

Dining Philosophers (see [6]**).** In this case study we modelled the Dining Philosophers Problem on an arbitrary graph structure using universally quantified rules, i.e. two philosophers need all adjacent forks to eat. We proved that no two adjacent philosophers can eat at the same time. The analysis also terminates without a path bound.

Public-private server. Here we modelled a system of communicating public and private servers and proved that communication between private servers is never leaked to public servers. This analysis needs a path bound to ensure termination.

The computation of the case studies above involves several combinatorial problems which had to be tackled in the implementation of Uncover. On the one hand it is NP-complete to check whether two graphs are related wrt. the subgraph or minor ordering. On the other hand the search for possible matches as well as the actual backward application of a rule are also potential sources of combinatorial explosion. This made it necessary to implement a careful memory management and early optimisations whenever enumerating graphs or matches.

5 Future Development

There are several ways to further improve and extend UNCOVER. To handle the combinatorial blow-up some optimisations are implemented, such as deleting rules which do not affect the analysis, but this could be extended further. This especially holds for universally quantified rules, which still have a lot of optimisation potential. Another obvious improvement is parallelisation, from which UNCOVER would greatly benefit due to the inherently parallel nature of a backward step. There are even some parts of the general framework, such as the induced subgraph ordering or injective matches and negative application conditions for the minor ordering, which still remain to be implemented. For convenience UNCOVERstill requires an automatic visualisation of its performed steps, to support a user in understanding how an error can occur.

Possible improvements also arise from the underlying formalism. The framework of [14] and the implementation of UNCOVER are already designed to allow an easy extension by additional orders. Furthermore, the framework would benefit in particular from an introduction of structural patterns or attributed graphs for describing sets of graphs. The former would for instance allow a finite representation of the class of all circles, even when using subgraphs. Whereas the latter improvement could allow more general rules and for instance the analysis of the leader election case study (Fig. 1) without fixing the number of processes. However, both extensions considerably increase the complexity of computing pred-bases.

References

1. Abdulla, P.A., Bouajjani, A., Cederberg, J., Haziza, F., Rezine, A.: Monotonic abstraction for programs with dynamic memory heaps. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 341–354. Springer, Heidelberg (2008)
2. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: Proceedings of LICS 1996, pp. 313–321. IEEE (1996)
3. AUGUR2. http://www.ti.inf.uni-due.de/research/tools/augur2/
4. Bansal, K., Koskinen, E., Wies, T., Zufferey, D.: Structural counter abstraction. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 62–77. Springer, Heidelberg (2013)
5. Bertrand, N., Delzanno, G., König, B., Sangnier, A., Stückrath, J.: On the decidability status of reachability and coverability in graph transformation systems. In: Proceedings of RTA 2012, vol. 15 of LIPIcs, pp. 101–116 (2012)
6. Delzanno, G., Stückrath, J.: Parameterized verification of graph transformation systems with whole neighbourhood operations. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 72–84. Springer, Heidelberg (2014)
7. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere!. Theor. Comput. Sci. **256**(1–2), 63–92 (2001)
8. Graph Backwards Tool (GB). http://www.it.uu.se/research/group/mobility/adhoc/gbt
9. Graphviz website. http://www.graphviz.org/
10. GROOVE. http://groove.cs.utwente.nl/

11. Holt, R.C., Schürr, A., Sim, S.E., Winter, A.: GXL. http://www.gupro.de/GXL/
12. Joshi, S., König, B.: Applying the graph minor theorem to the verification of graph transformation systems. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 214–226. Springer, Heidelberg (2008)
13. König, B., Stückrath, J.: Well-structured graph transformation systems with negative application conditions. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2012. LNCS, vol. 7562, pp. 81–95. Springer, Heidelberg (2012)
14. König, B., Stückrath, J.: A general framework for well-structured graph transformation systems. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 467–481. Springer, Heidelberg (2014)
15. Meyer, R.: Structural stationarity in the π-calculus. Ph.D. thesis, Carl-von-Ossietzky-Universität Oldenburg (2009)
16. Robertson, N., Seymour, P.: Graph minors XXIII. Nash-Williams' immersion conjecture. J. Comb. Theory, Ser. B **100**, 181–205 (2010)
17. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformation: Volume 1: Foundations. World Scientific Publishing, River Edge (1997)
18. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of ad hoc routing protocols. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 18–32. Springer, Heidelberg (2008)
19. Stückrath, J.: UNCOVER. http://www.ti.inf.uni-due.de/research/tools/uncover/