Graph Grammar Verification through Abstraction

Paolo Baldan[1], Barbara König[2], and Arend Rensink[3]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia
baldan@dsi.unive.it

[2] Institut für Formale Methoden der Informatik, Universität Stuttgart
koenigba@fmi.uni-stuttgart.de

[3] Department of Computer Science, University of Twente
rensink@cs.utwente.nl

Abstract. Until now there have been few contributions concerning the verification of graph grammars, specifically of infinite-state graph grammars. This paper compares two existing approaches, based on abstractions of graph transformation systems. While in the unfolding approach graph grammars are approximated by Petri nets, in the partitioning approach graphs are abstracted according to their local structure. We describe differences and similarities of the two approaches and explain the underlying ideas.

1 Introduction

Until now there have been few contributions concerning the verification of graph grammars, specifically of infinite-state graph grammars. For finite-state graph grammars some solutions have already been presented, ranging from approaches where transition systems are generated directly [15], to approaches based on partial order techniques [6], to encodings of graph grammars into languages accepted by existing model checkers [24, 11]. For a comparison of the first and last approach see also [20].

Graph transformation systems are (in almost all of the approaches) Turing-complete formalisms. Hence when dealing with infinite-state systems the model-checking problem is undecidable. In order to develop automatic verification techniques, it is therefore necessary to perform some sort of approximation in the spirit of abstract interpretation [10]. Taking an abstract view of the system might be equally useful in the case of finite-state systems, in order to reduce the size of the state space, which can be so large that any tool trying to explore it directly would take a finite, but unacceptably long time.

Abstraction may introduce into the transition systems so-called spurious runs, which have no counterpart in the original system. Hence it is necessary to identify a subset of properties whose validity on the abstract model implies their validity in the original system. For instance, due to the presence of spurious runs, it is usually not possible to verify on the abstraction the existence of a run with certain properties.

Dagstuhl Seminar Proceedings 04241
http://drops.dagstuhl.de/opus/volltexte/2005/29

Abstraction techniques of this kind can play an important role in the static analysis of systems characterised by dynamic creation and deletion of objects, mobility and reconfiguration. Such features are very common in today's software systems. Since dynamically evolving systems can often be described by graph transformation systems in a natural way, abstraction techniques as described above seem to be good candidates to use for their verification.

Two concrete examples for abstraction techniques for graph grammars are the analysis of dynamically changing networks of communicating processes (for instance communication protocols where the number of communication partners can grow during the run of protocol) and the analysis of pointer structures on the heap which are modified during the run of a program. Solving the last problem, which has also been addressed in [23], is crucial in order to obtain good analysis results for programs manipulating pointers.

2 Model-checking graph grammars

Our aim is to summarise and, in an informal way, compare the two approaches to the verification of graph grammars presented during the Dagstuhl workshop (see [5, 16]). We will refer to these as the *unfolding*, resp. the *partitioning* approaches.

It should be remarked beforehand that the level of maturity of the two approaches is not the same. The unfolding approach has been developed to a point where all the theory is present to perform actual verifications (see [8, 7]), and indeed a prototype implementation has been developed. The partitioning approach, on the other hand, is at a stage where it is understood how to define, and to some degree reason about, graph abstractions (see [17, 19]), but there are only preliminary ideas on how to transform such abstract graphs.

Both approaches are intended to enable, eventually, the same kind of verification: given a graph grammar and a logical property of graph transition systems (which are transition systems where the states have additional structure, namely, they are graphs), the aim is to determine whether or not that property holds for the particular graph transition system "generated" by the grammar. Moreover, it is the intention to make this work even in the face of infinite-state systems.

2.1 The model checking problem.

Let $\mathcal{G}$ be a graph grammar, with a fixed initial graph $\iota_{\mathcal{G}}$. We will refer to an abstract notion of rewriting, without concentrating on any specific approach to graph rewriting. Given a graph G and a rule R in $\mathcal{G}$, if R can be applied to G producing a new graph H, we will write $R : G \Rightarrow H$.

The *graph transition system* generated by the grammar $\mathcal{G}$ is denoted by $\mathcal{T}(\mathcal{G}) = (S, \rightarrow, \iota_{\mathcal{G}})$, where the states in S are graphs and the transition relation $\rightarrow$ relates two states, written $G \rightarrow H$, if there exists a rule R in $\mathcal{G}$ such that $R : G \Rightarrow H$. The initial state of the transition systems is $\iota_G \in S$, the initial graph of $\mathcal{G}$. Let $\mathcal{L}$ be a temporal logic where "state formulae" predicate over the structure of graphs (i.e., the basic predicates are taken from a logic which

is interpreted over graphs). The *model checking problem* is to decide, for any formula ϕ out of the logic $\mathcal{L}$, whether or not $\mathcal{T}(\mathcal{G}) \models \phi$.

A first observation is that the complexity of the model checking problem strongly depends on the choice of the logic $\mathcal{L}$. For instance, if one restrict to propositional temporal logic (such as LTL, CTL or the modal μ-calculus) where the propositions are closed "state formulae", then it is not necessary to store in the transition relation $\rightarrow$ full information about rewriting steps: a binary relation over the states, recording only the source and target states, suffices. Instead, if we switch to a predicate temporal logic, i.e., where quantification may occur outside the temporal operators, as in [15], the ability to "trace identity though time" is essential and thus the transition relation must give also some information about which graph items (nodes or edges) are preserved along the rewriting step. In the remainder of this abstract we restrict ourselves to propositional logic with "state formulae". As another (well-known) example of the influence of the logic on the complexity of the model checking problem, if $\mathcal{L}$ does not contain a "next state" modality then one may consider $\mathcal{T}(\mathcal{G})$ up to partial order reduction (see, e.g., [12]).

Regardless of the precise choice of the logic, however, part of the semantics of $\mathcal{L}$ will be given in the form of a modelling relation $G \models_C \phi$ for graphs G and "state formulae" ϕ. (The C in $\models_C$ stands for "concrete".) The relation $\models_C$ is an important parameter in the abstraction framework.

2.2 Criteria for abstraction.

Both the *unfolding* and the *partitioning* approach can be understood as defining an *abstraction* on graphs, and lifting transitions to the abstract graphs, thus obtaining an *abstract transition system* $Abs(()\mathcal{G})$ — even if, as we will see, the abstractions take different forms. Roughly, in the context of graph grammar verification, we may distinguish the following criteria for the "quality" of an abstraction.

- *Reflection of state formulae.* To make the abstraction useful in the verification activity, a clear relationship must exist between the state formulae satisfied by a graph G and the formulae satisfied by the corresponding abstraction $Abs(G)$. We will be interested in abstractions which *reflect* state formulae, namely such that whenever $Abs(G) \models \phi$ then $G \models_C \phi$, for appropriate state formulae ϕ.
  On the one hand we can restrict to a suitable fragment of the given logic, say $\mathcal{L}^{\leftarrow}$, in which only state formulae ϕ reflected by the abstraction are included (it is obviously desirable that this fragment is as large as possible). Alternatively, one may *define* the abstraction in order to obtain reflection or preservation of formulae in a predetermined fragment of the logic $\mathcal{L}$.
- *Preservation of rule applicability.* The applicability of any rule in $\mathcal{G}$ should be preserved, in the sense that if a rule R is applicable to a graph G then the same rule (or some abstraction of such a rule) should be applicable to $Abs(G)$.

– *Preservation of rule effects.* Deciding rule applicability is just the first step in the construction of the abstract transitions system $Abs(()\mathcal{G})$: the next step is to construct the (abstract) target state and (if required for the semantics of $\mathcal{L}$) the underlying derivation. To this aim the considered abstraction should enjoy some compatibility properties with respect to the rewriting relation: roughly any rewriting step at the concrete level should have a counterpart at the abstract level.

We informally outline two kinds of abstraction techniques which are relevant for the presented approaches.

Abstraction ordering. The first one is based on an *abstraction ordering*, which is a preorder $\lesssim$ over graphs: when $G \lesssim G'$ we say that the graph G is abstracted by G'. The requirements on the abstraction listed above instantiate as follows.

– *Reflection of state formulae.* A state formula ϕ is *reflected* by the abstraction order if whenever $G \lesssim G'$ and $G' \models_C \phi$ then $G \models_C \phi$.
– *Preservation of rule applicability.* The preservation of rule applicability is formalised by requiring that if a rule R is applicable to G and $G \lesssim G'$ then (some abstraction of) R is applicable to G'.
– *Preservation of rule effects.* The abstraction order preserves rule effects if $G \Rightarrow H$ using a rule R and $G \lesssim G'$ then there exists H' such that $G' \Rightarrow H'$ by using (some abstraction of) R and $H \lesssim H'$. This amounts to say that $\lesssim$ is a simulation relation.

If the abstraction reflects state formulae in $\mathcal{L}$ and preserves rule effects, then the abstract transition system $Abs(()\mathcal{G})$ can be used to check behavioural properties of the original system, expressed in a suitable fragment of the logic $\mathcal{L}$. Roughly, due to the presence of "spurious transitions" in the abstraction, one can check the fragment of the logic which allows one to express "universal" properties of the kind "for all computations ..." or "for all path sequences ...", but not properties like "there exists a run ...". For specific temporal logics (like CTL or the modal mu-calculus) these fragments are precisely characterised (see, e.g., [13]).

Abstraction equivalence. The abstraction can be based on an equivalence relation over graphs called *abstraction equivalence* and denoted by $\simeq$. An equivalence class of graphs $[G]_\simeq = \{G' : G' \simeq G\}$, is referred to as an *abstract graph*. Using this concept, we can lift $\models_C$, the modelling relation for state formulae, to abstract graphs in two different ways:

$$[G]_\simeq \models_A^\exists \phi \text{ iff } \exists G' \simeq G : G' \models_C \phi$$
$$[G]_\simeq \models_A^\forall \phi \text{ iff } \forall G' \simeq G : G' \models_C \phi \ .$$

Thus, $\models_A^\exists$ "overapproximates" the modelling relation whereas $\models_A^\forall$ "underapproximates" it. The requirements on the abstraction listed above instantiate as follows.

- *Reflection of state formulae.* In this case, since $\simeq$ is symmetric, we have that a state formula ϕ is *preserved* by the abstraction equivalence if whenever $G \simeq G'$, then $G \models_C \phi$ iff $G' \models_C \phi$.
  Observe that when a formula ϕ is reflected by the abstraction equivalence then $[G]_\simeq \models_A^\forall \phi$ if and only if $[G]_\simeq \models_A^\exists \phi$; in other words, we can decide the validity of ϕ on the abstract level with perfect precision.
- *Preservation of rule applicability.* The preservation of rule applicability is formalised by requiring that if a rule R is applicable to G and $G \simeq G'$ then (some abstraction of) R is applicable to G'.
- *Preservation of rule effects.* Rule effects are preserved by the abstraction equivalence if $G \simeq G'$ and $R : G \Rightarrow H$ for some R in $\mathcal{G}$ implies $R : G' \Rightarrow H'$ for some $H' \simeq G'$ — in other words, $\simeq$ is a bisimulation for the derivation relation. Note that this implies the preservation of rule applicability.

If the abstraction equivalence preserves state formula and rule effects, then the model checking problem can be solved completely on the abstract level. An example of an abstraction which has this desirable feature is graph isomorphism. In general, though, it is to be expected that any significant gain in time or space efficiency through abstraction will be accompanied by a loss of precision, which in this framework takes the form of approximation in one respect or another.

Combining the two approaches above, the abstraction can be based on an *abstraction preorder* $\lesssim$. The preorder, in turn, induces an *abstraction equivalence* denoted by $\simeq$ (defined by $G \simeq G'$ if $G \lesssim G'$ and $G' \lesssim G$) and a partial order over abstract graphs.

3 Abstraction in the two approaches

So far, we have not been concerned with the actual construction of the abstract states and transitions. Obviously, although equivalence classes serve very well as mathematical objects they are not a good basis for actual decision algorithms. In fact, the existence of a tractable representation of the abstraction function or the equivalence classes $[G]_\simeq$ might prove to be at least as important for the success of a particular approach as the preservation properties listed above.

3.1 The unfolding approach.

The unfolding approach is based on the the unfolding semantics of the given graph grammar [21, 2], a truly concurrent semantics originally introduced in the setting of Petri nets [14].

The logic used to specify behavioural properties of GTSs is a temporal logic $\mu\mathcal{L}2$, which can be seen as a variation of the propositional μ-calculus. The formulae of $\mu\mathcal{L}2$ are generated by closing *state predicates* under temporal modalities ($\Box$ and $\Diamond$), fixed-point operators (μ and ν), and standard logical connectives. In turn, state predicates, which are used to express the graph properties of interest, are formed according to the monadic second-order logic $\mathcal{L}2$ on graphs, where

quantification is allowed over (sets of) edges (see, e.g., [9].) Interesting graph properties can be expressed in $\mathcal{L}2$, like the non-existence or non-adjacency of edges with specific labels, and the absence of certain paths or of certain cycles. Such properties may be used to represent in the graph transformation model relevant properties of the system at hand, like security properties or deadlock-freedom.

The basis for the verification of $\mu\mathcal{L}2$ is the approximation technique, proposed in [1, 3, 4], which approximates the behaviour of GTSs by means of finite Petri net-like structures. More precisely, an approximated unfolding construction maps any given GTS $\mathcal{G}$ to finite structures, called *coverings* of $\mathcal{G}$, which provide "effective" (over-)approximations of the behaviour of $\mathcal{G}$.

The accuracy of the approximation can be chosen by the user and arbitrarily increased. Essentially one can require the approximation to be exact up to a certain causal depth k, thus obtaining the so-called *k-covering* $\mathcal{C}^k(\mathcal{G})$ of $\mathcal{G}$ (see [3]). The coverings are *Petri graphs*, i.e., structures consisting of a Petri net with a graphical structure over places. Each $\mathcal{C}^k(\mathcal{G})$ over-approximates the behaviour of $\mathcal{G}$ in the sense that every computation of $\mathcal{G}$ is mapped to a valid computation of $\mathcal{C}^k(\mathcal{G})$ and every graph reachable from the start graph can be mapped homomorphically to (the graphical component of) $\mathcal{C}^k(\mathcal{G})$ and its image is reachable in the Petri graph. Therefore, given a property over graphs *reflected* by graph morphisms, if it holds for all states reachable in the abstraction $\mathcal{C}^k(\mathcal{G})$ then it also holds for all reachable graphs in $\mathcal{G}$.

Finally in [4] a technique is proposed for reducing the verification of a $\mu\mathcal{L}2$ formula over a GTS $\mathcal{G}$ to the verification of a corresponding multiset formula over (the Petri net component of) a covering of $\mathcal{G}$. This is done for a fragment $\Box\mu\mathcal{L}2$ of $\mu\mathcal{L}2$ not containing in the temporal part the modality $\Diamond$ nor negation, and where the state predicates in are reflected by graph morphisms.

Compared to the general description in the previous section, the approach is based on an *abstraction preorder* over graphs, defined by $G \lesssim G'$ if there exists a graph morphism $\alpha : G \to G'$, bijective on edges and surjective on non-isolated nodes.

The induced *abstraction equivalence* $\simeq$ is isomorphism up to isolated nodes, which is the logical equivalence with respect to logic $\mathcal{L}2$, implying that abstraction equivalence reflects state formulae. Due to the shape of the graph rewriting rules (no isolated nodes in the left-hand side and no node is isolated when generated), abstraction equivalence also preserves rule effects. Hence we can work directly over abstract graphs $[G]_\simeq$ with perfect precision.

Furthermore, it can be seen that the abstraction order over abstract graphs reflects state properties for the fragment $\Box\mu\mathcal{L}2$ of the logic and it preserves rule effects, so that the general theory about abstraction orderings applies.

3.2 The partitioning approach.

In contrast to unfolding, the partitioning approach is based upon a particular construction for abstraction that seems intuitively reasonable, and for which the

ensuing logical properties are then studied. The fundamental idea is to define a notion of *local structure* on the elements of a given graph; for instance, the local structure of a node may be defined by the number of incident edges, or a function from edge types to the number of incident edges of that type; for edges, the local structure may be the tuple of local structures of the endpoints. Clearly, any function s associating to each graph element its local structure gives rise to a similarity $\sim_s$ over the graph elements of a graph, which in turn can be used to partition the graph.

In general one may expect $\sim$ to be a congruence on the algebraic structure of a graph (meaning that two (hyper)edges are similar only if they have similar endpoints and/or labels and/or types, depending on the choice of graph formalism); partitioning gives rise to graph epimorpisms from concrete to abstract graphs. To each element of the abstract graph we may subsequently associate the (uniquely defined) concrete local structure, as well as an indication of the number of concrete elements collected into it.

We have proved in [17] that a notion of local structure s consisting of a pair of functions from edge labels to the number of incoming resp. outgoing edges with that label, combined with a function returning the number of instances collected into an abstract node, gives rise to an abstraction function that preserves a fragment of two-variable first-order logic with counting quantifiers. Rule applicability or rule effects are generally not preserved; we are currently studying the construction, over abstract graphs, of the abstract derivation relation $\Rightarrow_A$. One reason to think why this approach may give rise to useful results is the analogy with the quite successful theory of shape graphs as proposed by Sagiv et al. in [22, 23].

A future direction of study is to define local structure relative to a set of first-order logic properties, possibly given as nested graph constraints as in [18]. In this setup s should return, for every graph element, the set of properties that can be satisfied by letting that element play the role of one of the variables in the corresponding formula. This more sophisticated notion is likely to give better, more controllable preservation properties, probably at the price making the abstract derivation relation $\Rightarrow_A$ even harder to construct.

4 Conclusion

It is not yet entirely clear what the strong and weak points of the two approaches, when compared to each other, are. It will be interesting to further compare the two approaches and their development in the future. One possible area of cooperation that seems to be especially fruitful is to compare the different logics and to determine to what extent these logics can be checked in both approaches.

References

1. P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings of CONCUR 2001*, volume 2154 of *LNCS*, pages 381–395. Springer Verlag, 2001.

2. P. Baldan, A. Corradini, and U. Montanari. Unfolding and Event Structure Semantics for Graph Grammars. In W. Thomas, editor, *Proceedings of FoSSaCS '99*, volume 1578 of *LNCS*, pages 73–89. Springer Verlag, 1999.
3. P. Baldan and B. König. Approximating the behaviour of graph transformation systems. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozemberg, editors, *Proceedings of ICGT'02*, volume 2505 of *LNCS*, pages 14–30. Springer Verlag, 2002.
4. P. Baldan, B. König, and B. König. A logic for analyzing abstractions of graph transformation systems. In R. Cousot, editor, *Proceedings of SAS'03*, volume 2694 of *LNCS*, pages 255–272. Springer Verlag, 2003.
5. Paolo Baldan. Verification of graph transformation systems' properties. Presentation at the Dagstuhl Seminar on Graph Transformations and Process Algebras for Modeling Distributed and Mobile Systems, June 2004. Joint work with Andrea Corradini and Barbara König.
6. Paolo Baldan, Andrea Corradini, and Barbara König. Verifying finite-state graph grammars: an unfolding-based approach. In *Proc. of CONCUR '04*, pages 83–98. Springer-Verlag, 2004. LNCS 3170.
7. Paolo Baldan, Andrea Corradini, and Barbara König. Verifying finite-state graph grammars: an unfolding-based approach. In Philippa Gardner and Nobuko Yoshida, editors, *Fifteenth International Conference on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
8. Paolo Baldan, Barbara König, and Bernhard König. A logic for analyzing abstractions of graph transformation systems. In R. Cousot, editor, *Static Analysis*, volume 2694 of *Lecture Notes in Computer Science*, pages 255–272. Springer-Verlag, 2003.
9. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations*, chapter 5. World Scientific, 1997.
10. Patrick Cousot. Abstract interpretation. *ACM Computing Surveys*, 28(2), 1996.
11. Fernando Luis Dottí, Luciana Foss, Leila Ribeiro, and Osmar Marchi Santos. Verification of distributed object-based systems. In *Proc. of FMOODS '03*, pages 261–275. Springer, 2003. LNCS 2884.
12. P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems*. Lecture Notes in Computer Science. Springer-Verlag, 1996.
13. Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.
14. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.
15. Arend Rensink. Towards model checking graph grammars. In M. Leuschel, S. Gruner, and S. Lo Presti, editors, *Workshop on Automated Verification of Critical Systems (AVoCS)*, Technical Report DSSE–TR–2003–2, pages 150–160. University of Southampton, 2003.
16. Arend Rensink. Abstract graph transformation. Presentation at the Dagstuhl Seminar on Graph Transformations and Process Algebras for Modeling Distributed and Mobile Systems, June 2004.
17. Arend Rensink. Canonical graph shapes. In D. A. Schmidt, editor, *Programming Languages and Systems — European Symposium on Programming (ESOP)*, volume 2986 of *Lecture Notes in Computer Science*, pages 401–415. Springer-Verlag, 2004.

18. Arend Rensink. Representing first-order logic using graphs. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *Lecture Notes in Computer Science*, pages 319–335. Springer-Verlag, 2004.
19. Arend Rensink. State space abstraction using shape graphs. In *Automatic Verification of Infinite-State Systems (AVIS)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2004. To appear.
20. Arend Rensink, Ákos Schmidt, and Dániel Varró. Model checking graph transformations: A comparison of two approaches. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *Lecture Notes in Computer Science*, pages 226–241. Springer-Verlag, 2004.
21. L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, Technische Universität Berlin, 1996.
22. Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Solving shape-analysis problems in languages with destructive updating. *ACM Trans. Program. Lang. Syst.*, 20(1):1–50, January 1998.
23. Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, May 2002.
24. Dániel Varró. Towards symbolic analysis of visual modeling languages. In *Workshop on Graph Transformation and Visual Modeling Techniques '02*, volume 72 of *ENTCS*. Elsevier, 2002.