Recognizable graph languages for the verification of dynamic systems

Christoph Blume

Universität Duisburg-Essen, Germany
christoph.blume@uni-due.de

The theory of regular word languages has a large number of applications in computer science, especially in verification. The notion of regularity can be straightforwardly generalized to trees and tree automata. Therefore it is natural to ask for a theory of regular graph languages.

There exist several notions of regular graph languages [10, 7, 8] – in this context called *recognizable* graph languages – which all turned out to be equivalent. Especially the notion of Courcelle is widely accepted. Very roughly, one can say that a property (or language) of graphs is recognizable whenever it can be derived inductively via an (arbitrary) decomposition of the graph. Alternatively recognizability can be defined via a family of Myhill-Nerode style congruences of finite index, i.e., congruences with finitely many equivalence classes.

The notion of recognizability by Bruggink and König [8] is based on a categorical definition of recognizability in terms of so-called *automaton functors (AF)*, which are a generalization of non-deterministic finite automata. An advantage of this notion of recognizability is that many familiar constructions on finite automata can be straightforwardly generalized to automaton functors.

Let $\mathit{Cospan}(\mathcal{Graph})$ be the category which has arbitrary graphs as objects and cospans of graphs which can be seen as a graph with a left and a right interface as arrows. If automaton functors from the category $\mathit{Cospan}(\mathcal{Graph})$ are considered this yields exactly the notion of recognizable graph languages mentioned above. Cospans of graphs are closely related to GTSs, in particular to the double-pushout (DPO) approach to graph transformation [13]. A DPO rule $\rho\colon L \xrightarrow{\rho_\mathsf{L}} I \xleftarrow{\rho_\mathsf{R}} R$ can be seen as a pair of cospans $\ell\colon \emptyset \to L \xrightarrow{\rho_\mathsf{L}} I$, $r\colon \emptyset \to R \xrightarrow{\rho_\mathsf{R}} I$. Then it holds that $G \Rightarrow_\rho H$ iff $\emptyset \to G \leftarrow \emptyset = c \circ \ell$ and $\emptyset \to H \leftarrow \emptyset = c \circ r$, for some cospan c.

In the following I will give a short overview of my research topics. The focus is on verification techniques based on recognizable graph languages for dynamic systems. Below I will briefly present some projects on which I am working:

Recognizability and Invariant Checking: One of the most straightforward approaches to verification is to provide an invariant and to show that it is preserved by all transformation rules. In the case of words, a language is an invariant for a rule $\ell \to r$ if it holds for all words u and v that $u\ell v \in L$ implies $urv \in L$. In the case of regular (word) languages the rule $\ell \to r$ preserves the language L iff ℓ, r are ordered w.r.t. a monotone well-quasi order such that L is upward-closed w.r.t this well-quasi order [12]. The coarsest such order is the Myhill-Nerode quasi order of a language L which relates arbitrary words v and w iff it holds for all words u and x that $uvx \in L$ implies $uwx \in L$. This is the coarsest monotone quasi order such that L is upward-closed w.r.t. this quasi

order and it can be computed by a fix-point iteration similar to the computation of the minimal finite automaton.

The notion of the Myhill-Nerode quasi order and the result that a rule $\ell \to r$ preserves a languages iff ℓ, r are ordered w.r.t. the Myhill-Nerode quasi order can be lifted to recognizable graph languages (based on $\mathit{Cospan}(\mathcal{G}\mathit{raph})$) [4]. The algorithm for computing the Myhill-Nerode quasi order can also be adapted to the more general setting and there exists a prototype implementation to check whether the language of all graphs containing a given subgraph is an invariant according to a given graph transformation rule [3].

Regular Model Checking: Another approach for the verification of distributed and infinite-state systems is regular model checking [6]. The main idea is to describe (infinite) sets of states as regular languages, i.e. every state is represented by a word, and transitions as regular relations which are represented by finite-state transducers. Verification can then be done by performing a forward or backward analysis. Note that in general the transitive closure of the application of transitions is not guaranteed to be a regular language, therefore it can be necessary to overapproximate the transitive closure in order to use this technique.

Since this approach has been extended to the setting of (regular) tree languages [5] it is a logical step to generalize regular model checking to (recognizable) graph languages. There already exists the notion of MSO-definable transductions invented by Courcelle [11], but this notion does not seem to be that useful, since these transductions are very complex and do not guarantee to preserve the recognizability in general which is required for a forward analysis. It has to be investigated how the notion of finite-state transducer can be generalized to (some kind of) "transduction functors" similar to the generalization of finite-state automata to automaton functors. The goal is to have a notion which is equivalent to finite-state transducers when restricted to word languages. In the case of words, there exists a characterization of transductions by Nivat [2, Chap. III, Thm. 3.2] in terms of regular languages and morphisms of free monoids. This is a possible starting point for the generalization, but it is not that obvious since a graph – unlike a word – can be decomposed in several ways. I have already invented a notion of transduction functor which is based on a category of sets and labeled relations. In this category every tuple is labeled by an arrow which indicates the output of the transduction functor according to the input. However, in order to get a better understanding of these transductions functors, the study of transductions between monoids which are not free is interesting.

Efficient Implementation of Automaton Functors: In general an automaton functor consists of infinitely many finite state sets. But if only recognizable graph languages of bounded path-width are allowed, it is possible to use automaton functors of bounded size. However, the automaton functors might still be very large. This is an important problem that has to be attacked in order to provide tools based on recognizability.

There already exists a prototype implementation of an automaton functor [3] (which is used for invariant checking) that uses an explicit representation of the automaton functors leading to a high memory consumption. A possible solution

to this problem is to find a good representation of the transition relations of the automaton functors. One kind of data structure which is very suitable for the compact representation of large relations are Binary Decision Diagrams (BDDs) [1]. An implementation using BDDs is currently under development and the first experiments have been very promising. One example which has been tested, is the AF accepting all graphs containing a specific subgraph. Using the explicit-state implementation it is only possible to compute this AF for interfaces of size up to 8. If one uses the BDD-based implementation it is possible to compute this automaton functor for interfaces of size up to 100.

Another problem is the determinization of automaton functors, which is required for many constructions, since the direct computation of deterministic automaton functors is not possible in practice due to the state explosion problem. A possible solution is a technique which uses BDDs for the search in powerset automata [9]. How this technique can be adapted to AFs has to be investigated.

The long-term goal is to provide a tool suite for the representation and manipulation of (bounded) automaton functors. Moreover, this tool suite should be usable to verify dynamic systems represented as graphs and GTSs.

The goal of my research is to suggest new directions in the verification of dynamic systems based on recognizable graph languages as well as to investigate how established analysis techniques for regular languages can be adapted to recognizable graph languages. Furthermore, the results will be used for an implementation to provide tools for verification based on automaton functors.

References

1. Andersen, H.R.: An introduction to binary decision diagrams. Course Notes (1997)
2. Berstel, J.: Transductions and Context-Free Languages. Teubner Verlag (1979)
3. Blume, C.: Graphsprachen für die Spezifikation von Invarianten bei verteilten und dynamischen Systemen. Master's thesis, Universität Duisburg-Essen (2008)
4. Blume, C., Bruggink, S., Kønig, B.: Recognizable graph languages for checking invariants. In: Proc. of GT-VMT '10. Elec. Communications of the EASST (2010)
5. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular tree model checking of complex dynamic data structures. In: SAS. Springer (2006)
6. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: CAV '00. Springer (2000)
7. Bozapalidis, S., Kalampakas, A.: Graph automata. Theor. Comp. Sci. 393 (2008)
8. Bruggink, S., König, B.: On the recognizability of arrow and graph languages. In: Proc. of ICGT '08. Springer (2008)
9. Cimatti, A., Roveri, M., Bertoli, P.: Searching powerset automata by combining explicit-state and symbolic model checking. In: TACAS '01. Springer (2001)
10. Courcelle, B.: The monadic second-order logic of graphs. I. recognizable sets of finite graphs. Inf. Comput. 85(1) (1990)
11. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations, chap. 5. World Scientific (1997)
12. de Luca, A., Varricchio, S.: Well quasi-orders and regular languages. Acta Inf. 31(6) (1994)
13. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: LICS (2005)