Modeling Terms by Graphs with Structure Constraints (Two Illustrations)

Clemens Grabmayer
Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
c.a.grabmayer@vu.nl

In my talk I want to explain two examples from my past and current work that highlight the usefulness of graph techniques for problems that have been approached predominantly as questions about terms: increasing sharing in functional programs, and tackling problems about Milner's process interpretation of regular expressions. The unifying element consists in modeling terms by term graphs or transition graphs with structure constraints (higher-order features or labelings with added conditions), and in being able to go back and forth between terms and graphs.

The first illustration concerns the definition, and efficient implementation of maximal sharing for the higher-order terms in the lambda calculus with letrec. For solving this problem, Jan Rochel and I used a representation pipeline from terms via higher-order term graphs and first-order term graphs to deterministic finite-state automata. The setting for the second illustration is Milner's process interpretation of regular expressions, which yields nondeterministic finite-state automata (NFAs) whose equality is studied under bisimulation. In current work with Wan Fokkink, I use labelings of process graphs that witness direct expressibility by a regular expression.

Here I briefly lay out some context, and give references for the topics that I will explain in my talk.

1 Maximal sharing of functional programs

The first example concerns the definition, and the efficient implementation of maximal sharing for functional programs, and more specifically, for the higher-order terms in the λ-calculus with letrec.

For this purpose, Jan Rochel and I develop a 'representation pipeline' from higher-order terms to deterministic finite-state automata (DFAs): (1) Terms in the λ-calculus with letrec can be represented by, appropriately defined, higher-order term graphs, which are first-order term graphs together with higher-order features such as a scope function, or an abstraction prefix function, that are defined on the set of vertices (see [7]); (2) higher-order term graphs are encoded as first-order term graphs (see also [7]), and (3) first-order term graphs are represented as DFAs (see [8]). In this way unfolding equivalence on terms is represented by bisimulation equivalence on term graphs (higher-order and first-order), and ultimately, by language equivalence of DFAs. In [8] we also define a readback operation from DFAs that arise by this representation pipeline back to terms in the λ-calculus with letrec. This operation makes it possible to go back and forth between terms and representing DFAs: it has the property that representation via (1), (2), and (3) is the inverse of the readback operation.

Via the mentioned correspondences, unfolding equivalence of terms in the λ-calculus with letrec can be computed in pseudo-quadratic time $O(n^2 \cdot \alpha(n))$ where α is the inverse Ackermann function (see [8]). Also via the correspondences described above, via DFA-minimization, and via the readback a maximally shared form of higher-order terms can be computed in $O(n^2 \cdot \log n)$ time (again see [8]).

In order to illustrate the maximal-sharing method as a useful and manageable optimizing transformation for the compilation of functional programs, we developed the software tool [11] that is available on

Submitted to:
TERMGRAPH 2018

Haskell's Hackage platform. Following the definition of maximally shared representations via the representation pipeline in [8] (see also Rochel's thesis [10] for more context), this tool transforms a given functional program in the λ-calculus with letrec (the Core language for the implementation of Haskell) into a term graph, and subsequently into a deterministic finite-state automaton (DFA). It prints intermediate representations textually, and displays the obtained DFA graphically. This DFA is then minimized, and finally a maximally shared representation of the original program is computed as the result.

Together with Vincent van Oostrom, I have set out to generalize this technique of representing higher-order terms as term graphs with added features that are needed for modeling scopes of binding constructs. But rather than capturing the constraints on the term graph structure by 'ad hoc' features, we now used 'nesting' as the single added structuring concept. In [6] we defined, and investigated the behavioral semantics of, 'nested term graphs' that arise as follows: by nesting first-order term graphs into the vertices of, initially, a first-order term graph, and then of nested term graphs that have already been formed.

2 Process interpretation of regular expressions

The second illustration concerns the process interpretation of regular expressions. Having developed a complete axiomatization of bisimulation equivalence for finite process graphs represented in μ-term notation [9] (1984), Milner turned to descriptions of finite process graphs by regular expressions. He introduced, also in [9], an interpretation of regular expressions as finite process graphs: 0 is interpreted as the deadlock process, 1 as the immediately terminating process, letters as actions that lead to termination, and the symbols '$+$', '$\cdot$', and $(\cdot)^*$ as operators that enable choice between processes, sequential composition of processes, and iteration of a process, respectively. This interpretation can also be viewed as a translation of regular expressions into nondeterministic finite-state automata (NFAs) whose equality is studied with respect to bisimulation, not via language equivalence. Unlike for the standard language interpretation, not every NFA can be expressed by a regular expression under the process interpretation: not every NFA is bisimilar to the process translation NFA of some regular expression.

Antimirov [1] (1996) arrived at essentially the same interpretation, but without process theory and bisimulation equivalence in mind. He pursued the goal of obtaining for a given regular expression e, in a natural way, an NFA that accepts the language $L(e)$ denoted by e, and that is substantially smaller than the DFA accepting $L(e)$ that is defined by a standard translation of regular expressions. For this purpose he introduced the concept of 'partial derivative' of a regular expression with respect to single letters. The NFA that is obtained by repeated applications of Antimirov's partial derivatives is closely related, and in fact bisimilar, to the NFA that is obtained for a regular expression by Milner's process interpretation.

Still in [9], Milner adapted the complete axiomatization by Salomaa [12] for language equivalence of regular expressions. He obtained a sound axiom system for bisimilarity of process graphs represented by regular expressions, but noticed that completeness for this system is not settled by Salomaa's arguments. In addition to this axiomatization problem, he formulated the problem of characterizing those process graphs that are bisimilar to process interpretations of regular expressions, and a star-height problem.

In my talk I will explain some of the results and partial results that have been obtained for these questions. Finally I will explain the modeling approach of my current work on the axiomatization problem.

The known approaches to Milner's questions fall, broadly speaking, into two groups that are distinguished by how they model processes that are represented by regular expressions: either by working with process terms whose operational semantics is governed by structural operational semantics (SOS) rules, or by reasoning about regular recursive process specifications of a certain structure.

Building on work from the process term tradition, Fokkink (1996-97) showed that the restriction of

Milner's system to exit-less iteration, which he called 'perpetual-loop' and 'terminal cycle', is complete for the general case with 'empty' 1-steps [4], and the easier case without [5]. For this he completely overturned Salomaa's proof technique of extending terms into its contrary, a strategy of term minimization.

Using recursive specifications to formalize processes that are induced by regular expressions, Baeten and Corradini (2005) introduced 'well-behaved specifications' [2]. These systems of equations are arranged according to trees with back-bindings ('palm trees') with a 'loop–exit' structure requirement. This concept enabled Baeten, Corradini, and myself to show that expressibility modulo bisimilarity of a finite process graph by a regular expression is decidable [3], although via a super-exponential procedure.

My current approach to the axiomatization problem (in work with Wan Fokkink) takes the conscious step to reasoning about process graphs for which the palm-tree form is relaxed significantly as constraint. A crucial step is the formulation of a concept of transition graph labeling that is inspired by Milner's notion of 'loop'. Transitions (action-labeled edges) are decorated by additional marker labels that witness that the syntax tree of a regular expression can be inscribed on to a (typically cyclic) process graph. In this way a labeling witnesses that the process graph can be expressed *directly* by a regular expression. This opens the way to develop bisimilarity-preserving transformations of directly expressible process graphs, in order to constructively connect any two given directly expressible process graphs that are bisimilar.

References

[1] Valentin Antimirov (1996): *Partial Derivatives of Regular Expressions and Finite Automaton Constructions*. Theoretical Computer Science 155(2), pp. 291–319, doi:10.1016/0304-3975(95)00182-4.

[2] J.C.M. Baeten & F. Corradini (2005): *Regular Expressions in Process Algebra*. In: Proceedings of LICS 2005, IEEE Computer Society 2005, pp. 12–19, doi:10.1109/LICS.2005.43.

[3] J.C.M. Baeten, F. Corradini & C.A. Grabmayer (2007): *A Characterization of Regular Expressions Under Bisimulation*. Journal of the ACM 54(2), doi:10.1145/1219092.1219094.

[4] W.J. Fokkink (1996): *An Axiomatization for the Terminal Cycle*. Technical Report Logic Group Preprint Series, number 167, Utrecht University. Available at `http://bit.ly/2uJVEDF` (on `semanticscholar.com`).

[5] W.J. Fokkink (1997): *Axiomatizations for the perpetual loop in process algebra*. In P. Degano, R. Gorrieri & A. Marchetti-Spaccamela, editors: Proceedings of the 24th Colloquium on Automata, Languages and Programming - ICALP'97, Bologna, LNCS 1256, Springer, pp. 571–581, doi:10.1007/3-540-63165-8_212.

[6] Clemens Grabmayer & Vincent van Oostrom (2015): *Nested Term Graphs*. In Aart Middeldorp & Femke van Raamsdonk, editors: Post-Proceedings of TERMGRAPH 2014, Vienna, Austria, July 13, 2014, EPTCS 183, Open Publishing Association, pp. 48–65, doi:10.4204/EPTCS.183.4. ArXived at:1405.6380v2.

[7] Clemens Grabmayer & Jan Rochel (2013): *Term Graph Representations for Cyclic Lambda Terms*. In: Proceedings of TERMGRAPH 2013, EPTCS 110, pp. 56–73, doi:10.4204/EPTCS.110. ArXiv:1302.6338v1.

[8] Clemens Grabmayer & Jan Rochel (2014): *Maximal Sharing in the Lambda Calculus with Letrec*. In: Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming, ICFP '14, ACM, New York, NY, USA, pp. 67–80, doi:10.1145/2628136.2628148.

[9] Robin Milner (1984): *A Complete Inference System for a Class of Regular Behaviours*. Journal of Computer and System Sciences 28(3), pp. 439 – 466, doi:10.1016/0022-0000(84)90023-0.

[10] Jan Rochel (2016): *Unfolding Semantics of the Untyped λ-Calculus with* letrec. Ph.D. thesis, Utrecht University. Defended on June 20, 2016. Available at `http://rochel.info/thesis/thesis.pdf`.

[11] Jan Rochel & Clemens Grabmayer (2014): *Maximal Sharing in the Lambda Calculus with letrec*. Haskell Implementation of the method of [8], available at `http://hackage.haskell.org/package/maxsharing/`.

[12] Arto Salomaa (1966): *Two Complete Axiom Systems for the Algebra of Regular Events*. Journal of the ACM 13(1), pp. 158–169, doi:10.1145/321312.321326.