From Compactifying Lambda-Letrec Terms to Recognizing Regular-Expression Processes

(Extended Abstract and Literature)

Clemens Grabmayer
Gran Sasso Science Institute
L'Aquila, Italy

clemens.grabmayer@gssi.it

My talk at the workshop aimed at describing a transferal of ideas from finding graph representations for terms in the Lambda Calculus with letrec, a universal model of computation, to recognizing process graphs that can be expressed by regular expressions via Milner's process interpretation, a proper subclass of finite-state processes. In both cases the construction of structure-constrained graphs was expedient in order to enable to go back and forth easily between, on the one hand, terms and regular expressions, and on the other hand, term-graphs and process graphs. These graph representations respect the appertaining operational semantics, but were conceived with specific purposes in mind: to optimize functional programs in the Lambda Calculus with letrec; and to reason with process graphs denoted by regular expressions, and to decide recognizability of these graphs.

As supplement to my talk, this extended abstract provides ample references and pointers to my past work with co-authors in these two areas. Also I formulate into these two areas.

1 Introduction

My talk [3] at the workshop aimed at motivating a fruitful transferal of ideas between two areas on which I worked in the (a bit removed, and then more recent) past: λ-calculus, and the implementation of functional programming languages (2009–2014), and the process theory of finite-state processes (from 2016). My intention was to show, informally and supported by many pictures: How a solution to the problem of finding adequate graph representations for terms in the Lambda Calculus with letrec, a universal model of computation, turned out to be very helpful in understanding process graphs that can be expressed by regular expressions (via Milner's process interpretation), a proper subclass of finite-state processes.

In both cases the definition of an adequate notion of structure-constrained (term or process) graph was the key to solve a specific practical problem. It was central that the structure-constrained graphs facilitate to go back and forth easily between, on the one hand, terms in the λ-calculus with letrec and term graphs, and on the other hand, regular expressions and process graphs. The graph representations respect the appertaining operational semantics, but were conceived with specific purposes in mind: to optimize functional programs in the Lambda Calculus with letrec; and respectively, to reason with process graphs denoted by regular expressions, and to decide recognizability of these graphs.

The purpose of this extended abstract is to supplement the slides of my talk [3] with a short description of my work with co-authors in these two areas, including ample references.

Section 2 summarizes work by Jan Rochel and myself that led us to the definition, and efficient implementation of maximal sharing for the higher-order terms in the λ-calculus with letrec. Specifically we formulated a representation-pipeline: Higher-order terms can be represented by, appropriately defined,

Submitted to:
DCM 2023

higher-order term graphs, those can be encoded as first-order term graphs, and these can in turn be represented as deterministic finite-state automata (DFAs). Via these correspondences and DFA minimization, maximal shared forms of higher-order terms can be computed.

Section 3 gives an overview of my work, in crucial parts done together with Wan Fokkink, on two non-trivial problems concerning the process semantics of regular expression. In Milner's process semantics, regular expressions are interpreted as nondeterministic finite-state automata (NFAs) whose equality is studied modulo bisimulation. Unlike for the standard language interpretation, not every NFA can be expressed by a regular expression. This raised a non-trivial recognition (or expression) problem, which was formulated by Milner (1984) next to a completeness problem for an equational proof system.

References

[1] Clemens Grabmayer (2018): *Modeling Terms by Graphs with Structure Constraints (Two Illustrations)*. Invited talk at the FSCD/FLoC Workshop *TERMGRAPH 2018*, Oxford, UK, July 7. Slides are available at `https://clegra.github.io/lf/TERMGRAPH-2018-invited-talk.pdf`.

[2] Clemens Grabmayer (2019): *Modeling Terms by Graphs with Structure Constraints (Two Illustrations)*. In Maribel Fernández & Ian Mackie, editors: *Proceedings Tenth International Workshop on Computing with Terms and Graphs, TERMGRAPH@FSCD 2018, Oxford, UK, 7th July 2018*, *EPTCS* 288, pp. 1–13, doi:10.4204/EPTCS.288.1.

[3] Clemens Grabmayer (2023): *From Compactifying Lambda-Letrec Terms to Recognizing Regular-Expression Processes*. Invited talk at the 13th International Workshop on *Developments in Computational Models*, affiliated with the conference FSCD, Sapienza Universita, Rome, July 2. Slides available at `https://clegra.github.io/lf/DCM-2023-invited-talk.pdf`.

2 Compactifying Lambda-Letrec Terms

in the NWO-project *Realizing Optimal Sharing (ROS)* that was headed by Vincent van Oostrom (rewriting and λ-calculus) and Doitse Swierstra (implementation of functional languages).

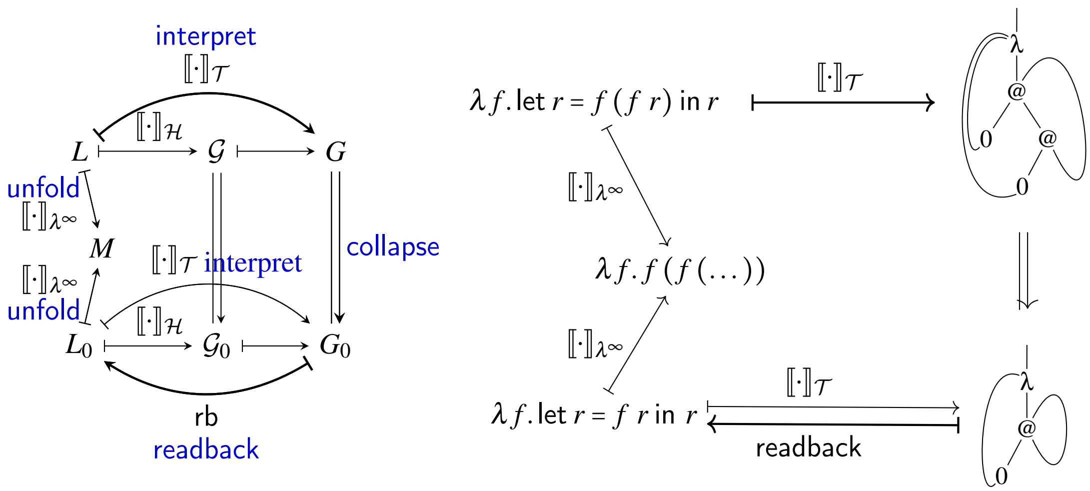
Jan and I wrote a string of papers together with as highlight the definition and efficient implementation of maximal sharing for functional programs in [11], presented on the International Conference on Functional Programming ICFP'14, Gothenburg, 2014. As part of our collaboration, Jan developed two highly useful and illustrative animation tools in Haskell (see descriptions on pages **??**–3).

Together we set out to bridge the gap between an evaluation method for functional programs that is known to be optimal in theory and the difficulty to make it beneficial to implementation practice. Starting from ideas that Jan had concerning a well-known optimization transformation in functional programming (developed in [15]) we focused on reasoning about infinite unfoldings of programs, and over the course of the project obtained the following main results:

1. Characterization of infinite unfoldings of programs ([8], RTA 2013),
2. Term graph representations ([10], TERMGRAPH 2013), and
3. Maximal Sharing for functional programs ([11], ICFP 2014).

Our collaboration culminated in the ICFP paper that demonstrated that theoretical results are able to lead to a practically useful compiler optimization. Likely *the* decisive factor for this success was that Jan had written a fantastic tool that implements and convincingly illustrates the maximal-sharing method.

These results obtained with Jan formed the basis of two invited presentations to which I have been invited: at the workshop TERMGRAPH 2018 at Oxford University, see [1], and at the workshop Computational Logic and Applications (CLA 2019), Université de Versailles, France, July 2, 2019, see [5].



1. term graph interpretations $[\![\cdot]\!]_\cdot$ of $\lambda_{\mathsf{letrec}}$-term L as:
   a. higher-order term graph $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$
2. bisimulation collapse $\Downarrow$ of first-order term graph G with as result G_0
3. readback rb of first-order term graph G_0 yielding $\lambda_{\mathsf{letrec}}$-term $L_0 = \mathsf{rb}(G_0)$.

Figure 1: Schematic representation of the maximal sharing method, and its application to a toy example: Maximum sharing proceeds of a $\lambda_{\mathsf{letrec}}$-term proceeds via three steps: interpretation as first-order term graphs, collapse of those via bisimilarity, and readback of $\lambda_{\mathsf{letrec}}$-terms from collapsed term graphs. On the top right these steps are illustrated for a redundant $\lambda_{\mathsf{letrec}}$-term formulation of a fixed-point combinator, yielding an efficient representation of such a combinator as $\lambda_{\mathsf{letrec}}$-term.

▷ *Maximal sharing prototype and illustration*:

Following the definition of maximally shared representations via a 'representation pipeline' in [11], this tool transforms a given functional program in the Core language of the λ-calculus with letrec into a term graph, and subsequently into a deterministic finite-state automaton (DFA). It prints intermediate representations, and graphically displays the obtained DFA. This DFA is then minimized, and finally a maximally shared representation of the original program is computed as the result.

▷ This tool [17] is available on Hackage via `http://hackage.haskell.org/package/maxsharing`.

References

[4] Stefan Blom (2001): *Term Graph Rewriting, Syntax and Sematics*. Ph.D. thesis, Vrije Universiteit Amsterdam. Available at `https://ir.cwi.nl/pub/29853/29853D.pdf` from webpage `https://ir.cwi.nl/pub/29853` for this thesis at CWI Amsterdam.

[5] Clemens Grabmayer (2019): *Modeling Terms in the λ-Calculus with letrec*. Invited talk at the Workshop *Computational Logic and Applications*, Université de Versailles, France, July 1–2. Slides available at `https://clegra.github.io/lf/CLA-2019-invited-talk.pdf`.

[6] Clemens Grabmayer & Vincent van Oostrom (2015): *Nested Term Graphs*. In Aart Middeldorp & Femke van Raamsdonk, editors: Post-Proceedings 8th International Workshop on *Computing with Terms and Graphs*, Vienna, Austria, July 13, 2014, *Electronic Proceedings in Theoretical Computer Science* 183, Open Publishing Association, pp. 48–65, doi:10.4204/EPTCS.183.4. ArXived at:1405.6380v2.

[7] Clemens Grabmayer & Jan Rochel (2012): *Expressibility in the Lambda-Calculus with Letrec*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.1208.2383. arXiv:1208.2383.

[8] Clemens Grabmayer & Jan Rochel (2013): *Expressibility in the Lambda Calculus with Mu*. In Femke van Raamsdonk, editor: *24th International Conference on Rewriting Techniques and Applications (RTA 2013)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 21, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 206–222, doi:10.4230/LIPIcs.RTA.2013.206. Available at `http://drops.dagstuhl.de/opus/volltexte/2013/4063`.

[9] Clemens Grabmayer & Jan Rochel (2013): *Expressibility in the Lambda Calculus with μ*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.1304.6284. arXiv:1304.6284. Extends [8].

[10] 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. ArXived at:1302.6338v1.

[11] 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.

[12] Clemens Grabmayer & Jan Rochel (2014): *Maximal Sharing in the Lambda Calculus with letrec*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.1401.1460. arXiv:1401.1460. Extends [11].

[13] Clemens Grabmayer & Jan Rochel (2014): *Term Graph Representations for Cyclic Lambda-Terms*. Technical Report, arxiv.org, doi:10.48550/arXiv.1304.6284. arXiv:1304.6284. Report extending [10] (proofs of main results added).

[14] 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`.

[15] Jan Rochel & Clemens Grabmayer (2011): *Repetitive Reduction Patterns in Lambda Calculus with Letrec*. In Rachid Echahed, editor: *Proceedings of the workshop TERMGRAPH 2011, 2 April 2011, Saarbrücken, Germany*, *Electronic Proceedings in Theoretical Computer Science* 48, Open Publishing Association, pp. 85–100, doi:10.4204/EPTCS.48.9. ArXived at:1102.2656v1.

[16] Jan Rochel & Clemens Grabmayer (2013): *Confluent Unfolding in the λ-calculus with* letrec. In: *Proceedings of IWC 2013 (2nd International Workshop on Confluence)*, pp. 17–22. Available at `http://www.jaist.ac.jp/~hirokawa/iwc2013/iwc2013.pdf`.

[17] Jan Rochel & Clemens Grabmayer (2014): *Maximal Sharing in the Lambda Calculus with letrec*. Implementation of the maximal sharing method described in [11, 12], available at `http://hackage.haskell.org/package/maxsharing/`.

[18] Jan Rochel & Clemens Grabmayer (2014): *Maximal Sharing in the Lambda Calculus with letrec*. Implementation of the maximal sharing method described in [11, 12], available at `http://hackage.haskell.org/package/maxsharing/`.

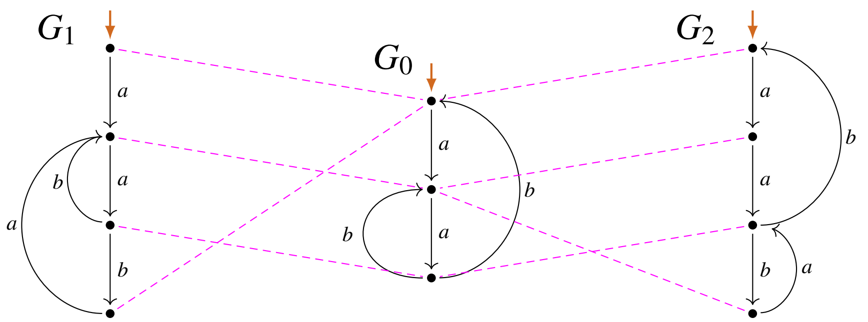
3 Recognizing Regular-Expression Processes

In [44] Milner introduced a process semantics $[\![\cdot]\!]_P$ that refines the standard language semantics $[\![\cdot]\!]_L$ for regular expressions. For regular expressions e that are constructed from constants 0, 1, letters over a

$$\frac{}{1{\Downarrow}} \qquad \frac{e_i{\Downarrow}}{(e_1+e_2){\Downarrow}} \qquad \frac{e_1{\Downarrow} \quad e_2{\Downarrow}}{(e_1\cdot e_2){\Downarrow}} \qquad \frac{}{(e^*){\Downarrow}}$$

$$\frac{}{a \xrightarrow{a} 1} \qquad \frac{e_i \xrightarrow{a} e_i'}{e_1+e_2 \xrightarrow{a} e_i'} \qquad \frac{e_1 \xrightarrow{a} e_1'}{e_1\cdot e_2 \xrightarrow{a} e_1'\cdot e_2} \qquad \frac{e_1{\Downarrow} \quad e_2 \xrightarrow{a} e_2'}{e_1\cdot e_2 \xrightarrow{a} e_2'} \qquad \frac{e \xrightarrow{a} e'}{e^* \xrightarrow{a} e'\cdot e^*}$$

Figure 2: Transition system specification $\mathcal{T}$ for computations enabled by regular expressions.



$P(a\cdot((a\cdot(b+b\cdot a))^*\cdot 0))$	$?\in im(P(\cdot))\ ?$	$P((1\cdot(((a\cdot a)\cdot(b\cdot a)^*)\cdot b)^*)\cdot 0)$
P-expressible, hence $[\![\cdot]\!]_P$-expr.	$[\![\cdot]\!]_P$-expressible	P-expressible, hence $[\![\cdot]\!]_P$-expr.
$a\cdot((a\cdot(b+b\cdot a))^*\cdot 0)$	$=_{[\![\cdot]\!]_P}$	$(1\cdot(((a\cdot a)\cdot(b\cdot a)^*)\cdot b)^*)\cdot 0$

Figure 3: Two process graphs G_1 and G_2 that are P-expressible, and hence $[\![\cdot]\!]_P$-expressible, because they are the process interpretations of regular expressions as indicated. G_1 and G_2 are bisimilar, as indicated via bisimulations (drawn as links ---) to their joint bisimulation collapse G_0. It follows that also G_0 is $[\![\cdot]\!]_P$-expressible, and that process semantics equality holds between the regular expressions with interpretations G_1 and G_2, respectively. In this example G_0 is actually also in the image of $P(\cdot)$, hence P-expressible, as witnessed for example by $G_0 = P(((1\cdot a)\cdot(a\cdot(b+b\cdot a))^*)\cdot 0)$.

given set A with the binary operators $+$ and $\cdot$, and the unary operator $(\cdot)^*$, Milner first defined a process interpretation $P(e)$ that can informally be described as follows: 0 is interpreted as a deadlocking process without any observable behavior, 1 as a process that terminates successfully immediately, letters from the set A are defined as atomic actions that lead to successful termination; the binary operators $+$ and $\cdot$ are interpreted as the operations of choice and concatenation of two processes, respectively, and the unary star operator $(\cdot)^*$ is interpreted as the operation of unbounded iteration of a process, but with the option to terminate successfully before each iteration.

Milner formalized the process interpretation in [44] as process graphs that are defined by induction on the structure of regular expressions. But soon afterwards a formal definition by means of a transition system specification (TSSs) that define finite labeled transition systems (LTSs) became more common. The TSS $\mathcal{T}$ in Figure 2 defines, via derivations from its axioms, labeled transitions $\xrightarrow{a}$ for actions a that occur in a regular expressions, and immediate successful termination via the unary predicate $\Downarrow$. The process interpretation $P(e)$ of a regular expression e is then defined as the sub-LTS that is defined by e in the LTS on regular expression that is defined via derivability in $\mathcal{T}$. See Figure 3 for suggestive examples of (bisimilar) process interpretations of rather simple regular expressions. In process graph illustrations

not $P(\cdot)$-expressible, and not $[\![\cdot]\!]_P$-expressible

$a\cdot(b+c)$ $\;=_{[\![\cdot]\!]_L}\;$ $\;\neq_{[\![\cdot]\!]_P}\;$ $a\cdot b + a\cdot c$

Figure 4: On the left: Two process graphs that are neither $P(\cdot)$-expressible (that is, not in the image of the process interpretation P) nor $[\![\cdot]\!]_P$-expressible (that is, not bisimilar to the process interpretation of any regular expression). On the right: two regular expressions with the same language semantics (associated language) but different process semantics, since the process interpretations are not bisimilar; therefore right-distributivity does not hold for $=_{[\![\cdot]\!]_P}$, which entails that fewer identities hold for $=_{[\![\cdot]\!]_P}$ than for $=_{[\![\cdot]\!]_L}$.

there and later we indicate the start vertex by a brown arrow $\rightarrow$, and the property of a vertex v to permit immediate successful termination by emphasizing v in brown as $\odot$ including an boldface ring.

Based on the process interpretation $P(\cdot)$, Milner then defined the process semantics of a regular expression e as $[\![e]\!]_P := [P(e)]_{\leftrightarrows}$ where $[P(e)]_{\leftrightarrows}$ is the equivalence class of $P(e)$ with respect to bisimilarity $\leftrightarrows$. In analogy to how language-semantics equality $=_{[\![\cdot]\!]_L}$ of regular expressions is defined from the language semantics $[\![\cdot]\!]_L$ (namely as $e =_{[\![\cdot]\!]_L} f$ if $L(e) = [\![e]\!]_L = [\![f]\!]_L = L(e)$, for all regular expressions e and f, where $L(g)$ is the language defined by a regular expression g) Milner was then interested in process-semantics equality $=_{[\![\cdot]\!]_P}$ that is defined, for all regular expressions e and f by:

$$\begin{aligned} e =_{[\![\cdot]\!]_P} f \;\; :\Longleftrightarrow \;\; & [\![e]\!]_P = [\![f]\!]_P \\ \Longleftrightarrow \;\; & P(e) \leftrightarrows P(f)\,. \end{aligned}$$

As the process interpretations of the regular expressions in Figure 4 are bisimilar, it follows that these regular expressions are linked by $=_{[\![\cdot]\!]_P}$.

Milner realized in [44] that the process semantics $[\![\cdot]\!]_P$ of regular expressions differs from the language semantics $[\![\cdot]\!]_L$ in at least two respects: first, $[\![\cdot]\!]_P$ is incomplete, and second, process-semantics equality $=_{[\![\cdot]\!]_P}$ satisfies fewer identities than language-semantics equality $=_{[\![\cdot]\!]_L}$.

We start by explaining incompleteness of $P\cdot$. Language semantics $L\cdot$ is complete in the following sense: every language that can be accepted by a finite-state automaton (a regular language) is the language that is defined by a regular expression; that is every regular language is $[\![\cdot]\!]_L$-expressible. However, a comparable statement does not hold for the process interpretation. Here we call a finite process graph *$[\![\cdot]\!]_P$-expressible* if it is bisimilar to a *P-expressible* process graph, by which we the process interpretation of some regular expression (and hence a graph in the image of $P(\cdot)$. While it is easier to argue that not very finite process graph is P-expressible, there are also finite process graphs that are not $[\![\cdot]\!]_P$-expressible, neither.

Process graphs that are in the image of the process interpretation P we here call *P-expressible*. Milner proved in [44] that the process graph $G_2^{(\mathrm{ne})}$ in Figure 4 not only is not P-expressible, but that it is also not $[\![\cdot]\!]_P$-expressible. He also conjectured that also $G_1^{(\mathrm{ne})}$ in Figure 4 is not $[\![\cdot]\!]_P$-expressible; this was later shown by Bosscher [24].

$$
\begin{array}{llll}
\text{(A1)} & e+(f+g)=(e+f)+g & \text{(A7)} & e=1\cdot e \\
\text{(A2)} & e+0=e & \text{(A8)} & e=e\cdot 1 \\
\text{(A3)} & e+f=f+e & \text{(A9)} & 0=0\cdot e \\
\text{(A4)} & e+e=e & \text{(A10)} & e^{*}=1+e\cdot e^{*} \\
\text{(A5)} & e\cdot(f\cdot g)=(e\cdot f)\cdot g & \text{(A11)} & e^{*}=(1+e)^{*} \\
\text{(A6)} & (e+f)\cdot g=e\cdot g+f\cdot g & &
\end{array}
\qquad
\frac{e=f\cdot e+g}{e=f^{*}\cdot g}\ \mathsf{RSP}^{*}\ (\text{if } f\not\Downarrow)
$$

Figure 5: Milner's equational proof system Mil for process semantics equality $=_{[\![\cdot]\!]_P}$ of regular expressions with the fixed-point rule RSP* in addition to the (not shown) basic rules for reasoning with equations (which guarantee that derivability in Mil is a congruence relation). From Mil the complete proof system for language equivalence $=_{[\![\cdot]\!]_L}$ due to Aanderaa arises by adding the axioms $e\cdot(f+g)=e\cdot f+e\cdot g$ and $e\cdot 0=0$ (which are not sound for $=_{[\![\cdot]\!]_P}$) and by dropping (A9) (which then is derivable).

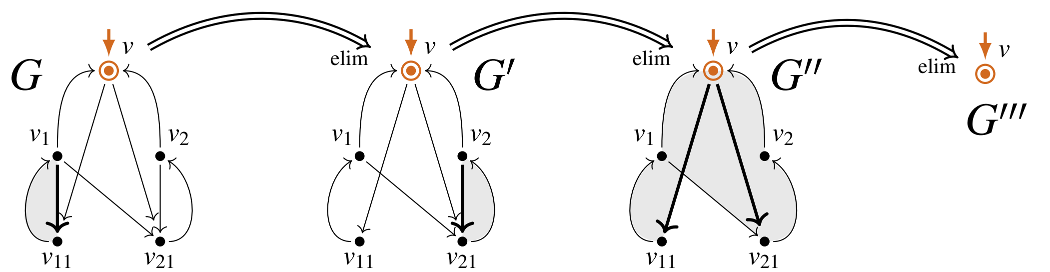


Figure 6: Example of a successful loop elimination process on the process graph G. Three elimination steps of loop subcharts, which are represented as shaded gray areas, lead to the process graph G''' without infinite behaviour. These steps witness that G satisfies the property LEE (as well as do G', G'' and G''').

Milner also noticed in [44] that some identities that hold for language-semantics equality $=_{[\![\cdot]\!]_L}$ are not true any longer for process semantics equality $=_{[\![\cdot]\!]_P}$. Most notably this is the case for right-distributivity $e\cdot(f+g)=e\cdot f+e\cdot g$, which is violated just as for the comparison of process terms via bisimilarity; see the well-known counterexample in Figure 4. The language-semantics identity $e\cdot 0=0$ is also violated in the process semantics. In order to define a natural sound adaptation (that we here designated by) Mil, see Figure 5, of the complete axiom systems for $=_{[\![\cdot]\!]_L}$ by Aanderaa [19] and Salomaa [45], Milner dropped these two identities from Aanderaa's system, but added the sound identity $0\cdot e=0$.

These two pecularities of the process semantics led Milner to formulating two questions concerning recognizability of expressible process graphs, and axiomatizability of process-semantics equality:

(E) How can $[\![\cdot]\!]_P$-expressible process graphs be characterized, that is, those finite process graphs that are bisimilar to process interpretations of regular expressions?

(A) Is the natural adaptation Mil to process-semantics equality $=_{[\![\cdot]\!]_P}$, see Figure 5, of Salomaa's and Aanderaa's complete proof systems for language-semantics equality $=_{[\![\cdot]\!]_L}$ complete for $=_{[\![\cdot]\!]_P}$?

While the decision problem underlying **(E)** has been shown to be solvable [21, 22] (but only with a super-exponential complexity bound), so far only partial solutions have been obtained for question **(A)**. These concern tailored restrictions of Milner's proof system that were shown to be complete for the following subclasses of star expressions:

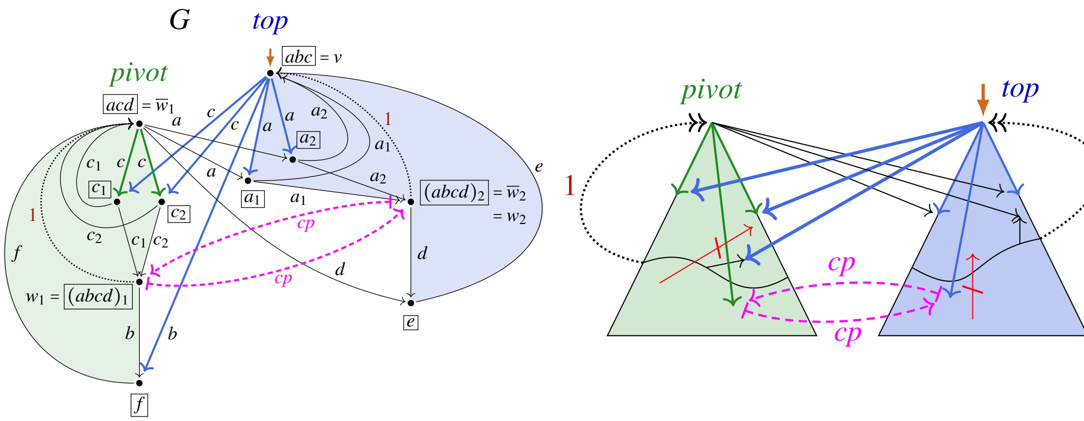


Figure 7: On the left: a prototypical example of a finite process graph G that is a twin-crystal. It consists of two interlinked parts, the top-part and the pivot-part, which by themselves are bisimulation collapsed, but contain vertices that have bisimilar counterparts in the other part. Dotted transitions labeled by 1 are empty transitions. The self-inverse counterpart function *cp* links bisimilar vertices in the two parts. On the right: schematic representation of a twin-crystal with top-part and pivot-part highlighted in color, and their interconnecting proper transitions.

(a) without 0 and 1, but with binary star iteration $e_1{}^{\circledast}e_2$ with iteration-part e_1 and exit-part e_2 instead of unary star (Fokkink and Zantema, 1994, [28]),

(b) with 0, and with iterations restricted to exit-less ones $(\cdot)^* \cdot 0$ in absence of 1 (Fokkink, 1997, [27]) and in the presence of 1 (Fokkink, 1996 [26]),

(c) without 0, and with restricted occurrences of 1 (Corradini, De Nicola, and Labella, 2002 [25]),

(d) '1-free' expressions formed with 0, without 1, but with binary iteration $\circledast$ (G, Fokkink, 2020, [41]).

While the classes (c) and (d) are incomparable, these results can be joined to apply to an encompassing proper subclass of the star expressions [41]. These partial results for **(A)** also yield partial results concerning **(E)**: expressibility modulo bisimilarity of a finite process graph by the process interpretation of a star expression in one of these classes is decidable in polynomial time.

- crystallization: reference to [36, 37] and the poster [38]
- reference to my work on the coinductive version cMil of Milner's system Mil in [32, 31, 39]

References

[19] Stål Aanderaa (1965): *On the Algebra of Regular Expressions*. Technical Report, Applied Mathematics, Harvard University.

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

[21] Jos Baeten, Flavio Corradini & Clemens Grabmayer (2005): *A Characterization of Regular Expressions under Bisimulation*. Technical Report, Technical University of Eindhoven. `https://pure.tue.nl/ws/files/2421293/200527.pdf` (TUE repository page `http://bit.ly/2xiqbXA`, shortened link).

[22] Jos Baeten, Flavio Corradini & Clemens Grabmayer (2007): *A Characterization of Regular Expressions Under Bisimulation*. *Journal of the ACM* 54(2), pp. 1–28, doi:10.1145/1219092.1219094.

[23] Jan Bergstra, Inge Bethke & Alban Ponse (1994): *Process Algebra with Iteration and Nesting*. The Computer Journal 37(4), pp. 243–258, doi:10.1093/comjnl/37.4.243.

[24] Doeko Bosscher (1997): *Grammars Modulo Bisimulation*. Ph.D. thesis, University of Amsterdam.

[25] Flavio Corradini, Rocco De Nicola & Anna Labella (2002): *An Equational Axiomatization of Bisimulation over Regular Expressions*. Journal of Logic and Computation 12(2), pp. 301–320, doi:10.1093/logcom/12.2.301.

[26] Wan Fokkink (1996): *An Axiomatization for the Terminal Cycle*. Technical Report, *Logic Group Preprint Series*, Vol. 167, Utrecht University.

[27] Wan Fokkink (1997): *Axiomatizations for the Perpetual Loop in Process Algebra*. In: Proc. ICALP'97, *LNCS* 1256, Springer, Berlin, Heidelberg, pp. 571–581, doi:10.1007/3-540-63165-8_212.

[28] Wan Fokkink & Hans Zantema (1994): *Basic Process Algebra with Iteration: Completeness of its Equational Axioms*. The Computer Journal 37(4), pp. 259–267, doi:10.1093/comjnl/37.4.259.

[29] Clemens Grabmayer (2020): *Structure-Constrained Process Graphs for the Process Semantics of Regular Expressions*. Technical Report, `arxiv.org`, doi:https://doi.org/10.48550/arXiv.2012.10869. arXiv:2012.10869. Report version of [34].

[30] Clemens Grabmayer (2020): *Structure-Constrained Process Graphs for the Process Semantics of Regular Expressions*. In: Pre-Proceedings of the FSCD workshop TERMGRAPH 2020, July 5, 2020, pp. 1–9. `http://termgraph.org.uk/2020/preproceedings/grabmayer.pdf`.

[31] Clemens Grabmayer (2021): *A Coinductive Version of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.2108.13104. arXiv:2108.13104. Extended report for [32].

[32] Clemens Grabmayer (2021): *A Coinductive Version of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. In Fabio Gadducci & Alexandra Silva, editors: 9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021), Leibniz International Proceedings in Informatics (LIPIcs) 211, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 16:1–16:23, doi:10.4230/LIPIcs.CALCO.2021.16. Extended report see [31].

[33] Clemens Grabmayer (2021): *Bisimulation Slices and Transfer Functions*. Technical report, Reykjavik University. Extended abstract for the 32nd Nordic Workshop on Programming Theory (NWPT 2021), `http://icetcs.ru.is/nwpt21/abstracts/paper5.pdf`.

[34] Clemens Grabmayer (2021): *Structure-Constrained Process Graphs for the Process Semantics of Regular Expressions*. Electronic Proceedings in Theoretical Computer Science 334, p. 29–45, doi:10.4204/eptcs.334.3. Extended report for [29].

[35] Clemens Grabmayer (2022): *A Coinductive Reformulation of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.2203.09501. arXiv:2203.09501. Special-issue journal submission, whose development started from [32, 31].

[36] Clemens Grabmayer (2022): *Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete (Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions)*. In: Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '22, Association for Computing Machinery, New York, NY, USA, pp. 1–13.

[37] Clemens Grabmayer (2022): *Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete (Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions)*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.2209.12188. arXiv:2209.12188.

[38] Clemens Grabmayer (2022): *Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete (Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions)*. Poster presented at LICS'22, Technion, Haifa, Israel, August 5. `https://clegra.github.io/lf/poster-lics-2022.pdf`.

[39] Clemens Grabmayer (2023): *A Coinductive Reformulation of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. Logical Methods in Computer Science Volume 19, Issue 2, doi:10.46298/lmcs-19(2:17)2023. Available at `https://lmcs.episciences.org/11519`.

[40] Clemens Grabmayer (2023): *The Image of the Process Interpretation of Regular Expressions is Not Closed Under Bisimulation Collapse*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.2303.08553. arXiv:2303.08553.

[41] Clemens Grabmayer & Wan Fokkink (2020): *A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity*. In: Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '20, Association for Computing Machinery, New York, NY, USA, p. 465–478, doi:10.1145/3373718.3394744.

[42] Clemens Grabmayer & Wan Fokkink (2020): *A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.2004.12740. arXiv:2004.12740. Report version of [41].

[43] Stephen C. Kleene (1951): *Representation of Events in Nerve Nets and Finite Automata*. In: Automata Studies, Princeton University Press, Princeton, New Jersey, USA, pp. 3–42, doi:10.1515/9781400882618-002.

[44] 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.

[45] 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.

4 Current and Future Work

In this section I touch on my present research, as well as list and motivate research questions and projects that have developed out of the work summarized in the past two sections.

4.1 Maximal Sharing at Run Time

Apart from using the maximal-sharing method for functional programs as a static optimization during compilation, one of the ideas[1] for applications that Rochel and I gathered in [11] was that maximal sharing could be used as an optimization also repeatedly at run-time. Making that thought fruitful, however, requires that representations of programs that are used in graph evaluators can be linked rather directly with λ-term-graph representations of $\lambda_{\mathsf{letrec}}$-terms that are used for the maximal-sharing method. This is because graph evaluators in implementations of functional languages typically use supercombinator representations of $\lambda_{\mathsf{letrec}}$-terms, and much computational overhead has to be expected in transformations to and from λ-term-graphs. However, supercombinator reduction in graph evaluators intuitively corresponds to *scope-sharing* forms of β-reduction (as opposed to the *context-sharing* forms of *parallel* or *optimal* β*-reduction* evaluators) of which implementation on λ-term-graphs is at least conceivable.

Research Question 1. *Coupling of maximal sharing with evaluation, generally, and more specifically:*

(i) Can the maximal-sharing method for terms in the λ-calculus with letrec *be coupled naturally with an efficient evaluation method? For example, with a standard graph-evaluation implementation.*

(ii) Do λ-term-graphs (which represent $\lambda_{\mathsf{letrec}}$-terms) permit a representation as interaction nets or as port graphs for which a form of β-reduction can be defined that is able to preserves, by means of multi-steps of interactions, scopes and also λ-term-graph form?

[1] I want to credit Jan Rochel with having been much more persistent than myself in trying to explore that idea.

In communication after the workshop, but related to my talk, Ian Mackie for pointing me to his interaction-net based implementation [48] of an evaluation method for the λ-calculus. I am grateful for this reference, because the interaction-net representation of λ-terms in [48] bears a close resemblance with λ-term-graphs, and facilitates scope preserving β-reduction. However, it likely is a quite non-trivial question to relate the two formalisms, λ-term-graphs and interaction-net representations of λ-terms in [48], closely together. This seems to be a promising in-road to approach part (ii) of Research Question 1.

4.2 Crystallization: Proof Verification, and Application to the Expressibility Problem

Currently I am writing two articles that will provide the details of the completeness proof of Milner's proof system Mil. The first article is going to explain the motivation of the crystallization process for process interpretations of regular expressions: a limit to the minimization under bisimulation of process graphs that are expressible by a regular expression. This limit will be shown specifically for the process graph G in Figure 7 with 1-transitions. The second article will detail the crystallization procedure by which process graphs with the property LEE (which are $[\![\cdot]\!]_P$-expressible) are minimized under bisimulation to obtain process graphs with LEE that are close to their bisimulation collapse. This central result will then be used, as explained in [36], to show that Milner's proof system Mil is complete with respect to process semantics equality $=_{[\![\cdot]\!]_P}$.

While the details of this completeness proof can be explained clearly conceptually, answering Milner's question **(A)**, a verification of the crystallization procedure and the completeness proof of Mil with respect to $=_{[\![\cdot]\!]_P}$ forms an important goal for me.

Research Project 2. *Formalization of the proofs for crystallization, and completeness of* Mil*:*

(a) Develop formalizations of structure constraints for process graphs in order to verify the correctness of the crystallization procedure for process graphs with LEE *by a proof assistant.*

(b) Use the correctness proof of crystallization to verify the completeness proof of Milner's proof system Mil *by a proof assistant.*

Separately I am working out now the proof that the loop existence and elimination property LEE can be decided in polynomial time. As a consequence of this fact it will follow that the restriction of the expressibility problem **(E)** to expressibility by regular expressions that are '1-free under star' can be solved efficiently. This is because the methods and results in [41] imply that a finite process graph G is expressible by a regular expression that '1-free under star' if and only if the bisimulation collapse of G satisfies LEE. Then it follows that expressibility of finite process graphs by regular expressions that are 1-free under star can be decided in polynomial time.

Research Question 3. *Is the problem of whether a finite process graph is* $[\![\cdot]\!]_P$*-expressible efficiently decidable? That is, is there a polynomial decision algorithm for it? Or is* $[\![\cdot]\!]_P$*-expressibility at least fixed-parameter tractable (in* FPT*) for interesting parameterizations?*

References

[46] Beniamino Accattoli & Ugo Dal Lago (2016): *(Leftmost-Outermost) Beta Reduction is Invariant, Indeed. Logical Methods in Computer Science* Volume 12, Issue 1, doi:10.2168/LMCS-12(1:4)2016. Available at `http://lmcs.episciences.org/1627`.

[47] Clemens Grabmayer (2019): *Linear Depth Increase of Lambda Terms in Leftmost-Outermost Beta-Reduction Rewrite Sequences*. Technical Report, `arxiv.org`. arXiv:1603.0730.

[48] Ian Mackie (1998): *YALE: Yet Another Lambda Evaluator Based on Interaction Nets*. In: *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, ICFP '98, Association for Computing Machinery, New York, NY, USA, p. 117–128, doi:10.1145/289423.289434.