Control-Flow Refinement for Complexity Analysis of Probabilistic Programs

Nils Lommen
RWTH Aachen University, Germany

Éléanore Meyer
RWTH Aachen University, Germany

Jürgen Giesl
RWTH Aachen University, Germany

Abstract

Recently, we showed how to use control-flow refinement (CFR) to improve automatic complexity analysis of integer programs. While up to now CFR was limited to classical programs, we extend CFR to *probabilistic* programs and show its soundness for complexity analysis. To demonstrate its benefits, we implemented our new CFR technique in our complexity analysis tool KoAT.

2012 ACM Subject Classification Software and its engineering → Software verification and validation; Theory of computation → Automated reasoning; Theory of computation → Logic and verification; Theory of computation → Probabilistic computation

Keywords and phrases (Positive) Almost-Sure Termination, Control-Flow Refinement, Complexity Analysis

Related Version Extended Abstract of our Conference Article [26]

1 A Birds-Eye-View on Control-Flow Refinement

There exist numerous tools for complexity analysis of (non-probabilistic) programs, e.g., [2–6, 10, 11, 15, 16, 18, 19, 24, 25, 28, 30, 32]. Our tool KoAT infers upper runtime and size bounds for (non-probabilistic) integer programs in a modular way by analyzing subprograms separately and lifting the obtained results to global bounds on the whole program [10]. Recently, we developed several improvements of KoAT [18, 24, 25] and showed that incorporating control-flow refinement (CFR) [13, 14] increases the power of automated complexity analysis significantly [18].

There are also several approaches for complexity analysis of *probabilistic* programs, e.g., [1, 7, 9, 21–23, 27, 29, 31, 34]. In particular, we also adapted KoAT's approach for runtime and size bounds, and introduced a modular framework for automated complexity analysis of probabilistic integer programs in [27]. However, the improvements of KoAT from [18, 24, 25] had not yet been adapted to the probabilistic setting. In particular, we are not aware of any existing technique to combine CFR with complexity analysis of probabilistic programs.

Thus, we develop a novel CFR technique for probabilistic programs which could be used as a black box by every complexity analysis tool. Moreover, to reduce the overhead by CFR, we integrate CFR natively into KoAT by calling it on-demand in a modular way. Our experiments show that CFR increases the power of KoAT for complexity analysis of probabilistic programs substantially.

The idea of CFR is to gain information on the values of program variables and to sort out infeasible program paths. For example, consider the probabilistic **while**-loop (1). Here, we flip a (fair) coin and either set x to 0 or do nothing.

$$\textbf{while}\ \ x > 0\ \ \textbf{do}\ \ x \leftarrow 0 \ \oplus_{1/2}\ \texttt{noop}\ \ \textbf{end} \tag{1}$$

The update $x \leftarrow 0$ is in a loop. However, after setting x to 0, the loop cannot be executed again. To simplify its analysis, CFR "unrolls" the loop resulting in (2).

$$\begin{array}{l}\textbf{while } x > 0 \textbf{ do } \texttt{break} \oplus_{1/2} \texttt{noop} \textbf{ end}\\ \textbf{if } x > 0 \textbf{ then } x \leftarrow 0 \textbf{ end}\end{array} \tag{2}$$

Here, x is updated in a separate, *non-probabilistic* **if**-statement and the loop does not change variables. Thus, we sorted out (infeasible) paths where $x \leftarrow 0$ was executed repeatedly. Now, techniques for probabilistic programs can be used for the **while**-loop. The rest of the program can be analyzed by techniques for non-probabilistic programs. In particular, this is important if (1) is part of a larger program.

Our novel CFR algorithm for *probabilistic* integer programs is based on the partial evaluation technique for non-probabilistic programs from [13, 14, 18]. In particular, our algorithm coincides with the classical CFR technique when the program is non-probabilistic. The goal of CFR is to transform a program $\mathcal{P}$ into a program $\mathcal{P}'$ which is "easier" to analyze. In the full version of this paper, we prove that both $\mathcal{P}$ and $\mathcal{P}'$ have the same *expected* runtime (see [26, Thm. 4]). Thus, our approach is not only sound but it also does not increase the expected runtime. We apply CFR only *on-demand* on a subprogram (thus, CFR can be performed in a *modular* way for different subprograms). In practice, we choose the subprogram heuristically and use CFR only on parts of the program where our currently inferred runtime bounds are "not yet good enough".

2 Implementation and Evaluation

Up to now, our complexity analyzer KoAT used the tool iRankFinder [13] for CFR of non-probabilistic programs [18]. To demonstrate the benefits of CFR for complexity analysis of probabilistic programs, we now replaced the call to iRankFinder in KoAT by a native implementation of our new CFR algorithm. KoAT is written in OCaml and it uses Z3 [12] for SMT solving, Apron [20] to generate invariants, and the Parma Polyhedra Library [8] for computations with polyhedra.

We used all 75 probabilistic benchmarks from [27, 29] and added 15 new benchmarks including our leading example and problems adapted from the *Termination Problem Data Base* [33], e.g., a probabilistic version of McCarthy's 91 function. Our benchmarks also contain examples where CFR is useful even if it cannot separate probabilistic from non-probabilistic program parts as in our leading example.

Table 1 shows the results of our experiments. We compared the configuration of KoAT with CFR ("KoAT+CFR") against KoAT without CFR. Moreover, as in [27], we also compared with the main other recent tools for inferring upper bounds on the expected runtimes of probabilistic integer programs (Absynth [29] and eco-imp [7]). As in the *Termination Competition* [17], we used a timeout of 5 minutes per example. The first entry in every cell is the number of benchmarks for which the tool inferred the respective bound. In brackets, we give the corresponding number when only regarding our new examples. The runtime bounds computed by the tools are compared asymptotically as functions which depend on the largest initial absolute value n of all program variables. So for example, KoAT+CFR finds a finite expected runtime bound for 84 of the 90 examples. A linear expected bound (i.e., in $\mathcal{O}(n)$) is found for 56 of these 84 examples, where 12 of these benchmarks are from our new set. AVG(s) is the average runtime in seconds on all benchmarks and AVG$^+$(s) is the average runtime on all successful runs.

	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{>2})$	$\mathcal{O}(EXP)$	$< \omega$	AVG^{+}(s)	AVG(s)
KoAT+CFR	11 (2)	56 (12)	14	2	1	84 (14)	11.68	11.34
KoAT	9	41 (1)	16 (1)	2	1	69 (2)	2.71	2.41
Absynth	7	35	9	0	0	51	2.86	37.48
eco-imp	8	35	6	0	0	49	0.34	68.02

Table 1 Evaluation of CFR on Probabilistic Programs

The experiments show that similar to its benefits for non-probabilistic programs [18], CFR also increases the power of automated complexity analysis for probabilistic programs substantially, while the runtime of the analyzer may become longer since CFR increases the size of the program. The experiments also indicate that a related CFR technique is not available in the other complexity analyzers. Thus, we conjecture that other tools for complexity or termination analysis of PIPs would also benefit from the integration of our CFR technique.
KoAT's source code, a binary, and a Docker image are available at:

https://koat.verify.rwth-aachen.de/prob_cfr

The website also explains how to use our CFR implementation separately (without the rest of KoAT), in order to access it as a black box by other tools. Moreover, the website provides a *web interface* to directly run KoAT online, and details on our experiments, including our benchmark collection.

References

1 Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: An efficient approach to termination of probabilistic programs. *Proc. ACM Program. Lang.*, 2(POPL), 2017. doi:10.1145/3158122.

2 Elvira Albert, Puri Arenas, Samir Genaim, and Germán Puebla. Automatic inference of upper bounds for recurrence relations in cost analysis. In *Proc. SAS*, LNCS 5079, pages 221–237, 2008. doi:10.1007/978-3-540-69166-2_15.

3 Elvira Albert, Puri Arenas, Samir Genaim, Germán Puebla, and Damiano Zanardini. Cost analysis of object-oriented bytecode programs. *Theor. Comput. Sci.*, 413(1):142–159, 2012. doi:10.1016/j.tcs.2011.07.009.

4 Elvira Albert, Miquel Bofill, Cristina Borralleras, Enrique Martín-Martín, and Albert Rubio. Resource analysis driven by (conditional) termination proofs. *Theory Pract. Log. Program.*, 19(5-6):722–739, 2019. doi:10.1017/S1471068419000152.

5 Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *Proc. SAS*, LNCS 6337, pages 117–133, 2010. doi:10.1007/978-3-642-15769-1_8.

6 Martin Avanzini and Georg Moser. A Combination Framework for Complexity. In *Proc. RTA*, LIPIcs 21, pages 55–70, 2013. doi:10.4230/LIPIcs.RTA.2013.55.

7 Martin Avanzini, Georg Moser, and Michael Schaper. A modular cost analysis for probabilistic programs. *Proc. ACM Program. Lang.*, 4(OOPSLA), 2020. URL: https://doi.org/10.1145/3428240.

8 Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program.*, 72:3–21, 2008. doi:10.1016/j.scico.2007.08.001.

9 Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Lena Verscht. A calculus for amortized expected runtimes. *Proc. ACM Program. Lang.*, 7(POPL), 2023. doi:10.1145/3571260.

10 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl. Analyzing Runtime and Size Complexity of Integer Programs. *ACM Trans. Program. Lang. Syst.*, 38:1–50, 2016. doi:10.1145/2866575.

11 Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. Compositional certified resource bounds. In *Proc. PLDI*, pages 467–478, 2015. doi:10.1145/2737924.2737955.

12 Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Proc. TACAS*, LNCS 4963, pages 337–340, 2008. doi:10.1007/978-3-540-78800-3_24.

13 Jesús J. Doménech and Samir Genaim. iRankFinder. In *Proc. WST*, page 83, 2018. http://wst2018.webs.upv.es/wst2018proceedings.pdf.

14 Jesús J. Doménech, John P. Gallagher, and Samir Genaim. Control-flow refinement by partial evaluation, and its application to termination and cost analysis. *Theory Pract. Log. Program.*, 19(5-6):990–1005, 2019. doi:10.1017/S1471068419000310.

15 Antonio Flores-Montoya. Upper and lower amortized cost bounds of programs expressed as cost relations. In *Proc. FM*, LNCS 9995, pages 254–273, 2016. doi:10.1007/978-3-319-48989-6_16.

16 Florian Frohn and Jürgen Giesl. Complexity analysis for Java with AProVE. In *Proc. iFM*, LNCS 10510, pages 85–101, 2017. doi:10.1007/978-3-319-66845-1_6.

17 Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In *Proc. TACAS*, LNCS 11429, pages 156–166, 2019. doi:10.1007/978-3-030-17502-3_10.

18 Jürgen Giesl, Nils Lommen, Marcel Hark, and Fabian Meyer. Improving Automatic Complexity Analysis of Integer Programs. In *The Logic of Software. A Tasting Menu of Formal Methods*, LNCS 13360, pages 193–228, 2022. doi:10.1007/978-3-031-08166-8_10.

19 Jan Hoffmann, Ankush Das, and Shu-Chun Weng. Towards automatic resource bound analysis for OCaml. In *Proc. POPL*, pages 359–373, 2017. doi:10.1145/3009837.3009842.

20 Bertrand Jeannet and Antoine Miné. Apron: A library of numerical abstract domains for static analysis. In *Proc. CAV*, LNCS 5643, pages 661–667, 2009. doi:10.1007/978-3-642-02658-4_52.

21 Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest precondition reasoning for expected runtimes of randomized algorithms. *J. ACM*, 65:1–68, 2018. doi:10.1145/3208102.

22 Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Expected runtime analyis by program verification. In Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva, editors, *Foundations of Probabilistic Programming*, page 185–220. Cambridge University Press, 2020. doi:10.1017/9781108770750.007.

23 Lorenz Leutgeb, Georg Moser, and Florian Zuleger. Automated expected amortised cost analysis of probabilistic data structures. In *Proc. CAV*, LNCS 13372, pages 70–91, 2022. doi:10.1007/978-3-031-13188-2_4.

24 Nils Lommen, Fabian Meyer, and Jürgen Giesl. Automatic Complexity Analysis of Integer Programs via Triangular Weakly Non-Linear Loops. In *Proc. IJCAR*, LNCS 13385, pages 734–754, 2022. doi:10.1007/978-3-031-10769-6_43.

25 Nils Lommen and Jürgen Giesl. Targeting Completeness: Using Closed Forms for Size Bounds of Integer Programs. In *Proc. FroCoS*, LNCS 14279, pages 3–22, 2023. doi:10.1007/978-3-031-43369-6_1.

26 Nils Lommen, Éléanore Meyer, and Jürgen Giesl. Control-Flow Refinement for Complexity Analysis of Probabilistic Programs in KoAT (Short Paper). In *Proc. IJCAR*, LNCS 14739, pages 233–243, 2024. doi:10.1007/978-3-031-63498-7_14.

27 Fabian Meyer, Marcel Hark, and Jürgen Giesl. Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes. In *Proc. TACAS*, LNCS 12651, pages 250–269, 2021. doi:10.1007/978-3-030-72016-2_14.

28 Georg Moser and Michael Schaper. From Jinja bytecode to term rewriting: A complexity reflecting transformation. *Inf. Comput.*, 261:116–143, 2018. doi:10.1016/j.ic.2018.05.007.

29 Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: Resource analysis for probabilistic programs. In *Proc. PLDI*, pages 496–512, 2018. URL: https://doi.org/10.1145/3192366.3192394.

30 Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs. *J. Autom. Reason.*, 51:27–56, 2013. doi:10.1007/s10817-013-9277-6.

31 Philipp Schröer, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. A Deductive Verification Infrastructure for Probabilistic Programs. *Proc. ACM Program. Lang.*, 7(OOPSLA):2052–2082, 2023. doi:10.1145/3622870.

32 Moritz Sinn, Florian Zuleger, and Helmut Veith. Complexity and resource bound analysis of imperative programs using difference constraints. *J. Autom. Reason.*, 59(1):3–45, 2017. doi:10.1007/s10817-016-9402-4.

33 TPDB (Termination Problem Data Base). URL: https://github.com/TermCOMP/TPDB.

34 Di Wang, David M. Kahn, and Jan Hoffmann. Raising expectations: Automating expected cost analysis with types. *Proc. ACM Program. Lang.*, 4(ICFP), 2020. URL: https://doi.org/10.1145/3408992.