Algebra-Based Loop Analysis

Laura Kovács
TU Wien
Vienna, Austria

ABSTRACT

Automating loop analysis, and in particular synthesizing loop invariants, is a central challenge in the computer-aided verification of programs with loops, with applications in compiler optimization, probabilistic programming and IT security. While this challenge is in general undecidable, several techniques have emerged to automatically summarize the functional behaviour of software loops, thus providing inductive loop invariants that may prevent programmers from introducing errors while making changes in their code. In this tutorial, we show that novel combinations of methods from computer algebra, algorithmic combinatorics and static loop analysis provide powerful workhorses to derive (all) polynomial loop invariants, synthesize affine loops from invariants, and infer quantitative properties over the value distributions of probabilistic loop variables.

CCS CONCEPTS

• **Theory of computation** → **Invariants**; **Logic and verification**;
• **Mathematics of computing** → **Combinatorics**.

KEYWORDS

Program Verification, Loop Synthesis, Loop Invariants, Algebraic Recurrences, Polynomial Ideals

ACM Reference Format:
Laura Kovács. 2023. Algebra-Based Loop Analysis. In *International Symposium on Symbolic and Algebraic Computation 2023 (ISSAC 2023), July 24–27, 2023, Tromsø, Norway*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3597066.3597150

OVERVIEW

One of the main challenges in computer-aided formal software verification comes with the burden to verify or synthesize recursive behavior arising from (unbounded) software loops.

Intuitively, formal verification assigns to every program statement meaning in form of a logical formula, and then tries to prove that every formula is implied by the previous ones, starting with the precondition [3–5]. If we are able to prove that eventually the postconditions holds, we established the correctness of the program with respect to the given precondition and postcondition. Within this process of deductive verification, arguably the most challenging program constructs are the ones with recursive behavior such as loops. For (unbounded) loops, it is not known how often a loop is iterated when statically analyzing the program, that is, there is no straightforward way to assign a logical formula to loops. Formal verification therefore requires additional program assertions for loops in the form of *inductive loop invariants*. Intuitively, a loop invariant is a formal description of the behavior of the loop, expressing loop properties that hold before and after each loop iteration. In order to automate formal verification, we need rigorous techniques for automatically inferring and analyzing loop invariants.

The present tutorial overviews algebra-based algorithms for loop analysis. The key ingredients of our work root in the combination of the following four approaches:

(i) *static analysis*, to model loops as algebraic recurrence equations over the loop counter and to impose structural constraints over loops in order to guarantee that the resulting recurrences equations can be reduced to linear recurrences with constant coefficients [10];
(ii) *symbolic summation and polynomial algebra*, to solve recurrences and eliminate loop counters/recurrence variables by applying Gröbner basis computation over closed-forms [7, 9];
(iii) *applied statistics*, to compute quantitative loop invariants over higher-order statistical moments of polynomial probabilistic loops [11];
(iv) *automated reasoning*, to simplify loop constraints and generate invariants towards ensuring functional correctness of loops [6, 8].

Illustrative Examples

We motivate (applications of) our work with the loops of Figure 1. Within *invariant synthesis*, we compute the polynomial relation $c = n^3$ as a *polynomial loop invariant* of the polynomial loop of Figure 1. On the other hand, within *loop synthesis*, we strength reduce the polynomial loop of Figure 1 to the linear loop of Figure 1, by replacing loop variable multiplications with additions among program variables; doing so, we ensure that the linear loop of Figure 1 satisfies the polynomial loop invariant $c = n^3$ of the polynomial loop of Figure 1. In other words, the polynomial and linear loops of Figure 1 both satisfy the invariant $c = n^3$, and hence these loops are equivalent modulo the invariant $c = n^3$.

While synthesizing loop invariants from loops is crucial for formal software verification, synthesizing loops from loop invariants is in particular useful within program optimization and repair [2]. The combination of the algebraic techniques presented in this tutorial therefore provide efficient workhorses both for software verification and compiler optimization. The strength of this combined framework stems from the fact that we compute the set of *all polynomial equality invariants* for a given (polynomial) loop, that is, we precisely capture the behavior of the loop. Furthermore, the algebraic problem constructed by our synthesis procedure precisely

ISSAC 2023, July 24–27, 2023, Tromsø, Norway

ACM ISBN 979-8-4007-0039-2/23/07.
https://doi.org/10.1145/3597066.3597150

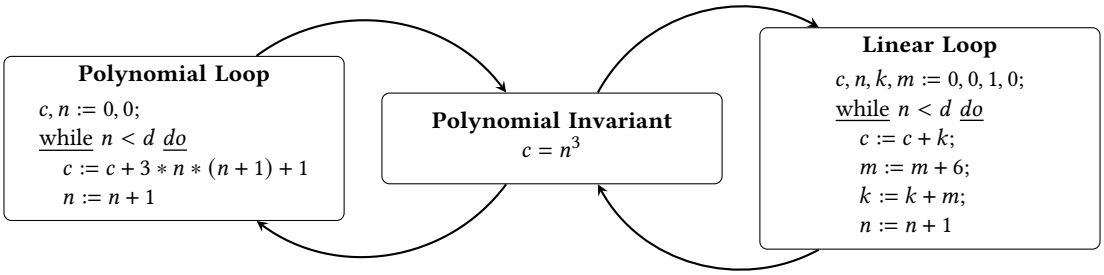

Figure 1: Algebra-Based Loop Analysis - Synthesizing Invariants and Loops

characterizes the set of *all (linear) loops*, modulo additional structural constraints, that satisfy the polynomial invariants given by the invariant generation procedure.

The central aspect of our work is that we represent numeric loops as systems of recurrence equations and employ properties of those recurrence systems to either generate invariants for a given loop or synthesize loops satisfying a given invariant. Our framework naturally extends to arbitrary loops whose functional summaries can be described by or reduced to linear recurrences with constant coefficients over loop counters [1], allowing for example the quantitative analysis of probabilistic loops by solving recurrences over expected values of loop variables [11].

ACKNOWLEDGMENTS

The work presented in this tutorials touches upon joint works with Daneshvar Amrollahi (TU Wien), Ezio Bartocci (TU Wien), Nikolaj Bjørner (Microsoft Research), Tudor Jebelean (RISC-Linz), Andreas Humenberger (TU Wien alumni), Maximilian Jaroscheck (TU Wien alumni), George Kenison (TU Wien), Marcel Moosbrugger (TU Wien), Miroslav Stankovic (TU Wien), and Anton Varonka (TU Wien). We acknowledge funding from the ERC consolidator grant ARTIST 101002685, the WWTF grant ProbInG ICT19-018, and the EU Marie Sklodowska-Curie Doctoral Network LogiCS@TU Wien Grant Nr. 101034440.

REFERENCES

[1] Daneshvar Amrollahi, Ezio Bartocci, George Kenison, Laura Kovács, Marcel Moosbrugger, and Miroslav Stankovic. 2022. Solving Invariant Generation for Unsolvable Loops. In *Proc. of SAS (Lecture Notes in Computer Science, Vol. 13790)*. Springer, 19–43. https://doi.org/10.1007/978-3-031-22308-2_3

[2] Keith D. Cooper, L. Taylor Simpson, and Christopher A. Vick. 2001. Operator Strength Reduction. *ACM Trans. Program. Lang. Syst.* 23, 5 (2001), 603–625.

[3] Edsger W. Dijkstra. 1976. *A Discipline of Programming*. Prentice-Hall.

[4] Robert W. Floyd. 1967. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics* 19 (1967), 19–32.

[5] C. A. R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (1969), 576–580.

[6] Andreas Humenberger, Daneshvar Amrollahi, Nikolaj S. Bjørner, and Laura Kovács. 2022. Algebra-Based Reasoning for Loop Synthesis. *Formal Aspects Comput.* 34, 1 (2022), 1–31. https://doi.org/10.1145/3527458

[7] Andreas Humenberger, Maximilian Jaroschek, and Laura Kovács. 2018. Aligator.jl - A Julia Package for Loop Invariant Generation. In *Proc. of CICM (Lecture Notes in Computer Science, Vol. 11006)*. Springer, 111–117. https://doi.org/10.1007/978-3-319-96812-4_10

[8] Andreas Humenberger, Maximilian Jaroschek, and Laura Kovács. 2018. Invariant Generation for Multi-Path Loops with Polynomial Assignments. In *Proc. of VMCAI (Lecture Notes in Computer Science, Vol. 10747)*. Springer, 226–246. https://doi.org/10.1007/978-3-319-73721-8_11

[9] Laura Kovács. 2008. Aligator: A Mathematica Package for Invariant Generation (System Description). In *Proc. of IJCAR (Lecture Notes in Computer Science, Vol. 5195)*. Springer, 275–282. https://doi.org/10.1007/978-3-540-71070-7_22

[10] Laura Kovács. 2008. Reasoning Algebraically About P-Solvable Loops. In *Proc. of TACAS (Lecture Notes in Computer Science, Vol. 4963)*. Springer, 249–264. https://doi.org/10.1007/978-3-540-78800-3_18

[11] Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Laura Kovács. 2022. This is the Moment for Probabilistic Loops. *Proc. ACM Program. Lang.* 6, OOPSLA2 (2022), 1497–1525. https://doi.org/10.1145/3563341