Parametric Bitwidth Rewrite Verification via Equality Saturation (work in progress)

Luigi Rinaldi
Imperial College London

Yann Herklotz
EPFL

John Wickerson
Imperial College London

Samuel Coward
Intel Corporation

ABSTRACT

Systems for rewriting arithmetic expressions play a crucial role in several modern hardware and software toolchains, such as datapath optimisers and peephole rewriting. Prior work has introduced techniques for formally verifying rewrites that either fix a concrete bitwidth or are parametrised by a single bitwidth. But many useful rewrites, such as associativity rules, involve more than one bitwidth parameter, making them conditional on these parameters, and prior work cannot reason about them. We are developing a tool that extends rewrite verification to *multiple* bitwidth parameters, leveraging e-graph term rewriting over uninterpreted functions. Our preliminary results highlight cases where our approach successfully verifies rewrites that other tools fail to prove. In the longer term, we aim to build an automated, Alive-style, push-button verifier for parametric hardware designs.

1 INTRODUCTION

Recent hardware optimisation tools have employed term rewriting over bitvector expressions to discover optimisations in RTL design at the arithmetic level [4, 5, 7, 19]. At the heart of these tools are conditional rewrites, *lhs* and *rhs* expressions containing parametric bitwidth terms which are equal if a condition is true. The rewrites usually reflect mathematical identities, such as associativity or distributivity of addition and multiplication, or identities containing shift operations. Consider the following circuit $((a_5 + b_3)_6 + c_3)_6$, where the subscript denotes the bitwidth of the term, an obvious optimisation would be to apply associativity of addition to obtain $(a_5 + (b_3 + c_3)_4)_6$ reducing a 6-bit adder to a 4-bit one. This rewrite is, in general, not always valid, and depends on the bitwidths of the operators and operands.

Verifying the correctness of these conditional rewrites, defined over parametric bitwidth terms, falls outside of the scope of current solvers. Instead, [5, 7] rely on industrial equivalence checkers to perform translation validation on the optimised RTL. Equivalence checkers also rely on term rewriting [10] to discover equivalences between arithmetic expressions in order to avoid bit-blasting and the exponential runtime it incurs. Recent work [8] highlights the limitations of this arithmetic rewriting, showing how a particular equivalence could not be proved for non-uniform bitwidths, and hypothesizing the issue may stem from an overly restrictive condition on a rewrite rule, possibly due to challenges in proving its correctness.

Peephole optimisations, which are a form of term rewriting over SSA IRs, lie at the core of modern software compilers, such as LLVM. These optimisation passes, such as the 'InstCombine' pass, have been at the centre of extensive formal verification efforts [12, 13], which aim to provide formal guarantees on the correctness of the rewrites. However, correctness is only guaranteed for specific bitwidths by enumerating all parameters of interest, and recent work in generalising these rewrites [14] highlights this challenge and calls for a tool that supports parametric bitwidth reasoning.

Solving equalities for parametric width bitvectors is currently being explored in the SMT solver domain, with some approaches encoding bitvector expressions as non-linear integer arithmetic [15] and others using dependently typed bitvectors in combination with interactive theorem provers [2, 9, 11], requiring manual proofs in some cases. These approaches are all based on the SMT-LIB theory of fixed-sized bitvectors [1] which enforces bitwidths to be concrete numbers, and by extension the tools allow for a single bitwidth parameter, limiting their applicability to the hardware domain.

We propose an approach for proving bitvector equalities over multiple parametric bitwidths based on term rewriting in e-graphs, using a translation to non-linear integer arithmetic similar to [15], and generating explainable proof certificates. In the following sections we describe the approach and present some preliminary results for a prototype implementation.

2 MODULO ARITHMETIC EQUIVALENCE

2.1 Equality Saturation for Verification

E-graphs and equality saturation naturally lend themselves to the task of verifying equality between two expressions in a language, as demonstrated in [8, 17, 18]. Given a *lhs* and a *rhs*, checking their equivalence amounts to instantiating them in the e-graph and saturating the e-graph using a set of rewrites; if the root nodes of the expressions belong to the same equivalence class then the two are equivalent. In this prototype, we use the egg [20] library to implement e-graphs and equality saturation.

2.2 Mapping Hardware Operations to Integer Arithmetic

We define a language which captures the semantics of a generic hardware description language in terms of non-linear integer arithmetic operations. A bitvector term of bitwidth p is interpreted as an integer modulo 2^p, where p is a symbolic constant assumed to

LATTE '25, March 30, 2025, Rotterdam, Netherlands

Name	Left-hand side	Right-hand side	Condition
add-1	$(a_q + b_r)_p$	$(a_q + b_r)$	$q < p \ \wedge r < p$
add-2	$(a_q + b)_p$	$(a + b)_p$	$q \geq p$
mul-1	$(a_q \times b_r)_p$	$a_q \times b_r$	$p \geq r + q$
mul-2	$(a_q \times b)_p$	$(a \times b)_p$	$q \geq p$
mul-pow2	$(a_p \times 2^{b_q})_s$	$a_p \times 2^{b_q}$	$s \geq p + 2^q - 1$
reduce	$(a_q)_p$	(a_q)	$p \geq q$
div	$(a_q \div b)_p$	$(a_q \div b)$	$p \geq q$

Table 1: Modulo simplification rewrites

be strictly greater than 0, $x_p \triangleq x \bmod 2^p$. In this work we interpret all variables as unsigned integers.

Bitvector arithmetic operators operating on q-many bits are similarly translated to integers by applying modulo 2^q to the resulting expression. For example, the 9-bit addition of 4 and 5-bit variables x and y is represented as $(x_4 + x_5)_9$, which is shorthand for $(x \bmod 2^4 + y \bmod 2^5) \bmod 2^9$.

Shift operations are expressed in terms of multiplication and euclidean division ($\div$) of exponential terms in the form 2^t. So, a left shifting operation $a \ll b$ is defined as $a \times 2^b$ and a right shift $a \gg b$ is defined as $a \div 2^b$, where $a \div b = \lfloor a/b \rfloor$. We also define a constant propagation analysis using the semantics described above.

2.3 Conditional Rewrite Rules

Equalities over parametric bitwidth variables and operators are usually only true under some condition on the bitwidths. Consider a simplified version of the previous example, associativity of addition for bitwidths p, q and r: $((a_p + b_p)_q + c_p)_r = (a_p + (b_p + c_p)_q)_r$; the equality holds if $q > p$ and $r > q$, i.e. the case where the adders capture the full precision of the operands, including overflows, but it also holds if $q \geq r$ since the r-bit addition discards any extra bits that might be introduced by the q-bit adder.

In order to discover equalities in the modular arithmetic domain we introduce a set of conditional rewrites (Table 1) which allow for removal of the modulo operator from an expression, allowing for a further set of standard arithmetic rewrites, defined over integers, to be applied, leading to equivalence. In order to apply the modulo-removing rewrites a symbolic condition must be provided, this is then used to symbolically evaluate the preconditions of Table 1, and choose whether to apply the rewrite or not.

To illustrate this, take the previous example with the $q \geq r$ condition, the prototype produces the following chain of rewrites, showing how the *lhs* and *rhs* are rewritten to a common expression; the arrow shows the direction the rewrite (in brackets) is applied to the expression on that line:

$$\begin{aligned} lhs &= ((a_p + b_p)_q + c_p)_r \downarrow \text{(add-2)} \\ &= ((a_p + b_p) + c_p)_r \\ \text{(associativity)} \uparrow &= (a_p + (b_p + c_p))_r \\ \text{(commutativity)} \uparrow &= ((b_p + c_p) + a_p)_r \\ \text{(add-2)} \uparrow &= ((b_p + c_p)_q + a_p)_r \\ \text{(commutativity)} \uparrow &= (a_p + (b_p + c_p)_q)_r = rhs \end{aligned} \tag{1}$$

We formally prove each rewrite in Table 1 using the Isabelle [16] proof assistant, providing a robust guarantee of the correctness of the overall equivalence. Having found equivalence, the individual steps that led to it can be extracted and used to generate an interpretable proof certificate in the Isabelle/HOL language, this is left as future work.

3 PRELIMINARY RESULTS

As a benchmark, we take the set of conditional rewrites defined in [6]; these are used to automate RTL datapath optimisation. Out of the 27 conditional rewrites, we consider 16, excluding those which contain operators not implemented yet: the ternary operator, negation, concatenation and more complex merging operators. The prototype is able to automatically verify all 16 conditional equivalences using the set of rewrites defined above and a set of basic arithmetic identities.

3.1 Comparison

All equivalences were manually proven in Isabelle using lemmas corresponding to the prototype's rewrites, relying on automated techniques like Sledgehammer[3] to assess the advantage of our custom e-graph solver. While Sledgehammer employs an SMT solver that also uses an e-graph approach for satisfiability, we observed four cases where proof automation fails, timing out after one minute, demonstrating the advantage of our prototype. For example, Isabelle failed to automatically prove the following equality:

$$t > 1 \wedge t > p \wedge s \geq p + q \implies ((a_p \times b_q)_s + b_q)_r = ((a_p + 1_1)_t \times b_q)_r \tag{2}$$

The other 3 cases are shift-merging operations and the $(a_p + (b_t \gg c_u)_q)_r = (((a_p \ll c_u)_s + b_t)_v \gg c_u)_r$ conditional rewrite.

Another interesting point of comparison is with [15], where authors use the Alive [13] test-suite as a benchmark for parametric bitwidth bitvector equalities and they are unable to prove any of the 'InstCombineShift' optimisations. This tool is able to prove the following equality:

$$(s \geq r \vee s > \max(p, q)) \wedge u \geq r \implies ((a_p \ll c_t)_u + (b_q \ll c_t)_u)_r = ((a_p + b_q)_s \ll c_t)_r \tag{3}$$

which is a much more general case of the 'InstCombineShift:497d' optimisation $(x \ll C) + (C2 \ll C) = (x + C2) \ll C$, where the variable x and constants $C, C2$ are of the same bitwidth.

4 CONCLUSIONS AND FUTURE WORK

We describe a method for finding equivalences between bitvector expressions with multiple parametric bitwidths and we present a prototype which is currently able to solve such equivalences given a condition, succeeding in case where other approaches currently fail. We are working on automatic formalisation of the proof, to generate a certificate, extending the language to be able to reason about signed integers and we are also exploring techniques for axiomizing bitwise operations in terms of integer arithmetic, similar to [15]. In the longer term, we aim to introduce a SystemVerilog front-end to allow for push-button verification of parametric modules, inspired by the online Alive2 rewrite verifier[1].

[1] https://alive2.llvm.org/

REFERENCES

[1] [n.d.]. SMT-LIB The Satisfiability Modulo Theories Library. https://smt-lib.org/theories-FixedSizeBitVectors.shtml

[2] Siddharth Bhat, Alex Keizer, Chris Hughes, Andrés Goens, and Tobias Grosser. 2024. Verifying Peephole Rewriting in SSA Compiler IRs. *LIPIcs, Volume 309, ITP 2024* 309 (2024), 9:1–9:20. https://doi.org/10.4230/LIPICS.ITP.2024.9 Artwork Size: 20 pages, 836196 bytes ISBN: 9783959773379 Medium: application/pdf Publisher: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[3] Sascha Böhme and Tobias Nipkow. 2010. Sledgehammer: Judgement Day. In *Automated Reasoning*, Jürgen Giesl and Reiner Hähnle (Eds.). Springer, Berlin, Heidelberg, 107–121. https://doi.org/10.1007/978-3-642-14203-1_9

[4] Chen Chen, Guangyu Hu, Dongsheng Zuo, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. 2024. E-Syn: E-Graph Rewriting with Technology-Aware Cost Functions for Logic Synthesis. https://doi.org/10.48550/arXiv.2403.14242 arXiv:2403.14242 [cs].

[5] Jianyi Cheng, Samuel Coward, Lorenzo Chelini, Rafael Barbalho, and Theo Drane. 2024. SEER: Super-Optimization Explorer for High-Level Synthesis using E-graph Rewriting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ACM, La Jolla CA USA, 1029–1044. https://doi.org/10.1145/3620665.3640392

[6] Samuel Coward, George A. Constantinides, and Theo Drane. 2022. Automatic Datapath Optimization using E-Graphs. In *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*. 43–50. https://doi.org/10.1109/ARITH54963.2022.00016 ISSN: 2576-2265.

[7] Samuel Coward, Theo Drane, and George A. Constantinides. 2024. ROVER: RTL Optimization via Verified E-Graph Rewriting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024), 1–1. https://doi.org/10.1109/TCAD.2024.3410154

[8] Samuel Coward, Emiliano Morini, Bryan Tan, Theo Drane, and George A. Constantinides. 2023. Datapath Verification via Word-Level E-Graph Rewriting. In *2023 Formal Methods in Computer-Aided Design (FMCAD)*. 92–100. https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_17 ISSN: 2708-7824.

[9] Burak Ekici, Arjun Viswanathan, Yoni Zohar, Cesare Tinelli, and Clark Barrett. 2023. Formal Verification of Bit-Vector Invertibility Conditions in Coq. In *Frontiers of Combining Systems*, Uli Sattler and Martin Suda (Eds.). Springer Nature Switzerland, Cham, 41–59. https://doi.org/10.1007/978-3-031-43369-6_3

[10] Alfred Koelbl, Reily Jacoby, Himanshu Jain, and Carl Pixley. 2009. Solver technology for system-level to RTL equivalence checking. In *Automation & Test in Europe Conference & Exhibition 2009 Design*. 196–201. https://doi.org/10.1109/DATE.2009.5090657 ISSN: 1558-1101.

[11] Hanna Lachnitt, Mathias Fleury, Leni Aniva, Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark Barrett, and Cesare Tinelli. 2024. IsaRare: Automatic Verification of SMT Rewrites in Isabelle/HOL. In *Tools and Algorithms for the Construction and Analysis of Systems*, Bernd Finkbeiner and Laura Kovács (Eds.). Vol. 14570. Springer Nature Switzerland, Cham, 311–330. https://doi.org/10.1007/978-3-031-57246-3_17 Series Title: Lecture Notes in Computer Science.

[12] Nuno P. Lopes, Juneyoung Lee, Chung-Kil Hur, Zhengyang Liu, and John Regehr. 2021. Alive2: bounded translation validation for LLVM. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. ACM, Virtual Canada, 65–79. https://doi.org/10.1145/3453483.3454030

[13] Nuno P. Lopes, David Menendez, Santosh Nagarakatte, and John Regehr. 2015. Provably correct peephole optimizations with alive. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '15)*. Association for Computing Machinery, New York, NY, USA, 22–32. https://doi.org/10.1145/2737924.2737965

[14] Manasij Mukherjee and John Regehr. 2024. Hydra: Generalizing Peephole Optimizations with Program Synthesis. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1 (April 2024), 725–753. https://doi.org/10.1145/3649837

[15] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Yoni Zohar, Clark Barrett, and Cesare Tinelli. 2021. Towards Satisfiability Modulo Parametric Bit-vectors. *Journal of Automated Reasoning* 65, 7 (Oct. 2021), 1001–1025. https://doi.org/10.1007/s10817-021-09598-9

[16] Tobias Nipkow, Markus Wenzel, Lawrence C. Paulson, Gerhard Goos, Juris Hartmanis, and Jan Van Leeuwen (Eds.). 2002. *Isabelle/HOL*. Lecture Notes in Computer Science, Vol. 2283. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45949-9

[17] Michael Stepp, Ross Tate, and Sorin Lerner. 2011. Equality-Based Translation Validator for LLVM. In *Computer Aided Verification*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, Berlin, Heidelberg, 737–742. https://doi.org/10.1007/978-3-642-22110-1_59

[18] Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2009. Equality saturation: a new approach to optimization. *SIGPLAN Not.* 44, 1 (Jan. 2009), 264–276. https://doi.org/10.1145/1594834.1480915

[19] Ecenur Ustun, Ismail San, Jiaqi Yin, Cunxi Yu, and Zhiru Zhang. 2022. IMpress: Large Integer Multiplication Expression Rewriting for FPGA HLS. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 1–10. https://doi.org/10.1109/FCCM53951.2022.9786123 ISSN: 2576-2621.

[20] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. egg: Fast and Extensible Equality Saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (Jan. 2021), 1–29. https://doi.org/10.1145/3434304 arXiv:2004.03082 [cs].