There and Back Again: A Netlist’s Tale with Much Egraphin’

Gus Henry Smith†, Zachary D. Sisco‡, Thanawat Techaumnaiwit‡, Jingtao Xia‡, Vishal Canumalla†, Andrew Cheung†, Zachary Tatlock†, Chandrakana Nandi§, Jonathan Balkind‡

† University of Washington ‡ University of California, Santa Barbara § Certora, Inc.

{gussmith, vishalc, acheung8, ztatlock}@cs.washington.edu {zsisco, thanawat, jingtaoxia, jbalkind}@ucsb.edu
chandra@certora.com

ABSTRACT

EDA toolchains are notoriously unpredictable, incomplete, and error-prone; the generally-accepted remedy has been to re-imagine EDA tasks as compilation problems. However, any compiler framework we apply must be prepared to handle the wide range of EDA tasks, including not only compilation tasks like technology mapping and optimization (the “there” in our title), but also decapsulation tasks like loop rerolling (the “back again”). In this paper, we advocate for equality saturation—a term rewriting framework—as the framework of choice when building hardware toolchains. Through a series of case studies, we show how the needs of EDA tasks line up suspiciously well with the features equality saturation provides.

1 INTRODUCTION

Hardware development toolchains are notorious for their unpredictability [12], incompleteness [17], and incorrectness [4]. These issues stem from the fact that most common toolchains do not treat EDA tasks as compilation problems, and instead often use ad hoc, unprincipled approaches to solving each problem. Existing projects such as MLIR CIRCT [1], LLHD [15], and Calyx [13] have made great strides towards reframing and restructuring hardware design tools using consistent compiler frameworks.

Finding an appropriate compiler framework is difficult, as the EDA tasks that must be supported are diverse. For example, any framework should certainly be able to capture all standard optimization tasks, such as register retiming, pipelining, and common subexpression elimination. However, another essential task beyond standard optimization is technology mapping—the process of implementing a high-level design specification using actual hardware primitives available on the target FPGA or ASIC process. To make things even more complicated, EDA tasks are not always “moving forward”: recent work has established hardware decomposition as a valuable tool for design tasks such as speedup on netlist simulations [16]. Thus, an ideal compiler framework must also be able to easily break and lower between levels of abstraction.

Equality saturation [19] is a compiler framework which has already proven its prowess in all of these tasks. Equality saturation is a non-destructive term rewriting technique that uses the e-graph data structure [10, 11] to compactly store potentially infinitely many equivalent terms. Recent work [27, 29] has developed fast and extensible libraries for efficient equality saturation. Previous work has shown equality saturation’s ability to implement decapsulation [9], procedural abstraction [2], optimization [7, 8, 20, 24, 25, 30], and mapping [6, 18, 23].

In this position paper, we advocate for the extensive application of equality saturation to EDA tasks. In fact, equality saturation has already shown early promise in being applied to various EDA tasks, including RTL optimization [5, 14], HLS optimization [3], multiplier optimization [21, 26], and repurposing CGRAs [28]. Ustun et al. also argue for equality saturation in datapath synthesis and optimization [22]. We make a larger claim in this paper that equality saturation has value beyond optimization in EDA tasks up and down the stack.

We now present four case studies highlighting different properties of equality saturation that makes it attractive at different stages in the hardware design workflow, from technology mapping to circuit-level analyses such as retiming and decomposition. These case studies demonstrate that the operational semantics of the various stages of the hardware design workflow are intuitively represented as rewrite rules. Each case study will explore the different properties of equality saturation that makes it attractive for implementing the different hardware passes.

2 CASE STUDY: HARDWARE LOOP REROLLING

Recent work considers the problem of hardware loop rerolling, that is, identifying repeated sequences of logic in a netlist and rerolling them into loops in higher-level HDL code [16]. This research fits into the larger problem of hardware decomposition, which lifts netlists to HDL code to help with design and analysis tasks. Loop rerolling for hardware decomposition uses a sketch-guided program synthesis technique to synthesize rerolled loops [16]. However, this technique scales poorly due to its reliance on SMT solvers to fill in the loop sketches.

Our in-progress work considers hardware loop rerolling through the lens of rewriting. Consider the following illustration of a rewrite rule which identifies a repeated logic block G and rewrites it into a for-loop with G parameterized over the loop variable i:

![Diagram of loop rerolling example]
While in this example, the indices $a$ and $b$ appear in monotonically increasing order on the logic blocks, this may not always be the case, which would make it harder to infer the closed form for the for-loop. More generally, loop rerolling is particularly challenging when the initial, unrolled program does not expose any high-level structure, i.e., the repetitive patterns of the program are obfuscated. Prior work shows that equality saturation can be used to discover this latent structure by applying carefully designed rewrite rules [9]. We envision scaling hardware loop rerolling by leveraging similar techniques.

3 CASE STUDY: STANDARD LIBRARY COMPONENT IDENTIFICATION

This case study is about finding components from a hardware standard library within a compiled artifact such as a netlist. The compiler optimizes the component in ways that using a sub-graph-isomorphism algorithm for identification will fail, and, for large enough designs, will not scale. An e-graph solves these problems in two ways: (1) it allows us to explore semantically equivalent versions of the same design to find the one where we can extract the standard library component and (2) it allows sub-graphs to be extracted out more efficiently due to the internal union-find structure. For standard library component identification, we can directly take the standard library component we are looking for and turn it into a rewrite rule for a half adder:

Within this rewrite rule, $i_0$ and $i_1$ are any arbitrary circuits. Equality saturation runs this rewrite rule (along with standard rules for Boolean algebra) on a larger design which pattern-matches parts of the design with the half-adder definition—rewriting that definition into an abstract half-adder component.

Challenges with this approach include matching on components where the compiler optimized away parts of the module or fused two modules together which share resources. Anti-unification techniques, as presented in babble [2], can help with the problem of partial matching. Further, a generalized problem of standard library identification is procedural abstraction, finding repeated instances of a procedure where there is no standard library as reference to match against.

4 CASE STUDY: SCALING TECHNOLOGY MAPPING VIA LIBRARY LEARNING

Our previous work Lakeado [17] demonstrates how the process of FPGA technology mapping—converting a high-level hardware design description into an implementation using FPGA-specific primitives—can be vastly improved via program synthesis. However, program synthesis is known to face scaling issues. Meanwhile, the process of technology mapping must scale to potentially massive hardware designs.

With equality saturation, we can scale these state-of-the-art technology mapping techniques via the application of library learning [2]. Library learning is the process of finding abstractions commonly used throughout a corpus of code—in our setting, finding hardware modules used repeatedly within a larger design. Within the equality saturation framework, library learning can be expressed simply as a rewrite which converts an expression into an abstracted module applied to a list of concrete inputs:

When applied repeatedly across a large design, this rewrite will find larger and larger abstracted submodules. By default, equality saturation deduplicates identical expressions, allowing us to discover submodules which are frequently reused across the design. These abstracted submodules are then perfect candidates for program synthesis. Furthermore, we can use information about frequency of appearance and other contextual information in the e-graph to filter and rank candidates. Thus, equality saturation gives us a path towards scaling currently limited state-of-the-art techniques using simple algebraic rewrites and its native deduplication ability.

5 CASE STUDY: CIRCUIT RETIMING

With an algebraic representation of the netlist we form a bidirectional rewrite rule that captures forward and backward retiming:

where Comb is a combinational gate. With only these two rules, equality saturation explores all possible ways of arranging registers in the design through non-destructive rewrites. Then, we use ILP (Integer Linear Programming) to retime the circuit according to a cost function—following prior work that effectively uses ILP extraction from an e-graph [3, 24].

The other side of retiming is undoing the effects of a retimed circuit by, for example, moving all registers as close as possible to their source. This pass is useful for decompilation by moving registers outside of a section of combinational logic to expose latent structure for other analyses such as standard library component identification and loop rerolling (Sections 2 and 3).

6 CONCLUSION

We present case studies demonstrating how equality saturation can be used to improve state of the art techniques for mitigating four concrete hardware challenges: decompilation through loop rerolling, library component identification and technology mapping through library learning, and optimum circuit retiming through efficient state space exploration and ILP extraction. We are already working on some of these topics and hope this paper encourages other researchers to consider equality saturation as a technique to mitigate EDA challenges in the future.
REFERENCES


