Augmenting Rewrite Rule Sets via Knuth-Bendix Completion

MICHAEL SCHIFFERER, Saarland University, Saarland Informatics Campus, Germany
MARCEL ULLRICH, Saarland University, Saarland Informatics Campus, Germany
SEBASTIAN HACK, Saarland University, Saarland Informatics Campus, Germany

Equality Saturation (EqSat) is a powerful technique for program optimization, systematically exploring the search space of candidate programs to overcome the phase ordering problem.

However, the feasibility and performance of EqSat rely heavily on the specific rewrite rules used to derive equivalent programs. These rule sets are typically handcrafted, requiring extensive domain expertise and carrying the risk of missing valuable transformations.

In this work, we evaluate Knuth-Bendix Completion (KBC) as a method for automatically generating and augmenting rewrite rules for EqSat. Our experiments show faster optimization as well as reaching better terms with previously missed optimizations.

1 Introduction

Optimizing compilers rely heavily on program transformations to improve code performance. However, applying these transformations sequentially introduces the phase ordering problem, where the order of applied optimizations heavily limits the final program quality. EqSat addresses this challenge by representing multiple equivalent programs simultaneously within a specialized data structure known as an e-graph. By systematically applying transformations without committing to a single path, EqSat exhaustively explores the search space of candidate programs to find the optimal representation.

While EqSat proves to be an effective optimization framework, its feasibility and performance depend on the underlying rewrite rules used to derive equivalent programs. Compiler developers handcraft these rule sets to carefully balance the trade-off between enabling a deep search space and preventing the e-graph from growing out of control. Designing these rule sets requires strong domain knowledge, intuition, and extensive testing. Because manual rule creation is prone to human error, developers often miss valuable, non-obvious transformations. Furthermore, little is known about the exact properties of highly effective rule sets, and the automated tools available to generate them remain limited.

To address this limitation, we evaluate Knuth-Bendix Completion (KBC), a well-established procedure from automated theorem proving, as a method for automatically generating and augmenting rewrite rules for program optimization. We develop a methodology to process an initial, handwritten rule set using KBC and evaluate the resulting rules in two distinct environments. First, we test the KBC-generated rules within the standard EqSat framework to see if they can uncover previously missed optimizations. Second, we evaluate them using a custom greedy rewriting engine, which trades the exhaustive exploration of EqSat for compile time and memory efficiency at the risk of result programs of inferior quality.

Our experiments show that augmenting handwritten rules with KBC-generated rules results in faster optimization and allows the system to reach better terms with previously missed optimizations. In this paper, we present

- different methods for generating and processing rewrite rule sets for program optimization using KBC,

Authors' Contact Information: Michael Schifferer, misc00021@stud.uni-saarland.de, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany; Marcel Ullrich, ullrich@cs.uni-saarland.de, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany; Sebastian Hack, hack@cs.uni-saarland.de, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany.

- an experimental evaluation demonstrating that KBC-augmented rule sets improve equality saturation in terms of overall simplification effectiveness and running time compared to strictly handwritten rules, and
- analyze the viability of greedy rewriting as a resource-efficient alternative to EqSat, finding that while it minimizes memory and time usage, it still achieves considerably less effective simplification than EqSat.

Section 2 establishes the background on term rewriting, equality saturation, and Knuth-Bendix completion. Section 3 details our methodology for rule generation and the experimental setup. Section 4 presents our results across both the EqSat and greedy rewriting frameworks. Finally, we discuss related work in Section 5 and our conclusion in Section 6.

2 Term Rewriting Systems

Given a *simplification order* $\succ$ on terms, KBC turns a set of equalities into a *confluent* and *terminating* Term Rewriting System (TRS). If successful, it ensures that equivalent terms ($t =_E^* t'$) simplify to a unique normal form.

A simplification order $\succ$ requires three properties:

(1) Compatibility: $s \succ t \implies f(\dots, s, \dots) \succ f(\dots, t, \dots)$ for all $f \in \Sigma$.
(2) Stability: $s \succ t \implies s\sigma \succ t\sigma$ for any substitution σ.
(3) Subterm ordering: If s is a proper subterm of t, then $t \succ s$.

A common implementation is the Knuth-Bendix Order (KBO), which weighs terms based on predefined symbol weights, breaking ties using operator precedence and lexicographic checks.

KBC is driven by two main concepts: First, *orientation*: an equality $s = t$ permanently becomes a directed rewrite rule $s \to t$ if $s \succ t$. Second, *critical pairs*: if a term u diverges into two different terms ($s \leftarrow_R u \rightarrow_R t$), we deduce $s = t$. This creates a new rule to bridge them. Visually, this creates a shortcut that "tunnels through peaks" in a proof, ensuring confluence.

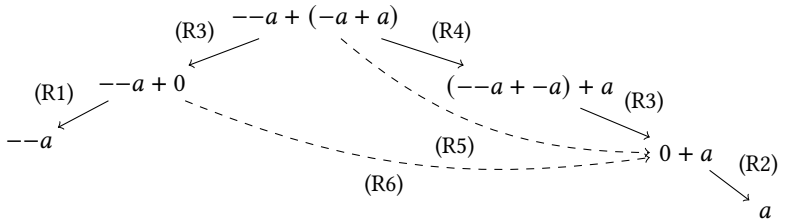


Fig. 1. A rewrite proof where critical pairs tunnel the peaks.

KBC continuously applies elementary inference rules to a set of equalities E and rewrite rules R:

$$\text{Orient}\ \frac{s \succ t}{E,\ R \cup \{s \to t\}} \qquad \text{Delete}\ \frac{E \cup \{s = s\},\ R}{E,\ R} \qquad \text{Deduce}\ \frac{s \leftarrow_R u \rightarrow_R t}{E \cup \{s = t\},\ R}$$

A standard KBC run results in success (yielding a confluent TRS), failure (due to unorientable equations like $X + Y = Y + X$), or non-termination.

3 Approach

To automate the generation of rewrite rules for program optimization, we design a pipeline consisting of two primary phases: synthesizing rule sets via KBC and deploying these rule sets across two distinct rewriting environments.

Rule Generation. We use the twee [Smallbone 2021] theorem prover to perform KBC on an initial set of handwritten arithmetic axioms. Twee derives Critical Pairs (CPs), augmenting the original rule set.

For unorderable rules, we investigate removing them vs keeping both orientations, left to right and right to left, as two rules. We create multiple rule set variants by using different completion limits (the maximum number of rules generated) and by evaluating whether to replace the original handwritten rules entirely or augment them with the newly generated rules. Finally, we isolate and process partial conditional operations (such as division) separately to ensure that the generated rules remain sound and avoid invalid states like division by zero.

The rules produced by KBC are oriented. However, keeping the original rules is often beneficial as EqSat uses expanding rules to explore a large state space of terms for possible optimization. We categorize the KBC rules into three categories:

- Normal rules: Rules that are already present in the original rule set or specializations of present rules
- Shortcut rules: Rules that result from combining multiple rule applications into a single step
- Novel rules: Rewrites that are not achieved by EqSat as they require inventing a value. Such a rule is $y \cdot (y^{-1} \cdot z) \rightarrow 1 \cdot z$ from the rules $x \cdot x^{-1} \rightarrow 1$ and $(x \cdot y) \cdot z \rightarrow x \cdot (y \cdot z)$. To reach this rewrite, a value for x in $1 = x \cdot x^{-1}$ has to be invented.

Special care in completion of rules is needed for partial functions with preconditions like division. The rules $0/x = 0$ and $x/x = 1$ allow proving $1 = 0$ for $x = 0$. Therefore, conditions need to be added that restrict instantiation with 0. We use the encoding defined by Claessen and Smallbone [2021] to encode preconditions as `if-then-else` constructs. The egg rule `rw!("cancel-div"; "(/ ?x ?x)" => "1" if is_not_zero("?x"))` is expressed as `ifeq(not_zero(x),true,x/x,1)`$\rightarrow$ `1`.

Rewriting Environments. Once we generate and process the rule sets, we deploy them into two distinct optimization frameworks to evaluate their performance.

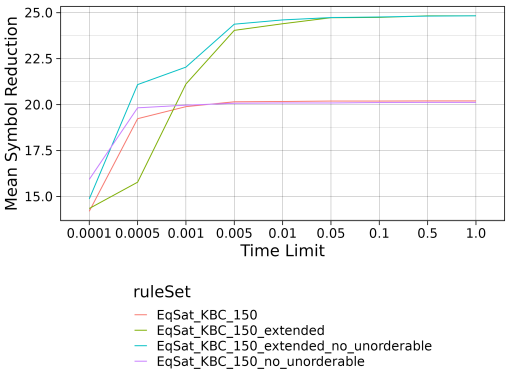
First, we integrate the rules into a standard EqSat engine. This environment leverages the e-graph data structure to exhaustively explore equivalent program representations simultaneously. By applying all possible rules at every step without committing to a single rewrite path, this framework allows us to observe the maximum simplification power of our KBC-generated rules.

Second, we apply the exact same rule sets within a custom greedy rewriting engine. Unlike EqSat, the greedy approach does not explore multiple paths. Instead, it applies simplifying rules sequentially and deterministically until no further rule matches exist. We select rules according to the generation order defined by Twee. Twee enumerates rules heuristically, starting with rules with few operators. By deploying the rules in both frameworks, we isolate the impact of the rule sets themselves and directly contrast the exhaustive, resource-intensive nature of EqSat against the lightweight, sequential execution of greedy rewriting.

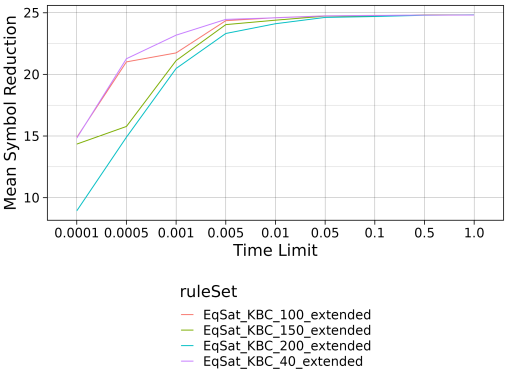
KBC produces a confluent rewrite system if it terminates. Such a rewrite system is desired as rules can be applied in an arbitrary order. We hypothesize that a partially completed rewrite system by KBC exhibits confluent properties allowing to apply it greedily.

4 Experimental Evaluation

We evaluate our KBC-generated rule sets based on mean symbol reduction, the average decrease in the total number of operators between the input and the optimized terms. We conduct our experiments across both the exhaustive EqSat engine and our custom greedy rewriting engine, testing randomly generated arithmetic terms of varying sizes (small=1000 terms with 3 binary operators, large=500 terms with 10 binary operators, and huge=250 terms with 25 binary operators). We use the arithmetic rewrite rules from Willsey et al. [2021a].



(a) Impact of rule set postprocessing on mean symbol reduction for huge terms.



(b) Simplification effectiveness across different time limits relative to the number of generated rules used to extend the base set.

Equality Saturation Performance. First, we evaluate how KBC-generated rules affect the simplification power of EqSat. We observe that strictly replacing the handwritten rules with KBC-generated rules often causes the system to get stuck in local optima, because KBC naturally filters out the complexity-increasing rules that EqSat relies on to bridge equivalent states. For instance, $(x^3 - 1)/(x - 1)$ first needs to be expanded to $(x^3 - x^2 + x^2 - x + x - 1)/(x - 1)$ before it can be simplified to $x^2 + x + 1$. The original rules contain rules in both directions to expand terms and reduce them. KBC only keeps the reducing rules in its rule set.

On the other hand, extending the original handwritten rules with KBC-generated rules yields consistently better results as seen in Figure 2a.

For example, on large terms containing partial functions, the KBC-augmented rule set (`E-KBC150`) achieves a mean symbol reduction of 10.46, outperforming the strictly handwritten baseline (`E-base`) at 9.73, see Table 1. On terms without partial functions, `E-KBC150` reaches a reduction of 14.13 compared to the baseline's 13.61. We find that generating up to 150 additional rules maximizes effectiveness; beyond this threshold, larger rule sets slow down the EqSat convergence without providing meaningful improvements in term reduction as shown in Figure 2b.

Greedy Rewriting Performance. We evaluate the rule sets within a greedy rewriting environment, where the engine applies rules sequentially without tracking multiple term variations. In this setting, the KBC-generated rule sets vastly outperform the original handwritten rules. The handwritten rules rely heavily on EqSat's ability to increase term size for exploration. Therefore, the handwritten rules fail to simplify terms in a greedy, sequential setup. In contrast, the KBC rules provide directly simplifying paths. Unlike in EqSat, greedy rewriting benefits significantly from much larger rule sets; we observe that simplification effectiveness continuously improves as we increase the rule set size up to 1,000 rules (`G-KBC1000`), after which the benefits begin to plateau.

Cross-Framework Comparison. Finally, we compare the best-performing KBC configurations of both frameworks to highlight the trade-off between optimization power and resource efficiency. EqSat consistently achieves higher overall simplification effectiveness, outperforming the greedy approach in mean symbol reduction.

However, greedy rewriting requires several orders of magnitude less time to reach its optimal solution. For instance, as shown in Table 1, on large terms with partial functions, `G-KBC1000` executes in just 0.000081 seconds, whereas `E-KBC150` requires 0.05 seconds. Ultimately, our experiments confirm that KBC successfully enhances EqSat for maximum optimization power, while

Table 1. Comparison of optimal rule sets across EqSat and greedy rewriting frameworks on small terms.

Term Set	Configuration	Mean Reduction	Time (s)
without div/pow, large	G-KBC1000	10.88	0.00009
	E-KBC100	**14.13**	1.0000
	E-base	13.61	0.5
with div/pow, large	G-KBC1000	8.92	0.000081
	E-KBC150	**10.46**	0.05
	E-base	9.73	0.005

simultaneously transforming greedy rewriting into a efficient alternative with significantly less optimization time and memory consumption.

5 Related Work

To accurately position our contributions, we discuss notable work in equality saturation, automated rewrite rule generation, and specialized e-graph variations.

Equality Saturation. EqSat was initially introduced to solve the phase ordering problem by systematically exploring equivalent programs [Tate et al. 2011]. The approach gained significant traction with the introduction of the `egg` framework [Willsey et al. 2021b], which drastically improves extensibility via domain-specific analyses and performance through e-graph rebuilding. Today, researchers apply EqSat to diverse domains, including floating-point error reduction in `Herbie` [Panchekha et al. 2015] and tensor program optimization in `Felix` [Zhao et al. 2024]. However, these practical applications rely almost exclusively on manually constructed rewrite rules, highlighting the need for the automated synthesis techniques explored in this work.

Rewrite Rule Synthesis. Recent advancements aim to ease the burden of manual rule creation and improve EqSat workflows (e.g., fast relational e-matching [Zhang et al. 2022] and `SmoothE` for complex term extraction [Cai et al. 2025]). For rule generation specifically, `Ruler` infers rewrite rules by enumerating terms within an e-graph and merging equivalence classes based on empirical behavior across input vectors [Nandi et al. 2021]. While `Ruler` successfully synthesizes minimal rule sets, its rules only guarantee empirical equivalence within EqSat and lack the formal confluence properties provided by completion algorithms. A recent survey on rewrite rule synthesis acknowledges KBC [Dick 1991] as a foundational technique but notes the lack of evaluation regarding its practical effectiveness in program optimization [Hong 2024]. Our work directly addresses this gap.

Acyclic E-Graphs. Recent variations of the e-graph data structure, such as ægraphs used in the Cranelift compiler, maintain strict acyclicity to enable fast, bidirectional conversions with control flow graphs [Fallin 2023]. Because ægraphs disallow cyclic derivations, they enforce a strict directionality requirement for most rewrite rules. This constraint aligns perfectly with KBC's focus on strictly directed rewrites, suggesting that KBC-generated rule sets may be uniquely well-suited for optimization passes within acyclic e-graph frameworks.

6 Conclusions

In this work, we evaluate KBC as a systematic method for automatically augmenting rewrite rules for program optimization. Our experimental results demonstrate that augmenting handwritten rule sets with KBC-generated rules significantly improves the simplification speed and effectiveness of EqSat.

The critical pairs generated by KBC act as a shortcut of rewrite chains, as well as providing novel rewrites previously not expressed by the directed nature of EqSat rule applications. However, adding too many critical pair rules can significantly increase the egraph size, thereby decreasing performance.

Furthermore, our evaluation reveals a clear trade-off with greedy rewriting: while it executes several orders of magnitude faster than EqSat, it achieves significantly lower simplification effectiveness under the tested conditions. Ultimately, KBC proves to be a valuable tool for enhancing EqSat and reducing the manual burden of rule engineering.

6.1 Future Work

This study establishes a foundation for automated rewrite rule synthesis using completion techniques, opening several avenues for future research.

- Evaluate KBC on other domains such as bitvectors
- Use KBC during EGraph saturation to find useful critical pairs
- Heuristics on helpful critical pairs generated by KBC
- Automatic rule generation with properties that make KBC superfluous

References

Yaohui Cai, Kaixin Yang, Chenhui Deng, Cunxi Yu, and Zhiru Zhang. 2025. SmoothE: Differentiable E-Graph Extraction. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 1020–1034. doi:10.1145/3669940.3707262

Koen Claessen and Nicholas Smallbone. 2021. Efficient encodings of first-order Horn formulas in equational logic. In *Proceedings of the Twenty-Fourth International Conference on Automated Reasoning (CADE-24)*. https://smallbone.se/papers/horn.pdf Accessed: 2025-10-21.

A. J. J. Dick. 1991. An introduction to Knuth-Bendix completion. *Comput. J.* 34, 1 (Feb. 1991), 2–15. doi:10.1093/comjnl/34.1.2

Chris Fallin. 2023. ægraphs: Acyclic E-graphs for Efficient Optimization in a Production Compiler. https://cfallin.org/pubs/egraphs2023_aegraphs_slides.pdf. Presentation slides, November 2023.

Charles Hong. 2024. A Survey of Rewrite Rule Synthesis. https://inst.eecs.berkeley.edu/~cs294-260/sp24/projects/charleshong/. Course project report, University of California, Berkeley.

Chandrakana Nandi, Max Willsey, Amy Zhu, Yisu Remy Wang, Brett Saiki, Adam Anderson, Adriana Schulz, Dan Grossman, and Zachary Tatlock. 2021. Rewrite rule inference using equality saturation. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 119 (Oct. 2021), 28 pages. doi:10.1145/3485496

Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. 2015. Automatically improving accuracy for floating point expressions. *SIGPLAN Not.* 50, 6 (June 2015), 1–11. doi:10.1145/2813885.2737959

Nicholas Smallbone. 2021. Twee: An Equational Theorem Prover. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 602–613. doi:10.1007/978-3-030-79876-5_35

Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2011. Equality Saturation: A New Approach to Optimization. *Logical Methods in Computer Science* Volume 7, Issue 1 (March 2011). doi:10.2168/lmcs-7(1:10)2011

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021a. egg: e-graphs good! — Math Rewrite Rules. https://github.com/egraphs-good/egg/blob/main/tests/math.rs. Accessed: 2025-10-22.

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021b. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (Jan. 2021), 1–29. doi:10.1145/3434304

Yihong Zhang, Yisu Remy Wang, Max Willsey, and Zachary Tatlock. 2022. Relational e-matching. *Proc. ACM Program. Lang.* 6, POPL, Article 35 (Jan. 2022), 22 pages. doi:10.1145/3498696

Yifan Zhao, Hashim Sharif, Vikram Adve, and Sasa Misailovic. 2024. Felix: Optimizing Tensor Programs with Gradient Descent. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 367–381. doi:10.1145/3620666.3651348

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009