

A Translation-Validated Query Optimizer: Combining Equality Saturation and SMT-Assisted Proof in Lean

Maggie Simmons

October 12, 2026

1 Overview

Query optimizers transform a query into a cheaper, equivalent plan via algebraic rewrites such as predicate pushdown, join reordering, and projection elimination. There are several past works that formally verify the equality preservation of query optimizers through different methods. One method is to encode queries into logic or a semiring semantics and decide equivalence with an SMT solver or an interactive prover [15, 5]. Another method is to certify an entire query-compilation pipeline inside a proof assistant [1, 4]. However, these approaches are often either restrictive with their allowed transformations or inefficient due to large search spaces and since applying successive optimizations can undo the benefits of previous transformations. The compiler community has recently addressed similar issues in compilation phase-ordering by rewriting non-destructively inside an e-graph via equality saturation. As a result, committing to one rewrite early never forecloses a later, interacting rewrite [12, 14]. This project builds a translation-validation based system demonstrating how past methods can be supplemented with compiler optimization techniques to build an efficient query optimizer consisting of a fixed, sound set of relational-algebra rewrite lemmas in Lean (some side conditions will be discharged automatically by an SMT solver), an external and untrusted equality-saturation search over those rules, and a lightweight Lean certificate checker that replays the search’s output against the lemmas to produce a kernel-checked equivalence theorem per optimized query.

2 Functionality and Properties

I intend to formalize the following functionality and prove the associated properties:

- **Relational Algebra and Semantics:** A toy relational-algebra IR (selection, projection, join, union) and a denotational bag semantics `eval : RA -> Database -> Multiset Tuple` in Lean 4.
- **Kernel Rewrite Rule Set:** Lean theorems proving soundness (with respect to `eval`) of a fixed set of 6-10 standard algebraic identities: selection/projection pushdown, selection splitting and merging, join commutativity and associativity, and selection-join interchange. Where a rule’s validity depends on a side condition (e.g., a predicate-implication check), that side condition will be discharged automatically via the `smt` tactic (backed in Lean by `cvc5`, with native proof reconstruction) rather than proved by hand.

- **External Equality-Saturation Search:** An untrusted optimizer in Rust, built on the `egg` e-graph library or an equivalent custom saturation loop, that searches for an optimized plan using exactly the proved rule set and emits a rewrite certificate (the sequence of rule applications it used).
- **Certificate Checker:** A Lean program that consumes a rewrite certificate together with the original and optimized query ASTs and re-derives `eval optimized = eval original` using only the proved kernel lemmas (plus congruence lemmas for rewriting under a subterm context), producing a kernel-checked theorem per query.
- **Phase-Ordering Case Study:** A pair of test queries, analogous to known two-pass conflicts in general-purpose compilers, on which a greedy fixed-order optimizer provably yields different (and suboptimal) plans depending on rule order, contrasted with the order-independent result of the saturation-based pipeline.
- **Update Optimization (Time Permitting):** Extension of the IR with one effectful update construct (state-threading semantics `Database -> Database`) and a proof of soundness for one or two update-specific rewrites, such as predicate pushdown into an `UPDATE ... WHERE` clause, or commutativity of two updates under an explicit side condition that their read/write sets are disjoint.

3 Project Description

Decision-procedure systems such as Cosette and EQUITAS [5, 15] automate equivalence checking for a decidable fragment but do not certify an optimizer. Query compiler systems such as Q*cert [1, 4] certify the entire pipeline but are infeasible for a term project. Translation validation, however, is quite doable: only a small, fixed set of primitive lemmas must be trusted and proved, while the search for a good plan is delegated to a fast, untrusted, external tool and easily checked afterwards by replaying its certificate against those lemmas.

While conducting my literature review for this project, I learned that Aurora [3] could be employed for the search side of this project. It is an `egg`-based e-graph optimizer for relational algebra, with learned rule selection and ILP extraction; however, correctness is only argued informally from e-graph structure. WeTune [13] and the more recent RuleScript [9] instead verify rewrite rules against symbolic query templates using an SMT-based equivalence prover (Z3 and QED, respectively); RuleScript explicitly argues against per-query checking as unneeded overhead once a rule is proved, and leaves the adapter that applies a proved rule to a concrete plan as trusted, unverified code. HoTTSQL [6] proves individual rewrite rules in Coq over a univalent semantics, with a self-checking search for the decidable conjunctive-query fragment, but is not connected to an external e-graph search and still trusts its SQL-to-internal-representation translation. The system closest in spirit to per-run validation is QO-Verify [8], which checks one rewritten query against the original by testing reachability within a production optimizer’s own search space rather than proving a general rule; however, it is one-sided (it never reports inequivalence), bounded by the host optimizer’s existing rules, and produces no replayable certificate. No existing system combines an untrusted e-graph search over queries, a small proved kernel of rewrite lemmas, and a kernel-checked, per-query certificate.

Note that the standard workaround to phase ordering in the compiler literature is non-destructive e-graph rewriting, which keeps all rewritten forms of a term alive so rule order stops mattering, deferring the choice of a best program to a final extraction step [12, 14]. SMT/ILP solvers enter mainly at that extraction step (optimal extraction under shared subterms is NP-hard in general) and in discharging rewrite side conditions, which is how we will employ SMT in this system. In Lean, the `smt` tactic preprocesses goals through `lean-auto` and sends them to `cvc5`, replaying

cvc5’s proof step by step so the kernel still checks everything [7, 10, 2]. An existing Lean e-graph tactic, `lean-egg` [11], is now deprecated in favor of Lean’s built-in `grind`, which is why the saturation search will be external to Lean, with only its output verified, rather than depending on the unmaintained Lean e-graph tooling.

If there is time remaining in this project, I want to address updates. Merging two syntactically distinct update sequences can change which writes are visible to which reads. Recent work extending equality saturation to effectful, imperative programs addresses this with explicit program-point context and a “linearity” invariant rewrites must preserve. This project could prove one or two concrete update rewrites sound under an explicit disjointness side condition, using the exercise to explain precisely why the general case is hard.

4 Milestones

4.1 Weeks 1–2: Setup, Semantics, and the Kernel Rule Set

Fix the relational-algebra IR (selection, projection, join, union) and the semantic domain (finite multisets of tuples over a fixed schema); set up the Lean 4 project with `lean-smt/lean-auto`; prototype the saturation search cheaply (e.g. Python or a naive Lean script) before committing to an `egg`/Rust FFI integration. Implement `eval : RA -> Database -> Multiset Tuple`, state the fixed kernel rule set (selection pushdown, splitting/merging, projection pushdown, join commutativity/associativity, selection-join interchange) as Lean theorems, and begin proving.

4.2 Week 3: Finish Kernel Proofs

Complete the remaining kernel proofs, using the `smt` tactic to discharge arithmetic and multiset-counting side goals. Every kernel rule must and checked; if not, reduce the rule set.

4.3 Week 4: External Search and the Phase-Ordering Demonstration

Get the external saturation engine producing rewrite certificates (rule, position, substitution) over the proved rule set, with a simple cost model for extraction. Construct the two-query phase-ordering example: a pair of queries where a greedy, fixed-order baseline optimizer gives a different, worse plan depending on which rule runs first, in contrast to the order-independent saturation result.

4.4 Week 5: Certificate Checker

Build the Lean verifier program, including the congruence lemmas needed to rewrite under an arbitrary subterm context. Produce a kernel-checked `eval optimized = eval original` theorem for at least 2-3 example queries; if running out of time, use simpler examples.

4.5 Week 6: Breadth and Buffer

If on schedule, extend to one more IR operator and 5-8 total example queries; otherwise, catch up on past objectives. Begin drafting the core-system sections of the final report.

4.6 Week 7: Optional – Update Optimization

Add a single effectful update construct with state-threading semantics. Prove soundness of predicate pushdown into an `UPDATE ... WHERE` clause and/or commutativity of two updates under an explicit disjoint-read/write-set side condition, reusing the Week 5 congruence-lemma infrastructure.

4.7 Week 8: Write-Up and Demonstration

Finalize the report and assemble the demonstration: the greedy optimizer failing at phase ordering, the saturation-based pipeline succeeding, and the kernel-checked Lean certificates for both the query case and (if completed) the update case.

References

- [1] Joshua S. Auerbach, Martin Hirzel, Louis Mandel, Avraham Shinnar, and Jérôme Siméon. Q*cert: A platform for implementing and verifying query compilers. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*, 2017. Demonstration track.
- [2] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.
- [3] George-Octavian Bărbulescu, Taiyi Wang, Zak Singh, and Eiko Yoneki. Learned graph rewriting with equality saturation: A new paradigm in relational query rewrite and beyond, 2024.
- [4] Véronique Benzaken, Évelyne Contejean, Mohammed Houssein Hachmaoui, Chantal Keller, Louis Mandel, Avraham Shinnar, and Jérôme Siméon. Translating canonical SQL to imperative code in Coq, 2022.
- [5] Shumo Chu, Chenglong Wang, Konstantin Weitz, and Alvin Cheung. Cosette: An automated prover for SQL. In *8th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2017.
- [6] Shumo Chu, Konstantin Weitz, Alvin Cheung, and Dan Suciu. HoTTSQL: Proving query rewrites with univalent SQL semantics. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2017.
- [7] Abdalrhman Mohamed, Tomaz Masekarenhas, Harun Khan, Haniel Barbosa, Andrew Reynolds, Yicheng Qian, Cesare Tinelli, and Clark Barrett. Lean-SMT: An SMT tactic for discharging proof goals in Lean, 2025.
- [8] Vivek Narasayya and Surajit Chaudhuri. Leveraging query optimizers to verify the soundness of LLM-based query rewrites for real-world workloads, and more! In *16th Conference on Innovative Data Systems Research (CIDR)*, 2026.
- [9] Sicheng Pan, Shuxian Wang, Wesley Zheng, Zirong Zeng, Vijay Sharma, and Alvin Cheung. An extensible and verifiable language for query rewrite rules. *Proceedings of the VLDB Endowment*, 18(1), 2025.
- [10] Yicheng Qian, Joshua Clune, Clark Barrett, and Jeremy Avigad. Lean-auto: An interface between Lean 4 and automated theorem provers, 2025.
- [11] Marcus Rossel and Andrés Goens. Bridging syntax and semantics of Lean expressions in E-graphs, 2024.
- [12] Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. Equality saturation: A new approach to optimization. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2009.

- [13] Zhaoguo Wang, Zhou Zhou, Yicun Yang, Haoran Ding, Gansen Hu, Ding Ding, Chuzhe Tang, Haibo Chen, and Jinyang Li. WeTune: Automatic discovery and verification of query rewrite rules. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2022.
- [14] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages*, 5(POPL), 2021.
- [15] Qi Zhou, Joy Arulraj, Shamkant Navathe, William Harris, and Dong Xu. Automated verification of query equivalence using satisfiability modulo theories. *Proceedings of the VLDB Endowment*, 12(11):1276–1288, 2019.