CONSTRUCTOR UNIVERSITY BREMEN

Bachelor of Science
Computer Science

Andrei Kozyrev

Equality saturation for solving equalities of relational expressions

Bachelor's Thesis

Scientific supervisor:
professor Anton Podkopaev

Reviewer:

Bremen
2023

Abstract

Modern CPUs are being developed exceptionally fast, and the number of cores is increasing rapidly. This has led to the development of multithreading, which is a technique that allows for the execution of multiple threads on a single CPU. Memory models are a fundamental aspect of multithreading and describe how memory is ordered at runtime in relation to source code. Currently, existing memory models are unsatisfactory and there is a need for new models that can be rigorously proven. In order to achieve this, formal verification using the Coq proof assistant is utilized, which enables automated proof checking and ensures the accuracy of results. Specialists in weak memory are continuosly improving the results in this domain.

One of the big and common problems in weak memory is the proof of equivalence of several memory models. Memory models are represented as expressions over relational language.

This thesis focuses on the automation of proving equalities over relational expressions in Coq. We are utilizing the techniques of equality saturation and E-graph data structure to generate proof of equaivalence for a given pair of terms. By automating these proofs, we can greatly increase the efficiency and accuracy of the proof process in weak memory.

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

1 Introduction

1.1 Motivation

Weak memory specializes on research that aims to improve the results in memory modeling. Weak memory focuses on determining the relative strength of different models. This is why one of the common challenges is to show equaivalences between several models. Given two memory models `A` and `B` we might want to check if model `A` is stronger then model `B`. If that condition holds, then the consistency of the model `A` would imply the consistency of the model `B`.

Memory models and propositions about them are usually represented as expressions over relational language. However, proofs of such the propositions are typically massive and very error prone. There were several cases of incorrect result in submited and published papers. Batty et al. [3] suggested an incorrect fix for the semantics of SC calls in C++, which was later documented by Lahav et al. [5]. Moreover Pichon-Pharabod et al. in 2016 suggested an incorrect proof of compilation in their paper [7].

Weak memory proofs are usually written in Coq [4], which is a proof assistant system. Coq helps to grant the correctness of the proof process. Coq provides a language called Calculus of Inductive Constructions (CIC), which is used to write the proofs. In CIC, proofs are expressed as formal mathematical objects, and the correctness of the proof is ensured by checking that the proof is consistent with the axioms and rules of logic. Coq is the standard of development in the field of weak memory, helping to unify and verify already complex proofs.

Nevertheless, weak memory proofs, even with a use of Coq, tend to be huge and convoluted. This is why in this thesis, we are focusing on automating a specific part of weak memory proofs, namely the proofs of equaivalences between several memory models. As already mentioned, memory models are defined as relations. Let's denote `r` and `r'` the two memory models or relations over a given set `A`. Now we may consider common op-

erations in relational language, e.g. transitive closure (`r*`), reflexive closure (`r?`), or composition (`r ;; r'`). An example of a proposition we want to prove may look like this:

$$\texttt{(r* ;; r?) ;; r?} \equiv \texttt{r*}$$

If we were to prove this statement in Coq by hand, we would have to consequently apply multiple theorems to rewrite both sides of the equivalence relation until they are syntactically equal. In this particular example, we would have to apply thorem `rt_cr` twice:

```
rt_cr : forall (A : Type) (r : relation A), r* ;; r? = r*
```

After `rt_cr` being applied to the left hand side (lhs) two times, the two sides become syntactically equal. `rt_cr` and other theorems that help to reason about relations are provided by the Hahn library [1].

The general problem discussed in this thesis is as follows: given a rewriting system, i.e. a set of theorems, and an equaivalence relation between two expressions, we want to find a sequence of rewrites that can be applied to both sides of the relation to make them syntactically equal. This is a wider problem than reasoning about relational expressions, thereafter the solution we propose could be adapted and used to solve other problems that involve derivability in two-sided associative calculus over given language in Coq. This thesis, in turn, focuses on applying the proposed technique to automate weak memory, where the rewriting system we use is the Hahn library.

1.2 Approach

Our approach to solve equivalences is based on the technique called *equality saturation* which utilizes the data structure, called an e-graph. E-graph stores an equivalence relation over terms of some language and allows to store potentially exponential amount of terms in a compact way. E-graph is a set of equaivalence classes (*e-classes*) and e-class is a set of

e-nodes, which represent equivalent terms in the language. Whilst e-nodes are function symbols, associated with a list of e-classes.

Equality saturation is a process of iteratively applying a set of rewrite rules to the e-graph until rewrites bring no more new information to the graph. That point is called saturation. The saturated e-graph represents a set of all possible terms equivalent to the origin, that can be obtained by applying the rewrite rules to the initial term.

In a saturated e-graph it becomes algorithmically easy to check if two terms are equaivalent and, moreover, to find a proof of their equivalence. Having a sequence of rewrites, we can apply them within Coq proof view. Just as we would do by hand, applying the theorems one by one.

1.3 Core implementation components

We have chosen fast and lightweight `egg` [8] rust library as an equality saturation engine for our project. `Egg` is a new and ambitious project, that has already proven its worth and was used in tools like *Ruler* [6].

In Coq you can add new functionality by writing a plugin. Plugins are written in Ocaml language. We have set the communication between Ocaml and Rust using the `ocaml-rs` [2] foreign function interface (FFI). To prove theorems in Coq, you use proof mode, which is activated when you start a proof, for example, by using the `Theorem` command. The proof state contains one or more unproven goals. Each goal consists of a conclusion, which is the statement to be proven, and a local context. When the proof is complete, you exit proof mode by using the `Qed` command. During proof mode, you use so called *tactics*, to gradually transform the proof. Tactics specify how to transform the proof state of an incomplete proof and are used to eventually generate a complete proof.

Commands in Coq are similar to tactics, but accessible from outside of proof mode, e.g. command `Print`, which prints the given term to console. A Coq plugin typically consists of a set of commands and tactics, that are

used from Coq as a front-end to the plugin. We have defined multiple tools inside our plugin, that are used to interact with the `egg` library:

- `Cegg solve` — a tactic that simplifies the lhs of the equation. It takes the conclusion of the goal, retrieves the lhs of the equivalence, parses it into an s-expression[1] and passes it to `egg`. `Egg` builds an e-graph E for it and extracts the "best" term t from E. A sequence of rewrites to achieve t is passed back to Ocaml and is applied to the lhs of the equivalence inside Coq proof mode.
- `Cegg solve eq` — a tactic that tries to prove the equivalence between the lhs and rhs of the equation, using `egg`.
- `Cegg config` — a command, that allows to configure the ruleset for `egg`. It takes a user-defined list of rewrite rules and caches it for the later use in `Cegg solve` and `Cegg solve eq`.

The source code and documentation for the thesis is available at:

https://github.com/K-dizzled/relations-via-egg

[1] S-expressions represent tree-like data

References

[1] Viktor Vafeiadis et al. "Hahn: a Coq library that contains a useful collection of lemmas and tactics about lists and binary relations". In: (2018). URL: https://github.com/vafeiadis/hahn.

[2] Zach Shipko et al. "ocaml-rs: OCaml extensions in Rust". In: (2021). URL: https://crates.io/crates/ocaml.

[3] Mark Batty, Alastair F. Donaldson, and John Wickerson. "Overhauling SC atomics in C11 and OpenCL". In: (2016). URL: https://doi.org/10.1145%2F2837614.2837637.

[4] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[5] Ori Lahav et al. "Repairing Sequential Consistency in C/C++11". In: *SIGPLAN Not.* 52.6 (2017), pp. 618–632. ISSN: 0362-1340. URL: https://doi.org/10.1145/3140587.3062352.

[6] Chandrakana Nandi et al. "Rewrite Rule Inference Using Equality Saturation". In: *Proc. ACM Program. Lang.* 5.OOPSLA (2021). DOI: 10.1145/3485496. URL: https://doi.org/10.1145/3485496.

[7] Jean Pichon-Pharabod and Peter Sewell. "A Concurrency Semantics for Relaxed Atomics That Permits Optimisation and Avoids Thin-Air Executions". In: *SIGPLAN Not.* 51.1 (2016), pp. 622–633. ISSN: 0362-1340. DOI: 10.1145/2914770.2837616. URL: https://doi.org/10.1145/2914770.2837616.

[8] Max Willsey et al. "Egg: Fast and Extensible Equality Saturation". In: *Proc. ACM Program. Lang.* 5.POPL (2021). DOI: 10.1145/3434304. URL: https://doi.org/10.1145/3434304.