

Towards Producing Shorter Congruence Closure Proofs in a State-of-the-art SMT Solver (Extended Abstract)

Bruno Andreotti, Haniel Barbosa, Oliver Flatt

PAAR 2024 at Nancy, France, 2024-07-02

- The property of congruence states that, for any terms x , y , and any function f ,

$$x = y \rightarrow f(x) = f(y)$$

- For a given set of equalities, the congruence closure is a minimal equivalence relation that satisfies them, as well as reflexivity, symmetry, transitivity and *congruence*

Congruence closure in SMT solvers

- Congruence closure algorithms are essential for solving the theory of equality and uninterpreted functions (EUF)
- The solver will provide a set of equalities and inequalities, and the congruence closure algorithm must determine if it is consistent

Congruence closure in SMT solvers

- However, for SMT solvers, it is not sufficient to just determine whether two terms are equivalent
- We also require an *explanation*, that is, a minimal set of equations that makes the terms equivalent

Congruence closure in SMT solvers

- However, for SMT solvers, it is not sufficient to just determine whether two terms are equivalent
- We also require an *explanation*, that is, a minimal set of equations that makes the terms equivalent
- Furthermore, we might also want a structured *proof*, that uses these equalities as assumptions to derive the equivalence of the two terms

Proof-producing congruence closure

- A proof-producing congruence closure algorithm was presented by Nieuwenhuis et al. in 2005 [1]
- It is based on a union-find data structure, and constructs an equality graph to represent the equivalence relation

Proof-producing congruence closure

- A proof-producing congruence closure algorithm was presented by Nieuwenhuis et al. in 2005 [1]
- It is based on a union-find data structure, and constructs an equality graph to represent the equivalence relation
- Then, finding the explanation for the equivalence of two terms consists in finding a path between them in the graph

Proof-producing congruence closure: Example

- TODO

Keeping redundant equalities

- You may have noticed that we discard equalities between terms we already know to be equivalent
- However, keeping these equalities might allow us to find shorter paths between terms, and thus shorter proofs

- Flatt et al. at FMCAD'22 [2] presented two proof producing congruence closure algorithms that make use of redundant equalities to find smaller proofs (called TREEOPT and GREEDY)

Keeping redundant equalities

- Flatt et al. at FMCAD'22 [2] presented two proof producing congruence closure algorithms that make use of redundant equalities to find smaller proofs (called `TREEOPT` and `GREEDY`)
- In the original work, they were implemented in an equality saturation tool

Keeping redundant equalities

- Flatt et al. at FMCAD'22 [2] presented two proof producing congruence closure algorithms that make use of redundant equalities to find smaller proofs (called `TREEOPT` and `GREEDY`)
- In the original work, they were implemented in an equality saturation tool
- Here, we present our effort to implement these algorithms in `cvc5` [3], a state-of-the-art SMT solver

Keeping redundant equalities: Example

- TODO

A side note: tree-size vs DAG-size

- Finding the minimal proof for the equivalence of two terms is an NP-hard problem [4]
- However, this problem becomes easier if we don't allow the reuse of proof steps

A side note: tree-size vs DAG-size

- Finding the minimal proof for the equivalence of two terms is an NP-hard problem [4]
- However, this problem becomes easier if we don't allow the reuse of proof steps
- We call this the *tree size* of the proof, in contrast to the *DAG size*

TREEOPT algorithm

- Optimal algorithm (with regards to proof tree size)
- Computes the weight of each congruence edge by finding the size of the explanation of its justification, until a fixed point
- When asked to explain the equivalence between two terms, simply finds the shortest path between them considering these weights

TREEOPT algorithm

```
function compute_weights():  
    let weights = {}  
    for edge in edges:  
        if not edge.is_congruence_edge():  
            weights[l, r] = 1  
  
    until fixed point:  
        for (l, r) in congruence_edges:  
            weights[l, r] = find_shortest_path(l, r, weights).size()  
    return weights
```

TREEOPT algorithm

```
function get_explanation(start, end, weights=compute_weights()):  
    let explanation = []  
    let path = find_shortest_path(start, end, weights)  
    for edge in path:  
        if edge.is_congruence_edge():  
            let (j1, j2) = edge.justification()  
            explanation += get_explanation(j1, j2, weights)  
        else:  
            explanation += edge  
    return explanation
```

- First, estimates the proof size of each congruence edge
- To explain the equivalence between two terms, finds the shortest path between them using these estimates as edge weights, but recurses when it encounters a congruence edge

GREEDY algorithm

```
function compute_weights():  
    let weights = {}  
    for edge in edges:  
        if edge.is_congruence_edge():  
            weights[l, r] = 1  
        else:  
            weights[l, r] = unoptimized_explanation(l, r).size()  
  
    return weights
```

GREEDY algorithm

```
function get_explanation(start, end, weights=compute_weights(), fuel):  
    if fuel == 0:  
        return unoptimized_explanation(start, end)  
  
    let explanation = []  
    let path = find_shortest_path(start, end, weights)  
    for edge in path:  
        if edge.is_congruence_edge():  
            let (j1, j2) = edge.justification()  
            explanation += get_explanation(j1, j2, fuel - 1)  
        else:  
            explanation += edge  
  
    return explanation
```

Dealing with backtracking

- Modern SMT solvers work by trying many possible (partial) solutions, and backtracking when a solution is determined to be invalid
- So, a congruence closure algorithm must be able to efficiently revert to a previous state when backtracking

Dealing with backtracking

- In our case, this is done by carefully recording the steps the congruence closure engine did, and undoing them when backtracking
- This includes clearing any cache that became invalid because of the backtrack (e.g., the TREEOPT and GREEDY edge weights)

Avoiding circular explanations

- When we were discarding redundant equalities, there could never be a redundant explanation
- This is because, after a congruence edge is added, the path between the terms of its justification will not change anymore

Avoiding circular explanations

- When we were discarding redundant equalities, there could never be a redundant explanation
- This is because, after a congruence edge is added, the path between the terms of its justification will not change anymore
- Now that we keep redundant edges, this is no longer true

Avoiding circular explanations: Example

- TODO

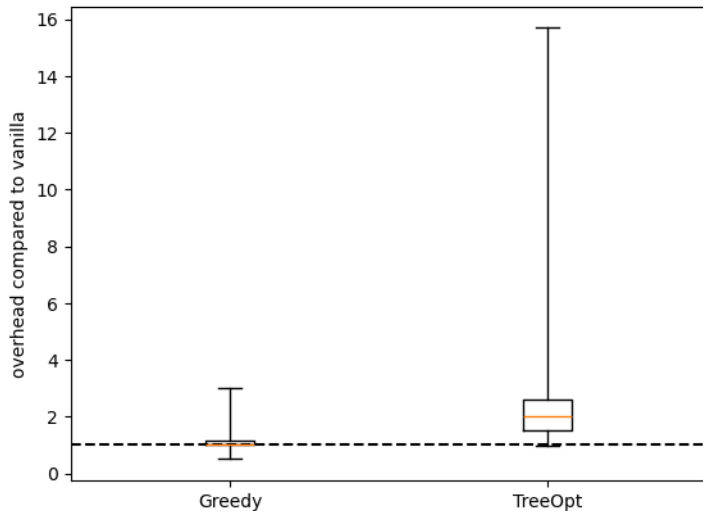
Avoiding circular explanations

- To prevent this problem, we store in each edge the *level* in which it was added
- Then, when explaining a congruence edge that was added in level n , we can only use edges whose level is no greater than n

- To evaluate our implementation, we ran `cvc5` on all the 7502 problems from the QF_UF logic from the SMT-LIB benchmarks library
- We ran the solver using each of the two new algorithms, as well as the original algorithm present in `cvc5` (from here on referred to as `VANILLA`)

Results

- Average runtime overhead was 1.18x for GREEDY, 2.68x for TREEOPT



- The new algorithms had no meaningful overall impact in the final proof size
- On average the proofs from GREEDY and TREEOPT were respectively 0.2% and 0.1% larger than the VANILLA proofs
- They were 80% smaller than the VANILLA proofs in the best case, and 70% bigger in the worst case

Results: local impact

- The final proof returned by the solver will include things other than congruence closure steps, and can be affected by many variables that are hard to control
- To measure the impact more directly, we also recorded the size of the proof returned by each call to `get_explanation`

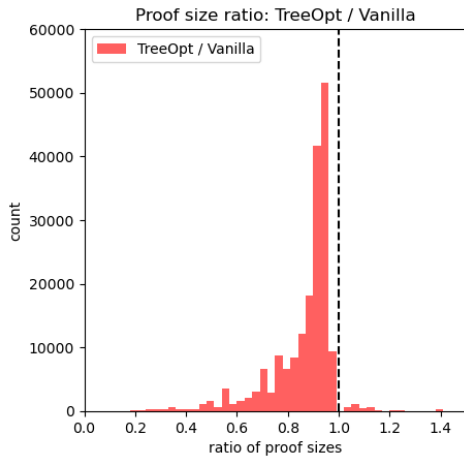
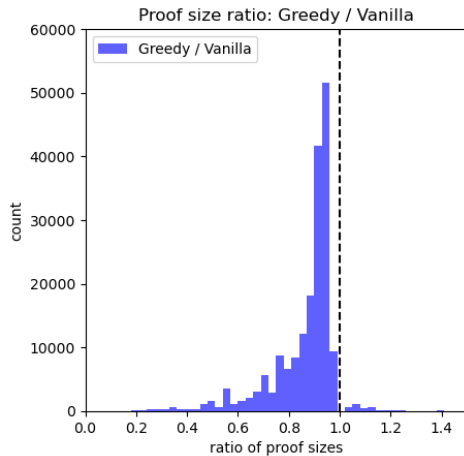
Results: local impact

- In 83% of cases, the proofs returned by the three algorithms had the same size
- On average, the “local” proofs from the two new algorithms are 2% smaller than the VANILLA proofs

Results: local impact

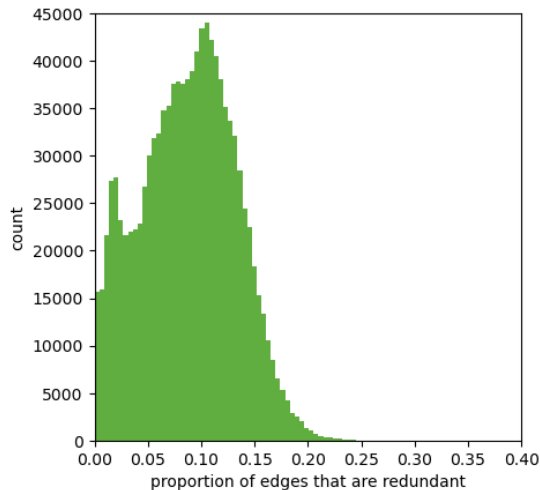
- In 83% of cases, the proofs returned by the three algorithms had the same size
- On average, the “local” proofs from the two new algorithms are 2% smaller than the `VANILLA` proofs
- If we exclude identical proofs, they are on average 14% smaller

Results: local impact



Results: extra edges

- We also measured the number of redundant equalities added
- On average, 8.7% of the equality graph edges were redundant, and in the most extreme case, 44.63%



- We were able to show that congruence closure algorithms that make use of redundant equalities can be efficiently implemented in a state-of-the-art SMT solver, with reasonable overhead
- However, they did not present a meaningful reduction in the final proof size
- Furthermore, even when looking at the local impact, the reduction in proof size is modest

- We were able to show that congruence closure algorithms that make use of redundant equalities can be efficiently implemented in a state-of-the-art SMT solver, with reasonable overhead
- However, they did not present a meaningful reduction in the final proof size
- Furthermore, even when looking at the local impact, the reduction in proof size is modest
- It is soon to say whether this or similar techniques will be practical for use in modern SMT solvers

- [1] Robert Nieuwenhuis and Albert Oliveras. “Proof-Producing Congruence Closure”. In: *Term Rewriting and Applications*. Ed. by Jürgen Giesl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 453–468. ISBN: 978-3-540-32033-3.
- [2] Oliver Flatt et al. “Small Proofs from Congruence Closure”. In: *2022 Formal Methods in Computer-Aided Design (FMCAD)*. 2022, pp. 75–83. DOI: 10.34727/2022/isbn.978-3-85448-053-2_13.
- [3] Haniel Barbosa et al. “cvc5: A Versatile and Industrial-Strength SMT Solver”. In: *Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part I*. Ed. by Dana Fisman and Grigore Rosu. Vol. 13243. Lecture Notes in Computer Science. Springer, 2022, pp. 415–442. DOI: 10.1007/978-3-030-99524-9_24. URL: https://doi.org/10.1007/978-3-030-99524-9_24.
- [4] Andreas Fellner, Pascal Fontaine, and Bruno Woltzenlogel Paleo. “NP-completeness of small conflict set generation for congruence closure”. In: *Form. Methods Syst. Des.* 51.3 (2017), pp. 533–544. ISSN: 0925-9856. DOI: 10.1007/s10703-017-0283-x. URL: <https://doi.org/10.1007/s10703-017-0283-x>.