DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Formalisation of a Congruence Closure Algorithm in Isabelle/HOL

Rebecca Ghidini

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Formalisation of a Congruence Closure Algorithm in Isabelle/HOL

Formalisierung eines Kongruenzhüllen-Algorithmus in Isabelle/HOL

Author:	Rebecca Ghidini
Supervisor:	Prof. Dr. Tobias Nipkow
Advisor:	Lukas Stevens
Submission Date:	15.09.2022

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.09.2022 Rebecca Ghidini

Acknowledgments

Thanks to Timmm and Manon.

Abstract

Contents

1 Introduction

1.1 Outline

Citation test [Lam94].

```
apply(simp)
apply(auto)
done
```

Figure 1.1: An example for a source code listing.

2 Preliminaries

2.1 Union Find with Explain Operation

2.2 Congruence Closure with Explain Operation

2.3 Isabelle/HOL

2.3.1 Union Find in Isabelle

3 Explain Operation for Union Find

3.1 The Union Find Data Structure

3.2 Implementation

3.2.1 Union

3.2.2 Helper Functions for Explain

3.2.3 Explain

3.3 Proofs

3.3.1 Invariant and Induction Rule

3.3.2 Termination Proof

3.3.3 Correctness Proof

4 Congruence Closure with Explain Operation

4.1 Implementation

For the implementation of the congruence closure algorithm, I followed the implementation described in the paper. [NO05]

4.1.1 Modified Union Find Algorithm

In order to implement an explain operation with reasonble runtime for the congruene closure data structure, the paper [NO05] introduced an alternative union find algorithm. The find algorithm remains the same, but a new data structure is introduced, called the proof forest, namely a forest which has as nodes the variables, and as edges the unions that were made. The forest structure is preserved, because redundant unions are ignored.

add_edge

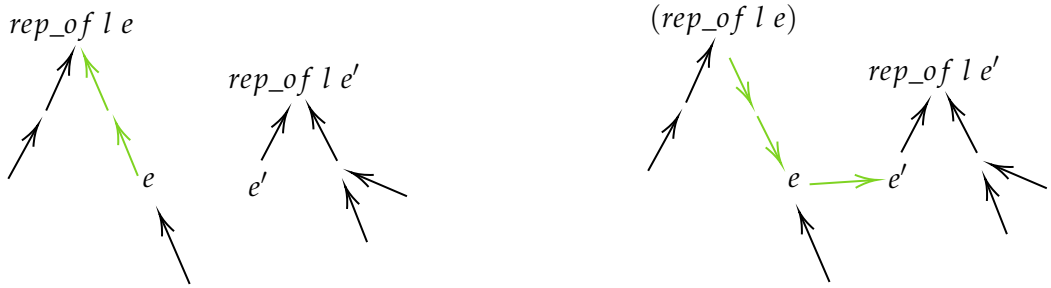
The tree has directed edges, and for each equivalence class there is a representative node, where all the edges are directed towards. To keep this invariant, each time and edge from e to e' is added, all the edges on the path from the root of to e are reversed. In my implementation, the forest is represented by an array which stores the parent of each node, exactly as in the union find array. My implementation for each added edge is the following.

```
function (domintros) add_edge :: "nat list => nat => nat => nat list"
where
"add_edge pf e e' = (if pf ! e = e
                     then (pf[e := e'])
                     else add_edge (pf[e := e']) (pf ! e) e)"
by pat_completeness auto
```

I was able to show that the `add_edge e e'` terminates, if the `ufa_invar` holds for the proof forest and e and e' do not belong to the same equivalence class.

```
lemma add_edge_domain:
assumes "ufa_invar l" "rep_of l y != rep_of l y'"
shows "add_edge_dom (l, y, y')"
```

Proof. I proved it by induction on the length of the path p from the root of y to y. The base case is when there is only one node in the path, therefore y must be equal to its representative, therefore `pf ! y = y`, and the algorithm terminates immediately. On the other hand, if y is not a root, there is a path p' from the root to the parent of y which is shorter than the path from the root to y. Given that only the y is modified in the recursive step, and y is not on the path p', the path p' is also present in the updated union find list. Also, the representative of y in the new list is equal to the representative of y', and the representative of the parent of y is still the old representative of y, therefore they are not in the same representative class, and we can apply the induction hypothesis and conclude that the recursive call terminates, therefore the function terminates. □



add_label

Additionally, each edge is labeled with the input equation or the input equations which caused the adding of this edge. This step is not necessaary for the union find algorithm by itself, but only for this algorithm when it is used within the congruence closure algorithm, because there are two possible reasons for the union of two elements a and b: either an equation $a = b$ was input, or two equations of the type $F(a_1, a_2) = a$ $F(b_1, b_2) = b$, where a1 and b1 bzw a2 and b2 were already in the same equivalence class before this union. Therefore we need to store the information about these input equations, in order to reconstruct the explanation in the end via the explain function. I implemented the labeling by using an additional list, which at each index contains the label of the outgoing edge, or `None` if there is no outgoing edge. The type of the label is `pending_equation`, which can be either `One equation` or `Two equation equation`, aka one or two equations. The name `pending_equation` derived from the fact that they are also the elements of the pending list, which is going to be described in the next section. Theoretically this allows also for invalid equations for example two equations of the

type $a = b$ and $c = d$, but we will prove in the next sections, that the equations in the labels list are always of a valid type.

Each time an edge, gets added to the proof forest, the labels need to be updated as well, not only the labels of the new edge, but also of the outgoing edges. The function which implements this is the following:

```
function (domintros) add_label :: "pending_equation option list => nat list => nat
=> pending_equation => pending_equation option list"
  where
"add_label pfl pf e lbl = (if pf ! e = e
               then (pfl[e := Some lbl])
               else add_label (pfl[e := Some lbl]) pf (pf ! e) (the (pfl ! e)))"
by pat_completeness auto
```

Similarly to the `path_to_root` function, `add_label` has the same recursive calls/case distinctions as `rep_of`, therefore it has the same domain.

```
lemma rep_of_dom_iff_add_label_dom: "rep_of_dom (pf, y) <-->
add_label_dom (pfl, pf, y, y')"
```

4.1.2 Congruence Closure Data Structure

4.1.3 Congruence Closure Algorithm

4.2 Abstract Formalisation of Congruence Closure

4.3 Correctness Proof

4.4 Implementation of the Explain Operation

5 Conclusion

5.1 Future work

List of Figures

Bibliography

[Lam94] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.

[NO05] R. Nieuwenhuis and A. Oliveras. "Proof-Producing Congruence Closure." In: *Elsevier* (2005).