CS560 Final Project
Congruence Closure Algorithm Implementation and Visualization

Adam El Youmi, Priya Kumari

12/11/2024

Introduction

This project is focused on implementing and visualizing the Congruence Closure (CC) Algorithm in a step-by-step manner. The primary objective is to develop an interactive and educational tool that simplifies understanding of the algorithm's mechanics. Designed for students and other interested learners, the tool allows users to input constraints, observe the formation of equivalence classes, and detect conflicts dynamically. It serves as both a learning aid and a practical demonstration of CC in action.

Overview

This project is focused on implementing and visualizing the Congruence Closure (CC) Algorithm in a step-by-step manner. Despite its widespread integration in SMT solvers, we found that no one had explicitly visualized the algorithm's process. Open-source implementations of CC are scarce, with only a few less-known repositories in Rust, Java, or Coq. Our goal is to fill this gap by creating an interactive tool that simplifies understanding and demonstrates CC dynamically for students and enthusiasts.

Our solution is coded in Python and used html for front-end implementation. Our source code can be found at the following repository. The Readme file in our repo contains a proper guide on how to run the program.

Our implementation is structured into the following primary steps:

- Formula reading and cleaning
- Arranging into a series of "equality" clauses and one "disequality" clause (python dict)
- Extraction of function names and literals (regex-based)
- Building a list of terms to obtain the initial equivalence classes before the algorithm runs
- Calling appropriate functions to infer if any two possible literals belong to same equivalence class
- Showing the result in a browser

We have used the following libraries:

- The **built-in regular expression package** handles regular expressions for parsing constraints.
- **Flask** provides a lightweight, easy-to-setup web framework for the interface.
- For managing the equivalence graphs, we utilized **NetworkX**, which allows us to add nodes, edges, and define graph layouts.
- We leveraged **Pyvis.network** to convert these graphs into interactive, browser-based visualizations with features like zooming, dragging, and hover effects, making the algorithm accessible and engaging for users.

Some formulas that we have utilized to properly demonstrate our model's capabilities can be found on the repository in formula.txt.

Implementation

1. **Formula reading : Input from user in the browser**

In our project, app.py is a flask based application that provides a web interface for users to input a formula. The web application begins by prompting the user to enter a formula. Upon submission, the backend runs the script (graph.py) with the provided formula, displays the result, and disables the input field. Once the result is displayed, the browser provides the user with an option to continue by asking if they wish to input another formula.

If the user selects "Yes", the interface refreshes, allowing them to enter a new formula for processing. If the user selects "No," the browser displays a thank-you message, marking the end of the session. Importantly, if the user has not made a choice between "Yes" or "No," the input field remains disabled to ensure clarity and a smooth user experience.

2. **Formula sanitizing**

In this project, formula parsing and sanitization are handled by the cc.py script, which ensures that user-inputted formulas are properly formatted and structured for processing. The sanitize function removes unnecessary whitespace and splits the formula into individual clauses, separating them based on logical conjunctions (&).

3. **Formula parsing**

This is also handled by cc.py script. The get_clauses_dict function plays a crucial role in organizing the parsed clauses into a structured dictionary, making it easier to process them in subsequent steps of the Congruence Closure algorithm. The function iterates through the list of clauses obtained from the sanitize function and categorizes them based on whether they represent equality (=) or inequality (!=) constraints.

For clauses containing an equality (=) operator, the function splits the clause into two parts which are stored as a list under the corresponding index key in the dictionary. For clauses containing an inequality (!=) operator, the function similarly splits the clause into two parts. However, instead of assigning these parts to a numbered key, they are added to the list associated with the -1 key. This ensures that the inequality constraint is stored in a dedicated space for efficient handling later in the algorithm.

4. **Extraction of function names and literals (regex-based)**

Functions like get_terms and function_chain in cc.py extract individual terms and nested expressions from the clauses, ensuring that both simple and complex function arguments are accounted for. This parsing process creates equivalence classes of terms while managing nested functions and multi-argument expressions effectively. This ensures clean and consistent data for downstream computations.

5. **Merging equivalence class and visualisation**

Using graph.py, a graph structure is initialized to represent equivalences. Each term in the formula is treated as a node in the graph, while equivalences are represented as edges connecting the corresponding nodes. The code iterates through each equivalence class and examines potential term replacements within the class. For each term, it checks if any other term in the class can be replaced by a matching subterm from a different class. If a valid replacement is found, it merges the equivalence classes and updates the graph visualization. If the replacement term exists in the terms list, it triggers a merge, otherwise, it updates the message to indicate the term is not found in the list. The process ensures the congruence closure is updated dynamically, reflecting changes in the graph visualization.

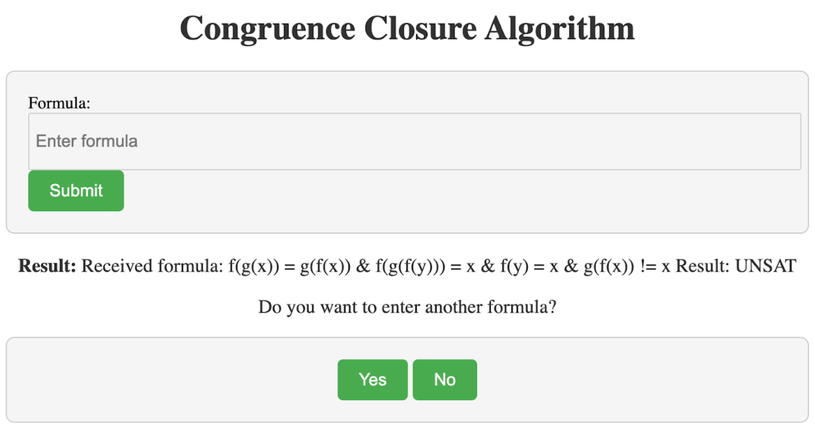
Color encoding is used to visually distinguish between different nodes and their relationships in the congruence graph. The colors used are:

1. **Red**: This color is assigned to the terms that appear in the inequality (!=) constraint in the initial formula provided by the user.
2. **Blue**: All other terms and extracted subterms that are not involved in the inequality are colored blue. This helps differentiate them from the inequality-related terms.
3. **Orange**: When two equivalence classes are merged based on the initial formula provided by the user, the edge between the merged terms is colored orange.
4. **Black:** When two equivalence classes are merged later by the subterm replacement mechanism in the core algorithm logic (e.g. new constraints that are not present in the initial formula), the edge between the merging terms is colored black.

Summary of Results

We used many formulas we came up with ourselves, as well as formulas from Homework 4, for both SAT and UNSAT cases, to verify our algorithm executes as expected. We ensured our algorithm relied both on the formula's constraints as well as additional inferred constraints to come up with the correct outcome. Some formulas that we have utilised to properly demonstrate our model's capabilities can be found on the repo in the formula.txt file.

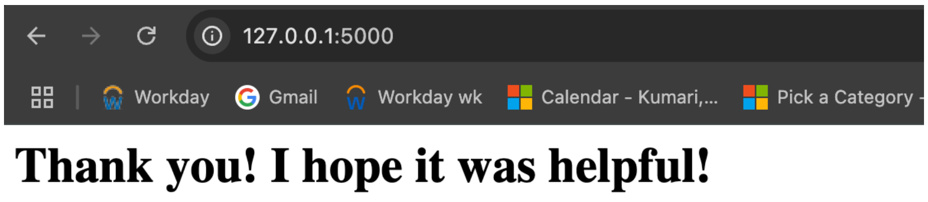
1. The web application provides an intuitive interface for users to input a formula and visualize the results of the Congruence Closure algorithm. After processing the input, the result is displayed, and the input field is disabled. Users can choose to enter another formula or end the session. If no choice is made, the input field remains disabled. This design ensures a smooth and user-friendly experience while effectively demonstrating the algorithm.


Congruence Closure Algorithm

Result: Received formula: f(g(x)) = g(f(x)) & f(g(f(y))) = x & f(y) = x & g(f(x)) != x Result: UNSAT

Do you want to enter another formula?

Figure 1: result of previous entered formula, Input field disabled, User wants to enter another formula: yes or no ?



Thank you! I hope it was helpful!

Figure 2: “Thank You” text when user says “No”

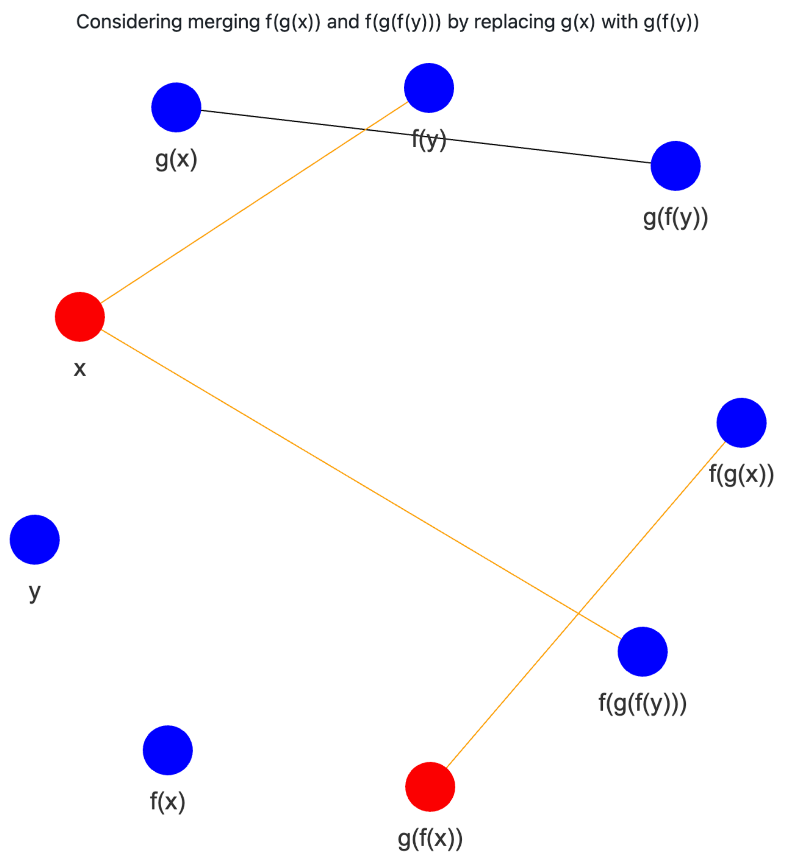

Considering merging f(g(x)) and f(g(f(y))) by replacing g(x) with g(f(y))

Figure 3: Intermediate execution step
The message explains what substitution is currently being considered

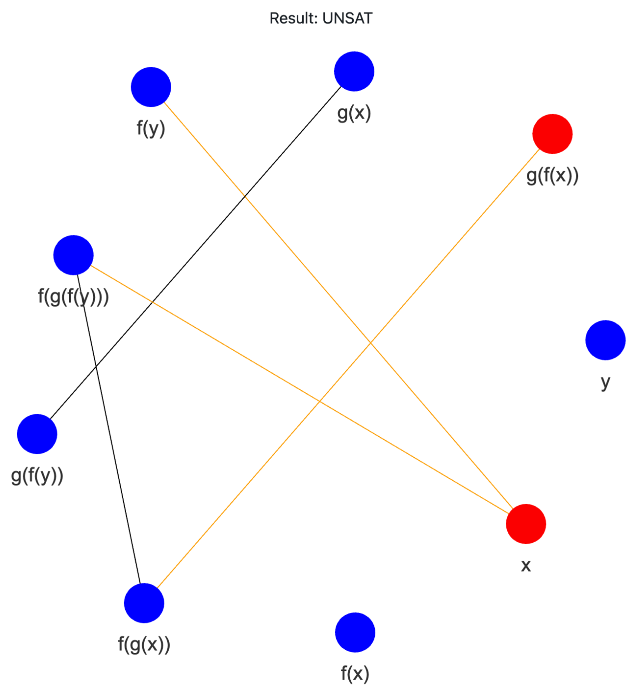


Figure 4: Result: UNSAT

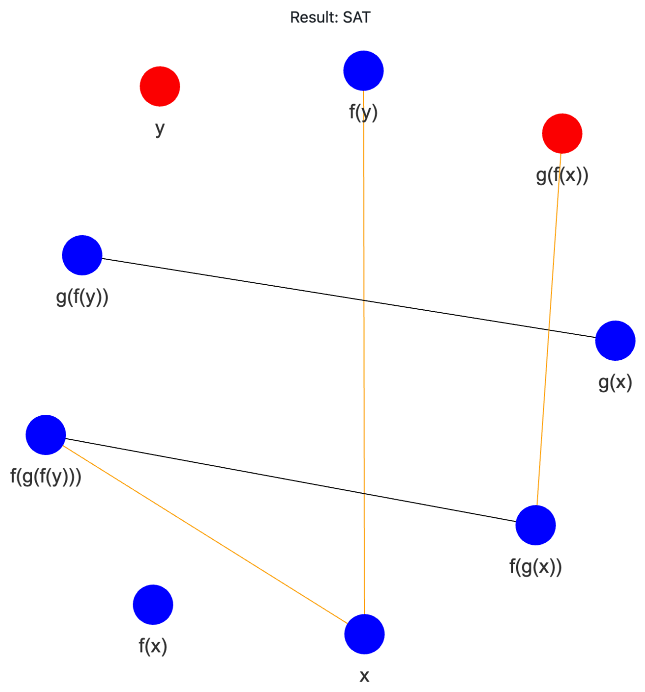


Figure 5: Result: SAT

We can see here how the two red (disequality) nodes are linked by a path in the graph when the formula is unsatisfiable, and how they are not when the formula is satisfiable.

Reflection

In this project, one of the key design challenges was ensuring the real-time, interactive visualization of the Congruence Closure algorithm. Integrating the backend (Flask and Python) with the front-end (interactive graph visualization using Pyvis and NetworkX) posed several technical hurdles. We had to ensure that the graph rendering was smooth and that the system could handle updates dynamically as the algorithm progressed.

In terms of algorithmic implementation, merging equivalence classes and managing the graph's dynamic state presented difficulties, particularly when dealing with complex term relations and inequalities. We used a series of condition checks and a visualizer class to show each step of the merging process. One challenge we faced here was maintaining a balance between computation time and visual updates to provide meaningful feedback to the user without overwhelming the interface. Another challenge was proper parsing of formulas to generate all possible subterms using regular expressions, especially when it came to multivariate functions like f(f(a,b),c).

What worked well was the integration of Flask with the Python backend, which provided a simple and effective way to build a user-friendly interface. The use of Pyvis for visualization made it easy to generate and interact with the graph, allowing us to show how the equivalence classes evolved over time.

For future enhancements, one can aim to implement a backtracking feature (which we did not), to allow users to navigate forwards and backwards to different stages of the execution. This would help users better understand the algorithm by reviewing earlier steps and observing how equivalence classes were merged. However, challenges such as managing visual graph layout consistency and performance issues with storing multiple graph states will need to be addressed. With further optimization in state management and memory handling, backtracking could enhance the tool's interactivity and educational value.

This work could form the basis for a real tool aimed at educational purposes, where students can input formulas and visualize how congruence closure works step-by-step. With future enhancement and any additional features one can think of, this could evolve into a full-fledged tool.

During this project, we gained hands-on experience with Flask for building web applications and integrating backend scripts, as well as working with NetworkX and PyVis for graph visualization. We also deepened our understanding of algorithms like congruence closure and data manipulation using Python, enhancing our skills in creating interactive, user-friendly tools.

Teamwork

Throughout the project, we worked closely together, dedicating equal time to each phase of development. Adam shared his knowledge on the core algorithm and graph management, while Priya contributed in building the Flask application and visualization. Together, we learned how to create graphs using NetworkX, experimenting with its features to refine the visualization.

Course Topics

In this project, we applied several key concepts from the course, particularly related to algorithms and data structures. We utilized **graph theory** for visualizing and managing these equivalence classes, employing libraries like **NetworkX** to model the relationships between terms. Additionally, concepts from **formal logic** covered in class were relevant in understanding the formulas and constraints being processed. The implementation of the algorithm in Python, alongside the creation of an interactive web application using Flask, helped reinforce our understanding of these topics.