Congruence Closure Solver

Project Report 2024-2025

Luca Panariello VR

1 Introduction

The project consist in the creation of a solver for the satisfiability of formula that belong to three different theories: theory of equality with free symbols, non-empty possibly cyclic lists, and arrays without extensionality.

The solver use the **Congruence Closure Algorithm** on a DAG, and has tree possible variation or euristics that can be choosen optionally:

- Non-recursive `FIND` function (`"r"`)
- Heuristic `UNION` function (`"e"`)
- Forbidden Set use (`"f"`)

This project is written in **Java** language and need to be compiled in order to use it, in the main folder there are two bash files that can help to build and run the program.

Along the solver there are a **generator** for the creation of synthetic set of literls customizable by the user (under a cert exent), and a parser for a limited set of **QF_UF SMT-LIB** files.

2 Project Folder

The project folder is organized in the following way:

- **classes**: this folder contain the core classes of the solver and the Congruence Closure Algorithm such as `Node` and `DAG`.
- **compiled**: this folder contatin the compiled java files (.class) since the solver work on the main project folder, no files should be added or edit.
- **debug**: this folder contain classes for debug purpouses.
- **generator**: this folder containain the properties files.
- **input**: this folder containg the txt files that have the formula or set of literals to be checkd, those files use the basic syntax of the solver.
- **output**: this folder contain the result submitted in the input folder.

- **preprocessing**: this folder contain the classes used for preprocessing purpouses.
- **smtlib_input**: this folder contains the .smt2 files from SMT-LIB without edit.
- **tests**: this folder contain files for test purpouses.

The main class is `Congruence Closure Solver` which act as central manager and is responsable to to read the name file given in input (that have to be in `input`, `generator`, `smtlib_input` folder based on the type of file), the options and retrieves the formula using a `FormulaReader`. The formula will be passed to a parser and then to the algorithm.

3 Input

The program accept both a set of literals delimiteted by `;`, that a formula with logical connectives.

It is also possible to submit a generator file with `.properties` extention to create a customizable set of literals, with randomic mechanisms.

Or to submit directly a `.smt2` file from QF_UF SMT-LIB, not all the files are compatible since SMT-LIB language is very complex.

More info about the syntax avaiable in the `README` file.

4 Parsing

The retrieved formula is submitted by a chain of parser:

1. `DNFParser`: after dropping the existential quantifiers and the universal quantifiers, parse the input formula and cast it in a `DNFTree`, a n-tree where the nodes are logical connectives or predicates, and the edge are the the scope of the logical connectives. (More info about the precedence used in in the `README` file) [IMMAGINE]
2. `LogicParser`: use the `DNFTrasformer` handle the a chain of transformation that reduce the `DNFTree` to a tree with an `or` root that have only have `and` children, which will have only predicate children (already negated if is the case), which will be the leaves.

   Then cast the leaves into an `ArrayList<String>` which will rappresent the list of cubes to solve one at time.

   Another task that the `LogicParser` performes is the splitting required upon receiving a formula that contain an istance of `select(score(...))`, the parser check the occurence and split the formula in the two versions, since it has handling a list of cubes (`or`), than the parser removed the splitted cube, and intail the new ones, ready to be checked at the end, and performing the operation again if necessary. [FORMULA?]