# High-Level Implementation Overview

## Group

- Vitor Fitzherbert Souza 2021031793
- Guilherme Soeiro de Carvalho Caporali 2021031955

## Code

The complete code used in this project can be found at github: vitorfitz/smt

## Introduction

We developed a software system aimed at solving Quantifier-Free Linear Real Arithmetic (QF_LRA) problems. These problems are extracted from the SMT-LIB benchmarks, widely used for formal verification and testing of algorithms in the domain of satisfiability modulo theories (SMT). SMT problems involve determining the satisfiability of logical formulas with respect to combinations of background theories.

The implementation leverages Python and integrates with multiple libraries to handle the parsing, abstraction, and resolution of these problems. SMT-LIB benchmark problems are available at Zenodo, providing a rich dataset for testing and development.

---

## Key Features

### SMT-LIB Parsing and Formula Abstraction

- **Parsing**: Using the `pysmt` library, we parse SMT-LIB problem definitions into Python objects representing logical formulas.
- **Boolean Abstraction**: Logical operations involving real numbers (e.g., `>`, `<`, `=`, etc.) are abstracted into Boolean variables. This abstraction allows the reduction of QF_LRA problems into propositional logic suitable for SAT solvers.

### Conversion to Conjunctive Normal Form (CNF)

The system uses the Tseitin transformations to convert the input formula into an equisatisfiable (not necessarily equivalent) formula in the CNF, since the CDCL algorithm requires formulas in this form.

The Tseitin transformations consist in adding new variables to the formula that assert the validity of each operation in the formula, resulting in an output that is linear in the size of the input expression, as opposed to the strict CNF conversion, which can increase in size exponentially. In the initial versions of the program the Tseitin transformations weren't used, however their implementation was crucial to enable the solution of more complex problems in reasonable amounts of time.

### Integration with SAT Solvers

- The propositional formulas in CNF are processed using the `pysat` library's Glucose3 solver, which efficiently finds satisfying assignments or proves unsatisfiability.

### Iterative Refinement

- A model generated by the SAT solver is verified against the original SMT problem using `pysmt` and the `z3` solver. If the model is invalid, an unsatisfiable core is derived, and the SAT problem is refined iteratively.

---

## Development Process

### Dataset Selection

- We utilized SMT-LIB benchmarks for Quantifier-Free Linear Real Arithmetic (QF_LRA). These datasets provide challenging formulas that reflect practical use cases in software and hardware verification.

### Key Libraries

- **pysmt**: For parsing and formula manipulation.
- **pysat**: For SAT solving.
- **z3**: For validation and unsatisfiable core generation.

### Testing

- The implementation was tested on various formulas from the SMT-LIB dataset to ensure correctness, efficiency, and compatibility with industry standards.

---

## Benchmark

In order to evaluate the performance of the implementation, a set of benchmarks was used. The extensive list problems can be found in the folder pj2-tests, also the results can be found in the *csv*.

A thorough analysis of the results of the benchmark indicates that the implementation is correct in almost every case, only yielding a misleading result in one of the problems, most likely due to the incorrect handling of deeply nested ITE (if then else) statements.

In regards to the time to solve each problem, we can see that the implemented solver takes much more time and is way less efficient than the standard ***z3***. This happens due to the extensive optimization techniques and the years of incremental improvements in the development of z3, while our version has only the simplest and more common optimizations.

Most notably, the program takes a long time to find a satisfiable assignment in some cases where one exists. This stems from the lack of heuristics to explore more promising assignments first. These heuristics weren't used because they are complex and internal to the solver, not exposed by the pySMT API.

The results obtained by running the benchmarks only prove the value of optimizations in the algorithm implementation. Each improvement, being small or big, when compounded over the long time of development of systems of this kind, results in a huge impact in the final performance.

---

## Conclusion

This project successfully implements a robust method for solving QF_LRA problems, demonstrating efficient integration of parsing, abstraction, CNF conversion, and SAT solving techniques. The system's iterative refinement ensures precise results, and its modular design makes it adaptable for extensions and other SMT problem domains.

---