CS335 Report 1

Dhruv, Kundan, Pragati

10th Feb, 2025

We successfully completed the first two milestones outlined in our project description:

- **Syntax Design & Parsing**: Defined the syntax for satisfiability constraint files and implemented a parser to accurately interpret and validate them. Additionally, we researched and analyzed the SMTLIB format for translating constraints and Chiron IR into SMTLIB.
- **SMTLIB Code Generation**: Developed SMTLIB code generation for assignment statements.

Milestone I: Constraints File

Syntax Overview

- A list of literals mapped to expressions.
- Each literal represents a comparative expression (e.g., `x < 20`, `y = 5`).
- The CNF (Conjunctive Normal Form) is expressed in terms of literals:
  - Each line represents a disjunction (OR) of literals.
  - The entire file represents a conjunction (AND) of all such lines.
  - denotes logical negation (NOT).

Example:

$$[\,[\,c, \sim a\,],\ [\,d, \sim e, \sim f\,]\,]$$

This represents the CNF formula:

$$(c \vee \neg a) \wedge (d \vee \neg e \vee \neg f)$$

Modules Implemented

1. `ConstraintParser.py`

Parses the constraints file and constructs two key data structures:

- **expr_dict**: A dictionary mapping literals to their corresponding expressions (stored in infix notation for readability).
- **literal_groups**: A 2D list representing the CNF expression in terms of literals.
  - Each inner list corresponds to a clause (disjunction of literals).
  - The entire list represents the CNF formula (conjunction of clauses).

2. `PrefixConvertor.py`

Converts expressions from **infix notation** to **prefix notation**, as required by SMTLIB.

- Constructs an **Abstract Syntax Tree (AST)** for the given expression.
- Performs a **preorder traversal** to generate the prefix notation.

3. `SMTLIBConvertor.py`

Uses `literal_groups` and `expr_dict` (converted to prefix notation) to generate the equivalent **SMTLIB representation** of the CNF expression.

4. `main.py`

Integrates all the modules:

- Takes the constraints file as a command-line argument.
- Parses and processes the constraints.
- Generates the corresponding **SMTLIB code** for the CNF formula.

Milestone II: SMTLIB Code for Assignment Statements

Modules Implemented

`IrToSmtlib.py`: This module includes a function that processes Chiron IR by iterating over each instruction, converting expressions from infix to prefix notation (using `PrefixConvertor.py`), and then translating them into equivalent SMTLIB assert statements. Currently, it supports only assignment statements.

- Adjustments made:
  - Removed the ’`:`’ prefix from each identifier name.
  - Replaced ’`=`’ with ’`==`’ for AST compatibility.

Usage

We have integrated this feature with the `chiron.py` module. To enable it, use the `-smt` or `--smtlib` flag. The **cmdparser** in `chiron.py` has been updated to support this flag.

```
python3 chiron.py –smt <path_to_file>
python3 chiron.py ––smtlib <path_to_file>
```

Currently, SMTLIB statements are printed to the console. Future improvements will include integrating a **Z3 solver module** to check satisfiability directly, eliminating the need to print SMTLIB code manually.