3/11/2019

Automated Reasoning about Software: Final Project

General Instructions:

- The project is due **28/2/2020**
- Submission is in pairs, unless otherwise approved. You are required to clearly indicate which partner contributed to which part of the code.
- The goal of the project is to give you an opportunity to implement the techniques learned in class, as opposed to checking your ability to handle corner cases and irregular inputs. Please make a reasonable effort to create a stable tool, but do not exaggerate. Document any major assumptions that you make.
- The project has 3 main parts, which correspond to the main topics covered in class. You are advised to complete each unit when we finish the material in class, as opposed to waiting until the last moment to write the whole project.
- You can use any reasonable programming language. When you submit your code, please provide a script to compile it on the school computers.

Part 1: SAT Solver

In this part of the project you will create a CDCL-based SAT solver. The solver will take as input a Boolean formula with the operators: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$, and will either determine that the formula is unsatisfiable, or return a satisfying assignment. The major components of the solver will include:

- Tseitin's transformation: code that transforms the input formula into CNF (in polynomial time)
- Preprocessing: remove redundant literals, delete trivial clauses
- Deduction steps: unit propagation
- Case splitting, with non-chronological backtracking
- Conflict analysis using implication graphs, with the Unique Implication Point heuristic
- Learning conflict clauses
- A decision heuristic (DLIS or VSIDS)
- Watch literals

Part 2: SMT Solver for Uninterpreted Functions

In this part of the project you will write a simple DPLL(T) SMT solver, that can determine the satisfiability of quantifier-free T_{UF} formulas. The solver will use the SAT solver from Part 1. Specifically, the SMT solver will:

- Take a T_{UF} formula, create a Boolean abstraction thereof, and feed it to the SAT solver.
- Retrieve a satisfying assignment from the SAT solver, and determine its T-satisfiability using an efficient implementation of the Congruence Closure algorithm.

- Have partial assignments checked by the theory solver
- Report any T-conflicts to the SAT solver
- Perform T-propagations from the theory solver to the SAT solver.
- Bonus: add support for a (non-trivial) T-explain rule for conflict analysis

Part 3: LP Theory Solver

In this part you will add a simplex based theory solver to your SMT solver. Note: you do not need to support theory combination; i.e., you can assume that the SMT solver will receive pure rational formulas. Also, you can assume that the input adheres to the standard form. You are required to implement:

- Bland's and Dantzig's selection rules
- The revised simplex algorithm
- LU basis factorization, and refactorization when the number of Eta matrices exceeds a threshold
- Periodic checks for numerical stability, and refactorization if needed.