Software Engineering and Programming Languages
Prof. Dr. Michael Leuschel
David Geleßus, Fabian Vu

Introduction to Logic Programming – WS 2023
Programming project

1 Submission and evaluation

It is sufficient if your submission passes all tests and benchmarks reasonably to be admitted to write the final exam. The runtime of your submission is not important. Of course, it is not acceptable to hard code any test.

You should work on the project alone. You can of course exchange information with one another, but please do not share any code.

Please contact us early if you have any problems or questions. You can either send an email to Fabian.Vu@hhu.de or dagel101@hhu.de, visit the tutorials or use the forum in the ILIAS.

Deadline: January 30, 2024

Exam: February 06, 2024

Submit your solution as a zip archive in the ILIAS.

2 Task: SAT Solver

The goal is to implement a Prolog programm that is able to transform propositional logic formulas into conjunctive normalform and solve them afterwards using the DPLL algorithm as discussed in the lecture. Please use the file `sat_solver.pl` for your implementation.

The input is a syntax tree that is built recursively from the following Prolog terms:

- literals $lit(\ldots)$
  - the constants `lit(true)` and `lit(false)`
  - Prolog variables such as `lit(X)`
- the equivalence $\rho \Leftrightarrow \phi$, in Prolog `equivalence(`$\rho$`,`$\phi$`)`
- the implication $\rho \Rightarrow \phi$, in Prolog `implies(`ρ`,`ϕ`)`
- the conjunction $\rho \wedge \phi$, in Prolog `and(`ρ`,`ϕ`)`
- the disjunction $\rho \vee \phi$, in Prolog `or(`ρ`,`ϕ`)`
- the negation $\neg\rho$, in Prolog `not(`ρ`)`

- `exactly_one_pos(ListOfVars)`: exactly one variable in `ListOfVars` is true
- `min_one_pos(ListOfVars)`: at least one variable in `ListOfVars` has to be true

The terms can be nested arbitrarily.

Your SAT solver should provide three Prolog predicates:

- `normalise(+Formula, -NFormula)` The first argument is a propositional logic formula using the terms described above. This predicate should normalise the input formula to only use constants, variables, conjunction, disjunction, and negation. Your implementation thus has to rewrite equivalences, implications, and the terms `exactly_one_pos/1` and `min_one_pos/1`.
- `to_cnf(+Formula, -CNF)`. The first argument is a propositional logic formula using the terms described above. Your implemenation should call `normalise/2` for the input formula first. The conjunctive normalform should be returned as a **list of lists** in the second argument. Hereby, each inner list is a clause (a disjunction) while the list of lists is connected by a conjunction.

  Examples:

  - `and(lit(X),lit(true))` is transformed to `[[X],[true]]`
  - `and(lit(X), or(lit(true),lit(false))` is transformed to `[[X],[true,false]]`

  The negation is still represented as the term `not/1`:
  `and(not(lit(X)),lit(true))` is transformed to `[[not(X)],[true]]`.
- `solve(+CNF)`. The predicate receives a CNF as returned by `to_cnf/2` and tries to find an assignment of variables occurring in `CNF` that make the formula satisfiable. The variables (`X` in the above example) should be assigned with either `true` or `false`. The predicate should fail if there exists no satisfiable assignment of variables for a propositional logic formula. Implement the **DPLL algorithm** for this as discussed in the lecture. That means, you should implement unit propagation, clause simplification, and variable branching.

The benchmarks can be found in `resources/sat_benchmarks/` and can be run using `run_benchmarks/0` in `sat_benchmarks.pl`.

Note that the problem of satisfiability solving is NP-completeness.

The file `large_formula_example.cnf` contains a more complex propositional logic formula which your solver can probably not solve in a reasonable amount of time. In order to do so, we need more sophisticated techniques of SAT solving such as conflict-driven clause learning or a watched literals scheme. Such techniques are not discussed in the lecture since they are beyond the scope of this introductory course.