104824 – INTRODUCTION TO SYSTEMS PROGRAMMING – Spring 2025 ..... Assignment
Sat Solver

Academic Integrity. You are reminded that you should understand and comply with basic standards of academic integrity –in particular, in matters related to misrepresentation by deception or by other fraudulent means of submitted work. Lack of academic integrity can result in serious consequences – the minimum of which is a grade of 0 (zero) on the assignment. The following illustrates only three forms of academic dishonesty: (1) plagiarism, e.g., the submission of work that is not one's own or for which other credit has been obtained; (2) improper collaboration; (3) copying or using unauthorized material.

Submission. You must submit your solution as a *.zip* file containing Java files with *.java* extension. As for any additional files you want to submit, the only accepted formats are: plain text file (*.txt*); markdown files (*.md*); and PDF files (*.pdf*). Your solution must also include any file provided, with or without your modifications, except for this pdf.

Accompanying this guide, I'll provide a *Java* project, contained inside a *sat-solver.zip* file.

Context

The provided code allows to represent boolean expressions, to construct boolean expressions, and to evaluate them. The types of boolean expressions that we will use are as follows:

- Constant: `C` either `True` or `False`.
- Variable: `Var v` a name, that requires an interpretation to be evaluated.
- Negation: `Not E` the negation of a boolean expression.
- Conjunction: `LEFT AND RIGHT` a conjunction of two boolean expressions.
- Disjunction: `LEFT OR RIGHT` a disjunction of two boolean expressions.
- Exclusive disjuntion: `LEFT XOR RIGHT` an exclusive disjunction of two boolean expressions.
- Implication: `ANTECEDENT IMPLIES CONSEQUENT` an implication of a boolean expression by another boolean expression.
- Bi-implication: `LEFT IFF RIGHT` a bi-implication of two boolean expressions.

The semantics of these expressions are given in the context of an interpretation `I`, a mapping of variable names on to constant `boolean` values, i.e.: `true` and `false`.

- `eval(C, I)` evaluates to `C`.
- `eval(Var v, I)` evaluates to `I[v]`.
- `eval(Not E)` evaluates to ¬ `eval(E, I)`.
- `eval(LEFT AND RIGHT, I)` evaluates to `eval(LEFT, I) and eval(RIGHT, I)`.
- `eval(LEFT AND RIGHT, I)` evaluates to `eval(LEFT, I) or eval(RIGHT, I)`.
- `eval(LEFT AND RIGHT, I)` evaluates to `eval(LEFT, I) xor eval(RIGHT, I)`.
- `eval(ANTECEDENT IMPLIES CONSEQUENT, I)` evaluates to ¬ `eval(ANTECEDENT, I) or eval(CONSEQUENT, I)`.
- `eval(LEFT IFF RIGHT, I)` evaluates to `eval(LEFT, I) = eval(RIGHT, I)`.

Exercises

All implementations must consider the check of preconditions, postconditions, and invariants, using exceptions, even if there are no explicit task or instruction to do so. Any missing precondition, postcondition, and invariant, must be completed. **Checks for postconditions must only be implemented when the method does not return**.

Task 1 Complete implementation of all incomplete methods, search for `TODO` comments and `UnsupportedOperationException`s.

Task 2 Modify *Main.java* to build some specific expressions, for each expression you must print the following:

1. The expression itself (using method `toString()`).
2. The variables of the expression (on a single line).
3. All possible interpretations (one on each line).
4. If the expression is satisfiable.
5. If the expression is a tautology (or not).
6. If the expression is a contradiction (or not).
7. All interpretations that satisfies the expression (one on each line).
8. All interpretations that do not satisfy the expression (one on each line).

The expressions to use are the following:

Expression 1 `p or (not p)`

Expression 2 `p and q`

Expression 3 `False implies q`

Expression 4 `p implies (not p)`

Expression 5 `p and (not p)`

Expression 6 `p xor (q implies r)`

Expression 7 `(not (p or q)) iff (not p and not q)`

Task 3 Could interface `Expression` be a `FunctionalInterface`? *Justify your answer.*

You can add any method or class that you consider necessary, although the necessity and correctness of that class and/or method will be evaluated.