Finding redundant sub formulas in SAT  
Formal Methods for Software Engineering WS22/23

S G D S V V RamanaMurthy Katakam (123549)

Group AG

Bauhaus-Universität Weimar

**Abstract.** Automated theorem proving has long been a focus of computer science research. In this field, logical formulas are specified using a variety of languages, such as Limboole. This SAT solving serves as a pivotal point for computational problem-solving across diverse domains, like AI planning. This report shows you how I identified redundant sub formulas and elements in sub formulas which will impact solver efficiency.

Initially, redundant sub formulas are identified using some other methods like elimination, sub assumption which have limitation and time complexity issues. So here I have used semantic entailment between formulas to find redundancy. Here in this project, I used DIMAC format, and CNF significance is also mostly important in this aspect since when formula is defined in SAT then we have to convert to CNF for further usage as DIMAC. This conversion is mainly done using SYMPY, PYCOSAT, ITERTOOLS library.

Finally, once any redundant elements are found in the formula, you see the resulting formula when the particular redundant element is removed, or redundant sub formula is removed.

1. Introduction
   1. Background

The language Limboole is straightforward and effective for expressing Boolean formulas. Due to its simplicity and effectiveness in solving complex issues, it is frequently used in automated theorem proving systems. Boolean variables and operators like AND, OR, and NOT are used in Limboole formulas. Standard Boolean operations as well as some sophisticated features like XOR and implication are supported by the language. Typically, Limboole formulas are presented in text-based, easily readable formats.

The existence of redundant sub-formulas within the formula representation poses a substantial issue in SAT solution. Redundant sub-formulas are formula components that do not add to overall satisfiability or excess in the formula but may unduly increase the search space and computer resources needed to solve the problem. Identifying and deleting redundant sub-formulas is critical for increasing the efficiency and scalability of SAT solvers.

CNF is a standard form which helps to represent these logical representational SAT formulas as conjunction of clauses and each clause are with disjunction of literals. It is also one of the standardized and structured ways of representing these Boolean formulas. Using some methods or techniques like elimination of implications, negation normal form conversions are helpful to convert any Boolean or logical formulas into CNF standard.

DIMACs is a standard file format type where it is used to represent the Boolean formulas precisely for SAT instances. Also, it is an efficient way to encode this logical formula after converting them into Conjugative normal form (CNF). This Dimac format will have list of clauses that are held together with AND operand, and each of these clauses are made with OR operand, thus we can say that Dimac is simple with Or and And operands.

* 1. Motivation

My project goal is to offer a quick and effective method for finding redundant sub formulas in the provided SAT formula or dimac formula. To make this possible I have used Python mainly to use libraries efficiently since SAT solvers are NP based problems which can be solved efficiently using libraries.

In contrast to make it more useful the whole process of finding redundant sub formulas are done using DIMAC (Discrete Mathematics and Theoretical Computer science) this is achieved from cnf (conjugative normal form) of SAT formula in textual format.

1. Problem Statement
2. Analyze a given Limboole or DIMACS formula for redundant sub formulas and elements.
3. Tool should produce a list of redundant elements and a simplified specification where the redundancy is removed.
4. Example & Intuition

**Understand the DIMACs syntax and proportional logic parsing.**

* Initially we define the formula and also variables.
* Then using conversion class, we define sat formula to it and then we do next steps as follows.
* Converting the proportional logic into CNF and then to DIMACs
* Then we make sub formulas from the original formula
* We then check for semantic entailment of modified formula to original formula.
* If they are semantically equivalent, then that particular removed sub clause is redundant. Else it is non redundant.

1. Implementation

formula\_conversions: initially this class helps to read the SAT formula and convert the SAT formula into CNF and also to DIMACs structure, this SAT to CNF conversion is done with help of sympy library and mainly used to\_cnf function from sympy.

1. In case formula is declared as SAT formula, then this class is useful, where we can get cnf formula using to\_cnf, and then we get individual variables from this cnf formula.
2. We can also get corresponding dimac formula when SAT formula is declared, while getting this dimac formula we use mappings to make sure we correspond to integer values.
3. We can also get cnf formula from dimac formula.

dimac\_solver: in this code the dimac formula is solved using libraries like pycosat, itertools and copy. picosat is used to check the satisfiability of dimac formula. This particular code is implemented to check the satisfiability of the formula and also check semantic entailment between two formulas and also check semantic entailment while modifying the Dimac structure negations.

1. Individual static methods are implemted for checking satisfiability, semantic entailment, restructuring dimac formula while checking semantic entailment of the formulas.
2. We can also check semantic entailment while modifying the individual elements in clauses by varying their sign i.e negating the element in clauses.
3. I also implemented code for negating the dimac formula.
4. I also implemented code for getting subsets from the dimac formula by removing each clause at a time.
5. check\_semantic\_entailment is for checking the semantic equivalence of two formulas.
6. check\_dimac\_redundant\_elements is for checking the semantic equivalence for individual elements in the dimac formula.
7. The above implementations of two methods check\_dimac\_redundant\_sub\_formulas, check\_dimac\_redundant\_elements return list of lists with modified formula when redundant elements or redundant sub formulas exists.

mappings: this particular python code is implemented separately just to make it private. And this is used for mapping the variables to values for Dimac structure.

1. This is particularly implemented for mapping the variables to integers and vice versa. Also, when random variables are generated and assigned to integer values when dimac formula is directly declared.
2. This is implemented separately just to make sure it won’t be accessed directly to make sure it as encapsulated.
   1. Code Example

This code is for checking semantic entailment of original formula and modified formula and returns True (if they are semantically equivalent) returns False (if they are not):

|  |
| --- |
| **def** restructure**(**f1**,** f2**):**  result\_formula **=** **[]**  **for** clause **in** f1**:** result\_formula**.**append**(**clause**[:])**  **for** clause **in** f2**:** result\_formula**.**append**(**clause**[:])**  **return** result\_formula  **def** check\_semantic\_entailment**(**main\_form**,** sub\_form**):**  negated\_sub\_form **=** negated\_dimac**(**sub\_form**)**  negated\_main\_form **=** negated\_dimac**(**main\_form**)**  form1 **=** restructure**(**main\_form**,** negated\_sub\_form**)**  form2 **=** restructure**(**sub\_form**,** negated\_main\_form**)**  satVal\_form1 **=** check\_satisfiability**(**form1**)**  satVal\_form2 **=** check\_satisfiability**(**form2**)**  **if** satVal\_form1 **==** "UNSAT" **and** satVal\_form2 **==** "UNSAT"**:**  **return** **True**  **else:**  **return** **False** |

* 1. Tool usage

PyCharm is a popular IDE (integrated Development Environment) that is widely used for python development. It is developed by JetBrains and also a cross-platform application that can be used on various operating systems.

Python developers frequently use PyCharm, which offers a variety of tools and features that make it simpler to write and test Python code. Developers can choose the Python version interpreter whether Python2.x, or Python 3.x.

PyCharm offers a number of coding aid features, including syntax highlighting, code completion, and code refactoring, once a project is configured. These tools make it easier for developers to write Python code that is clear, effective, and manageable.

Moreover, PyCharm comes with a strong debugger that helps programmers easily debug Python code. Debuggers enable developers to walk through the code, examine all instances, and do real-time data structure analysis. This facilitates the identification and resolution of issues that arise throughout the development process.

All things considered, PyCharm provides Python programmers with an easy-to-use and effective development environment, enabling them to produce high-quality code with confidence and simplicity.

I implemented this project with the following options:

* $main.py or test1.py the main code and test code to run all other implemented algorithm codes.
* $formula\_conversions (mandatory) the to convert the SAT formula to CNF and also to DIMACs.
* $dimac\_solver (mandatory) helps to solve the converted Dimac for semantic entailment and satisfiability.
* $mapping (mandatory) to convert and reconstruct the formula into DIMAC or to CNF format from SAT format and DIMAC for values mapping.
  1. Limitations of the Implementation

Converting SAT format to DIMAC using CNF and handled the remaining code worked well:

1. Before working for the redundancy in the formulas, I wanted to convert the sat format to dimac since parsing of SAT formula is difficult for further operations.
2. Thus, I eliminated all other operands other than And, Or, Not in formula using CNF.
3. The only main limitation is that even though when I got the redundant elements or redundant sub formulas in the formula, it is quite complex to determine at what position they are relative to original SAT formula on the other hand if the input is given in CNF or dimac then this won’t be a limitation.
4. Discussion & Conclusion

To conclude, I could successfully find the redundant sub formulas in the SAT formula if present as list of clauses in the formula. I did a decent job on this project. Working on this project allowed me to gain more in-depth knowledge of the characteristics of the Limboole formula, different libraries in python, CNFs, DIMACs and the OOPS concept for Python-based language. Even though I encountered minor difficulties while implementing the code, such as usages of different python libraries in order to work with Dimac structures than SAT formula, I’m ultimately able to finish it very quickly.

References

1. Lectures on Formal Methods for Software Engineering.
2. Some examples generated to test the code are made out from ChatGPT.
3. Sympy, pycosat, itertools, copy libraries are used in python.
4. Code published in [GitHub](https://github.com/DattaKatakam/finding_redundant_elements) .