Artificial Intelligence and Decision Systems
Mini-projects, 2017/2018
Assignment #2

Instituto Superior Técnico
João Louro 78450
Emanuel Fernandes 66088

1 Problem Statement

A possible Resolution-based propositional logic reasoner can consist of two main elements:

1. a converter of Logical Sentences from propositional logic into the Clausal Normal Form (CNF);
2. a Theorem Prover for Propositional Logic, based on the Resolution principle, which assumes a Clausal Normal Form knowledge base.

This report describes the implementation of such a logic reasoner, developed in the version 3.6.1 of the Python Programming Language.

2 Conversion to CNF

Input to Knowledge Base

The first part of the program to be developed corresponds to the conversion of the logic statements in the input file to the Causal Normal Form. In order to do this, the CNF converter program, which is called *convert.py*, must read from *stdin* sentences as was asked in the assignment. The reading of the file is performed by the function *read_stdin*.

Having access to the input file, the next action corresponds to building the knowledge base from the information present in the file, in a way that can be processed by the program. The storing of the sentences in the knowledge base is done by the function *correctly_into_lists*.

The obtained knowledge base is a list, in which every sentence is also a list. Each of these sentences in list form may have from one to three entries. A relation between the size of the sentences and its "type" can be inferred. If a sentence is a list of one entry, it corresponds to an atom. If it has two entries, it is a negative sentence. In the case it has three entries the sentence can be an implication, conjunction, or other kind of logic statement. The propositional logic sentences are sometimes quite complex, which leads to the appearance of lists inside of lists, with several levels. This leads to the recursiveness of the function *correctly_into_lists*.

Transformation to CNF

After having the knowledge base in the right form, its sentences are translated into the CNF, considering the transformation rules. This transformation in performed on every sentence in the knowledge base, by applying logical rules to it, with the goal of obtaining a conjunction of disjunctions. In the code, this is performed by function *to_cnf*. This function will then receive as *input* a sentence of the knowledge base, and, by applying the transformation rules in a recursive manner, will return as *output* the same sentence in CNF form.

3 Resolution Prover

After having the knowledge base in the CNF form, the resolution prover was developed. This prover receives the sentences of the knowledge base in the CNF form and proves whether a conjecture α can be derived from the knowledge base ($KB \vDash \alpha$).

In order to achieve this, the resolution solver starts by creating a set of clauses (C) that contain the KB and the negated conjecture ($C = KB \cap \{\neg\alpha\}$). Using the Full Resolution Rule, **unit preference** strategy and some simplifications (isolated literals, tautology removal, idempotent subsets and **factoring**), the algorithm iteratively infers new clauses.

A conclusion is reached when the empty clause is generated (*true*, proof by contradiction) or no new clauses have been generated (*false*). This is done by checking if the newly generated clauses are a subset of C. The program ends by *outputting* a True or False conclusion.

4 Strategies and Simplifications Employed in the Resolution

Using the Resolution Rule to solve CNF problems can be intractable depending on the problem complexity, so in order to increase efficiency, there are some strategies that can be implemented.

Unit Preference

This strategy prefers unit clauses (clauses with only one literal) in order to derive the empty clause faster. For instance, for multiple unit clauses, this happens when the resolution is applied to a literal and its complement. It can also be shown that when the resolution is applied to a unit clause and a sentence containing its complement, a shorter clause will always be derived. This strategy is very effective early on for simple problems. However, it doesn't reduce the branching rate for medium-complexity problems.

Ordering by number of clauses

In the beginning of every resolution cycle, the set of clauses C is reordered by decreasing length (unit clauses in the last positions). This is a generalization of the unit preference strategy, since smaller sentences will derive

other smaller sentences faster. Again, this strategy is mostly effective for simple problems.

Isolated Literals

If a clause contains a literal that has no complement anywhere else in the rest of the set, that clause can be removed. That literal will never be resolved due to the fact that any new clauses derived from that will always contain the literal, therefore never achieving the empty clause.

Tautology Removal

If a clause in C is a tautology, it can be safely discarded (deleted from C), since no new clauses can be derived from it (redundant information). Whenever new clauses (*resolvents*) are generated by the full resolution rule, if they're a tautology, they are also discarded.

Factoring

If a literal occurs more than once in a clause, the duplication can be removed (Idempotency of disjunction).

5 Performance Analysis

A performance analysis of both programs working in tandem was done on a computer with an *Intel Core I5-6500 @ 3.20 GHz* $\times 4$ processor. An average branching factor (**b**) was calculated by the average of new clauses created (discovered) on each iteration. The running time for each of the input files provided by the professor is shown in table 1:

Text File	Time (s)	b	N. Iter.
sentences.txt	0.032	2.0	1
trivial.txt	0.036	1.0	2
p1.txt	0.056	7.8	5
p2.txt	0.036	0	1
p3.txt	0.032	5.7	3
p4.txt	0.028	3.0	1

Table 1: Performance Results

The problems made available are of relatively low complexity, since the maximum branching factor was 8 and the maximum number of iterations was 5. Even so, some conclusions can still be drawn from the table:

- All the programs are quick at reaching the conclusions, due to both the simplicity of the problems and the unit clause strategy;
- The average running time depends on processor availability, therefore its values are not very relevant;
- Number of iterations is the most significant value for the algorithm performance, since the prover has to go through all the combinations in the set of clauses each time;
- The average branching factor influences the increasing complexity within each iteration (every clause learned adds to the number of combinations);
- Some of the problems, although more complex, run faster. This is probably because they are being aided by the fact that the processor uses multiple cores to parallelize (and therefore accelerate) both the prover and the converter algorithms.

6 Conclusion

The use of recursive functions was essential in this project, namely in the CNF converter section. This is due to the fact that a sentence in propositional logic can be formed by other logic sentences. Besides the CNF converter, the logic reasoner was also developed, to prove theorems in propositional logic.

For the CNF converter part (*convert.py*), the code outputs the converted sentences correctly (in the form required in the problem statement). This means that all lines are in conjunction with each other, and that within lines, all literals are disjunctions of each other. This output can either be output to the terminal line or fed directly into the prover.

In the prover part (*prove.py*), the program follows the cycle described in page 244 of [1], and prints the final result to the terminal, with the conclusion reached: *True* if the knowledge base entails the conjecture, or *False* otherwise.

Even with no simplifications coded into the CNF converter part, since the main objective was to use both parts in tandem, the resolver handles all these simplifications, therefore eliminating redundant code and speeding it up, at the cost of having clauses with repeated literals and also increasing the length of some clauses in CNF.

As desired, the inference algorithm is sound and complete, meaning it only derives entailed sentences and it can derive any sentence that is entailed. Furthermore the reasoner can also deduce a tautology with no knowledge base.

7 References

[1] Norvig, P., Russel, S., (2003), *Artificial Intelligence: A Modern Approach*, Prentice Hall, Third Edition