Report Assignment #2: CNF converter

Authors: Simon Bilgeri (422852), Rui Lopes (31689) – Group 29

Python Version: 3.4.2

Code usage: python prop2cnf.py <path to labyrinth input file>, default: test.txt

In order to convert the sentences in propositional logic to CNF (Clausal Normal Form), many small auxiliary functions were written and each transformation step is performed recursively. With these functions at hand, it is possible to check whether a given sentence is an atom, negation, conjunction, disjunction, implication, equivalence or literal (atom, or negation of atom).
For conversion, 4 recursive transformations are applied. The recursion is designed similarly in all cases. If the sentence is a literal, it is returned. Otherwise, the recursion goes on until the literal is reached.

1. Eliminate all equivalences with two implications ((A=>B)&(B=>A)).
2. Eliminate all implication with (~A|B)
3. Propagate all negations until only literals left (negation of atom).
    a. Negation of negation → eliminate double negation
    b. Negation of disjunction → apply de Morgan's rule
    c. Negation of conjunction → apply de Morgan's rule
4. Distribute disjunctions until no changes
    a. If left, or both sentence(s) conjunction → (Sentence1[1]|Sentence2)&(Sentence1[2]|Sentence2)
    b. If right sentence conjunction → (Sentence2[1]|Sentence1)&(Sentence2[2]|Sentence1)

The resulting sentence is now in CNF. Nevertheless, it is in tree form and the clauses have to be extracted. This is done in the following way:

1. If sentence is a literal add it directly to the CNF list.
2. If sentence is a conjunction split it and add it to temporary CNF list.
3. If sentence is a disjunction, merge it (get rid of 'or'), and flatten it (get rid of brackets) and add it to the CNF list.

With the CNF list of lists (clauses) it is now easier to perform simplifications. The following 4 simplifications are performed.

1. If a clause contains identical elements in its disjunction, these elements are removed. [B|B]=B
2. If a clause contains a tautology, e.g. [B|~B], this clause is removed.
3. Redundant, identical clauses are removed. E.g. [[A],[A]]=[[A]]
4. If a clause is a subset of another clause, the other clause is removed. [[A],[A,B]]=[A]

Furthermore, for better readability, the function *show_nice_format(sentence)* converts the input format in a more convenient way.

In the menu, it is possible to show all input sentences and to choose only one sentence with step by step explanation of the conversion procedure. All intermediate steps are shown in the more convenient format. The second option converts all sentences, simplifies the resulting knowledgebase and writes in the file "CNF.txt".

In order to test all transformation and simplification steps developed, the test sentences in “test.txt” are applied.

The first five sentences test if the written auxiliary functions work correctly. (0. atom; 1. negation, literal; 2. equivalence, implication; 3. disjunction; 4. conjunction)

Sentence 2(S2) also tests the first two transformation rules (Equivalence and implication elimination)

S5 checks if the elimination of the double negation works

S6 checks if the negation of a conjunction is correctly converted to a disjunction

S7 checks if the negation of a disjunction is correctly converted to a conjunction

S8 tests the distribution of disjunction

S9 and S10 check if the extraction of the clauses works correctly. (no redundant brackets in CNF list)

S11 checks the first simplification rule (get rid of identical literals in clause)

S12 checks the second simplification rule (get rid of clauses with a tautology)

S13 tests if redundant clauses are removed

S14 tests if a clause is removed, if another clause is a subset of it.

S15 contains all transformation rules in one sentence

S16-S19 are the given test sentences in the assignment

S20 is a very large sentence to test the capabilities of the converter and simplifications

Taken all test sentence together, the Knowledge Base(KB) is not satisfiable (e.g. A&~A). Nevertheless, the test sentences shall only validate the written algorithm.

All in all, it can be stated, that the resulting KB can be considerably reduced following a few simplification rules. Anyway, conversion from propositional logic to CNF can lead to an exponential explosion of resulting clauses.