Artificial Intelligence and Decision Systems (IASD)
Mini-projects, 2017/2018

Assignment #2

Version 1.1 (28-Nov-2017)

Propositional logic reasoner

1 Introduction

A propositional logic reasoner based on the Resolution principle consists of the following key elements:

1. a program to convert logical sentences in propositional logic into the clausal normal form (CNF), and
2. a resolution-based theorem prover for propositional logic, assuming a CNF knowledge base.

The goal of this mini-project is to create, in Python, these two elements of the resolution-based propositional logic reasoner.

2 CNF converter

The main function of the converter takes as input sentences, and returns lists of disjunctions (which are lists themselves). The input sentence is represented as a syntactic tree, using tuples, according to the following rules: a *sentence* is one of

atom	$\longrightarrow$	*string*
negation	$\longrightarrow$	`('not',` *sentence* `)`
conjunction	$\longrightarrow$	`('and',` *sentence* , *sentence* `)`
disjunction	$\longrightarrow$	`('or',` *sentence* , *sentence* `)`
implication	$\longrightarrow$	`('=>',` *sentence* , *sentence* `)`
equivalence	$\longrightarrow$	`('<=>',` *sentence* , *sentence* `)`

The output should be a sequence of disjunctions, where each one of them is a list of literals; each literal can be a symbol (*i.e.,* a string) or a negation of a symbol (as above).

Suggestions:

- Write a set of small auxiliary functions to test if a given sentence is an atom, a negation, a conjunction, etc.
- Work recursively, and test for the aplicability of each transformation rule for each subsentence in the recursion.
- Consider not only the transformation rules, but also simplification rules.

3 Theorem prover

Given a knowledge base KB of facts, and a sentence α to prove, the program should return `True` or `False` depending on whether α can be proved given KB, *i.e.,* $KB \vdash \alpha$. The reasoner should use the **resolution** inference method and the **factoring** rules, together with the **unit preference** strategy.

4 Assignment goals

1. Develop Python (version 3) code to implement the propositional logic reasoner described above. The code must include two programs, one to perform the conversion to CNF (section 2), and one to perform the theorem proving (section 3). These two programs should be named `convert.py` and `prove.py`.

   Note: All code should be adequately commented.

2. The CNF convertor (`convert.py`) should read from `stdin` sentences, one per line, and output to `stdout` disjunctions, one per line. Example usage:

   ```
   python3 convert.py < sentences.txt
   ```

   If `sentences.txt` contains

   ```
   ('<=>', 'A', 'B')
   'A'
   ```

   The output should be

   ```
   [('not', 'A'), 'B']
   [('not', 'B'), 'A']
   'A'
   ```

Suggestion: Use the builtin Python function `eval()` to convert a string into a Python object. This way you can trivially parse the input file, line by line, into Python data structures.

3. The theorem prover (`prove.py`) should read from `stdin` CNF clauses, one per line, using the same format of the CNF convertor output. It is assumed that the negated conclusion $\neg\alpha$ was already converted to CNF and included in the input. The output is a single line, to `stdout`, containing either `True` or `False`, as described in section 3. Example usage:

   ```
   python3 prover.py < cnf.txt
   ```

   Considering an input file `cnf.txt` containing

   ```
   [('not', 'A'), 'B']
   [('not', 'B'), 'A']
   'A'
   'B'
   ```

   The output should be

   ```
   False
   ```

4. Both programs should work in tandem, that is, the output of `convert.py` can be directly fed into `prove.py` using shell pipes. For instance, considering $KB = \{A \Leftrightarrow B, A\}$ and $\alpha \equiv B$; to prove $KB \vdash \alpha$, one can store this problem into a file named `problem.txt` containing

   ```
   ('<=>', 'A', 'B')
   'A'
   ('not', 'B')
   ```

   (note the negated conclusion) and execute

   ```
   python3 convert.py < problem.txt | python3 prover.py
   ```

   The output should be

   ```
   True
   ```

5. The deliverable of this Lab Assignment consists of a ZIP file containing:

   - the Python source code, including the two programs mentioned above, and
   - the short report (in **PDF** format) with no more than 2 pages.

Submission: electronic submission via *fenix*[1].
Deadline: 23h59 of 22-Dec-2017
(Projects submitted after the deadline will not be considered for evaluation.)

[1]Go to *Student* > *Submit* > *Projects* after login.