CS262 Logic and Verification: Coursework assignment

Resolution proof system in Prolog

Hand in via Tabula by 12 noon on Monday 27th April, 2020

1 The rules

This assignment is worth 15% of the module overall. The absolute number of points that can be achieved is 50. The assignment is stated in a quite concise way below, but you will realize that it comprises a relatively large chunk of work. It is your responsibility to divide this work up into reasonable steps, and to plan ahead what you will do when. To complete the assignment successfully, you will combine many of the concepts we learned in lecture and put them together in Prolog. Moreover, you will have to do some independent reading. Specifically, the assignment is Exercise 3.3.3 from Fitting's book mentioned in the literature section of the module's Moodle website, and the book provides a step-by-step guide to coming up with a solution, which you will see when reading the previous sections. The book is available online via the 'Reading list' link provided on the Moodle website.

You will have the opportunity to visit the lab sessions in weeks 8-10 of term 2, where tutors will be present and ready to answer your questions that may come up during solving the assignment. They will also take time to revisit and explain some of the fundamental concepts (concerning resolution and Prolog) that you will need to solve the assignment. There will also be additional Prolog problems provided that you may work on during the lab sessions to prepare you for this coursework and practice your Prolog skills.

This is an individual assignment and you should therefore complete it on your own.

2 The assignment

Write a resolution theorem prover in Prolog, and test it on the ten propositional formulas given below. [40+10 points]

Use the tableaux theorem prover specified in Fitting's book as a starting point. Your program should be able to handle negation ($\neg$), and all primary ($\wedge, \vee, \rightarrow, \leftarrow, \uparrow, \downarrow, \not\rightarrow, \not\leftarrow$) and secondary ($\equiv, \not\equiv$) connectives in an input formula.

3 Step-by-step guide

As a first step, write a Prolog program that transforms a given propositional formula into conjunctive normal form. In this, follow the example program given in Section 2.9 of Fitting's book, which shows a program to transform a propositional formula into *disjunctive* normal form.

In your program, you need to define the unary operator `neg` and the binary operators `and`, `or`, `imp`, `revimp`, `uparrow`, `downarrow`, `notimp`, and `notrevimp`, which denote the eight aforementioned primary connectives (see Table 2.1 in the book). Also reuse the predicates `conjunctive/1`, `disjunctive/1` that recognize α- and β-formulas, and the predicates `components/3` and `component/2`

that split formulas into their components (see Table 2.2 in the book). The interface of this part of the program should be a predicate `clauseform(X,Y)`, where `Y` is the conjunctive normal form of the formula `X`. For example, for the query `?- clauseform(a imp b, Y).`, the result should be `Y=[[neg a, b]]`, and for the query `?- clauseform(neg(a uparrow b), Y)`, the result should be `Y=[[a],[b]]`.

Then augment your program so that it can also handle the secondary binary operators `equiv` and `notequiv` (see Table 2.1 in the book). Finally, test your program on larger formulas containing all of the operators mentioned before.

As a second step, implement the atomic resolution rule using predicates `resolutionstep/2` and `resolution/2` (analogous to `singlestep/2` and `expand/2`, respectively), and a check for a closed resolution expansion (explained in Section 3.3 of the book), and put these together, following the implementation described in Section 3.2, and the remarks at the end of Section 3.3. The outermost interface of your program should be a predicate `test/1`, which takes a propositional formula as input, and prints 'YES' if it has a proposition proof, and 'NO' otherwise.

You may also want to think about ways of making your program more efficient, by eliminating duplicate variables and clauses early on in the process.

4 The test data

Use the resolution proof system to decide which of the following are theorems and which are not.

```
1. ((x imp y) and x) imp y
2. (neg x imp y) imp (neg (x notimp y) imp y)
3. ((x imp y) and (y imp z)) imp (neg neg z or neg x)
4. (x imp (y imp z)) equiv ((x imp y) imp z)
5. (x notequiv y) equiv (y notequiv x)
6. (x notequiv (y notequiv z)) equiv ((x notequiv y) notequiv z)
7. (neg x downarrow neg y) revimp neg (x uparrow y)
8. (neg x revimp neg y) and ((z notrevimp u) or (u uparrow neg v))
9. ((x or y) imp (neg y notrevimp z)) or (neg neg (z equiv x) notrevimp y)
10. (neg (z notrevimp y) revimp x) imp ((x or w) imp ((y imp z) or w))
```

5 Handing in your assignment

You should upload your Prolog program to Tabula in a single plain text file called `resolution.pl`. **No other file formats other than plain text files are allowed.** The beginning of the file should contain a comment section enclosed by `/* ... */`, which shows the result of running your program on each of the ten propositional formulas shown above. Each result should appear on a new line, starting with the formula number, followed by `YES` if your program found a resolution proof, and `NO` if your program did not find a resolution proof. **The names and arities of the predicates in your program code must be exactly as specified above in Section 3.**

Here is how a submission file may look like:

```
/*
1. NO
2. NO
3. YES
4. YES
```

```
5. NO
6. YES
7. YES
8. NO
9. YES
10. NO
*/

... % the actual program code
```