CS2002 Computer Systems: Practical 2: Logic

Ian Gent

In this practical you will implement a program to print truth tables for logical formulas, and use that system to prove some laws of Boolean algebra and solve some logic puzzles. This practical is worth 25% of the continuous assessment portion of the module and is due 21.00 on Tuesday Feb 23.

1. **Programming Truth Tables**

   Write a program `ttable` in C to compute and print all rows of a truth table for a propositional formula. The exact format for printing is up to you but the format should be easy for humans to read and discover the partial result for any subformula of the input formula. It should also have in each row the values of the input variables and have as the last number in each row the result for the whole formula.

   `ttable` should take two arguments: the first is the number of propositional variables in the formula, and the second is a string representing the formula. The input string for the formula should be in the syntax detailed later. For example, the following command at a unix prompt:

```
./ttable 3 "(-((a#c)&1)>-(b=(0|-a))))"
```

   might print the following to the standard output:

```
a b c : (-((a#c)&1)>-(b=(0|-a)))) : Result
==========================================
0 0 0 :  1   0  01 11  0 011       :   1
0 0 1 :  0   1  11 11  0 011       :   1
0 1 0 :  1   0  01 00  1 011       :   0
0 1 1 :  0   1  11 10  1 011       :   1
1 0 0 :  0   1  11 10  1 000       :   1
1 0 1 :  1   0  01 00  1 000       :   0
1 1 0 :  0   1  11 11  0 000       :   1
1 1 1 :  1   0  01 11  0 000       :   1
```

   In the above example, note that the value of each subformula is aligned vertically with the propositional symbol for the subformula. This is one of several ways you can make it easy to find the truth value of subformulas.

   - A formula is one of: an atomic formula, a negated formula, or a compound formula.
   - An atomic formula is either a constant or a Boolean variable.
   - A constant is either 0 or 1, and has the constant value 0 or 1 independent of the values of any variables in the problem.
   - A Boolean variable is one of the 26 lower case letters of the alphabet a,b,c,... z. Its value is the value of the appropriate variable in the current row.
   - A negated formula starts with a - (minus) and is followed by a formula. Its value is the negation of the value of following formula.

- A compound formula starts with a left parenthesis (, followed by a formula, a compound logical operator, another formula, and finally a right parenthesis). The possible compound logical operators are `|` for or, `&` for and, `#` for exclusive or, > for implication, and = for equivalence. The value of a compound formula is the value of the logical operator applied to the two subformulas. For example the formula `(1>0)` is a compound formula with the value 0, since $1 \to 0$ is 0.

HINT: The syntax of the formula strings is designed to be very easy to parse. You might find not need to construct an explicit parse tree at all in your program. If you are struggling, you might try starting with formula strings with just one compound operator (e.g. `|`), and non nested formulas.

Here are some notes on what is required or not of your program.

- The program must be written in C (C99 version). It should not use any library except the the C standard library.
- The executable should be called ttable. You should provide a makefile that will produce file ttable and any other necessary files.
- Your program IS allowed to assume that the values of the chars 'a' to 'z' are consecutive and in that order. Although this is not strictly necessary in the C standard and is implementation-defined, this is true for most modern compilers/architectures.
- You program should work with between 1 and 26 Boolean variables but need not work with more variables. Note that $2^{26} \approx 67,000,000$, so your program may take a long time if given 26 variables, but it should work correctly.[1]
- You can assume that the input formula string is no longer than 1000 characters, although if you need this assumption you should check the input and deal appropriately with an error.
- The number of variables used for any problem are defined by the first argument n: the variables used are the first n letters of the alphabet, even if not all those variables appear in the formula.
- Ill formed formulas should be detected and appropriate error messages given.
- You are not required to accept whitespace in the input string, but you are allowed to accept it and ignore it.

2. **Laws of Boolean Algebra**

   Use your program to show that De Morgan's Laws and the Distributivity Laws of Boolean algebra are correct. Explain in your report how you used truth tables and your program to do this.

3. **Encoding and Solving Logic Puzzles**

   In this part you are to encode puzzles into propositional logic. Then use your truth table program to find out the answer to the puzzle. You must include statements of what each propositional variable represents. If necessary also clearly state any additional assumptions you are making and how these are encoded into logic. In all cases if there is no solution, or multiple solutions, state this and relate to your output.

   (a) Ian plays a game with Chris. In the game a coin is tossed and can come down heads or tails. If the coin is heads then Chris wins. If the coin is tails then Ian loses. What is the outcome of this game?

   (b) There are three people of different ages, called Ian, Steve and Chris. Either Chris or Steve is the oldest. Either Ian is the oldest or Chris is the youngest. Who is the oldest, who is in the middle, and who is the youngest?

   (c) Five people (Ann, Barbara, Charles, Deborah and Eleanor) might have attended a dinner together, but perhaps not all of them went. We know the following facts:

   - Either D or C or both attended the dinner.

[1]My implementation took about 5 minutes for an example with 26 variables: your code might run faster or slower than that.

- Either B or E but not both attended.
- If A attended, then so did B.
- E attended if and only if D attended.
- If C attended, then both A and D attended.

Who (if anybody) attended the dinner?

4. **Extensions**

   If you wish, you can extend your truth table program in various ways. For any extensions you implement, they should all run from the same program (either ttable or ttable2 if you wish to keep versions separate) controlled by command line flags and with default behaviour being the same as ttable. You may have your own ideas, but here are some suggestions:

   - Report the status of a formula as to which category(ies) it falls in: valid (the same as tautology), invalid, unsatisfiable, satisfiable.
   - Allow two input strings and print out only the lines where the first one is true. This is a bit like using the first string as a database and the second as a query on that database.
   - Allow two input strings and report whether or not they are equivalent
   - Turn your program into a logic solver - instead of printing all lines it prints nothing until a true line is found and then reports it - or states that no true line exists.

   Alternatively you could solve more complex puzzles like the following puzzle by Raymond Smullyan, or find others yourself to encode and solve.

   > "I placed three boxes on the table. I explained that one box contained a red card, one a black card, and the other contained a prize but no card. Sentences were written on all three lids, and I explained that a true sentence was written on the box with the red card, a false sentence on the box with the black card, and that the sentence on the box with the prize could be true or false. Here are the sentences.
   > Box 1: This box contains a prize.
   > Box 2: The sentence on Box 1 is true.
   > Box 3: Box 2 contains a black card.
   > Which box contains the prize?"

Submission

Submit a zip file containing your report as a pdf and code. The report should contain a discussion of your implementation including testing, your encoding of puzzles into propositional logic, and your truth table program allowed you to solve them. Your code directory should include a makefile which will compile your program to produce the executable ttable.

Rubric

There is no fixed weighting between the different parts of the practical. Your submission will be marked as a whole according to the standard mark descriptors published in the Student handbook at

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html`

You should submit your work on MMS by the deadline. The standard lateness penalties apply to coursework submitted late, as indicated at

`https://studres.cs.st-andrews.ac.uk/Library/Handbook/academic.html#lateness-penalties`

I would remind you to ensure you are following the relevant guidelines on good academic practice as outlined at:

`https://studres.cs.st-andrews.ac.uk/Library/Handbook/academic.html#Good_Academic_Practice`