CS2002 Computer Systems:
Practical W05: Logic

Ian Gent, plus thanks to Steve Linton

Deadline: **Wednesday, February 14, 2024, 9pm**

MMS deadlines are definitive and subject to change, so please check

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 30% of the continuous assessment portion of the module.

Part 1: Programming Truth Tables

Write a program `ttable` in C to compute and print all rows of a truth table for a propositional formula, given in Reverse Polish Notation (RPN) which was explained in CS2001 last semester. The output format should be exactly as explained below.

`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 'ac#1&-a-0|b=->'
```

should print the following to the standard output:

```
a b c : ac#1&-a-0|b=-> : Result
==============================
0 0 0 :   0 01 1 1 011 :   1
0 0 1 :   1 10 1 1 011 :   1
0 1 0 :   0 01 1 1 100 :   0
0 1 1 :   1 10 1 1 101 :   1
1 0 0 :   1 10 0 0 101 :   1
1 0 1 :   0 01 0 0 100 :   0
1 1 0 :   1 10 0 0 011 :   1
1 1 1 :   0 01 0 0 011 :   1
```

In the above example, note that the value of each subformula is aligned vertically with the propositional symbol for the subformula. The formula is the RPN version of

`-((a#c)&1) > -((-a | 0) = b)`

The output of ttable should be formatted as in the above example, including the header line and spacing. Values should be printed under each logical operator (−, `|`, `&`, `#`, > or =) but not under variables or constants in the formula.

The following describes the syntax and meaning of formulas for this practical.

- A formula is one of: an atomic formula, a negated formula, or a compound formula. Compound formulas are always binary, i.e. have exactly two arguments.
- An atomic formula is either a constant or a Boolean variable.
- A constant is either 0 or 1, and has the constant value 0 (for false) or 1 (for true) 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 is a formula F followed by a $-$ (minus). Its value is the negation of the value of F.
- A compound formula is precisely two formulas followed by a compound logical operator. 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 `10>` is a compound formula with the value 0, since $1 \rightarrow 0$ is 0.

Here are some notes on the requirements of your program.

- The program must be written in C. 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 an executable file `ttable` correctly when run on lab machines.
- 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.
- 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 should not accept with fewer than 1 or more than 26 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]
- Your program should accept input strings up to 1000 characters (inclusive) but reject longer strings. The input formula should not have any characters not mentioned above, so e.g. should not have whitespace.
- If a truth table is successfully constructed, your program should return the zero value. If your program detects an invalid formula as input, i.e. not correct RPN formulas and/or not obeying the constraints listed above, it should return a the value 1 and print out what the error was.

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

Part 2: Using Truth Tables to Solve Logical Problems

In this part you are to use your program to solve logical problems. First, you will be asked to use the program to explain a version of De Morgan's law. In the remaining questions, you will have to encode English 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.

1. Use your program to show that the following variant of De Morgan's Law is correct:

$$\neg((A \vee B) \vee (C \vee D)) = ((\neg A \wedge \neg B) \wedge (\neg C \wedge \neg D))$$

   Explain in your report how you used truth tables and your program to do this.

2. 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?

3. 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 Deborah or Charles or both attended the dinner.
   - Either Barbara or Eleanor but not both attended.
   - If Ann attended, then so did Barbara.
   - Eleanor attended if and only if Deborah attended.
   - If Charles attended, then both Ann and Deborah attended.

   Who (if anybody) attended the dinner?

4. The following puzzle is by Raymond Smullyan, from '*The Riddle of Scheherazade*', 1998.

   "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?"

Testing and StacsCheck

Some StacsCheck tests for this practical will be released in the next few days. These are not intended to make sure that your code is likely to be correct but will allow you to check some basic functionality. You should do additional tests to make sure your program is running correctly, and report on these in your submission.

As part of the marking process we may run additional StacsCheck tests which we have not provided to you in advance.

What to hand-in:

Submit the report and source code via MMS. The report should be in PDF format and contain a discussion of your implementation including testing, your encoding of puzzles into propositional logic, and evidence of how 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. The submission must be zipped into a single archive for MMS.

INCLUDE your source code! Work without source code, however good, is likely to attract a very much lower mark than it would otherwise get.

INCLUDE your report! Work without a report, however good, is likely to attract a very much lower mark than it would otherwise get!

Marking:

Your submission will be marked as a whole according to the standard mark descriptors for CS2002 published at:

`https://studres.cs.st-andrews.ac.uk/CS2002/0-General/descriptors.pdf`

Since this practical has a mixture of coding and logic, we will provide some clarity on what "almost all" functionality means those descriptors. We interpret this to mean Part 1 mainly complete in functionality and puzzles 1 to 3 done correctly in Part 2, and a provide a report. "Full functionality" means full functionality in Part 1 and all puzzles completed correctly in Part 2.

Lateness and GAP:

Late work will be penalised according to Scheme A, 1 mark per 24 hours or part thereof. See:

`http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`

Remember that the University policy on Good Academic Practice applies, see:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`