Assignment 1, 2019

Released: 27 August. Deadline: 15 September at 23:00

Purpose

To improve your understanding of propositional logic, first-order predicate logic, mechanized reasoning, and discrete mathematics. To develop skills in analysis and formal reasoning about complex concepts, and to practise writing down formal arguments with clarity.

Challenge 1

Consider the four first-order predicate formulas

$$
\begin{aligned}
F &= \forall x\ (\neg P(x,x)) \\
G &= \forall x\ \forall y\ \forall z\ (P(x,y) \wedge P(y,z) \Rightarrow P(x,z)) \\
G' &= \forall x\ \forall y\ \forall z\ (P(x,y) \wedge P(y,z) \Rightarrow \neg P(x,z)) \\
H &= \forall x\ \forall y\ (P(x,y) \Rightarrow \neg P(y,x))
\end{aligned}
$$

a. Show that $F \vee G \vee G' \vee H$ is not valid.

b. Show that $F \wedge G' \wedge H$ is satisfiable.

c. Show that $(F \wedge G) \Rightarrow H$ is valid.

Challenge 2

For this challenge use the following predicates:

- $F(x)$, which stands for "the force is with x"
- $J(x)$, which stands for "x is a Jedi master"
- $E(x,y)$, which stands for "x exceeds y"
- $P(x,y)$, which stands for "x is a pupil of y"
- $V(x)$, which stands for "x is venerated"

and use the constant 'a' to denote Yoda.

a. For each of the statements $S_1, \ldots S_4$, express it *as a formula in first-order predicate logic* (not clausal form):

S_1: "The force is with every Jedi master."

S_2: "Yoda exceeds some Jedi master."

S_3: "A Jedi master is venerated if (1) the force is with him/her *and* (2) all his/her pupils exceed him/her."

S_4: "Every Jedi master without pupils is venerated."

(Be careful to get these translations right, including S_3; most of the following depends on this.)

b. Translate S_1–S_3 to clausal form.

c. Translate *the negation of* S_4 to clausal form.

d. Give a proof by resolution to show that S_4 follows from the other statements.

Challenge 3

Let $\mathcal{B} = \{0, 1\}$ be the set of Boolean values, so that $\mathcal{B}^n$ is the set of n-tuples of Boolean values (or Boolean *vectors*). We use a bold variable $\boldsymbol{b}$ to denote a Boolean vector of n Boolean values $(b_1, b_2, \ldots, b_n)$

An n-dimensional Boolean *vector function* $\boldsymbol{f} : \mathcal{B}^n \to \mathcal{B}^n$ is *reversible* iff for every Boolean vector $\boldsymbol{y} \in \mathcal{B}^n$ there is exactly one vector $\boldsymbol{x} \in \mathcal{B}^n$ such that $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{y}$. Intuitively, the function is reversible iff we can uniquely determine the input vector from an output vector, allowing us to reliably 'reverse' the evaluation of the function. For example, $\boldsymbol{f}(b_1, b_2) = (b_2, b_1)$ is a 2-dimensional reversible Boolean vector function, and $\boldsymbol{g}(b_1, b_2) = (b_1 \wedge b_2, b_1 \vee b_2)$ is a 2-dimensional Boolean vector function that is not reversible (because, for example, $(0, 1) = \boldsymbol{g}(0, 1) = \boldsymbol{g}(1, 0)$).

(This kind of logic function finds important applications in the theory of quantum computation, where the laws of quantum mechanics constrain computations to proceed reversibly.)

a. Consider the Boolean vector function $\boldsymbol{f_a}(b_1, b_2) = (\neg b_1, b_1 \oplus b_2)$. Show that this function is reversible.

b. Consider the Boolean vector function $\boldsymbol{f_b}(b_1, b_2, b_3) = (b_1 \oplus b_2, b_2 \oplus b_3, b_3 \oplus b_1)$. Show that this function is *not* reversible.

c. Is the Boolean vector function $\boldsymbol{f_c}(b_1, b_2, b_3) = (b_1 \oplus b_3, \neg b_3, b_1 \oplus b_2)$ reversible? Justify your answer.

d. Give a formula, in terms of n, for the fraction of distinct n-dimensional Boolean vector functions that are reversible.

   Note that two n-dimensional Boolean vector functions are considered distinct iff they produce different output vectors for some input vector. For example, we should not count $\boldsymbol{f}(b_1) = (\neg b_1)$ and $\boldsymbol{g}(b_1) = (b_1 \oplus \mathsf{t})$ as two distinct functions—despite syntactic differences, $\boldsymbol{f}$ and $\boldsymbol{g}$ produce matching outputs for all inputs. Here we are only interested in counting functions with differing behaviour.

Challenge 4

Find ways to avoid unnecessarily complex propositional formulas:

a. Give the shortest possible formula that is equivalent to $(P \Rightarrow Q) \Rightarrow \neg Q$.
b. Express $P \Rightarrow ((P \wedge Q) \Rightarrow R)$ using no connectives other than $\Rightarrow$.
c. Express $(P \Rightarrow (Q \vee R)) \wedge (P \Leftrightarrow Q)$ using no connectives other than $\Leftrightarrow$.

The answers to this challenge must be submitted through Grok. For each question, you are to submit the answer as a Haskell expression of type `Exp`, where `Exp` is the type of structure trees used for propositional formulas in the second half of Worksheet 1 (see Grok). For variable names (of type Char), use `'P'`, `'Q'`, and `'R'`, that is, use upper case.

Challenge 5

Consider the following light switching game. We begin with an n by n grid of square buttons, numbered 1–n^2, initially dark, as shown below (left) for $n = 4$. Pressing any button will cause that button to light up, along with its immediate neighbours above, below, and to its left and right in the grid. For example, the result of pressing button 11 is depicted below (middle). Subsequent button presses will continue to light up nearby buttons, or switch them back off if they are already alight. For example, the result of pressing button 15 (after button 11) is shown below (right). Note that because buttons 15 and 11 were both alight, they became dark again. Note also that the grid does not 'wrap around': button 15 has no effect on button 3. The aim of the puzzle, given a particular light configuration, is to find a sequence of button-presses that will produce this configuration. To help you familiarise yourself with this puzzle, we provide a grid for the $n = 4$ case at `https://comp30026.far.in.net/puzzle`.

Your task is to write a Haskell program capable of solving this puzzle. Given a grid size n and a light configuration as input, your program must output a sequence of buttons which, when pressed in order, will produce the desired configuration.

The answer to this challenge must be submitted through Grok, in the form of a function `lights` of type `Int -> [Int] -> [Int]` The first input parameter provides the dimensionality of the puzzle grid, n, and will always be a positive integer. The second input parameter represents the buttons to be switched on in the desired configuration (any buttons not listed are to be switched off). The return value should be a list representing the sequence of buttons to press in your solution to the puzzle, or, in case the configuration is impossible to achieve, the return value should be an empty list. For example, the input/output pair below corresponds to a 4 by 4 puzzle in which the desired configuration is the one above (right), and to one possible solution to this puzzle:

```
> lights 4 [7, 10, 12, 14, 16]
[11, 15]
```

Hint 1: Our suggested approach is for you to model the puzzle as a propositional satisfiability problem. In this way, the task becomes a simple translation from a puzzle description to a propositional formula, and you can use tools like those you have constructed in assessed worksheets 1 and 2 to solve the formula itself (we provide similar tools on Grok). The code required for this approach is minor compared to the code required to implement a puzzle-specific search algorithm.

Hint 2: There are many ways to approach this modelling task. To get you started on the right track, consider the following questions (assume $n = 4$): (1) What light configuration results from the sequence `[15, 11]`? How does this compare to the above example? (2) What light configuration results from the sequence `[6, 1, 6]`? How about `[6, 1, 6, 6]`? Is it ever useful to press a button more than once? (3) Could you determine the final state of a single button by looking at a long sequence of button presses, without simulating the whole sequence?

Challenge 6

Consider a single-digit display which will be used to show any decimal digit, or a dash:

$$0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ -$$

Arrays of such displays are used to display numbers in digital clocks, washing machines and other devices. A *seven-segment* display is made up by seven LEDs (labelled a–g), as shown below on the right. For example, to show the digit 3, all display segments except b and e are lit up.

Each of the seven display segments a–g can be considered a propositional function of four propositional variables W, X, Y and Z, with input $WXYZ$ specifying the desired decimal digit in binary notation (or 1010 for a dash). For example, input 3 is given by $WXYZ = 0011$ (that is, W and X are false and Y and Z are true). For that input combination, the outputs b and e should be 0 (False) while the rest should be 1 (True). Given this, the truth tables for all the functions a–g are as shown in the table below.

W	X	Y	Z	a	b	c	d	e	f	g
0	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0	1
0	0	1	1	1	0	1	1	0	1	1
0	1	0	0	0	1	1	1	0	1	0
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	0	1	1	1	1
0	1	1	1	1	0	1	0	0	1	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	1	0	0	0
1	0	1	1	*	*	*	*	*	*	*
1	1	0	0	*	*	*	*	*	*	*
1	1	0	1	*	*	*	*	*	*	*
1	1	1	0	*	*	*	*	*	*	*
1	1	1	1	*	*	*	*	*	*	*
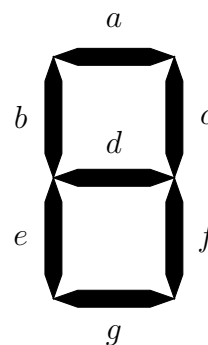
An asterisk (*) in the table marks an output that we don't care about. It is assumed that the last five input combinations will never happen, so we are free to let the corresponding outputs be whatever is convenient.

The single-digit display must be implemented with logic circuitry. Here we assume that only three types of logic gates are available. An *and-gate* takes two inputs and produces, as output, the conjunction ($\wedge$) of the inputs. Similarly, an *or-gate* implements disjunction ($\vee$). Finally, an *inverter* takes a single input and negates it.

The task here is to design a circuit for all of a–g using as few gates as possible. We can specify the circuit by writing down the Boolean equations for each of the outputs a–g. For example, we can define $f = X \vee \neg Y \vee Z$, which shows that we can implement f with as little as three gates. Since we want to use as few gates as possible, it is important to look for cases where outputs can be *shared*, or reused. For example, if we have already implemented b then we could define $f = b \vee Z$, using just one gate. Note that for any sub-circuit that you want to share (that is, you want to direct its output to different places), the output must have a name. You can define as many "helper" functions as you please, to create the smallest possible solution. For example, suppose you have implemented some function using

$\neg(X \vee Y)$ as a sub-circuit, and now you realise that there is another function that could also use that sub-circuit. Rather than duplicate the sub-circuit, you want the two functions to share, so you name the output (say, u). Defining $u = \neg(X \vee Y)$ and letting the two functions refer to u instead of $\neg(X \vee Y)$ corresponds exactly to the sharing of the circuit.

The answer to this challenge must be submitted through Grok. Submit a text file consisting of one line per definition. This file will be tested automatically, so it is important that you follow the notational conventions exactly. We write $\neg$ as `-` and $\vee$ as `+`. We write $\wedge$ as `.`, or, simpler, we just leave it out, so that concatenation of expressions denotes their conjunction. Here is an example set of equations (for a different problem):

```
# An example of a set of equations in the correct format:
a = -Y Z + Y -Z + X -Y -Z
b = u + X (Y + Z)
c = X + -(Y Z)
d = u + X a
u = -X -Y
# u is an auxiliary function introduced to simplify b and d
```

Empty lines, and lines that start with '#', are ignored. Input variables are in upper case. Negation binds tighter than conjunction, which in turn binds tighter than disjunction. So the equation for a says that $a = (\neg Y \wedge Z) \vee (Y \wedge \neg Z) \vee (X \wedge \neg Y \wedge \neg Z)$. Note the use of a helper function u, allowing b and d to share some circuitry. Also note that we do not allow any feedback loops in the circuit. In the example above, d depends on a, so a is not allowed to depend, directly or indirectly, on d (and indeed it does not).

Submission and assessment

Some of the problems are harder than others. All should be solved and submitted by students individually. Your solution will count for 12 marks out of 100 for the subject. Each question is worth 2 marks. Marks are primarily allocated for correctness, but elegance and how clearly you communicate your thinking will also be taken into account. For Challenge 6, there will be 1 mark for a correct solution. After that, your mark depends on which quartile your solution sits in, when we order all solutions by increasing number of gates. If it falls in the best quartile (that is, within the 25% of solutions that use the fewest gates), we add 1 mark. If you are in the next quartile, we instead add 0.5 marks. Otherwise no marks are added to the mark for correctness.

The deadline is 15 September at 23:00. Late submission will be possible, but a late submission penalty will apply: a flagfall of 1 mark, and then 1 mark per 12 hours late.

For challenges 1–3, submit a PDF document via the LMS. This document should be no more than 2 MB in size. If you produce an MS Word document, it must be exported and submitted as PDF, and satisfy the space limit of 2 MB. We also accept *neat* hand-written submissions, but these must be scanned and provided as PDF, and again, they must respect the size limit. If you scan your document, make sure you set the resolution so that the generated document is no more than 2 MB. The LMS site, from which you downloaded this document, has advice to help you satisfy the 2 MB requirement while maintaining readability.

Being neat is easier if you type-set your answers, but not all typesetting software does a good job of presenting mathematical formulas. The best tool for this is LaTeX, which is worth learning if you expect that you will later have to produce large technical documents. Admittedly, diagrams are tedious to do with LaTeX, even when using sophisticated packages

such as Tikz. You could, of course, mix typeset text with hand-drawn diagrams. In case you want to use the assignment to get some LATEX practice, we will leave the source for this document in the content area where you find the PDF version.

For challenge 4–6, submit files on Grok. The required format for submitted solutions will be clear when you open the Grok modules, but as mentioned above, for Challenge 4 we want expressions of type `Exp`, for Challenge 5 we want a Haskell function, and for Challenge 6 we want a text file in the format specified above.

For Challenges 4 and 5 you will receive immediate feedback from some elementary tests. These merely check that your input has the correct format, so they should not be relied upon for correctness. We will test comprehensively after the deadline. For Challenge 6, we provide feedback concerning both well-formedness and correctness. Note that on Grok, "saving" your file does not mean submitting it for marking. To submit, you need to click "mark". You can submit as many times as you like. What gets marked is the last submission you have made before the deadline.

Make sure that you have enough time towards the end of the assignment to present your solutions carefully. A nice presentation is sometimes more time consuming than solving the problems. Start early; note that time you put in early usually turns out more productive than a last-minute effort. For Challenge 6 in particular, you don't want to submit "improved" solutions a few minutes before the deadline, as they may turn out to be wrong and you won't have time to fix this.

Individual work is expected, but if you get stuck, email Harald or Matt a precise description of the problem, bring up the problem at the lecture, or (our preferred option) use the LMS discussion board. The COMP30026 LMS discussion forum is both useful and appropriate for this; soliciting help from sources other than the above will be considered cheating and will lead to disciplinary action.

Harald and Matt
26 August 2019
(Challenge 6 comment corrected 8 September 2019)