

Isabelle coursework exercises

John Wickerson

Autumn term 2019

Tasks are largely independent from each other, and are arranged in roughly increasing order of difficulty. Tasks labelled (★) are expected to be quite straightforward. Tasks labelled (★★) should be manageable but may require quite a bit of thinking, and it may be necessary to consult additional sources of information, such as the Isabelle manual and Stack Overflow. Tasks labelled (★★★) are challenging and open-ended. It is not expected that many students will complete these, but partial credit will be given to partial answers. The task labelled (★★★★) is *very* hard, but again, some credit will be given to students who show they have given it some thought.

Submission process. You are expected to produce a single Isabelle theory file called `YourName.thy`. This file should contain all of the definitions and proofs for all of the tasks below that you have attempted. You are not expected to complete all tasks. You *are* expected, however, to provide detailed annotations throughout your file (in the form of `(*comments*)`) that demonstrate the extent to which you have understood the automated theorem proving process.

Task 1 (★) Prove that $2\sqrt{2}$ is irrational.

Task 2 (★★) Here are the first few *L numbers*.

$L(0)$		0
$L(1)$	•	1
$L(2)$	••	3
$L(3)$	•••	5
$L(4)$	••••	7

Encode in Isabelle the following recursive function for calculating L numbers:

$$L(n) = \begin{cases} n & \text{if } 0 \leq n \leq 1 \\ 2 + L(n - 1) & \text{if } 2 \leq n \end{cases}$$

Give a closed form for $L(n)$. Prove in Isabelle that your closed form and the recursive definition coincide for all $n \geq 0$.

Task 3 (★★) Below are the first few *pyramidal numbers*. Pyramidal number n (written $py(n)$) is the number of spheres stacked in a pyramid whose base is a square with sides of length n .

$py(0)$	0
$py(1)$	1
$py(2)$	5
$py(3)$	14
$py(4)$	30

Encode in Isabelle the following recursive function for calculating pyramidal numbers:

$$py(n) = \begin{cases} 0 & \text{if } n = 0 \\ n^2 + py(n - 1) & \text{if } 1 \leq n \end{cases}$$

Here is a closed form for pyramidal numbers:

$$py(n) = \frac{(2n + 1)(n + 1)n}{6}$$

Prove in Isabelle that the recursive definition and the closed form coincide for all $n \geq 0$.

Task 4 (★) Write a function that checks whether a circuit contains two consecutive NOT gates. Use this function to state and prove a theorem that whenever opt_{NOT} is applied to a circuit, the resultant circuit never contains two consecutive NOT gates.

Task 5 (★) State and prove a theorem that once opt_{NOT} has been applied to a circuit, applying it again has no effect. (We say that opt_{NOT} is *idempotent*.)

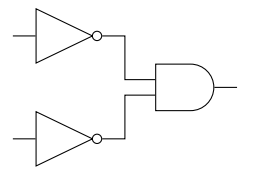
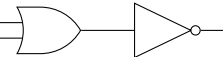
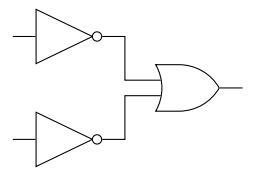

	can be replaced with	
	can be replaced with	

Table 1: Optimising a circuit according to De Morgan's laws

Task 6 (★★) Write a function called opt_{DM} that optimises a circuit using De Morgan's laws, as shown in Table 1. Prove that your optimisation is sound, using the following theorem.

theorem $simulate (opt_{DM} c) \rho = simulate c \rho$

Is your optimisation idempotent?

Task 7 (★) Prove that it is sound to apply both of these optimisations successively to any circuit.

Task 8 (★★) Let us now consider the effect of these optimisations on other properties of a circuit. The following function calculates the *area* of a circuit, by counting each NOT, OR, and AND gate as having an area of 1, and constants and inputs as taking up no area.

```

1 fun area :: circuit => nat where
2   area (NOT c) = 1 + area c
3 | area (AND c1 c2) = 1 + area c1 + area c2
4 | area (OR c1 c2) = 1 + area c1 + area c2
5 | area _ = 0

```

State and prove theorems that opt_{NOT} and opt_{DM} never *increase* the area of a circuit.

Task 9 (★★) Define a function that calculates the *delay* of a circuit. We shall say that the delay of circuit is the length of the longest path from the circuit's output to one of its inputs. Prove that opt_{NOT} and opt_{DM} never increase the delay of a circuit.

Task 10 (★★★) Define a function that performs *constant folding*. Wherever it sees a gate with TRUE or FALSE as one of its inputs, it should try to replace

the gate with `TRUE`, `FALSE`, or the other input, depending on the gate. For instance, it should replace `NOT FALSE` with `TRUE`, it should replace `AND FALSE c` with `FALSE`, and it should replace `OR c FALSE` with `c`. Prove that constant folding is sound, and that it never increases the area or delay of a circuit. Also, prove that if constant folding is applied to a circuit that has no inputs, the circuit thus obtained must be `TRUE` or `FALSE`.

Task 11 (**)** So far we have only considered circuits with no fan-out, i.e. circuits with only a single output wire. This is not very realistic. Devise a data structure that allows circuits with multiple outputs to be represented. Define a function that optimises these circuits by removing gates that do not lead, directly or indirectly, to an output. (Such gates can be considered ‘dead’, as they cannot affect the behaviour of a circuit.) Prove that removing dead gates is sound.