Isabelle coursework exercises

John Wickerson

Autumn term 2020

Tasks are largely independent from each other, and are arranged in roughly increasing order of difficulty. Tasks labelled ($\star$) are expected to be reasonably straightforward. Tasks labelled ($\star\star$) 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 ($\star\star\star$) are more ambitious still. It is not expected that students will complete all parts of all the tasks. Partial credit will be given to partial answers. If you are unable to complete a proof, partial credit will be given for explaining your thinking process in the form of `(*comments*)` in the Isabelle file.

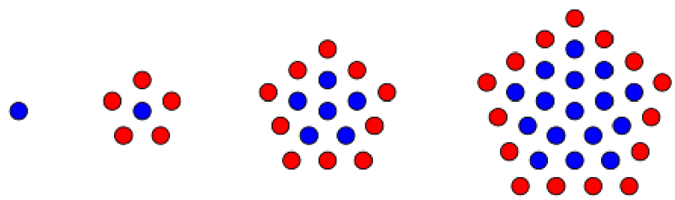
Submission process. You are expected to produce a single Isabelle theory file called `YourName.thy`. This file should contain all of the definitions and proofs for all of the tasks below that you have attempted.

Task 1 ($\star$) Prove that

$$\frac{3}{\sqrt{2}}$$

is irrational.

Task 2 ($\star\star$) Here are the first few *centred pentagonal numbers*.



1 6 16 31

Here is a recursive function for calculating centred pentagonal numbers:

$$
pent(n) \quad = \quad \begin{cases} 1 & \text{if } n = 0 \\ 5n + pent(n-1) & \text{otherwise} \end{cases}
$$

Prove that

$$
pent(n) \quad = \quad \frac{5n^2 + 5n + 2}{2}
$$

holds for all $n \geq 0$.

Task 3 ($\star\star$) The Fibonacci numbers (named after the Italian mathematician Leonardo Bonacci, c.1170–1250) are defined using the following recursive definition:

$$
\begin{aligned}
fib(0) &= 0 \\
fib(1) &= 1 \\
fib(n) &= fib(n-1) + fib(n-2) \quad \text{for all } n > 1
\end{aligned}
$$

The Lucas numbers (named after the French mathematician Édouard Lucas, 1842–1891) are defined quite similarly:

$$
\begin{aligned}
luc(0) &= 2 \\
luc(1) &= 1 \\
luc(n) &= luc(n-1) + luc(n-2) \quad \text{for all } n > 1
\end{aligned}
$$

This task is about establishing some relationships between Lucas numbers and Fibonacci numbers.

1. Prove that $luc(n) \geq fib(n)$ holds for all $n \geq 0$.
2. Prove that $luc(n+1) = fib(n) + fib(n+2)$ holds for all $n \geq 0$.

Task 4 ($\star\star\star$) This task builds on the `circuit` datatype from the worksheet.

Let us define the *leaves* of a circuit to be its constants (i.e. `TRUE` and `FALSE`) and its inputs. Here is a function that calculates the *delay* of a circuit – that is, the length of the longest path from a leaf to the circuit's output. (NB: this function was the topic of Task 9 of the 2019 coursework.)

```
fun delay :: circuit ⇒ nat where
  delay (NOT c) = 1 + delay c
| delay (AND c1 c2) = 1 + max (delay c1) (delay c2)
| delay (OR c1 c2) = 1 + max (delay c1) (delay c2)
| delay _ = 0
```

Let us define a circuit to be *balanced* if all leaves are the same distance from the output. In other words, all paths from a leaf to the output have the same length. Write a function called `is_balanced` that checks whether a given circuit is balanced. For instance,

```
is_balanced (AND (NOT TRUE) TRUE)
```

should evaluate to `False`, and

```
is_balanced (AND (NOT TRUE) (OR TRUE (INPUT 1)))
```

should evaluate to `True`.

Now write a function called `balance` that takes any circuit and produces a balanced circuit that has the same input/output behaviour. Ensure that your function is correct by proving two theorems. First, prove for any circuit c that `balance(`c`)` has the same input/output behaviour as c. Second, prove for any circuit c that `is_balanced(balance(`c`))` holds.

Task 5 ($\star\star\star$) Extend the `circuit` datatype with a new constructor that represents a 2-input `NAND` gate. The `NAND` gate is a *universal* gate, in the sense that any other gate can be represented using some combination of `NAND` gates – a fact that was first proved by Henry Sheffer in 1913.

Devise an optimisation called `transform_to_NAND` that transforms any given circuit into a `NAND`-only representation. Ensure that your optimisation is correct by proving two theorems.

1. Prove that `transform_to_NAND` never changes the behaviour of a circuit.
2. Prove that `transform_to_NAND` always produces circuits that use only `NAND` gates, inputs, and the `TRUE` constant.

Now extend the $delay$ function so that it defines `NAND` gates to have a delay of 1 (just like `AND` and `OR` gates). Devise an upper bound on the delay incurred by `transform_to_NAND`. That is, if the circuit c has delay d, find a function f such that you can prove that the delay of `transform_to_NAND(`c`)` never exceeds $f(d)$. As a hint: myself, I found that $f(d) = 2d + 1$ worked, but your definition of f may vary depending on the details of how you implement `transform_to_NAND`.