Haskell

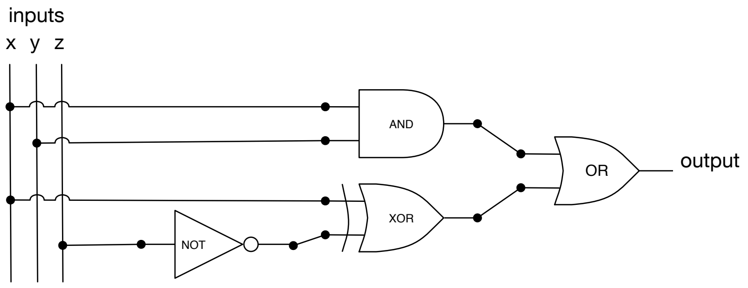
Part 1: Logical Circuits

This assignment concerns *networks of logical gates* or *circuits* for short. A circuit is a network of logical gates (AND, OR, NOT, . . .) that implement boolean logic.

This assignment considers only logical gates with one or more inputs and exactly one output. In general a circuit can consist of multiple gates whose inputs and outputs are arbitrarily connected with one another. In this assignment we only consider circuits that correspond to boolean formulas. For example, a particular circuit can be presented by the formula:

$$(x \wedge y) \vee (\neg z \oplus x)$$

Every variable presents an input of the circuit and the logical connectives represent logical gates ($\wedge$ for AND, $\vee$ for OR, $\neg$ for NOT, and $\oplus$ for XOR). The result of the formula corresponds to the output of the circuit. In short, the above formula corresponds to the circuit below:



Task 1a. Define a new datatype `Circuit` that represents a circuit (or a logical formula). A circuit is one of:

1. one of the inputs – every input is characterised by its name, a `String`;
2. a logical NOT gate applied to an underlying circuit;
3. a logical AND gate applied to two underlying circuits;
4. a logical OR gate applied to two underlying circuits; or
5. a logical XOR gate applied to two underlying circuits.

Task 1b. Use the `Circuit` datatype to define a few trivial circuits:

1. `cinput :: String -> Circuit` is the circuit that returns the input with the given name.
2. `cnot :: Circuit -> Circuit` is the circuit that returns the logical NOT of the given circuit.
3. `cand :: Circuit -> Circuit -> Circuit` is the circuit that returns the logical AND of the two given circuits.

4. `cor :: Circuit -> Circuit -> Circuit` is the circuit that returns the logical OR of the two given circuits.

5. `cxor :: Circuit -> Circuit -> Circuit` is the circuit that returns the logical XOR of the two given circuits.

Task 1c. Use the functions from Task 1b to define the above example circuit as `example :: Circuit`.

Task 1d. Write the function `candMany :: [Circuit] -> Circuit` that returns the logical AND of the given list of circuits. You may assume that the list is non-empty.

Part 2: Fun with Circuits

Task 2a. Write an instance of the `Show` type class for `Circuit` that turns a circuit into a `String` that represents the corresponding formula in a textual form. The expected textual form can best be explained with an example:

```
> example
OR(AND(x,y),XOR(NOT(z),x))
```

Note that there are no spaces in the text.

Task 2b. Our circuits contain 4 different kinds of gates (NOT, AND, OR, XOR). The last two are actually not essential. Indeed, we can express XOR in terms of the first three, and OR in terms of the first two kinds of gate. Namely:

$$\begin{aligned} x \oplus y &= (x \wedge \neg y) \vee (\neg x \wedge y) \\ x \vee y &= \neg(\neg x \wedge \neg y) \end{aligned}$$

Write the function `simplify :: Circuit -> Circuit` which eliminates all OR and XOR gates from the given circuit and returns an equivalent circuit that contains only NOT and AND gates. Make use of the above two equations.

```
> simplify example
NOT(AND(NOT(AND(x,y)),NOT(NOT(AND(NOT(AND(NOT(z),NOT(x))),NOT(AND(NOT(NOT(z)),x)))))))
```

Task 2c. Write a function `size :: Circuit -> Int` which returns the number of gates in a given circuit. The inputs are not gates and do not count.

```
> size example
4
> size (cinput "x")
0
> size (simplify example)
15
```

Task 2d. The *gate delay* of a circuit is the maximum number of gates on a path from an input of the circuit to its output. For the example the gate delay is 3 because there are 3 gates (NOT, XOR and OR) on the path between the input z and the output of the circuit. Write the function `gateDelay :: Circuit -> Int` which returns the gate delay of the given circuit.

```
> gateDelay example
3
> gateDelay (cinput "x")
0
> gateDelay (simplify example)
9
```

Task 2e. Write the function `inputs :: Circuit -> [String]` which returns the list of input names of the given circuit. The names can be in any order, but the list should not contain any duplicates.

```
> inputs (cinput "x")
["x"]
> inputs example
["x","y","z"]
```

Part 3: Simulate Circuits

Task 3a. Write the function `simulate :: Circuit -> [(String,Bool)] -> Bool` that simulates which output the given circuit returns if the inputs take the given values.

```
> simulate (cnot (cinput "x")) [("x",True)]
False
> simulate (cnot (cinput "x")) [("x",False)]
True
> simulate example [("x",True),("y",False),("z",True)]
True
```

Task 3b. Write the function `combinations :: Int -> [[Bool]]` such that `combinations` n generates the list of all possible combinations of n boolean values. **Note:** the combinations have to be ordered in a particular way. You can derive the required order from the examples below:

```
> combinations 0
[[]]
> combinations 1
[[False],[True]]
> combinations 2
[[False,False],[False,True],[True,False],[True,True]]
> combinations 3
[[False,False,False],[False,False,True],[False,True,False],[False,True,True]
,[True,False,False],[True,False,True],[True,True,False],[True,True,True]]
```

Task 3c. Write the function `tabulate :: Circuit -> IO ()` which prints the given circuit in the form of a table as illustrated below. Make use of the functions `inputs`, `simulate` and `combinations`. For the sake of simplicity you may assume that the names of inputs consist of a single character.

```
> tabulate (cinput "x")
x | output
0 | 0
1 | 1
> tabulate (cnot (cinput "x"))
x | output
0 | 1
```

```
1 | 0
> tabulate example
x y z | output
0 0 0 | 1
0 0 1 | 0
0 1 0 | 1
0 1 1 | 0
1 0 0 | 0
1 0 1 | 1
1 1 0 | 1
1 1 1 | 1
```