Gequoteerde Haskell-oefening – (Studie van) Declaratieve Talen

NAAM: RICHTING:

LOKAAL:

- Je krijgt twee uur om deze opdracht **individueel** op te lossen.
- Je raadpleegt enkel afgedrukte kopies van de slides en de cursusnota's (eventueel met handgeschreven nota's). Daarnaast mag de volgende elektronische documentatie raadplegen:
  - de beperkte library documentatie `http://zvon.org/other/haskell/Outputcomplex/index.html`, die ook op de departementale machines staat onder `///localhost/examen/DOC/Manual-haskell/Outputglobal/index.html`
  - de ghci manual van jouw Haskell installiatie; op de departementale machines staat die onder `///usr/share/doc/ghc-doc/html/libraries/base/index.html`
- In de map **1819_Gequoteerde/Haskell_Woensdag op Toledo** vind je de nodige bestanden evenals de indienmodule. Je vindt deze eveneens op **e-systant** onder **Haskell/Graded Sessions 18-19**.
- Je oplossing zet je in een bestand `Myhaskell.hs` en de eerste lijnen van dit bestand moeten je naam, studentennummer en richting bevatten.

```
-- Jan Jansen
-- r0123456
-- master cw
```

- Na twee uur, of wanneer je klaar bent, dien je het `Myhaskell.hs` bestand in via Toledo. **Let op: indienen via e-systant is niet voldoende.**

Haskell

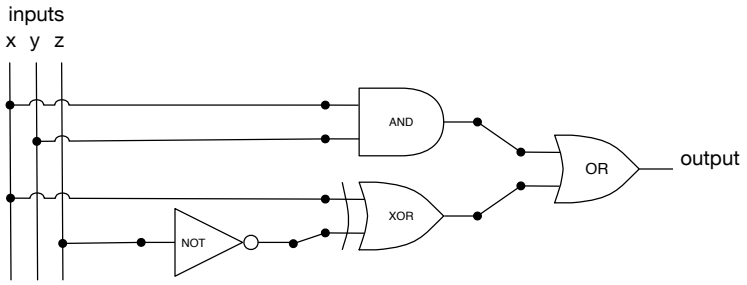
Deel 1: Logische Circuits

Deze opgave gaat over *logische schakelingen* of *circuits*. Een circuit is een samenstelling van logische poorten (AND, OR, NOT, ...) die werken volgens de booleaanse logica. In deze opgave beperken we ons tot logische poorten die één of meerdere inputs hebben en juist één output.

In het algemeen kan een circuit bestaan uit meerdere poorten waarvan de in- en uitgangen op willekeurige wijze met elkaar verbonden zijn. In deze opgave beperken we ons echter tot circuits die 1-op-1 overeenkomen met booleaanse formules. Bijvoorbeeld een mogelijk circuit kan voorgesteld worden door de formule:

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

Elke variabele stelt een input van het circuit voor en de logische combinatoren stellen logische poorten voor ($\wedge$ voor AND, $\vee$ voor OR, $\neg$ voor NOT en $\oplus$ voor XOR). Het resultaat van de formule komt overeen met de output van het circuit. Kortom, het de bovenstaande formule komt overeen met het onderstaande circuit:



Taak 1a. Maak een nieuw datatype `Circuit` dat een circuit (of booleaanse formule) voorstelt. Een circuit is één van:

1. één van de inputs – elke input wordt gekenmerkt door zijn naam, een `String`;
2. een logische NOT poort toegepast op een onderliggend circuit;
3. een logische AND poort toegepast op twee onderliggende circuits;
4. een logische OR poort toegepast op twee onderliggende circuits; of
5. een logische XOR poort toegepast op twee onderliggende circuits.

Taak 1b. Gebruik het `Circuit` datatype om enkele triviale circuits te schrijven:

1. `cinput :: String -> Circuit` is het circuit dat de input met gegeven naam teruggeeft.
2. `cnot :: Circuit -> Circuit` is het circuit dat de logische NOT van het gegeven circuit teruggeeft.
3. `cand :: Circuit -> Circuit -> Circuit` is het circuit dat de logische AND van de twee gegeven circuits teruggeeft.

4. `cor :: Circuit -> Circuit -> Circuit` is het circuit dat de logische OR van de twee gegeven circuits teruggeeft.

5. `cxor :: Circuit -> Circuit -> Circuit` is het circuit dat de logische XOR van de twee gegeven circuits teruggeeft.

Taak 1c. Maak gebruik van de functies uit Taak 1b. om het voorbeeldcircuit van hierboven te definiëren als `example :: Circuit`.

Taak 1d. Schrijf de functie `candMany :: [Circuit] -> Circuit` die de logische AND teruggeeft van de lijst van gegeven circuits. Ga uit van de veronderstelling dat de lijst niet leeg is.

Deel 2: Spelen met circuits

Taak 2a. Maak een instantie van de `Show` type class voor `Circuit` die een circuit omzet naar een `String` die de overeenkomstige formule voorstelt in een tekstuele vorm. De tekstuele vorm kunnen we best uitleggen aan de hand van ons lopende voorbeeld:

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

Merk op dat er geen spaties in de tekst voorkomen.

Taak 2b. Onze circuits bevatten 4 soorten poorten (NOT, AND, OR, XOR). De laatste twee zijn echter niet essentieel. We kunnen namelijk XOR uitdrukken in functie van de drie eerste, en OR in functie van de twee eerste soorten poorten. Namelijk:

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

Schrijf de functie `simplify :: Circuit -> Circuit` die alle OR en XOR poorten in het gegeven circuit elimineert en een equivalent circuit teruggeeft dat enkel NOT en AND poorten bevat. Maak hierbij gebruik van de bovenstaande gelijkheden.

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

Taak 2c. Schrijf de functie `size :: Circuit -> Int` die het aantal componenten teruggeeft in het gegeven circuit. Inputs zijn geen componenten en tellen dus niet mee.

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

Taak 2d. De *gate delay* van een circuit is het grootste aantal componenten dat op een pad ligt tussen een invoer van het circuit en de uitvoer. Bij het voorbeeld is de gate delay 3 omdat er 3 componenten (NOT, XOR and OR) liggen op het pad tussen de invoer z en de uitvoer. Schrijf de functie `gateDelay :: Circuit -> Int` dat de gate delay voor een gegeven circuit berekent.

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

Taak 2e. Schrijf de functie `inputs :: Circuit -> [String]` die voor een gegeven circuit de lijst van namen van de inputs teruggeeft. De lijst bevat geen dubbels, maar de elementen mogen in willekeurige volgorde staan.

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

Deel 3: Circuits Simuleren

Taak 3a. Schrijf de functie `simulate :: Circuit -> [(String,Bool)] -> Bool` die simuleert welke output het gegeven circuit oplevert als de inputs de gegeven waardes aannemen.

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

Taak 3b. Schrijf de functie `combinations :: Int -> [[Bool]]` zodat `combinations` n de lijst genereert met alle mogelijke verschillende combinaties van n verschillende booleaanse waarden. **Let op:** de combinaties moeten in een bepaalde volgorde staan. Je kan het patroon afleiden van onderstaande voorbeelden:

```
> 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]]
```

Taak 3c. Schrijf de functie `tabulate :: Circuit -> IO ()` die een gegeven circuit afdrukt in tabelvorm zoals hieronder. Maak hierbij handig gebruik van de functies `inputs`, `simulate` en `combinations`. Je mag ervan uitgaan dat de namen van de inputs bestaan uit 1 letter.

```
> 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
```