Deeltentamen Grammatica's en ontleden 18 december 2003

Let op: opgave 1 t/m 4 tellen voor (slechts) 5 punten mee, opgave 5 en 6 ieder voor (maar liefst) 40.
In opgave 1 t/m 4 staan steeds drie beweringen. Eén daarvan is nonsens; de uitspraak is zo onzinnig dat je aan het vaststellen van de waarheid niet eens toekomt: er zit bij wijze van spreken een typeringsfout in. Van de andere twee uitspraken is de ene waar, en de andere onwaar. Geef bij elke opgave aan

- welke uitspraak nonsens is, en vertel in 1 zin waarom
- welke uitspraak waar is: toon dat kort aan, of geef een tegenvoorbeeld voor de onware uitspraak

1. Als L en M talen zijn, dan
   a. $L \cup M$ is gelijk aan $\{s \cup t \mid s \in L, t \in M\}$
   b. $L\,M$ is gelijk aan $\{s\,t \mid s \in L, t \in M\}$
   c. L^n is gelijk aan $\{s^n \mid s \in L\}$
2. Zij G en contextvrije grammatica.
   a. Als $A \to b\,C$ een regel van G is, dan geldt $A\,X \stackrel{*}{\Rightarrow} b\,C\,X$
   b. Als $A\,X \to b\,C$ een regel van G is, dan geldt $A\,X \stackrel{*}{\Rightarrow} b\,C$
   c. Als $A \to b\,C\,X$ een regel van G is, dan geldt $A \stackrel{*}{\Rightarrow} b\,C$
3. Voor contextvrije grammatica's geldt:
   a. Sommige ambigue grammatica's kunnen worden getransformeerd naar een niet-ambigue grammatica met dezelfde taal.
   b. Sommige ambigue talen kunnen worden getransformeerd naar een niet-ambigue taal met dezelfde grammatica.
   c. Er zijn geen transformaties mogelijk die ambiguïteit verwijderen.
4. Voor contextvrije grammatica's geldt:
   a. Een grammatica met een eindig alfabet kan een taal beschrijven met oneindig lange zinnen.
   b. Een grammatica met een eindig alfabet kan een taal beschrijven met oneindig veel zinnen.
   c. Een grammatica met een oneindig alfabet kan een taal beschrijven met oneindig veel zinnen.

5. a. In de Parser-library is het type van parsers als volgt gedefinieerd:

```
type  Parser a b  =  [a] -> [ (b,[a]) ]
```

   Beschrijf kort in woorden wat de polymorfe types `a` en `b` voorstellen, waarom het resultaat van de functie een lijst is, en waarom er tupels in die lijst zitten.

   b. Gegeven zijn de parser-combinators `<+>`, `<*>`, `<$>`, en `many`. Schrijf met behulp daarvan nu zelf de parser-combinator

```
listOf :: Parser a b -> Parser a c -> Parser a [b]
```

   zo dat `listOf p s` één of meer van de constructies herkend door `p` herkent, gescheiden door constructies herkend door `s`.

(zie achterkant voor vervolg)

c. We bekijken de taal van expressies die een type aanduiden in een Haskell-achtige taal, die echter simpeler is dan het echte Haskell. In deze taal zijn er de volgende soort types:
   - twee primitieve types, namelijk `Int` en `Bool`
   - lijst-types, zoals `[Int]` en `[[Bool]]`
   - het nul-tupel `()`, tweetupels zoals `(Int,[Bool])` en meertupels zoals `(Bool,Bool,Int)`
   - functietypes zoals `Int->Bool`
   - en voor groepering mag elk type altijd tussen haakjes gezet worden, zoals in `(Int->Bool)`

   Er zijn in deze taal (anders dan in de echte Haskell) dus *geen* type-variabelen, algebraische **data**-typen, classes enz. Merk ook op dat er geen één-tupels bestaan.

   We gaan een parser schrijven voor expressies in deze taal. Laat je niet in verwarring brengen door het feit dat de parser in Haskell geschreven is, en de taal ook Haskell-achtig is: je kunt voor elke taal parsers schrijven, dus ook voor dit taaltje.

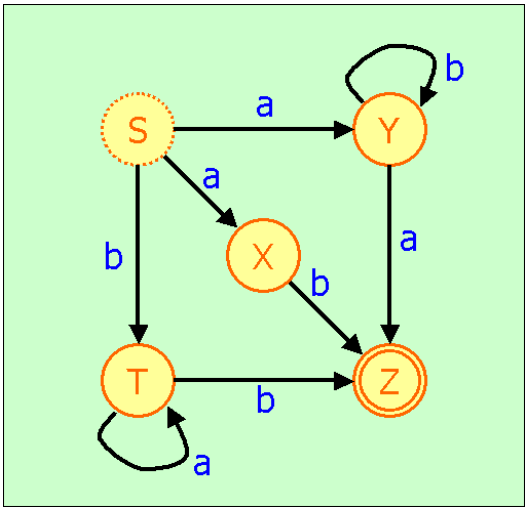
   Definieer eerst een datatype `TypeExpr`, die ontleedbomen van type-expressies in dit taaltje beschrijft.

   Definieer daarna de eigenlijke parser

   ```
   parseTypeExpr :: Parser Char TypeExpr
   ```

   Je mag daarbij alle parser-combinators uit de library gebruiken. Zorg ervoor dat de operator `->` uit het taaltje net zo associeert als in de "echte" Haskell.

6. In onderdeel *a* t/m *c* is gegeven de volgende NFA (Nondeterministische Finite-state Automaton), waarin S de enige start-toestand is, en Z de enige eind-toestand.



   a. construeer een RG (Reguliere Grammatica) die dezelfde taal beschrijft

   b. construeer een DFA (Deterministische Finite-state Automaton) die dezelfde taal beschrijft (mag met een tekening)

   c. construeer een RE (Reguliere Expressie) die dezelfde taal beschrijft

   In onderdeel *d* en *e* is gegeven de volgende RE (Reguliere Expressie):

   ```
   ( a + a* b ) c
   ```

   d. construeer een RG (Reguliere Grammatica) die dezelfde taal beschrijft

   e. construeer een NFA (Non-deterministische Finite-state Automaton) die dezelfde taal beschrijft (mag met een tekening).