Deeltentamen Grammatica's en ontleden 22 december 2005

Let op: opgave 1 t/m 4 tellen voor (slechts) 5 punten mee, opgave 5 voor maar liefst 50 punten, en opgave 6 voor 30 punten.
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. Er geldt:
   (a) Elke reguliere zin is ook contextvrij
   (b) Elke reguliere taal is ook contextvrij
   (c) Elke contextvrije grammatica is ook regulier
2. Een links-recursieve grammatica is vaak een probleem, want
   (a) zo'n grammatica is altijd ambigu
   (b) zo'n grammatica termineert niet
   (c) het is lastig om een parser voor de taal ervan te schrijven
3. Stel G is een contextvrije grammatica (T, N, R, S).
   Dan is w een zin van de taal van G als er een $u \in (N \cup T)^*$ is met
   (a) $S \to u$ en $u \Rightarrow w$
   (b) $S \to u \ \in \ R$ en $u \Rightarrow w$
   (c) $S \Rightarrow u$ en $u \stackrel{*}{\Rightarrow} w$
4. Als G_1 en G_2 contextvrije grammatica's zijn, dan is
   (a) $L(G_1) \cup L(G_2)$ altijd contextvrij
   (b) $G_1 \cup G_2$ soms contextvrij
   (c) $L(G_1) \cap L(G_2)$ nooit contextvrij

5. Hieronder staat een voorbeeld van een Java-programma. Het is bedoeld om te laten zien wat er zoal mogelijk is in een Java-programma. Daaronder volgt toelichting, *waarin we ook aangeven welke aspecten van Java in deze opgave buiten beschouwing blijven.*

```
class Hallo
{
    int p; int q;  String[] s;

    int m1(int x, double y)
    {   int z;
        z=x+1;
        if (z>y)
             return 1+2;
        else return 5;
    }

    void m2(int[] a, List<String> b)
    {   int x;
        x=0;
        while (x<10)
        {   try                          { a[1][2]=3; x++;  }
            catch (ArrayBoundException e){ m4(); x=m1(x,x); }
            catch (Exception e)          { m4();            }
            x++;
        }
    }

    int[][] m3( List<List<String>>[][][] c, List<String[]> d )
    {   if (p>1)
        {   int h;
            h=p;
        }
        List<String> j;
    }

    void m4()
    {
    }

    double r;
}
```

Een klasse bestaat uit een header met daarin de naam (in het voorbeeld `Hallo`), en een body met declaraties (zoals die van `p`, `q`, `s` en `r`) en methode-definities (`m0` tot en met `m4`). Aan de declaratie van `r` kan je zien dat declaraties niet altijd bovenaan de klasse hoeven te staan. *Interfaces en subklasses laten we zitten, en er is dus ook geen 'extends' of 'implements'.*

Een declaratie bestaat uit een type en een naam. *Anders dan in het echte Java moet het type bij elke declaratie herhaald worden. De declaraties van* `p` *en* `q` *kunnen dus niet worden gecombineerd.* Het type is een van de zeven standaardtypes, of een object-type dat met een hoofdletter wordt geschreven. Een object-type mag 'generic' zijn: dan wordt er tussen punthaken een ander type *(maar niet meer dan één)* achter geschreven. Elk type mag een array zijn: dan wordt er een leeg paar vierkante haken achter geschreven. Arrays mogen meer-dimensionaal zijn: dan staan er meerdere paren vierkante haken.

Methodes hebben een type als resultaat, of `void`. *Er zijn geen constructor-methodes.* Methodes hebben nul of meer parameters, waarvan het type in de header wordt gedeclareerd. De body van een methode moet tussen accolades staan, en bestaat uit nul of meer declaraties en/of statements. *Methodes, en ook declaraties, hebben geen verdere attributen. Dus geen 'public', 'static', 'final', 'throws...' enz, en ook geen declaraties-met-initialisatie.*

Eén van de mogelijke statement-vormen is een expressie met een puntkomma er achter. Daarmee zijn al een heleboel gevallen afgedekt, onder andere toekenningen zoals `x=0;`, increments zoals `x++;` en methode-aanroepen zoals `m4();`. De andere statement-vormen zijn:

- if-statement (met optioneel else-gedeelte)
- while-statement
- return-statement (met een resultaat-waarde)
- try-catch-statement (met één of meerdere catch-gedeeltes)
- samengesteld statement tussen accolades (mogelijk met eigen declaraties)

De overige statement-vormen (for, break, switch, do, throw, return zonder resultaat) worden buiten beschouwing gelaten.

(a) **De lexicale fase**

Gegeven is het datatype `Token`, waarmee de bouwstenen van een Java-programma kunnen worden gerepresenteerd:

```
data Token = RHOpen   | RHSluit       -- Ronde Haken     ()
           | VHOpen   | VHSluit       -- Vierkante Haken []
           | PHOpen   | PHSluit       -- PuntHaken       <>
           | AcOpen   | AcSluit       -- Accolades       {}
           | Komma    | Puntkomma
           | KeyIf    | KeyElse
           | KeyWhile | KeyReturn
           | KeyTry   | KeyCatch
           | KeyClass | KeyVoid
           | StdType String           -- de 7 standaardtypes
           | UpperId String           -- uppercase identifiers
           | LowerId String           -- lowercase identifiers
           | ...                      -- overige tokens hier niet van belang
```

Hierin worden namen gegeven aan de diverse soorten haakjes, leestekens, en keywords. Namen zijn gelabeld als een standaardtype, of een identifier die met een lowercase of uppercase letter begint. De zeven standaardtypes zijn beschikbaar in een lijst strings:

```
stdTypes :: [String]
stdTypes = ["int", "long", "double", "float", "byte", "short", "boolean"]
```

Opgave: schrijf een parser

```
lexStdType :: Parser Char Token
```

die de zeven standaardtypes kan herkennen.

(b) **De ontleedboom**
De ontleedboom van een Java-expressie kan worden beschreven met het datatype

```
data Expr = Constante String
          | Variabele String
          | Operatie  Expr String Expr
          | ...
          | ...   -- nog circa tien andere gevallen
```

Dit type, waarvan de precieze details er hier niet toe doen, mag je als gegeven beschouwen.
Het volgende type vat de essentiële onderdelen van een Java-klasse samen:

```
type Klasse = (String, [Elem])
```

Hierin is `Elem` een type dat de mogelijke onderdelen van een klasse-body beschrijft:

```
data Elem = D Decl
          | M Meth
          | S Stat
```

(weliswaar komen in een klasse-body geen losse statements voor, maar dat deze mogelijkheid in het type `Elem` wordt geboden komt later handig uit).

Opgave: schrijf nu zelf vier type-definities of datatype-definities voor de ontleedboom van:
- `Meth` (een methode-definitie)
- `Decl` (een variabele-declaratie)
- `Type` (een Java-type)
- `Stat` (een Java-statement)

In alle gevallen mag je je beperken tot de versimpeling zoals hierboven beschreven.

(c) **De parser**
We gaan nu parsers schrijven die, uitgaande van een lijst van tokens, de diverse ontleedbomen proberen samen te stellen. Het alfabet is hier dus steeds `Token`, het ontleedboomtype varieert:

```
pKlasse :: Parser Token Klasse
pMeth   :: Parser Token Meth
pDecl   :: Parser Token Decl
pType   :: Parser Token Type
pStat   :: Parser Token Stat
pExpr   :: Parser Token Expr
```

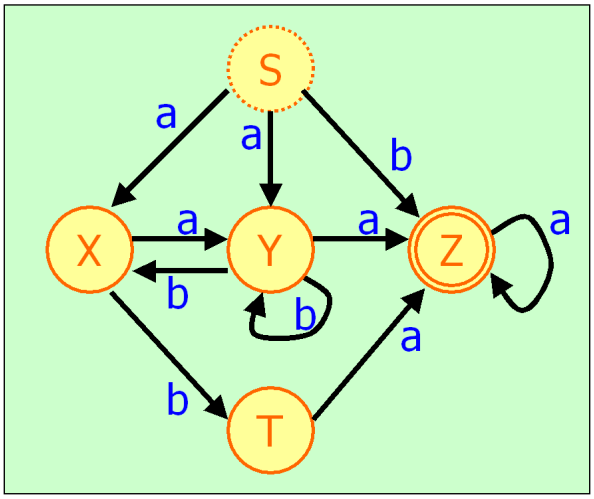
We nemen aan dat de parser `pExpr` al bestaat. Verder zijn alvast gegeven:

```
pPK :: Parser Token Token
pPK =  symbol Puntkomma
pStdType :: Parser Token String
pStdType = getName <$> satisfy isStdType
       where getName   (StdType s) = s
             isStdType (StdType _) = True
             isStdType _           = False
pUpperId, pLowerId :: Parser Token String
```

Uiteraard mag je de parser-combinator library gebruiken, met de operatoren `<|>` of $\oplus$, `<*>` of $\otimes$, `<$>` of ®, en de functies `symbol`, `satisfy`, `many`, `many1`, `listOf`, `option` en `pack`. Je mag ook zelf extra hulpfuncties schrijven.

Opgave: schrijf de volgende parsers:
- `pKlasse`
- `pMeth`
- `pDecl`
- `pType` of `pStat`, naar keuze

6. Gegeven is de volgende NFA (Nondeterministic Finite-state Automaton), waarin S de enige start-toestand is, en Z de enige eind-toestand.



(a) Contrueer een Reguliere Grammatica die dezelfde taal beschrijft.

(b) Construeer een DFA (Deterministic Finite-state Automaton) die dezelfde taal beschrijft (dit mag met een tekening).

(c) Gegeven zijn twee contextvrije grammatica's $G_1 = (T_1, N_1, R_1, S_1)$ en $G_2 = (T_2, N_2, R_2, S_2)$. De doorsnede van N_1 en N_2 is leeg.
Laat nu $G = (T_1 \cup T_2,\ N_1 \cup N_2 \cup \{S\},\ R_1 \cup R_2 \cup \{S \to S_1 S_2\},\ S)$
waarbij S een nieuw startsymbool is.

- Welke taal beschrijft G ?
- Deze constructie werkt niet voor reguliere grammatica's. Waarom niet?
- Beschrijf de constructie van een grammatica met dezelfde taal als G, die regulier is als G_1 en G_2 dat zijn.

Einde