Uitwerking tweede deeltentamen Grammatica's en Ontleden, 2 februari 2006

1. (a) Een reguliere taal kan beschreven worden door een DFA. De n waarvan het bestaan in de stelling geclaimd wordt is het aantal toestanden van die automaat. Als een (deel)zin namelijk langer dan n is, moet op het corresponderende pad een bepaalde toestand tweemaal voorkomen. De w waarvan het bestaan in de stelling geclaimd wordt is het stuk van het pad dat met die lus correspondeert; die string is niet leeg omdat elke pijl in een DFA, dus ook in die lus, een bijschrift heeft. De lus kun je willekeurig vaak herhalen ($i > 1$) of juist weglaten ($i = 0$), en dan heb je nog steeds een pad van begin naar eind.

   (b) Als

voor alle	n	$: n \in \mathbb{N}$	:
er bestaan	u, z, y	$: uzy \in L$ en $\lvert z\rvert \geq n$	:
voor alle	v, w, x	$: z = vwx$ en $\lvert w\rvert > 0$	:
er bestaat een	$i \in \mathbb{N}$	$: uvw^i xy \notin L$	

   dan is L geen reguliere taal.

   Die conclusie, en het woordje 'als', horen er ook bij, want anders is het helemaal geen stelling meer, maar een losse uitspraak over een taal L waarvan verder helemaal niets gezegd wordt.

   (c)
   - $\{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}\}$ is geen reguliere taal. Bewijs met de stelling uit (b): Zij n gegeven. Neem $u = \epsilon$, $z = \mathsf{a}^n$, en $y = \mathsf{b}^n$ (die voldoet aan de eisen, want $uzy = \mathsf{a}^n\mathsf{b}^n \in L$ en $|z| = n \geq n$). Elke opsplitsing van z bestaat alleen uit a's, dus w ook. Neem $i = 2$, dan bevat uvw^ixy meer a's dan b's, en zit dus niet in L. Volgens de stelling is L dan niet regulier.
     Een bewijs dat begint met 'neem $n = 4$' is fout, want n wordt juist door de 'tegenpartij' gekozen!
   - $\{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}\}$ is wel een contextvrije taal, want hij wordt gegenereerd door de contextvrije grammatica $S \to \epsilon|\mathsf{a}S\mathsf{b}$.
     Elke poging om dit met een pumping-lemma te bewijzen is fout, want met pumping-lemma's kun je alleen maar aantonen dat talen **niet** *regulier of CF zijn. Aantonen dat een taal* **wel** *regulier of CF is, doe je door er gewoon een grammatica voor te geven.*

   (d)
   - Het complement van een reguliere taal is weer regulier. Neem de automaat-representatie van de taal. Maak de automaat zo nodig totaal met een extra toestand waar alle nog niet aanwezige pijlen naartoe gaan. Verwissel nu de status van eind- en niet-eind-toestanden. Dan heb je de automaat voor het complement.
     De doorsnede is het complement van de vereniging van de complementen, en omdat complement (zie boven) en vereniging (gegeven v.) regulariteit behouden, doet de doorsnede dat dus ook.
   - Contextvrijheid wordt *niet* behouden onder doorsnede. Bewijs uit het ongerijmde:
     - $\mathsf{a}^n\mathsf{b}^n$ is contextvrij (opgave c)
     - c^k is contextvrij (gegeven i.)
     - $\mathsf{a}^n\mathsf{b}^n\mathsf{c}^k$ is contextvrij (gegeven vi.)
     - $\mathsf{a}^k\mathsf{b}^n\mathsf{c}^n$ is contextvrij (net zo'n redenering)
     - de doorsnede daarvan, $\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n$, zou ook contextvrij zijn

     dit is in tegenspraak met gegeven ii.

2. (a) De firsts wordt bepaald van een nonterminal, de lookaheadset van een productieregel. De firsts bevat de symbolen waarmee de string die door die nonterminal wordt geproduceerd kan beginnen, de lookaheadset bevat de symbolen waarmee de input kan beginnen op het moment dat de regel wordt gebruikt (ook als die symbolen geen deel uitmaken van de met de regel corresponderende ontleedboom).

   (b) De lookaheadsets van de verschillende alternatieven van elke nonterminal moeten onderling disjunct zijn.

   (c) Bij het ontleden kan op grond van het eerste symbool de keuze uit de mogelijke alternatieven gemaakt worden; het ontleden is dus deterministisch.

   (d) Neem de vereniging van de firsts van de symbolen aan de rechterkant van de productieregel, zoals daarvoor het predicaat empty geldt. Indien dit predicaat voor alle symbolen geldt, neem dan ook nog de follow van het symbool aan de linkerkant van de productieregel erbij.

   (e) LL begint met stack waarop alleen het startsymbool staat, en gaat door zolang de stack niet leeg is. LR begint met een lege stack, en gaat door totdat de stack alleen het startsymbool bevat.

   (f) LL vervangt de top van de stack door de rechterkant van de gekozen productieregel. LR haalt mogelijk meerdere elementen van de stack af, namelijk de complete rechterkant van de gekozen productieregel, en vervangt die door de enkele nonterminal van de linkerkant van die regel.

3. Het programma:

```
-- onderdeel a
type ExpAlgebra i b
 = ( ( Int              -> i
     , String           -> i
     , i -> i           -> i
     , b -> i -> i      -> i
     , String -> i -> i -> i
     )
   , ( i -> i -> b
     , b -> b -> b
     )
   )
-- onderdeel b
foldExp :: ExpAlgebra i b -> IntExp -> i
-- onderdeel c
vars :: IntExp -> [String]
vars e = nub (foldExp varAlg e)
varAlg :: ExpAlgebra [String] [String]
varsAlg = ( (fCon, fVar, fAdd, fIf, fLet), (fEq, fAnd))
  where fCon c        = []
        fVar s        = [s]
        fAdd e1 e2    = e1 ++ e2
        fIf  e1 e2 e3 = e1 ++ e2 ++ e3
        fLet s e1 e2  = e1 ++ e2
        fEq  e1 e2    = e1 ++ e2
        fAnd e1 e2    = e1 ++ e2
```

```
-- onderdeel d
eval :: IntExp -> Int
eval e = foldExp evalAlg e []
evalAlg :: ExpAlgebra (Env->Int) (Env->Bool)
evalAlg =  ( (fCon,fVar,fAdd,fIf,fLet),(fEq,fAnd))
  where fCon c        env = c
        fVar s        env = env ? s
        fAdd e1 e2    env = e1 env + e2 env
        fIf  e1 e2 e3 env = if e1 env then e2 env else e3 env
        fLet s e1 e2  env = e2 ((s,e1 env):env)
        fEq  e1 e2    env = e1 env == e2 env
        fAnd e1 e2    env = e1 env && e2 env
-- onderdeel e
code :: IntExp -> Code
code e = foldExp codeAlg e [("x",0)]
codeAlg :: ExpAlgebra (Env->Code) (Env->Code)
codeAlg =  ( (fCon,fVar,fAdd,fIf,fLet),(fEq,fAnd))
  where fCon c        env = [LDC c]
        fVar s        env = [LDL (env ? s)]
        fAdd e1 e2    env = e1 env ++ e2 env ++ [ADD]
        fIf  e1 e2 e3 env = let c2 = e2 env
                                c3 = e3 env
                                n2 = codesize c2
                                n3 = codesize c3
                            in  e1 env ++ [BRF (n2+2)] ++ c2 ++ [BRA n3] ++ c3
        fLet s e1 e2  env = e1 env ++ [STL (length env)] ++ e2 ((s,length env):env)
        fEq  e1 e2    env = e1 env ++ e2 env ++ [EQ]
        fAnd e1 e2    env = e1 env ++ e2 env ++ [AND]
-- onderdeel f
fAnd e1 e2 env = let c1 = e1 env
                     c2 = e2 env
                     n1 = codeSize c1
                     n2 = codeSize c2
                 in c1 ++ [BRF (n2+2)] ++ c2 ++ [BRA 2] ++ [LDC 0]
```