UITWERKING aanvullende toets Grammatica's en Ontleden, 25 februari 2004

De cursieve tekst vormt geen deel van de antwoorden, maar slechts een toelichting daarop.

1. (a) *Dit is definitie 2.3 op blz. 16.*

   De taal $L\ M$ is gedefinieerd als $\{st \mid s \in L \wedge t \in M\}$, of in woorden: strings waarvan het beginstuk een zin van L is, en de rest een zin van M.

   En dus niet: 'zinnen uit de ene taal en zinnen uit de andere taal' (dat is $L \cup M$), en ook niet 'zinnen die zowel in L als M zitten' (dat is $L \cap M$).

   (b) Ja, de concatenatie van twee contextvrije talen is ook weer contextvrij. Laten S' en S'' de startsymbolen van de contextvrije grammatica's van L en M zijn. Zorg er door herbenoemen voor dat de grammatica's verschillende nonterminals gebruiken. Voeg de regels nu samen, en voeg een nieuwe regel $S \rightarrow S'\ S''$ toe, waarbij S een nieuw startsymbool is. Dit is de contextvrije grammatica van $L\ M$.

   (c) Ja, ook de concatenatie van twee reguliere talen is weer regulier. *De constructie is wat ingewikkelder dan bij (b), want $S \rightarrow S'S''$ voldoet niet aan de eisen van een reguliere taal. Maar dat wil niet zeggen dat het onmogelijk is. Het kan namelijk ook anders, op de manier zoals ook gebruikt wordt in stelling 16 in hoofdstuk 5 (transformatie van reguliere expressie naar regulier grammatica):* Laten de reguliere grammatica's voor de twee talen gegeven zijn. Zorg er door herbnoemen voor dat de grammatica's verschillende nonterminals gebruiken. Zet nu in alle regels van de grammatica van L die op ϵ of een terminal eindigen het startsymbool van de grammatica van M er achter. Samen met de onveranderde regels van de grammatica van M vormt dit de reguliere grammatica van $L\ M$.

   Je kunt noch in (b), noch in (c) de pumping lemma's gebruiken. Die zijn namelijk bedoeld om te bewijzen dat iets NIET contextvrij of regulier is, maar hier zijn ze dat juist WEL.

2. (a)
```
data Expr = Con Int
          | Var Char
          | Add Expr Expr
data Pred = LE  Expr Expr
          | And Pred Pred
          | Not Pred
          | All Char Expr Expr Pred
```

   (b) *Expressies met prioriteiten maak je met behulp van een extra nonterminal 'factor' (omdat er geen vermenigvuldiging is, is een derde nonterminal niet nodig):*

```
factor =     Con <$> natural
         <|> Var <$> satisfy isLower
expr   =     factor
         <|> (\l _ r -> Add l r) <$> factor <*> (token "+") <*> expr
```

   of in plaats van die tweede regel nog korter:

```
expr   = chainr factor ((\x->Add)<$>token "+")
```

   Op dezelfde manier worden in predicaten prioriteiten geïntroduceerd:

```
basic  =     (\l _ r -> LE l r) <$> expr <*> token "<=" <*> expr
         <|> (\_ x   -> Not x)  <$> token "!" <*> basic
         <|> parenthesized pred
medium = chainr basic ((\x->And)<$>token "&")
pred   =    medium
         <|> (\_ c _ l _ u _ b -> All c l u b) <$>
              token "A" <*> satisfy isLower <*> token "IN [" <*>
              expr <*> token ".." <*> expr <*> token "]:" <*> pred
```

   (c)
```
PredAlgebra a b = ( ( Int           -> a
                    , Char          -> a
                    , a -> a        -> a
                    )
                  , ( a -> a        -> b
                    , b -> b        -> b
                    , b             -> b
```

```
                        , Char -> a -> a -> b
                        )
                    )
```

(d) *Om de vrije variableen te bepalen, gebruiken we voor beide carrier sets* `[Char]`*. Bij een quantificatie wordt een variabele gebonden, dus mocht die al vrij voorkomen in de body, dan wordt hij uit de lijst vrije variabelen verwijderd.*

```
free  :: PredAlgebra [Char] [Char]
free = ( ( \x -> []  , \x -> [x] , (++)  )
       , ( (++), (++), id, \c l u b -> l++u++filter (/=c) b )
       )
```

(e)
```
value :: PredAlgebra (Env->Int) (Env->Bool)
value = ( ( \x       -> \e -> x
          , \x       -> \e -> e ? x
          , \x y     -> \e -> x e + y e
          )
        , ( \x y     -> \e -> x e <= y e
          , \x y     -> \e -> x e && y e
          , \x       -> \e -> ! (x e)
          , \c l u b -> \e -> and [ b ((c,n):e) | n <- [l e .. u e] ]
          )
        )
```

Om de waarheidswaarde van een predicaat te vinden, kun je folden met deze algebra, en aan de resulterende functie een leeg environment voeren:

```
testpred :: Pred -> Bool
testpred = foldPred value []
```

3. (a) *Een productieregel met bijvoorbeeld drie symbolen aan de rechterkant heeft vier mogelijke cursorposities. Elke mogelijke cursorpositie is een toestand, dus:*

   Het aantal toestanden is de som van de lengtes van de rechterkanten van productieregels, plus het aantal productieregels.

   (b) De toestandsautomaat is **non**-deterministisch.

   Het volgende plaatje in het diktaat was de transformatie van deze non-deterministische automaat naar een deterministische, dus dat zet je op het goede spoor. Maar dit is natuurlijk geen argument... Een technische reden is:

   In de gegeven automaat komen epsilon-overgangen voor, en dat maakt een automaat non-deterministisch. (Immers, als er naast de epsilon-overgangen ook nog andere overgangen zijn, dan kan de automaat daar niet een deterministische keuze uit maken).

   En een andere reden is:

   Er zijn toestanden van waaruit verschillende overgangen met hetzelfde label vertrekken. Dat gebeurt als de cursor 'hopt' over een nonterminal waarvoor er meerdere productieregels zijn (bijvoorbeeld T in het voorbeeld).

   (c) Bij een 'reduce' wordt het bovenste gedeelte van de stack herkend als de rechterkant van een productieregel, en daarom vervangen door de bijbehorende nonterminal aan de linkerkant van de productieregel.

   (d) De toestandsautomaat wordt gerund met de inhoud van de stack als input. Als dat eindigt in een toestand met een cursorpositie aan het eind dan kan er een 'reduce' worden gedaan, anders kiezen we voor 'shift'.

   (e) De LR-ontleedfunctie termineert als de stack alleen nog maar het startsymbool bevat. De LL-ontleedfunctie is hier duaal aan in die zin dat het daar juist begint met het startsymbool op de stack.