UITWERKING tentamen Grammatica's en Ontleden, 29 januari 2004

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

1. (a) *een algebra is een tupel met voor elke constructor van een datatype een functie met evenveel parameters als de constructor. In plaats van recursief gebruik van het datatype komt een polymorfe type-variabele, die tevens het resultaat van de functies is:*

```
type ExprAlgebra a = ( Int -> a
                     , Char -> a
                     , a -> a -> a
                     , a -> a -> a
                     , Char -> a -> a -> a -> a
                     )
```

   (b) *In de opgave staat al: 'aan de hand van een algebra' en 'een expressie doorlopen'. De twee parameters zijn dus een algebra en een expressie. Het resultaat is het carrier-type van de algebra:*

```
foldExpr :: ExprAlgebra a -> Expr -> a
```

   (c) *Als carrier-set gebruiken we de lijst van variabelen, characters dus. De algebra geeft aan wat deze in de vijf gevallen is: In een constante zitten geen variabelen. Een variabele heeft één vrije variabele, namelijk zichzelf. Bij een optelling en vermenigvuldiging verzamelen we de vrije variabelen in de deelexpressies. Ook bij een Sigma nemen we de vrije variabelen van de deelexpressies, maar daaruit filteren we de variabele die door de Sigma wordt gebonden: die is immers niet meer vrij. Zoals in de opgave was aangegeven, geldt de binding alleen voor de derde deelexpressie.*

```
free :: ExprAlgebra [Char]
free = ( \n       -> []
       , \x       -> [x]
       , \a b     -> a ++ b
       , \a b     -> a ++ b
       , \x a b c -> a ++ b ++ filter (/=x) c
       )
```

   Sommigen riepen `free` *resursief aan, zoals in* `free a`*. Dat kan niet, want* `free` *is een algebra en geen functie. De recursieve aanroep zit verborgen in de* `fold` *die later met zo'n algebra aan de slag gaat. Sommigen gebruikten gevalsonderscheid met patronen:* `free (Con n) = ....` *Dat klopt niet, want* `free` *werkt niet op bomen; ook dit is iets wat in* `fold` *gebeurt. Sommigen schreven* `free = fold freeAlg where ....` *Dat klopt niet helemaal: gevraagd werd om de algebra, niet om daar ook alvast* `fold` *op toe te passen.*

   (d) *Om op een 'good' manier met variabelen in een expressie om te gaan, moet de carrier-set een functie zijn die een environment meekrijgt. Alle functies in de algebra krijgen dus een environment als extra parameter, en dan wijst het zich verder vanzelf: een constante staat voor zichzelf en heeft het environment niet nodig, een variabele kun je opzoeken in het environment, bij een optelling en een vermenigvuldiging geeft je het environment recursief door aan de deelexpressies. Alleen het geval Sigma is subtiel. We rekenen eerst de ondergrens en de bovengrens uit in het oude environment:* `a e` *en* `b e`*. Dan maken we een lijst van alle waarden tussen die onder- en bovengrens. Voor al die waarden gaan we de body uitrekenen, maar dan in een environment dat is uitgebreid met een binding van* `x` *aan de respectievelijke waarden. De resultaten worden tenslotte opgeteld: het is immers een sommatie.*

```
eval :: ExprAlgebra (Env -> Int)
eval = ( \n       -> \e -> n
       , \x       -> \e -> e ? x
       , \a b     -> \e -> a e + b e
       , \a b     -> \e -> a e * b e
       , \x a b c -> \e -> sum [ c ((x,n):e) | n <- [a e .. b e] ]
       )
```

   (e) *We roepen eerst de parser aan, en filteren uit de resultatenlijst de oplossingen met lege reststring. Van de oplossingen pakken we de eerste met* `hd`*, en daarvan negeren we*

de reststring met `fst`*. Van die expressie gaan we de vrije variabelen berekenen, door de algebra nu daadwerkelijk aan* `fold` *aan te bieden. En we evalueren de expressie in een initieel leeg environment. Het antwoord* `n` *wordt alleen met* `Yes` *opgeleverd als er inderdaad percies één ontleedresultaat is, en geen vrije variabelen in de expressie. Lazy evaluatie zorgt ervoor dat in geval van een vroegtijdige fout (bijvoorbeeld tijdens het parsen) de vervolg-stappen niet uitgevoerd worden.*

```
value xs | length rs==1 && null vs = Yes n
         | otherwise               = No
    where rs = filter (null.snd) (parseExpr xs)
          e  = fst (hd rs)
          vs = foldExpr free e
          n  = foldExpr eval e []
```

2. (a) *De contrapositie van $P \to Q$ is $\neg Q \to \neg P$. In dit geval luidt die dus:*
Als in een taal L

voor alle	c, d	: $c, d \in \mathbb{N}$	:
er is een	z	: $z \in L$ en $\|z\| > c$	:
voor alle	u, v, w, x, y	: $z = uvwxy$ en $\|vx\| > 0$ en $\|vwx\| \leq d$	:
er is een	$i \in \mathbb{N}$	: $uv^i wx^i y \notin L$	

dan is L geen contextvrije taal.

Die conclusie hoort er ook bij! Met alleen het blok quantoren heb je alleen $\neg Q$ opgeschreven, en dat is geen complete stelling. Let ook op dat de vierde regel eindigt met $\notin L$.

(b) We gebruiken de stelling uit (a). Laten c en d gegeven zijn. Neem $n = \max(c+1, d)$ en laat $z = \mathsf{a}^n\mathsf{b}^n\mathsf{c}^n$ (er geldt dus $|z| > c$). Laat nu vwx een deelstring zijn van z met lengte $\leq < d$. Deze deelstring bevat niet meer dan twee verschillende terminals, omdat de lengte te kort is om de n b's die zich tussen de a's en de c's bevinden te overbruggen. Ook de deel-deel-strings v en x bevatten dus niet meer dan twee verschillende terminals. Neem $i = 2$. De string uv^iwx^iy bevat ten opzichte van $uvwxy$ dus niet van alle drie de terminals extra symbolen. Omdat v en x niet allebei leeg zijn, zijn er dus extra symbolen, maar niet alle drie. De resulterende string bevat dus niet evenveel van de drie verschillende symbolen, en is dus $\notin L$. Uit het lemma volgt nu dat L niet contextvrij is.

Enkelen merkten op dat in het tentamen $|z| > c$ werd vereist, terwijl in het diktaat staat $|z| \geq c$. Dat maakt dat in dit bewijs nodig is om $c+1$ te gebruiken in plaats van c. Dit was echter niet een bedoelde complicatie, en $n = max(c, d)$ is daarom ook goedgerekend. In het diktaat wordt trouwens eerst $<$ gebruikt, en in de contrapositieve formulering ineens $\leq$, wat natuurlijk niet consequent is.

(c) Ja, de klasse van contextvrije talen is gesloten onder vereniging. Als twee talen contextvrij zijn, zijn daar dus CF grammatica's van te geven. Zet in de ene grammatica een accent achter elke nonterminal, in de andere een dubbel accent. Maak nu een nieuwe grammatica, met daarin de regels van de oorspronkelijke grammatica's, een nieuw startsymbool S, en twee extra regels $S \to S'$ en $S \to S''$. In de eerste herschrijfstap wordt S dus herschreven naar het startsymbool van een van beide grammatica's, en dus zijn alle zinnen van beide talen afleidbaar.

Het bewijs is dus gewoon een directe constructie. Elke poging om dit met behulp van de pompstelling te bewijzen is gedoemd om te falen. Met de pompstelling kun je namelijk alleen maar bewijzen dat iets juist niet een contextvrije taal is.

(d) Nee, de klasse van contextvrije talen is niet gesloten onder doorsnede. Stel dat het wel zo was. Dan zou ook de doorsnede van de contextvrije talen $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^m \mid n, m \in \mathbb{N}\}$ en $\{\mathsf{a}^m\mathsf{b}^n\mathsf{c}^n \mid n, m \in \mathbb{N}\}$ contextvrij zijn. Maar die doorsnede is $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \in \mathbb{N}\}$, en dat is in tegenspraak met het resultaat in (b).

3. (a) *De firsts zijn de terminals waarmee een deelzin geproduceerd door een bepaalde nonterminal kan beginnen:*

$$\mathit{firsts}(N) \;=\; \{\; \mathsf{x} \in T \;\mid\; N \stackrel{*}{\Rightarrow} \mathsf{x}\,\alpha \;\}$$

(b)

nonterminal	*empty*	*firsts*	*follow*
S	nee	{ p, q, r, x, y }	$\{\#\}$
A	ja	{ x, y }	$\{\mathsf{p}, \mathsf{q}, \mathsf{w}\}$
B	nee	{ p, q }	$\{\#\}$
C	nee	{ x, y, z }	$\{\#\}$
D	ja	{ y }	$\{\mathsf{w}, \mathsf{x}, \mathsf{z}\}$
E	ja	{ x }	$\{\mathsf{w}, \mathsf{z}\}$

(c)

	productie			lookaheadset	*toelichting*
1.	S	$\rightarrow$	$A\ B$	$\{\mathsf{p}, \mathsf{q}, \mathsf{x}, \mathsf{y}\}$	*firsts(A)* $\cup$ *firsts(B)*
2.	S	$\rightarrow$	$\mathsf{r}\ A\ \mathsf{w}$	$\{\mathsf{r}\}$	*direct*
3.	A	$\rightarrow$	$D\ E$	$\{\mathsf{p}, \mathsf{q}, \mathsf{w}, \mathsf{x}, \mathsf{y}\}$	*firsts(D)* $\cup$ *firsts(E)* $\cup$ *follow(A)*
4.	B	$\rightarrow$	p	$\{\mathsf{p}\}$	*direct*
5.	B	$\rightarrow$	$\mathsf{q}\ C$	$\{\mathsf{q}\}$	*direct*
6.	C	$\rightarrow$	$E\ \mathsf{z}$	$\{\mathsf{x}, \mathsf{z}\}$	*firsts(E)* $\cup$ $\{\mathsf{z}\}$
7.	C	$\rightarrow$	y	$\{\mathsf{y}\}$	*direct*
8.	D	$\rightarrow$	ε	$\{\mathsf{w}, \mathsf{x}, \mathsf{z}\}$	*follow(D)*
9.	D	$\rightarrow$	y	$\{\mathsf{y}\}$	*direct*
10.	E	$\rightarrow$	ε	$\{\mathsf{w}, \mathsf{z}\}$	*follow(E)*
11.	E	$\rightarrow$	x	$\{\mathsf{x}\}$	*direct*

(d) Als het symbool op de top van de stack een nonterminal is, wordt die vervangen door de rechterkant van een van de regels voor die nonterminal. Díe regel wordt uitgekozen, van wiens lookaheadset het eerstvolgende symbool op de input een element is.

(e) Shift: een symbool van de input wordt gelezen en op de stack gezet.
Reduce: op de stack wordt de rechterkant van een productieregel herkend, en vervangen door de bijbehorende nonterminal.