Tweede deeltentamen Grammatica's en Ontleden, 27 januari 2005, 9–12 uur

1. We bekijken de taal van logische predicaten, zoals

$$\forall x \in \{1,2,3\} : \exists y \in \{1,4\} : (x = y \ \vee \ \forall z \in \{1\} : x \neq z)$$

De basis-predicaten zijn gelijkheden en ongelijkheden tussen variabelen; predicaten kunnen worden gecombineerd met $\wedge$, $\vee$ en $\rightarrow$; en er zijn quantoren $\forall$ en $\exists$ die een variabele laten lopen over een eindige lijst integers. In deze taal is er dus *geen* not-operator!

Het volgende Haskell datatype definieert ontleedbomen voor dit soort predicaten:

```
data Pred  = Eq  String String
           | NEq String String
           | And Pred   Pred
           | Or  Pred   Pred
           | Imp Pred   Pred
           | All String Range Pred
           | Exi String Range Pred
type Range = [Int]
```

Bovenstaand voorbeeld wordt door een parser omgezet in:

```
All "x" [1,2,3] (Exi "y" [1,4] (Or (Eq "x" "y") (All "z" [1] (NEq "x" "z"))))
```

Gegeven is verder een environment-type en een bijbehorende opzoek-operator:

```
type Env = [(String,Int)]
(?) :: Env -> String -> Int
```

(a) Definieer een Haskell `type` voor een bij het type `Pred` horende algebra.

(b) Definieer de functie

```
foldPred :: PredAlgebra c -> Pred -> c
```

waarmee aan de hand van een algebra een ontleedboom recursief kan worden doorlopen.

(c) Definieer een algebra `freeAlg` zodat `foldPred freeAlg` alle vrije variabelen van een predicaat bepaalt (in een lijst, waarbij het niet erg is als die dubbelen bevat). Geef ook het type van `freeAlg`, dus vul in:

```
freeAlg :: ....
freeAlg  = ....
```

Vrije variabelen zijn variabelen die niet door een omvattende $\forall$ of $\exists$ worden gebonden.

(d) We gaan een functie schrijven die de ontkenning van een predicaat bepaalt (of in ieder geval, een predicaat dat daaraan equivalent is). Als er in de taal een Not-constructie had bestaan, was dat heel gemakkelijk geweest:

```
neg :: Pred -> Pred
neg p = Not p
```

maar in deze taal bestaat die constructie niet.

Toch is het mogelijk om de functie te schrijven, en wel door:

```
neg :: Pred -> Pred
neg = foldPred negAlgebra
```

Schrijf de hiervoor benodigde `negAlgebra`.

(e) Definieer een functie

```
eval :: Pred -> Bool
```

die de waarde van een predicaat bepaalt. Je mag daarbij aannemen dat de expressie geen vrije variabelen bevat. Dit is mogelijk door de body van een quatificatie te evalueren voor alle mogelijke waarden die in de range van zijn variabele zijn aangegeven.

Het recursief doorlopen van de ontleedboom moet gedaan worden met behulp van een algebra. Geef van die algebra ook het type, en vergeet ook het 'hoofdprogramma' niet!

1

2. Hier zijn vier (zogeheten 'pomp'-)stellingen:

   - **1:** In een contextvrije taal L geldt:

er bestaan	c, d	: $c, d \in \mathbb{N}$	:
voor alle	z	: $z \in L$ en $\lvert z\rvert \geq c$	:
er bestaan	u, v, w, x, y	: $z = uvwxy$ en $\lvert vx\rvert > 0$ en $\lvert vwx\rvert \leq d$	:
voor alle	$i \in \mathbb{N}$	: $uv^iwx^iy \in L$	

   - **2:** Als

voor alle	c, d	: $c, d \in \mathbb{N}$	:
er bestaat een	z	: $z \in L$ en $\lvert z\rvert \geq c$	:
voor alle	u, v, w, x, y	: $z = uvwxy$ en $\lvert vx\rvert > 0$ en $\lvert vwx\rvert \leq d$	:
er bestaat een	$i \in \mathbb{N}$	: $uv^iwx^iy \notin L$	

     dan is L geen contextvrije taal

   - **3:** In een reguliere taal L geldt:

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

   - **4:** 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	$i \in \mathbb{N}$	: $uvw^ixy \notin L$	

     dan is L geen reguliere taal

   (a) Bewijs één van deze vier stellingen.
   (Je mag zelf kiezen welke. Ze zijn alle vier waar, maar van sommige is het bewijs moeilijker, dus denk goed na welke je kiest!)

   (b) Bewijs dat de klasse van contextvrije talen meer talen bevat dan de klasse van reguliere talen.

   (c) Is de klasse van reguliere talen gesloten onder complementatie (d.w.z.: is het complement van een reguliere taal weer een reguliere taal)? Geef een bewijs van het antwoord!

   (d) Is de klasse van contextvrije talen gesloten onder vereniging? Geef een bewijs van het antwoord!

zie volgende blad!

3. *de deelopgaven wegen in de verhouding 1:1:1:4:3.*

   (a) Bij LL(1)-ontleden spelen de *lookaheadsets* een belangrijke rol. Van wat voor soort dingen kun je de lookaheadset bepalen, en wat voor soort dingen krijg je dan in handen? (Met andere woorden: wat is het type van de functie die de lookeaheadsets bepaalt?)

   (b) Waarom was het niet nodig om lookaheadsets te bepalen bij het bouwen van ontleders met de Haskell parser-combinator-library?

   (c) Aan welke voorwaarde moet een grammatica voldoen om LL(1)-ontleden mogelijk te maken?

   (d) We maken een simpele representatie van een grammatica als lijst van tweetupels in Haskell:

   ```
   type Gram = [ (Char, String) ]
   ```

   nonterminals worden voorgesteld door hoofdletters, terminals door kleine letters, en het startsymbool is altijd `'S'`. De te gebruiken grammatica is beschikbaar in een globale variabele `gram::Gram`. (Ook de lookahead-functie heeft toegang tot die globale variabele).

   De functie `check` controleert of een string in de taal van deze grammatica zit:

   ```
   check :: String -> Bool
   ```

   Deze functie maakt gebruik van een stackmachine `run`. De stack bevat de nog te herkennen nonterminals.

   ```
   type Stack = [Char]
   run :: Stack -> String -> Bool
   run []     []                 = ...
   run []     (x:xs)             = ...
   run (a:as) []                 = ...
   run (a:as) (x:xs) | isLower a = ...
                     | isUpper a = ...
   check = run ...
   ```

   Vul in wat op de puntjes moet staan. Je mag de functie `lookaheadset` aanroepen die de lookaheadset bepaalt. Uiteraard mag je zelf ook extra hulpfuncties definiëren.

   (e) Bij LR-ontleden wordt gebruik gemaakt van een soortgelijke, maar net iets andere stackmachine. Daarbij speelt op een zeker moment een Nondeterministische Finite-state Automaat (NFA) een rol (die later ook nog omgevormd wordt tot een Deterministische automaat (DFA)).

      i. Wat is het alfabet van die NFA?

      ii. Wat zijn de toestanden van die NFA?

      iii. Afhankelijk van de eindtoestand van de DFA wordt door de stackmachine een bepaalde beslissing genomen. Welke beslissing is dat?