Grammatica's en Ontleden – Deeltentamen 1 (van 2)

Maandag 18 december 2006 (15:00-17:00)

Dit examen bestaat uit vier meerkeuze vragen en twee open vragen. In totaal zijn er 100 punten te verdienen: 7 punten per meerkeuze vraag, 40 punten voor vraag 5 en 32 punten voor vraag 6. Als er om een Haskell functie wordt gevraagd dan mag je alles uit de `Prelude` *en uit de modules* `List`, `Char` *en* `ParseLib` *gebruiken. Het is niet toegestaan om tijdens het tentamen het diktaat of ander lesmateriaal te raadplegen. Veel succes!*

---

1) Doorsnede van twee talen (7 PUNTEN)

Gegeven zijn de talen L en M met hetzelfde alfabet. Hieronder staan twee beweringen over deze talen:

Bewering 1: "Als L en M allebei context-vrij zijn, dan is $L \cap M$ ook een context-vrije taal."

Bewering 2: "Als L en M allebei regulier zijn, dan is $L \cap M$ ook een reguliere taal."

a) Beide beweringen zijn onjuist

b) Alleen bewering 1 is waar

c) Alleen bewering 2 is waar

d) Beide beweringen zijn waar

2) Parser combinators (7 PUNTEN)

De functie *listOf* construeert uit twee parsers een nieuwe parser. De nieuwe parser kan worden gebruikt om één of meer keer iets te parsen (eerste argument) dat gescheiden is door iets anders (tweede argument). Ter herinnering: het type van deze combinator is:

$$listOf :: Parser\ s\ a \to Parser\ s\ b \to Parser\ s\ [a]$$

Laat de functie *list* als volgt gedefinieerd zijn:

$$list\ x\ xs = x : xs$$

Welke van de onderstaande definities is een correcte implementatie van *listOf*?

a) $listOf\ p\ s = list <\$> p <*> s <*> listOf\ p\ s <|> succeed\ []$

b) $listOf\ p\ s = list <\$> p <*> many1\ ((\lambda x\ y \to y) <\$> s <*> p)$

c) $listOf\ p\ s = list <\$> p <*> option\ ((\lambda x\ y \to y) <\$> s <*> listOf\ p\ s)\ []$

d) $listOf\ p\ s = list <\$> p <*> option\ ((\lambda x\ y \to y) <\$> s <*> p)\ []$

3) Grammatica transformatie (7 PUNTEN)

Laat $G = (T, N, R, S)$ een reguliere grammatica zijn voor de reguliere taal L en $S' \notin N$. Welke van de onderstaande grammatica's genereert de taal L^* en is nog steeds regulier?

a) $(T, N \cup \{S'\}, R' \cup \{S' \to S, S' \to \epsilon\}, S')$, zodanig dat R' alle regels uit R bevat waarbij achter iedere productieregel die niet eindigt op een hulpsymbool S' wordt geplakt.

b) $(T, N \cup \{S'\}, R \cup \{S' \to S\ S', S' \to \epsilon\}, S')$

c) $(T, N \cup \{S'\}, R' \cup \{S' \to S, S' \to \epsilon\}, S')$, zodanig dat R' precies alle regels uit R bevat plus de productieregels $X \to S'$ voor alle hulpsymbolen X uit de verzameling N.

d) De grammatica G zelf.

4) Links-recursie verwijderen (7 PUNTEN)

Met de volgende grammatica kan een treinreis worden beschreven:

$$\begin{array}{lcl} \mathit{TS} & \to & \mathit{TS}\ \mathit{Tijd}\ \mathit{Tijd}\ \mathit{TS} \mid \mathit{Station} \\ \mathit{Station} & \to & \mathit{Identifier} \\ \mathit{Tijd} & \to & \mathit{Nat} : \mathit{Nat} \end{array}$$

Welke van de onderstaande grammatica's is niet meer links-recursief en nog wel equivalent?

a)
$$\begin{array}{lcl} \mathit{TS} & \to & \mathit{TS}\ (\mathit{Tijd}\ \mathit{Tijd}\ \mathit{Station})* \\ \mathit{Station} & \to & \mathit{Identifier} \\ \mathit{Tijd} & \to & \mathit{Nat} : \mathit{Nat} \end{array}$$

b)
$$\begin{array}{lcl} \mathit{TS} & \to & \mathit{Station} \mid \mathit{Station}\ Z \\ Z & \to & \mathit{Tijd}\ \mathit{Tijd}\ \mathit{TS} \mid \mathit{Tijd}\ \mathit{Tijd}\ \mathit{TS}\ Z \\ \mathit{Station} & \to & \mathit{Identifier} \\ \mathit{Tijd} & \to & \mathit{Nat} : \mathit{Nat} \end{array}$$

c)
$$\begin{array}{lcl} \mathit{TS} & \to & Z\ \mathit{Tijd}\ \mathit{Tijd}\ \mathit{TS} \mid \mathit{Station} \\ Z & \to & \mathit{TS} \mid \epsilon \\ \mathit{Station} & \to & \mathit{Identifier} \\ \mathit{Tijd} & \to & \mathit{Nat} : \mathit{Nat} \end{array}$$

d)
$$\begin{array}{lcl} \mathit{TS} & \to & \mathit{Station}\ \mathit{Tijd}\ \mathit{Tijd}\ Z \\ Z & \to & \mathit{Station} \mid \mathit{TS} \\ \mathit{Station} & \to & \mathit{Identifier} \\ \mathit{Tijd} & \to & \mathit{Nat} : \mathit{Nat} \end{array}$$

5) Lexen en parsen (40 PUNTEN)

We gaan een lexer en een parser schrijven voor een taal die bestaat uit proposities met variabelen en universele kwantoren. Eerst worden een aantal voorbeeldzinnen gegeven uit de taal, daarna volgt nog een toelichting.

```
                voor alle x,y : x of niet y
xs en (waar of onwaar of (voor alle y : niet y))
                p en (q) of niet (p en q)
                 waar of niet niet waar
```

In proposities mogen variabelen worden gebruikt: zo'n variabele bestaat uit één of meer kleine letters. Sommige woorden (bijvoorbeeld `niet`) hebben een speciale betekenis en mogen niet als variabele worden gebruikt. Iedere geldige propositie mag tussen haakjes geschreven worden. Universele kwantificatie wordt genoteerd door `voor alle` (twee losse woorden), gevolgd door één of meer variabelen gescheiden door komma's, gevolgd door een dubbele punt en een propositie. Tenslotte hebben we nog de binaire operatoren `en` en `of` (deze worden infix geschreven), de unaire operator `niet`, en de twee constanten `waar` en `onwaar`.

Als eerste gaan we een lexer schrijven. Net als bij de eerste twee praktikumopgaven zijn we niet geïnteresseerd in whitespace: spaties dienen alleen om de tokens van elkaar te scheiden en mogen door de lexer worden weggegooid. Hieronder staat een datatype voor de tokens en de definitie van de tabel *terminals*. Deze definities mag je gebruiken om de lexer te definiëren.

```
data Token = Variabele String | Waar | Onwaar | Voor | Alle
           | Niet | En | Of | Komma | DPunt | Open | Sluit
  deriving (Show, Eq)

terminals =
  [(Waar,   "waar"  )
  ,(Onwaar, "onwaar")
  ,(Voor,   "voor"  )
  ,(Alle,   "alle"  )
  ,(Niet,   "niet"  )
  ,(En,     "en"    )
  ,(Of,     "of"    )
  ,(Komma,  ","     )
  ,(DPunt,  ":"     )
  ,(Open,   "("     )
  ,(Sluit,  ")"     )
  ]
```

a) Schrijf een lexer voor de taal van proposities met behulp van de parser combinators. Het resultaat moet van het type [*Token*] zijn. Geef ook het type van de lexer.

Voordat we een grammatica en een parser kunnen schrijven voor de taal moeten we zeer precies specificeren hoe een zin uit de taal moet worden ontleed. Om een universeel gekwantificeerde propositie als onderdeel van een grotere propositie te gebruiken moeten er haakjes omheen geschreven worden. De unaire operator `niet` heeft de hoogste prioriteit. De zin `niet niet p` is een geldige propositie en betekent `niet (niet p)`. Van de twee binaire operatoren krijgt `en` de hoogste prioriteit. Beide operatoren zijn rechts associatief.

b) Geef productieregels die de taal van proposities beschrijven en die niet ambigu zijn. Het is toegestaan (maar niet verplicht) om EBNF notatie te gebruiken.

c) Definieer een Haskell datatype *Prop* om ontleedbomen van proposities te representeren.

d) Schrijf een parser voor proposities: ga er van uit dat de invoer al verwerkt is tot een lijst van tokens. De grammatica van onderdeel **b** is een goede basis voor de parser. Het resultaat van de parser moet het datatype zijn dat je voor onderdeel **c** hebt gekozen. Geef ook het type van de parser.

Hint: mocht je niet uit de semantische functies komen, schrijf dan wel de parser op waarin je de semantische functies open laat.

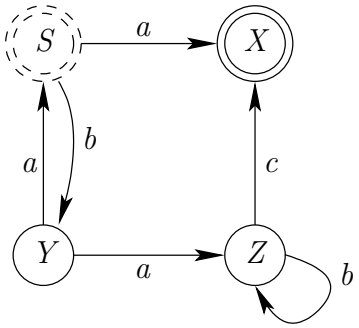
e) Schrijf een functie

$$test :: String \rightarrow Prop$$

die een string ontleedt en een propositie teruggeeft. Je mag er van uit gaan dat de string altijd een geldige propositie voorstelt. De functie *start* om een parser mee op te starten mag *niet* bekend worden verondersteld. Als je deze functie wilt gebruiken dan moet je deze eerst nog definiëren.

6) Reguliere talen (32 PUNTEN)

Gegeven is de volgende niet-deterministische eindige toestandsautomaat (NFA), waarin S de enige start-toestand is en $\{S, X\}$ de verzameling is van eind-toestanden.



a) Geef drie verschillende strings uit $\{a, b, c\}^*$ die behoren tot de taal van de NFA.

b) Geef drie verschillende strings uit $\{a, b, c\}^*$ die *niet* behoren tot de taal van de NFA.

c) Construeer een deterministische toestandsautomaat (DFA) die dezelfde taal beschrijft als de NFA (dit mag met behulp van een tekening).

d) Geef een reguliere expressie die dezelfde taal beschrijft als de NFA.