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

Dinsdag 18 december 2007 (15:00-17:00)

Johan Jeuring

Dit examen bestaat uit acht meerkeuze vragen en een open vraag. Geef altijd het beste *antwoord op een meerkeuzevraag. In totaal zijn er 90 punten te verdienen: 6 punten per meerkeuze vraag, en 42 punten voor de open vraag. 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) Zinnen van context-vrije grammatica's (6 PUNTEN)

Een grammatica heeft de volgende produkties:

$$S \quad \rightarrow \quad \mathtt{a}S\mathtt{b} \mid S\mathtt{a} \mid \epsilon$$

Als we nog één produktie toevoegen, kunnen we de zin

`aabbaabb`

afleiden. Welke van de volgende produkties dient te worden toegevoegd?

1. $S \rightarrow SS$
2. $S \rightarrow \mathtt{a}S\mathtt{bb}$
3. $S \rightarrow \mathtt{bb}S\mathtt{aa}$
4. Alle antwoorden zijn goed.

2) Taal van een grammatica (6 PUNTEN)

Wat is de taal van de grammatica met de volgende produkties?

$$\begin{array}{lcl} S & \rightarrow & AS\mathtt{b} \mid \mathtt{c} \\ A & \rightarrow & \mathtt{a} \end{array}$$

1. $\{\mathtt{ac}y \mid y \in \{\mathtt{b}\}^*\}$
2. $\{\mathtt{a}^n\mathtt{cb}^n \mid n \in Nat\}$
3. $\{x\mathtt{cb} \mid x \in \{\mathtt{a}\}^*\}$
4. Alle antwoorden zijn fout.

3) Grammatica transformaties (6 PUNTEN)

Pas de 'removing left-recursion' grammatica transformatie toe op de volgende grammatica.

$$E \rightarrow E{+}E \mid E{*}E \mid Digs \mid (E)$$

Wat is het precieze resultaat?

1. $$\begin{aligned} E &\rightarrow T{+}E \mid T \\ T &\rightarrow F{*}T \mid F \\ F &\rightarrow Digs \mid (E) \end{aligned}$$

2. $$\begin{aligned} E &\rightarrow E\ Z \mid Digs \mid (E) \\ Z &\rightarrow {+}E \mid {*}E \end{aligned}$$

3. $$\begin{aligned} E &\rightarrow Digs\ Z? \mid (E)\ Z? \\ Z &\rightarrow {+}E\ Z? \mid {*}E\ Z? \end{aligned}$$

4. $$E \rightarrow Digs{+}E \mid Digs{*}E \mid (E){+}E \mid (E){*}E \mid Digs \mid (E)$$

4) Abstracte syntax (6 PUNTEN)

Welke abstracte syntax past het best bij de grammatica met de volgende produkties?

$$S \rightarrow \mathtt{a}S \mid S\mathtt{bb}S \mid \epsilon$$

1. ```
   data S = S [(S,(S,S),[S])]
   ```

2. ```
   data S = SingleA Char
          | TwoB Char Char
          | Empty
   ```

3. ```
   data S = SingleA S
          | TwoB S S
          | Empty
   ```

4. ```
   type S = (Char,(Char,Char),())
   ```

5) Type van parser combinators (6 PUNTEN)

Wat is het type van de parser `p`?

```
p = (\x y z -> (x,z)) <$> natural
```

1. `Parser Char (Int -> Int -> (Int,Int))`
2. `Parser Char (a -> b -> (Int,b))`
3. `Parser Char (Int -> a -> b -> (Int,b))`
4. `Parser Char (Int,b)`

6) Gebruik van parser combinators (6 PUNTEN)

Een collegerooster bestaat uit regels onder elkaar, waarin een regel bestaat uit een dag, tijd, naam van het vak, en naam docent. Neem aan dat de types `Dag`, `Tijd`, `Naamvak`, `Naamdocent`, en de parsers

```
pDag         ::  Parser Char Dag
pTijd        ::  Parser Char Tijd
pNaamvak     ::  Parser Char Naamvak
pNaamdocent  ::  Parser Char Naamdocent
```

voorgedefinieerd zijn. Hoe wordt de functie `pCollegerooster`,

```
pCollegerooster :: Parser Char [(Dag,Tijd,Naamvak,Naamdocent)]
```

die een collegerooster ontleedt, gedefinieerd?

1.
```
pCollegerooster =
  (\x y -> x) <$>
  listOf ((\u v w x -> (u,v,w,x)) <$>
              pDag
         <*> pTijd
         <*> pNaamvak
         <*> pNaamdocent
         )
         (token "\n")
  <*> token "\n"
```

2.
```
pCollegerooster =
    ((\u v w x y z -> (u,v,w,x):z) <$>
        pDag
    <*> pTijd
    <*> pNaamvak
    <*> pNaamdocent
    <*> token "\n"
    <*> pCollegerooster
    )
 <|>
    succeed []
```

3.
```
pCollegerooster =
  many ((\u v w x y -> (u,v,w,x)) <$>
            pDag
       <*> pTijd
       <*> pNaamvak
       <*> pNaamdocent
       <*> token "\n"
       )
```

4. Alle antwoorden zijn goed.

7) Eindige automaten

Stel de automaat M heeft toestanden $\{S, A, B, C\}$, waarbij S de start toestand is, en B de eindtoestand. M's input alfabet is $\{\texttt{a}, \texttt{b}, \texttt{c}\}$. Automaat M moet tenminste de volgende zinnen accepteren:

```
b
ab
aca
aaca
```

Welke van de volgende toestandstransitiefunkties heeft M hiervoor nodig?

1. ```
   d S a = S
   d S b = B
   d S c = A
   d A a = B
   ```

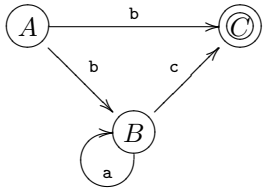
2. ```
   d S a = A
   d S b = B
   d A a = A
   d A c = B
   ```

3. ```
   d S a = A
   d S b = B
   d A a = B
   d A c = A
   ```

4. Alle antwoorden zijn fout.
   ```

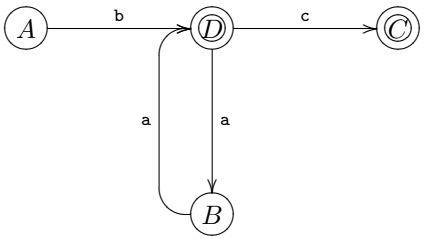
8) Van non-deterministische naar deterministische automaten

Gegeven is de volgende nondeterministische automaat M met start toestand A:

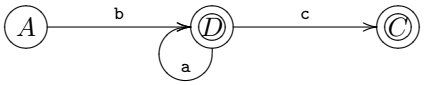


Welke van de deterministische automaten met de volgende toestandstransitiefunkties accepteert dezelfde taal? (De toestandsverzameling van de automaten zijn de toestanden die in de transitiefunktie worden genoemd, de begintoestand is altijd `A`, de eindtoestanden `C` (indien aanwezig) en `D`.)

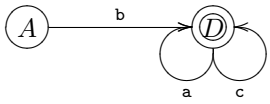
1.



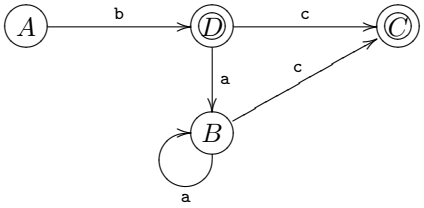
2.



3.



4.



9) Ontleden (42 punten)

Een vereenvoudigde winkelbon ziet er als volgt uit:

```
melk 1.19
boter 0.99
cola 1.79
kaas 7.49
```

Een andere bon, die we `broodstring` zullen noemen, ziet er als volgt uit:

```
brood 3.55
yoghurt 1.99
kwark 1.49
```

Het doel van deze opgave is een parser te schrijven die een lijstje van produkten en prijzen ontleedt, en die vervolgens de gehele bon, inclusief een lege regel en het totaal bedrag, oplevert. Denk ook na over wat je met newlines doet.

1. Geef een concrete syntax voor een winkelbon waarmee onder andere de bovenstaande voorbeelden kan worden gegenereerd. Neem aan dat produkten identifiers zijn, en prijzen twee natuurlijke getallen gescheiden door een punt. De produkties voor identifiers en natuurlijke getallen hoef je niet te geven.

2. Geef een abstracte syntax `AbstractBon` voor winkelbonnen.

3. Schrijf een parser `parseBon` van het type

   ```
   parseBon :: Parser Char AbstractBon
   ```

   die een `String` als input neemt, en als ontleedresultaat een `AbstractBon` oplevert.

4. Schrijf een functie `totaal`

   ```
   totaal :: AbstractBon -> (Int,Int)
   ```

   die het totaal bedrag in (euros,centen) van de produkten in een `AbstractBon` berekent. Bijvoorbeeld, als `ab` een resultaat is van `parseBon broodstring`, dan

   ```
   ? totaal ab
   (7,3)
   ```

5. Schrijf een funktie `printBon`

   ```
   printBon :: AbstractBon -> String
   ```

   Die een abstracte bon afdrukt, inclusief een lege regel en een regel waarin het totaal wordt gegeven. Bijvoorbeeld:

   ```
   ? parse_result ((printBon <$> parseBon) broodstring)
   "brood 3.55\nyoghurt 1.99\nkwark 1.49\n\ntotaal 7.03"
   ```

   waarbij `broodstring` weer de input string is die hierboven staat.