Department of Information and Computing Sciences
Utrecht University

INFOB3TC – Exam 1

Johan Jeuring

Wednesday, 21 December 2016, 11:00–13:00

Preliminaries

- The exam consists of 12 pages (including this page). Please verify that you got all the pages.
- Fill out the answers **on the exam itself**.
- Write your **name** and **student number** here:

- The maximum score is stated at the top of each question. The total amount of points you can get is 100.
- Try to give simple and concise answers. Write readable text. Do not use pencils or pens with red ink. You may use Dutch or English.
- When writing grammar and language constructs, you may use any set, sequence, or language operations covered in the lecture notes.
- When writing Haskell code, you may use Prelude functions and functions from the following modules: *Data.Char*, *Data.List*, *Data.Maybe*, and *Control.Monad*. Also, you may use all the parser combinators from the `uu-tc` package. If you are in doubt whether a certain function is allowed, please ask.

Good luck!

Multiple-choice questions

In this series of 10 multiple-choice question, you get:

- 5 points for each correct answer,
- 1 point if you do not answer the question,
- and 0 points for a wrong answer.

Answer these questions with *one of* a, b, c, or d. Sometimes multiple answers are correct, and then you need to give the *best* answer.

1 (5 points). A grammar has the following productions:

$$T \quad \rightarrow \quad \mathtt{y} \mid \mathtt{x}T\mathtt{x} \mid T\mathtt{xyx}T$$

Which of the following sequences is a sentence in the language of T?

a) yxyxxxyxx

b) xxxyyyxxx

c) yxyxyxyx

d) yxyxxxxxyxy

•

2 (5 points). A grammar has the following productions:

$$T \quad \rightarrow \quad \epsilon \mid T\mathtt{x} \mid \mathtt{x}T\mathtt{y}$$

If we add a single production to this grammar, we can derive the sentence xxyyxxyy. Which of the following productions do we have to add?

a) $T \rightarrow \mathtt{x}T\mathtt{yy}$

b) $T \rightarrow \mathtt{yy}T\mathtt{xx}$

c) $T \rightarrow TT$

d) All of the above answers are correct.

•

3 (5 points)**.** You want to write a parser using the standard parser combinator approach for the following grammar:

$$\begin{aligned} S &\rightarrow R\texttt{a} \mid S\texttt{a} \mid \texttt{z} \\ R &\rightarrow \texttt{b}R \mid \texttt{b}S \end{aligned}$$

Before you construct the parser, you first transform the grammar by:

a) Removing left-recursion obtaining

$$\begin{aligned} S &\rightarrow (R\texttt{a})Z? \mid \texttt{z}Z? \\ Z &\rightarrow \texttt{a}Z? \\ R &\rightarrow \texttt{b}R \mid \texttt{b}S \end{aligned}$$

b) Left-factoring obtaining

$$\begin{aligned} S &\rightarrow R\texttt{a} \mid S\texttt{a} \mid \texttt{z} \\ R &\rightarrow \texttt{b}T \\ T &\rightarrow R \mid S \end{aligned}$$

c) Left-factoring, inlining, and removing unused productions obtaining

$$\begin{aligned} S &\rightarrow \texttt{b}T\texttt{a} \mid S\texttt{a} \mid \texttt{z} \\ T &\rightarrow \texttt{b}T \mid S \end{aligned}$$

d) Removing left-recursion, left-factoring, introducing +/*, inlining, and removing unused productions obtaining

$$\begin{aligned} S &\rightarrow \texttt{b}T\texttt{a}^{+} \mid \texttt{za}^{*} \\ T &\rightarrow \texttt{b}T \mid S \end{aligned}$$

4 (5 points)**.** Suppose we have a parser *pExpr* :: *Parser Char Expr*, where the datatype *Expr* has a constructor *Let Identifier Expr Expr*. What is the type of the following parser combinator?

```
pDecl = Let <$  token "let"
            <*> identifier
             <* symbol '='
            <*> pExpr
             <* token "in"
            <*> pExpr
```

a) $\mathit{Parser\ Char\ (Identifier \to Expr \to Expr \to Expr)}$

b) $\mathit{Parser\ Char\ ((Identifier, Expr, Expr) \to Expr)}$

c) $\mathit{Parser\ Char\ (String \to Identifier \to Char \to Expr \to String \to Expr \to Expr)}$

d) $\mathit{Parser\ Char\ Expr}$

•

5 (5 points). The parser *sepBy p sep* parses one or more occurrences of *p* (for example, a parser for integers), separated by *sep* (for example, a parser for a comma).

$$\mathit{sepBy} :: \mathit{Parser\ Char\ a \to Parser\ Char\ b \to Parser\ Char\ [a]}$$

Which of the below definitions is the correct implementation of *sepBy*?

a) $\mathit{sepBy\ p\ sep} = (:) <\$> p <*> \mathit{option}\ ((\lambda x\ y \to y) <\$> \mathit{sep} <*> \mathit{sepBy\ p\ sep})\ []$

b) $\mathit{sepBy\ p\ sep} = (:) <\$> p <*> \mathit{many}_1\ ((\lambda x\ y \to y) <\$> \mathit{sep} <*> p)$

c) $\mathit{sepBy\ p\ sep} = (:) <\$> p <*> \mathit{sep} <*> \mathit{sepBy\ p\ sep} <|> \mathit{succeed}\ []$

d) $\mathit{sepBy\ p\ sep} = (:) <\$> p <*> \mathit{option}\ ((\lambda x\ y \to y) <\$> \mathit{sep} <*> p)\ []$

•

An AVL tree is a classical data structure, designed in 1962 by Georgy Adelson-Velsky and Evgenii Landis. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. The datatype *AVL* is defined as follows in the module *Data.Tree.AVL*.

```
data AVL e = E                      — Empty Tree
           | N (AVL e) e (AVL e)    — right height = left height + 1
           | Z (AVL e) e (AVL e)    — right height = left height
           | P (AVL e) e (AVL e)    — left height = right height + 1
```

6 (5 points). What is the algebra type for the datatype *AVL*?

a) **type** $\mathit{AVLAlg\ e\ r} = (r, r \to e \to r, r \to e \to r, r \to e \to r)$

b) **type** $AVLAlg\ r = (r, r \to r \to r \to r, r \to r \to r \to r, r \to r \to r \to r)$

c) **type** $AVLAlg\ e\ r = (r, r \to e \to r \to r, r \to e \to r \to r, r \to e \to r \to r)$

d) **type** $AVLAlg\ r = (r, r \to r \to r, r \to r \to r, r \to r \to r)$

•

7 (5 points). How do you define the function *foldAVL*, the standard *fold* on the datatype *AVL*?

a) $foldAVL\ (e, n, z, p) = fold$ **where**
$$\begin{aligned}
&fold\ E &&= e\\
&fold\ (N\ l\ m\ r) &&= n\ (fold\ l)\ (fold\ m)\ (fold\ r)\\
&fold\ (Z\ l\ m\ r) &&= z\ (fold\ l)\ (fold\ m)\ (fold\ r)\\
&fold\ (P\ l\ m\ r) &&= p\ (fold\ l)\ (fold\ m)\ (fold\ r)
\end{aligned}$$

b) $foldAVL\ (e, n, z, p) = fold$ **where**
$$\begin{aligned}
&fold\ E &&= e\\
&fold\ (N\ l\ m\ r) &&= n\ l\ m\ r\\
&fold\ (Z\ l\ m\ r) &&= z\ l\ m\ r\\
&fold\ (P\ l\ m\ r) &&= p\ l\ m\ r
\end{aligned}$$

c) $foldAVL\ (e, n, z, p) = fold$ **where**
$$\begin{aligned}
&fold\ E &&= e\\
&fold\ (N\ l\ m\ r) &&= n\ (fold\ l)\ m\ (fold\ r)\\
&fold\ (Z\ l\ m\ r) &&= z\ (fold\ l)\ m\ (fold\ r)\\
&fold\ (P\ l\ m\ r) &&= p\ (fold\ l)\ m\ (fold\ r)
\end{aligned}$$

d) $foldAVL\ (e, n, z, p) = fold$ **where**
$$\begin{aligned}
&fold\ E &&= e\\
&fold\ (N\ l\ m\ r) &&= n\ l\ (fold\ m)\ r\\
&fold\ (Z\ l\ m\ r) &&= z\ l\ (fold\ m)\ r\\
&fold\ (P\ l\ m\ r) &&= p\ l\ (fold\ m)\ r
\end{aligned}$$

•

8 (5 points)**.** The height of an *AVL* tree is an essential concept in *AVL* trees. How do you define the function *heightAVL* as a *foldAVL*?

a) $heightAVL = foldAVL\ (e, n, z, p)$ **where**
$$\begin{aligned} e\quad &= 0 \\ n\ l\ m\ r &= 1 + heightAVL\ r \\ z\ l\ m\ r &= 1 + heightAVL\ r \\ p\ l\ m\ r &= 1 + heightAVL\ l \end{aligned}$$

b) $heightAVL = foldAVL\ (e, n, z, p)$ **where**
$$\begin{aligned} e\quad &= 0 \\ n\ l\ m\ r &= 1 + max\ (heightAVL\ l)\ (heightAVL\ r) \\ z\ l\ m\ r &= 1 + max\ (heightAVL\ l)\ (heightAVL\ r) \\ p\ l\ m\ r &= 1 + max\ (heightAVL\ l)\ (heightAVL\ r) \end{aligned}$$

c) $heightAVL = foldAVL\ (e, n, z, p)$ **where**
$$\begin{aligned} e\quad &= 0 \\ n\ l\ m\ r &= 1 + r \\ z\ l\ m\ r &= 1 + r \\ p\ l\ m\ r &= 1 + l \end{aligned}$$

d) $heightAVL = foldAVL\ (e, n, z, p)$ **where**
$$\begin{aligned} e\quad &= 0 \\ n\ l\ m\ r &= 1 + foldAVL\ (e, n, z, p)\ r \\ z\ l\ m\ r &= 1 + foldAVL\ (e, n, z, p)\ r \\ p\ l\ m\ r &= 1 + foldAVL\ (e, n, z, p)\ l \end{aligned}$$

•

9 (5 points)**.** Suppose we have an *AVL*-tree with integers, and an environment that maps integers to strings. We want to replace the integers in the *AVL*-tree by their corresponding strings in the environment. You can use the function $lookup :: Env \to Int \to String$ to look up strings in the environment. Define the function

$$replace :: AVL\ Int \to Env \to AVL\ String$$

that replaces all integers in an *AVL*-tree by the strings to which they are bound in the environment.

a) $replace\ env = foldAVL\ (e, n, z, p)$ **where**
$e = E$
$n = \lambda l\ m\ r \rightarrow N\ l\ (lookup\ env\ m)\ r$
$z = \lambda l\ m\ r \rightarrow Z\ l\ (lookup\ env\ m)\ r$
$p = \lambda l\ m\ r \rightarrow P\ l\ (lookup\ env\ m)\ r$

b) $replace = foldAVL\ (e, n, z, p)$ **where**
$e = \lambda env \rightarrow E$
$n = \lambda env\ l\ m\ r \rightarrow N\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$
$z = \lambda env\ l\ m\ r \rightarrow Z\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$
$p = \lambda env\ l\ m\ r \rightarrow P\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$

c) $replace = foldAVL\ (e, n, z, p)$ **where**
$e = \lambda env \rightarrow E$
$n = \lambda l\ m\ r\ env \rightarrow N\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$
$z = \lambda l\ m\ r\ env \rightarrow Z\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$
$p = \lambda l\ m\ r\ env \rightarrow P\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$

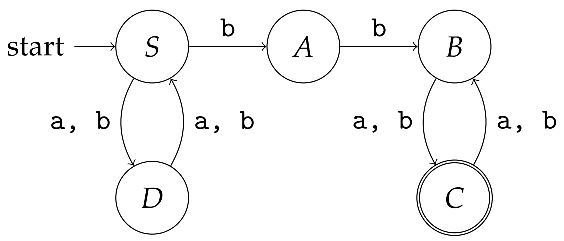
d) $replace\ env = foldAVL\ (e, n, z, p)$ **where**
$e = E$
$n = \lambda l\ m\ r \rightarrow N\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$
$z = \lambda l\ m\ r \rightarrow Z\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$
$p = \lambda l\ m\ r \rightarrow P\ (l\ env)\ (lookup\ env\ m)\ (r\ env)$

10 (5 points)**.** Consider the following language:

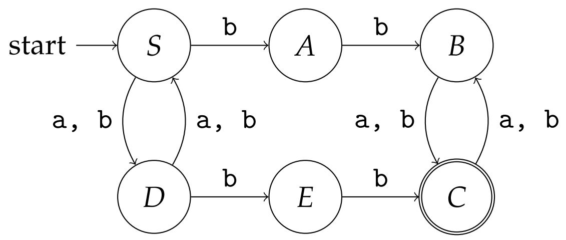
$$L \quad = \quad \{x \mid x \in \{\mathtt{a}, \mathtt{b}\}^*, \text{length } x \text{ is odd}, \mathtt{bb} \text{ is a substring of } x\}$$

Which of the following automata, with start state S, generates L?

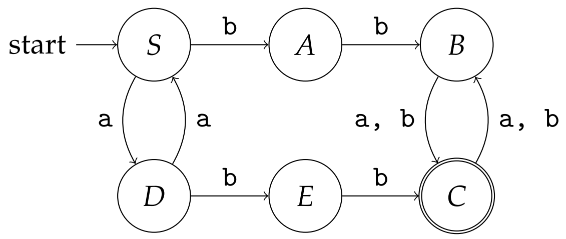
a)



b)



c)

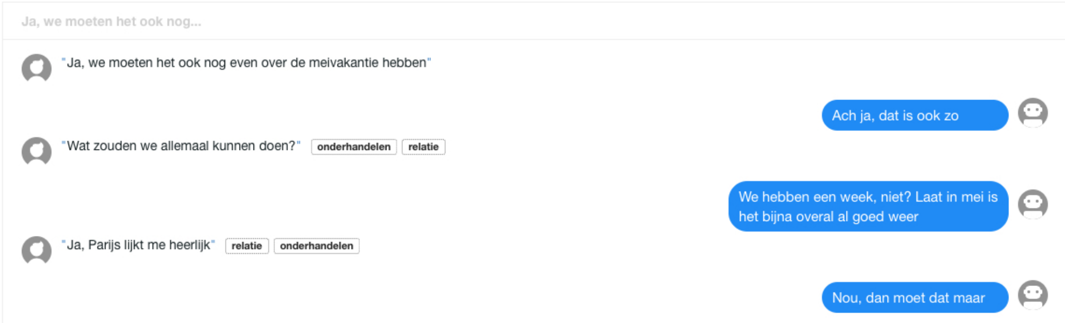


d) All three automata generate L.

•

Open answer questions

On `wit.ai` (nowadays owned by Facebook) you can create your own chatbots. Here is an example discussion with a chatbot I created on `wit.ai`:



The `wit.ai` website receives many chatbot discussions, and analyses these. To analyse a discussion, it has to be parsed. The concrete syntax of the above discussion looks as follows:

```
Client:
  Ja, we moeten het ook nog even over de meivakantie hebben
Bot:
  Ach ja, dat is ook zo
Client:
  Wat zouden we allemaal kunnen doen?
  {Onderhandelen=5
  ,relatie=5
  }
Bot:
  We hebben een week, niet? Laat in mei is het bijna overal al goed weer
Client:
  Ja, Parijs lijkt me heerlijk
  {Onderhandelen=-5
  ,relatie=-5
  }
Bot:
  Nou dan moet dat maar
```

A chatbot-discussion consists of a list of alternating statements between a Client and a Bot, where the Client starts the discussion. Each statement starts with an identifier of who speaks (Bot or Client), followed by a colon, followed by spaces and/or newlines, and then a sentence. The Client statements may be followed by scores on a number of parameters, where parameters and scores are separated by an '='. The scores are presented between braces { and }.

11 (15 points)**.** Give a concrete syntax (a context-free grammar) of this language for chatbot-discussions. You may use a non-terminal symbol called *String* to recognise the content of a sentence (a string not containing a newline), and a non-terminal called *Integer* to recognise a score. Describe the language as precisely as possible, but you may ignore occurrences of spaces (you may include them as well). •

12 (15 points)**.** Define an abstract syntax (a (data) type *Discussion* in Haskell) that corresponds to your concrete syntax given as an answer in Task 11, which you can use to represent a chatbot-discussion in Haskell. •

13 (20 points). Define a parser *pDiscussion* :: *Parser Char Discussion* that parses sentences from the language of chatbot-discussions. Define your parser using parser combinators. •