Department of Information and Computing Sciences
Utrecht University

INFOB3TC – Solutions for Exam 1

Johan Jeuring

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

Please keep in mind that there are often many possible solutions and that these example solutions may contain mistakes.

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:

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

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

a) yxyxxxyxx

b) xxxyyyxxx

c) yxyxyxyx

d) yxyxxxxxyxy

•

Solution 1. a). The number of y's has to be odd, and there is always an x beside a y. ○

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

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

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.

•

Solution 2. d).

Marking

○

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

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

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

a) Removing left-recursion obtaining

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

b) Left-factoring obtaining

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

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

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

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

$$\begin{array}{lcl} S & \rightarrow & \mathtt{b}T\mathtt{a}^{+} \mid \mathtt{za}^{*} \\ T & \rightarrow & \mathtt{b}T \mid S \end{array}$$

Solution 3. d).

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?

$$\begin{array}{lll} pDecl = Let & <\$ & token\ \texttt{"let"} \\ & <\!\ast\!> & identifier \\ & <\ast & symbol\ \texttt{'='} \\ & <\!\ast\!> & pExpr \\ & <\ast & token\ \texttt{"in"} \\ & <\!\ast\!> & pExpr \end{array}$$

a) $Parser\ Char\ (Identifier \rightarrow Expr \rightarrow Expr \rightarrow Expr)$

b) $Parser\ Char\ ((Identifier, Expr, Expr) \rightarrow Expr)$

c) $Parser\ Char\ (String \rightarrow Identifier \rightarrow Char \rightarrow Expr \rightarrow String \rightarrow Expr \rightarrow Expr)$

d) $Parser\ Char\ Expr$

Solution 4. d).

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).

$$sepBy :: Parser\ Char\ a \rightarrow Parser\ Char\ b \rightarrow Parser\ Char\ [a]$$

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

a) $sepBy\ p\ sep = (:) <\$> p <\!\ast\!> option\ ((\lambda x\ y \rightarrow y) <\$> sep <\!\ast\!> sepBy\ p\ sep)\ []$

b) $sepBy\ p\ sep = (:) <\$> p <\!\ast\!> many_1\ ((\lambda x\ y \rightarrow y) <\$> sep <\!\ast\!> p)$

c) $sepBy\ p\ sep = (:) <\$> p <\!\ast\!> sep <\!\ast\!> sepBy\ p\ sep <|> succeed\ []$

d) $sepBy\ p\ sep = (:) <\$> p <\!\ast\!> option\ ((\lambda x\ y \rightarrow y) <\$> sep <\!\ast\!> p)\ []$

Solution 5. a). ◦

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*.

$$
\begin{aligned}
&\textbf{data}\ AVL\ e = E && \text{— Empty Tree}\\
&\quad\mid N\ (AVL\ e)\ e\ (AVL\ e) && \text{— right height = left height + 1}\\
&\quad\mid Z\ (AVL\ e)\ e\ (AVL\ e) && \text{— right height = left height}\\
&\quad\mid P\ (AVL\ e)\ e\ (AVL\ e) && \text{— left height = right height + 1}
\end{aligned}
$$

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

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

b) $\textbf{type}\ AVLAlg\ r = (r, r \to r \to r \to r, r \to r \to r \to r, r \to r \to r \to r)$

c) $\textbf{type}\ AVLAlg\ e\ r = (r, r \to e \to r \to r, r \to e \to r \to r, r \to e \to r \to r)$

d) $\textbf{type}\ AVLAlg\ r = (r, r \to r \to r, r \to r \to r, r \to r \to r)$

•

Solution 6. c). ◦

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

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

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

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

d) $foldAVL\ (e, n, z, p) = fold$ **where**

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

•

Solution 7. c). ◦

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**

$$
\begin{aligned}
&e &&= 0\\
&n\ l\ m\ r &&= 1 + heightAVL\ r\\
&z\ l\ m\ r &&= 1 + heightAVL\ r\\
&p\ l\ m\ r &&= 1 + heightAVL\ l
\end{aligned}
$$

b) $heightAVL = foldAVL\ (e, n, z, p)$ **where**

$$
\begin{aligned}
&e &&= 0\\
&n\ l\ m\ r &&= 1 + max\ (heightAVL\ l)\ (heightAVL\ r)\\
&z\ l\ m\ r &&= 1 + max\ (heightAVL\ l)\ (heightAVL\ r)\\
&p\ l\ m\ r &&= 1 + max\ (heightAVL\ l)\ (heightAVL\ r)
\end{aligned}
$$

c) $heightAVL = foldAVL\ (e, n, z, p)$ **where**

$$
\begin{aligned}
&e &&= 0\\
&n\ l\ m\ r &&= 1 + r\\
&z\ l\ m\ r &&= 1 + r\\
&p\ l\ m\ r &&= 1 + l
\end{aligned}
$$

d) $heightAVL = foldAVL\ (e, n, z, p)$ **where**

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

•

Solution 8. c). ◦

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**
$e = E$
$n = \lambda l\ m\ r \to N\ l\ (lookup\ env\ m)\ r$
$z = \lambda l\ m\ r \to Z\ l\ (lookup\ env\ m)\ r$
$p = \lambda l\ m\ r \to P\ l\ (lookup\ env\ m)\ r$

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

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

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

•

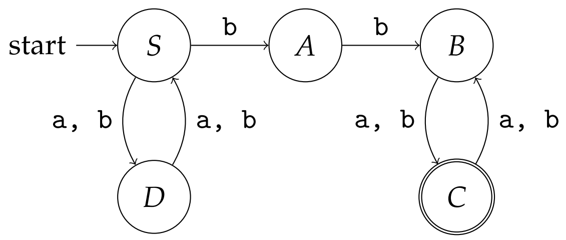
Solution 9. c). ○

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

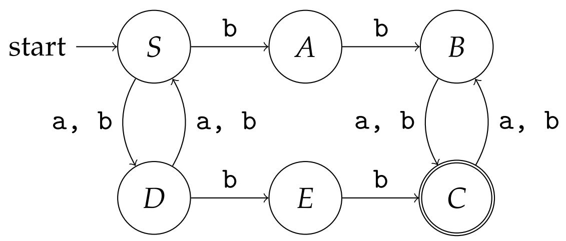
$$L = \{x \mid x \in \{\mathtt{a}, \mathtt{b}\}^*, \text{length } x \text{ is odd}, \mathtt{bb} \text{ is a substring of } x\}$$

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

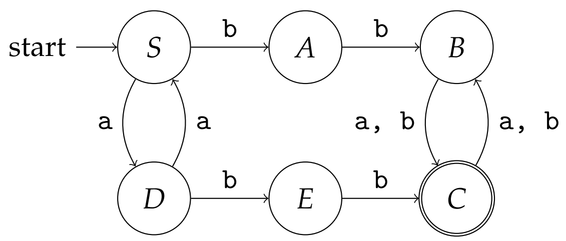
a)



b)



c)



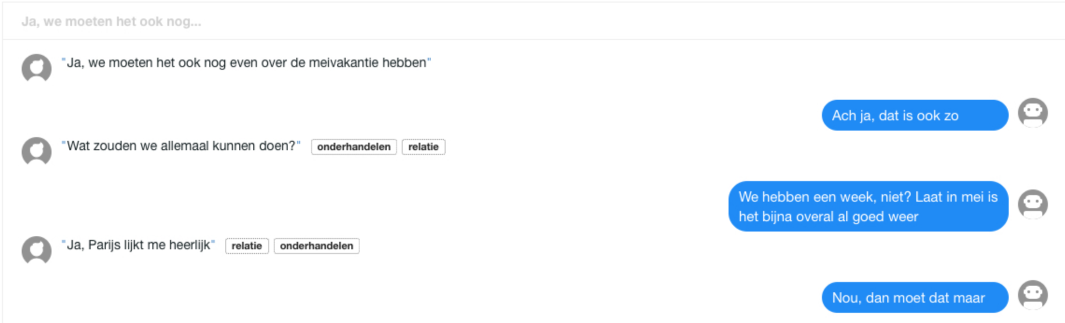
d) All three automata generate L.

•

Solution 10. b). (abb is not accepted by a, and all strings starting with ba are not accepted by c) ◦

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). •

Solution 11.

```
Discussion → (Client Bot)*
Client     → "Client:\n" String "\n" ("{" Scores "}\n") ?
Scores     → Score "\n" | Score "\n," Scores
Score      → Identifier "=" Integer
Bot        → "Bot:\n" String "\n"
```

Here is the above example sentence:

```
example = client1 ++ bot1 ++ client2 ++ bot2 ++ client3 ++ bot3
client1 = "Client:\n  Ja, we moeten het ook nog even over de meivakantie hebben\n"
bot1    = "Bot:\n  Ach ja, dat is ook zo\n"
client2 = "Client:\n  Wat zouden we allemaal kunnen doen?\n  " ++ "{" ++ scores2 ++ "}\n"
scores2 = "Onderhandelen=5\n  ,relatie=5\n  "
bot2    = "Bot:\n  We hebben een week, niet? " ++ bot2a
bot2a   = "Laat in mei is het bijna overal al goed weer\n"
client3 = "Client:\n  Ja, Parijs lijkt me heerlijk\n  " ++ "{" ++ scores3 ++ "}\n"
scores3 = "Onderhandelen=-5\n  ,relatie=-5\n  "
bot3    = "Bot:\n  Nou dan moet dat maar\n"
```

Marking

a (-1): Parameter defined as a *String* (should be an *Identifier*)
b (-1): No newlines between Bot and Client statements (inside the statements the newlines do not have to be present)
c (-3): A Bot statement may be followed by a score
d (-3): The Bot and CLient statement are not necessarily alternating
e (-2): The *Parameter* non-terminal is undefined
f (-2): Scores are not optional
g (-1): No comma's between scores
h (-1): No braces around scores
i (-1): Minor errors
j (-3): Bot: and Client: do not appear in the grammar
k (-5): Pretty printer instead of grammar
l (-2): The grammar only allows exactly two parameters
m (-1): Productions are not written with an →, but with an = or a :
n (-1): One comma too many in the scores
o (-4): Either a score or a sentence, but not both

p (-2): Scores appear after the Bot instead of the Client
q (-2): *Identifier* or *String* instead of Bot and Client
r (-3): The *Parameter*= part in the score is not described
s (-1): The : after Bot and CLient is not described
t (-1): The = in the score is not described
u (-2): Only two particular scores are modelled
v (-6): No keywords or characters are described
w (-3): Scores can be nested
x (-1): Client and Bot appear in the wrong order

○

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. •

Solution 12.

$$
\begin{array}{ll}
\textbf{type}\ \textit{Discussion} & = [(\textit{Client},\textit{Bot})] \\
\textbf{type}\ \textit{Client} & = (\textit{Sentence},\textit{Maybe Scores}) \\
\textbf{type}\ \textit{Sentence} & = \textit{String} \\
\textbf{type}\ \textit{Scores} & = [\textit{Score}] \\
\textbf{type}\ \textit{Score} & = (\textit{Identifier},\textit{Int}) \\
\textbf{type}\ \textit{Bot} & = \textit{String} \\
\textbf{type}\ \textit{Identifier} & = \textit{String}
\end{array}
$$

Marking

a (-2): *Identifier* instead of *String*
b (-2..-6): different syntactic errors, such as omitted tuple-parentheses/comma's; application of base types, etc
c (-3): **type**-definition has a constructor
d (-3): **type**-definition has a choice between constructors
e (-3): multiple constructors with the same name
f (-1): *Maybe* modelled with lists
g (-3): **data**-constructors considered types
h (-5): modelling concrete syntax for *Bot* and *Client* in a type
i (-3): *String* instead of *Int* for a score
j (-1): using **data** where **type** would have been better
k (-1..-10): miscellaneous mistakes
l (-1): *integer* instead of *Int*
m (-3): *Maybe* modelled with a separate datatype
n (-2..-10): not following the concrete syntax (often no alternating list anymore, but many other mistakes)
o (-3): **data** with no constructors

p (-2..-15): concrete syntax instead of abstract syntax
q (-5): *many* instead of list, *some* instead of a non-empty list
r (-5): no **data** or **type**

○

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. ●

Solution 13.

```
pDiscussion :: Parser Char Discussion
pDiscussion = many ((,) <$> pClient <*> pBot)
pClient     :: Parser Char Client
pClient     = (,)
            <$ tokensp "Client:\n"
            <*> pSentence
            <* tokensp "\n"
            <*> optional (pack (tokensp "{") pScores (tokensp "}\n"))
pBot        :: Parser Char Bot
pBot        = tokensp "Bot:\n"
            *> pSentence
            <* tokensp "\n"
pScores     :: Parser Char Scores
pScores     = listOf (pScore <* tokensp "\n") (tokensp ",")
pScore      :: Parser Char Score
pScore      = (,) <$> identifier <* symbol '=' <*> integersp
pSentence   :: Parser Char Identifier
pSentence   = greedy (satisfy (\c -> (c /= '\n')))
spaces      = greedy (satisfy (== ' '))
tokensp s   = token s <* spaces
integersp   = integer <* spaces
-- Parser test case
test        = fst $ head $ pDiscussion example
```

Marking
a (-1..-5): Type errors when building up the abstract syntax
a1 (-3): (:) <$> *many* ...
a2 (-3): using a datatype instead of a constructor when constructing abstract syntax
b (-1..-10): does not follow the concrete syntax
b1 (-3): optional (non-)terminals not represented optionally
b2 (-1..-2): forgetting newlines etc

c (-1..-5): typos, obvious confusion
c1 (-2): *option* misses second argument
d (-1..-5): erroneous usage of parser combinators