Department of Information and Computing Sciences
Utrecht University

INFOB3TC – Solutions for Exam 3

Sean Leather

Monday, 12 March 2012, 14:00 – 17:00

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

1 Questions

1.1 Grammar Transformation

1 (20 points)**.** Consider the following grammar over the alphabet $\{\mathtt{m}, \mathtt{n}, \mathtt{8}, \mathtt{\#}, \mathtt{@}\}$:

$$\begin{aligned} S &\rightarrow \mathtt{m}\,\mathtt{8}\,P \mid \mathtt{m}\,\mathtt{8}\,P\,S \\ P &\rightarrow P\,Q\,\mathtt{\#} \mid \mathtt{n} \\ Q &\rightarrow \mathtt{@}\,P \end{aligned}$$

Transform this into a minimal grammar from which we can immediately derive the simplest and most efficient parser. You may use any of the following transformations:

Inline nonterminal	Remove duplicate productions
Introduce nonterminal	Remove left-recursion
Introduce $\cdot^*$	Remove unreachable production
Introduce $\cdot^+$	Left-factoring
Introduce $\cdot?$	

•

Solution 1.

Begin with initial grammar:

$$\begin{aligned} S &\rightarrow \mathtt{m}\,\mathtt{8}\,P \mid \mathtt{m}\,\mathtt{8}\,P\,S \\ P &\rightarrow P\,Q\,\mathtt{\#} \mid \mathtt{n} \\ Q &\rightarrow \mathtt{@}\,P \end{aligned}$$

Introduce $\cdot^+$:

$$\begin{aligned} S &\rightarrow (\mathtt{m}\,\mathtt{8}\,P)^+ \\ P &\rightarrow P\,Q\,\mathtt{\#} \mid \mathtt{n} \\ Q &\rightarrow \mathtt{@}\,P \end{aligned}$$

Remove left-recursion:

$$\begin{aligned} S &\rightarrow (\texttt{m}\,\texttt{8}\,P)^+ \\ P &\rightarrow \texttt{n} \mid \texttt{n}\,R \\ R &\rightarrow Q\,\texttt{\#} \mid Q\,\texttt{\#}\,R \\ Q &\rightarrow \texttt{@}\,P \end{aligned}$$

Introduce $\cdot^+$:

$$\begin{aligned} S &\rightarrow (\texttt{m}\,\texttt{8}\,P)^+ \\ P &\rightarrow \texttt{n} \mid \texttt{n}\,R \\ R &\rightarrow (Q\,\texttt{\#})^+ \\ Q &\rightarrow \texttt{@}\,P \end{aligned}$$

Introduce $\cdot?$:

$$\begin{aligned} S &\rightarrow (\texttt{m}\,\texttt{8}\,P)^+ \\ P &\rightarrow \texttt{n}\,R? \\ R &\rightarrow (Q\,\texttt{\#})^+ \\ Q &\rightarrow \texttt{@}\,P \end{aligned}$$

Inline nonterminal:

$$\begin{aligned} S &\rightarrow (\texttt{m}\,\texttt{8}\,P)^+ \\ P &\rightarrow \texttt{n}\,(Q\,\texttt{\#})^+? \\ Q &\rightarrow \texttt{@}\,P \end{aligned}$$

Introduce $\cdot^*$:

$$\begin{aligned} S &\rightarrow (\texttt{m}\,\texttt{8}\,P)^+ \\ P &\rightarrow \texttt{n}\,(Q\,\texttt{\#})^* \\ Q &\rightarrow \texttt{@}\,P \end{aligned}$$

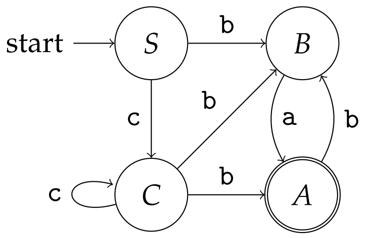
Inline nonterminal:

$$\begin{aligned} S &\rightarrow (\texttt{m}\,\texttt{8}\,P)^+ \\ P &\rightarrow \texttt{n}\,(\texttt{@}\,P\,\texttt{\#})^* \end{aligned}$$

○

1.2 Regular Languages

2 (20 points)**.** Consider the following nondeterministic finite state automaton (NFA):



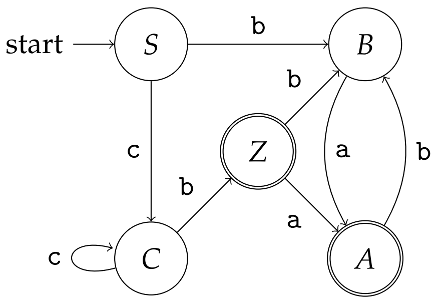
(a) Construct a deterministic finite state automaton (DFA) from the NFA.

(b) Give a regular grammar for the DFA.

●

Solution 2.

(a)



(b)

$$\begin{aligned}
S &\rightarrow \texttt{b}\,B \\
S &\rightarrow \texttt{c}\,C \\
B &\rightarrow \texttt{a}\,A \\
A &\rightarrow \texttt{b}\,B \\
A &\rightarrow \varepsilon \\
C &\rightarrow \texttt{c}\,C \\
C &\rightarrow \texttt{b}\,Z \\
Z &\rightarrow \texttt{b}\,B \\
Z &\rightarrow \texttt{a}\,A \\
Z &\rightarrow \varepsilon
\end{aligned}$$

○

1.3 LL Parsing

3 (20 points)**.** Consider the grammar in the table below:

NT	Production	*empty*	*emptyRhs*	*first*	*firstRhs*	*follow*	*lookAhead*
S							
	$S \rightarrow \texttt{1}\,B\,\texttt{9}$						
B							
	$B \rightarrow \texttt{2}\,C$						
C							
	$C \rightarrow B\,\texttt{3}$						
	$C \rightarrow \texttt{5}$						

(a) Complete the table by computing the values in the columns for the appropriate rows. Use *True* and *False* for property values and set notation for everything else.

(b) Is the grammar LL(1)? Explain how you arrived at your answer. If it is is not, transform the grammar so that it is LL(1). Give the rows of the table that differ. You don't need to write the whole table again.

•

Solution 3.

(a) The table:

NT	Production	*empty*	*emptyRhs*	*first*	*firstRhs*	*follow*	*lookAhead*
S		*False*		{1}		ε	
	$S \rightarrow 1\,B\,9$		*False*		{1}		{1}
B		*False*		{2}		{9,3}	
	$B \rightarrow 2\,C$		*False*		{2}		{2}
C		*False*		{2,5}		{9,3}	
	$C \rightarrow B\,3$		*False*		{2}		{2}
	$C \rightarrow 5$		*False*		{5}		{5}

(b) The grammar is LL(1).

○

1.4 Parser Combinators

4 (20 points)**.** The following grammar describes JSON, the JavaScript Object Notation, a data-interchange format.

$$
\begin{array}{lcl}
\textit{Object} & \rightarrow & \{\,\} \mid \{\,\textit{Members}\,\} \\
\textit{Members} & \rightarrow & \textit{Pair} \mid \textit{Pair}\ ,\ \textit{Members} \\
\textit{Pair} & \rightarrow & \textit{String}\ :\ \textit{Value} \\
\textit{Value} & \rightarrow & \textit{String} \mid \textit{Number} \mid \textit{Object} \mid \textit{Array} \mid \texttt{true} \mid \texttt{false} \mid \texttt{null} \\
\textit{Array} & \rightarrow & [\,] \mid [\,\textit{Elements}\,] \\
\textit{Elements} & \rightarrow & \textit{Value} \mid \textit{Value}\ ,\ \textit{Elements}
\end{array}
$$

For *String* and *Number*, you may assume the standard Haskell *String* and *Double* formats and values.

(a) Give an abstract syntax for the grammar using a family of Haskell datatypes.

(b) Define parsers for the datatypes using the parser combinators covered in the course.

(c) Define the algebra type and fold functions for the datatypes.

•

Solution 4.

(a)

```
data Object   = Object Members
type Members  = [Pair]
data Pair     = Pair String Value
data Value    = StringV String
              | NumberV Double
              | ObjectV Object
              | ArrayV Elements
              | TrueV
              | FalseV
              | NullV
type Elements = [Value]
```

(b) Bonus: handling white space, implementing *string*, implementing *double*.

```
space :: Parser Char String
space = greedy (satisfy isSpace)
string :: Parser Char String
string = pack (symbol '"') (many (satisfy isAlphaNum)) (symbol '"') <* space
double :: Parser Char Double
double = sign <*> unsigned <* space
sign :: Parser Char (Double → Double)
sign = (const negate <$> symbol '-') `option` id
unsigned :: Parser Char Double
unsigned = (λn f → read n + f) <$> many₁ digit <*> (fraction `option` 0)
fraction :: Parser Char Double
fraction = (λp n → read ('0' : p : n)) <$> symbol '.' <*> many₁ digit
list :: Parser Char a → Parser Char b → Parser Char [a]
list p s = (:) <$> p <*> greedy (s *> space *> p) <<|> succeed []
value :: Parser Char Value
value  =  StringV       <$> string
      <|> NumberV       <$> double
      <|> ObjectV       <$> object
      <|> ArrayV        <$> elements
      <|> const TrueV   <$> token "true" <* space
      <|> const FalseV  <$> token "false" <* space
```

```
    <|> const NullV <$> token "null" <* space
object :: Parser Char Object
object = Object <$> braced (list pair (symbol ',')) <* space
pair :: Parser Char Pair
pair = Pair <$> string <* (symbol ':' <* space) <*> value
elements :: Parser Char Elements
elements = bracketed (list value (symbol ',')) <* space
```

(c)

```
type Alg o m p v e =
  (m → o
  , (p → m → m, m)
  , String → v → p
  , (String → v, Double → v, o → v, e → v, v, v, v)
  , (v → e → e, e)
  )
foldO :: Alg o m p v e → Object → o
foldO alg@(m, _, _, _, _) (Object mems) = m (foldM alg mems)
foldM :: Alg o m p v e → Members → m
foldM alg@(_, (cons, _), _, _, _) (p : ps) = cons (foldP alg p) (foldM alg ps)
foldM alg@(_, (_, nil), _, _, _)  []       = nil
foldP :: Alg o m p v e → Pair → p
foldP alg@(_, _, p, _, _) (Pair s v) = p s (foldV alg v)
foldV :: Alg o m p v e → Value → v
foldV alg@(_, _, _, (sv, _, _, _, _, _, _), _) (StringV s)  = sv s
foldV alg@(_, _, _, (_, nv, _, _, _, _, _), _) (NumberV n)  = nv n
foldV alg@(_, _, _, (_, _, ov, _, _, _, _), _) (ObjectV o)  = ov (foldO alg o)
foldV alg@(_, _, _, (_, _, _, av, _, _, _), _) (ArrayV e)   = av (foldE alg e)
foldV alg@(_, _, _, (_, _, _, _, tv, _, _), _) TrueV        = tv
foldV alg@(_, _, _, (_, _, _, _, _, fv, _), _) FalseV       = fv
foldV alg@(_, _, _, (_, _, _, _, _, _, nv), _) NullV        = nv
foldE :: Alg o m p v e → Elements → e
foldE alg@(_, _, _, _, (cons, _))  (v : vs) = cons (foldV alg v) (foldE alg vs)
foldE alg@(_, _, _, _, (_, nil))   []       = nil
```

○

1.5 Context-Free Grammars

5 (20 points)**.** Consider the following grammar G over the alphabet $\{$ `0`, `a`, `b`, $[$, $]$ $\}$:

$$
\begin{aligned}
S &\rightarrow \texttt{0} \mid \texttt{00} \mid \texttt{[}\, T\, \texttt{]} \\
R &\rightarrow \texttt{a} \mid \texttt{b} \\
T &\rightarrow \varepsilon \mid R\, R\, T
\end{aligned}
$$

(a) Is the grammar context-free? Regular? Why?

(b) Give the language $L(G)$ of the grammar G without referring to G itself.

(c) Is the language $L(G)$ context-free? Regular? Why?

●

Solution 5.

(a) The grammar is context-free because each production has the form $A \rightarrow a$ where A is a nonterminal and a is a sequence of terminals and nonterminals. The grammar is not regular because the production $T \rightarrow R\,R\,T$ is not of the form $A \rightarrow x\,B$ or $A \rightarrow x$ where x is a sequence of terminals.

(b) $L(G) = \texttt{0} \cup \texttt{00} \cup \{\, \texttt{[}\, (s\, s)^*\, \texttt{]} \mid s \in \texttt{a} \cup \texttt{b} \}$

(c) The language is regular because we can define a regular expression for it:

$$\texttt{0} + \texttt{00} + \texttt{[}\, ((\texttt{a} + \texttt{b})\, (\texttt{a} + \texttt{b}))^*\, \texttt{]}$$

Since the language is regular, it is also context-free.

○