Department of Information and Computing Sciences
Utrecht University

INFOB3TC – Solutions for Exam 1

Sean Leather, Johan Jeuring

Thursday, 13 December 2012, 08:30–10:30

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

Questions

Grammar Analysis and Transformation

Consider the following context-free grammar over the alphabet $\{\texttt{a}, \texttt{d}, \texttt{e}, \texttt{i}, \texttt{r}, \texttt{s}, \texttt{v}\}$ with the start symbol X:

$$
\begin{aligned}
X &\to \texttt{a}Y\texttt{se} \\
Y &\to Y\texttt{i} \mid Y\texttt{er} \mid Z\texttt{v} \\
Z &\to \texttt{d}
\end{aligned}
$$

1 (15 points)**.** Describe a sequence of transformations for simplifying this grammar. The resulting grammar should be minimal and suitable for deriving a parser (using parser combinators). The grammar should not be ambiguous and should not result in inefficiency or nontermination in the parser.

You may use any of the transformations in the following list or another transformation discussed during the lecture or in the lecture notes.

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

For each transformation step in the sequence, describe the transformation and give the transformed grammar. You may use at most two transformations in one step, but you must mention both of them (e.g. "Inline S and remove unreachable production").

•

Solution 1.

Initial grammar:

$$X \to \texttt{a}Y\texttt{se}$$
$$Y \to Y\texttt{i} \mid Y\texttt{er} \mid Z\texttt{v}$$
$$Z \to \texttt{d}$$

Inline Z and remove unreachable production:

$$X \to \texttt{a}Y\texttt{se}$$
$$Y \to Y\texttt{i} \mid Y\texttt{er} \mid \texttt{dv}$$

Left-factor Y:

$$X \to \texttt{a}Y\texttt{se}$$
$$Y \to YI \mid \texttt{dv}$$
$$I \to \texttt{i} \mid \texttt{er}$$

Remove left-recursion of Y:

$$X \to \texttt{a}Y\texttt{se}$$
$$Y \to \texttt{dv} \mid \texttt{dv}Z$$
$$Z \to I \mid IZ$$
$$I \to \texttt{i} \mid \texttt{er}$$

Introduce $Z?$ and I^+:

$$X \to \texttt{a}Y\texttt{se}$$
$$Y \to \texttt{dv}Z?$$
$$Z \to I^+$$
$$I \to \texttt{i} \mid \texttt{er}$$

Inline Z and remove unreachable production:

$$X \to \texttt{a}Y\texttt{se}$$
$$Y \to \texttt{dv}(I^+)?$$
$$I \to \texttt{i} \mid \texttt{er}$$

Introduce I^*:

$$X \to \texttt{a}Y\texttt{se}$$
$$Y \to \texttt{dv}I^*$$
$$I \to \texttt{i} \mid \texttt{er}$$

Inline Y and remove unreachable production:

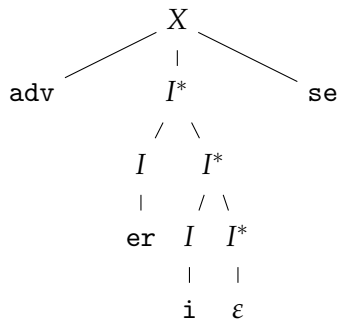
$$X \to \texttt{adv}I^*\texttt{se}$$
$$I \to \texttt{i} \mid \texttt{er}$$

∘

2 (5 points)**.** Choose a word w that is in the language and at least 8 symbols in length. Give the parse tree for w with either the original or simplified grammar. •

Solution 2.

$w =$ `adverise`



◦

3 (5 points)**.** Define a parser using parser combinators for the simplified grammar from the previous part. The input and output are lists of characters. •

Solution 3.

$px = (\lambda x\ y\ z \to x +\!\!+ concat\ y +\!\!+ z)$ <$> *token* `"adv"` <*> *many pi* <*> *token* `"se"`
$pi =$ *token* `"i"` <|> *token* `"er"`

◦

Datatypes and Semantic Functions

A certain file format describes documents containing a list of houses followed by a list of trees. A house has a name (a string) and a color (a string). A tree has a name and the name of the house to which the tree belongs.

4 (5 points)**.** Give an abstract syntax (as a family of datatypes) for the file format. •

Solution 4.

data *File* = *File* [*House*] [*Tree*]
data *House* = *House String String*
data *Tree* = *Tree String String*

◦

5 (10 points)**.** Define the algebra type and fold for the file format datatype. •

Solution 5.

```
type FileAlgebra f h t =
  ([h] → [t] → f          — File
  , String → String → h   — House
  , String → String → t   — Tree
  )

foldFile :: FileAlgebra f h t → File → f
foldFile (f, h, t) = ff
  where ff (File hs ts)       = f (map fh hs) (map ft ts)
        fh (House nm clr)     = h nm clr
        ft (Tree nm owner)    = t nm owner
```

○

6 (10 points)**.** Define the semantic function *treesOfGreen* using an algebra and the fold. This function produces a list of the names of trees owned by `"green"` houses in a given file. •

Solution 6.

```
type Houses = [String]
type Trees  = [String]
greenHouseAlgebra :: FileAlgebra Trees Houses (Houses → Trees)
greenHouseAlgebra =
  (λhs ts → concat (map ($concat hs) ts)
  , λnm clr → if clr == "green" then [nm] else []
  , λnm owner → λhouses → if nm `elem` houses then [owner] else []
  )

treesOfGreen :: File → Trees
treesOfGreen = foldFile greenHouseAlgebra
```

○

Parser Combinators

We have learned about parser combinators defined with the following type:

```
type Parser s a = [s] → [(a, [s])]
```

The input to a parser is a stream of symbols, and the output is a list of successes where each result is paired with its unconsumed input. Failure to parse is indicated with an empty result list.

Let us extend this parser to one that gives a bit more information about failure, namely the position of the failure-inducing symbol in the input stream. We indicate position as an *Int*:

```
type Pos = Int
```

The parser type now looks like this:

```
type Parser s a = Pos → [s] → Either Pos [(a, Pos, [s])]
```

Whenever a parser fails, it will return *Left pos* where *pos* is the maximum index into the input ([*s*]) where parsing failed. A successful parse will advance the resulting position by the number of symbols consumed.

We can run the parser with *parse* and initialize the position with 0:

```
parse :: Parser s a → [s] → Either String a
parse p inp = case p 0 inp of
  Left  pos           → Left ("Error at position " ++ show pos)
  Right ((x, _, _): _) → Right x
```

Use the new *Parser* type for the following questions.

7 (5 points)**.** Define the parser combinator *failp*. •

Solution 7.

```
failp :: Parser s a
failp pos _ = Left pos
```

◦

8 (10 points)**.** Define the parser combinator *satisfy*. •

Solution 8.

```
satisfy :: (s → Bool) → Parser s s
satisfy p pos (x : xs) | p x = Right [(x, succ pos, xs)]
satisfy _ pos _              = Left pos
```

◦

9 (10 points)**.** Define the parser combinator (≪|>) (left-biased greedy version of (<|>)). •

Solution 9.

```
(≪|>) :: Parser s a → Parser s a → Parser s a
(p ≪|> q) pos xs = case p pos xs of
                     Left _ → q pos xs
                     r      → r
```

◦

Context-Free Grammars

Let $B = \{\mathtt{0}, \mathtt{1}\}$. Give context-free grammars for the languages in the following questions, if possible. If not, explain why.

10 (5 points)**.** $L_1 = \{ b^2 b^R \mid b \in B^* \}$ where s^R is the reverse of s. •

Solution 10. Language L_1 is not context-free. ○

11 (10 points)**.** The set of all sequences $s \in B^*$ with no consecutive 1s. In other words, the sequence 11 is not a subsequence of s. •

Solution 11.
Several options:

$$S \to (\mathtt{1?0})^*\mathtt{1?}$$

$$S \to \mathtt{0}^*(\mathtt{10}^+)^*\mathtt{1?}$$

$$S \to \mathtt{0}S \mid \mathtt{10}S \mid \mathtt{1} \mid \varepsilon$$

A dual of one of these is also acceptable.

○