Department of Information and Computing Sciences
Utrecht University

INFOB3TC – Exam 1

Sean Leather, Johan Jeuring

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

Preliminaries

- The exam consists of 3 pages (including this page). Please verify that you got all the pages.
- Write your **name** and **student number** on all submitted work. Also include the total number of separate sheets of paper.
- The maximum score is stated at the top of each question. The total amount of points you can get is 90.
- Try to give simple and concise answers. Write readable text. Do not use pencils or pens with red ink.
- Please write your text in 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!

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:

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

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"). •

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. •

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. •

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. •

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

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. •

Parser Combinators

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

$$\textbf{type}\ \mathit{Parser}\ s\ a = [s] \to [(a, [s])]$$

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*:

$$\textbf{type}\ \mathit{Pos} = \mathit{Int}$$

The parser type now looks like this:

$$\textbf{type}\ \mathit{Parser}\ s\ a = \mathit{Pos} \to [s] \to \mathit{Either}\ \mathit{Pos}\ [(a, \mathit{Pos}, [s])]$$

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*. •

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

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

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. •

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