Department of Information and Computing Sciences
Utrecht University

INFOB3TC – Exam 1

Sean Leather

Thursday, 15 December 2010, 08:30–10:30

Preliminaries

- The exam consists of 4 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 100.
- 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

Context-Free Grammars

1 (20 points)**.** Consider the following language definitions:

(a) $L_1 = \{ a\,w \mid a \in A^2 \wedge w \in A^* \wedge |w| > 0 \}$ where $A = \{ \mathtt{t}, \mathtt{u}, \mathtt{v} \}$

(b) L_2 is the language defined by the following grammar over the alphabet $\{ \mathtt{a}, \mathtt{b}, \mathtt{z} \}$:

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

(c) $L_3 = \{ \mathtt{<}\,t\,\mathtt{>}\,c\,\mathtt{<}\,/\,t\,\mathtt{>} \mid t \in L_1 \wedge c \in L_2 \}$

If possible, give another definition of the language using one of the following approaches: an enumeration, a context-free grammar, or a predicate. Do not use the same approach that is used in the question.

If you cannot give any alternative definitions, explain why the other approaches do not work.

Note that $|w|$ denotes the length of the word w.

●

Grammar Analysis and Transformation

2 (30 points)**.** Consider the following context-free grammar over the alphabet $\{ \mathtt{?}, \mathtt{/}, \mathtt{a}, \mathtt{b} \}$:

$$\begin{aligned} S &\rightarrow \mathtt{?}\;T \mid \mathtt{a} \mid \mathtt{b} \\ T &\rightarrow S \mid S\;\mathtt{/}\;S \end{aligned}$$

(a) This grammar presents a variant of the "dangling `else`" problem. Describe the problem in full, including an example of a problematic sentence for this grammar and an explanation of why the example is problematic.

(b) If you were the designer of this language, how would you solve this problem? There are multiple solutions. Describe one possible solution in full, including any new grammars as necessary. Show how your example, or some modification thereof, is no longer problematic.

●

3 (20 points)**.** Transform the grammar below into a minimal grammar from which we can immediately derive the simplest and most efficient parser.

$$\begin{aligned} S &\rightarrow S\;a\;R \\ S &\rightarrow T \end{aligned}$$

$T \rightarrow a\ R$
$R \rightarrow b\ c$
$P \rightarrow S\ S$

Name each transformation step and show the grammar after that transformation. 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 $\cdot?$	

●

Parsing Grammar Descriptions

4 (30 points)**.** There are many other notations for describing context-free (EBNF) grammars than the one we have used in the course. The grammar below describes one of those notations.

$$
\begin{array}{lll}
\mathit{Production} & \rightarrow & \mathit{Name}\ \texttt{=}\ \mathit{Expression}?\ \texttt{.} \\
\mathit{Expression} & \rightarrow & \mathit{Alternative}\ (\texttt{|}\ \mathit{Alternative})^* \\
\mathit{Alternative} & \rightarrow & \mathit{Term}^+ \\
\mathit{Term} & \rightarrow & \mathit{Name} \mid \mathit{Token}\ (\ldots\ \mathit{Token})? \mid \mathit{Group} \mid \mathit{Option} \mid \mathit{Repetition} \\
\mathit{Group} & \rightarrow & \texttt{(}\ \mathit{Expression}\ \texttt{)} \\
\mathit{Option} & \rightarrow & \texttt{[}\ \mathit{Expression}\ \texttt{]} \\
\mathit{Repetition} & \rightarrow & \{\ \mathit{Expression}\ \}
\end{array}
$$

In this grammar for a language of grammars, production rules are described with a name and an optional expression and end with a dot. An expression is a nonempty sequence of alternatives with a `|` separator. An alternative is a nonempty sequence of terms. A term can be either a name, a single token, a range of tokens with minimum and maximum separated by a two dots, a subexpression, an optional expression, and a possibly empty sequenced expression.

(a) The grammar for *Name* is a word with an initial uppercase character from the Latin alphabet followed by any number of alphabetic or numeric terminals. Give the necessary productions, the type (either as a datatype or a type synonym), and the parser (by using combinators you know and/or defining a new one) for *Name*.

(b) The grammar for *Token* is a sequence of alphanumeric or whitespace terminals between double quotes. Given the necessary productions, the type, and the parser for *Token*.

(c) Give the abstract syntax for the grammar of productions in the form of a family of Haskell types.

(d) Define an efficient parser using the above grammar and abstract syntax.

(e) Define the algebra type and fold for the datatype used for *Production*.

(f) Using the above fold, define the following semantic functions.

   a) The function *tokens* collects all possible tokens of a production into a list of tokens. Assume that you can enumerate all tokens in a range with a function *enumRange* which takes two tokens and results in a list of tokens.

   b) The function *productions* splits the alternatives of a production into multiple productions. It produces a list of productions where each new production has the same name as the input production and only one of the alternatives.

•