

# Exercises for Language Processors (WS2001-2002) \*

Prof. Dr. Peter Thiemann  
Simon Helsen

## 5 Exercise Sheet

*Handout:* 25.1.2002    *Return:* 29.1.2002    *Total marks:* 20 points

### 5.1 Validation of SGML

(10 points) An SGML-DTD describes how an SGML-tree should look like for a particular application. The nodes of such a tree are elements and its leaves text data or *EMPTY*. As with XML, every element is allowed to have one or more sub-trees, described by a regular expression. A document is *valid* if —at each node— the sequence of the element names of the child trees belongs to the language of the regular expression. Consider the following DTD:

```
<!-- Lunch at the steakhouse -->
<!ELEMENT lunch (meal)+ -- one meal per person -->
<!ELEMENT meal (appetiz?, steak, custname) +(drink)>
<!-- The plus sign after the content model followed by
one or more elements within parentheses declares an
"inclusion". An inclusion indicates that the elements
can appear anywhere in the element to which they are
attached and in any of its subelements. You can have
one or more DRINK elements any time during MEAL.
(This feature is not available in XML.)
-->
<!ELEMENT appetiz (soup | salad) >
<!ELEMENT soup EMPTY --soup of the day -->
<!ELEMENT salad EMPTY>
<!ELEMENT steak EMPTY>
<!ELEMENT drink (beer|cola)>
<!ELEMENT (cola|beer) EMPTY>
<!ELEMENT custname (#PCDATA) >
```

---

\*©2002: it is not permitted to lend, distribute or copy any of these exercises and their model solutions either by paper or in electronic form other than for personal use. The authors cannot be held responsible for mistakes or inaccuracies in either formulation or solution of the exercises.

For this DTD, we define the following context-free grammar, where you can assume that  $n$  is some string and hence, a terminal-symbol:

$$\begin{aligned}
 E &\rightarrow \langle \text{lunch} \rangle L \langle / \text{lunch} \rangle \\
 &\quad | \langle \text{meal} \rangle L \langle / \text{meal} \rangle \\
 &\quad | \langle \text{appetiz} \rangle L \langle / \text{appetiz} \rangle \\
 &\quad | \langle \text{soup} \rangle L \langle / \text{soup} \rangle \\
 &\quad | \langle \text{salad} \rangle L \langle / \text{salad} \rangle \\
 &\quad | \langle \text{steak} \rangle L \langle / \text{steak} \rangle \\
 &\quad | \langle \text{drink} \rangle L \langle / \text{drink} \rangle \\
 &\quad | \langle \text{cola} \rangle L \langle / \text{cola} \rangle \\
 &\quad | \langle \text{beer} \rangle L \langle / \text{beer} \rangle \\
 &\quad | \langle \text{custname} \rangle n \langle / \text{custname} \rangle \\
 \\
 L &\rightarrow \epsilon \mid EL
 \end{aligned}$$

Not every document accepted by the above grammar is a valid for our DTD. Design an attribute grammar, which calculates whether a document is valid, *i.e.*, all sub-elements are conform the regular expressions in the DTD. Note that **drinks** need special treatment!

*Hint:* use the derivative function for regular expressions.

## 5.2 LAG(k) Attributed Grammars

- An attribute grammar in Bochmann normal form is LAG(1) if for all productions  $\pi = B \rightarrow B_1 \dots B_n$  and attributions

$$(\pi, j).b = \dots (\pi, i).a \dots$$

where  $i, j \in \{\varepsilon, 1, \dots, n\}$  we have the following restrictions:

- $j = \varepsilon$  or;
- $i < j$  (lexicographically).

This is a *left-attributed* grammar since all attributes only depend on attributes to the left. In such a grammar, all attribute values can be determined in one depth-first left-to-right traversal of the derivation tree.

- An attribute grammar is LAG(k) if for some  $k$ , the attribute set  $\mathcal{A}$  can be  $k$ -partitioned, *i.e.*:

$$\mathcal{A} = \mathcal{A}_1 \cup \dots \cup \mathcal{A}_k \text{ and } \mathcal{A}_l \cap \mathcal{A}_m = \emptyset \text{ if } l, m \in \{1, \dots, k\} \text{ and } l \neq m$$

where we have the following restrictions on the equations, and also  $a \in \mathcal{A}_u$  and  $b \in \mathcal{A}_v$ :

- $u < v$
- $u = v$  and  $j = \varepsilon$  or;

–  $u = v$  and  $i < j$ .

An LAG( $k$ ) attribute grammar needs  $k$  depth-first left-to-right traversals of the derivation tree to calculate all attribute values, where the  $v$ th traversal computes the values of the attributes in  $\mathcal{A}_v$ .

1. (5 points) Consider the following attribute grammar with  $\mathcal{A} = \{a, b, c, d\}$ :

$$\begin{array}{lll} \pi_1 : Z \rightarrow X & X.b = 1 & \pi_3 : X \rightarrow s \quad X.a = X.b \\ \pi_2 : X \rightarrow WX & X.a = W.c & \pi_4 : W \rightarrow t \quad W.c = W.d \\ & X_1.b = X.b & \\ & W.d = X_1.a & \end{array}$$

Does there exist a  $k$  which makes the grammar LAG( $k$ )? Explain.

2. (5 points) Consider the following attribute grammar with  $\mathcal{A} = \{a, b, c, d, e, f, g\}$ :

$$\begin{array}{lll} \pi_1 : Z \rightarrow X & X.b = 1 & \pi_2 : X \rightarrow WXY \quad X.a = W.c + X_1.a \\ \pi_3 : X \rightarrow s & X.a = X.b & X.e = Y.g \\ & X.e = X.b & X_1.b = X.b \\ \pi_4 : W \rightarrow t & W.c = W.d & W.d = X_1.e \\ \pi_5 : Y \rightarrow u & Y.g = Y.f & Y.f = X_1.e \end{array}$$

Does there exist a  $k$  which makes the grammar LAG( $k$ )? Explain.