Fill out the fields above.
Make use only of an indelible pen in black or blue color.
Do not make use of “Tipp-Ex” or something similar.
To solve the exam you have 90 minutes.
Check that you received 7 pages.
The maximum number of points that you can obtain in this exam is 50.
You need 20 points to pass.
Assignment 1 Berry-Sethi Approach

Use the Berry-Sethi Algorithm and transform the expression $r = ((a|b)^*d)(b|c)^*$ as follows:

1. Draw the regular expression as a tree.
2. Compute the empty attribute.
3. Compute the first attribute.
4. Compute the next attribute.
5. Compute the last attribute.
6. Construct and draw the automaton.
7. Is the resulting automaton deterministic or not?
8. Is the resulting automaton minimal or not?
Suggested Solution 1

[11 Points]

- $e = f$
  - $f = \{1, 2, 3\}$
  - $n = \emptyset$
  - $l = \{3, 4, 5\}$

- $e = t$
  - $f = \{1, 2, 3\}$
  - $n = \emptyset$
  - $l = \{3, 4, 5\}$

- $e = f$
  - $f = \{1\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{2\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{3\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{4\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{5\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{1\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{2\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{3\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{4\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{5\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{1\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{2\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{3\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{4\}$
  - $n = \emptyset$
  - $l = \{3\}$

- $e = f$
  - $f = \{5\}$
  - $n = \emptyset$
  - $l = \{3\}$

[3 Points]

The automaton is deterministic.

The automaton is not minimal. The states 3 •, 4 •, and 5 • as well as •r, 1 •, and 2 • can be merged into one state:

[1 Point]

The automaton is deterministic.
Consider grammar $G = (N, T, \delta, S')$ with the terminals $T = \{a, b\}$, non-terminals $N = \{S', A, B\}$, start symbol $S'$ and productions $\delta$:

$$
\delta : \quad S' \rightarrow A \\
A \rightarrow B \mid B b A \mid B b B a \\
B \rightarrow a
$$


2. [2 P] Can you create an LL(1) Parser from this Automaton? Either create the Lookahead-Table for $G$, or elaborate on the situation, in which the LL(1) Parser would run into a conflict.

3. [8 P] Claim 1: “The canonical LR(0) Automaton for $G$ is conflict-free.” Prove or disprove with facts deduced from the canonical LR(0)-Automaton.

Suggested Solution 2

1. 

<table>
<thead>
<tr>
<th>State</th>
<th>Production</th>
<th>Transition</th>
<th>[a]</th>
<th>[b]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(A \rightarrow \cdot BbA)</td>
</tr>
<tr>
<td>1</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(A \rightarrow \cdot B)</td>
</tr>
<tr>
<td>2</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(S' \rightarrow \cdot A)</td>
<td>(A \rightarrow \cdot BbBa)</td>
</tr>
<tr>
<td>3</td>
<td>(A \rightarrow \cdot BbA)</td>
<td>(A \rightarrow \cdot BbA)</td>
<td>(A \rightarrow \cdot Bb)</td>
<td>(B \rightarrow \cdot a)</td>
</tr>
<tr>
<td>4</td>
<td>(A \rightarrow \cdot BbBa)</td>
<td>(A \rightarrow \cdot BbBa)</td>
<td>(A \rightarrow \cdot Bb)</td>
<td>(B \rightarrow \cdot a)</td>
</tr>
<tr>
<td>5</td>
<td>(A \rightarrow \cdot B)</td>
<td>(A \rightarrow \cdot B)</td>
<td>(A \rightarrow \cdot BbA)</td>
<td>(B \rightarrow \cdot a)</td>
</tr>
<tr>
<td>6</td>
<td>(A \rightarrow Bb \cdot A)</td>
<td>(A \rightarrow Bb \cdot A)</td>
<td>(A \rightarrow \cdot Bb)</td>
<td>(A \rightarrow \cdot B)</td>
</tr>
<tr>
<td>7</td>
<td>(A \rightarrow Bb \cdot A)</td>
<td>(A \rightarrow Bb \cdot A)</td>
<td>(A \rightarrow \cdot Bb)</td>
<td>(A \rightarrow \cdot BbA)</td>
</tr>
<tr>
<td>8</td>
<td>(A \rightarrow Bb \cdot A)</td>
<td>(A \rightarrow Bb \cdot A)</td>
<td>(A \rightarrow \cdot Bb)</td>
<td>(A \rightarrow \cdot B)</td>
</tr>
<tr>
<td>9</td>
<td>(A \rightarrow Bb \cdot Bb)</td>
<td>(A \rightarrow Bb \cdot Bb)</td>
<td>(A \rightarrow \cdot Bb)</td>
<td>(B \rightarrow \cdot a)</td>
</tr>
<tr>
<td>10</td>
<td>(B \rightarrow \cdot a)</td>
<td>(A \rightarrow B \cdot Bb)</td>
<td>(A \rightarrow \cdot B)</td>
<td>(B \rightarrow \cdot a)</td>
</tr>
<tr>
<td>11</td>
<td>(A \rightarrow B \cdot bA)</td>
<td>(b \rightarrow A \cdot Bb)</td>
<td>(A \rightarrow \cdot B)</td>
<td>(B \rightarrow \cdot a)</td>
</tr>
<tr>
<td>12</td>
<td>(A \rightarrow B \cdot bA)</td>
<td>(b \rightarrow A \cdot Bb)</td>
<td>(A \rightarrow \cdot B)</td>
<td>(B \rightarrow \cdot a)</td>
</tr>
<tr>
<td>13</td>
<td>(A \rightarrow BbB \cdot a)</td>
<td>(A \rightarrow BbB \cdot a)</td>
<td>(A \rightarrow \cdot B)</td>
<td>(B \rightarrow \cdot a)</td>
</tr>
</tbody>
</table>

2. There are three variant productions, \(A \rightarrow BbA\), \(A \rightarrow BbBa\) and \(A \rightarrow B\). We have to compare their potential yield and right contexts:

\[
First_1(B) \cap Follow_1(A) = First_1(BbBa) \cap Follow_1(A) = First_1(BbA) \cap Follow_1(A)
\]

\[
\{a\} \cap_1 Follow_1(A) = \{a\} \cap_1 Follow_1(A) = \{a\} \cap_1 Follow_1(A)
\]

\[
\{a\} = \{a\} = \{a\}
\]

→ a lookahead of size 1 is not sufficient, to decide which alternative to take

3. \(q_1\) contains a shift-reduce conflict between the items \(A \rightarrow B\) and \(A \rightarrow B \cdot bA\)

4. There is no conflict whatsoever: All final states with more than one item have only closed items, whose lookahead sets differ from the right contexts of the non-closed items marker.
Assignment 3 Attribute Grammars  [9 Points]

The following grammar represents the fraction of a language that treats expression statements. The expressions support binary operators, value assignment as well as multidimensional array access. A few specific properties of this language are not treated with syntactical rules, instead they are addressed via the semantical analysis.

<table>
<thead>
<tr>
<th>rule</th>
<th>production</th>
<th>attribute system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$S' := { stmts }$</td>
<td>$valid[0] := valid[2]$</td>
</tr>
<tr>
<td>2</td>
<td>$stmts := expr ; stmts$</td>
<td>$valid[0] := valid[1] \land valid[3]$</td>
</tr>
<tr>
<td>3</td>
<td>$\varepsilon$</td>
<td>$valid[0] := true$</td>
</tr>
<tr>
<td>5</td>
<td>$\mid expr = expr$</td>
<td>$lhs[0] := false$ $valid[0] := lhs[1] \land valid[3]$</td>
</tr>
<tr>
<td>7</td>
<td>$\mid \text{const}$</td>
<td>$lhs[0] := false$ $valid[0] := true$</td>
</tr>
<tr>
<td>8</td>
<td>$\mid \text{var}$</td>
<td>$lhs[0] := true$ $valid[0] := true$</td>
</tr>
<tr>
<td>10</td>
<td>$\mid expr$</td>
<td>$valid[0] := valid[1]$</td>
</tr>
<tr>
<td>11</td>
<td>$expr := \text{incop} expr$</td>
<td>$lhs[0] := false$ $valid[0] := lhs[2]$</td>
</tr>
<tr>
<td>12</td>
<td>$expr := expr \text{incop}$</td>
<td>$lhs[0] := false$ $valid[0] := lhs[1]$</td>
</tr>
</tbody>
</table>

Complete the attribute grammar, such that

1. the boolean attribute $valid$ evaluates to true if in its particular subtree the only expressions that appear on the left hand sides of assignments are either variables or array access expressions.

2. expressions with unary pre-/postfix incrementors ($++$) become possible

3. only pre-/postfix incrementor expressions of array accesses and variables contribute to $valid$ evaluating to true

4. pre-/postfix incrementor expressions that appear on left hand sides of an assignments contribute to $valid$ evaluating to false

Attention:

- You may introduce further attributes by need
- You may introduce further productions/variants by need