T-61.231 Principles of Pattern Recognition

Answers to exercise 9: 25.11.2002

1. $L = \{a^n b | n = 1, 2, \ldots\}$

   a)

$$\begin{aligned} V_T &= \{a, b\} \\ V_N &= \{S, A\} \\ P &= \{S \rightarrow aA, A \rightarrow aA|b\} \end{aligned}$$

   b) A parser can be made for example with a recursive descent approach:

```
#include <stdio.h>

void A() {
  switch (getchar()) {
  case 'a':
    A();
    break;
  case 'b':
    break;
  default:
    puts("String not valid");
    exit(0);
  }
}

int main() {
  printf("Please enter input string:\n");
  switch (getchar()) {
  case 'a':
    A();
    break;
  default:
    puts("String not valid");
    exit(0);
  }
  if (getchar()=='\n')
    puts("String is ok");
  else
    puts("String not valid");
  exit(0);
}
```

   Some example runs:

```
Please enter input string:
ba
String not valid

Please enter input string:
aab
String is ok
```

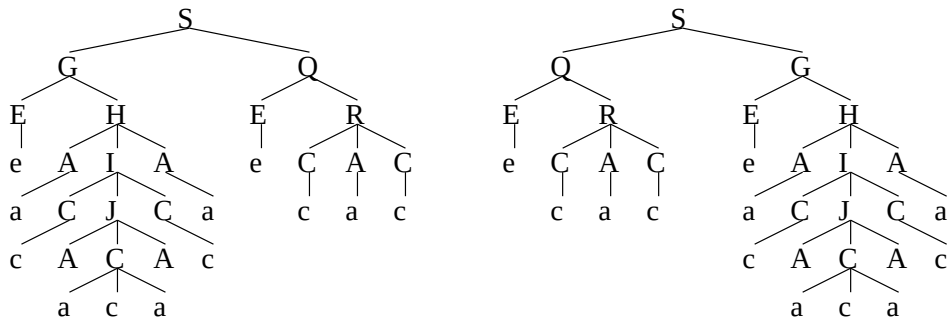
```
Please enter input string:
aaaaa
String not valid

Please enter input string:
aabb
String not valid
```

2.

$$
\begin{aligned}
V_T &= \{a, c, e\} \\
V_N &= \{G, Q, E, H, I, A, C, J, R\} \\
P &= \{S \to GQ, S \to QG, G \to EH, H \to AIA, I \to CJC, J \to ACA, \\
&\qquad Q \to ER, R \to CAC, A \to a, C \to c, E \to e\}
\end{aligned}
$$

The resulting parsing (derivation) trees are

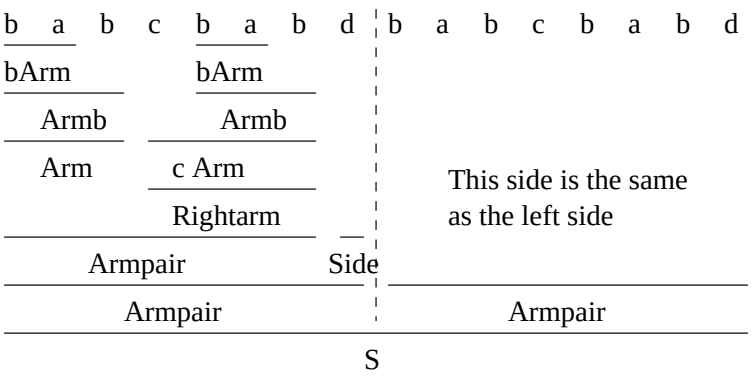


Thus the grammar accepts the strings 'eacacacaecac' and 'ecaceacacaca'.

From the figure, the parts string can be read as 'aceacacacaec'. Since the part is obviously circular, the question is actually whether we can choose the starting point so that the resulting string is one of those produced by the grammar. This can be done, as by moving the two first characters in the read string to the end, the string becomes 'eacacacaecac', which is exactly the same as the first string from the grammar.

3. For this task a bottom-up parsing approach is most useful. Now the question is, can we change the string to the starting symbol by using the production rules?

The read string is *babcbabdbabcbabd*. To check this,



$\Rightarrow$ The grammar accepts this string

Now lets move the staring point by one to produce *abcbabdbabcbabdb*:

```
a   b   c   b   a   b   d   b   a   b   c   b   a   b   d   b
Arm         bArm            bArm            bArm            Side
Armb            Armb            Armb            Armb
Arm         c Arm           Arm             c Arm
            Rightarm                        Rightarm
    Armpair             Side        Armpair             Side
        Armpair                         Armpair
                            S
```

⇒ The grammar accepts this string

And still by one, to *bcbabdbabcbabdba*:

```
b   c   b   a   b   d   b   a   b   c   b   a   b   d   b   a
                                                    Side  bArm
                                                NOT ALLOWED

b   c   b   a   b   d   b   a   b   c   b   a   b   d   b   a
                                                    Side    Arm
                                                    Side
                                                NOT ALLOWED

b   c   b   a   b   d   b   a   b   c   b   a   b   d   b   a
                                                        Side
                                                    Side
                                                NOT ALLOWED
```

⇒ The grammar does not accept this string

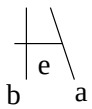
Note: You can get 'a' only from the rule '$Arm \rightarrow a$' and 'd' only from the rule '$Side \rightarrow d$'

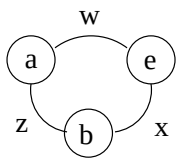
4. With attributed graphs nodes correspond to the primitives and arcs to the relations between the primitives.

$$G_i = \{N_i, P_i, R_i\}$$

Where N_i is a node set, P_i a node property set and R_i a set of node relations.

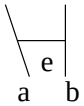
22:

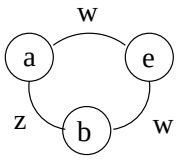




	a	b	e
a	–	z	w
b	z	–	x
e	w	x	–

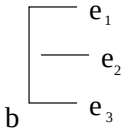
25:

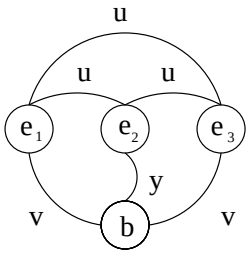




	a	b	e
a	–	z	w
b	z	–	w
e	w	w	–

39:





	b	e_1	e_2	e_3
b	–	v	y	v
e_1	v	–	u	u
e_2	y	u	–	u
e_3	v	u	u	–

Isomorphism for attributed graphs is defined as two attributed graphs being isomorphic if there is a 1:1 and onto assignment of nodes so that all nodes are compatible. In other words, one graph can be transformed to the other by simply reindexing the nodes.

For the graphs for characters 22 and 25, the graphs are not isomorphic due to the fact that the node relations do not match.

None of the graphs are isomorphic with the graph for character 39, as the number of nodes does not match.