Task 7.1: Is in LL(1)-form?

Definition:
Grammar is in LL(1)-form, if for any two rules associated to any nonterminal symbol A, $A \to \omega_1$ and $A \to \omega_2$, $\omega_1 \neq \omega_2$, holds:

$$\begin{aligned} &\text{FIRST}(\{\omega_1\}\text{FOLLOW}(A)) \\ \cap\ &\text{FIRST}(\{\omega_2\}\text{FOLLOW}(A)) \quad = \quad \emptyset. \end{aligned}$$

Rules of S satisfy clearly this condition (every rule begins with different terminal). What about rules of L?

$FIRST(\{LandS\}FOLLOW(L)) = FIRST(\{'(',' p',' q'\}\{')',' a',' o'\} = \{'(',' a',' o'\}$

$FIRST(\{LorS\}FOLLOW(L)) = FIRST(\{'(',' p',' q'\}\{')',' a',' o'\} = \{'(',' a',' o'\}$

$FIRST(\{LandS\}FOLLOW(L)) \bigcup FIRST(\{LorS\}FOLLOW(L)) = \{'(',' a',' o'\} \neq \emptyset$

so is not in LL(1)-form (we don't have to study the third rule any more).

Task 7.2: Transform to LL(1)-form and write a recursive parser

Rules:

1. Left factorization

- Grammar with productions

  $$A \to \alpha\beta_1 \mid \alpha\beta_2, \qquad \alpha \neq \epsilon,\ \beta_1 \neq \beta_2$$

  cannot be in LL(1)-form

- Solution: Let's take a new symbol A' and replace the previous productions with:

  $$\begin{aligned} A &\to \alpha A' \\ A' &\to \beta_1 \mid \beta_2, \end{aligned}$$

  in which α is the longest common prefix of $\alpha\beta_1$ and $\alpha\beta_2$

2. Removing direct left recusrion:

- *Direct* left recursion, i.e.direct derivatoons $A \Rightarrow A\gamma$ can be avoided by replacing productions

$$A \to A\beta \mid \alpha, \qquad \beta \neq \epsilon,$$

with productions

$$\begin{aligned} A &\to \alpha A' \\ A' &\to \beta A' \mid \epsilon \end{aligned}$$

- Productions of form $A \to A$ can be simply dropped.

Two possibilities:

1. $S->(L)|p|q$
   $L->LandS|LorS|S$

   After left factorization:

   $S->(L)|p|q$
   $L->LL'|S$
   $S'->andS|orS$

   After removing left recursion:

   $S->(L)|p|q$
   $L->SS'$
   $S'->L'S'|\epsilon$
   $L'->andS|orS$

   which is equivalent with:

   $S->(L)|p|q$
   $L->SS'$
   $S'->andSS'|orSS'|\epsilon$

2. Or we can remove left recursion in the beginning:

   $S->(L)|p|q$
   $L->SS'$
   $S'->andSS'|orSS'|\epsilon$

So two ways to the same result.

Parser

```
function S {
if (next=='(') {
     next=getnext;
     L;
     if (next<>')')
          ERROR;
     else next=getnext;
}
else if (next<>'p') or (next<>'q')
     ERROR;
     else next=getnext;
}

function L {
S;
L';
}

function L' {
if (next=='a') or (next=='o') {
     if (next=='a') {
          read('and');
          next=getnext;
     }
     else if (next=='o') {
          read('or');
          next=getnext;
          }
     S;
```

```
    L';
    }
}
```