T-79.148 **Kevät 2002**
Tietojenkäsittelyteorian perusteet
Harjoitus 7
Demonstraatiotehtävien ratkaisut

4. **Tehtävä**: Osoita, että yhteydettömien kielten luokka on suljettu yhdiste-, katenaatio- ja sulkeumaoperaatioiden suhteen, so. jos kielet $L_1, L_2 \subseteq \Sigma^*$ ovat yhteydettömiä, niin samoin ovat myös kielet $L_1 \cup L_2$, $L_1 L_2$ ja L_1^*.

   Vastaus: Olkoon L_1 ja L_2 yhteydettömiä kieliä. Määritellään nyt kieliopit $G_1 = (V_1, \Sigma_1, R_1, S_1)$ ja $G_2 = (V_2, \Sigma_2, R_2, S_2)$, siten, että $L(G_1) = L_1$ ja $L(G_2) = L_2$. Vaaditaan lisäksi, että $(V_1 - \Sigma_1) \cap (V_2 - \Sigma_2) = \emptyset$, eli kieliopeissa ei esiinny samoja välikkeitä. Koska kieliopin välikkeet voidaan tarvittaessa nimetä uudelleen, ei tämä aseta oleellista rajoitusta.

   Unioni: Olkoon S uusi välike ja $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}, S\}$. Nyt $L(G) = L(G_1) \cup L(G_2) = L_1 \cup L_2$. Näin on, koska S:stä voidaan johtaa vain S_1 tai S_2, joista voidaan edelleen johtaa vain sanoja jotka kuuluvat jompaan kumpaan aiemmista kielistä (sääntöjen sekaannukselta vältytään, koska nonterminaalijoukot ovat pistevieraita).

   Katenaatio: Tällä kertaa uusi kielioppi $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \to S_1 S_2\}, S\}$. Nyt $L(G) = L_1 L_2$.

   Kleenen tähti: Tällä kertaa uusi kielioppi $G = (V_1 \cup \{S\}, \Sigma_1, R_1 \cup \{S \to \epsilon \mid SS_1\}, S\}$. Nyt $L(G) = L_1^*$

5. **Tehtävä**:

   (a) Osoita, että seuraava yhteydetön kielioppi on moniselitteinen:

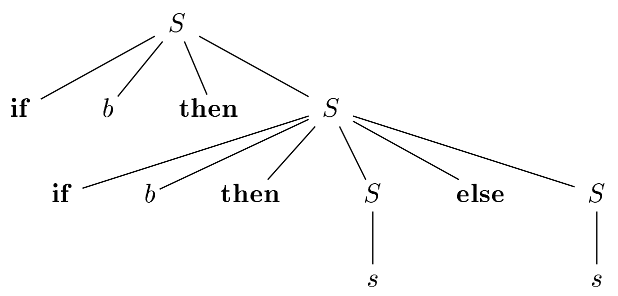
   $$\begin{aligned} S &\to \textbf{if } b \textbf{ then } S \\ S &\to \textbf{if } b \textbf{ then } S \textbf{ else } S \\ S &\to s. \end{aligned}$$

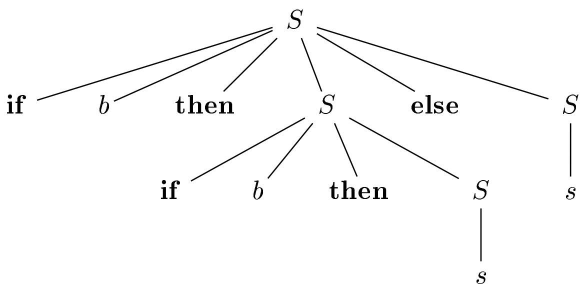
   (b) Muodosta (a)-kohdan kieliopin kanssa ekvivalentti, so. saman kielen tuottava yksiselitteinen kielioppi.

   Vastaus: Yhteydetön kielioppi G on moniselitteinen, mikäli on olemassa sana $w \in L(G)$ siten, että w:llä on ainakin kaksi erilaista jäsennyspuuta. Tehtävän kieliopille yksinkertaisin tällainen sana on:

   $$\textbf{if } b \textbf{ then if } b \textbf{ then } s \textbf{ else } s,$$

   joka voidaan jäsentää seuraavasti:





Yleensä ohjelmointikielissä halutaan **else**-lause liittää lähinpään mahdolliseen **if**-lauseeseen. Ylläolevista puista ensimmäinen vastaa tätä käytäntöä.

Määritellään kielioppi seuraavasti:

$$
\begin{aligned}
&G = (V, \Sigma, P, S)\\
&V = \{S, B, U, s, b, \textbf{if}, \textbf{then}, \textbf{else}\}\\
&\Sigma = \{s, b, \textbf{if}, \textbf{then}, \textbf{else}\}\\
&P = \{S \to B \mid U\\
&\qquad B \to \textbf{if } b \textbf{ then } B \textbf{ else } B \ \mid s\\
&\qquad U \to \textbf{if } b \textbf{ then } S \mid \textbf{if } b \textbf{ then } B \textbf{ else } U\}
\end{aligned}
$$

Tässä välikeellä B saadaan johdettua vain ohjelmia, joissa kaikilla **if**-lauseilla on sekä **then**- että **else**-haarat. Välikkeellä U johdetaan sitten **if**-lauseet, joista puuttuu **else**-haara.

6. **Tehtävä**:Laadi rekursiivisesti etenevä jäsentäjä edellisten harjoitusten tehtävän 6 kieliopille.

   Vastaus: Alla oleva C-ohjelma toteuttaa rekursiivisen jäsentäjän kieliopille:

$$
\begin{aligned}
&C \to S \mid S; C\\
&S \to a \mid \textbf{begin } C \textbf{ end} \mid \textbf{for } n \textbf{ times do } S
\end{aligned}
$$

   Tässä on yksinkertaistettu hieman edellisen laskuharjoituskerran 6. tehtävän kielioppia korvaamalla erilliset numerot terminaalilla n, joka tarkoittaa mitä tahansa numeroa.

   Tärkeimmät ohjelmassa esiintyvät funktiot ovat:

   - `C()`, `S()` — toteuttavat kieliopin varsinaiset säännöt
   - `lex()` — lukee syötteestä seuraavan lekseemin ja tallettaa sen globaaliin muuttujaan `current_tok`.
   - `expect(int token)` — yrittää lukea syötteestä lekseemin *token*. Mikäli lukeminen epäonnistuu annetaan virheilmoitus.
   - `consume_token()` — merkitään tämänhetkinen lekseemi käytetyksi. Tämä (tai jokin muu vastaava funktio) tarvitaan siksi, että joissain tapauksissa täytyy syötettä lukea yksi lekseemi eteenpäin ennen kuin tiedetään, mitä sääntöä täytyy käyttää.

   Käytännössä ohjelmointikielten jäsentäjät toteutetaan yleensä käyttäen *lex*- ja *yacc*-työkaluja[1]. Näistä *lex* muodostaa tilakonepohjaisen selaajan, joka tunnistaa säännöllisillä lausekkeilla määritellyt lekseemit, ja *yacc* tekee pinoautomaattipohjaisen jäsentimen annetulle yhteydettömälle kieliopille.

[1] Tai niiden johdannaisia.

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

/* Define the alphabet */
enum TOKEN  { DO, FOR, END, BEGIN, TIMES, OP, SC, NUMBER, ERROR };
const char* tokens[] = { "do", "for", "end", "begin", "times", "a",
                         ";", "NUMBER", NULL };

/* A global variable holding the current token */
int current_tok = ERROR;

/* Maximum length of a token */
#define TOKEN_LEN 128

/* declare functions corresponding to nonterminals */
void S(void);
void C(void);

int lex(void);
void consume_token(void);
void error(char *st);
void expect(int token);

void C(void)
{
  S();
  lex();
  if (current_tok == SC) {
    consume_token();
    C();
    printf("C => S ; C\n");
  } else {
    printf("C => S\n");
  }
}

void S(void)
{
  lex();
  switch (current_tok) {
  case OP:
    consume_token();
    printf("S => a\n");
    break;
  case BEGIN:
    consume_token();
    C();
    expect(END);
    printf("S => begin C end\n");
    break;
  case FOR:
    consume_token();
    expect(NUMBER);
```

```
    expect(TIMES);
    expect(DO);
    S();
    printf("S => for N times do S\n");
    break;
  default:
    error("Parse error");
  }
}

/* int lex(void) returns the next token of the input. */
int lex(void)
{
  static char token_text[TOKEN_LEN];
  int pos = 0, c, i, next_token = ERROR;

  /* Is there an existing token already? */
  if (current_tok != ERROR)
    return current_tok;

  /* skip whitespace */
  do {
    c = getchar();
  } while (c != EOF && isspace(c));
  if (c != EOF) ungetc(c, stdin);

  /* read token */
  c = getchar();
  while (c != EOF && c != ';' && !isspace(c) && pos < TOKEN_LEN) {
    token_text[pos++] = c;
    c = getchar();
  }
  if (c == ';') {
    if (pos == 0) /* semicolon as token */
      next_token = SC;
    else { /* trailing semicolon, leave it for future */
      ungetc(';', stdin);
    }
  }
  token_text[pos] = '\0'; /* trailing zero */

  /* identify token */
  if (isdigit(token_text[0])) { /* number? */
    next_token = NUMBER;
  } else { /* not a number */
    for (i = DO; i < NUMBER; i++) {
      if (!strcmp(tokens[i], token_text)) {
        next_token = i;
        break;
      }
    }
  }
  current_tok = next_token;
  return next_token;
```

```
}

void consume_token(void)
{
  current_tok = ERROR;
}

void error(char *st)
{
  printf(st);
  exit(1);
}

/* try to read a 'token' from input */
void expect(int token)
{
  int next_tok = lex();
  if (next_tok == token) {
    consume_token();
    return;
  } else
    error("Parse error");
}

int main(void)
{
  int i;
  C();
  return 0;
}
```