Tik-79.148 **Kevät 2001**
Tietojenkäsittelyteorian perusteet
Laskuharjoitus 6
Ratkaisut

Demonstraatiotehtävät:

1. Aakkoston $\Sigma = \{a, b, (,), \cup, ^*, \emptyset\}$ sallitut säännölliset lausekkeet voi generoida kieliopilla $G = (V, \Sigma, R, S)$,

$$
\begin{aligned}
V =& \Sigma \cup \{S, T, F\} \\
R =& \{S \to T \cup S, S \to T \\
& T \to F, T \to F^*, \\
& T \to FT, T \to F^*T \\
& F \to \emptyset, F \to a, \\
& F \to b, F \to (S)\}
\end{aligned}
$$

Esimerkiksi säännöllinen lauseke $(a \cup bb)^*b$ johdetaan $S \to T \to F^*T \to (S)^*T \to (T \cup S)^*T \to (F \cup S)^*T \to (a \cup S)^*T \to (a \cup T)^*T \to (a \cup FT)^*T \to (a \cup bT)^*T \to (a \cup bF)^*T \to (a \cup bb)^*T \to (a \cup bb)^*F \to (a \cup bb)^*b$

Formaalit kielet muodostavat hierarkian, jossa jokainen porras on aina aidosti edellistä vahvempi ilmaisuvoimaltaan. Niinpä esimerkiksi säännölliset kielet voidaan kuvata säännöllisillä lausekkeilla, ja kielet, joihin kuuluvat kaikki tietyllä aakkostolla muodostettavat säännölliset lausekkeet, voidaan kuvata yhteydettömillä kieliopeilla. Ei kuitenkaan yleisesti päde, että tiettyn kieliluokan tuottavat kieliopit voitaisiin aina kuvata seuraavan luokan kieliopeilla.

2. a) Kieltä $L = \{a^m b^n \mid m \geq n\}$ vastaava yhteydetön kielioppi on:

$$
\begin{aligned}
\Sigma =& \{a, b\} \\
V =& \{a, b, S, A, B\} \\
R =& \{S \to ASB, S \to e \\
& A \to Aa, A \to a, \\
& B \to b, B \to e\}
\end{aligned}
$$

b) Kieltä $L = \{uawb \mid u, w \in \{a, b\}^*, |u| = |w|\}$ vastaava yhteydetön kielioppi on

$$
\begin{aligned}
\Sigma =& \{a, b\} \\
V =& \{a, b, S, T\} \\
R =& \{S \to Tb, T \to aTa, \\
& T \to aTb, T \to bTb \\
& T \to bTa, T \to a\}
\end{aligned}
$$

3. (*soveltava*)

Melkein kaikki nykyisin käytössä olevat ohjelmointikielet pohjautuvat yhteydettömiin kielioppeihin, joilla kuvataan kielen rakenne. Vain kaikkein yksinkertaisimmat kielet ovat puhtaasti yhteydettömiä, sillä esimerkiksi muuttujien tyyppitarkistuksen tekeminen pelkillä yhteydettömillä säännöillä on mahdotonta. Käytännössä asia hoidetaan siten, että ohjelma jäsennetään kieliopin mukaan puurakenteeksi välittämättä mitään muuttujien tyypeistä. Kun ohjelma on saatu jäsennetyksi, tarkistetaan saadusta rakenteesta onko muuttujia käytetty oikein.

Tehtävänä oleva yhteen- ja vähennyslaskujen tekeminen voidaan toteuttaa yksinkertaisella kieliopilla:

$$
\begin{aligned}
&G = (V, \Sigma, R, S) \\
&\Sigma = \{number, +, -, (,)\} \\
&R = \{S \to number, S \to S + S, S \to S - S, S \to (S)\}
\end{aligned}
$$

Yllä on abstrahoitu kaikki mahdolliset kokonaisluvut terminaalisymboliksi *number*. Ehkä yksinkertaisin mahdollinen tapa toteuttaa jäsennin on tehdä siitä rekursiivinen, missä kutakin nonterminaalia vastaa yksi funktio. Esimerkiksi yllä olevaan kielioppiin tulee vain yksi funktio, kutsutaan sitä vaikkapa nimellä `expr`. Funktion rungossa luetaan syötettä niin pitkälle, että voidaan tunnistaa, mitä sääntöä täytyy käyttää. Pseudokoodiesityksenä funktio `expr` näyttää tältä:

```
integer expr()
  integer value

  Jos sääntö on muotoa S -> number,
     value = number()

  Jos sääntö on muotoa S -> S + S,
     value = expr() + expr()

  Jos sääntö on muotoa S -> S - S
     value = expr() - expr()

  Jos sääntö on muotoa S -> ( S )
     lue '('
     value = expr();
     lue ')'

  return value
```

Alla on C-ohjelmointikielellä kirjoitettu yksinkertainen jäsennin annetulle kieliopille.

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

```
/* Read the next input symbol */
int get_token(char expect);

int number(void);
int expr(void);

int current_token;

void error(char *st)
{
  printf("%s\n", st);
  exit(0);
}

int number(void)
{
  int num;
  ungetc(current_token, stdin);
  scanf("%d", &num);
  return num;
}

int get_token(char expect)
{
  int ch;
  do {
    ch = getchar();
  } while (ch == ' '); /* skip spaces in input */
  return ch;
}

int expr(void)
{
  int left_value = 0;
  int right_value = 0;
  int result = 0;

  current_token = get_token(NONE);

  if (isdigit(current_token)) {
    left_value = number();
    current_token = get_token(NONE);
  } else if (current_token == '(') {
    left_value = expr();
    if (current_token != ')') {
      error("parenthesis error");
    }
  } else {
    error("expression has to begin with a number or '('");
  }
```

```
  switch (current_token) {
  case '\n':
    result = left_value;
    break;
  case '+':
    right_value = expr();
    result = left_value + right_value ;
    break;
  case '-':
    right_value = expr();
    result = left_value - right_value ;
    break;
  case ')':
    result = left_value;
    break;
  default:
    error("invalid character in an expression");
  }
  return result;
}

int main(void)
{
  int value = 0;

  value = expr();
  printf("Result: %d\n", value);
}
```