Tik-79.148 **Kevät 2000**
Tietojenkäsittelyteorian perusteet
Laskuharjoitus 5
Ratkaisut

1. (2.3.7a / 2.5.1a)

   Konstruktion perusajatuksena on käydä systemaattisesti läpi kaikki mahdolliset reitit alkutilasta kaikkiin lopputiloihin. Aluksi kaikki tilat numeroidaan yksikäsitteisesti, ja sen jälkeen muodostetaan reitti käyttäen rekursiivisia kaavoja.

   Rekursiiviset säännöt ovat:

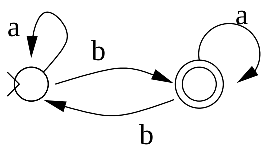
$$R(i,j,1) = \begin{cases} \{\sigma \in \Sigma \mid \delta(q_i,\sigma) = q_j\} & i \neq j \\ \{e\} \cup \{\sigma \in \Sigma \mid \delta(q_i,\sigma) = q_j\} & i = j \end{cases}$$
$$R(i,j,k+1) = R(i,j,k) \cup R(i,k,k)R(k,k,k)^*R(k,j,k)$$

   Merkintä $R(i,j,k)$ tarkoittaa kaikkia niitä reittejä tilasta q_i tilaan q_j, jotka eivät käy välillä tilassa q_k tai sitä suuremmassa tilassa.

   Ensimmäinen säännöistä sanoo, että mikäli välillä ei saa käydä missään tilassa, ainoat reitit tilojen q_i ja q_j välillä ovat suorat siirtymät.

   Säännöistä jälkimmäinen tarkoittaa, että reitit tilasta q_i tilaan q_j käymättä välillä tilassa q_{k+1} tai sitä suuremmassa tilassa, voidaan jakaan kahteen ryhmään: niihin, joissa ei käydä edes tilassa q_k, ja niihin, joissa käydään tilassa q_k.

   Tilakonetta



   vastaava säännöllinen lauseke saadaan seuraavasti:

$$\begin{aligned} L(M) &= R(1,2,3) \\ R(1,2,3) &= R(1,2,2) \cup R(1,2,2)R(2,2,2)^*R(2,2,2) \\ R(1,2,2) &= R(1,2,1) \cup R(1,1,1)R(1,1,1)^*R(1,2,1) \\ R(2,2,2) &= R(2,2,1) \cup R(2,1,1)R(1,1,1)^*R(1,2,1) \end{aligned}$$

$$\begin{aligned} R(1,2,1) &= R(2,1,1) = b \\ R(1,1,1) &= R(2,2,1) = e \cup a \end{aligned}$$

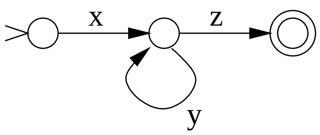
   Lopuksi sijoitetetaan saadut lausekkeet aikaisempiin yhtälöihin, ja sievennetään tulokset:

$$\begin{aligned}
R(1,2,2) &= b \cup (e \cup a)(e \cup a)^* b = a^* b \\
R(2,2,2) &= (e \cup a) \cup b(e \cup a)^* b = e \cup a \cup ba^* b \\
R(1,2,3) &= a^* b \cup a^* b(e \cup a \cup ba^* b)^* (e \cup a \cup ba^* b) \\
&= a^* b(a \cup ba^* b)^*
\end{aligned}$$

2. (2.4.5 / 2.6.5)

   Tilakoneiden pumppauslemman mukaan jokaiselle äärettömälle säännölliselle kielelle L löytyy jokin $k \geq 0$ siten, että kielen kaikki sitä pidemmät sanat w ($|w| > k$) voidaan jakaa kolmeen osaan x, y ja z ($y \neq e$) siten, että $xy^n z \in L, n \geq 0$.

   Toisin sanoen, mikäli säännöllinen kieli on ääretön, täytyy olla olemassa jokin merkkijono, jonka voi lisätä sanan keskelle mielivaltaisen monta kertaa. Lemma seuraa suoraan siitä, että sekä tilakoneiden tilajoukko että aakkosto ovat äärellisiä. Mikäli koneen kuvaama kieli on ääretön, kuuluu siihen mielivaltaisen pituisia sanoja. Kun sananpituus kasvaa suuremmaksi kuin tilakoneen tilojen määrä, täytyy laskennan aikana vierailla ainakin yhdessä tilassa kaksi tai useampia kertoja, eli tilakoneessa on silmukka. Tämä silmukka voidaan suorittaa mielivaltaisen monta kertaa (myös 0).



   Tehtävässä piti todistaa, että $L = \{ww^R \mid w \in \{a,b\}^*\}$ ei ole säännöllinen. Todistus onnistuu helpommin, mikäli määritellään kieli $L' = L \cap (ab)^*(ba)^*$. Koska säännöllisten kielten joukko on suljettu leikkauksen suhteen, ei L voi olla säännöllinen, jos L' ei ole säännöllinen.

   Tarkastelemalla kieleen L kuuluvia sanoja huomataan, että

$$L' = (ab)^n (ba)^n, n \geq 0 \ .$$

   Jotta L' voisi olla säännöllinen, pitäisi siis löytää y, joka voidaan lisätä sanaan mielivaltaisen monta kertaa.

   Nyt y voi olla muotoa:

   1° $y = (ab)^r$ tai $y = (ba)^r, r > 0$ Nyt kuitenkin toinen puolisko kasvaisi pidemmäksi kuin toinen, joten syntyneet sanat eivät kuulu enää kieleen.

   2° $y = bb, y = aa, y = a(ba)^r$ tai $(ba)^r b, r \geq 0$ eivät myöskään kelpaa, sillä kielen sanoissa on aina yhtä monta a- kuin b-kirjainta.

   3° $y = a(ba)^r bb(ab)^r b$ ei käy, koska tällöin sanassa esiintyy ba ennen viimeisiä ab:ta.

   Näin huomattiin, että mikään y:n arvo ei ole mahdollinen. L' ei siis ole säännöllinen, joten myöskään L ei voi olla säännöllinen.

3. (2.4.8 / -)

   Säännöllisten kielten pumppauslemmalla voidaan osoittaa, että kieli *ei* ole säännöllinen, mutta sillä ei voida todistaa kielen säännöllisyyttä. Esimerkiksi kieli, jonka sanat ovat muotoa $aa^*b^nc^n$ ei ole säännöllinen (koska b^nc^n-osaa ei pysty määrittelemään säännöllisellä lausekkeella), mutta jokaisen kieleen kuuluvan sanan alkuun voidaan lisätä merkki a mielivaltaisen monta kertaa (eli pumppauslemman ehto toteutuu).

   Kielen todistaminen säännölliseksi onnistuu yleensä helpoiten käyttämällä kieliperheen sulkeumaominaisuuksia: säännöllisten kielten joukko on suljettu unionin, katenaation, Kleenen tähden, komplementoinnin ja leikkauksen suhteen.

   Tehtävässä on annettuna jokin säännöllinen kieli L, jonka avulla muodostetaan kieli

   $$L' = \{xy \mid x \in L \text{ ja } y \notin L\}$$

   Kieli L' on muodostettu katenaation avulla kielestä L ja sen komplementista $\overline{L}$ ($y \notin L \rightarrow y \in \overline{L}$). Koska säännöllisten kielten joukko on suljettu katenaation ja komplementin suhteen ja L on säännöllinen, on myös L' säännöllinen.

   Toinen tapa esittää todistus olisi rakentaa kielen L' hyväksyvä tilakone. Koska L on säännöllinen, on olemassa deterministinen tilakone M, joka hyväksyy sen. Muodostetaan nyt deterministinen tilakone $\overline{M}$ siten, että se on muuten samanlainen kuin M, mutta hyväksyvien ja hylkäävien lopputilojen joukot vaihdetaan keskenään, eli $\overline{M}$ hyväksyy kaikki ne sanat, jotka M hylkää ja päinvastoin. Yhdistetään nyt nämä tilakoneet epädeterministiseksi koneeksi M', siten, että kaikista M:n lopputiloihin lisätään tyhjä siirtymä $\overline{M}$:n alkutilaan. Nyt M' hyväksyy kielen L', joten L' on säännöllinen.

   Kielten sulkeumaominaisuuksia voidaan käyttää myös vastauksen alussa olevan kielen säännöttömyyden osoittamiseen. Kieli $L = aa^*b^nc^n$ on kielten $L_1 = aa^*$ ja $L_2 = b^nc^n$ katenaatio. Jotta L olisi säännöllienen, täytyy sekä L_1:n että L_2:n olla säännöllisiä. Näin ei kuitenkaan ole, sillä L_2 ei ole säännöllinen, mikä voidaan osoittaa pumppauslemman avulla.

4. (2.4.5 / 2.6.5)

   Ohjelmointikielen kääntäjää tehtäessä ensimmäinen vaihe on yleensä kielen leksikaalisen analysoijan suunnittelu. Leksikaalinen analysointi on ohjelman jäsennyksen ensimmäinen osa, ja siinä pyritään etsimään ohjelmasta yhteenkuuluvat merkkijonot (lekseemit). Esimerkiksi koodinpätkä `if (value == 10)` voitaisiin jakaa osiin seuraavasti:

   (a) Lekseemi: '`if`' Tyyppi: varattu sana
   (b) Lekseemi: '`(`' Tyyppi: ryhmitysmerkki
   (c) Lekseemi: '`value`' Tyyppi: muuttuja
   (d) Lekseemi: '`==`' Tyyppi: operaattori
   (e) Lekseemi: '`10`' Tyyppi: numero
   (f) Lekseemi: '`)`' Tyyppi: ryhmitysmerkki

Yleensä analysoija tallettaa lukemansa lekseemin jonkin muuttujan arvoksi ja palauttaa jäsentimelle lekseemin tyypin.

Leksikaaliset analysoijat tehdään yleensä käyttäen apuna työkalua (esim lex tai flex), jossa eri lekseemityypit määritellään käyttäen säännöllisiä lausekkeita. Työkalu tuottaa säännöllisistä lausekkeista deterministisen tilakoneen, jonka voi yleensä helposti liittää kielen jäsentimeen. Koska analysoijan täytyy osata vastata monimutkaisempaan kysymykseen kuin tavanomaisen tilakoneen, poikkeaa niiden semantiikka hieman tavallisten koneiden semantiikasta. Leksikaalinen analysoija ei lue automaattisesti koko syötettä kerrallaan, vaan se pysähtyy heti tunnistettuaan yhden lekseemin ja jättää lopun syötteen ennalleen.

Seuraava Flex-ohjelma tunnistaa tehtävässä annetut leksikaaliset elementit:

```
%{
#include <stdio.h>
#include <stdlib.h>
/* First we define the syntactic classes */
typedef enum { VARIABLE = 256, WHILE = 257, IF = 258,
               ELSE = 259, EQUAL = 260, NUMBER = 261,
               MAX_TYPE } lexical_types;
%}
/* Next, we define the regular expressions for syntactic elements
   that are longer than one symbol */
EQUAL          ==
IF             if
WHILE          while
ELSE           else
VARIABLE       [a-z][a-z]*
NUMBER         [0-9][0-9]*
WHITESPACE     [\t\r\n ]*
/* Now we define what we want to do with the regexps. The order
   matters. */
%%
{WHITESPACE}   { /* do nothing */ }
{IF}           { return IF; }
{WHILE}        { return WHILE; }
{ELSE}         { return ELSE; }
{EQUAL}        { return EQUAL; }
{VARIABLE}     { return VARIABLE;} /* the actual text of the variable
                                      is now held in global variable
                                      char *yytext */
{NUMBER}       { return NUMBER; }  /* again, value is in yytext */
.              { return *yytext;}  /* for other inputs we just return
                                      the ascii value of the input
                                      character and let the caller
                                      worry about error checking */
%%
/* A simple code example using the scanner */
```

```
int main(void) {
    char *token_class[MAX_TYPE];
    int token, i;
    /* initialize the token_class array */
    for (i = 0; i < MAX_TYPE; i++) {
        token_class[i] = "error";
    }
    token_class['+'] = "plus"; token_class['-'] = "minus";
    token_class['*'] = "times"; token_class['/'] = "div";
    token_class['='] = "assign"; token_class[NUMBER] = "number";
    token_class['('] = "open parenthesis";
    token_class[')'] = "close parenthesis";
    token_class['{'] = "open brace"; token_class['}'] = "close brace";
    token_class[VARIABLE] = "variable";
    token_class[WHILE] = "while"; token_class[IF] = "if";
    token_class[ELSE] = "else"; token_class[EQUAL] = "equal";

    while ((token = yylex())) {
        printf("%s", token_class[token]);
        switch (token) {
        case NUMBER:
            printf(": %d", atoi(yytext));
            break;
        case VARIABLE:
            printf(": %s", yytext);
            break;
        default:
            break;
        }
        printf("\n");
    }
    return 0;
}
```

Esimerkkiohjelman saa käännettyä unix-ympäristössä käskyillä:

```
% flex simple.lex
% gcc -Wall lex.yy.c -o lexer -lfl
```

Tämän jälkeen kokeillaan ohjelmaa syötteellä ”`if (value == 10) |`”:

```
% echo 'if (value == 10) |' | ./lexer
if
open parenthesis
variable: value
equal
number: 10
close parenthesis
error
%
```