2. Формални граматики. Пораждане на езици. Класификация на граматики по Хомски. Регулярни изрази, регулярни езици и крайни автомати

Цел на упражнението

Упражнението представя въведение в теорията на формалните езици и граматики. Представени са основните понятия, класификацията на формалните граматики по Хомски. Направен е кратък обзор на регулярните изрази и на съответствието между регулярни изрази, множества и крайни автомати.

2.1. Формални езици

Всеки формален език се определя чрез азбука и формална граматика.

Азбука ∑ представлява крайно множество от символи. Низ над ∑ се нарича всяка крайна редица от символи от ∑. Низът се дефинира чрез следните две правила:

- Празният низ ε принадлежи на азбуката, т.е. $\varepsilon\epsilon\sum$
- Ако **x** е низ над ∑ и **a** е низ над ∑, то низ **xa**, получен при операция конкатенация, също е низ над ∑.

Формален език над азбуката ∑ се нарича множество от низове (изречения) над ∑, включително ε.

Нека $\sum^*$ е множеството, съдържащо всички низове над ∑, включително ε, а $\sum^+ = \sum^* - \{\varepsilon\}$. Тогава всеки формален език L над ∑ представлява подмножество на $\sum^*$.

2.2. Формални граматики

Формална (пораждаща) граматика се нарича математическа система, която определя (описва) формален език. Формална граматика G се задава като множество G = {∑, N, P, S}, където:

∑ - множество от терминални символи на езика

N - множество от нетерминални (помощни) символи на езика, което не се пресича с множеството на терминалните символи ∑

SϵN – стартов (начален) символ на граматиката

P – правила (продукции) за пораждане (извеждане) на низове (изречения) от езика

Правилата са във вида α -> β, където

$\alpha\epsilon(N \cup \sum)^+$, а $\beta\epsilon(N \cup \sum)^*$.

Лявата и дясната страна на всяко правило представляват низове от терминални и/или нетерминални символи. Символът -> означава импликация (от α следва β).

Терминалните символи са символи, които не могат да бъдат заместени с други символи от граматиката. Нетерминалните (променливите) символи се заместват с други символи от правилата на граматиката.

2.3. Пораждане на езици

Всяка формална граматика G поражда формален език L над азбука ∑. Това означава, че всеки низ w от езика L може да бъде породен (изведен) от правилата на граматиката G.

Пораждане (извеждане) на низ w от езика L означава да се построи такава последователност от прилагане на правилата P, при която, започвайки от стартовия символ S, прилагайки правилата P, получаваме низа w. При прилагане на правило от вида α -> β подниз α се замества с низ β.

Пример 1:

Формалната граматика G, която поражда език L = $\{a^m b^n \mid m,n >= 0\}$ се дефинира със следните множества:

$G_1 = \{\sum, N, P, S\}$

∑ = {a, b} – терминални символи

N = {S, A, B} – нетерминални символи

S – стартов символ

P – правила, които са следните:

1. S → AB
2. A → aA
3. A → Ɛ
4. B → bB
5. B → Ɛ

Например низ **aaabb** се поражда (извежда) чрез прилагане на правилата на граматиката в следната последователност:

S → AB → aAB → aaAB → aaaB → aaabB → aaabb

Безкраен език се описва с рекурсивни правила. Правило от вида A -> Aα, където AϵN, $\alpha\epsilon(N \cup \sum)^*$, се нарича ляво рекурсивно, а правило от вида A -> αA се нарича дясно рекурсивно правило. Двойно рекурсивно е правило от вида A -> αAβ.

2.4. Класификация на граматики по Хомски

В зависимост от вида на правилата P формалните граматики се разделят на следните четири групи, които формират т.нар. класификация на Хомски.

Регулярни граматики (означават се с G3). Техните правила са във вида:

A -> x, A -> xB или A -> Ɛ, където A, BϵN, x$\epsilon\sum$, Ɛ – празен низ.

Контекстно-свободни граматики (означават се с G2). Техните правила са във вида:

A -> α, където AϵN, $\alpha\epsilon(N \cup \sum)^*$

Контекстно-зависими граматики (означават с G1). Техните правила са във вида:

α -> β, където $\alpha\epsilon(N \cup \sum)^+$, $\beta\epsilon(N \cup \sum)^*$

Без ограничения (означават се с G0). Техните правила са във вида α -> β. Единственото ограничение към правилата на граматиките без ограничения е α≠ Ɛ.

Например граматиката е от пример 1 не е регулярна (G3), защото съдържа правило (1), в дясната страна на което има низ от два нетерминални символа. Граматиката е контекстно-свободна (G3).

Типът на граматиката определя множеството от низове (съответно множеството от формални езици), които се пораждат от граматиката, т.е. типът на граматиката определя нейната мощност. Колкото по-силни са ограниченията към правилата на граматиката, толкова по-малка е нейната мощност. Най-голяма описателна мощност имат граматиките без ограничения (G0), а най-малка регулярните граматики (G3), т.е.:

G3 ⊂ G2 ⊂ G1 ⊂ G0.

В езиците за програмиране най често се използват регулярни граматики (G3) и контекстно-свободни граматики (G2). С регулярна граматика се описва лексиката на повечето програмни езици. Регулярната граматика се реализира с краен автомат без памет. Този тип граматики обаче не са достатъчно мощни за описание на синтаксиса на програмните езици. Например видът на правилата на G3 не допуска структури със скоби (например съставен оператор във вида {…} или begin … end). Това ограничение налага синтаксисът на повечето езици за програмиране да се задава с контекстно-свободна граматика (G2) и се разпознава с краен автомат със стек. Граматичните правила могат да се прилагат отляво надясно или отдясно наляво(съответният синтактичен анализ се нарича низходящ или възходящ). Обикновено избраният метод на синтактичен анализ налага еквивалентни преобразувания на пораждащата граматика (например замяна на лява рекурсия с дясна или обратно).

Един език за програмиране може да бъде дефиниран с повече от една граматика. Такива граматики се наричат еквивалентни.

2.5. Регулярни изрази, регулярни езици и крайни автомати

Съществува съответствие между регулярни изрази, регулярни граматики и крайни автомати. Всеки регулярен израз описва регулярно множество над азбука и може да се представи и реализира с краен автомат без памет.

2.5.1. Регулярни изрази

Регулярните изрази се използват за дефиниране на правила за представяне на множества от низове, които са валидни в даден формален език. Правилата се задават със следните три оператора:

- Съединяване (конкатенация) xy
- Алтернатива x | y (записва се още като x+y)
- Повторение x* - означава, че x се повтаря 0 или повече пъти

Регулярните изрази се дефинират формално чрез следните рекурсивни правила:

1) Всеки символ от азбуката на формален език е регулярен израз

2) Празният низ ε е регулярен израз

3) Ако R_1 и R_2 са регулярни изрази, то (R_1), (R_2), R_1R_2, $R_1 | R_2$, R_1^*, R_2^* са също регулярни изрази

4) Никакъв друг израз не е регулярен

Чрез регулярни изрази мога да бъдат дефинирани т.нар. лексеми в програмните езици – ключови думи, идентификатори, оператори, разделители, константни.

Например регулярният израз $(0|1|2|3|4|5|6|7|8|9)^+$ дефинира целочислена константа, т.е. цяло число, съставено от една или повече цифри. Символът ‘+’ е специален символ в регулярния израз, който означава едно или повече повторения.

Регулярен е език, който се поражда с регулярна граматика.

2.5.2. Крайни автомати

Всеки регулярен израз и всеки регулярен език може да се представи с еквивалентен краен автомат без памет. Крайният автомат се състои от:

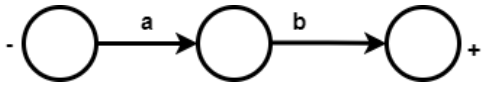
1) Краен брой състояния, едно от които е начално, а някои състояния (или нито едно) са крайни;

2) Азбука Σ от възможни входни символи;

3) Краен брой преходи, които дефинират за всяко състояние на автомата кое е следващото състояние, в което преминава автоматът при даден символ от азбуката.

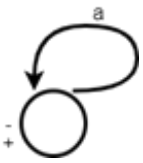
Крайният автомат се представя с краен ориентиран граф. Началното състояние се отбелязва със символ ‘-’, а крайните състояния с ‘+’. Крайният автомат разпознава низ w над азбука ∑, ако чрез конкатенация на символите на низа w се получава път в крайния ориентиран граф на автомата от начален до краен връх.

Пример 2. Регулярен израз R = ab се разпознава със следния краен автомат (фиг. 1).



Фиг. 1. Краен автомат за разпознаване на R = ab

Пример 3. Регулярен израз R = a* се разпознава със следния краен автомат (фиг. 2).



Фиг. 2. Краен автомат за разпознаване на R = a*

Пример 4:

Дадено е множеството от низове, които започват с последователност от символ **a** (нула, един или повече) и завършват с **bc**, т.е. регулярният израз описва множество {bc, abc, aabc, aaabc, aaaabc, …}.

а) Да се състави регулярен израз, който описва множеството

б) Да се построи краен автомат, който разпознава множеството

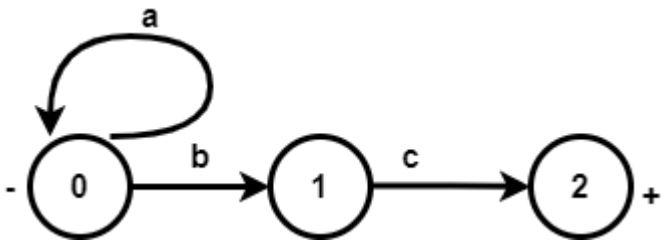
в) Да се напише програма, която реализира крайния автомат

Решение:

а) Регулярният израз, който описва множеството {bc, abc, aabc, aaabc, aaaabc, …} е

R = a^*bc.

б) Краен автомат за разпознаване на множеството {bc, abc, aabc, aaabc, aaaabc, …} (фиг. 3)



Фиг. 3. Краен автомат за разпознаване на R = a*bc

в) Примерна програма за реализация на крайния автомат

```
#include <iostream>
#include <cassert>
#include <string>
```

```
using namespace std;

typedef int fsm_state;
typedef char fsm_input;

bool is_final_state(fsm_state state)
{
   return (state == 2) ? true : false;
}

fsm_state get_start_state(void) { return 0; }

fsm_state move(fsm_state state, fsm_input input)
{
   // our alphabet includes only 'a', 'b' and 'c'
   switch (state)
   {
      case 0:
         if (input == 'a') { return 0; }
         else if (input == 'b') { return 1; }
         break;
      case 1:
         if (input == 'c') { return 2; }
         break;
       default:
         assert(0);
   }
}

bool recognize(string str)
{
   if (str == "") return false;

   fsm_state state = get_start_state();
   string::const_iterator i = str.begin();
   fsm_input input = *i;

   while (i != str.end())
   {
      state = move(state, *i);
      cout << "Current input : " << *i << endl;
      cout << "The automaton goes into state : " << state << endl;
      ++i;
   }

   if (is_final_state(state))
      return true;
   else
      return false;
```

```
}

// simple driver for testing
int main()
{
   recognize("aaaaaabc") ? cout << "String accepted" : cout << "String not accepted";
   return 0;
}
```

Тест с входен низ aaabc

```
Current input : a
The automaton goes into state : 0
Current input : a
The automaton goes into state : 0
Current input : a
The automaton goes into state : 0
Current input : b
The automaton goes into state : 1
Current input : c
The automaton goes into state : 2
String accepted
```

Крайният автомат разпознава низа aaabc, защото низът удовлетворява правилото дефинирано в регулярния израз R = a^*bc.

Тест с входен низ aaab

```
Current input : a
The automaton goes into state : 0
Current input : a
The automaton goes into state : 0
Current input : a
The automaton goes into state : 0
Current input : b
The automaton goes into state : 1
String not accepted
```

Крайният автомат не разпознава низа aaab, защото низът не удовлетворява правилото, дефинирано в регулярния израз R = a^*bc (липсва символ **c** в края на низа).

Обяснение на програмата

Програмата реализира краен автомат за разпознаване на регулярно множество, описано с регулярния израз R = a^*bc. Програмата чете входния низ символ по символ. Поредният прочетен символ се записва в променлива input. В зависимост от поредния прочетен символ автоматът преминава в ново състояние според графа на крайния

автомат (фиг. 3). Ако след прочитане на всички символи от входния низ автоматът се намира в крайно състояние (връх 2 на фиг. 3), това означава, че автоматът разпознава входния низ. В противен случай автоматът отхвърля входния низ.

Програмата се състои от следните функции:

1. is_final_state

Тази функция проверява дали автоматът се намира в крайно състояние (връх 2 в графа на фиг. 3).

2. get_start_state

Функцията установява автомата в началното състояние (връх 0 в графа на фиг. 3).

3. move

Функцията осъществява преход на автомата в ново състояние в зависимост от поредния прочетен входен символ.

4. recognize

Функцията чете всички символи от входния низ и извиква функцията move за преход в ново състояние.

5. main

Main е главна функция за стартиране и тестване на програмата.

Контролни въпроси:

1. Какво представлява формалният език?

2. Какво е предназначението на формалната граматика?

3. Какви са подмножествата, които съставят формалната граматика?

4. Каква е разликата между терминален и нетерминален символ в граматиката?

Задачи:

1. Дадена е формална граматика G = {∑, N, P, S}

∑ = {a, b} – терминални символи

N = {S, A, B} – нетерминални символи

S – стартов символ

Правила P:

1. S -> aSBA
2. S-> abA

3. AB -> BA
4. bB -> bb
5. bA -> ba
6. aA -> aa

Граматиката поражда език $L(G) = \{a^n b^n a^n \mid n \geq 1\}$.

Да се приложат правилата на граматиката за извеждане на следните изречения:

а) aba

б) aabbaa

в) aaabbbaaa

2. Да се определи от какъв тип по класификацията на Хомски са следните граматики.

а) G = {∑, N, P, S}, ∑ = {a, b, c}, N = {S, A, B}, S – стартов символ

Правила P:

1. S -> aS
2. S -> aA
3. A -> bA
4. A -> bB
5. B -> cB
6. B -> c

Граматиката G поражда език $L = \{a^i b^k c^l \mid i, k, l \geq 1\}$.

б) G = {∑, N, P, S}, ∑ = {a, b, c}, N = {F, E}, F – стартов символ

Правила P:

1. F -> aF
2. F -> aE
3. E -> bEc
4. E -> bc

Граматиката G поражда език $L = \{a^i b^k c^k \mid i, k \geq 1\}$.

Изведете по два произволни низа от всяка граматика. По какво се различават низовете (и съответно езиците), породени от двата типа граматики?

3. Правилото за валидно име на променлива в програмните езици е следното – името на променлива задължително започва с буква, следвана от последователност от букви и/или цифри. За опростяване на задачата приемаме, че буквите са **a** и **b**, а цифрите са **0** и **1**.

а) Да се състави регулярен израз, който описва правилото за валидно име на променлива

б) Да се построи краен автомат, който разпознава множеството от валидни имена на променливи

в) Да се напише програма на език по избор, която реализира крайния автомат за разпознаване на валидните имена на променливи

Упътване. Вж. пример 4 в упражнението.