Formális nyelvek és fordítóprogramok

Programozási feladatok (laborfeladatok)

Bodó Zalán

Babeș–Bolyai Tudományegyetem
Matematika és Informatika Kar

Tartalom

I. Formális nyelvek és automaták

I.1.

Egy programban oldjuk meg:

(a) Írjunk eljárást, amely kizárja a determinisztikus véges automatából a nem elérhető állapotokat.

(b) Írjunk eljárást, amely kizárja a determinisztikus véges automatából a nem produktív állapotokat.

A bemeneti állomány alakja:

- első sor: állapotok, szóközökkel elválasztva
- második sor: bemeneti ábécé elemei, szóközökkel elválasztva
- harmadik sor: kezdő állapot
- negyedik sor: végállapotok, szóközökkel elválasztva
- következő sorokban egy-egy átmenet: állapot bemenet állapot, szóközökkel elválasztva

A kimeneti formátum megegyezik a bemeneti formátummal, ahol a reprezentált automatából már kizártuk az élérhető vagy produktív állapotokat, illetve a hozzájuk tartozó átmeneteket.

I.2.

Vizsgáljuk meg, hogy két véges determinisztikus automata ekvivalens-e.
A bemeneti állomány formátumához lásd az előző feladatot.

I.3.

Írjunk programot, amely nemdeterminisztikus véges automata esetén vizsgálja, hogy az felismer-e egy adott szót.
A bemeneti állomány alakja:

- első sor: állapotok, szóközökkel elválasztva
- második sor: bemeneti ábécé elemei, szóközökkel elválasztva
- harmadik sor: kezdő állapotok, szóközökkel elválasztva
- negyedik sor: végállapotok, szóközökkel elválasztva
- következő sorokban egy-egy átmenet: állapot bemenet állapotok, szóközökkel elválasztva

I.4.

Reguláris kifejezés átalakítása nemdeterminisztikus automatává. Egy reguláris kifejezés csak *konkatenáció*, *vagy* (| vagy + szimbólumok) és *iteráció* (* szimbólum) műveleteket tartalmazhat, illetve kerek zárójeleket a csoportosítás végett. A program bemenetét fájlból vagy akár a standard inputról is kaphatja. A kimenet formátumához lásd az előző feladat bemenetének leírását.

I.5.

Tetszőleges véges automata grafikus megjelenítése. A megjelenítéshez használjuk/használhatjuk a `Graphviz` nevű alkalmazás `dot` nevű programját (http://www.graphviz.org). A feladat tulajdonképpen `dot` forráskód generálását jelenti. A bemenet fájból történik.

Példa

Bemenet:

```
1 2 3
a b
1
3
1 a 2
2 b 1
2 b 3
```

DOT kód:

```
digraph G{
 ranksep=0.5;
 nodesep=0.5;
 rankdir=LR;
 node [shape="circle",fontsize="16"];
 fontsize="10";
 compound=true;

i1 [shape=point, style=invis];
3 [shape=doublecircle];

i1 -> 1;
1 -> 2 [label=a];
2 -> 1 [label=b];
2 -> 3 [label=b];
}
```

I.6.

Vizsgáljuk meg, hogy egy veremautomata felismer-e egy adott szót (végállapottal vagy üres veremmel).
A bemeneti állomány alakja:

- 1. sor: állapotok, szóközökkel elválasztva
- 2. sor: bemeneti ábécé elemei, szóközökkel elválasztva
- 3. sor: veremábécé elemei, szóközökkel elválasztva
- 4. sor: kezdőállapot
- 5. sor: veremmemória kezdőjele
- 6. sor: végállapotok, szóközökkel elválasztva
- következő sorokban egy-egy átmenet:

$$q_i \;\; a \;\; z_i \;\; z_{i1} z_{i2} \ldots z_{ik} \;\; q_j$$

vagyis állapot, bemeneti szimbólum, veremszimbólum, új veremszimbólumok, új állapot.

I.7.

Reguláris nyelvtan átalakítása véges automatává, az automata megjelenítése (itt fel kell használni az automata megjelenítéshez kapcsolódó feladatot).
A nyelvtant a következő módon adjuk meg a bemeneti állományban:

- 1. sor: nemterminális szimbólumok
- 2. sor: terminális szimbólumok
- 3. sor: kezdőszimbólum
- következő sorok: szabályok a köv. formában:

$$A\ a\ B$$

vagy

$$A\ a$$

ahol A, B nemterminális, a terminális szimbólum.

A kimeneti állományhoz lásd az I.3. feladatot.

II. Fordítóprogramok

Amit a fordítóprogramnak ismernie kell:

- 2 alaptípus (egész és valós) + tömb típus
- változók deklarálása
- változó érvényességi körének állíthatósága
- értékadás
- típuskonverzió
- legkevesebb 3 aritmetikai művelet, zárójelek használata az ezekkel felépített kifejezésekben (pl. összeadás, kivonás, szorzás)
- kiírás, beolvasás
- `if-then-else` + `while` utasítás (egymásba ágyazás!)
- logikai relációk az `if` utasítás feltételében; legkevesebb 2 ilyen (pl. == és !=)
- logikai összekötők; legalább 2 ilyen (pl. a logikai és és a tagadás)
- hibajelzés + hibaelfedés

A fordítóprogram szimbólumainak, utasításainak megválasztása tetszőleges.

Kiértékelés

- az utolsó feladatnál:
  - 80%: pontozás a megvalósított funkcionalitások/feature-ök függvényében
  - 20%: a nyelv ötletessége, újszerűsége

II.1.

Listázó program elkészítése `Flex`-szel.
Ehhez természetesen szükséges a programnyelv elgondolása, előzetes megtervezése, hogy tudjuk, milyen terminális szimbólumok (lexikális egységek, tokenek) fognak szerepelni benne. A listázó program kimenete olyan szöveges fájl legyen, melyben minden sor a programnyelv egy terminális szimbólumát tartamazza a következő formában:

```
[sor: <x>, oszlop: <y>, hossz: <z>] <terminális szimbólum>
```

ahol `<x>`, `<y>` és `<z>` a terminális szimbólum sora a bemeneti fájlban, a szimbólum kezdeti karakterpozíciója az illető sorban, illetve a lexikális egység hossza, a `<terminális szimbólum>` pedig maga a terminális szimbólum.
(A listázó programban használhatjuk az `yylineno` változót.)

II.2.

A többkarakteres lexikális egységek elnevezése + a listázó program módosítása, hogy a `Bison`-nak vissza is térítse a tokeneket + a grammatika megírása.
Tervezzük meg a programnyelv grammatikáját használva a már definiált tokeneket, majd fordítsuk egybe a `Flex` és `Bison` által generált elemzőket. Küszöböljük ki a megjelenő léptetés/redukció és redukció/redukció típusú konfliktusokat.

II.3.

Szimbólumtábla elkészítése.
A szimbólumok hash-tábláját (többek között) arra fogjuk használni, hogy lássuk, már deklarálva volt-e egy hivatkozott változó – azaz a program egy adott pontján hivatkozhatunk-e a változóra –, illetve, hogy nem használtuk-e már ugyanezt a változónevet deklaráláskor.
A szintaktikus elemzés végén listázzuk a szimbólumtáblát a következő formában:

```
<szimbólum> -> <típus>, ...
```

Ez a lépés már szemantikai műveletek hozzáadását jelenti.

II.4.

Assembly kód generálása.
Írjunk **két** (rövidebb) programot az általunk készített programozási nyelvben (pl. prímszámteszt, n-edik Fibonacci szám kiszámítása, legnagyobb közös osztó meghatározása, valamely rendezési algoritmus stb.), bemutatva ezzel a nyelv tulajdonságait, képességeit.