Tietorakenteiden harjoitustyö

Tuomas Tynkkynen

Aiheen määrittely

Säännöllisten lausekkeiden kääntäjä/tulkki. Säännöllinen lauseke tässä tapauksessa rajautuu säännöllisiin kieliin, eli rajataan perusoperaatioihin (konkatenaatio, yhdiste ja Kleenen tähti).

Algoritmit

Tulkitsemiskeinona olisi tarkoitus olla muunnos DFA:han, käyttämällä yleisesti tunnettuja menetelmiä säännöllisen lausekkeen jäsennyspuun muunnoksesta NFA:ksi, josta muunnos DFA:ksi käyttämällä ‘osajoukkomenetelmää’ merkkijonon tunnistamisen yhteydessä.

Tällä olisi tarkoitus pyrkiä toteutukseen, joka lausekkeen kääntämisen jälkeen tunnistaa merkkijonon O(n) ajassa merkkijonon pituuden suhteen. Tilavaativuus vastaavasti riippumaton tunnistettavan merkkijonon pituudesta. DFA:n käyttö tunnistukseen tosin vie tilavaativuuden O(2^k) lausekkeen pituuden suhteen.

Syöte

Säännöllisen lausekken syntaksi

Kuten edellä mainittiin, pitäydytään säännöllisten kielen luokan ilmaisuvoimaisuudessa. Syntaksina samanlainen kuin useimmissa ohjelmointikielten regex-toteutuksissa. Eli siis yhdisteoperaattorina ‘|’, toisto- ja valinnaisuusoperaattorit ‘*’, ‘+’ ja‘?’. Merkkiluokkia voi antaa samaan tapaan hakasulkeissa ‘[]’, joiden sisällä yksittäisiä merkkejä tai välejä (esim. a-z), sekä ‘.’ tunnistaa minkä tahansa aakkoston merkin. Operaattoreiden soveltamisjärjestys sama kuin normaalisti, eli ensin tähti + muut, sitten konkatenaatiot, ja viimeisenä yhdisteet. Sulkeilla voi ryhmitellä kuten normaalisti.

```
regex -> konkat '|' regex | konkat
konkat -> määrä konkat | määrä
määrä -> päätemerkki määrä_symboli
määrä_symboli -> '+' | '?' | '*' | epsilon
päätemerkki -> '(' regex ')' | merkki | '.' | merkkijoukko
merkkijoukko -> '[' '^'? (merkki ('-' merkki)?)* ']'
```

Syötetiedoston muoto, käyttäminen

Ohjelma toimii grep-tyylisesti, eli ohjelmalle annetaan komentoriviparametrina säännöllinen lauseke, sekä syötetiedosto. Ohjelma tulostaa syötetiedostosta kaikki ne rivit, jotka kuuluvat annetun lausekkeen tunnistamaan kieleen.

Testaustiedosto

Oikean toiminnallisuuden varmistamiseksi ohjelmassa on testausominaisuus. Testitiedostossa määritellään testejä useille eri lausekkeille, jolle voidaan antaa valinnaisesti vertailtavaksi sama lauseke postfix-muodossa, jolla testataan oikea lausekkeen jäsennys, sekä merkkijonoja, joiden kuuluvuutta lausekkeen määrittämään kieleen testataan.
Esimerkkitiedosto:

```
=(a|b)*=ab|*
+
+a
+aa
+bb
+ab
-c
-ac
=a*|b*=a*b*|
+
+a
+aa
+bb
-ab
-ba
-c
```

Testattava lauseke valitaan syntaksilla **=***lauseke***=***[postfix-muoto], postfix-muodossa konkatenaatiota ilmaisee #. (*Postfix-muodon antaminen siis vapaaehtoista) Testiohjelma kääntää kyseisen lausekkeen, ja testi epäonnistuu, jos kääntäminen epäonnistui tai lausekkeen postfix-muoto ei vastaa oletettua.

Lausekkeen valinnan jälkeen voidaan antaa merkkijonoja, joilla lauseketta testataan. Rivi **+***mjono* kertoo, että lausekkeen tunnistamaan kieleen pitäisi kuulua merkkijono mjono, vastaavasti **-** tarkoittaa, että kyseistä merkkijonoa ei tule hyväksyä.

Rakenne

Aputietorakenteet

- List - yhteen suuntaan linkitetty lista, pääasiassa Map:n hajautusketjuihin
- Map - assosiatiivinen taulukko, toteutettu hajautustauluilla
- Set - matemaattinen joukko

Tilakoneiden rakennuspalikat

- Node - DFA:n tai NFA:n solmu
- Transition - irtonainen siirtymä solmujen välillä, josta puuttuu kohdesolmu

NFA:n luonti

- Lexer - jaottelee semanttisesti syötelausekkeen operaattoreihin ja merkkijoukkoihin
- Parser - jäsentää ja rakentaa NFA:n lausekkeesta, tuottaa myös postfix-muodon testausta varten

Tunnistus & ajaminen

- DfaMatcher - merkkijonojen tunnistus muuntamalla NFA:n DFA:ksi laiskasti ja simuloimalla muodostunutta DFA:ta
- Main - komentoriviparametrien käsittely, grep-toiminnallisuus
- Tester - testaustoiminnallisuus
- Debug - debugtulosteita. Ei käytössä tuotantoversiossa

Testaus

Edellä mainittiin testausominaisuus, joka testaa sekä oikeaa jäsentämistä, sekä merkkijonojen tunnistusta. Lisäksi eräitä monimutkaisempia aputietorakenteita on testattu yksikkötesteillä, jotka sijaitsevat tiedostoissa *Test.py, ja ne ajetaan python-tulkilla aivan normaalisti.

Jatkokehittelyä

- Aakkosto on mielivaltaisesti rajoitettu kirjaimiin a-z, ja sitä voisi aivan hyvin laajentaa kaikkiin asciii-merkkeiin
- DFA- ja NFA-solmut pitäisivät olla eri luokkina, koska ne loppujen lopuksi eroavat toiminnallisuudeltaan enemmän kuin kävi mielessä
- Merkkijoukot voisi toteuttaa fiksummalla tavalla, varsinkin jos aakkoston kokoa kasvatetaan. Nykyisellä toteutuksella merkkijoukko (esim. [a-z]) kuluttaa muistia lineaarisesti suhteessa aakkosten lukumäärään. Parempi tapa olisi esim. niiden säilytys jonkinlaisessa puurakenteessa NFA-solmuissa
- Muunnoksen DFA:ksi voisi tehdä kokonaan yhdellä kerralla, jolloin sen voisi optimoida ja

mahdollisesti seriailisoida tiedostoon myöhempää käyttöä varten

Lähteitä

- Jäsennys perustuu Ohjelmointikielten periaatteet-kurssin materiaaleihin osoitteessa https://www.cs.helsinki.fi/i/vihavain/k10/okk/content3.html
- +- ja ?-operaattoreiden NFA-muodostus, cachetusidea http://swtch.com/~rsc/regexp/regexp1.html
- teoria http://www.cs.helsinki.fi/u/floreen/lama/luennot.pdf