Harri Pitkänen

Transduktorit ja rekisteritransduktorit

Tietotekniikan
pro gradu -tutkielma
24. elokuuta 2008

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Harri Pitkänen
Yhteystiedot: hatapitk@iki.fi
Työn nimi: Transduktorit ja rekisteritransduktorit
Title in English: Transducers and registered transducers
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 81
Tiivistelmä: Transduktori on äärellisen automaatin laajennus, jolla voidaan esittää relaatioita säännöllisten kielten välillä. Transduktoreja käytetään muun muassa luonnollisten kielten sanojen tunnistamiseen ja generointiin. Rekisteritransduktorissa transduktoriin on liitetty äärellinen määrä apumuistia. Tutkielmassa esitetään notaatio rekisteritransduktorien kuvaamiseksi säännöllisillä lausekkeilla ja arvioidaan rekisteritransduktorien tehokkuutta tavanomaisiin transduktoreihin verrattuna.
English abstract: A transducer is an extension of a finite state automaton that expresses a relation between two regular languages. Transducers are used for example in language technology to recognise and generate words. A registered transducer is a transducer with a finite amount of auxiliary memory. This thesis introduces a notation for representing registered transducers using regular expressions, and compares the efficiency of registered transducers and ordinary transducers.
Avainsanat: transduktori, rekisteritransduktori, äärellinen automaatti, rekisteriautomaatti, säännöllinen kieli, kieliteknologia
Keywords: transducer, registered transducer, finite automaton, registered automaton, regular language, language technology

Esipuhe

Kiinnostukseni kieliteknologiaan heräsi kesällä 2005, jolloin ryhdyin harrastusmielessä kehittämään vapaasti lisensoitua oikolukuohjelmaa suomen kielelle. Olin tuolloin vielä fysiikan opiskelija, mutta lopulta ohjelmistokehitys vei mukanaan, ja päädyin vaihtamaan pääaineenikin tietotekniikkaan.

Harrastuksestani johtuen valitsin transduktorit kandidaattitutkielmani aiheeksi keväällä 2007. Tästä huolimatta pro gradu -tutkielman aiheen valinta samana kesänä ei ollut aivan helppoa. Aihepiiri kiinnosti edelleen, mutta epäilin samalla, miten pitkälle tietoni ja taitoni riittäisivät aiheen tutkimuksessa. Päädyin lopulta keskittymään tässä tutkielmassa rekisteritransduktoreihin, jotka vaikuttivat hyödyllisiltä juuri suomen kielessä tyypillisten sananmuodostussääntöjen kuvaamisessa.

Nyt vuotta myöhemmin voin helpottuneena todeta, ettei työn valmistuminen kaatunut ylitsepääsemättömiin vastoinkäymisiin. Kiitos tästä kuuluu monille, jotka eri tavoin ovat edesauttaneet työn valmistumista. Sekä kandidaatitutkielmaani että tätä pro gradu -tutkielmaa ohjannut Jarmo Ernvall auttoi työn suunnan määrittelyssä ja antoi arvokasta palautetta sen sisällöstä. Voikko-oikolukuohjelmiston käyttäjiltä ja muilta kehittäjiltä saamani palaute on auttanut minua ymmärtämään paremmin niitä lingvistisiä ongelmia, joihin tässä tutkielmassa etsitään teknisiä ratkaisuja. Kiitän myös ollut vanhemmiltani Kaijalta ja Pentiltä sekä ystäviltäni, erityisesti Riikalta, saamastani kannustuksesta.

Sanasto

Aakkosto (*alphabet*) Niiden symbolien joukko, joista merkkijonot muodostuvat.

Deterministinen (*deterministic*) Automaatti on deterministinen, jos sen tilalta voi lähteä korkeintaan yksi sallittu siirtymä millä tahansa annetulla syötteellä.

Formaali kieli (*formal language*) Aakkostosta muodostettujen merkkijonojen osajoukko.

FST (*Finite State Transducer*) Äärellinen transduktori.

FSRT (*Finite State Registered Transducer*) Rekisteritransduktori.

Katenaatio (*concatenation*) Merkkijonojen tai transduktoreiden liittäminen peräkkäin.

Kieliopillinen sana (*grammatical word*) Merkkijono, joka kuvaa symbolisesti luonnollisen kielen sanan rakennetta. Koostuu tavallisesti sanan perusmuodosta ja taivutusmuotoja kuvaavista leimoista.

Leima (*tag*) Symboli, joka kuvaa sanan kieliopillista ominaisuutta kuten sanaluokkaa tai taivutusmuotoa.

Luonnollinen kieli (*natural language*) Ihmisten välisessä kommunikaatiossa käytettävä puhuttu tai kirjoitettu kieli.

Merkkidiakriittit (*flag diacritics*) XFST-ohjelmiston rekisterioperaatioita vastaava toiminto.

Merkkijono (*string*) Äärellisestä symbolijoukosta valittujen symboleiden äärellinen järjestetty jono.

Morfologia (*morphology*) Sanojen rakennetta tutkiva kielitieteen osa-alue.

Pinoautomaatti (*pushdown automaton*) Abstrakti kone, joka tunnistaa kontekstittoman kielen.

Pintamuoto (*surface form*) Merkkijono, joka esittää luonnollisen kielen sanaa sen kirjoitetussa (tai puhutussa) muodossa.

Sana (*word*) Luonnollisen kielen sana. Joissakin formaaleja kieliä käsittelevissä teksteissä termillä ”sana” voidaan viitata muihinkin merkkijonoihin.

SFST Stuttgartin yliopisto kehittämä ohjelmisto transduktorien käsittelyyn.

Säännöllinen kieli (*regular language*) Formaali kieli, joka voidaan esittää säännöllisen lausekkeen avulla.

Tilasiirtymäkaavio (*state transition diagram*) Graafinen esitys automaattien ja transduktorien kuvaamiseksi.

Äärellinen automaatti (*finite automaton*) Abstrakti kone, joka tunnistaa säännöllisen kielen.

Äärellinen kieli (*finite language*) Formaali kieli, joka koostuu äärellisestä merkkijonojen joukosta.

XFST Xeroxin kehittämä ohjelmisto äärellisten automaattien ja transduktorien käsittelyyn.

Sisältö

Esipuhe **i**

Sanasto **ii**

1 Johdanto **1**

2 Automaatit ja transduktorit **3**
2.1 Formaalien kielten käsitteitä ja merkintöjä 3
2.2 Säännölliset lausekkeet 4
2.3 Äärellinen automaatti 5
2.4 Transduktorin määritelmä 6
2.5 Transduktorit tilasiirtymäkaavioina 8
2.6 Transduktorien ominaisuuksia ja erikoistapauksia 10
2.6.1 Yksikäsitteiset transduktorit 10
2.6.2 Kirjaintransduktori 10
2.6.3 Mealyn tilakone 11
2.6.4 Yleistetty jonokone 11
2.6.5 Sekventiaalinen transduktori 11
2.6.6 P-subsekventiaalisuus 12
2.7 Transduktoriin liittyvät automaatit 13
2.8 Transduktoreille tehtäviä muunnoksia 14
2.8.1 Käänteistransduktorin muodostaminen 14
2.8.2 Yhdistetyt transduktorit 14
2.8.3 Transduktorien yhdiste ja katenaatio 15
2.8.4 Transduktorien leikkaus 15
2.9 Transduktorien optimointi 15
2.9.1 Sekventiaalisten transduktorien minimointi 17

3 Rekisteritransduktorit **20**
3.1 Tavanomaisten transduktorien rajoituksia 20
3.2 Rekisteritransduktori ja -automaatti 22
3.3 Rekisteritransduktorien tilasiirtymäkaaviot 23
3.4 Rekisteritransduktorien optimointi 25

3.5 Muita automaattien ja transduktorien laajennuksia 29
3.5.1 Merkkidiakriitit 29
3.5.2 Pinoautomaatit ja -transduktorit 30
3.5.3 Kontekstittoman kielen jäsentäminen transduktorilla 32

4 Transduktorien sovelluksista 34
4.1 Transduktorien käyttö kieliteknologiassa 34
4.1.1 Sanojen morfologinen analyysi 34
4.1.2 Sanojen generointi 37
4.1.3 Konekääntäminen 37
4.1.4 Puheentunnistus 38
4.2 Transduktorien muita sovelluksia 39
4.3 Rekisteritransduktorit kieliteknologiassa 39
4.4 Ohjelmointiteknisiä ongelmia 41

5 Rekisteritransduktorin toteutus 43
5.1 Tilasiirtymäverkot kaarilistoina 43
5.2 Rekisteritransduktorit säännöllisinä lausekkeina 44
5.3 Transduktorin muodostaminen säännöllisestä lausekkeesta 47
5.3.1 Säännöllisten lausekkeiden jäsentäminen 47
5.3.2 Transduktorin muodostaminen lausekkeiden jäsennyksestä . 51
5.4 Transduktorin optimointi 52
5.5 Ohjelman rakenne ja toiminta 55
5.6 Rajoitettu toisto 57
5.7 Koejärjestely 59
5.8 Testituloksia 60

6 Johtopäätökset 66

7 Lähteet 68

Liitteet

A Rekisteritransduktorien rakentaminen osatransduktoreista 71

B Merkkijonon muuntaminen rekisteritransduktorissa 73

1 Johdanto

Useissa tietokoneohjelmistoissa käsitellään ja muunnetaan merkkijonoja. Yleensä nämä muunnokset ovat ohjelmoijan kannalta toteutuksen yksityiskohtia, joihin ei suuresti kiinnitetä huomioita. Joissakin sovelluksissa merkkijonomuunnokset ovat kuitenkin hyvin oleellisessa asemassa, jolloin myös niiden suorittamiseen ja muunnossääntöjen määrittelyn menetelmiin on kiinnitettävä erityistä huomiota. Eräs tällainen sovellusalue on luonnollisia kieliä käsittelevät ohjelmistot. Esimerkiksi taivutettujen suomen kielen sanojen muuntaminen luotettavalla tavalla perusmuotoonsa vaatii laajan säännöstön, johon on koodattava tieto kaikista oleellisista kielen sanoista ja niiden mahdollisista taivutusmuodoista.

Tässä pro gradu -tutkielmassa käsitellään erästä merkkijonomuunnosten toteutuksessa käytettyä tekniikkaa, äärellisiä transduktoreja sekä erityisesti niistä edelleen kehitettyjä rekisteritransduktoreja. Nämä molemmat ovat äärellisten automaattien laajennuksia, joilla on mahdollista toteuttaa relaatioita kahden säännöllisen kielen välillä. Tutkielman aihe liittyy kirjoittajan vuonna 2007 kirjoittamaan äärellisiä transduktoreja käsitelleeseen kandidaattitutkielmaan.

Tutkielman ensimmäisessä luvussa määritellään äärellinen automaatti, transduktori sekä muita oleellisia formaalien kielten käsitteitä. Lisäksi luvussa esitellään oleellisimmat transduktorien erikoistapaukset sekä tärkeimmät tavat transduktorien yhdistelyyn. Luvussa annetaan myös esimerkkejä yksinkertaisten transduktorien esittämisestä tilasiirtymäkaavioiden avulla.

Luvussa 3 perehdytään transduktorien rajoituksiin ja menetelmiin, joilla näitä rajoituksia voidaan kiertää. Erityisesti luvussa esitellään rekisteritransduktorit eli transduktorit, joihin on liitetty rajoitettu määrä apumuistia. Vertailun vuoksi luvussa kerrotaan myös muista transduktorien laajennuksista, joista osa on ilmaisuvoimaltaan aidosti äärellisiä transduktoreja voimakkaampia, mutta vaativat toimiakseen rajoittamattoman määrän muistitilaa. Luvussa käsitellään myös rekisteritransduktorien optimointiin liittyviä menetelmiä.

Luvussa 4 käsitellään transduktorien sovelluksia erityisesti kieliteknologian piirissä, jossa ne ovat saavuttaneet suuren suosion niin kirjoitetun kielen analysoinnissa kuin myös esimerkiksi puheen tunnistuksessa. Kieliteknologiassa transduktorien käytön tekee houkuttelevaksi se, että niiden avulla on mahdollista koodata suuri määrä lingvististä tietoa suhteellisen pieneen tilaan ja toteuttaa hyvin tehok-

kaita algoritmeja syötetiedon käsittelyyn. Joissakin tilanteissa transduktorien rajat tulevat tälläkin sovellusalueella vastaan, ja näissä tilanteissa rekisteritransduktorit voivat osoittautua hyödyllisiksi.

Tutkielman teoriaosassa esiteltyjen tekniikoiden avulla toteutettiin sovellus, jonka avulla rekisteritransduktoreja voi muodostaa, optimoida, havainnollistaa ja käyttää merkkijonojen muuntamiseen. Tämän sovelluksen toteutus kuvataan luvussa 5. Sovellusta varten kehitettiin merkintätapa rekisteritransduktorien esittämiseksi säännöllisten lausekkeiden avulla. Luvussa esitetään myös tuloksia kokeista, joissa toteutetun sovelluksen tehokkuutta verrattiin erään valmiin transduktoriohjelmiston (Stuttgart Finite State Transducer Tools) avulla saatuihin tuloksiin.

Tutkielman viimeisessä luvussa 6 pohditaan tutkimuksen tuloksia ja esitetään muutamia tarkemman tutkimuksen arvoisia kysymyksiä, joihin tämän tutkielman puitteissa ei saatu vastausta.

2 Automaatit ja transduktorit

Koska äärelliset transduktorit ovat äärellisten automaattien laajennus, luvussa 2.1 esitellään automaateille ja transduktoreille yhteisiä käsitteitä ja käytettyjä merkintätapoja. Luvussa 2.2 kerrotaan säännöllisistä lausekkeista, joiden avulla mikä tahansa säännöllinen kieli on mahdollista kuvata äärellisen mittaisen merkkijonon avulla. Luvussa 2.3 määritellään äärellinen automaatti ja luvussa 2.4 äärellinen transduktori. Määritelmän selventämiseksi luvussa 2.5 kerrotaan, kuinka transduktoreja voi havainnollistaa tilasiirtymäkaavioiden avulla.

Luvussa 2.6 annetaan määritelmät eräille oleellisille transduktorien erikoistapauksille sekä ominaisuuksille, joita kaikilla transduktoreilla ei ole. Luvussa 2.7 tutustutaan transduktorien ja automaattien väliseen yhteyteen. Oleellisimmat transduktoreille suoritettavat muunnosoperaatiot kuvataan luvussa 2.8, ja lopuksi luvussa 2.9 kerrotaan transduktorien optimoinnin menetelmistä.

2.1 Formaalien kielten käsitteitä ja merkintöjä

Merkkijono (engl. *string*) on äärellisestä symbolijoukosta valittujen symboleiden äärellinen järjestetty jono [12, 1.1]. Symbolijoukko eli *aakkosto* (engl. *alphabet*) voi olla esimerkiksi ”suomen kielen kirjaimet”, jolloin siitä muodostetut merkkijonot voivat olla suomen kielen sanoja ($''hevonen''$) tai merkityksettömiä kirjainyhdistelmiä ($''kdrhk''$). Jatkossa merkkijonot esitetään pienillä, kursivoiduilla kirjaimilla tai lainausmerkeissä, mikäli tämä on selvyyden vuoksi tarpeellista. Jos merkkijonoissa käytetään muita symboleja kuin tavallisia kirjaimia, ne esitetään kulmasuluissa $\langle ... \rangle$. Tyhjän merkkijonon symbolina käytetään kreikkalaista kirjainta ϵ (epsilon). Merkkijonomuuttujat merkitään pienillä kursivoiduilla kirjaimilla ilman lainausmerkkejä.

Merkkijonojen *katenaatio* (engl. *concatenation*) tarkoittaa niiden liittämistä peräkkäin. Esimerkiksi merkkijonoista $a = ''kerros''$ ja $b = ''talo''$ voidaan muodostaa katenaatio $ab = ''kerrostalo''$.

Aakkostosta muodostuvien merkkijonojen osajoukkoa kutsutaan *formaaliksi kieleksi* (engl. *formal language*). Laskennan teoriassa on usein tapana ymmärtää sanan *kieli* tarkoittavan aina formaalia kieltä. Koska tässä tutkielmassa käsitellään paljon myös ihmisten välisessä kommunikaatiossa käytettäviä *luonnollisia kieliä*, käytetään näistä käsitteistä täydellisiä nimiä, jos sekaannuksen vaara on olemassa. Luonnolli-

set kielet eivät sellaisenaan täytä formaalein kielen määritelmää, sillä niissä kieleen kuuluvien ja sen ulkopuolelle jäävien merkkijonojen välille ei voida tehdä tarkkaa rajaa. Usein luonnolliset kielet määritellään esimerkiksi niiden puhujayhteisön tuottamien lauseiden joukkona, mutta tämäkin määritelmä on kovin epämääräinen [7]. Kielitieteessä kielellä tavallisesti tarkoitetaan lauseiden muodostamaa kieltä, mutta erityisesti tässä tutkielmassa käsitellään myös luonnollisten kielten sanojen muodostamia kieliä.

Formaaleja kieliä voidaan luokitella monella eri tavalla. Tavallisesti luokitukset perustuvat siihen, missä muodossa kieli voidaan esittää, tai millainen kone sen tunnistamiseen tarvitaan. *Äärelliset kielet* koostuvat äärellisestä joukosta merkkijonoja, joten ne voidaan kuvata yksinkertaisesti luettelemalla kieleen kuuluvat merkkijonot. (Tietenkään tämä ei aina ole käytännössä mahdollista, jos merkkijonojen määrä tai pituus on hyvin suuri.)

Äärettömästä määrästä merkkijonoja koostuvien formaalien kielten luokituksista Noam Chomskyn kehittämä Chomskyn hierarkiaksi kutsuttu luokitus on ehkä kaikkein tunnetuin. Chomskyn vuonna 1956 julkaisema artikkeli [6] esittelee kolme menetelmää kielioppien kuvaamiseen: säännölliset kielet (*finite state language* tai *regular language*), kaksi lauserakennekielioppien (*phrase structure grammar*) luokkaa (*derivable language* ja *terminal language*) sekä muunnoskieliopit (*transformational grammar*). Nykyään Chomskyn hierarkiaan esitetään tavallisesti kuuluvaksi neljä kielioppityyppiä [11, 2.3]: tyyppi 3 eli säännölliset kieliopit (*regular grammar*), tyyppi 2 eli kontekstittomat kieliopit (*context free grammar*), tyyppi 1 eli kontekstiset kieliopit (*context sensitive grammar*) ja tyyppi 0 eli rajoittamattomat kieliopit (*unrestricted grammar*). Näiden kielioppityyppien välillä vallitsee hierarkia siten, että kaikki tyypin 3 kieliopit ovat myös tyypin 2 kielioppeja, kaikki tyypin 2 kieliopit tyypin 1 kielioppeja ja kaikki tyypin 1 kieliopit tyypin 0 kielioppeja.

2.2 Säännölliset lausekkeet

Säännöllisille kielille tunnusomainen piirre on se, että ne voidaan aina kuvata *säännöllisen lausekkeen* (engl. *regular expression*) avulla [12, s. 28]. Vastaavasti kaikki säännölliset lausekkeet kuvaavat jotain säännöllistä kieltä. Olkoon säännöllisen lausekkeen a kuvaamaa kieli $L(a)$. Kaikki merkkijonot (myös ϵ) ovat itsessään säännöllisiä lausekkeita, jolloin ne esittävät kieltä, joka sisältää täsmälleen tuon kyseisen merkkijonon: $L(a) = \{a\}$. [1] Säännöllisiä lausekkeita voidaan yhdistellä uusiksi säännölli-

[1] Myös tyhjä kieli $\emptyset$ on säännöllinen, ja $L(\emptyset) = \emptyset$. Se on kuitenkin eri kieli kuin säännöllisen lausekkeen ϵ kuvaama kieli $L(\epsilon) = \{\epsilon\}$.

siksi lausekkeiksi kolmen perusoperaation avulla:

1. Lausekkeiden a ja b *yhdiste* (engl. *union*) $a|b$ (merkitään usein myös $a \cup b$) kuvaa kielen, johon sisältyvät sekä lausekkeen a että b kielet: $L(a|b) = L(a) \cup L(b)$.

2. Lausekkeiden a ja b *tulo* (engl. *cross product*) ab kuvaa kielen, joka sisältää kaikki sellaiset katenaatiot, joissa alkuosa on lausekkeen a merkkijono ja loppuosa lausekkeen b merkkijono: $L(ab) = L(a)L(b)$.

3. Lausekkeen a *sulkeuma* (engl. *closure* tai *Kleene closure*) $a^* = \epsilon|a|aa|aaa|...$, ts.

$$L(a^*) = \{\epsilon\} \cup \left(\bigcup_{i \geq 1} \left\{ \prod_{j=1}^{i} a \right\} \right)$$

   Jos Σ on aakkosto, Σ^* on kaikkien aakkoston symboleista yhdistelemällä saatavien merkkijonojen muodostama kieli.

Näiden kolmen operaation avulla voidaan rakentaa kaikki säännölliset lausekkeet, ja siten myös kaikki säännölliset kielet. Sulkumerkkien avulla voidaan osoittaa haluttu ”laskujärjestys”, mutta jos niitä ei ole käytetty, käsitellään ensin sulkeumat, sitten tulot ja lopuksi yhdisteet. Seuraavassa muutama esimerkki säännöllisistä lausekkeista:

- $(''piha''|''lehti'')(''koivu''|''puu'')$ tuottaa merkkijonot $''pihakoivu''$, $''pihapuu''$, $''lehtikoivu''$ ja $''lehtipuu''$.

- $''pikku''^*''serkku''$ tuottaa merkkijonot $serkku$, $pikkuserkku$, $pikkupikkuserkku$, $pikkupikkupikkuserkku$ jne.

Juuri säännölliset kielet ovat kieliä, joita äärellisillä automaateilla ja äärellisillä transduktoreilla voidaan käsitellä. On kuitenkin olemassa formaaleja kieliä, jotka eivät ole säännöllisiä. Esimerkiksi suurin osa ohjelmointikielistä on tällaisia.

2.3 Äärellinen automaatti

Äärellinen automaatti (engl. *finite automaton*) on abstrakti kone, joka tunnistaa säännöllisen kielen. Äärellinen automaatti määritellään tilasiirtymäverkkona viisikoksi $M = (Q, A, E, q_0, F)$, jossa Q on äärellinen joukko *tiloja* (engl. *state*), A on automaatin *syöteaakkosto* (engl. *input alphabet*), E on tilasiirtymien joukko, q_0 on alkutila ja F lopputilojen joukko. Lisäksi $q_0 \in Q$, $F \subseteq Q$ ja tilasiirtymät $e \in E$ voidaan esittää muodossa $e = (q, a, q')$, jossa q on siirtymän lähtötila, q' siirtymän kohdetila ja a

siirtymässä luettava merkistön A merkkijono. Tilasiirtymäjoukon E sijasta voidaan tilasiirtymät määritellä myös siirtymäfunktion $\delta : Q \times A \rightarrow Q$ avulla.

Automaatin *tilanteen* (engl. *configuration*) määrää pari $(q, w) \in Q \times A^*$. Tässä siis q on eräs automaatin tiloista ja w syötemerkkijonon jäljellä oleva osa. Automaatin alkutilanne syötteellä a on (q_0, a).

Automaatti on *deterministinen* (engl. *deterministic*), jos jokaiselta sen tilalta millä tahansa syötteellä lähtee korkeintaan yksi sallittu siirtymä [11, s. 82]. Jokaisesta epädeterministisestä äärellisestä automaatista voidaan muodostaa ekvivalentti deterministinen äärellinen automaatti. Epädeterminististä n-tilaista automaattia vastaava deterministinen automaatti voi pahimmillaan vaatia 2^n tilaa, mutta käytännön ongelmissa tämä tilojen määrän kasvu jää yleensä merkittävästi vähäisemmäksi [11, s. 146].

Äärellisen automaatin M *minimoinnilla* (engl. *minimisation*) tarkoitetaan sellaisen ekvivalentin äärellisen automaatin M' rakentamista, jossa tilojen määrä on mahdollisimman pieni. Äärellisen automaatin minimointitehtävän ratkaisuun on olemassa useita algoritmeja, jotka tyypillisesti perustuvat ekvivalenttien tilojen tunnistamiseen ja yhdistämiseen. Parhaat näistä algoritmeista ovat erittäin tehokkaita: esimerkiksi Hopcroftin algoritmi [24, 4.5] kykenee määrittämään tilojen ekvivalenssiluokat ajassa $\mathcal{O}(n \log n)$, jossa n on automaatin tilojen lukumäärä [24].

2.4 Transduktorin määritelmä

Äärellinen tai *äärellistilainen transduktori* (engl. *finite state transducer*) voidaan ajatella abstraktiksi koneeksi, joka toteuttaa matemaattisen relaation kahden säännöllisen kielen välillä. Esitetään tässä sille eräs konkreettinen määritelmä. Lawson [17] määrittelee äärellisen transduktorin kuusikoksi $T = (Q, A, B, q_0, E, F)$, jossa Q on äärellinen joukko tiloja, A on transduktorin syöte- ja B tulosteaakkosto, q_0 alkutila, E tilasiirtymien joukko ja F lopputilojen joukko. Tällöin $q_0 \in Q$, $F \subseteq Q$ ja tilasiirtymät $e \in E$ voidaan esittää muodossa $e = (q, a, b, q')$, jossa q on siirtymän lähtötila, q' siirtymän kohdetila, a siirtymässä luettava merkistön A merkkijono ja b tulosteeseen tuleva merkistön B merkkijono.

Transduktori muuntaa sille annetun syötemerkkijonon joukoksi tulostemerkkijonoja lähtien liikkeelle tilasta q_0. Alkutilasta sallittuja siirtymiä ovat kaikki siirtymät $(q_0, a, b, q') \in E$, joissa a on syötemerkkijonon alussa oleva merkkijono (voi myös olla tyhjä merkkijono), b mikä tahansa merkkijono ja q' mikä tahansa tila. Sallittuja siirtymiä voi olla yksi, enemmän kuin yksi tai ei yhtään. Jos niitä ei ole yhtään, syöte hylätään. Jos sallittuja siirtymiä on yksi, poistetaan syötemerkkijonon alusta merk-

kijono a, lisätään tulostemerkkijonon (joka alkutilassa on tyhjä) loppuun merkkijono b ja siirrytään tilaan q'. Mikäli sallittuja siirtymiä on enemmän kuin yksi, suoritus haarautuu siten, että jokainen siirtymä käsitellään erikseen kuten yhden sallitun siirtymän tapauksessa.

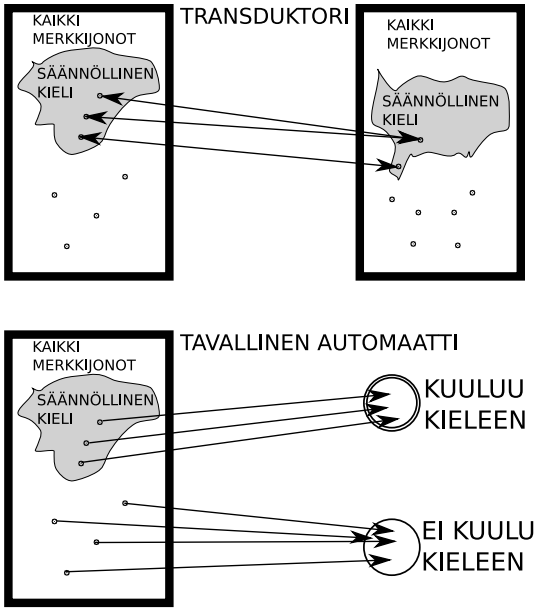
Muut siirtymät tapahtuvat täsmälleen samalla periaatteella kuin siirtymät alkutilasta. Ainoastaan siirtymän lähtötila voi olla toinen, jäljellä oleva syötemerkkijono voi olla lyhyempi ja tulostemerkkijonoon on voinut kertyä merkkejä. Siirtymiä jatketaan, kunnes syötemerkkijono on luettu loppuun eli jäljellä on vain tyhjä merkkijono. Mikäli tällöin ollaan jossakin transduktorin lopputiloista, transduktori hyväksyy syötteen ja tulostaa siirtymissä kertyneen tulostemerkkijonon. Jos taas päädyttiin muuhun kuin lopputilaan, syötettä ei hyväksytä (ainakaan kyseisen siirtymien muodostaman polun perusteella) eikä kertynyttä tulostemerkkijonoa tulosteta.

Jotta transduktori hyväksyisi syötteen, on ainakin yhden sallittujen siirtymien polun siis päädyttävä johonkin lopputilaan. Kaikki lopputilaan päätyneet polut tuottavat yhden tulostemerkkijonon, joita siis voi syntyä nolla (syöte hylätään) tai yksi tai useampia (syöte hyväksytään). Transduktori määrittelee siten *relaation* aakkoston A ja B merkkijonojen välille ($T : A^* \rightarrow B^*$), mutta tämä relaatio ei välttämättä ole funktio. Käytetään transduktorin T määrittelemästä relaatiosta merkintää $R(T)$, jossa $(a, b) \in R(T)$ jos ja vain jos syötemerkkijonoon a liittyy tulostemerkkijono b transduktorissa T.

Kuten automaateille, myös transduktoreille voidaan määritellä tilanne. Transduktorin tilanne on kolmikko $(q, w, v) \in Q \times A^* \times B^*$, jossa w on syötemerkkijonon jäljellä oleva osa ja v tulostemerkkijonon alkuosa. Transduktorin alkutilanne syötteellä a on (q_0, a, ϵ).

Ei ole aivan itsestään selvää, että yllä kuvatun koneen hyväksymät merkkijonot muodostavat aina säännöllisen kielen, ja että myös sen tulostamien merkkijonojen kieli on säännöllinen. Näin kuitenkin on. Tuloksen matemaattinen todistaminen on suhteellisen monimutkaista, joten todistus sivuutetaan tässä. Sen voi löytää esimerkiksi Berstelin kirjasta [5, s. 70]. Transduktorin määrittelemää relaatiota $R(T)$ kutsutaan *säännölliseksi relaatioksi* (engl. *regular relation* [2]) [17, s. 22]. Kaikki kahden säännöllisen kielen väliset relaatiot eivät kuitenkaan ole säännöllisiä. Jos relaatio $R : A^* \rightarrow B^*$ esitetään joukkona järjestettyjä pareja $(a, b) \in A^* \times B^*$, voidaan säännöllinen relaatio määritellä niiden avulla samaan tapaan kuin säännöllinen kieli määriteltiin merkkijonojoukon tapauksessa. Säännöllisillä relaatioilla on muitakin karakteristisia ominaisuuksia; Berstel on kirjassaan listannut näistä useita [5, s. 53 –

[2] Useissa lähteissä käytetään myös termiä *rational relation*. Käsitteillä on erilaiset määritelmät, mutta äärellisten aakkostojen tapauksissa ero ei ole merkitsevä [5, s. 48].



Kuva 2.1: Äärellisen transduktorin ja äärellisen automaatin formaaleille kielille suorittamat operaatiot kaaviokuvan muodossa. Äärellinen transduktori toteuttaa relaation kahden säännöllisen kielen välille, kun taas tavallinen äärellinen automaatti ainoastaan tunnistaa säännöllisen kielen.

55].

Äärellisten automaattien ja transduktorien keskinäistä suhdetta on havainnollistettu kuvassa 2.1.

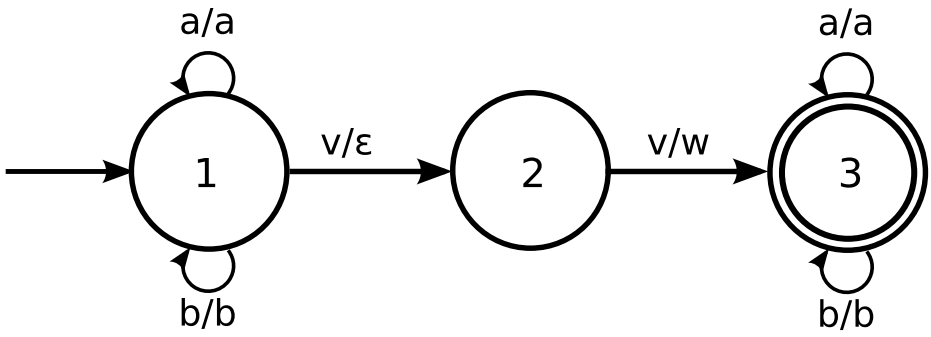
Samoin kuin äärellisen automaatin, myös transduktorin määritelmässä tilasiirtymien joukko E voidaan korvata siirtymäfunktiolla $\delta(q, a) : Q \times A^* \rightarrow \mathcal{P}(Q \times B^*)$, jossa q on siirtymän lähtötila ja a siirtymään liittyvä syötemerkkijono. Näiden esitysten välillä pätee yhteys

$$\delta(q, a) = \{(q', b) \in Q \times B^* : (q, a, b, q') \in E\}$$

2.5 Transduktorit tilasiirtymäkaavioina

Tilasiirtymäkaavio (engl. *state transition diagram*) tai lyhyemmin *tilakaavio* (eng. *state diagram*) on graafinen esitys automaattien ja transduktorien rakenteen ja toiminnan havainnollistamiseksi [11, s. 142]. Tilakaaviossa ympyrät ovat tiloja, ja näistä kaksinkertaisella reunaviivalla piirretyt lopputiloja. Alkutila merkitään ympyrää kohti osoittavalla erillisellä nuolella, joka ei lähde mistään tilasta.

Tilojen väliset nuolet kuvaavat tilasiirtymiä. Transduktoreja kuvaavissa kaavois-



Kuva 2.2: Yksinkertainen äärellinen transduktori esitettynä tilasiirtymäkaavion muodossa. Tilat voidaan esittää myös ilman numeroita, mutta tässä kuvassa ne on numeroitu selvyyden vuoksi.

sa siirtymien yhteyteen merkitään aina jakoviivan vasemmalle puolelle syötemerkkijonosta luettavat merkit ja oikealle puolelle tulostemerkkijonoon lisättävät merkit. Joskus erottimena käytetään jakoviivan sijasta kaksoispistettä. Merkintöjen lyhentämiseksi sellaisiin siirtymiin, joissa luetaan ja tulostetaan sama merkkijono, voidaan merkitä pelkästään kyseinen merkkijono ilman jakoviivaa.

Esimerkki tilasiirtymäkaaviosta on esitetty kuvassa 2.2. Kuvan transduktorin syöteaakkosto on $\{a, b, v\}$ ja tulosteaakkosto $\{a, b, w\}$. Transduktori hyväksyy sellaiset syötemerkkijonot, jotka sisältävät täsmälleen yhden merkkijonon $''vv''$ ja lisäksi vapaasti merkkejä a tai b. Tulostemerkkijono on muuten sama kuin syöte, mutta syötteen $''vv''$ korvautuu merkillä w.

Tarkastellaan esimerkin vuoksi, mitä tapahtuu syötemerkkijonolle $''avvb''$ kuvan 2.2 transduktorissa. Lähtötilana on tila 1, joten aluksi transduktorin tilanne on $(1, avvb, \epsilon)$. Tilasta 1 lähteviin siirtymiin liittyvät syötemerkkijonon osat ovat $''a''$, $''b''$ ja $''v''$. Ainoastaan siirtymä $(1, a, a, 1)$ on sallittu, joten siirtymän jälkeen transduktorin tilanne on $(1, vvb, a)$. Tästä edetään siirtymän $(1, v, \epsilon, 2)$ kautta tilanteeseen $(2, vb, a)$, siitä edelleen tilanteeseen $(3, b, aw)$ ja lopuksi tilanteeseen $(3, \epsilon, awb)$. Tila 3 on transduktorin lopputila, joten syötemerkkijono hyväksytään ja tulosteeksi saadaan $''awb''$.

Jos kuvan 2.2 transduktoriin syötetään merkkijono $''ab''$, päättyy suoritus tilanteeseen $(1, \epsilon, ab)$. Koska tila 1 ei ole lopputila, syöte hylätään. Syötteellä $''bvb''$ päädytään tilanteeseen $(2, b, b)$. Tilasta 2 lähtevä siirtymä ei tässä tilanteessa ole sallittu, joten syöte hylätään.

Myöhemmin esitettävissä tilakaavioissa käytetään suorakulmioita kuvaamaan sellaisia transduktorin osia, joiden yksityiskohtaisesta rakenteesta ei olla kiinnostuneita. Tilasiirtymän yhteyteen piirretty kysymysmerkki on lyhennemerkintä, joka

vastaa siirtymäjoukkoa $\{a/a : a \in A\}$. Tällöin on oltava $A \subseteq B$, mutta jottei seuraavassa esitettävä transduktorin suunnan vaihtaminen aiheuttaisi tulkintaongelmia, käytetään tätä lyhennemerkintää jatkossa vain transduktoreissa, joiden syöte- ja tulosteaakkostot ovat täsmälleen samat.

2.6 Transduktorien ominaisuuksia ja erikoistapauksia

2.6.1 Yksikäsitteiset transduktorit

Transduktori on *yksikäsitteinen* (engl. *unambiguous*) [4, 2.5], jos jokainen syötemerkkijono tuottaa korkeintaan yhden tulostemerkkijonon.

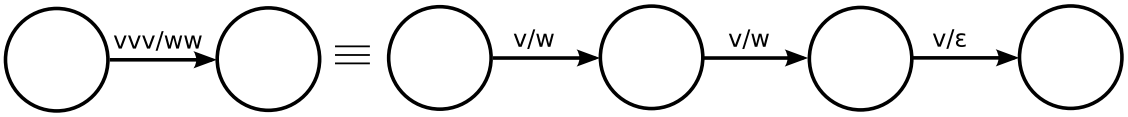
Jos transduktori ei ole yksikäsitteinen, se on *monikäsitteinen* (engl. *ambiguous*). Monikäsitteiset transduktorit voidaan edelleen luokitella kolmeen eri luokkaan sen mukaan, miten vakavasta monikäsitteisyydestä on kyse [21, 1.3.5]. Transduktori on *tasaisesti monikäsitteinen* (engl. *uniformly ambiguous*), jos on olemassa positiivinen kokonaisluku N siten, että kaikilla syötemerkkijonoilla transduktori tuottaa korkeintaan N tulostemerkkijonoa. Mikäli transduktori ei ole tasaisesti monikäsitteinen, mutta se tuottaa jokaiselle syötemerkkijonolle äärellisen määrän tulostemerkkijonoja, transduktoria kutsutaan *äärellisesti monikäsitteiseksi* (engl. *simply finitely ambiguous*). Tällöin ei ole mahdollista asettaa mitään kiinteää syötemerkkijonosta riippumatonta ylärajaa tulostemerkkijonojen määrälle.

Monikäsitteisyyden kannalta hankalin luokka on *äärettömän monikäsitteiset* (engl. *infinitely ambiguous*) transduktorit. Näissä ainakin yhdellä syötemerkkijonolla transduktori tulostaa äärettömän määrän tulostemerkkijonoja. Näin voi käydä, jos transduktori sisältää tilasiirtymäsilmukan, jossa kaikkiin siirtymiin liittyy tyhjä syötemerkkijono.

2.6.2 Kirjaintransduktori

Kirjallisuudessa esiintyy useita toisistaan jonkin verran poikkeavia määritelmiä äärellisille transduktoreille. Luvussa 2.4 esitettyä määritelmää voidaan muokata siten, että tilasiirtymissä syötteet ja tulosteet ovat yksittäisiä merkkejä tai tyhjiä merkkijonoja, eivät pituudeltaan mielivaltaisia merkkijonoja. Tällaista transduktoria kutsutaan *kirjaintransduktoriksi* (engl. *letter transducer*).

Kirjaintransduktorin on ilmaisuvoimaltaan ekvivalentti merkkijonosiirtymiä sisältävän transduktorin kanssa [21, 1.3.2]. Tämä nähdään helposti seuraavalla tavalla: Yksittäiset merkit ovat erikoistapaus mielivaltaisen mittaisesta merkkijonosta, jo-



Kuva 2.3: Mielivaltaisia merkkijonoja lukevat ja tulostavat siirtymät voidaan ekvivalentisti esittää jonona yksittäisiä merkkejä käsitteleviä siirtymiä. Mikäli syöte- ja tulostemerkkijonot ovat erimittaisia, lisätään siirtymiin tarvittava määrä tyhjiä merkkijonoja ϵ.

ten niitä sisältävät siirtymät ovat kelvollisia siirtymiä molempien määritelmien mukaan. Toisaalta mielivaltaisia merkkijonoja sisältävät siirtymät on aina mahdollista esittää jonona yksittäisiä merkkejä käsitteleviä siirtymiä. Tätä on havainnollistettu kuvassa 2.3.

2.6.3 Mealyn tilakone

Kirjaintransduktorin erikoistapaus on transduktori, jossa siirtymien syötteenä ja tulosteena on täsmälleen yksi ei-tyhjä merkki. Tällaista transduktoria kutsutaan *Mealyn tilakoneeksi* (engl. *Mealy machine*) [22, 2.4]. Mealyn tilakoneita käytetään mm. digitaalielektroniikassa synkronisten kytkentöjen mallintamiseen [20, 5.2]. Koska jokaisessa siirtymässä luetaan ja tulostetaan yksi merkki, Mealyn tilakoneen tekemä muunnos on pituuden säilyttävä: kaikki tulostemerkkijonot ovat saman mittaisia syötemerkkijonon kanssa. Normaalilla transduktorilla tällaista rajoitusta ei ole, joten nähdään, että Mealyn tilakoneet ovat ilmaisuvoimaltaan niitä aidosti heikompia.

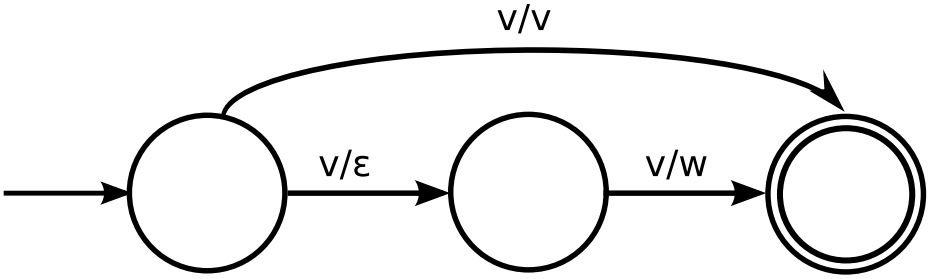
Mealyn tilakonetta kutsutaan myös *ϵ-vapaaksi kirjaintransduktoriksi* (engl. *ϵ-free letter transducer*) [21, s. 19].

2.6.4 Yleistetty jonokone

Transduktoria, jonka siirtymissä luetaan täsmälleen yksi syötemerkkijonon merkki mutta tulostetaan mielivaltaisen mittainen merkkijono, kutsutaan *yleistetyksi jonokoneeksi* (engl. *generalised sequential machine*) [22, 2.4][14, s. 5].

2.6.5 Sekventiaalinen transduktori

Transduktori on *sekventiaalinen* (engl. *sequential*) [19, 2.1], jos se on syötteen suhteen *deterministinen* (engl. *deterministic*) yleistetty jonokone. Syötteen suhteen deterministisyydellä tarkoitetaan sitä, että yhdeltäkään transduktorin tilalta ei voida millään



Kuva 2.4: Transduktori, joka muuntaa merkkijonon $''v''$ merkkijonoksi $''v''$ ja merkkijonon $''vv''$ merkkijonoksi $''w''$. Muut merkkijonot transduktori hylkää. Koska kuhunkin syötemerkkijonoon liittyy korkeintaan yksi tulostemerkkijono, transduktori on yksikäsitteinen. Se ei kuitenkaan ole sekventiaalinen, sillä lähtötilalta lähtee kaksi siirtymää, joiden syötemerkkijonona on $''v''$.

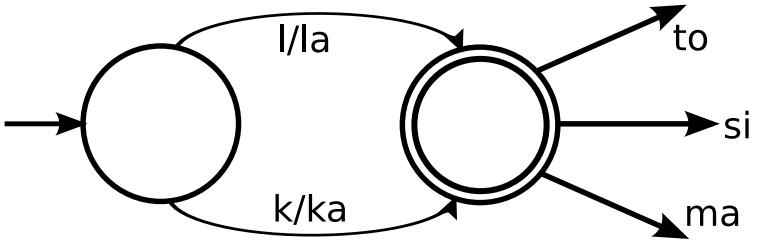
syötteellä valita kahta tai useampaa lähtevää siirtymää. Käytännössä tämä tarkoittaa sitä, että tilalta lähtevien siirtymien syötemerkit eivät voi olla samoja. Koska sekventiaalinen transduktori on yleistetyn jonokoneen erikoistapaus, myöskään tyhjä merkkijono ϵ ei ole sallittu sen tilasiirtymän syötemerkkijonona.

Sekventiaalisen transduktorin suoritus etenee haarautumatta yhtä polkua pitkin, joten se voi tuottaa korkeintaan yhden tulostemerkkijonon annetulle syötteelle. Sekventiaalinen transduktori on siis aina yksikäsitteinen. Ominaisuudet eivät kuitenkaan ole ekvivalentit, sillä kaikki yksikäsitteiset transduktorit eivät ole sekventiaalisia. Eräs tällainen transduktori on esitetty kuvassa 2.4.

Sekventiaalisuus on hyödyllinen ominaisuus käytännön sovelluksissa, sillä sekventiaalisen transduktorin läpikäyntiin tarvittavien tilasiirtymien määrä rajoittuu maksimissaan syötemerkkijonon pituuteen. Vaikka itse transduktori olisi sekventiaalinen, sen luvussa 2.8.1 määriteltävä käänteistransduktori ei sitä välttämättä ole.

2.6.6 P-subsekventiaalisuus

Sekventiaalisen transduktorin syötemerkkijonon pituuteen nähden lineaarinen suoritusaika voidaan joskus saavuttaa ilman, että joudutaan luopumaan mahdollisuudesta palauttaa useita tulostemerkkijonoja. *P-subsekventiaalisessa* (engl. *p-subsequential*) [19, 2.2] transduktorissa sekventiaalisen transduktorin lopputiloihin liittyy enimmillään p kappaletta siirtymiä, joihin liittyvät merkkijonot lisätään tulostemerkkijonon loppuun. P-subsekventiaalinen transduktori voi palauttaa enintään p tulostetta annettua syötemerkkijonoa kohti, eli se saa olla korkeintaan tasaisesti monikäsitteinen. Kuvassa 2.5 on esimerkki 3-subsekventiaalisesta transduktorista.



Kuva 2.5: 3-subsekventiaalinen transduktori, joka tulostaa eräitä suomenkielisiä sanoja. Esimerkiksi syöte $''l''$ antaa tulokseksi merkkijonot $''lato''$, $''lasi''$ ja $''lama''$.

2.7 Transduktoriin liittyvät automaatit

Transduktori $T = (Q, A, B, q_0, E, F)$ voidaan myös tulkita äärelliseksi automaatiksi, jonka aakkosto on $A^* \times B^*$. [3] Tätä äärellistä automaattia kutsutaan transduktorin *perustana olevaksi automaatiksi* (engl. *underlying finite state automaton*) [21, s. 15].

Lisäksi transduktorin syöte- ja tulostepuolet voidaan käsitellä erillisinä automaatteina. Näin saadaan kaksi uutta automaattia, joiden tilasiirtymäverkko on suunnattuna verkkona isomorfinen transduktorin ja sen perustana olevan automaatin kanssa, mutta joissa siirtymiin liittyvät merkkijonot sisältävät pelkästään transduktorin syöte- tai tulostemerkkijonot. Kirjallisuudessa näille automaateille on annettu useita nimiä. Beesley ja Karttunen [4, s. 30] kutsuvat syötemerkkijonot sisältävää automaattia transduktorin *yläprojektioksi* (engl. *upper projection*) ja tulostemerkkijonot sisältävää automaattia transduktorin *alaprojektioksi* (engl. *lower projection*). Roche ja Schabes [21, s. 15] puolestaan käyttävät vastaavista automaateista nimityksiä *ensimmäinen* ja *toinen projektio*. Transduktorille $T = (Q, A, B, q_0, E, F)$ nämä projektiooperaatiot voidaan määritellä muodollisesti kirjoittamalla

$$\begin{aligned} p_1(T) &= (Q, A^*, E_1, q_0, F) \\ p_2(T) &= (Q, B^*, E_2, q_0, F), \end{aligned}$$

missä

$$\begin{aligned} E_1 &= \{(q, a, q') : (q, a, b, q') \in E\} \\ E_2 &= \{(q, b, q') : (q, a, b, q') \in E\}. \end{aligned}$$

[3] Koska äärellisen automaatin määritelmässä vaaditaan, että sen aakkoston on oltava äärellinen, tulisi aakkostona oikeammin käyttää jotain joukon $A^* \times B^*$ äärellistä osajoukkoa. Tällainen on kuitenkin aina mahdollista tarvittaessa muodostaa, kun huomioidaan, että transduktorin tilasiirtymien määrä on äärellinen. Vaihtoehtoisesti voidaan myös muuntaa transduktori ensin kirjaintransduktoriksi ja käyttää sitten automaatin aakkostona joukkoa $(A \cup \{\epsilon\}) \times (B \cup \{\epsilon\})$, joka on äärellinen.

2.8 Transduktoreille tehtäviä muunnoksia

2.8.1 Käänteistransduktorin muodostaminen

Transduktorin määritelmä on symmetrinen syöte- ja tulostekielten valinnan suhteen. Siispä syöte- ja tulostekielet voidaan vaihtaa keskenään [4, s. 14][21, s. 17]. Käytännössä tämä tarkoittaa sitä, että jos on olemassa transduktori $T = (Q, A, B, q_0, E, F)$, on sen käänteistransduktori $T' = (Q, B, A, q_0, E', F)$, jossa

$$E' = \{(q, b, a, q') \colon (q, a, b, q') \in E\}$$

Käänteistransduktoria muodostettaessa on huomioitava, että käänteistransduktorin ominaisuudet saattavat poiketa huomattavasti alkuperäisen transduktorin ominaisuuksista. Transduktori voi esimerkiksi olla sekventiaalinen, mutta jos sen tulostepuolella on ϵ-siirtymien muodostamia suljettuja lenkkejä, sen käänteistransduktori saattaakin olla äärettömästi monikäsitteinen. Äärettömästi monikäsitteisen transduktorin suoritus ei pääty kaikilla syötteillä, joten käytännön sovelluksissa käänteistransduktorien käytössä on syytä olla varovainen.

Tilasiirtymäkaavioissa voidaan käänteistransduktori muodostaa yksinkertaisesti vaihtamalla siirtymissä jakoviivalla erotetut merkkijonot keskenään.

2.8.2 Yhdistetyt transduktorit

Käytännön kannalta eräs transduktorien hyödyllisimmistä ominaisuuksista on se, että kaksi transduktoria $T : A^* \to B^*$ ja $U : B^* \to C^*$ voidaan liittää *yhdistetyksi transduktoriksi* $U \circ T : A^* \to C^*$ [5, s. 59] [17, s. 23]. Transduktorien yhdistäminen (engl. *composition*) tapahtuu samaan tapaan kuin yhdistettyjen funktioiden muodostaminen matematiikassa. Syötemerkkijono muunnetaan ensin transduktorissa T, ja transduktorin T tulostemerkkijonoja käytetään transduktorin U syötemerkkijonoina. Edellä esiteltyjä merkintöjä käyttäen yhdistetyn transduktorin $U \circ T$ määrittelemälle relaatiolle $R(U \circ T)$ pätee

$$(a, c) \in R(U \circ T) \Leftrightarrow \exists b \in B^* : (a, b) \in R(T), (b, c) \in R(U).$$

Aivan kuten funktioiden yhdistäminen, transduktorien yhdistämienkin yleistyy useamman kuin kahden transduktorien ketjuille. Yhdistämisoperaatio on assosiatiivinen, eli $(U \circ T) \circ S = U \circ (T \circ S) = U \circ T \circ S$ [4, s. 51].

Transduktorien yhdisteltävyys mahdollistaa sen, että monimutkaiset säännöllisten kielten väliset muunnokset voidaan toteuttaa pienissä osissa, ja lopullinen

muunnos saadaan yhdistämällä osat toisiinsa. Transduktoreita hyödyntävien sovellusten kehityksessä tätä voi verrata komponenttipohjaiseen ohjelmointiin. Yksinkertaisista transduktoreista on helpompi jäljittää virheitä kuin yhdestä monimutkaisesta transduktorista, ja niitä voi myös käyttää uudelleen eri sovelluksissa.

2.8.3 Transduktorien yhdiste ja katenaatio

Yhdistettyjen transduktorin $U \circ T$ lisäksi voidaan transduktoreille määritellä säännöllisten lausekkeiden tapaan transduktorien U ja T *yhdiste* (engl. *union*) $U \cup T$ ja *katenaatio* (engl. *concatenation*) UT [4, s. 49].

Transduktorien yhdisteelle pätee

$$(a, b) \in R(U \cup T) \Leftrightarrow (a, b) \in R(U) \vee (a, b) \in R(T).$$

Transduktorien yhdiste muodostetaan liittämällä niiden alkutilat yhteen tai muodostamalla uusi alkutila, joka liitetään kunkin yhdisteen osan alkutilaan.

Transduktorien katenaatio puolestaan muodostetaan yhdistämällä toisen transduktorin lopputilat toisen transduktorin alkutiloihin. Katenaatiossa

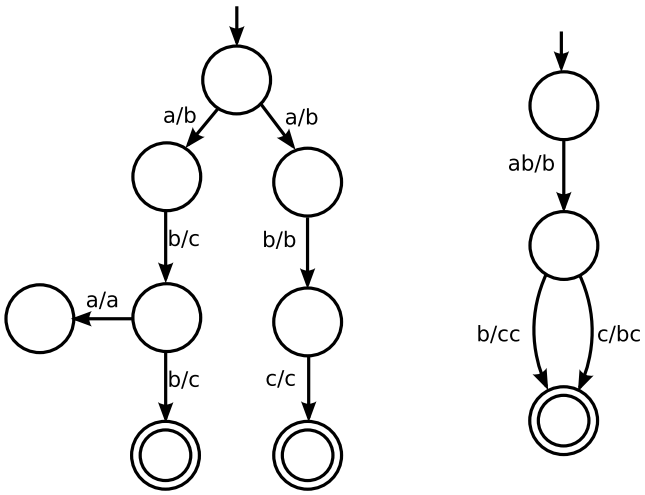
$$(aa', bb') \in R(UT) \Leftrightarrow (a, b) \in R(U) \wedge (a', b') \in R(T).$$

2.8.4 Transduktorien leikkaus

Kahdesta saman aakkoston äärellisestä automaatista A ja B voidaan aina muodostaa äärellinen automaatti $A \cap B$, jonka tunnistama kieli on automaattien A ja B tunnistamien kielten leikkaus: $L(A \cap B) = L(A) \cap L(B)$ [21, s. 6]. Tämä hyödyllinen tulos ei kuitenkaan yleisesti päde transduktoreille. Käyttämällä hyväksi transduktorien ja automaattien välistä yhteyttä voidaan leikkausoperaatio tästä huolimatta toteuttaa kaikille ϵ-vapaille kirjaintransduktoreille leikkaamalla niiden perustana olevat automaatit [21, s. 19].

2.9 Transduktorien optimointi

Sama säännöllisten kielten välinen relaatio on mahdollista toteuttaa usean eri transduktorin avulla. Suorituskyvyn ja muistin käytön kannalta ei kuitenkaan ole lainkaan samantekevää, millaista transduktoria missäkin tilanteessa käytetään. Kuvassa 2.6 on esitetty kaksi transduktoria, jotka molemmat toteuttavat saman relaation. Toinen on kuitenkin toista selkeästi tehokkaampi sekä muistin käytön että suorituskyvyn kannalta.



Kuva 2.6: Kaksi erilaista transduktoria, jotka kuvaavat saman relaation kahden säännöllisen kielen välillä. Molemmat transduktorit ovat yksikäsitteisiä, mutta ainoastaan oikeanpuoleinen transduktori on sekventiaalinen. Vasemmanpuoleisessa transduktorissa on kahdeksan tilaa ja seitsemän tilasiirtymää, kun oikeanpuoleisessa sekä tiloja että siirtymiä on molempia vain kolme. Merkkijonon $''abc''$ analyysi vasemmanpuoleisessa transduktorissa vaatii viiden tilansiirtymän läpikäynnin, oikeanpuoleisessa puolestaan kahden.

Automaatin tai transduktorin muuntamista sellaiseksi ekvivalentiksi automaatiksi tai transduktoriksi, jossa on mahdollisimman vähän tiloja, kutsutaan automaatin tai transduktorin minimoinniksi. Jotta minimointi olisi mielekästä, voidaan vaatia, että minimoitaessa säilytetään jokin transduktorin oleellinen ominaisuus, esimerkiksi se, että siirtymien syötemerkkijonot ovat korkeintaan yhden merkin mittaisia. Jos näin ei tehdä, luvussa 2.4 esitetyn määritelmän mukainen transduktori voidaan vaikkapa kahden äärellisen kielen välillä aina minimoida transduktoriksi, jossa on korkeintaan kaksi tilaa. Tämä on mahdollista, koska hyväksyttäviä merkkijonoja on äärellinen määrä, ja kuhunkin niistä liittyy äärellinen määrä tulostemerkkijonoja. Kahteen tilaan minimoitu transduktori voidaan siten muodostaa asettamalla toiseksi tilaksi transduktorin alkutila ja tekemällä toisesta tilasta lopputila. Jokaista transduktorin syöte-tulostemerkkijonoparia varten tehdään alkutilalta lopputilalle erillinen siirtymä. Tällainen menettely ei kuitenkaan ole tavallisesti järkevää, sillä tarvittavien tilasiirtymien määrä voi olla valtava poistuvien tilojen lukumäärään verrattuna.

Kuten luvussa 2.3 todettiin, äärelliset automaatit voidaan yleensä determinisoida ilman tilojen määrän merkittävää kasvua, ja deterministisen automaatin minimointiin tilojen suhteen on olemassa erittäin tehokkaita ja yleispäteviä algoritmeja.

Transduktoreilla tilanne ei ole aivan näin hyvä. Ensimmäisenä ongelmana tulee vastaan se, että transduktoreja ei yleisessä tapauksessa ole lainkaan mahdollista determinisoida syöte- tai tulostemerkkien suhteen. Tämä johtuu siitä, että kaikki transduktorit eivät ole yksikäsitteisiä. Kuitenkin myös transduktoreille voidaan soveltaa automaattien determinisointi- ja minimointialgoritmeja muuntamalla aakkostosta A aakkostoon B operoiva transduktori ensin kirjaintransduktoriksi, ja minimoimalla tämän perustana oleva aakkoston $(A \cup \{\epsilon\}) \times (B \cup \{\epsilon\})$ äärellinen automaatti.

Tämä menetelmä soveltuu kaikille transduktoreille, ja voi tuottaa ekvivalentin transduktorin, jossa on alkuperäistä vähemmän tiloja. Tämä transduktori ei kuitenkaan välttämättä ole tilamäärältään pienin mahdollinen tai muutoinkaan erityisen optimaalinen.

2.9.1 Sekventiaalisten transduktorien minimointi

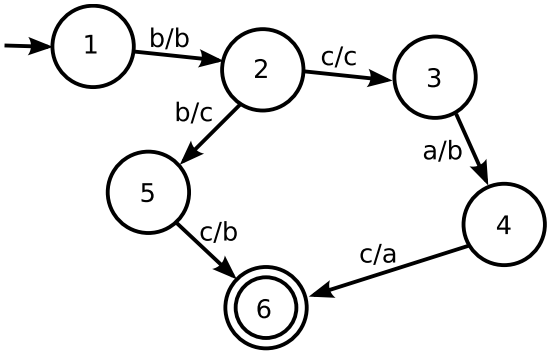
Tutkitaan esimerkkinä sekventiaalisen transduktorin minimoinnista kuvassa 2.7 olevaa transduktoria. Jos transduktori tulkitaan äärelliseksi automaatiksi, jonka aakkostona on merkkijonojen joukko $A = \{b/b, c/c, a/b, c/a, b/c, c/b\}$, sille on olemassa tilojen nimiä lukuun ottamatta yksikäsitteinen minimoitu esitysmuoto. Todistus tälle tulokselle sekä minimiautomaatin muodostamiseen sopiva algoritmi on esitelty Hopcroftin ja Ullmanin kirjassa [12, s. 67].

Äärellisten automaattien minimointialgoritmin ideana on jakaa automaatin tiloja $q \in Q$ luokkiin $X \in \mathcal{P}(Q)$ niin kauan, kunnes päädytään tilanteeseen, jossa siirtymät luokkien välillä tapahtuvat tilasta riippumatta samalla tavalla. Tarkemmin sanoen lopullisessa osituksessa X pätee, että annetulla tilalla $q \in X$ ja syötemerkillä $a \in A$ siirtymän kohdetila $\delta(q, a) \in X'$ jos ja vain jos $\delta(q', a) \in X'$ kaikilla $q' \in X$.

Merkitsemällä tilaluokkia roomalaisilla numeroilla ja taulukoimalla mahdolliset siirtymien lopputilat ja vastaavat tilaluokat päädytään ensimmäisen vaiheen jälkeen seuraavaan tilojen ositukseen:

		b/b	c/c	a/b	c/a	b/c	c/b
I	1	2 I					
	2		3 I			5 I	
	3			4 I			
	4				6 II		
	5						6 II
II	6						

Koska seuraavassa vaiheessa jokainen automaatin tila muodostaisi oman ekvi-



Kuva 2.7: Sekventiaalinen transduktori, joka on automaattina minimaalinen.

valenssiluokkansa, havaitaan, että kuvan 2.7 transduktori on automaattina jo valmiiksi minimaalinen.

Automaattina minimaalisia transduktoreja on kuitenkin usein mahdollista minimoida edelleen muuttamalla syöte- ja tulostemerkkien sijoittelua siirtymien välillä. Mehryar Mohrin vuonna 1994 julkaisema [18] algoritmi sekventiaalisten transduktorien minimointiin on esimerkki tällaisesta tehokkaasta transduktorien minimointialgoritmista, jota ei kuitenkaan voi soveltaa ei-sekventiaalisille transduktoreille.

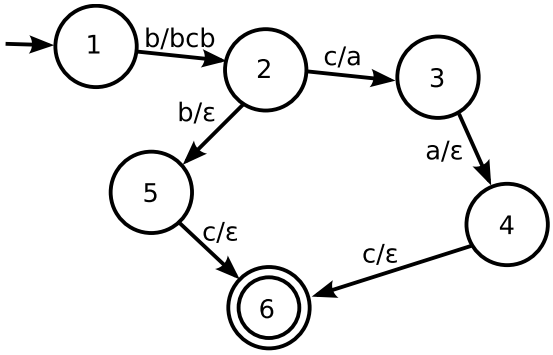
Mohrin minimointialgoritmin ideana on muokata transduktoria kahdessa vaiheessa siten, että ensin siirtymien tulostemerkkijonoja siirretään mahdollisimman lähelle transduktorin alkutilaa. Muita muutoksia transduktoriin ei tässä vaiheessa vielä tehdä. Toisessa vaiheessa transduktorin perustana olevaan automaattiin sovelletaan äärellisten automaattien minimointialgoritmia.

Sovelletaan kuvan 2.7 transduktoriin nyt Mohrin minimointialgoritmin ensimmäistä vaihetta. Nähdään, että tilalta 2 lähtee kaksi sallittua polkua, joihin liittyvät tulostemerkkijonot ovat $''cba''$ ja $''cb''$. Näiden yhteinen etuliite $''cb''$ voidaan siirtää tilalle 2 tulevaan siirtymään. Vastaavasti tilalta 3 lähtee ainoastaan yksi sallittu polku. Tähän liittyvä tulostemerkkijono $''ba''$ voidaan siirtää tilalta 2 tilalle 3 tulevaan siirtymään. Kun nämä muunnokset on tehty, päädytään kuvan 2.8 transduktoriin, joka on täysin ekvivalentti alkuperäisen transduktorin kanssa.

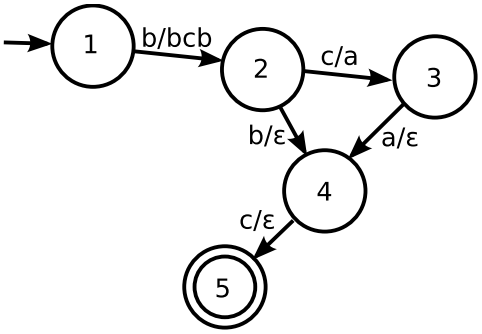
Toisin kuin alkuperäistä, kuvan 2.8 muunnettua transduktoria voidaan kuitenkin minimoida myös automaattina. Tämä minimointi tuottaa lopulta seuraavan tilojen osituksen:

		b/bcb	c/c	a/ϵ	c/ϵ	b/ϵ
I	1	2 II				
II	2		3 III			5 IV
III	3			4 IV		
IV	4				6 V	
	5				6 V	
V	6					

Muodostamalla ekvivalenssiluokista I - V uudet transduktorin tilat, saadaan lopputuloksena kuvan 2.9 mukainen transduktori, jossa on yksi tila vähemmän kuin alkuperäisessä transduktorissa.



Kuva 2.8: Kuvan 2.7 transduktori, jossa tulostemerkkejä on siirretty Mohrin minimointialgoritmin ensimmäisen vaiheen mukaisesti.



Kuva 2.9: Transduktori, joka on saatu soveltamalla äärellisten automaattien minimointialgoritmia kuvan 2.8 transduktoriin.

3 Rekisteritransduktorit

3.1 Tavanomaisten transduktorien rajoituksia

Aikaisemmissa luvuissa on todettu, että transduktorit tekevät muunnoksia säännöllisten kielten välillä. Niinpä niillä ei voi lainkaan tunnistaa tai muuntaa ei-säännöllisiä kieliä. Esimerkiksi aritmeettiset lausekkeet koostuvat kokonaislukujen yhteen-, vähennys-, kerto- ja jakolaskusta sulkumerkkien avulla ryhmiteltynä. Aritmeettisten lausekkeiden muodostama kieli ei kuitenkaan ole säännöllinen, joten aritmeettisia lausekkeita ei voi yleisessä tapauksessa käsitellä transduktorien avulla lainkaan.

Tämä voidaan todistaa käyttämällä apuna ns. pumppauslemmaa, joka on matemaattinen tapa ilmaista se, että säännöllinen kieli ei voi sisältää rajoittamattomasti toistuvia sisäkkäisiä rakenteita. Pumppauslemman mukaan säännölliselle kielelle A pätee, että on olemassa kokonaisluku $n \geq 1$, jolla kaikilla merkkijonoilla $x \in A$, $|x| \geq n$, on olemassa ositus $x = uvw$ siten, että $|uv| \leq n$, $|v| \geq 1$ ja $uv^iw \in A$ kaikilla kokonaisluvuilla $i \geq 0$ [11, s. 45].

Oletetaan, että aritmeettiset lausekkeet muodostavat säännöllisen kielen. Kiinnitetään kokonaisluku n ja valitaan merkkijono $x = ({}^n7)^n$, joka on aritmeettinen lauseke. Nyt $|x| = n + 1 + n = 2n + 1 > n$. Osituksessa $x = uvw$ on silloin $u = ({}^j$, $v = ({}^k$ ja $w = ({}^{n-j-k}7)^n$ kokonaisluvuilla j ja k, joille pätee $j + k \leq n$ ja $k \geq 1$. Pumppauslemman mukaan (valitsemalla $i = 0$) myös uw on aritmeettinen lauseke. Kuitenkin $uw = ({}^j({}^{n-j-k}7)^n = ({}^{n-k}7)^n$, ja koska $k \geq 1$, lausekkeessa on vähemmän aloittavia kuin lopettavia sulkumerkkejä, eikä se ole kelvollinen aritmeettinen lauseke. Koska pumppauslemman olettaminen päteväksi aritmeettisille lausekkeille johti ristiriitaan, aritmeettiset lauseet eivät muodosta säännöllistä kieltä.

Myös säännöllisten kielten joukosta löytyy tapauksia, joiden käsittely transduktorien avulla on käytännössä hankalaa, vaikkakin mahdollista. Esimerkiksi yksinkertainen suomi-englanti-sanakirja voidaan koodata äärelliseksi transduktoriksi T, joka kuvaa relaatiota suomenkielisten sanojen ja niiden englanninkielisten vastineiden välillä. Tämä on mahdollista, sillä sanakirja sisältää äärellisen määrän hakusanoja ja niiden käännöksiä, ja äärelliset kielet ovat aina säännöllisiä.

Joissakin tilanteissa voi kuitenkin olla tarpeen laajentaa sanakirjaa vaikkapa siten, että sanojen ympärillä sallitaan myös sulkumerkkien käyttö. Kuten edellä todettiin, transduktorin avulla ei ole mahdollista yleisessä tapauksessa tarkistaa, on-

Kuva 3.1: Sanoja suomesta englantiin kääntävä transduktori T voidaan muuttaa muotoon, joka hyväksyy sanat myös sulkumerkkien sisällä. Tämä muutos kuitenkin kasvattaa transduktorin koon kaksinkertaiseksi.

ko mielivaltaisessa sulkumerkkejä sisältävässä merkkijonossa jokaisella aloittavalla sululla vastaava lopettava sulku. Mutta jos sisäkkäisiä sulkuja ei tarvita, kielen tunnistaminen transduktorin avulla on mahdollista. Tällä tavoin laajennettu kieli sisältää edelleen äärellisen määrän sanoja, joten senkin tunnistaminen voidaan toteuttaa äärellisen transduktorin avulla. Laajennettu transduktori T' löytäisi sanalle *kissa* englanninkielisiä vastineita seuraavaan tapaan:

$$kissa \rightarrow \{cat\} \tag{3.1}$$

$$(kissa) \rightarrow \{(cat)\} \tag{3.2}$$

$$(kissa \rightarrow \emptyset \tag{3.3}$$

Tapauksessa 3.1 sana ei ole suluissa, jolloin transduktori T' toimii samoin kuin alkuperäinen transduktori T. Tapauksessa 3.2 sana on suluissa. Tällöin transduktori T' palauttaa vastaavan englanninkielisen sanan suluissa, mutta transduktori T palauttaisi tyhjän joukon. Tapauksessa 3.3 sanan alussa on sulku, mutta vastaava sulku sanan lopusta puuttuu. Tätä muotoa olevat merkkijonot eivät kuulu kummankaan transduktorin lähtökieleen, joten ne palauttavat tulokseksi tyhjän joukon.

Koska transduktorin tilasiirtymä ei voi riippua jo käsitellystä merkkijonosta, transduktoriin T' on sisällytettävä sanat suomesta englantiin muuntava osatransduktori kahteen kertaan kuvan 3.1 mukaisesti. Tämä johtaa siihen, että transduktorin T' toteuttamiseen vaadittava tilasiirtymäverkko on noin kaksi kertaa niin suuri kuin transduktorin T tilasiirtymäverkko. Tämä on suuri ongelma käytännön toteutusten kannalta, sillä tilasiirtymäverkon koko vaikuttaa suoraan sovelluksen tarvitseman muistin määrään. Samasta ongelmasta on kyse myös silloin, kun säännöllisiä kielioppeja rakennetaan yhdistelemällä yksinkertaisempia kielioppimoduuleja, ja joitakin moduuleista joudutaan toistamaan useassa paikassa kieliopin sisällä [21, 11.3].

3.2 Rekisteritransduktori ja -automaatti

Yael Cohen-Sygalin ja Shuly Wintner [8] esittävät ratkaisuksi edellä mainittuihin ongelmiin muunnelmaa perinteisistä automaateista ja transduktoreista, jossa otetaan käyttöön rajoitettu määrä työmuistia rekistereiden muodossa. Cohen-Sygalin ja Wintnerin *rekisteritransduktori* (engl. *finite state registered transducer*) on kahdeksikko

$$T = (Q, A, B, q_0, \Gamma, n, E, F),$$

jossa Q on äärellinen joukko tiloja, A on transduktorin syöte- ja B tulosteaakkosto, q_0 alkutila, Γ *rekisteriaakkosto* (engl. *register alphabet*), n rekistereiden lukumäärä, E tilasiirtymien joukko ja $F \subseteq Q$ lopputilojen joukko. Rekisteriaakkosto Γ on äärellinen aakkosto, joka sisältää vähintään symbolin $\#$.

Tilasiirtymät $e \in E$ voidaan esittää muodossa

$$e = (q, a, b, o, i, \gamma, q'),$$

jossa q on siirtymän lähtötila ja q' siirtymän kohdetila. a on siirtymässä luettava merkistön A merkki (tai tyhjä merkkijono ϵ) ja b tulosteeseen tuleva merkistön B merkki (tai ϵ). Rekisterioperaatio o voi olla joko R tai W, i on kokonaisluku väliltä $[0, n]$, ja $\gamma \in \Gamma$.

Rekisteritransduktorin tilanne määritellään vastaavasti kuin tavanomaisen transduktorin tilanne, mutta siihen on lisätty rekistereiden arvoja kuvaava vektori. Siispä rekisteritransduktorin tilanne c on nelikko

$$c = (q, w, v, u) \in Q \times A^* \times B^* \times \Gamma^n,$$

jossa q on rekisteritransduktorin tila, w syötemerkkijonon jäljellä oleva osa, v tulostemerkkijonon alkuosa ja u rekistereiden sisältö. Rekisteritransduktorin alkutilanne syötteellä w on $c_0 = (q_0, w, \epsilon, \#^n)$.

Rekistereiden lisäksi tämä rekisteritransduktorin määritelmä poikkeaa luvussa 2.4 esitetystä transduktorin määritelmästä siten, että siinä transduktorina on itse asiassa kirjaintransduktori. Luvussa 2.6 kuitenkin osoitettiin, että kirjaintransduktorien ilmaisuvoima ei poikkea lainkaan tavallisista transduktoreista.

Rekisteritransduktorin tilasiirtymiä $e = (q, a, b, o, i, \gamma, q')$ on kolmea erilaista tyyppiä [8, 3.1]:

1. $(q, a, b, R, 0, \#, q')$: Vastaa tavanomaisen transduktorin siirtymää (q, a, b, q'). Kaikkien rekistereiden arvot pysyvät tässä siirtymässä ennallaan. Rekisteri-indeksi 0 ei viittaa mihinkään transduktorin todellisista rekistereistä, vaan se on ainoastaan muodollinen rekisteri, jonka arvon sovitaan olevan aina $\#$.

2. $(q, a, b, R, i, \gamma, q')$, jossa $i > 0$: Vastaa tavanomaisen transduktorin siirtymää (q, a, b, q'), mutta on sallittu vain, jos rekisterin i arvo tilalla a on γ. Operaatiota R kutsutaan *lukuoperaatioksi*.

3. $(q, a, b, W, i, \gamma, q')$, jossa $i > 0$: Vastaa tavanomaisen transduktorin siirtymää (q, a, b, q'), mutta siirryttäessä tilalle q' asetetaan rekisterin i arvoksi γ. Operaatiota W kutsutaan *kirjoitusoperaatioksi*.

Cohen-Sygal ja Wintner eivät esitä näille tilasiirtymille erityisiä nimiä. Tämä saattaa johtua siitä, että he esittävät rekisteritransduktorille vielä edellistäkin yleistetymmän määritelmän, jossa yhteen siirtymään voi liittyä yhden rekisterioperaation sijaan sarja rekisterioperaatioita. Tällä yleistyksellä pyritään vähentämään transduktorissa tarvittavien tilasiirtymien määrää [8, s. 58]. Yleistyksen mukana menetetään kuitenkin mahdollisuus luokitella siirtymät yksinkertaisesti kolmeen eri tyyppiin.

Tässä tutkielmassa pitäydytään edellä esitetyssä yksinkertaisemmassa määritelmässä, jossa siirtymissä voi olla korkeintaan yksi rekisterioperaatio. Tällöin siirtymätyyppejä voidaan kutsua (esitysjärjestyksessä) nimillä *perussiirtymä*, *lukusiirtymä* ja *kirjoitussiirtymä*. Seuraavassa luvussa esiteltävissä tilasiirtymäkaavioissa rekisterioperaatioita kuitenkin yhdistetään tilan säästämiseksi.

Rekisteriautomaatti (engl. *finite state registered automaton*) voidaan määritellä vastaavalla tavalla kuin rekisteritransduktori, mutta ilman tulosteaakkostoa ja tulostemerkkijonoja. Siispä rekisteriautomaatti on

$$M = (Q, A, q_0, \Gamma, n, E, F),$$

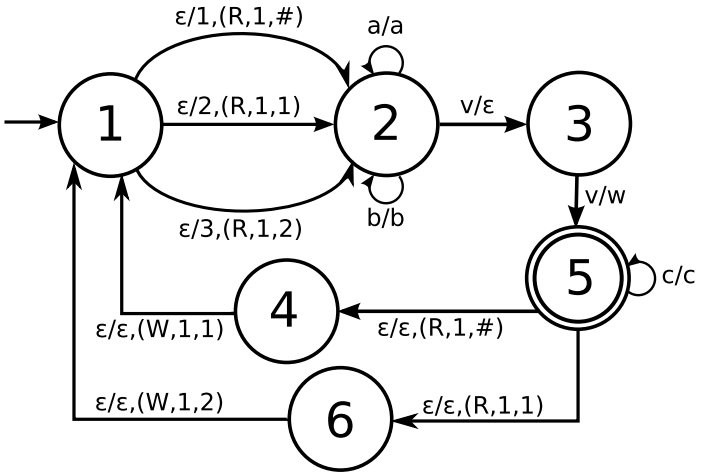
jossa Q on äärellinen joukko tiloja, A on transduktorin syöteaakkosto, q_0 alkutila, Γ rekisteriaakkosto, n rekistereiden lukumäärä, E tilasiirtymien joukko ja $F \subseteq Q$ lopputilojen joukko. Tilasiirtymä $e \in E$ on muotoa

$$e = (q, a, o, i, \gamma, q'),$$

jossa q on siirtymän lähtötila, q' siirtymän kohdetila, $a \in A \cup \{\epsilon\}$, o on rekisterioperaatio, i rekisterin nimi ja γ rekisteriaakkoston symboli.

3.3 Rekisteritransduktorien tilasiirtymäkaaviot

Jotta rekisteritransduktoreja voitaisiin havainnollistaa tilasiirtymäkaavioiden avulla, on rekisterioperaatiot esitettävä tilasiirtymien yhteydessä. Otetaan käyttöön mer-



Kuva 3.2: Esimerkki rekisteritransduktorista. Rekisteritransduktorissa on yksi rekisteri, jonka rekisteriaakkosto on $\{\#, 1, 2\}$. Tämä transduktori tunnistaa kielen $L((a|b)^*vvc^*(\epsilon|(a|b)^*vvc^*)(\epsilon|(a|b)^*vvc^*))$. Transduktorin tulostemerkkijonoissa kunkin muotoa $(a|b)^*vvc^*$ olevan komponentin eteen lisätään järjestysnumero 1, 2 tai 3. Lisäksi merkkijonot vv korvautuvat merkillä w.

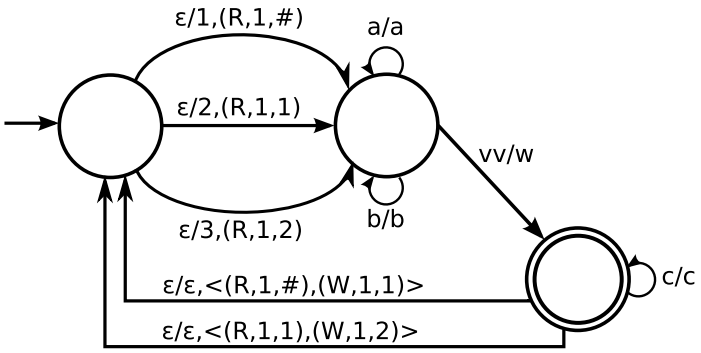
kintä, jossa perussiirtymiä merkitään samalla tavalla kuin siirtymiä tavanomaisissa transduktoreissa. Siispä rekisteritransduktorin siirtymä $(q, a, b, R, 0, \#, q')$ merkitään tilasiirtymäkaavioon piirtämällä tilalta q tilalle q' johtava nuoli, jonka viereen kirjoitetaan merkintä a/b. Lukusiirtymissä $(q, a, b, R, i, \gamma, q')$ käytetään merkintää $a/b, (R, i, \gamma)$ ja kirjoitussiirtymissä $(q, a, b, W, i, \gamma, q')$ merkintää $a/b, (W, i, \gamma)$. Esimerkki tätä merkintätapaa käyttävästä tilasiirtymäkaaviosta on kuvassa 3.2.

Tällä tavoin esitettynä tilasiirtymäkaaviot kuvaavat rekisteritransduktoreja täsmällisesti luvussa 3.2 annetun määritelmän mukaisesti. Tällainen graafinen esitys vie kuitenkin joissakin tapauksissa paljon tilaa, eikä ole kovin helposti luettavissa. Graafisen esityksen yksinkertaistamiseksi otetaan käyttöön kaksi lyhennysmerkintää, joiden avulla tilasiirtymäkaavioista voidaan poistaa kaikki ei-lopputilat, joihin liittyy ainoastaan yksi tuleva ja lähtevä siirtymä.

1. Peräkkäiset syöte- ja tulostemerkkijonot yhdistetään kuvan 2.3 mukaisesti.

2. Peräkkäiset rekisterioperaatiot yhdistetään listaksi rekisterioperaatioita. Rekisterioperaatiot luetaan vasemmalta oikealle. Siispä jos yhdistetään kaksi siirtymää, joista ensimmäiseen liittyy rekisterioperaatio (R, i, γ) ja toiseen (W, j, γ), yhdistetyn siirtymän rekisterioperaatioiksi merkitään $< (R, i, \gamma), (W, j, \gamma) >$.

Kuvassa 3.3 on esitetty kuvan 3.2 rekisteritransduktori esitettynä näitä merkintöjä hyväksi käyttäen. Tässä tapauksessa transduktorin esittämiseen tarvittavien ti-



Kuva 3.3: Kuvan 3.2 rekisteritransduktori lyhennetyn tilasiirtymäkaavioiden merkintätavan avulla esitettynä. Esitykseen tarvitaan kuuden tilan sijasta ainoastaan kolme tilaa.

lojen määrä on vähentynyt kuudesta kolmeen.

Lyhennetty merkintätapa ei kuitenkaan ole aivan ongelmaton. Vaikka sen avulla esitetty transduktori toteuttaakin aina täsmälleen saman relaation kuin alkuperäinen lyhentämätön esitys, lyhennetyn esityksen mukaista transduktoria ei ole mahdollista yksikäsitteisesti palauttaa takaisin alkuperäiseen muotoon. Esimerkin tapauksessa lyhennetyn muodon siirtymä vv/w voidaan palauttaa lyhentämättömään muotoon joko siirtymäpariksi $v/\epsilon, v/w$ tai $v/w, v/\epsilon$.

3.4 Rekisteritransduktorien optimointi

Kuten muitakin automaatteja ja transduktoreja, myös rekisteriautomaatteja ja -transduktoreja on usein mahdollista muokata siten, että niiden tunnistama säännöllinen kieli tai toteuttama relaatio ei muutu, mutta automaatin tai transduktorin muut ominaisuudet paranevat. Cohen-Sygal ja Wintner ovat tutkineet näitä muunnoksia rekisteriautomaattien osalta, ja pohtivat artikkelissaan algoritmeja rekisteritransduktorien ϵ-siirtymien poistolle, ylimääräisten rekisterioperaatioiden poistolle ja automaatin minimoinnille rekisterien, tilasiirtymien ja tilojen määrän suhteen [8, 3.3-3.4].

Cohen-Sygal ja Wintner määrittelevät rekisteriautomaatin ϵ-siirtymäksi siirtymän, jossa syötemerkkijonosta ei lueta merkkiä, mutta johon kuitenkin voi liittyä rekisterioperaatioita. He osoittavat, että jokaiselle ϵ-siirtymiä sisältävälle rekisteriautomaatille voidaan aina konstruoida ekvivalentti rekisteriautomaatti, joka ei sisällä ϵ-siirtymiä. Esitetty menetelmä kuitenkin olettaa, että yhteen siirtymään voidaan liittää useampi kuin yksi rekisterioperaatio. Sitä ei siksi voida yleistää tässä tutkielmassa määritellyille rekisteritransduktoreille, joissa siirtymään voi liittyä vain yksi

rekisterioperaatio.

Tietyissä tilanteissa rekisteritransduktoreista (ja -automaateista) on mahdollista suoraan poistaa ylimääräisiä rekisterioperaatioita. Tarkastellaan transduktorin tilasiirtymäverkon osaa, jossa tilat ja niiden väliset siirtymät muodostavat jonon, jonka muihin kuin ensimmäiseen ja viimeiseen tilaan liittyy ainoastaan yksi tuleva ja yksi lähtevä siirtymä, eivätkä kyseiset tilat ole lopputiloja. Kerätään siirtymäjonosta kuhunkin rekisteriin kohdistuvat rekisterioperaatiot omaksi listakseen, ja tutkitaan kukin näistä listoista erikseen. Cohen-Sygalin ja Wintnerin esittämien sääntöjen mukaan [8, s. 65] tilasiirtymäjonon rekisterioperaatiot saadaan optimaalisiksi seuraavasti:

1. Jos kirjoitusoperaatiota (W, n, a) seuraa välittömästä lukuoperaatio (R, n, b) joillakin $a \neq b$, rekisterioperaatioiden jono ei ole *toteutuva* (engl. *satisfiable*). Tällöin ei ole mahdollista, että suoritus voisi koskaan edetä koko tutkittavana olevan tilasiirtymäjonon läpi, jolloin sekä siirtymät että jonon sisällä olevat tilat voidaan poistaa transduktorista.

2. Jos lukuoperaatiota (R, n, a) seuraa välittömästä lukuoperaatio (R, n, b) joillakin $a \neq b$, rekisterioperaatioiden jono ei ole toteutuva, ja siirtymät sekä jonon sisällä olevat tilat voidaan poistaa transduktorista.

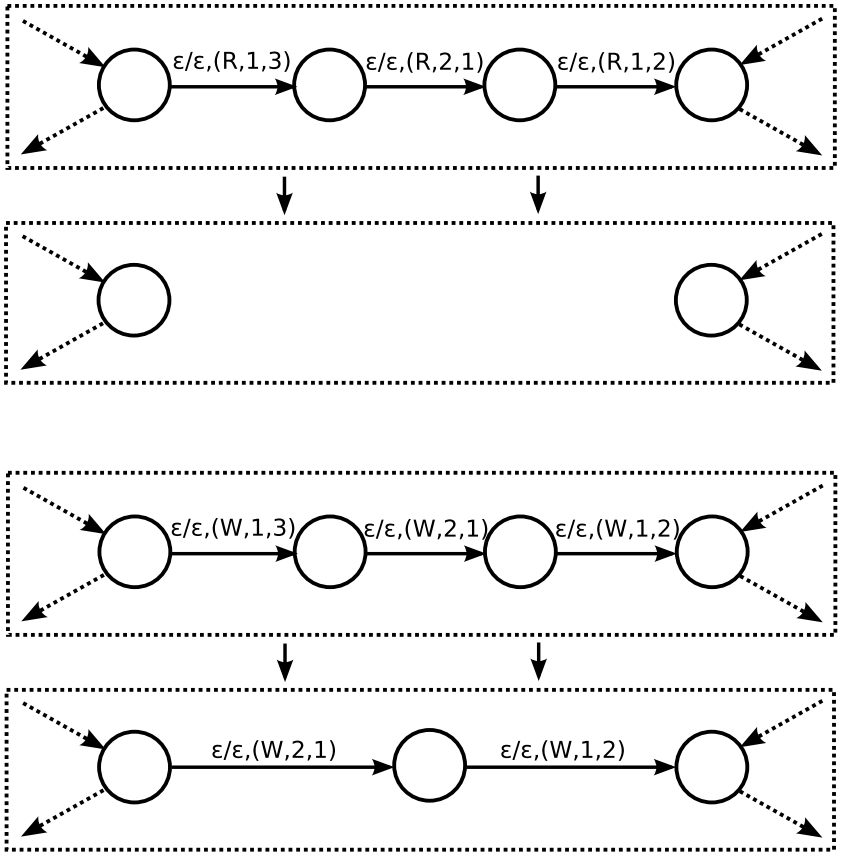
3. Jos kaikki rekisterioperaatiot ovat kirjoitusoperaatioita, niistä kaikki paitsi viimeinen voidaan poistaa transduktorista.

4. Jos kaikki rekisterioperaatiot ovat lukuoperaatioita, yhtä lukuun ottamatta kaikki muut voidaan poistaa transduktorista.

5. Jos operaatiojono sisältää sekä luku että kirjoitusoperaatioita, jono on kohtien 1 ja 2 mukaan toteutuva ja jonon ensimmäinen operaatio on kirjoitusoperaatio, kaikki paitsi viimeinen kirjoitusoperaatio voidaan poistaa transduktorista.

6. Jos operaatiojono sisältää sekä luku että kirjoitusoperaatioita, jono on kohtien 1 ja 2 mukaan toteutuva ja jonon ensimmäinen operaatio on lukuoperaatio, kaikki paitsi jonon ensimmäinen operaatio sekä jonon viimeinen kirjoitusoperaatio voidaan poistaa transduktorista. Myös viimeisen kirjoitusoperaation voi poistaa, jos siinä kirjoitetaan rekisteriin sama arvo joka luetaan jonon ensimmäisessä operaatiossa.

Esimerkkejä optimoinnin vaikutuksista on kuvassa 3.4. Rekisterioperaatioiden optimoinnin jälkeen transduktorista voi poistaa sellaiset tilat ja tilasiirtymät, joita



Kuva 3.4: Kaksi esimerkkiä transduktorin rekisterioperaatioiden optimoinnista. Ylemmässä esimerkissä rekisterioperaatioiden jono ei ole toteutuva, joten siirtymät ja niiden välissä olevat tilat voidaan poistaa kokonaan. Alemmassa esimerkissä rekisteriin 1 kohdistuu kaksi peräkkäistä kirjoitusoperaatiota, joista ensimmäisen voi poistaa.

ei voi saavuttaa millään syötemerkkijonolla tai joista ei voi päätyä transduktorin lopputilaan.

Rekisteritransduktorin minimoiminen rekisterien määrän suhteen onnistuu muuntamalla rekisteritransduktori tavanomaiseksi äärelliseksi transduktoriksi, jossa rekistereitä ei ole yhtään. Choen-Sygal ja Wintner toteavat tulokset rekisteriautomaateille, mutta eivät esitä tarkkaa todistusta [8, s. 58]. Todistetaan tämä esittämällä yleispätevä tapa suorittaa tämä muunnos rekisteritransduktoreille; sama menetelmä toimii myös rekisteriautomaateilla.

Olkoon rekisteritransduktorissa T tiloja n kappaletta, rekistereitä m kappaletta ja T:n rekisteriaakkostossa p symbolia. Nimetään T:n tilat kokonaisluvuilla $1..n$, rekisterit $1..m$ ja rekisteriaakkoston symbolit $1..p$.

Rakennetaan nyt uusi äärellinen transduktori T', jossa on np^m+1 tilaa. Yksi tiloista

ta on transduktorin alkutila, jolle annetaan nimi S. Loput T':n tilat nimetään $m+1$ -komponenttisilla vektoreilla, joissa ensimmäinen komponentti saa kokonaislukuarvot $1..n$ ja loput m komponenttia muista komponenteista riippumatta kokonaislukuarvot $1..p$. Merkitään transduktorin T' tilan a nimen komponenttia i merkinnällä $a[i-1]$. Siispä ensimmäinen komponentti on $a[0]$ ja viimeinen $a[m]$. Transduktorin T' tila a on lopputila jos ja vain jos transduktorin T tila $a[0]$ on lopputila.

Määritellään T':n tilasiirtymät siten, että tilalta $a \neq S$ johtaa siirtymä tilalle $a' \neq S$ jos ja vain jos seuraavat kolme ehtoa pätevät:

- Transduktorissa T on siirtymä e tilalta $a[0]$ tilalle $a'[0]$.
- Siirtymä e on sallittu rekisterin i arvon ollessa $a[i]$ kaikilla $1 \leq i \leq m$.
- Jos rekisterin i arvo on $a[i]$ kaikilla $1 \leq i \leq m$, niin siirtymän e jälkeen rekisterin i arvo on $a'[i]$ kaikilla $1 \leq i \leq m$.

Jokaista rekisteritransduktorista T löytyvää nämä ehdot täyttäviää siirtymää e kohti tulee transduktoriin T' siirtymä e' tilalta a tilalle a', jossa e':n syötemerkki on e:n syötemerkki ja e':n tulostemerkki on e:n tulostemerkki. Lisäksi transduktorin T' tilalta S asetetaan lähtemään ϵ-siirtymä tilalle a jos ja vain jos $a[0]$ on transduktorin T alkutila.

Tällä määrittelyllä transduktori T' toteuttaa saman relaation kuin rekisteritransduktori T. Uuden transduktorin tilojen määrä $np^m + 1$ on kuitenkin eksponentiaalisesti verrannollinen alkuperäisen transduktorin rekistereiden lukumäärään m.

Äärelliset automaatit on mahdollista muuntaa rekisteriautomaateiksi, jossa on kaksi rekisteriä ja kolme tilaa, ja siirtymiin voi liittyä kaksi rekisterioperaatiota [8, s. 61]. Tulos pätee myös rekisteriautomaateille, sillä ne voi ensin muuntaa tavanomaisiksi äärellisiksi automaateiksi edellä esitetyllä menettelyllä. Tästä tuloksesta ei kuitenkaan ole apua tilojen määrän suhteen minimoinnissa, jos vaaditaan, että siirtymään voi liittyä vain yksi rekisterioperaatio. Se ei myöskään sellaisenaan päde rekisteritransduktoreille.

Rekisteriautomaattien tilasiirtymien määrän minimoinnista Cohen-Sygal ja Wintner eivät esitä muita tuloksia kuin sen, että kyseessä on NP-kova ongelma [8, s. 67]. He kuitenkin toteavat, että käytännön toteutuksissa oleellista on pyrkiä minimoimaan samanaikaisesti sekä tilojen, tilasiirtymien että rekistereiden määrää. Lisäksi pitäisi pyrkiä nopeuttamaan automaattien suoritusta välttämällä tilanteita, joissa yhdeltä tilalta lähtisi samalla syötteellä useita sallittuja siirtymiä. Jos automaatissa ei tapahdu tällaisia haarautumisia, sen suoritusaika riippuu ainoastaan lineaarisesti syötemerkkijonon pituudesta. Tällaisen lineaarisen rekisteriautomaatin muodostaminen mielivaltaisesta rekisteriautomaatista on NP-täydellinen ongelma [8, s. 68].

3.5 Muita automaattien ja transduktorien laajennuksia

3.5.1 Merkkidiakriitit

XFST on Xeroxin kehittämä työkaluohjelmisto äärellisten automaattien ja transduktorien käsittelyyn [26]. XFST ei tue rekisteriautomaatteja tai -transduktoreja, mutta se sisältää *merkkidiakriiteiksi* (engl. *flag diacritics*) kutsutun toiminnon, joka on periaatteeltaan hyvin samankaltainen rekisterioperaatioiden kanssa.

Merkkidiakriitit muodostuvat kaksikoista (*operaatio*, *piirre*) tai kolmikosta (*operaatio*, *piirre*, *arvo*). Näistä piirteet vastaavat rekisteritransduktorin rekistereitä ja arvot rekisteritransduktorin rekisteriaakkoston arvoja. Alkutilassa merkkidiakriitit saavat ns. neutraalin arvon, joka vastaa rekisteritransduktorin rekisterin alustusarvoa # [4, 7.3].

Cohen-Sygalin ja Wintnerin rekisteritransduktorien ja XFST:n merkkidiakriittien välillä voidaan löytää kaksi merkittävää eroa. Ensinnäkin siinä missä rekisteritransduktorit käyttävät vain kahta eri rekisterioperaatiota (luku ja kirjoitus), merkkidiakriitteja varten on määritelty seuraavat kuusi eri operaatiotyyppiä [4, 7.3.2]:

1. Operaatio **P** (*Positive (re)setting*) vastaa rekisteritransduktorin kirjoitusoperaatiota: se asettaa annetulle piirteelle annetun arvon.

2. Operaatio **R** (*Require*) vastaa rekisteritransduktorin lukuoperaatiota: se estää siirtymän, mikäli annetulla piirteellä ei ole annettua arvoa. Operaatio voidaan suorittaa myös ilman *arvo*-parametria, jolloin siirtymä estetään, mikäli piirteen arvo on neutraali.

3. Operaatio **D** (*Disallow*) on R-operaatiolle käänteinen operaatio. Se estää siirtymän, mikäli annetulla piirteellä on annettu arvo. Ilman parametriä *arvo* se estää siirtymän kaikilla paitsi piirteen neutraalilla arvolla.

4. Operaatio **U** (*Unification*) sallii siirtymän vain, mikäli annetun piirteen arvo on neutraali tai sama kuin parametrina annettu arvo. Lisäksi se asettaa piirteen arvon annetun parametrin mukaiseksi. Beesleyn ja Karttusen mukaan U-operaatio on kaikkein eniten käytetty merkkidiakriittien tyyppi [4, 7.3.1].

5. Operaatio **C** (*Clear*) asettaa annetun piirteen arvon neutraaliksi. Sen yhteydessä ei käytetä parametria *arvo*.

6. Operaatio **N** (*Negative (re)setting*) asettaa annetun piirteen arvoksi parametrin *arvo* negaation. Negaatiot käyttäytyvät myöhemmin tulevissa R-, D- ja U-operaatioissa jossain määrin kuin piirteen arvo olisi mikä tahansa muu kuin

kyseinen arvo. Tarkempia negaation käyttäytymiseen liittyviä sääntöjä ei käsitellä tässä.

Toinen ero rekisterioperaatioiden ja merkkidiakriittien välillä on siinä, että transduktorin tilasiirtymäverkossa merkkidiakriitit sijoitetaan suoraan siirtymän syötemerkkien joukkoon, kun rekisterioperaatiot liittyvät siirtymiin erillisinä attribuutteina. Käytännössä tämä ero näkyy transduktorisovellusten kehittäjälle siinä, että merkkidiakriitit on erikseen asetettava sekä siirtymän syöte- että tulostepuolelle, mikäli niiden halutaan vaikuttavan samalla tavalla sekä transduktorissa itsessään että sen käänteistransduktorissa.

Cohen-Sygal ja Wintner [8, s. 55] mainitsevat merkkidiakriittien heikkoudeksi sen, että menetelmälle ei ole esitetty yhtä vahvaa matemaattista kuvausta kuin rekisteritransduktoreille. Lisäksi heidän mukaansa merkkidiakriittien käsittelyyn liittyviä algoritmeja ja niiden laskennallista kompleksisuutta ei ole analysoitu niitä käsittelevissä artikkeleissa yhtä tarkasti kuin vastaavia rekisteritransduktoreihin liittyviä algoritmeja.

3.5.2 Pinoautomaatit ja -transduktorit

Pinoautomaatti (engl. *pushdown automaton*) poikkeaa äärellisistä automaateista ja transduktoreista siten, että se siihen kuuluu tilasiirtymäverkon lisäksi rajoittamaton määrä apumuistia. Tämä laajennus on teoreettisesti merkittävä, sillä se mahdollistaa aidosti säännöllisiä kieliä laajemman kielijoukon tunnistamisen.

Pinoautomaatin apumuisti esitetään pinona (engl. *stack*). Hopcroftin ja Ullmanin [12, 5.2] määritelmän mukainen pinoautomaatti

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

jossa Q on äärellinen joukko tiloja, Σ on syöteaakkosto, Γ pinoaakkosto, δ siirtymärelaatio, q_0 alkutila, Z_0 pinon päällimmäinen (ja ainoa) merkki automaatin alkutilassa ja $F \in Q$ lopputilojen joukko.

Siirtymärelaatio δ ohjaa pinoautomaatin siirtymiä tilalta toiselle liittämällä automaatin nykyisen tilan q, syötemerkkijonon seuraavan merkin a ja pinon päällimmäisen merkin g muodostamaan kolmikkoon (q, a, g) joukon sallittuja seuraajatiloja sekä niitä vastaavat pinon päälle lisättävät merkit. Siirtymässä voidaan myös jättää lukematta merkkejä syötemerkkijonosta, eli sallitaan, että $a = \epsilon$.

Kaikissa siirtymissä pinon päältä poistetaan täsmälleen yksi merkki g, mutta sen tilalle lisättäviä merkkejä voi olla mikä tahansa äärellinen määrä, myös nolla.

Tästä syystä pino voi vapaasti kasvaa tai lyhentyä. On olemassa kaksi eri määritelmää sille, missä tilanteessa pinoautomaatti hyväksyy syötemerkkijonon. Ensimmäinen määritelmä perustuu äärellisen automaatin tavoin siihen, onko pinoautomaatti viimeisen siirtymän jälkeen päätynyt johonkin lopputiloista F. Toinen määritelmä hyväksyy syötteen silloin, kun viimeisen siirtymän jälkeen pino on tyhjä. Voidaan osoittaa, että nämä määritelmät ovat ekvivalentit: kaikki kielet, jotka voidaan tunnistaa jollakin lopputilojen mukaan hyväksyvän pinoautomaatin avulla, voidaan myös tunnistaa tyhjän pinon mukaan hyväksyvän pinoautomaatin avulla ja päin vastoin [12, s. 114].

Hopcroft ja Ullman myös todistavat [12, s. 115], että pinoautomaateilla tunnistettavat kielet ovat täsmälleen niin sanotut *kontekstittomat kielet* (engl. *context-free language*). Kieli on kontekstiton, jos se voidaan tuottaa kontekstittoman kieliopin avulla. Hopcroftin ja Ullmanin [12, 4.2] mukaan kontekstiton kielioppi on

$$G = (V, T, P, S),$$

jossa V on *muuttujien* tai *välikemerkkien* (engl. *variable*, *nonterminal symbol*) joukko, T on *päätemerkkien* (engl. *terminal*) joukko, $P \in V \times (V \cup T)^*$ *produktioiden* (engl. *production*) joukko ja $S \in V$ on kieliopin *lähtösymboli* (engl. *start symbol*).

Päätemerkeistä koostuva merkkijono kuuluu kontekstittoman kieliopin G kuvaamaan kieleen, mikäli se voidaan johtaa lähtösymbolista S yhden tai useamman produktion avulla. Esimerkiksi jos $V = \{A, S\}$, $T = \{a, b\}$ ja S on lähtösymboli, seuraavat produktiot muodostavat kontekstittoman kieliopin kielelle $L(G) = a^n b^{2n}$:

$$S \rightarrow A \tag{3.4}$$

$$A \rightarrow \epsilon \tag{3.5}$$

$$A \rightarrow aAbb \tag{3.6}$$

Voidaan nähdä, että merkkijono $aabbbb$ kuuluu kieleen $L(G)$, koska se voidaan johtaa lähtösymbolista S seuraavasti:

$$S \xrightarrow{3.4} A \xrightarrow{3.6} aAbb \xrightarrow{3.6} aaAbbbb \xrightarrow{3.5} aabbbb$$

Pinoautomaatit voidaan yleistää *pinotransduktoreiksi* (engl. *pushdown transducer*) vastaavalla tavalla kuin äärelliset automaatit yleistettiin äärellisiksi transduktoreiksi: lisäämällä tilasiirtymiin tulostemerkkijonot [22, 5.4].

Koska pinoautomaatit ja -transduktorit kykenevät tunnistamaan kieliä, joita äärellisillä automaateilla ei voi käsitellä, voisi ajatella, että äärellisiä automaatteja tai

transduktoreja ei käytännössä kannattaisi käyttää ollenkaan. Näin ei kuitenkaan ole. Rajattoman apumuistin käyttö tuottaa myös ongelmia, sillä äärellisille automaateille ja transduktoreille pätevät voimakkaat matemaattiset tulokset ja tehokkaat algoritmit eivät aina yleisty pinoautomaateille. Esimerkiksi mielivaltaisen kontekstittoman kielen kuvajoukko pinotransduktorin määräämässä relaatiossa ei välttämättä ole kontekstiton [22, 5.4].

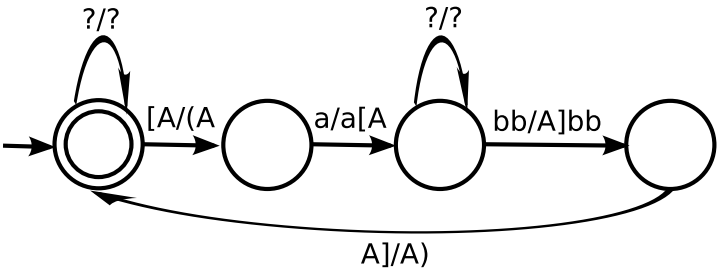
3.5.3 Kontekstittoman kielen jäsentäminen transduktorilla

Vaikka transduktorilla ei olekaan mahdollista suoraan tunnistaa ei-säännöllisiä kieliä, voidaan niitä kuitenkin käyttää hyväksi myös yleisten kontekstittomien kielten tunnistamisessa ja jäsentämisessä. Emmanuel Rochen [21, 8.3] esittelemässä menetelmässä kontekstittoman kielen produktiosääntöinä esitetty kielioppi muunnetaan transduktoriksi siten, että jokaisesta produktiosta $p_i \in P$ tehdään oma transduktorinsa T_i. Varsinaisessa jäsennysprosessissa käytetään näistä muodostettua transduktorien yhdistettä $T = \bigcup_i T_i$.

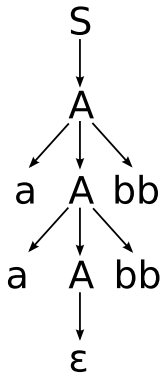
Jäsennettävä merkkijono syötetään transduktorin T läpi. Jos transduktori hylkää syötteen, merkkijono ei kuulu kyseiseen kieleen. Jos transduktori hyväksyy syötteen, verrataan transduktorin tuottamaa tulosta syötteenä annettuun merkkijonoon. Mikäli ne ovat samat, kuuluu alkuperäinen merkkijono kieleen, ja tuloksena saatu merkkijono kuvaa jäsennyspuuta sarjallistetussa muodossa. Jos taas transduktorin tuottama tulos poikkeaa syötteestä, toistetaan prosessia syöttämällä tuloksena saatu merkkijono tai merkkijonot uudelleen saman transduktorin läpi. Tätä jatketaan, kunnes transduktori joko hylkää syötteen tai saavuttaa tilan, jossa syöte ei enää muutu.

Tutkitaan esimerkin vuoksi edellisen luvun kontekstitonta kielioppia, joka muodostuu produktioista 3.4 – 3.6. Muodostetaan produktiota 3.6 vastaavan transduktori Rochen [21, s. 281] käyttämää merkintätapaa noudattaen. Tämä transduktori on esitetty kuvassa 3.5. Transduktori muuntaa esimerkiksi merkkijonon $[AakissabbA]$ muotoon $(Aa[AkissaA]bbA)$, säilyttää merkkijonon $kissa$ sellaisenaan ja hylkää merkkijonon $[AkissaA]$. Samaan tapaan muodostetaan muitakin produktioita vastaavat transduktorit: kukin korvausta vailla oleva välikesymboli esitetään hakasulkurakenteen ja korvattu välikesymboli kaarisulkurakenteen avulla. Jäsentimessä käytettävä transduktori T muodostetaan kolmen produktiotransduktorin yhdisteestä.

Merkkijono $aabbbb$ jäsentyy transduktorin T avulla siten, että se sijoitetaan korvausta vailla olevaa lähtösymbolia S esittävään hakasulkurakenteeseen $[SaabbbbS]$. Tämä merkkijono syötetään toistuvasti transduktorin T läpi, kunnes se lopulta ei



Kuva 3.5: Produktiota $A \rightarrow aAbb$ vastaava Rochen [21, s. 281] transduktoreihin perustuvassa kontekstittomien kielten jäsennysmenetelmässä käytetty osatransduktori. Siirtymät $?/?$ ovat lyhennysmerkintöjä, joilla tarkoitetaan sitä, että mikäli muu siirtymä tilalta ei ole sallittu, transduktori siirtää yhden merkin syötemerkkijonosta tulostemerkkijonoon sellaisenaan.



Kuva 3.6: Rochen kontekstittomien kielten transduktorijäsentimen tuottaman jäsennyksen $(S(Aa(Aa(AA)bbA)bbA)S)$ esitys jäsennyspuun muodossa.

enää muutu:

$$
\begin{aligned}
[Saabbbb S] &\rightarrow \\
(S[Aaabbbb A]S) &\rightarrow \\
(S(Aa[AabbA]bA)S) &\rightarrow \\
(S(Aa(Aa[AA]bbA)bbA)S) &\rightarrow \\
(S(Aa(Aa(AA)bbA)bbA)S) &\rightarrow \\
(S(Aa(Aa(AA)bbA)bbA)S)
\end{aligned}
$$

Lopputulokseksi saatu merkkijono voidaan esittää jäsennyspuuna, jossa kukin kaarisulkurakenne vastaa yhtä jäsennyspuun sisäsolmua (kuva 3.6).

4 Transduktorien sovelluksista

Valtaosa transduktorien ja rekisteritransduktorien sovelluksista liittyy kieliteknologiaan. Transduktorien osalta kieliteknologian sovelluksia käsitellään luvussa 4.1 ja muita sovellusalueita luvussa 4.2. Luvussa 4.3 kerrotaan rekisteritransduktorien mahdollisista sovelluksista kieliteknologian piirissä. Ohjelmointitekniseltä kannalta transduktorien käyttö on sovelluksissa joskus hankalaa. Tätä ongelmaa selitetään luvussa 4.4.

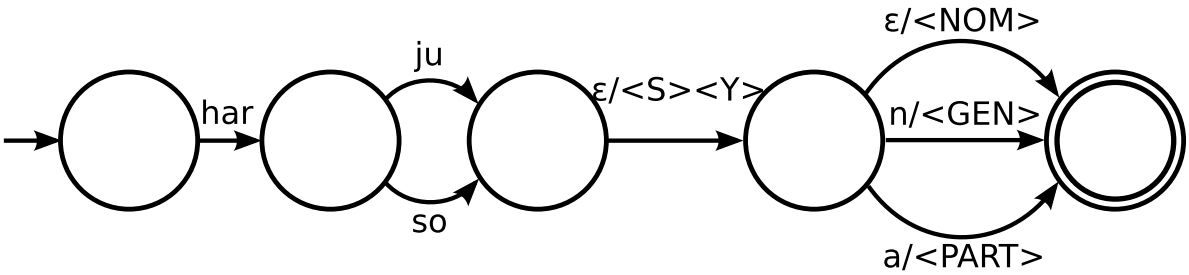
4.1 Transduktorien käyttö kieliteknologiassa

4.1.1 Sanojen morfologinen analyysi

Morfologia eli *muoto-oppi* on sanojen muodostumista, taivutusta, johtamista ja muuta sanojen rakennetta tutkiva kielitieteen osa-alue. *Morfologinen analyysi* tarkoittaa siis sanan jakoa rakenneosiinsa ja näiden osien merkityksen tutkimista.

Morfologiassa sellaisia sanan merkityksellisiä osia, joita ei enää voida jakaa pienempiin osiin, kutsutaan *morfeemeiksi*. Morfeemit esiintyvät sanoissa konkreettisina merkkijonoina, joita kutsutaan *morfeiksi*. Jos jokin morfeemi voi ilmetä sanoissa useana erilaisena morfina, näitä morfeja kutsutaan kyseisen morfeemin *allomorfeiksi*. Esimerkiksi sana $"hopeaa"$ muodostuu morfeista $"hopea"$ ja $"a"$, joista ensimmäinen on sanan hopea juuri ja toinen yksikön partitiivin taivutuspääte. Samoin $"hopeata"$ koostuu morfeista $"hopea"$ ja $"ta"$, ja morfeemit ovat samat kuin edellisessä esimerkissä. Havaitaan, että yksikön partitiivilla on tässä kaksi eri allomorfia, $"a"$ ja $"ta"$. Sanojen kirjoitetussa tai puhutussa tekstissä esiintyviä muotoja kutsutaan *pintamuodoiksi* (engl. *surface form*) [15, s. 35].

Morfologisen analyysin tavoitteena on löytää sanoille niiden mahdolliset merkitykset. Nämä merkitykset esitetään *kieliopillisten sanojen* (engl. *grammatical word*) muodossa. Kieliopillinen sana koostuu sanan perusmuodosta (yleensä siitä muodosta, missä sana esitettäisiin esimerkiksi sanakirjoissa) ja sanaluokkaa, johtimia ja taivutusmuotoja kuvaavista *leimoista* (engl. *tag*). Edellä esitettyjä pintamuotoja $"hopeaa"$ ja $"hopeata"$ molempia vastaa kieliopillinen sana *hopea*<*S*><*Y*><*PART*>, jossa <*S*> on sanaluokkaa (substantiivi) kuvaava leima, <*Y*> tarkoittaa yksikköä ja



Kuva 4.1: Morfologinen transduktori sanojen $''harju''$ ja $''harso''$ nominatiivi-, genetiivi- ja partitiivimuodoille.

<*PART*> partitiivia. [1]

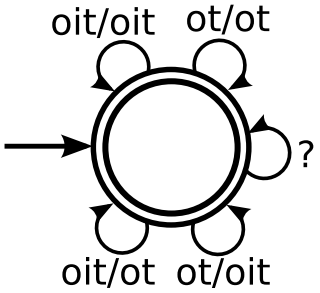
Yksinkertaistettu esimerkki morfologisessa analyysissa käytettävästä transduktorista on esitetty kuvassa 4.1. Tämä transduktori tunnistaa sanat $''harju''$ ja $''harso''$ kolmessa taivutusmuodossa, joten se hyväksyy yhteensä kuusi eri merkkijonoa. Esimerkiksi pintamuodolle $''harsoa''$ saadaan transduktorista tulosteeksi kieliopillinen sana *harso*<*S*><*Y*><*PART*>. Käytännön sovelluksissa esiintyvät transduktorit tunnistavat yleensä tuhansia sanoja ja enemmän taivutusmuotoja, mutta periaate niissä on sama.

Ajatus äärellisten transduktorien käytöstä morfologisessa analyysissa esitettiin jo 1970-luvulla. Beesleyn ja Karttusen [4, s. 33] mukaan ensimmäisen kerran asiasta kirjoitti Douglas Johnson vuonna 1972 julkaistussa kirjassaan *Formal aspects of phonological description*. Jo pitkään lingvisteillä oli ollut tapana kehitellä monimutkaisia korvaussääntöjä, joissa kieliopillisista sanoista vaihe vaiheelta edettiin kohti sanan pintamuotoa. Johnson havaitsi, että nämä säännöt voitiin esittää transduktorien avulla. Edelleen transduktoreja yhdistämällä koko kieliopillisten sanojen ja pintamuotojen välinen muunnossäännöstö voitiin kuvata yhden transduktorin avulla.

Myös Kimmo Koskenniemi esitti vuonna 1983 äärellisiä automaatteja oman ns. *kaksitasomorfologiansa* tekniseksi toteutusvälineeksi [16, s. 95]. Tuolloin Koskenniemi myös julkaisi oman suomen kielen morfologisen analysaattorinsa, jonka säännöstö oli toteutettu käsin suunnitellun äärellisen automaatin avulla. Hän esitti jo tuolloin, että automaatit voitaisiin koostaa myös tietokoneavusteisesti [16, s. 105]. Myöhemmin osoitettiin, että koko Koskenniemen kaksitasomorfologia on täysin toteutettavissa äärellisten transduktorien avulla [4, s. 34].

Morfologisissa transduktoreissa kieliopillisten sanojen leimoille varataan kulle-

[1] Kirjallisuudessa leimojen nimet lyhennetään yleensä vastaavista englanninkielisistä termeistä, mutta tässä tutkielmassa käytetään suomenkielisiä lyhenteitä. Yleistä on myös merkitä leimat muodossa *hopea+S+Y+PART*.



Kuva 4.2: Transduktori T, jota voidaan käyttää apuna kirjoitusvirheitä sisältävien sanojen tunnistamisessa.

kin oma aakkoston symboli. Siten transduktorit operoivat kahden eri aakkoston välillä, koska pintamuodoissa leimoja ei käytetä. Kieliopillisten sanojen muoto ja käytettävät leimat eivät suoraan määräydy analysoitavasta kielestä, vaan ne voidaan valita suhteellisen vapaasti sovelluksen tarpeiden mukaan. Usein morfologiset transduktorit kootaan pienemmistä osatransduktoreista. Tällöin joudutaan myös kehittämään mahdollisesti useitakin välikieliä ja -aakkostoja, jotka ovat eräänlaisia kieliopillisten sanojen ja pintasanojen välimuotoja. Leimojen järjestys on myös transduktorin kehittäjän valittavissa. Yleensä on kuitenkin helpointa asettaa ne mahdollisimman tarkasti samaan järjestykseen, missä leimoja vastaavat morfeemit esiintyvät sanan pintamuodossa. [4, 5.2.2]

Esimerkkinä morfologisen analyysin suorittamisesta useassa vaiheessa on kuvan 4.2 transduktori T, joka on suunniteltu liitettäväksi oikein kirjoitettuja suomen kielen sanoja tunnistavaan transduktoriin U. Olkoon transduktori U sellainen, että

$$\begin{aligned}(R(U))(''kirjoitan'') &= \{''kirjoittaa < V >< Y_1 >< IND >< PREES >''\} \\ (R(U))(''kirjotan'') &= \emptyset\end{aligned}$$

(<V><Y1><IND><PREES> tarkoittaa verbin yksikön ensimmäisen persoonan indikatiivin preesensmuotoa.)

Vaikka $''kirjotan''$ on virheellinen muoto, se on yleinen ja voi siten käytännössä esiintyä analysoitavassa tekstissä. Jos morfologista transduktoria käytetään esimerkiksi osana automaattista kielenkäännösohjelmistoa, tällaiset virheellisetkin sanat olisi hyvä kyetä tunnistamaan. Tämä voidaan tehdä käyttämällä yhdistettyä transduktoria $U \circ T$, joka muuntaa merkkijonon $''kirjottaa''$ merkkijonoksi $''kirjoittaa''$:

$$\begin{aligned}(R(U \circ T))(''kirjoitan'') &= \{''kirjoittaa < V >< Y_1 >< IND >< PREES >''\} \\ (R(U \circ T))(''kirjotan'') &= \{''kirjoittaa < V >< Y_1 >< IND >< PREES >''\}\end{aligned}$$

Seuraavat esimerkit osoittavat, että transduktori T osaa korjata vastaavan vir-

heen myös toiseen suuntaan, ja että tällaiset korjaukset tuottavat toisinaan myös virheellisiä analyyseja:

$$\begin{aligned}(R(U \circ T))(''jaksoitan'') &= \{\ ''jaksottaa < V >< Y_1 >< IND >< PREES >''\} \\ (R(U \circ T))(''loitota'') &= \{\ ''loitota < V >< INFINITIIVI_1 >'', \\ &\qquad\quad ''lotota < V >< INFINITIIVI_1 >''\}\end{aligned}$$

Edellä esitetyistä esimerkeistä nähdään, että jos oikein kirjoitetut sanat tunnistava transduktori on olemassa, hiukan virheellisten sanojen tunnistaminen käy suhteellisen helposti, kunhan virheanalyyseihin osataan varautua. Tällainen virheellisten muotojen tunnistus onkin syytä toteuttaa aina erillään varsinaisesta morfologisesta analyysista [4, 9.5.3].

Esimerkkinä morfologisesta transduktorista voidaan mainita 1990-luvun alkupuolella kehitetty 80 000 perusmuodossa olevan ranskankielisen sanan DICOS-sanakirjan ja siitä muodostetun 700 000 taivutetun sanan DICOF-sanaston toisiinsa liittävä DICOF-transduktori. Tämä transduktori liittää DICOF-sanastossa olevaan taivutettuun pintamuotoon (esimerkiksi $''chevaux''$, ”hevoset”) sanan DICOS-sanakirjassa olevan perusmuodon ja taivutusta kuvaavat leimat ($''cheval < N >< mp >''$, substantiivin ”hevonen” maskuliinisuvun monikko). [21, 2] DICOF-transduktorin vaatima muistitila on kokonaisuudessaan 948 kilotavua [21, s. 79], eli keskimäärin yhden sanan kaikkien muotojen tallentamiseen vaadittava muistitila on kyseisessä transduktorissa noin 12 tavua.

4.1.2 Sanojen generointi

Sanojen generoinnilla tarkoitetaan pintamuodon tuottamista haluttua merkitystä vastaavalle sanalle. Se on siis morfologiseen analyysiin käänteisprosessi. Luvussa 2.8.1 todettiin, että äärellisiä transduktoreja voi tietyin edellytyksin käyttää ”takaperin”, joten morfologiseen analyysiin kehitetty transduktori soveltuu sellaisenaan myös sanojen generointiin. Tällöin on kuitenkin tarpeen huolehtia siitä, että kieliopillisten sanojen leimojen järjestys on systemaattisesti sama kaikissa transduktorin tunnistamissa sanoissa. Tämä siksi, että transduktoria sanojen generointiin käyttävän sovelluksen on osattava muodostaa tarvitsemansa sanan kieliopillinen muoto täsmälleen sellaisena merkkijonona, joka transduktorissa vastaa kyseistä sanaa.

4.1.3 Konekääntäminen

Morfologista analyysia ja sanojen generointia tarvitaan usein molempia samassa ohjelmistossa. Konekääntäminen on hyvä esimerkki tällaisesta sovelluksesta. Siinä

lähdetekstin sanat voidaan analysoida transduktorin avulla, jolloin lauseet muuntuvat jonoiksi kieliopillisia sanoja (tai vastaavia rakenteita). Osalle sanoista voidaan saada useita mahdollisia analyyseja. Tämän jälkeen voidaan käyttää jotakin toista tilanteeseen sopivaa tekniikkaa, jolla sanojen järjestystä ja leimoja tutkimalla päätellään lauseen kieliopillinen rakenne, ja muunnetaan se vastaavaksi kohdekielen rakenteeksi. Samalla myös yksittäiset sanat korvataan vastaavilla kohdekielen kieliopillisilla sanoilla. Lopuksi nämä kieliopilliset sanat muunnetaan pintamuodoiksi käyttämällä kohdekielelle tehtyä morfologista transduktoria generointisuuntaan. Tällä menetelmällä on onnistuttu kehittämään mm. tehokas persian ja englannin kielen välillä toimiva konekäännösohjelmisto [1].

4.1.4 Puheentunnistus

Puheentunnistuksessa tavoitteena on koneellisesti tunnistaa tallennetusta akustisesta havaintoaineistosta siinä esiintyvät sanat ja lauseet. Tämä tehtävä voidaan suorittaa osissa siten, että ensin aineistosta pyritään tunnistamaan siinä mahdollisesti esiintyvät äänteet. Näin syntyvät äännejonot muunnetaan tunnistettavasta kielestä riippuvan äännemallin avulla sanoiksi, ja lopuksi sanat kootaan lauseiksi [19, 4.1]. Prosessiin liittyy paljon epävarmuutta, sillä esimerkiksi huonolaatuisesta nauhoitteesta tai epäselvästä puheesta äänteet voidaan tulkita usealla eri tavalla. Lisäksi sama äännejono voi vastata useaa merkitykseltään ja mahdollisesti kirjoitusasultaankin erilaista sanaa, jotka kuitenkin lausutaan samalla tavalla.

Puheentunnistuksessa tarvittavia muunnoksia on mahdollista toteuttaa äärellisten transduktorien avulla. Tällöin on havaittu hyödylliseksi käyttää tavallisen äärellisen transduktorin muunnelmaa, jossa tilasiirtymiin on liitetty numeeriset painot [19, 3 – 4]. Nämä painot kuvaavat siirtymän todennäköisyyden käänteisarvon logaritmia. Laskemalla yhteen syötemerkkijonoa vastaavien polkujen varrella olevat painot ja valitsemalla polku, jolle tämä summa on pienin, voidaan todennäköisin polku erottaa muista [19, 3].

Koska puheentunnistus on käytännön sovelluksissa usein tehtävä reaaliajassa, käytettyjen algoritmien tehokkuus on olennaista. Tällaisia painotettuja transduktoreja yhdistämällä ja optimoimalla on päästy tässä suhteessa hyviin tuloksiin [19, 4.1].

4.2 Transduktorien muita sovelluksia

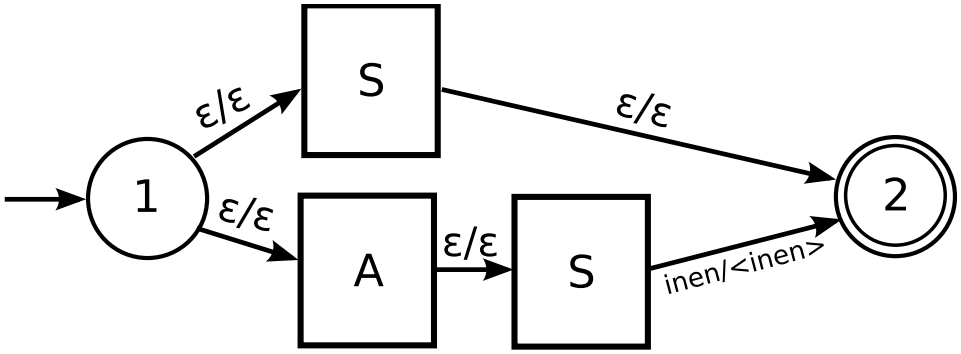
Transduktorit ovat osoittautuneet käyttökelpoisiksi myös tiedon louhinnassa rakenteellisista tai osittain rakenteellisista tekstidokumenteista. Chun-Nan Hsun ja Ming-Tzung Dungin vuonna 1998 julkaisemassa artikkelissa [13] esitetään menetelmä, jolla transduktoreja apuna käyttäen voidaan HTML-muotoisesta tekstistä, siis WWW-sivuilta, poimia automaattisesti määrämuotoista tietoa kuten henkilöiden yhteystietoja. Menetelmä edellyttää, että tiedot on esitetty jonkinlaisena rakenteellisena listana, vaikkapa HTML-luettelona. Listan tietueiden ei kuitenkaan tarvitse olla keskenään täysin samanmuotoisia, vaan attribuuttien määrä ja järjestys voi vaihdella tietueiden välillä. Transduktoreja käytetään menetelmässä muunnettaessa HTML-muotoista lähdetekstiä jonoksi tunnistettuun tietueeseen liittyviä attribuutteja.

Transduktorit ovat käyttökelpoisia apuvälineitä myös teoreettisessa tietojenkäsittelytieteessä sekä matematiikassa, jossa ne edustavat abstraktia mallia rajoitetulla muistilla varustetulle automaatille. Voidaan esimerkiksi osoittaa, että mielivaltaisen binääriluvun kertominen vakiolla 5 on mahdollista suorittaa äärellisellä transduktorilla, mutta vain, jos binääriluku on annettu käyttäen käänteistä esitystä (vähiten merkitsevä bitti ensin). Samoin voidaan osoittaa, että kahden mielivaltaisen binääriluvun kertolaskua ei voida suorittaa äärellisellä transduktorilla, jolloin se ei myöskään onnistu koneella, jossa on rajoitettu määrä muistia. [14, s. 5]

4.3 Rekisteritransduktorit kieliteknologiassa

Tavanomaisten transduktorien rakenteesta johtuen niillä on hankalaa käsitellä tehokkaasti eräitä luonnollisissa kielissä esiintyviä morfologisia piirteitä. Cohen-Sygal ja Wintner [8] sekä Beesley ja Karttunen [4, s. 212] listaavat muutamia tällaisia ilmiöitä:

1. Jos sanan pintamuodossa kauaksi toisistaan jäävien morfeemien välillä on riippuvuuksia, kasvaa transduktorin koko merkittävästi. Esimerkiksi *sirkumfiksit* eli sanan vartalon molemmille puolille liitettävät affiksit ovat tässä suhteessa ongelmallisia. Suomen kielessä ei sirkumfiksejä käytetä, mutta yhdyssanojen muodostuksessa voidaan havaita samanlaisia pitkän etäisyyden riippuvuuksia. Suomessa adjektiivia ja substantiivia ei sellaisenaan voi tavallisesti kirjoittaa yhteen yhdyssanaksi, mutta jos niiden perään liitetään johdin *-inen*, on yhdistäminen sallittua. Esimerkiksi $"kovapinta"$ ja $"pintainen"$ eivät ole kieliopillisesti oikeita suomen kielen sanoja, mutta $"kovapintainen"$ on. Tällöin adjek-



Kuva 4.3: Transduktori, joka tunnistaa suomen kielen substantiivit ja muotoa adjektiivi+substantiivi+inen olevat yhdyssanat. S on substantiivit tunnistava transduktori ja A adjektiivit tunnistava transduktori. Yksinkertaisuuden vuoksi mahdollisia yhdysmerkkejä sanan osien välissä tai tilannetta, jossa substantiivi päättyy kirjaimeen i, ei huomioida. Havaitaan, että tällaisen transduktorin rakentaminen vaatii osan S kopioimisen kahteen eri paikkaan. Tämä johtuu siitä, että transduktorilla ei ole käytettävissä apumuistia, johon voitaisiin tallentaa tieto siitä, onko aikaisemmin kuljettu osan A kautta vai ei.

tiivin ja *-inen*-johtimen välillä oleva riippuvuus vaatii transduktorin osien kopiointia kuvan 4.3 mukaisesti.

2. Muun muassa heprean kielessä sanoja voidaan taivuttaa tai johtaa lisäämällä kirjaimia sanan vartalon sisään. Joskus nämä kirjaimet on jopa asetettava lomittain vartalon kirjainten kanssa. Myös tässä tapauksessa transduktorin koko kasvaa, koska sanan eri muodot joudutaan esittämään transduktorissa erillisinä polkuina.

3. Joissakin kielissä sanoista voidaan johtaa uusia sanoja kahdentamalla kyseinen sana tai jokin sen osa. Malaijin ja indonesian kielissä sanan monikkomuoto muodostetaan kirjoittamalla sana kahdesti peräkkäin. Tätä ilmiötä kutsutaan *reduplikaatioksi*. Reduplikaatio on myös transduktorien kannalta ongelmallista, sillä kieli $\{ww : w \in \Sigma^*\}$ ei ole säännöllinen, eikä siten käsiteltävissä äärellisten transduktorien avulla. Käytännön kieliteknologisissa sovelluksissa ongelma voidaan kiertää hyväksymällä ainoastaan rajoitetun mittaisten merkkijonojen kahdentaminen, mutta tällöinkin transduktorin koko voi helposti kasvaa haitallisen suureksi [8].

Kaikissa näissä ongelmatapauksissa rekistereiden käyttö voi vähentää tarvetta transduktorin osien kopioimiseen. Sirkumfiksien tapauksessa voidaan käyttää rekisteriä, jonka arvo asetetaan käsiteltäessä sanan alussa olevaa sirkumfiksin osaa,

ja vastaava loppuosa hyväksytään sanan lopussa vain, jos rekisterin arvo on oikea. Sanan vartalon sisällä tapahtuvat taivutukset voidaan käsitellä kirjoittamalla rekisteriin taivutusmuotoa vastaava arvo silloin, kun sanassa ensimmäisen kerran tavataan tähän taivutusmuotoon viittaava merkki, ja hyväksymällä myöhemmin vain sellaisia taivutukseen liittyviä merkkejä, jotka ovat sopusoinnussa alkuperäisen havainnon kanssa.

Eräs reduplikaatiota läheisesti muistuttava ongelma ratkaistaan rekisteritransduktoria käyttäen tämän tutkielman luvussa 5.6, jossa käsitellään tämän ongelman yleistystä, merkkijonojen rajoitettua toistoa.

On syytä huomata, että reduplikaation ja muidenkin edellä mainittujen tapausten käsittelyyn on jo ennen rekisteritransduktoreja kehitetty muitakin ratkaisuja. Tällainen on esimerkiksi Beesleyn ja Karttusen *COMPILE-REPLACE*-menetelmä [3], jossa transduktorin tilasiirtymäverkon siirtymiin sisällytetään tavallisten merkkijonojen lisäksi säännöllisiä lausekkeita. Beesleyn artikkelissa [2] on esitelty muitakin menetelmiä, joista yksi on luvussa 3.5.1 mainittu merkkidiakriittien käyttö.

4.4 Ohjelmointiteknisiä ongelmia

Automaattien, transduktorien ja rekisteritransduktorien käyttöön käytännön sovelluksissa liittyy myös ongelmia, jotka eivät johdu säännöllisten kielten tai säännöllisten lausekkeiden ilmaisuvoiman rajoituksista. Wintner [25] on verrannut XFST:n [26] ohjelmistoa käyttävää äärellisiin automaatteihin perustuvaa heprean morfologisen kieliopin toteutusta saman kielen tunnistavaan Java-ohjelmointikielellä tehtyyn suoraan toteutukseen. Hän havaitsi useita osa-alueita, joissa suora toteutus osoittautui äärellisiä automaatteja vahvemmaksi.

Monimutkaisia ohjelmistoja kehitettäessä on hyödyllistä pystyä jakamaan ohjelmisto toisistaan riippumattomiin komponentteihin sekä toteuttamaan tietorakenteita käsittelevät algoritmit siten, että ne eivät tarpeettomasti tulisi riippuvaisiksi algoritmin kannalta epäoleellisista tietorakenteiden piirteistä. Oliopohjaisena kielenä Java mahdollistaa ohjelmiston kehittämisen tällä tavoin. Siispä esimerkiksi morfologisen kieliopin toteutuksessa voidaan substantiivit ja adjektiivit toteuttaa luokkina, joilla on yhteinen yläluokka (nominit).

Automaatit ja transduktorit sen sijaan koodataan tavallisesti säännöllisten lausekkeiden avulla, jolloin ainoa käsiteltävä tietotyyppi on merkkijono. Jos tällä tavoin toteutettuun kielioppiin halutaan tehdä jokin muutos, esimerkiksi lisätä merkkijonoon uusi taivutusmuotoa kuvaava leima, joudutaan usein tekemään muutoksia kaikkiin niihin kieliopin osiin, jotka käsittelevät tätä merkkijonoa. Kunnollisen tyyp-

pijärjestelmän puuttuessa kääntäjäkään ei usein osaa varoittaa, mikäli jokin tarvittava muutos jää epähuomiossa tekemättä.

Säännölliset lausekkeet sisältävät koko tunnistettavan kielen määrittelyn. Luonnollisten kielten tapauksessa tämä sisältää siis sekä kielen sanaston että sanojen muodostuksessa noudatettavat säännöt. Tämä hankaloittaa sanaston, sananmuodostussääntöjen ja ohjelmiston teknisen toteutuksen erottamista toisistaan, jolloin helposti joudutaan tilanteeseen, jossa yhden henkilön on osattava kehittää näitä kaikkia samanaikaisesti. Perinteisiä ohjelmointimenetelmiä käytettäessä voidaan kielen tunnistavan ohjelmiston tekninen toteutus jättää ohjelmoinnin ammattilaisille, ja sanasto sekä sananmuodostussäännöt voidaan kuvata muodossa, joka on lingvistien kannalta mielekäs ja helppo käyttää. [25, s. 103]

Automaatteja ja transduktoreja muodostettaessa niiden minimointi ja muu optimointi on usein hidasta. Erityisesti yhdistetyn automaatin tai transduktorin muodostaminen voi lisäksi vaatia erittäin paljon työmuistia. [25, s. 103] Useimmiten käännös on suoritettava kokonaan, vaikka automaattiin tai transduktoriin tehty muutos olisi pieni. Tämä hidastaa muutosten testaamista. Java-toteutuksessa riittää tavallisesti kääntää vain muuttunut luokkatiedosto uudelleen.

5 Rekisteritransduktorin toteutus

Tässä luvussa toteutetaan yksinkertainen sovellus, joka lukee rekisteritransduktorin kuvauksen tekstitiedostosta ja rakentaa sitä vastaavan tilasiirtymäverkon tietokoneen keskusmuistiin. Muistiin ladattua transduktoria hyödyntämällä sovellus kykenee tekemään merkkijonomuunnoksia transduktorin syöte- ja tulostekielten välillä.

Rekisteritransduktori voidaan kuvata joko tilasiirtymäverkkona tai säännöllisen lausekkeen avulla. Toteutettava ohjelma tukee molempien esitystapojen käyttöä. Tilasiirtymäverkon kaarilistaesityksestä kerrotaan luvussa 5.1. Luvuissa 5.2 ja 5.3 puolestaan määritellään rekisteritransduktorin kuvaamisessa käytettävien säännöllisten lausekkeiden muoto ja kerrotaan, kuinka sovellus muuntaa säännölliset lausekkeet tilasiirtymäverkoksi. Tilasiirtymäverkon muodostamisen jälkeen sitä on suorituskyvyn parantamiseksi opitmoitava. Sovellukseen toteutetut optimointimenetelmät on kuvattu luvussa 5.4 ja sovelluksen yleistason jako C++-kielen luokkiin luvussa 5.5.

Luvussa 5.6 määritellään säännöllisille lausekkeille rajoitetun toiston operaattori. Erityisesti tätä operaattoria hyödyntäen luvuissa 5.7 ja 5.8 testataan toteutettua ohjelmaa ja verrataan tuloksia toisen transduktoriohjelmiston avulla saatuihin.

5.1 Tilasiirtymäverkot kaarilistoina

Suoraviivaisin tapa kuvata rekisteritransduktori $T = (Q, A, B, q_0, \Gamma, n, E, F)$ on yksinkertaisesti listata eksplisiittisesti kaikki T:n komponentit. Toteutettavan sovelluksen yksinkertaistamiseksi asetetaan näille komponenteille seuraavat rajoitukset:

1. Tilojen joukko Q esitetään kokonaislukuina väliltä $[0, N-1]$, jossa N on tilojen lukumäärä.

2. Syöte- ja tulosteaakkostot A ja B muodostuvat C++-kielen kahdeksanbittisen `char`-tietotyypin arvoista (siis kokonaisluvuista väliltä $[0, 255]$) pois lukien arvot 0 (NUL), 10 ja 13 (rivinvaihtomerkit LF ja CR), 32 (välilyönti), 40 ja 41 (sulkumerkit), 42 (*), 47 (/), 91 ja 93 (hakasulkumerkit) ja 124 (|). Nämä rajaukset mahdollistavat sen, että sekä tässä luvussa esiteltävässä kaarilistaesityksessä

että seuraavan luvun säännöllisissä lausekkeissa ei tarvitse huolehtia erikseen kontrollimerkkien käsittelystä.

3. Alkutila q_0 on aina tila 0.

4. Rekisteriaakkosto Γ sisältää kokonaisluvut väliltä $[0, M]$, jossa $M + 1$ on rekisterisymbolien lukumäärä. Rekisterin alustussymbolina $\#$ toimii rekisterin arvo 0.

Näillä rajoituksilla rekisteritransduktori tulee täysin kuvatuksi, kun seuraavat tiedot saadaan määritettyä:

1. Tilojen lukumäärä N,

2. rekistereiden lukumäärä n,

3. suurin rekisteriaakkostoon kuuluva kokonaisluku M,

4. lopputilojen joukko $F \in Q$ ja

5. tilasiirtymät E.

Tilasiirtymät $e = (q, a, b, o, i, \gamma, q') \in E$ voidaan esittää listana. Tämä lista on käytännössä tilasiirtymäverkon esitys kaarilistana. Toteutettavassa ohjelmassa kaarilistan alkio esitetään rivinä transduktorin kuvauksen sisältävässä tiedostossa. Yhtä kaarta esittävän rivin muoto on

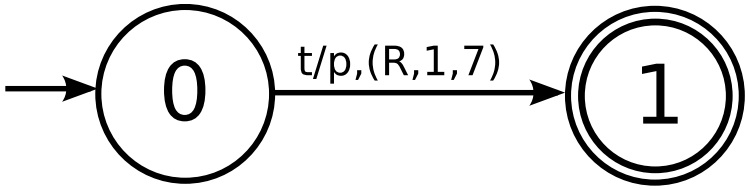
$$q{-}{>}q' \quad a/b \quad (o, i, \gamma)$$

jossa tyhjämerkki ϵ syöte- tai tulostemerkkinä a tai b esitetään välilyönnin avulla. Rekisterioperaatio (o,i,γ) jätetään pois, jos kyseiseen siirtymään ei liity rekisterioperaatiota, eli $(o, i, \gamma) = (R, 0, \#)$.

5.2 Rekisteritransduktorit säännöllisinä lausekkeina

Rekisteritransduktorin rakentaminen suunnittelemalla sille tilasiirtymäverkko ja listaamalla verkon kaaret on havainnollinen ja käytännöllinen menetelmä silloin, kun työ tehdään käsin ja transduktori on suhteellisen pieni. Transduktorit voivat kuitenkin sisältää tuhansia tai jopa miljoonia tiloja ja tilasiirtymiä. Tällöin prosessia on pakko automatisoida, mihin tällainen graafinen menetelmä ei kovin hyvin sovellu.

Säännöllisten kielten kuvaamisessa käytetyt säännölliset lausekkeet soveltuvat tämän ongelman ratkaisuun äärellisten automaattien suunnittelussa. Säännölliset



Kuva 5.1: Säännöllistä lauseketta `[t/p/R,1,7]` vastaavan transduktorin tilasiirtymäkaavio. Kaarilistaesityksessä transduktorin tilasiirtymä kirjoitettaisiin muodossa `0->1 t/p (R,1,7).`

lausekkeet voi kehittää hallittavan kokoisissa osissa ja antaa tietokoneohjelman yhdistää ne yhdeksi lausekkeeksi. Säännöllisestä lausekkeesta voidaan sitten täysin automaattisesti rakentaa ja minimoida sitä vastaava äärellinen automaatti. Olisi hyödyllistä pystyä tuottamaan rekisteritransduktoreja samalla tavalla. Äärellisiä automaatteja vastaavat säännölliset lausekkeet eivät kuitenkaan sellaisenaan sovellu transduktorien tai rekisteritransduktorien kuvaamiseen.

Määritellään nyt säännöllisten lausekkeiden laajennus, jonka avulla myös transduktorien ja rekisteritransduktorien esittäminen onnistuu. Tämä laajennus pohjautuu toisaalta Beesleyn ja Karttusen transduktoreja tukeviin säännöllisiin lausekkeisiin [4, 2.3] ja toisaalta Cohen-Sygalin ja Wintnerin rekisteriautomaatteja varten kehittämään merkintään [8, 4]. Merkintää on kuitenkin yksinkertaistettu ottamalla mukaan vain ne operaattorit, jotka välttämättä tarvitaan, jotta mielivaltaisten säännöllisten relaatioiden ja rekisterioperaatioiden esittäminen on mahdollista.

Yksinkertainen alkeellista rekisteritransduktoria (tai oikeastaan transduktorin tilasiirtymää) kuvaava säännöllinen lauseke esitetään seuraavasti

[*syötemerkkijono*/*tulostemerkkijono*/*operaatio*,*rekisteri*,*arvo*]

Tämä lauseke esittää transduktoria, jossa alku- ja lopputilan välillä on siirtymä, johon annetun syöte- ja tulostemerkkijonon lisäksi liittyy rekisterioperaatio, jonka tyyppi on *operaatio*, kohderekisteri *rekisteri* ja rekisteristä luettava tai sinne kirjoitettava arvo on *arvo*. Esimerkki tällaisesta lausekkeesta ja sitä vastaavasta transduktorista on kuvassa 5.1.

Käytetty merkintätapa mahdollistaa sen, ettei siirtymissä olevia tyhjiä ϵ-merkkijonoja tarvitse merkitä millään erillisellä merkillä. Esimerkiksi a/ϵ-siirtymä voidaan kuvata säännöllisellä lausekkeella `[a/]`. Merkintöjen lyhentämiseksi otetaan lisäksi käyttöön lyhennysmerkinnät `[a/b]` := `[a/b/R,0,0]` ja `a` := `[a/a]`.

Näin määritellyistä alkeislausekkeista voidaan yhdisteen, tulon ja sulkeuman avulla muodostaa uusia lausekkeita täsmälleen samalla tavalla kuin luvussa 2.2 ku-

vatuilla säännöllisillä lausekkeilla. Kontekstittoman kieliopin avulla rekisteritransduktorien säännölliset lausekkeet voidaan siis määritellä seuraavasti:

$$
\begin{aligned}
lauseke &\rightarrow lausekeC \\
lauseke &\rightarrow \epsilon \\
lausekeC &\rightarrow lausekeC \mid lausekeB \\
lausekeC &\rightarrow lausekeB \\
lausekeB &\rightarrow lausekeB\ lausekeA \\
lausekeB &\rightarrow lausekeA \\
lausekeA &\rightarrow lausekeA\ \texttt{*} \\
lausekeA &\rightarrow \texttt{(}\ lausekeC\ \texttt{)} \\
lausekeA &\rightarrow \texttt{[}\ smjono\ \texttt{/}\ tmjono\ \texttt{/}\ operaatio\ \texttt{,}\ rekisteri\ \texttt{,}\ arvo\ \texttt{]} \\
lausekeA &\rightarrow \texttt{[}\ smjono\ \texttt{/}\ tmjono\ \texttt{]} \\
lausekeA &\rightarrow stmerkki \\
smjono &\rightarrow smjono\ smerkki \\
smjono &\rightarrow \epsilon \\
tmjono &\rightarrow tmjono\ tmerkki \\
tmjono &\rightarrow \epsilon \\
operaatio &\rightarrow \text{R} \\
operaatio &\rightarrow \text{W}
\end{aligned}
$$

Tässä kieliopissa kaikki produktioiden oikealla puolella olevat symbolit, jotka eivät ole välikemerkkejä, ovat päätemerkkejä (pystyviiva | mukaan lukien). Välikemerkit $smerkki$, $tmerkki$, $stmerkki$, $rekisteri$ ja $arvo$ muodostavat produktiot on jätetty pois, koska niiden sisältö riippuu transduktoriin liittyvistä aakkostoista. Ne korvautuvat transduktorista riippuvilla yksittäisillä merkeillä siten, että jos rekisteritransduktori $T = (Q, A, B, q_0, \Gamma, n, E, F)$, niin

$$\begin{aligned} \forall x \in A : smerkki &\rightarrow x \\ \forall x \in B : tmerkki &\rightarrow x \\ \forall x \in A \cap B : stmerkki &\rightarrow x \\ \forall x \in \{0, 1, \ldots, n\} : rekisteri &\rightarrow x \\ \forall x \in \Gamma : arvo &\rightarrow x \end{aligned}$$

Toteutettavassa sovelluksessa säännöllisen lausekkeen avulla määritellyille transduktoreille pätevät seuraavat ominaisuudet:

1. Aakkostot A ja B ovat samat kuin luvussa 5.1 määritellyt aakkostot kaarilistan avulla esitettävälle rekisteritransduktorille.

2. Rekistereiden lukumäärä n on sama kuin suurin säännöllisessä lausekkeessa rekisterin nimenä käytetty kokonaisluku.

3. Rekisteriaakkosto Γ muodostuu kokonaisluvuista väliltä $[0, M]$, jossa M on suurin säännöllisessä lausekkeessa rekisterin arvona käytetty kokonaisluku.

Tilat ja tilasiirtymät generoidaan säännöllisiä lausekkeita käytettäessä automaattisesti luvussa 5.3 esiteltävän algoritmin mukaan.

Transduktorin tai rekisteritransduktorin esitys säännöllisenä lausekkeena ei ole yksikäsitteinen. Esimerkiksi säännölliset lausekkeet

(a | b)*[v/][v/w](a | b)*

(a | b)*[vv/w](a | b)*

(a* | b*)*[vv/w](a | b)* ja

(b | a)*[vv/w](a | b)*

kuvavat kaikki samaa relaatiota, jonka kuvan 2.2 transduktori toteuttaa.

5.3 Transduktorin muodostaminen säännöllisestä lausekkeesta

5.3.1 Säännöllisten lausekkeiden jäsentäminen

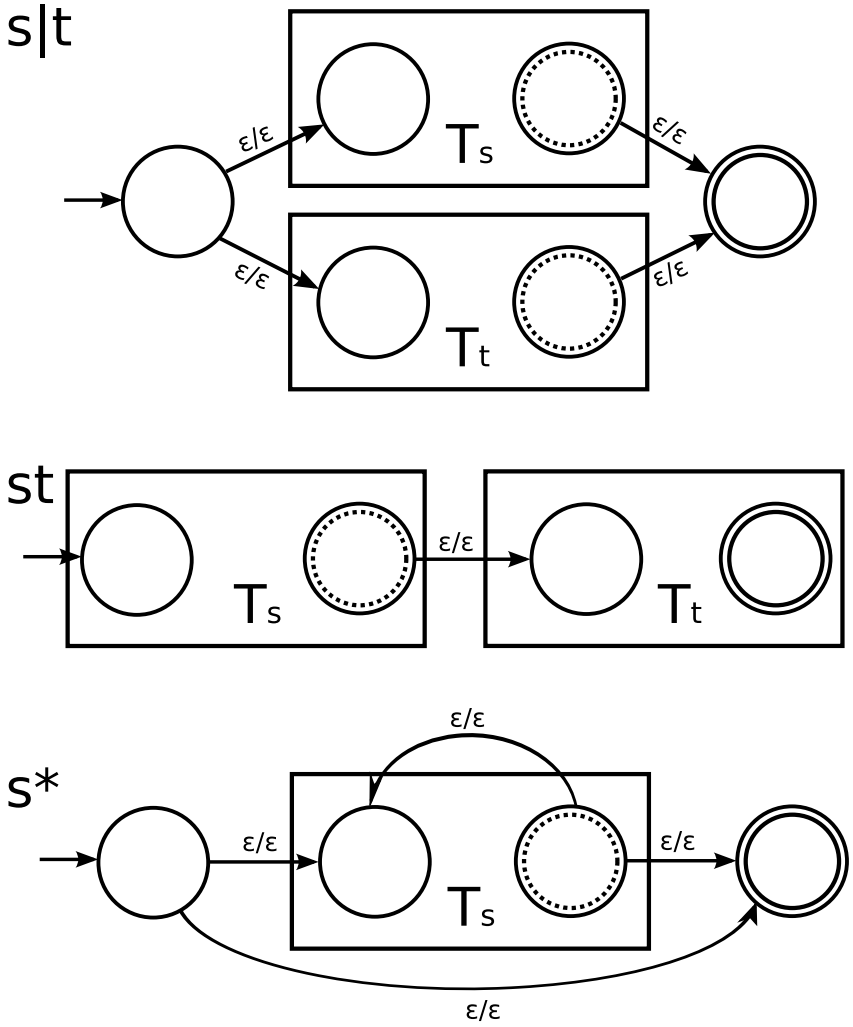
Ensimmäinen vaihe transduktorin muodostamisessa säännöllisestä lausekkeesta on säännöllisen lausekkeen jäsentäminen. Tämä voidaan tehdä luvussa 5.2 annetun

kontekstittoman kieliopin avulla. Kontekstittomien kielioppien jäsentämiseen käytetään tavallisesti *rekursiivisesti eteneviä jäsentimiä* (engl. *recursive descent parser*) [11, 6.6]. Kyseinen kielioppi on kirjoitettu helppolukuisessa muodossa, mutta jäsentimen muodostaminen sille suoraan on vaikeaa, koska kieliopissa esiintyy *vasenta rekursiota*. Tämä tarkoittaa sitä, että osa kieliopin välikesymboleista voi olla oman lausejohdoksensa vasemmanpuoleisena terminä. Välikesymbolin vasen rekursio vaikeuttaa jäsentimen kehittämistä siksi, että se voi johtaa ikuiseen silmukkaan jäsentimen koodissa.

Kontekstiton kielioppi on aina mahdollista kirjoittaa muodossa, jossa vasenta rekursiota ei esiinny [11, 6.4]. Vasemman rekursion poistamisen lisäksi luvussa 5.2 esitettyä kielioppia voidaan yksinkertaistaa huomioimalla, että toteutettavassa sovelluksessa on päätetty käyttää samoja syöte- ja tulosteaakkostoja. Näin päädytään seuraavaan rekisteritransduktorien säännöllisten lausekkeiden kontekstittomaan kielioppiin:

$$
\begin{aligned}
lauseke &\rightarrow lausekeD \\
lauseke &\rightarrow \epsilon \\
lausekeD &\rightarrow lausekeC \mid lausekeD \\
lausekeD &\rightarrow lausekeC \\
lausekeC &\rightarrow lausekeB\ lausekeC \\
lausekeC &\rightarrow lausekeB \\
lausekeB &\rightarrow lausekeA\ \texttt{*} \\
lausekeB &\rightarrow lausekeA \\
lausekeA &\rightarrow \texttt{(}\ lausekeD\ \texttt{)} \\
lausekeA &\rightarrow \texttt{[}\ mjono1\ mjono1\ \texttt{R}\ \texttt{,}\ rekisteri\ \texttt{,}\ arvo\ \texttt{]} \\
lausekeA &\rightarrow \texttt{[}\ mjono1\ mjono1\ \texttt{W}\ \texttt{,}\ rekisteri\ \texttt{,}\ arvo\ \texttt{]} \\
lausekeA &\rightarrow \texttt{[}\ mjono1\ mjono2 \\
lausekeA &\rightarrow merkki \\
mjono1 &\rightarrow \texttt{/} \\
mjono1 &\rightarrow merkki\ mjono1 \\
mjono2 &\rightarrow \texttt{]} \\
mjono2 &\rightarrow merkki\ mjono2 \\
\forall x \in A = B : merkki &\rightarrow x \\
\forall x \in \{0, 1, \ldots, n\} : rekisteri &\rightarrow x \\
\forall x \in \Gamma : arvo &\rightarrow x
\end{aligned}
$$

Kontekstittoman kieliopin rekursiivisesti etenevässä jäsentimessä jokaista välikesymbolia vastaa yksi jäsentimen proseduuri. Proseduurin on kyettävä päättelemään, mikä (jos mikään) sitä vastaavan välikesymbolin johdoksista esiintyy annetun merkkijonon alussa. Yllä olevassa kieliopissa tämä vaatii periaatteessa useiden vaihtoehtojen kokeilua. Esimerkiksi jos proseduuri $lausekeB$ saa syötteeksi merkkijonon $''a * b''$, sopivat sen molemmat lausejohdokset $lausekeA*$ ja $lausekeA$ syötteen alkuun. Kuitenkin kielioppia tutkimalla havaitaan, että jälkimmäisen lausejohdoksen valitseminen johtaisi myöhemmin umpikujaan, joten ainoastaan ensimmäinen on käytännössä mahdollinen.



Kuva 5.2: Transduktorien muodostaminen säännöllisille lausekkeille $s|t$, st ja s^*, kun lauseketta s vastaa transduktori T_s ja lauseketta t transduktori T_t. Perustuu Hopcroftin ja Ullmanin äärellisten automaattien ja säännöllisten lausekkeiden yhteyttä selittävään kuvaan [12, kuva 2.14]. Tässä konstruktiossa kaikilla muodostettavilla transduktoreilla on täsmälleen yksi lopputila, jotta operaatioita olisi helpompi ketjuttaa. Muuta syytä esimerkiksi transduktorien T_s ja T_t lopputilojen yhdistämiseen lauseketta $s|t$ vastaavassa transduktorissa ei ole.

5.3.2 Transduktorin muodostaminen lausekkeiden jäsennyksestä

Olisi mahdollista muokata rekisteritransduktorien säännöllisten lausekkeiden kielioppia edelleen siten, että vaihtoehtoiset tulkinnat kullekin välikesymbolille poistuisivat. Nyt esitetyssä muodossa on kuitenkin se hyvä puoli, että kutakin säännöllisten lausekkeiden operaatiota (yhdiste, tulo ja sulkeuma) vastaa yksi välikesymboli ja siten yksi jäsentimen proseduuri ($lausekeD$, $lausekeC$ ja $lausekeB$). Tällöin on mahdollista helposti muodostaa säännöllistä lauseketta vastaava rekisteritransduktori muodostamalla alkeistransduktorit proseduurissa $lausekeA$ ja muokkaamalla proseduureissa $lausekeD$, $lausekeC$ ja $lausekeB$ transduktorien tilasiirtymäverkkoja kuvan 5.2 mukaisesti. Näin säännöllistä lauseketta vastaava rekisteritransduktorin tilasiirtymäverkko saadaan muodostettua suoraan lausekkeen jäsennyksen yhteydessä.

Rekisteritransduktorien kohdalla transduktorien yhdistäminen ei ole yhtä suoraviivaista kuin tavallisilla äärellisillä automaateilla ja transduktoreilla. Ensin on ratkaistava kysymys siitä, mitä tapahtuu, jos yhdistettävissä transduktoreissa on samoihin rekistereihin kohdistuvia rekisterioperaatioita. Cohen-Sygal ja Winter analysoivat tätä ongelmaa artikkelissaan rekisteriautomaattien osalta [8, 3.2]. He esittävät ratkaisuja, joissa joko rekistereitä uudelleen nimeämällä tai uusia rekisterioperaatioita lisäämällä saadaan varmistettua, että yhdistettävien transduktorien rekistereihin jäävät arvot eivät vaikuta muiden osatransduktorien toimintaan.

Tämä ratkaisu ei kuitenkaan ole mielekäs tässä toteutettavassa sovelluksessa. Otetaan esimerkkinä tästä ongelmasta säännöllinen lauseke

[a/b/W,1,1][a/b/R,1,1]

Lauseke on kahden säännöllisen lausekkeen [a/b/W,1,1] ja [a/b/R,1,1] katenaatio. Jos näistä erillisistä lausekkeista muodostettaisiin transduktorit, saataisiin lausekkeesta [a/b/W,1,1] saman relaation toteuttava transduktori kuin lausekkeesta [a/b], sillä kirjoitusoperaatio onnistuu aina. Lauseketta [a/b/R,1,1] vastaava transduktori puolestaan hylkäisi kaikki syötemerkkijonot, koska rekisterin 1 arvo ei voi olla 1 transduktorissa, joka ei sisällä yhtään kirjoitusoperaatiota.

Koska jälkimmäinen transduktori hylkää kaikki syötteet, myös niiden katenaatio hylkää kaikki syötteet. Tämä käyttäytyminen ei kuitenkaan vastaa sitä, mitä säännöllisellä lausekkeella [a/b/W,1,1][a/b/R,1,1] tarkoitetaan. Ei ole mielekästä käsitellä alkeislausekkeiden rekisterioperaatioita toisistaan riippumattomina, sillä tällöin rekisterioperaatioilla ei olisi mahdollista tehdä mitään hyödyllistä. Siispä toteutettavassa ohjelmistossa oletetaan, että kukin samaa rekisterin nimeä käyttävä operaatio kohdistuu aina samaan rekisteriin.

Toiminnallisuus, jonka avulla laajoja transduktoreita olisi mahdollista yhdistää ilman, että yhden transduktorin sisällä tehtävä rekisterioperaatio vaikuttaisi toisen transduktorin toimintaan, olisi myös hyödyllinen. Sitä ei kuitenkaan tämän tutkielman puitteissa käsitellä tämän enempää. Myöhemmin esiteltävissä säännöllisissä lausekkeissa ja niiden osissa käytetään selvyyden vaatiessa lihavoituja kirjaimia **i**, **j** ja **k** viittaamaan "paikallisiin" rekistereihin, jotka konkreettisissa säännöllisissä lausekkeissa on nimettävä niin, ettei kyseisen rekisterin arvoa muuteta muualla lausekkeessa. Lisäksi nämä paikalliset rekisterit alustetaan lausekkeen alussa eksplisiittisesti aloitusarvoonsa, jolloin lauseketta voi käyttää sulkeuman sisällä niin, että rekisteriin lausekkeen suorituksen jälkeen jäänyt arvo ei vaikuta lausekkeen myöhemmillä suorituskerroilla.

5.4 Transduktorin optimointi

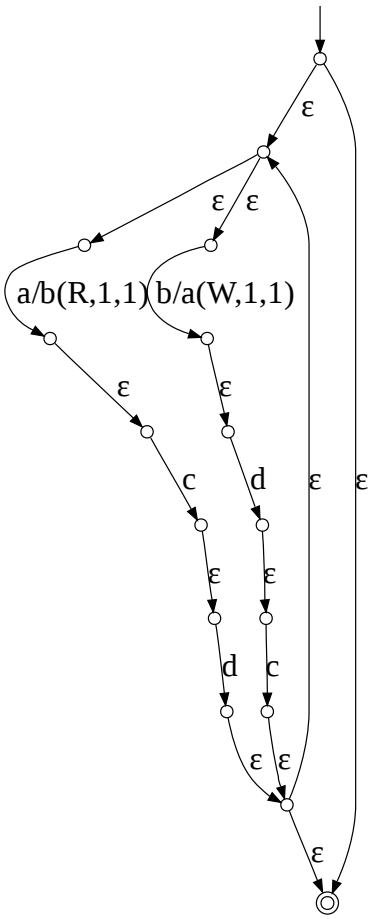
Edellä kuvatulla tavalla muodostettua transduktoria on usein mahdollista optimoida eri tavoin. Kuvasta 5.2 nähdään, että transduktoriin lisätään osatransduktoreja muodostettaessa suuri määrä ϵ-siirtymiä. Jos ϵ-siirtymän kohdetilalle ei ole muita tulevia siirtymiä, voidaan siirtymän lähtö- ja kohdetiloja pitää ekvivalentteina: jos toinen näistä tiloista saavutetaan millä tahansa syötteellä, myös toinen saavutetaan, eikä transduktorin merkkijonoissa tai rekistereissä tapahdu muutosta tilojen välillä siirryttäessä. Tästä syystä tilat voidaan yhdistää, ja ϵ-siirtymä niiden välillä poistaa.

Jos taas ϵ-siirtymän kohdetilalle tulee muitakin siirtymiä, tilaa ei voida poistaa. Tällöin voidaan kuitenkin kopioida kaikki kohdetilalta lähtevät siirtymät lähtemään myös ϵ-siirtymän lähdetilalta, jonka jälkeen ϵ-siirtymä voidaan poistaa. Tämä menettely nopeuttaa transduktorin suoritusta, mutta myös kasvattaa transduktorin kokoa, jos kopioitavia siirtymiä on enemmän kuin yksi.

Toinen helppo optimoinnin mahdollisuus tarjoutuu silloin, kun tilalta lähtee kaksi tai useampia siirtymiä, joiden syötemerkki, tulostemerkki ja rekisterioperaatio ovat samat, ja kohdetilalle ei ole muita tulevia siirtymiä. Tällöin näiden siirtymien kohdetilat voidaan yhdistää, ja siirtymät korvata yhdellä siirtymällä.

Sovellukseen toteutettiin nämä edellä mainitut optimointimenetelmät. Myös luvussa 3.4 esitelty rekisterioperaatioiden optimointimenetelmä olisi ollut toteutettavissa, mutta sitä ei tähän sovellukseen toteutettu.

Jotta optimoinnin vaikutusta transduktorin nopeuden lisäksi sen kokoon voidaan arvioida, sovellus tulostaa transduktorin muodostamisen jälkeen siinä olevien tilojen ja tilasiirtymien lukumäärän. Lisäksi optimoinnin tulosten oikeellisuuden tarkistamiseksi toteutettiin sovellukseen toiminto, jolla tilasiirtymäverkko saa-

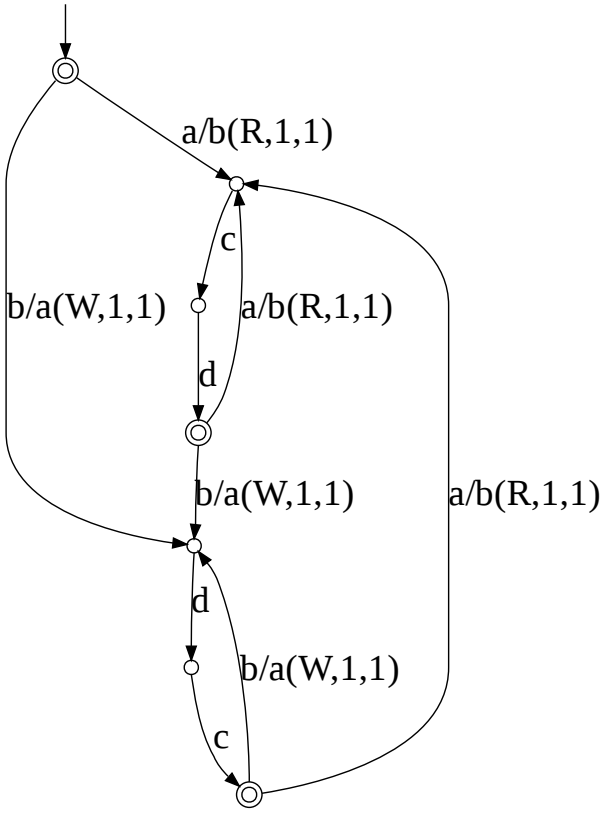


Kuva 5.3: Säännöllistä lauseketta `([a/b/R,1,1]cd|[b/a/W,1,1]dc)*` vastaava optimoimaton transduktorin tilasiirtymäverkko.

daan tulostettua Graphviz-ohjelmapakettiin [10] kuuluvan *dot*-ohjelman käyttämässä muodossa. Dot on erityisesti suunnattujen verkkojen visualisointiin tarkoitettu ohjelma, jonka avulla tilasiirtymäverkko saadaan helposti esitettyä graafisessa muodossa. Kuvassa 5.3 on esimerkin vuoksi esitetty säännöllistä lauseketta

```
([a/b/R,1,1]cd|[b/a/W,1,1]dc)*
```

vastaava optimoimaton transduktorin tilasiirtymäverkko ja kuvassa 5.4 sama tilasiirtymäverkko optimoinnin jälkeen.



Kuva 5.4: Säännöllistä lauseketta `([a/b/R,1,1]cd|[b/a/W,1,1]dc)*` vastaava optimoitu transduktorin tilasiirtymäverkko.

Taulukko 5.1: Sovelluksen luokat ja niiden vastuut.

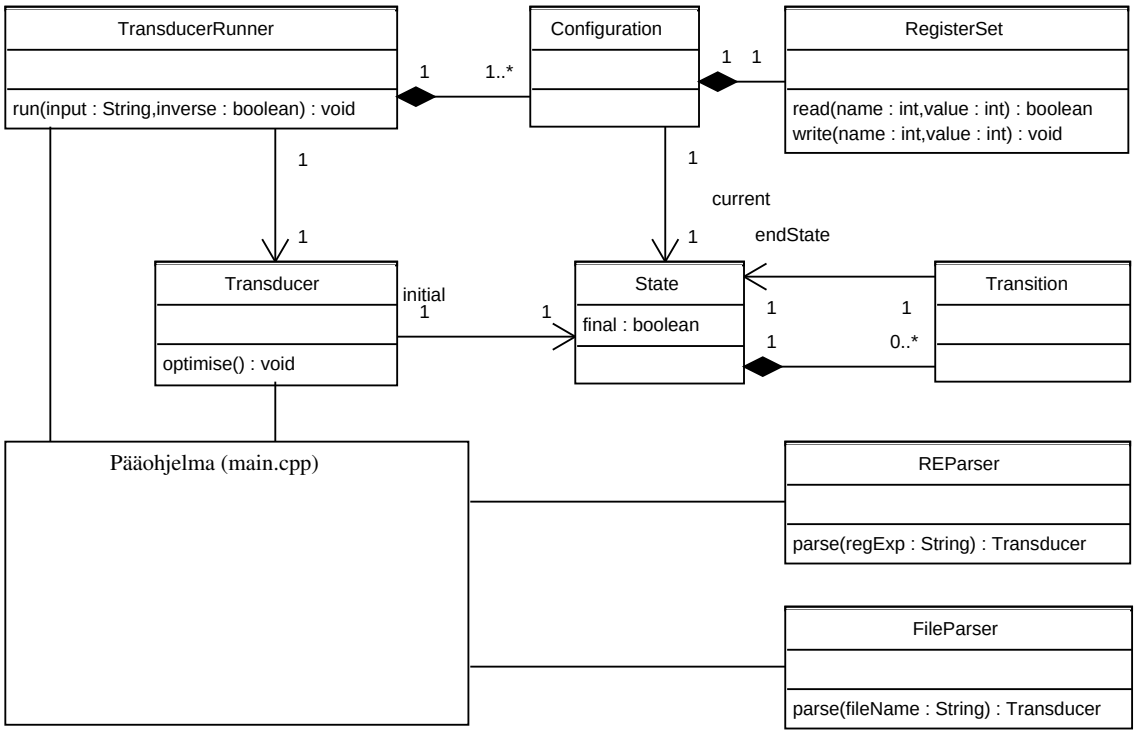
Luokka	**Vastuu**
Configuration	Rekisteritransduktorin tilanne ja uusien tilanteiden suora johtaminen.
FileParser	Rekisteritransduktorin kuvaustiedoston jäsentäminen ja transduktorin muodostamisen ohjaaminen.
RegisterSet	Rekisteritransduktoriin liittyvät rekisterit ja niiden operaatiot.
REParser	Säännöllisten lausekkeiden jäsentäminen ja transduktorin muodostaminen säännöllisestä lausekkeesta.
State	Rekisteritransduktorin tila ja tilalta lähtevien tilasiirtymien hallinta.
Transducer	Rekisteritransduktori ja sen ominaisuudet. Transduktorien yhdistäminen, optimoiminen ja analysointi.
TransducerRunner	Merkkijonomuunnosten tekeminen transduktorin avulla luokkien *Configuration* ja *Transducer* avulla.
Transition	Tilasiirtymä, siirtymän sallittavuuden tarkistaminen, tarvittavien rekisterioperaatioiden suorittaminen sekä syöte- ja tulostemerkkijonojen siirtymissä tapahtuva muokkaaminen.

5.5 Ohjelman rakenne ja toiminta

Edellisissä luvuissa mainittujen rekisteritransduktorien esitysmuotojen testaamiseksi toteutettiin C++-kielinen sovellus. Sovellus suunniteltiin oliokeskeisesti siten, että rekisteritransduktoreihin liittyvät keskeiset käsitteet tulisivat mahdollisimman havainnollisesti esiin sovelluksen luokkajaossa. Niinpä esimerkiksi tiloille, tilasiirtymille, rekistereille ja transduktorin tilanteelle tehtiin omat luokkansa. Sovelluksen suunnittelussa ja toteutuksessa käsitteiden ja algoritmien selkeys asetettiin etusijalle. Suorituskykyä optimoitiin mahdollisuuksien mukaan, muttei kutenkaan sovelluksen rakenteen selkeyden kustannuksella.

Sovelluksen C++-luokat on esitelty taulukossa 5.1. Luokkien lisäksi ohjelma sisältää luokkiin kuulumattomia apufunktioita (tiedostot `util.hpp` ja `util.cpp`) sekä pääohjelman `main.cpp`. Luokat ja niiden väliset suhteet, tärkeimmät attribuutit ja metodit sekä pääohjelman käyttämät luokat on esitetty luokkakaaviona kuvassa 5.5.

Sovelluksessa ei ole interaktiivista käyttöliittymää. Sitä käytetään antamalla käyn-



Kuva 5.5: Sovelluksen luokkakaavio. Kaaviossa on mukana vain luokkien tärkeimmät metodit ja attribuutit.

nistysparametreiksi suoritettava toiminto, käsiteltävän rekisteritransduktorin sisältävän tiedoston nimi ja toimintokohtaisia lisäparametreja. Sovellus tulostaa toimenpiteen tulokset standarditulostusvirtaan ja mahdolliset virheilmoitukset standardivirhevirtaan.

Ohjelman toiminta etenee siten, että käynnistysvaiheessa se lukee luokan `FileParser` avustuksella rekisteritransduktorin määrittelytiedoston ja luo sen avulla uuden `Transducer`-olion. Rekisteritransduktori voidaan määritellä joko kaarilistamuodossa tai säännöllisen lausekkeen avulla. Jälkimmäisessä tapauksessa säännöllinen lauseke jäsennetään luokassa `REParser` toteutetun rekursiivisesti etenevän jäsentimen avulla. Jäsennyksen yhteydessä luodaan alkeislausekkeista erillisiä `Transducer`-olioita, jotka liitetään toisiinsa käyttämällä `Transducer`-luokan metodeja `unionWith` ja `concatenateWith`. Säännöllisen lausekkeen sulkeuma muodostetaan kutsumalla lauseketta vastaavan `Transducer`-olion `closure`-metodia. Näiden metodien toteutukset on esitetty liitteessä A.

Kun rekisteritransduktoria vastaava `Transducer`-olio on saatu muodostettua, sitä voidaan käyttää merkkijonomuunnosten tekemiseen. Tämä tapahtuu `TransducerRunner`-olion avulla. `TransducerRunner` muodostaa rakentimessaan tau-

lukon alustamattomia `Configuration`-olioita. Tämän taulukko on vakiokokoinen, ja sen alkioiden määrä antaa ylärajan transduktorissa kuljettavien polkujen pituudelle.

Varsinainen merkkijonon käsittely tapahtuu `TransducerRunner`-olion `run`-metodissa, jonka lähdekoodi on liitteessä B. Metodi alustaa ensimmäisen `Configuration`-olion vastaamaan transduktorin alkutilannetta. Tämän jälkeen metodi etenee silmukkaan, jossa transduktorin tilasiirtymäverkkoa käydään läpi syvyyshaun järjestyksessä. Kussakin sallitussa siirtymässä alustetaan seuraava `Configuration`-olio vastaamaan transduktorin tilannetta siirtymän kohdetilalla, eli muodostetaan tilanteen suora johdos. Tämä alustus suoritetaan `Configuration`-luokan metodissa `set`.

Sovellusta kehitettäessä havaittiin, että rekursion käyttö tilasiirtymäverkon läpikäynnissä oli huomattavan tehotonta. Tästä syystä `TransducerRunner`-luokan toteutuksessa päädyttiin lopulta käyttämään verkon läpikäyntiin edellä mainittua yhtä silmukkaa ja simuloimaan rekursiota `Configuration`-olioista muodostuvan taulukon ja taulukkoindeksin avulla. Rekursioon perustuvassa ratkaisussa käytettävissä oleva pinomuistin määrä olisi antanut rajan suurimmalle mahdolliselle tilanteiden seuraajapuun syvyydelle. Tässä yhden silmukan ratkaisussa raja voidaan määritellä sovelluksen lähdekoodissa (käytettävissä olevan työmuistin puitteissa).

`TransducerRunner`-luokan `run`-metodille voidaan antaa lisäparametri, joka kertoo, halutaanko suorittaa transduktori vai sen käänteistransduktori. Tätä parametria käytetään sekä `Transition`-luokan tilasiirtymän sallittavuutta testaavassa metodissa että syöte- ja tulostemerkkijonoja muokkaavissa metodeissa ehdollisesti vaihtamaan siirtymien syöte- ja tulostemerkkien roolit.

5.6 Rajoitettu toisto

Määritellään luvussa 2.2 esitettyihin säännöllisten lausekkeiden merkintöihin perustuen uusi merkintä säännöllisen lausekkeen a rajoitetulle toistolle:

$$a^n = \prod_{j=1}^{n} a,$$

jossa $n \geq 1$. Tämä merkintä yleistyy sellaisenaan myös transduktorien säännöllisille lausekkeille. Lausekkeiden lyhentämiseksi otetaan käyttöön merkintä ”laskuritransduktorille”, joka koostuu luku- ja kirjoitusoperaatioiden yhdisteestä. Laskuritransduktori sallii siirtymän, jos rekisterin i arvo on pienempi kuin n, ja samalla

kasvattaa kyseisen rekisterin arvoa yhdellä. Merkitään tätä transduktoria

$$count(i, n) = \cup_{j=1}^{n}[//R, i, j-1][//W, i, j] \tag{5.1}$$

jolloin lausekkeen rajoitettu toisto voidaan esittää muodossa

$$a^n = [//W, i, 0](count(i, n)a)^*[//R, i, n] \tag{5.2}$$

Esimerkiksi transduktori, joka muuttaa merkkijonon "aaa" merkkijonoksi "bbb", eli $[a/b]^3$, voidaan esittää rekisteritransduktorin säännöllisenä lausekkeena joko perinteiseen tapaan ilman rekistereitä muodossa

```
[a/b][a/b][a/b]
```

tai rekistereitä hyödyntäen

```
[//W,1,0](([//R,1,0][//W,1,1]|[//R,1,1][//W,1,2]|
        [//R,1,2][//W,1,3])[a/b])*[//R,1,3]
```

Ensi silmäyksellä näyttäisi siltä, että rekistereitä käyttämällä rajoitettua toistoa sisältävät lausekkeet ovat monimutkaisempia kuin ilman rekistereitä. On tosiaankin totta, että molemmissa muodoissa lausekkeen pituus on lineaarisesti riippuva rajoitetun toiston eksponentista. Rekistereiden käytön etu tuleekin näkyviin vasta silloin, kun toistettava lauseke on hyvin monimutkainen. Tällöin rekistereiden käyttö voi johtaa merkittävästi pienempään transduktoriin, koska toistettavaa lauseketta ei tarvitse toistaa itse transduktorissa.

Jos merkitään lauseketta a vastaavan transduktorin vaatimaa tilaa merkinnällä $S(a)$, niin perinteisille transduktoreille $S_{ei-rek}(a^n) = \mathcal{O}(nS(a))$. Tämä johtuu luvussa 3.1 kuvatusta syystä: transduktoria ei voi toteuttaa muutoin kuin toistamalla lauseketta a vastaavaa transduktoria n kertaa. [1] Mutta rekisteritransduktoreille $S_{rek}(a^n) = \mathcal{O}(n) + S(a)$, mikä voidaan nähdä lausekkeesta 5.2, kun huomataan, että lausekkeen 5.1 mukaisesti lauseke $count(i, n)$ on kooltaan lineaarisesti verrannollinen parametriin n.

[1] Transduktorin minimoinnillakaan ei tähän voida oleellisesti vaikuttaa, koska ainoa tapa lisätä tilainformaatiota tavanomaiseen transduktoriin on lisätä siihen uusia tiloja. Ainoita poikkeustapauksia ovat triviaalit transduktorit, jotka eivät hyväksy yhtään merkkijonoa tai hyväksyvät ja tulostavat tyhjän merkkijonon, sillä nämä eivät muutu lainkaan toistossa.

5.7 Koejärjestely

Jotta edellä esitellyn ohjelman toimintaa voitaisiin kunnolla arvioida, tarvitaan testitapaus, jossa rekisterien käytön vaikutus ohjelman suorituskykyyn tulisi esille. Valitaan kaksi aakkostoa $A = 0, 1, 2, .., 8, 9$ ja $B = a, b, c, .., y, z$. Olkoon $f : A^3 \rightarrow B^3$ jollakin satunnaisuuteen perustuvalla menetelmällä muodostettu kuvaus. Koska joukossa B^3 on enemmän alkioita kuin joukossa A^3, f ei voi olla surjektio. Ei myöskään vaadita sen olevan injektio. Olkoon $T = T(f)$ kuvausta f vastaava transduktori. Tämä transduktori voidaan muodostaa, koska f on kuvauksena äärelliseltä joukolta äärelliselle joukolle säännöllinen. Olkoon edelleen t jokin transduktoria T vastaava säännöllinen lauseke.

Valitaan testattaviksi transduktorit

$$\mathsf{s}_n = \texttt{[//W,1,0]}(\textit{count(1,n)}\texttt{–}\textit{t})^*\texttt{[//R,1,}n\texttt{]}$$

positiivisilla kokonaislukuarvoilla n. Esimerkiksi transduktori s_1 muuntaisi merkkijonon $''-367''$ merkkijonoksi $''-ghn''$ ja transduktori s_2 merkkijonon $''-865-367''$ merkkijonoksi $''-vbp-ghn''$. Kukin kokonaisluvulla indeksoitu transduktori hyväksyy syötteessä ainoastaan vakiomäärän numeroryhmiä. Siispä transduktori s_1 hylkäisi syötteen $''-865-367''$ ja transduktori s_2 hylkäisi syötteen $''-367''$. Myös kuvaus numeroyhdistelmiltä kirjainyhdistelmille on sama kaikissa ryhmissä.

Vertailun vuoksi samat transduktorit toteutettiin myös *Stuttgart Finite State Transducer Tools* -ohjelmiston (SFST) [23] avulla. Tämä Stuttgartin yliopistossa kehitetty transduktoriohjelmisto on saatavilla vapaalla lisenssillä C++-lähdekoodimuodossa. Ohjelmistoa käytettiin testeissä sellaisenaan. Sen lähdekoodiin tehtiin ainoastaan pieniä korjauksia, jotka olivat tarpeen, jotta ohjelmisto saatiin käännettyä GCC-kääntäjän [9] version 4.3.1 avulla. Samaa kääntäjää käytettiin myös rekisteritransduktorisovelluksen kääntämiseen. Molemmat ohjelmistot käännettiin käyttäen samaa GCC:n optimointitasoa -O2.

Transduktorien tehokkuutta testattiin vertaamalla suoritusaikoja, jotka vaadittiin 1 000 000 syötemerkkijonon käsittelyyn. Syötemerkkijonot valittiin siten, että ne sisälsivät suunnilleen yhtä paljon transduktorin hyväksymään kieleen kuuluvia ja siihen kuulumattomia merkkijonoja. Käytännössä syötemerkkijonojen valinta tapahtui siten, että muodostettiin ensin lista 1 000 000 hyväksyttävästä merkkijonosta, ja tätä listaa muokattiin satunnaisesti siten, että yksittäisiä merkkejä korvattiin toisilla merkeillä.

Kukin koe toistettiin kolme kertaa 1,8 GHz:n AMD Athlon 64 -suorittimella varustetussa tietokoneessa, jota ei muutoin kuormitettu kokeiden aikana. Suoritusker-

roista kirjattiin niiden keskimääräiset suoritusajat sekä optimoitujen transduktorien tilojen ja tilasiirtymien lukumäärät.

Tilojen ja tilasiirtymien lukumäärää käytettiin transduktorin muistin käytön arvioimiseen arvioimalla ohjelmiston `State`- ja `Transition`-luokkien rakenteen perusteella, että kohtuullisen optimaalisessa tilasiirtymäverkon toteutuksessa yhden tilan tallentamiseen vaadittaisiin 2 tavua ja tilasiirtymän tallentamiseen 7 tavua muistitilaa. SFST:n tilasiirtymäverkon kokoarvioksi otettiin suoraan ohjelmiston levylle kirjoittaman optimoidun transduktoritiedoston koko.

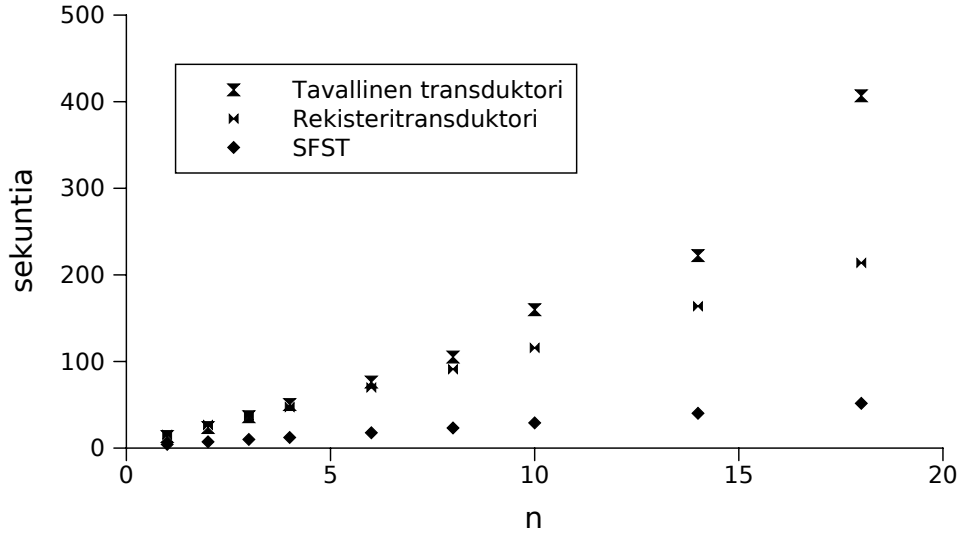
5.8 Testituloksia

Luvussa 5.7 kuvattua koejärjestelyä noudattaen mitattiin transduktorien suoritusaikaa ja muistin käyttöä. Suoritusajat on esitetty kuvassa 5.6 ja muistin käyttö kuvassa 5.7.

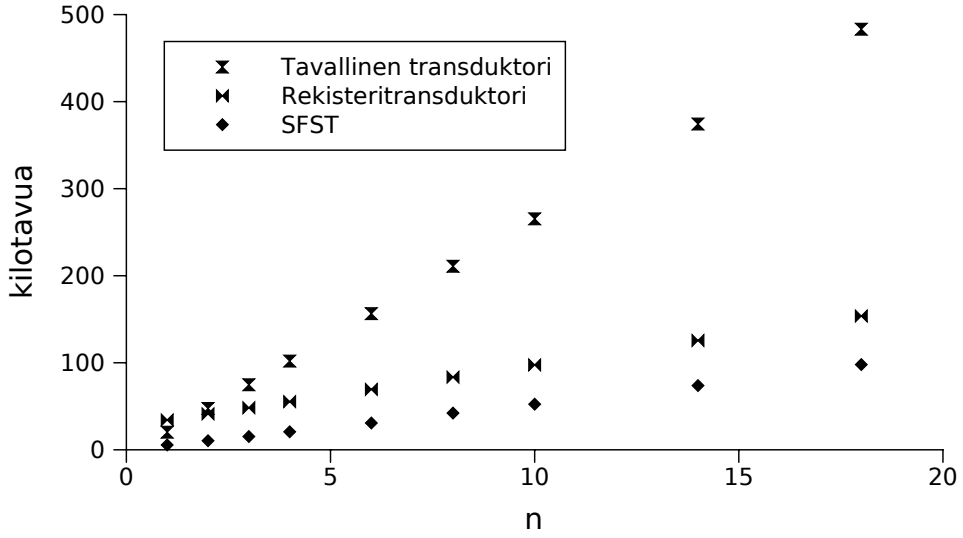
Kuvista havaitaan, että kaikilla transduktorityypeillä sekä suoritusaika että transduktorin koko kasvoivat likimain lineaarisesti transduktorin hyväksymien syötemerkkijonojen pituuden eli toistettavien jaksojen lukumäärän funktiona. Suoritusajoissa SFST oli nopein. Rekisteritransduktorin vaatima suoritusaika oli tasaisesti noin nelinkertainen siihen verrattuna. Tavallinen transduktori oli rekisteritransduktoria hiukan nopeampi lyhyillä merkkijonoilla ($n \leq 3$), mutta pitkillä merkkijonoilla tavallisten transduktorien suoritusaika kasvoi hiukan nopeammin. Erityisesti pisimmillä testatuilla merkkijonoilla ($n = 18$) tavallisen transduktorin suoritusaika poikkesi aikaisemmasta lineaarisesta käyttäytymisestä. Odotettua pidempi suoritusaika saattoi tässä tapauksessa johtua siitä, että transduktori ei enää kokonaisuudessaan mahtunut suorittimen 512 kilotavun käteismuistiin, jolloin tiedon siirto keskusmuistista käteismuistiin on vienyt suhteellisesti enemmän aikaa kuin muissa koetilanteissa.

Muistin käytön suhteen kaikkien transduktorityyppien koko kasvoi hyvin tarkasti lineaarisesti hyväksyttävien syötemerkkijonojen pituuden funktiona. Tavallisen transduktorin ja SFST:n tapauksessa koko oli käytännössä suoraan verrannollinen syötemerkkijonossa hyväksyttävien jaksojen lukumäärään. Tämä oli odotettu tulos, sillä ilman rekistereitä kutakin merkkijonon jaksoa täytyy tilasiirtymäverkossa vastata yksi jakson tunnistava osatransduktori. Mutta myös rekisteritransduktorin koko kasvoi selvästi jaksojen määrän funktiona.

Taulukossa 5.2 on esitetty tavallisen transduktorin ja rekisteritransduktorin tilojen ja tilasiirtymien määrät kahdella eri hyväksyttävien jaksojen lukumäärällä optimoidun ja optimoimattoman tilasiirtymäverkon tapauksessa. Tavallisessa transduk-



Kuva 5.6: Ohjelmistojen suoritusajat luvun 5.7 koejärjestelyssä tavallisella transduktorilla, rekisteritransduktorilla ja SFST:n transduktorilla. n on toistettavien jaksojen määrä transduktorin hyväksymissä syötteissä.



Kuva 5.7: Transduktorin muistin käyttö (tilasiirtymäverkon koko) luvun 5.7 koejärjestelyssä tavallisella transduktorilla, rekisteritransduktorilla ja SFST:n transduktorilla. n on toistettavien jaksojen määrä transduktorin hyväksymissä syötteissä.

Taulukko 5.2: Tavallisen transduktorin ja rekisteritransduktorin tilojen ja tilasiirtymien määrät luvun 5.7 kokeessa. n on hyväksytyissä merkkijonoissa olevien jaksojen lukumäärä. Tulokset on esitetty sekä luvun 5.4 mukaisesti optimoiduille että optimoimattomille transduktoreille.

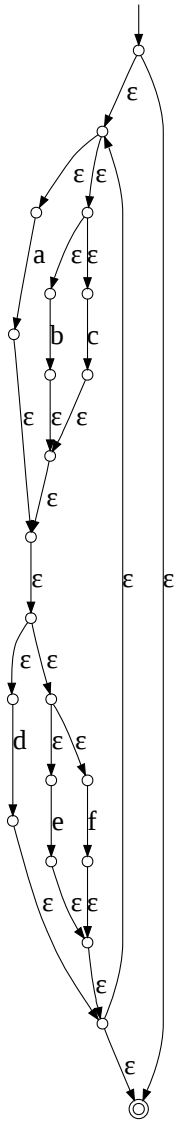
		Optimoimaton		Optimoitu	
		n = 1	n = 18	n = 1	n = 18
Transduktori	Tilat	6000	108000	2250	40483
	Siirtymät	6998	125981	2249	57465
Rekisteritransduktori	Tilat	6010	6112	2254	2288
	Siirtymät	7010	7129	4253	21304

torissa sekä tilojen että tilasiirtymien määrä kasvaa merkittävästi toistojen määrän kasvaessa, eikä optimointi auta tähän.

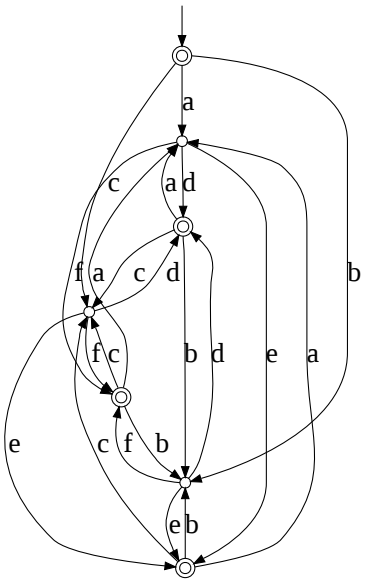
Optimoimattomassa rekisteritransduktorissa toistojen määrän lisääminen ei juurikaan lisää tilojen tai tilasiirtymien määrää. Tilojen määrä ei myöskään kasva optimoidussa rekisteritransduktorissa, mutta siinä tilasiirtymien määrä on lähes kolminkertaistunut siirryttäessä yhdestä toistosta 18 toistoon. Tämä siirtymien määrän kasvu selittää myös sen, miksi rekisteritransduktorin muistin tarve kasvaa selvästi toistojen määrän funktiona.

Jotta rekistereitä voisi kunnolla käyttää hyväksi muistitilan säästämisessä, täytyy luvun 5.4 optimointimenetelmää muuttaa. Tämä voidaan tehdä siten, että ϵ-siirtymiä ei poisteta niissä tapauksissa, joissa poistaminen johtaisi siirtymien kokonaismäärän kasvuun. Kuvat 5.8, 5.9 ja 5.10 havainnollistavat muutoksen vaikutusta optimoinnin lopputulokseen säännöllisellä lausekkeella `((a|b|c)(d|e|f))*`. Kuvassa 5.8 on optimoimaton transduktori (tai tässä erikoistapauksessa äärellinen automaatti) muodossa, joka saadaan luvun 5.3.2 menetelmällä. Kuvassa 5.9 tämä transduktori on optimoitu siten, että kaikki ϵ-siirtymät on poistettu. Optimoinnin tuloksena tilojen määrä on vähentynyt, mutta tilasiirtymien määrä on pysynyt melko suurena. Kuvassa 5.10 transduktoriin on jätetty ne ϵ-siirtymät, joiden kohdetilalle tulee ja jolta lähtee vähintään kaksi siirtymää. Tässä transduktorissa siirtymien määrä on pienin, mutta tiloja kaksi enemmän kuin kuvan 5.9 transduktorissa.

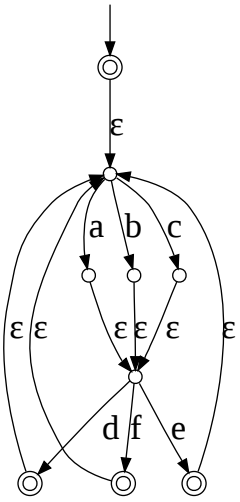
On syytä huomata, että tämänkään muutoksen jälkeen transduktori ei vielä ole kokonsa puolesta optimaalinen. Kuvan 5.10 transduktorista olisi vielä mahdollista poistaa sen kaikki lopputilat muuttamalla lopputiloilta lähtevien siirtymien yhteinen kohdetila lopputilaksi. Tällä tavalla transduktorista poistuisi neljä tilaa ja neljä ϵ-siirtymää.



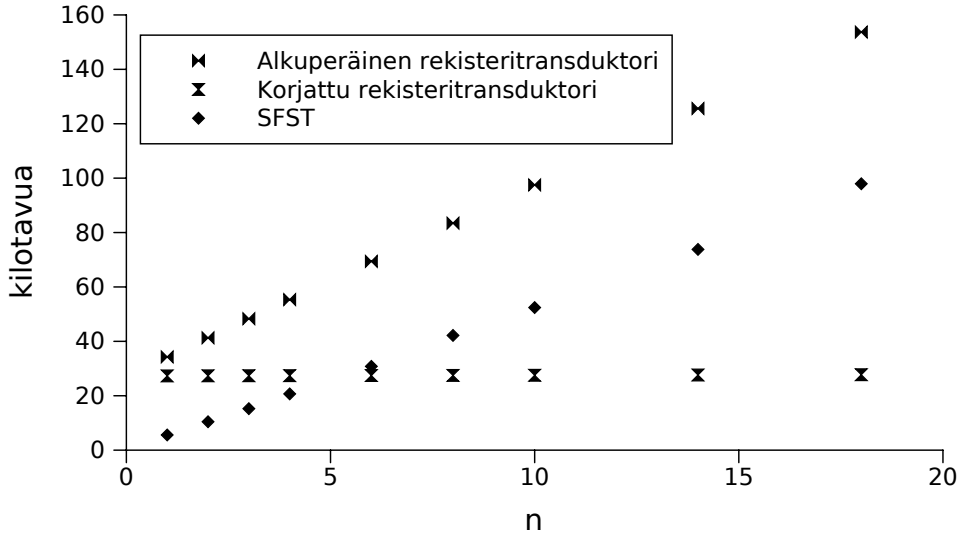
Kuva 5.8: Säännöllistä lauseketta `((a|b|c)(d|e|f))*` vastaava optimoimaton transduktori, jossa on 22 tilaa ja 27 tilasiirtymää.



Kuva 5.9: Säännöllistä lauseketta `((a|b|c)(d|e|f))*` vastaava optimoitu transduktori, josta kaikki ϵ-siirtymät on poistettu. Transduktorissa on 7 tilaa ja 21 tilasiirtymää.



Kuva 5.10: Säännöllistä lauseketta `((a|b|c)(d|e|f))*` vastaava optimoitu transduktori, josta on poistettu vain ne ϵ-siirtymät, joiden poistaminen ei kasvata siirtymien kokonaismäärää. Transduktorissa on 9 tilaa ja 13 tilasiirtymää.



Kuva 5.11: Transduktorin muistin käyttö (tilasiirtymäverkon koko) luvun 5.7 koejärjestelyssä alkuperäisellä ϵ-vapaalla rekisteritransduktorilla, optimointimenetelmään tehdyn korjauksen jälkeisellä rekisteritransduktorilla ja SFST:n transduktorilla. n on toistettavien jaksojen määrä transduktorin hyväksymissä syötteissä.

Optimointimenetelmän korjauksen jälkeen luvun 5.7 koe toistettiin rekisteritransduktoreille uudelleen. Rekisteritransduktorin suoritusajat pysyivät mittaustarkkuuden rajoissa samoina kuin alkuperäisellä rekisteritransduktorilla kuvassa 5.6. Muistin käytössä sen sijaan tapahtui merkittävää parannusta. Kuvassa 5.11 on verrattu alkuperäisen ja korjatun rekisteritransduktorin muistin käyttöä SFST:n muistin käyttöön. Korjatun rekisteritransduktorin muistin käyttö pysyy lähes vakiona merkkijonon pituudesta riippumatta, joten riittävän pitkillä merkkijonoilla se toimii myös SFST:tä vähemmällä muistilla.

6 Johtopäätökset

Äärelliset transduktorit ovat äärellisten automaattien laajennus, jota voidaan käyttää muunnettaessa merkkijonoja kahden säännöllisen kielen välillä. Tutkielmassa esiteltiin transduktorin rakenne, transduktoreille tehtävät tavallisimmat muunnosoperaatio ja niiden optimointiin käytettyjä menetelmiä.

Transduktorit havaittiin erittäin käyttökelpoisiksi erityisesti kieliteknologian sovelluksissa, joissa käsiteltävissä merkkijonoissa ei esiintynyt riippuvuuksia kaukana toisistaan olevien merkkijonon osien välillä. Tärkeimpiä näistä sovelluksista on sanojen morfologinen analyysi useimmissa luonnollisissa kielissä. Tutkielmassa osoitettiin, että jos muunnettavissa merkkijonoissa esiintyy pitkän välimatkan riippuvuuksia, äärellisen transduktorin tilasiirtymäverkon koko voi kasvaa rajusti. Tällaisia pitkän välimatkan riippuvuuksia ovat esimerkiksi riippuvuuden merkkijonoon liitettävien etu- ja jälkiliitteiden välillä.

Tutkielmassa pohdittiin ratkaisuja edellä mainittujen riippuvuuksien käsittelemiseksi tavoilla, jotka eivät johtaisi transduktorin koon merkittävään kasvuun. Erityisen tarkastelun kohteeksi otettiin rekisteritransduktorit, joissa perinteisistä transduktoreista poiketen on käytettävissä rajoitettu määrä apumuistia. Tämä apumuisti esitetään rekistereinä, joihin voidaan kirjoittaa ja joiden arvoja voidaan lukea transduktorin tilasiirtymiin liitettävien rekisterioperaatioiden avulla.

Rekisteritransduktorien toimivuuden arvioimiseksi tutkielmassa kehitettiin sovellus, jonka avulla niitä voitiin muodostaa, havainnollistaa ja käyttää merkkijonojen muuntamiseen. Sovelluksen tarpeita varten kehitettiin notaatio rekisteritransduktorien esittämiseksi säännöllisten lausekkeiden avulla yhdistämällä ja laajentamalla toisaalta tavanomaisille transduktoreille ja toisaalta rekisteriautomaateille kehitettyjä säännöllisten lausekkeiden esitystapoja.

Sovelluksen toimivuutta testattiin rajoitettua toistoa käsittelevällä tehtävällä, jossa samaa äärellisten kielten välistä muunnostransduktoria katenoitiin itsensä kanssa siten, että transduktori sai hyväksyä vain määrätyn mittaisia merkkijonoja. Sovelluksen nopeutta sekä muodostetun transduktorin kokoa verrattiin toisen transduktoriohjelmiston (SFST) toimintaan.

Testeissä havaittiin, että rekisteritransduktorin optimointivaiheessa ϵ-siirtymien poistoon on kiinnitettävä erityistä huomiota. Täydellinen ϵ-siirtymien poisto rekisteritransduktorista johti transduktorin koon merkittävään kasvuun sallittujen tois-

tojen määrän funktiona, jolloin rekisterien käytöstä saatava hyöty suureksi osaksi menetettiin. Paras tulos saatiin jättämällä transduktoriin joitakin ϵ-siirtymiä. Tällä tavoin muodostettu rekisteritransduktori oli suurilla toistojen määrällä muistin käytön kannalta SFST-ohjelmistoa taloudellisempi.

Suoritusajoissa SFST oli rekisteritransduktoria nopeampi, mutta suhteellinen nopeusero ei muuttunut syötteen pituuden mukana. On mahdollista, että nykyistä toteutusta optimoimalla rekisteritransduktorien suoritusnopeus saataisiin SFST:n tasolle.

Tehdyt kokeet edustivat teoreettista esimerkkitapausta, jossa rekisterien käytön tiedetään tehostavan transduktorin muistin käyttöä selkeästi. Kokeiden tuloksista tai aikaisemmista rekisteritransduktoreja käsittelevästä kirjallisuudesta ei kuitenkaan voida suoraan päätellä, olisiko rekisteritransduktorien käytöstä merkittävää apua vaikkapa suomen kielen morfologisen analysaattorin toteutuksessa. Vastauksen saaminen tähän kysymykseen vaatisi lisää kokeiluja esimerkiksi suomen kielen yhdyssanoja ja johdoksia käsittelevällä transduktorilla, sillä näissä esiintyy jonkin verran riippuvuuksia sanassa kaukana toisistaan olevien osien välillä.

Rekisteritransduktorien esittämiseen käytettäviä säännöllisiä lausekkeita tulisi myös kehittää paremmin todellisia sovelluksia tukeviksi. Ainakin rekistereiden nimeämiseen liittyvät ongelmat olisi pyrittävä ratkaisemaan siten, että lausekkeita yhdisteltäessä olisi mahdollista valita, nimetäänkö rekisteri automaattisesti uudelleen, mikäli samaa rekisterin nimeä on käytetty useammassa kuin yhdessä osassa. Kieliteknologisia sovelluksia varten olisi lausekkeiden notaatioon hyvä lisätä muissa nykyään saatavilla olevissa transduktoriohjelmistoissa käytössä olevia lyhennysmerkintöjä.

7 Lähteet

[1] Jan W. Amtrup, *Morphology in Machine Translation Systems: Efficient Integration of Finite State Transducers and Feature Structure Descriptions*, Machine Translation, vol 18, no. 3 s. 217 – 238, 2003.

[2] Kenneth R. Beesley, *Constraining separated morphotactic dependencies in finite-state grammars*, Proceedings of the International Workshop on Finite-State Methods in Natural Language Processing, 1998.

[3] Kenneth R. Beesley, Lauri Karttunen, *Finite-state non-concatenative morphotactics*, Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, Hong Kong, 2000.

[4] Kenneth R. Beesley, Lauri Karttunen, *Finite State Morphology*, CSLI Publications, Stanford, USA, 2003.

[5] Jean Berstel, *Transductions and Context-Free Languages*, saatavilla PDF-muodossa <URL: `http://www-igm.univ-mlv.fr/~berstel/LivreTransductions/LivreTransductions.pdf`>, 19.2.2007. Alkuperäinen, samanniminen teos painettu 1973 (Teubner Verlag).

[6] Noam Chomsky, *Three models for the description of language*, IEEE transactions on information theory, vol. IT-2, s. 113 – 124, 1956.

[7] Noam Chomsky, *Knowledge of Language*, Praeger, New York, 1986.

[8] Yael Cohen-Sygal, Shuly Wintner, *Finite-State Registered Automata for Non-Concatenative Morphology*, Computational Linguistics vol. 32, Issue 1, s. 49 – 82, 2006.

[9] *The GNU Compiler Collection (GCC)*. <URL: `http://gcc.gnu.org`>, viitattu 15.6.2008.

[10] *Graphviz - Graph Visualization Software*. <URL: `http://www.graphviz.org`>, viitattu 29.6.2008.

[11] Dick Grune, Ceriel J. H. Jacobs, *Parsing Techniques - a Practical Guide*, 2nd edition, Springer, USA, 2008.

[12] John E. Hopcroft, Jeffrey D. Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley, Reading, USA, 1979.

[13] Chun-Nan Hsu, Ming-Tzung Dung, *Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web*, Information Systems, Vol. 23, No. 8, 1998.

[14] Juhani Karhumäki, *Sanat ja automaatit*, Turku Centre for Computer Science National Publication No 3, Turku, 2003.

[15] Lauri Karttunen, *Applications of Finite-State Transducers in Natural Language Processing*, Lecture Notes in Computer Science, 2088, 2001.

[16] Kimmo Koskenniemi, *Two-level morphology: A general computational model for word-form recognition and production* (väitöskirja), Publications of the Department of General Linguistics, University of Helsinki, 1983.

[17] M. V. Lawson, *Finite automata*, saatavilla PDF-muodossa <URL: `http://www.ma.hw.ac.uk/~markl/preprints/Lawson.pdf`>, viitattu 3.11.2007.

[18] Mehryar Mohri, *Minimization of sequential transducers*, Lecture Notes in Computer Science, 807, 1994.

[19] Mehryar Mohri, *Finite-State Transducers in Language and Speech Processing*, Computational Linguistics, vol. 23, Issue 2, s. 269 – 311, 1997.

[20] Panu-Kristian Poiksalo, *Digitaalitekniikan perusteet*, Tampereen teknillinen yliopisto, Vantaa, 2005.

[21] Emmanuel Roche, Yves Schabes, *Finite-State Language Processing*, MIT Press, USA, 1997.

[22] G. Rozenberg, A. Salomaa (toim.), *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*, Springer, Berlin, 1997.

[23] Helmut Schmid, *Stuttgart Finite State Transducer Tools*, versio 1.2. Ohjelmisto saatavilla lähdekoodimuodossa <URL: `http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/SFST.html`>, viitattu 15.6.2008.

[24] Bruce W. Watson, *A Taxonomy of Finite Automata Minimization Algorithms*, Computing Science Note 93/44, Eindhoven University of Technology, Hollanti, 1994. <URL: ftp://ftp.win.tue.nl/pub/techreports/pi/automata/taxonomy/2nd.edition/mintax.ps.gz>, viitattu 12.6.2008.

[25] Shuly Wintner, *Finite-State Technology as a Programming Environment*, Lecture Notes in Computer Science, 4394, 2007.

[26] <URL: http://www.xrce.xerox.com/competencies/content-analysis/fst/>, viitattu 22.1.2008.

A Rekisteritransduktorien rakentaminen osatransduktoreista

`Transducer`-luokan metodien `unionWith` ja `concatenateWith` avulla muodostetaan kahden transduktorin yhdiste ja katenaatio. Metodi `makeClosure` muuttaa transduktorin sulkeumakseen, eli jos transduktoria T vastaa alun perin säännöllinen lauseke r, niin kutsun `T.makeClosure()` jälkeen sitä vastaa säännöllinen lauseke r^*.

Alla oleva lähdekoodi sisältää toteutukset näille transduktorien rakentamisessa käytettäville kolmelle metodille. Muotoa `Transition(`*`endState`*`, 0, 0)` olevalla konstruktorilla luodaan uusi ϵ-siirtymä, jonka kohdetila on *endState*.

```
/**
 * Adds the states and transitions from trasducer t to
 * this transducer to form an union of transducers.
 * Deletes transducer t.
 */
void Transducer::unionWith(Transducer * t) {
  State * initial = new State(false);
  initial->addTransition(Transition(this->initialState, 0, 0));
  initial->addTransition(Transition(t->initialState, 0, 0));
  this->initialState = initial;

  State * final = new State(true);
  this->finalState->addTransition(Transition(final, 0, 0));
  this->finalState->setFinal(false);
  t->finalState->addTransition(Transition(final, 0, 0));
  t->finalState->setFinal(false);
  this->finalState = final;

  if (t->registerCount > registerCount)
    registerCount = t->registerCount;
  if (t->maxRegisterValue > maxRegisterValue)
    maxRegisterValue = t->maxRegisterValue;
  delete t;
}
```

```
/**
 * Concatenates this transducer with transducer t.
 * States and transitions from t will be merged to this
 * transducer. Deletes transducer t.
 */
void Transducer::concatenateWith(Transducer * t) {
  this->finalState->addTransition(Transition(t->initialState, 0, 0));
  this->finalState->setFinal(false);
  this->finalState = t->finalState;

  if (t->registerCount > registerCount)
    registerCount = t->registerCount;
  if (t->maxRegisterValue > maxRegisterValue)
    maxRegisterValue = t->maxRegisterValue;
  delete t;
}

/**
 * Turns this transducer into a closure of itself.
 */
void Transducer::makeClosure() {
  State * initial = new State(false);
  State * final = new State(true);
  initial->addTransition(Transition(this->initialState, 0, 0));
  initial->addTransition(Transition(final, 0, 0));
  this->finalState->addTransition(Transition(this->initialState, 0, 0));
  this->finalState->addTransition(Transition(final, 0, 0));
  this->finalState->setFinal(false);
  this->initialState = initial;
  this->finalState = final;
}
```

B Merkkijonon muuntaminen rekisteritransduktorissa

Alla on lähdekoodi luokan `TransducerRunner` metodille `run`. Se tulostaa standarditulostusvirtaan syötemerkkijonoa `inputString` vastaavat tulostemerkkijonot. Metodissa esiintyvä muuttuja `transducer` on luokan attribuutti, joka viittaa transduktoriin, jota merkkijonon muuntamiseen käytetään. Muuttuja `configs` on vakiokokoinen taulukko `Configuration`-olioita. Sitä käytetään metodissa pinomuistina, johon talletetaan transduktorin tilanteet alkutilasta nykyiselle tilalle johtavan polun varrelta.

```
/**
 * Run the transducer with given input. Prints all valid output
 * strings to stdout. If isInverse is true, run the inverse
 * transducer.
 */
void TransducerRunner::run(string &inputString, bool isInverse) {
  // Depth of the current path from initial state
  int depth = 0;
  // Pointer to the start of input string
  const char * inputStart = inputString.c_str();
  // Pointer to the position after last character in input string
  const char * inputEnd = inputStart + inputString.length();
  // Current output string
  string output("");
  // Length of the current output string
  size_t outputLength = 0;

  // Set the initial configuration
  configs[0].setFirst(transducer, inputStart);

  // Main loop
  while (true) {
    // Check if we have processed all transitions from current state
    if (configs[depth].nextTransition == configs[depth].transitionEnd) {
      if (depth == 0) return; // All paths have been processed
```

```
      // Move back to the previous state and try the next transition
      if (configs[depth--].lastOutput) output.erase(--outputLength);
      configs[depth].nextTransition++;
      continue;
    }

    // Check if current transition is allowed. If not, continue
    // with next transition.
    char nextIn = '\0';
    if (configs[depth].input != inputEnd)
      nextIn = configs[depth].input[0];
    if (!configs[depth].nextTransition->allowInput(nextIn,
        configs[depth].registers, isInverse)) {
      configs[depth].nextTransition++;
      continue;
    }

    // We have found an allowed transition. Follow it and create a
    // new configuration on the stack.
    if (depth >= MAX_PATH_LENGTH - 1) {
      cerr << "Maximum path depth reached!" << endl;
      return;
    }
    configs[depth+1].set(configs+depth, &*configs[depth].nextTransition,
                         isInverse);
    if (configs[++depth].lastOutput) {
      output += configs[depth].lastOutput;
      outputLength++;
    }

    // If the newly created configuration is in a final state and
    // there is no input left, print current output.
    if (configs[depth].state->isFinal() &&
        configs[depth].input == inputEnd) {
      cout << output << endl;
    }

  }
}
```