БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ

Минимални ациклични автомати: конструкции, алгоритми, приложения

Стоян Милков Михов

ДИСЕРТАЦИЯ

за присъждане на образователна и научна степен

Доктор

НАУЧЕН КОНСУЛТАНТ: проф. д.м.н. Димитър Скордев

София, 1999

Съдържание

Увод

Актуалност на проблема

Крайните автомати имат важни приложения в много области на информатиката, включващи бази данни, компилаторни технологии, обработка на естествен език и други. Напоследък се наблюдава значително увеличаване на интереса към приложенията на крайните автомати в компютърната лингвистика. В съвременната компютърна лингвистика се налага обработване на огромни корпуси от текстове, съдържащи десетки или стотици милиони думи, което прави неприложимо използването на не много ефективни методи. Тъй като крайните автомати притежават някои съществени предимства – скорост и компактност – те в момента са най-успешната технология за лингвистична обработка на текстове.

Детерминираните ациклични крайни автомати намират широко приложение при решаването на различни задачи, свързани с обработката на естествен език. Ацикличният краен автомат представя крайно множество от думи, а при различните модификации на крайните автомати – крайно множество от думи с анотации към тях, крайно множество от думи с техни индекси, крайно множество от двойки думи и т.н. Известно е, че за всяко регулярно множество от думи съществува единствен (с точност до изоморфизъм) минимален краен автомат, който го описва. Намирането на минималния автомат е съществено за приложенията поради компактността му. Съществуват различни методи за построяването на минимален автомат, но при стандартните методи, в процеса на получаването, е необходимо да се строят междинни автомати, които са огромни по размер в сравнение с резултатния минимален автомат. Това обяснява интереса към разработването на ефективни методи и алгоритми за директното построяване на минимален ацикличен краен автомат.

Цел на дисертацията

Настоящата работа има за цел да представи формалните математически концепции и методи, разработени от автора, за директно построяване на минимални ациклични крайни автомати, както и някои техни приложения.

Математическото описание целѝ формалното изследване на новите концепции и методи както в класическия случай на крайни автомати, така и при техни различни модификации като крайни автомати с етикети на крайните състояния, моноидни преобразуватели, връщащи индекс на думите, и n-лентови крайни автомати. Представянето се съпровожда с математически доказателства за коректност на новите методи.

Важна част от настоящата работа е формалното описание чрез псевдо код на алгоритмите за построяване на минимални ациклични крайни автомати, които алгоритми се базират на методите, представени в математичекото описание. Също така се излагат доказателства за коректност на алгоритмите и оценки на тяхната алгоритмична сложност.

Друг акцент на работата е представянето на някои приложения на разработените алгоритми в областта на компютърната лингвистика. Специално внимание се обръща върху реално действащи компютърни системи, в които тези алгоритми са реализирани, като при това се правят експериментални сравнения с реализации на други методи.

Обосновка и състояние на изследванията по проблема

Теория на крайните автомати и преобразуватели

Поради широкото приложение на крайните автомати, тяхната теория е сравнително добре изучена. Въвеждане в теорията на крайните автомати може да се намери в [HU79, ДПД84, Br84, Ma96]. Крайните автомат се дефинират елегантно и притежават нетривиални и интересни свойства [Sa81]. Еквивалентността им с регулярните езици, която за първи път се доказва от Клини, има широко приложение. От практическа гледна точка е удобно езиците да се описват чрез регулярни изрази (и техни разширения), а конкретното представяне в компютрите да се осъществява чрез крайни автомати поради голямата им ефективност. Преглед на съвременните методи и алгоритми за конструиране на краен автомат от регулярен израз може да се намери в [Wa93a, Wa95].

Изследват се също така различни модификации на крайните автомати с интересни допълнителни свойства. В [HU79, Br84] са разгледани някои от разработените по-рано разновидности като например: двупосочен краен автомат, автоматът на Мили, автоматът на Мур, частичният автомат на Мили, автоматът на Рабин-Скот, двулентовият автомат на Елго-Мезей, двулентовият автомат на Елго-Мезей-Шефердсън и други.

Друга интересна разновидност на крайните автомати са крайните автомати с етикети на крайните състояния. Грубо казано, обработката на дадена дума в този случай дава информация не само за това, дали думата е разпозната от автомата, или не е, но при разпознаване думата се класифицира, като се извежда допълнителен етикет. Изследване и приложения на тези автомати за описание на речникови знания за естествен език са представени в [Re91].

В [Бу78, Br84, Ми93] се изследват крайни преобразуватели, при които се допуска езикът от втората лента да бъде от произволен моноид, а вместо конкатенация се използва съответната моноидна операция. Показват се различни математически свойства на тези обекти, теорема на Клини за тях и други. Моноидният преобразувател, на който изходният моноид е моноидът $\mathbb{N}$, се използва за получаване на индекс на думата в лексикографски подреден списък, описван от ацикличен автомат. Подобно описание може да се намери в [Ro92].

Двулентовите автомати са една често използвана разновидност на крайните автомати. Те се наричат крайни преобразуватели поради свойството си да служат за превод от един регулярен език на друг. Тъй като лентите са две, множеството, което описват, не е формален език от думи, а релация между двойка думи. Съществуването на модификация на теоремата на Клини за крайни преобразуватели оправдава назоваването на множествата, които се описват от преобразувателите като регулярни. Задълбочено изследване по този въпрос се предлага в [KK94]. Там се обръща внимание върху моделирането на трансформационни правила чрез преобразуватели и се прави изследване на математическите свойства на преобразувателите. Така например се показва, че сечение на регулярни релации не винаги е регулярна релация. Изследва се интересният клас от регулярни релации между думи с еднаква дължина, за които горното свойство е в сила. Сечението на регулярни релации е необходимо за ефективната реализация на паралелни трансформационни правила.

Един от основните практически проблеми, които възникват при използването на автомати, е, че често се получават огромни по размери автомати. Поради това е целесъобразно да се работи с минимални автомати. В [Wa93b] са представени най-добрите съвременни методи за минимизиране на автомати. Трябва да отбележим, че най-ефективен се оказва методът на Хопкрофт [Ho71] със сложност $O(n \log(n))$. В случаите на ациклични автомати се получават по-добри резултати, отколкото в общия случай. В [Re91, Re92, Ro92] е показан линейният алгоритъм на Ревуз за минимизиране на ациклични крайни автомати, който е значително по-ефективен от класическите по-общи алгоритми за минимизиране. В практиката най-често се налага използването на ациклични автомати за разпознаване (или класифициране в случай на автомат с етикети) на даден списък от думи. Поради това е от особен интерес задачата за построяване на минимален автомат по даден списък. По този повод в дисертацията на Ревуз [Re91] четем:

> "Il n'existe pas d'algorithme rapide pour construire directement l'automate déterministe minimal d'un langage fini donné, à partir du langage sous forme de liste. Il est nécessaire de procéder en deux étapes. Dans une première étape un automate qui reconnaît le langage de la liste est fabriqué. Puis dans une seconde étape la minimisation de cet automate est effectuée." [1]

[1] Буквален превод на български: „Не съществува бърз алгоритъм за директно построяване на минимален детерминиран краен автомат за даден краен език, като се тръгне от език под формата на списък. Необходимо е да се процедира на два етапа. При първия етап се създава автомат, разпознаващ езика от списъка. При втория етап този автомат се минимизира."

Нека първо скицираме накратко алгоритъма на Ревуз. Алгоритъмът по време на първия етап построява от дадения списък детерминиран краен автомат, разпознаващ този списък. Във варианта, представен в [Ro92], този първоначален автомат представлява дърво и се строи тривиално. Вторият етап се състои в минимизирането на дървото, като се слепват еквивалентните състояния. За да се извърши ефективно тази процедура, първо на всяко състояние се изчислява височината на поддървото с начало това състояние. Слепване се извършва само на състояния, които са с равна височина на поддърво. Работи се по нарастващ ред на тези разстояния. Текущите състояния с дадена височина се бележат с етикет, който зависи само от това, дали състоянието е крайно, и от това с какви символи в какви състояния се преминава. Тези етикети в този случаи еднозначно определят езика на подавтомата с начало това състояние. Извършва се сортировка по тези етикети, като при това състоянията с еднакви етикети се слепват. Показва се, че по този начин алгоритъмът е с линейна сложност [Re92] и резултатът от изпълнението му е минимален краен автомат.

Недостатъкът на този метод се състои в това, че междинният автомат, създаден по време на първия етап от алгоритъма, е с огромни размери, което прави алгоритъма трудно приложим на практика. По-нататък в дисертацията си Ревуз пише по този въпрос:

> "Cette méthode, très simple à mettre en oeuvre, a l'inconvénient de créer un arbre de trop grande taille pour nos dictionnaires." [2]

За да се избегне този проблем, в [Ro92] се предлага да се минимизират първо части от списъка, като след това съответните автомати се обединят, използвайки стандартните алгоритми за обединение на детерминирани крайни автомати. Това е само частично решение на проблема, защото алгоритъмът за обединение на два детерминирани крайни автомата с по n състояние създава детерминиран автомат с n^2 състояния. Поради това се налага допълнително минимизиране на резултата, което забавя и затруднява процедурата.

В настоящата работа ще покажем, че твърдението на Ревуз, че не същест-

[2] Буквален превод на български: „Този метод, много лесен за реализация, има недостатъка да създава прекалено голямо дърво за нашите речници."

вува директен алгоритъм за създаване на минимален ацикличен краен автомат, вече не е в сила. Ще представим ефективен алгоритъм, който по даден лексикографски подреден списък на думи построява директно минимален автомат, описващ списъка. За първи път методът публично е представен през декември 1997 година на Научната сесия в памет на проф. Тагамлицки в Софийския университет. Докладът [Mi98] е предаден през февруари 1998 и е приет за печат през април 1998 година. През юли 1998 година в [DWW98] подобен алгоритъм (без формална математическа концепция и доказателство за коректност) е представен и от Ян Дациук, Брус Уатсон и Ричард Уатсон. След кореспонденция с авторите се получи споразумение работата [Mi98] да бъде цитирана от тях като предхождаща.

Друг подход за директно построяване на минимален ацикличен краен автомат е представен в [BBHECS85, CV97a, CV97b]. Той обаче е предназначен за приложение само в частния случай, когато езикът на автомата се състои от всички поддуми на дадена дума. В този случай се показва, че съществува алгоритъм, който директно построява минималния автомат, като времето за построяването му е пропорционално на дължината на изходната дума и не използва допълнителна памет. Този метод съществено използва свойствата на езика, поради което е неприложим в случая за построяване на минимален автомат за произволен предварително зададен краен език.

Също така има множество изследвания, които предлагат разширения на регулярните изрази с цел създаване на по-удобен език за описание. Регулярните изрази могат да се използват както за описание на регулярни езици, така и за регулярни релации. Най-често срещаните разширения на регулярните изрази са свързани с оператори за инвертиране, сечение и допълнение (на регулярни езици). Много по-разнообразни са разширенията на регулярните изрази за описание на релации. Някои от по-известните разширения при регулярните релации са операторите за декартово произведение, композиция, обръщане, инвертиране и проекция. Целта е, без да се излиза от класа на регулярните релации, да се създадат мощни и удобни оператори. Най-интересни са операторите за замяна, които служат за описание на трансформационни правила. Такива оператори могат да изглеждат по следния начин:

$$A \longrightarrow B \ / \ C \_\_ D$$

Горният регулярен израз изразява регулярната релация, при която поддумата A се заменя с поддумата B, ако A се намира в непосредствения контекст между поддумите C и D. В по-общия случай A, B, C и D могат да бъдат регулярни езици. Различни разширения на регулярните изрази, както и методите за трансформирането им до крайни автомати и преобразуватели, са описани в [KK96, Ka95, Ka96]. Методите и алгоритмите за сечение, обединение и разлика на детерминирани крайни автомати са добре известни [HU79]. При тези методи се строи автомат, чиито състояния са декартово произведение на състоянията на изходните автомати. Поради това автоматът, който се получава, е с огромни размери. Ние ще разгледаме случая на ациклични автомати и ще представим метод и алгоритъм за директното построяване на минимален ацикличен краен автомат, представящ сечение, обединение или разлика на ациклични крайни автомати. Този метод и базираният на него алгоритъм са представени за първи път в [Mi99].

Приложения на крайните автомати в компютърната лингвистика

В компютърната лингвистика крайни автомати и преобразуватели се използват за най-разнобразни цели, започвайки от предварителна обработка на текстове, през анализи на думите в текстове до плитък синтактичен анализ. В [Mo95] и [KCGS96] може да се намери преглед на някои от приложенията на крайните автомати в обработката на естествения език.

За предварителната обработка на текстове в [Si93] се предлага задаването на краен автомат, разпознаващ границите на изречения. В [GT94] се решава задачата за разбиване на текста на думи и изречения чрез крайни преобразуватели.

Най-широко крайните автомати се използват за описание на граматични речници. В [Re91, Ro92, Si93] се моделират граматични речници чрез крайни автомати с етикети. По-мощен начин за описание на речници е чрез използване на трансформационни правила, описващи фонетичните изменения. При този подход речникът се представя чрез крайни преобразуватели. Изследвания в това направление се предлагат в [Ko83, KK94, KKZ92, Ka93, Ka94]. Конкретни реализации на този подход са изложени в [An90, An94, KB92].

В [Mo97] е представено приложение на крайни автомати за разпознаване на говор. Откриване на граматични шаблони в текстовете чрез използване на крайни автомати е описано в [Si93, Sc96, KCGS96]. Друго приложение на крайните автомати е свързано с решаването на задачата за определяне на частите на речта на думите в изречения [RS95]. Става въпрос за това, че много от думите могат да имат по няколко значения. Например *коса* може да бъде както глагол, така и съществително и задачата се състои в това, да се определи каква част на речта има думата в дадено срещане в изречението. Крайните автомати намират приложение също при повърхнинен синтактичен анализ.

Създадени са и универсални системи за работа с крайни автомати и преобразуватели. За най-мощна в момента се счита системата XFST на изследователския център на компанията XEROX в Гренобъл [KGK97]. Основните преимущества на системата са възможностите за задаване на разширени регулярни изрази и ефективните методи за опериране с крайни автомати и преобразуватели.

Приноси на дисертацията

Дисертацията цялостно представя разработените от автора конструкции, алгоритми и приложения на минималните ациклични крайни автомати.

Оригиналните приноси на автора могат се обобщят в следните точки:

- Дефиниране и изследване на минимални освен в дадена дума крайни автомати. Това понятие се дефинира в секция 1.4.

- Метод и алгоритъм за директно построяване на минимален ацикличен краен автомат по даден списък. Алгоритъмът има сложност $O(n\log(m))$ по отношение на време и $O(m)$ по отношение на памет, където n е броят на символите във входния списък, а m е броят на състоянията на резултатния минимален автомат. Идеята за алгоритъма и първата му реализация са направени в съавторство с Богдан Александров.

- Метод и алгоритъм за директно построяване на минимален ацикличен краен автомат с етикети и моноиден преобразувател, връщащ индекс. Алгоритъмът е обобщение на горния и притежава същата сложност.

- Метод и алгоритъм за директно построяване на минимален ацикличен краен автомат, представящ обединение, сечение или разлика на ациклични крайни автомати. Сложността по отношение на време на алгоритъма е $O(n\log(m))$ за обединение и $O(n_1 + n\log(m))$ за сечение и разлика, където n_1 е общият брой символи на първия автоматен език, n е общият брой символи в резултатния език, а m е размерът на резултатния минимален автомат. По отношение на памет сложността на алгоритъма е $O(m)$.

- Дефиниране и изследване на нормализирани n-лентови крайни автомати (секция 1.7).

- Методи за директно построяване на минимален нормализиран ацикличен n-лентов краен автомат.

- Приложение на метода и алгоритъма за директно построяване на минимален моноиден преобразувател, връщащ индекс, в задачата за индексиране на текст.

- Разработване на JAVA библиотеката MAFSA, в която са имплементирани посочените методи и алгоритми. Чрез използването на тази програмна библиотека е осъществено изпълнението на проекта към НФНИ „Граматичен WWW сървър за български език“.

- Разработване на библиотеката за обработка на корпуси от текстове на ЦЛПОИ. Тази библиотека също реализира някои от гореспоменатите алгоритми. Чрез нея е осъществена основната лингвистична обработка на текстовете в програмните системи MARK-ALISTeR EXTRA и BILEDITA. Тези системи са включени в международните научни проекти по програмата Copernicus – GLOSSER и BILEDITA. Библиотеката е разработена в съавторство с Богдан Александров.

Апробация

Резултати, представени в работата, са успешно приложени в няколко европейски научни проекта, в които авторът е участник:

- BILEDITA #790 COPERNICUS'94 JRP,
- GLOSSER #343 COPERNICUS'94 JRP,
- TELRI #1202 COPERNICUS'94 CA.

Авторът е участвал в горепосочените проекти като научен сътрудник в Секцията „Лингвистично моделиране“ към ЦЛПОИ на БАН. Той е и ръководител на проекта МУ-МИ-04/96 към НФ „Научни изследвания“ за създаване на граматичен WWW сървър за български език.

Резултатите, представени в дисертацията, са докладвани от автора на следните научни прояви:

- Годишно-отчетна среща за проекта BILEDITA Мюнхен, Октомври 1996.
- Втори европейски семинар "Language Applications for a Multilingual Europe", Каунас, Април 1997.
- Отчетна среща за проекта GLOSSER, Гронинген, Април 1997.
- Конференция"Language Teaching & Language Technology", Гронинген, Април 1997.
- Трети европейски семинар "Translation Equivalence", Монтекатини, Октомври 1997.
- Научна сесия в памет на проф. Тагамлицки, София, Декември 1997.
- Семинар за представяне на Граматичен WWW сървър, София, Април 1998.

Публикации във връзка с дисертацията

Авторът има досега шест публикации, свързани с темата, три от които са публикувани в чуждестранни издания и три са публикувани или приети за печат в Годишника на Софийския университет – Факултет по математика и информатика. Пет от публикациите са самостоятелни и една е в съавторство.

Структура на дисертацията

В първата глава се припомнят означенията и основните математически понятия за крайни автомати, крайни автомати с етикети, моноидни преобразуватели и n-лентови крайни автомати. Представят се техни основни свойства. Дефинира се новото понятие мининимален освен в дадена дума автомат. С негова помощ се представя методът за директно построяване на минимален ацикличен краен автомат за даден списък от думи. Разглеждат се разширенията на този метод за директно построяване на минимален ацикличен краен автомат с етикети и моноиден преобразувател, връщащ индекс. Следва методът за директно построяване на минимален ацикличен краен автомат, представящ сечение, обединение или разлика на ациклични крайни автомати. Накрая в тази глава е дадено преобразувание, чрез което горните методи са приложени за директно построяване на минимален нормализиран ацикличен n-лентов краен автомат.

Втората глава съдържа формални описания на алгоритмите, които се базират на методите, представени в първата глава. Дадени са ефективни алгоритми за директно построяване по даден списък от думи на минимален ацикличен краен автомат, краен автомат с етикети и моноиден преобразувател, връщащ индекс. След това са представени алгоритми за директно построяване на минимален ацикличен краен автомат, представящ сечение, обединение или разлика на ациклични крайни автомати. Изложени са и ефективни алгоритми за разпознаване и класифициране на дума от автомат, както и алгоритми за намиране на индекс на дума и дума по нейния индекс от моноиден преобразувател, връщащ индекс. За основните алгоритми са представени доказателства за коректност и оценка на алгоритмична сложност.

Третата глава представя приложения на методите и алгоритмите за директно построяване на минимален ацикличен краен автомат в компютърната лингвистика. Показва се как се построяват минимален ацикличен краен автомат с етикети и минимален ацикличен краен преобразувател, моделиращи граматични речници. След това се дава приложение на моноиден преобразувател връщащ индекс за индексиране на текст. В края на главата са представени реално действащи компютърни системи, в които описаните конструкции и алгоритми са реализирани.

Благодарности

Първо искам да изразя искрената си благодарност на проф. д-р Димитър Скордев за проявените от него грижи и търпение. Чрез неговите забележки, идеи и препоръки работата стана много по-издържана от математическа гледна точка. Голяма помощ ми беше оказана от моите колеги доц. Тинко Тинчев, н.с. Светла Коева, ас. Богдан Александров и ас. Антон Зиновиев. Техните ценни идеи и препоръки внесоха съществен порядък в черновите ми. Искам да изкажа моята благодарност на колектива на ЦЛПОИ за плодотворната обстановка, в която тази работа беше осъществена. И последно искам да изразя своята признателност на мойте близки за търпението, което бе проявено от тях, и за това, че ме освободиха от много други натоварвания.

Глава 1

Математически концепции и резултати

Основната цел на тази глава е формалното представяне на математическите концепции и новите методи за директно построяване на минимални ациклични крайни автомати. За целта първо се представят накратко основните дефиниции и свойства на крайните автомати и регулярните езици. След това се излагат дефиниции и свойства на различни модификации и разширения на класическите крайни автомати, които се използват за решаване на различни приложни задачи. В главата се представят n-лентовите крайни автомати, регулярните релации, моноидните преобразуватели и крайните автомати с етикети на крайните състояния. Допълнително са разгледани крайните преобразуватели и моноидните преобразуватели връщащи индекс. Основните оригинални резултати, представени в главата, са методите за директно построяване на минимален ацикличен автомат. Разгледани са случаите на построяване на автомата по даден списък от думи и построяване на автомат, представящ сечение, обединение или разлика на автомати. Описани са също така модификациите на методите в случайте на ацикличен моноиден преобразувател, връщащ индекса на думите в списъка, на автомат с етикети на крайните състояния и n-лентов краен автомат.

1.1 Детерминирани и недетерминирани крайни автомати. Регулярни изрази и езици

Крайни автомати

Дефиниция 1. Детерминиран краен автомат ще наричаме петорката $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$, където:

- Σ е крайна азбука;
- S е крайно множество от състояния;
- $s \in S$ е началното състояние;
- $F \subseteq S$ е множеството от крайни състояния;
- $\mu : S \times \Sigma \to S$ е частична функция на преходите.

Функцията μ се разширява естествено до частична функция над $S \times \Sigma^*$ индуктивно:

$$\begin{cases} \mu^*(r, \varepsilon) = r \\ \mu^*(r, \sigma a) = \begin{cases} \mu(\mu^*(r, \sigma), a) \text{ , в случай че } \mu^*(r, \sigma) \text{ и } \mu(\mu^*(r, \sigma), a) \text{ са дефинирани} \\ \text{не е дефинирана в противен случай} \end{cases} \end{cases},$$

където $r \in S, \sigma \in \Sigma^*, a \in \Sigma$.

По-нататък ще работим с дефиниция на крайни автомати с частична функция на преходите. Основната разлика при тази дефиниция се състои в това, че не е необходимо въвеждането на състояние без изходящи преходи (т.е. не крайно състояние r, за което е изпълнено $\forall a \in \Sigma(\mu(r, a) = r)$).

Ще използваме следните означения:

- $!\mu^*(r, \sigma)$ ще означава, че $\mu^*(r, \sigma)$ е дефинирано;
- $\mu^*(r, \sigma) \cong x$, ще означава $!\mu^*(r, \sigma)\ \&\ \mu^*(r, \sigma) = x$.

Дефиниция 2. Нека е даден детерминираният краен автомат $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$. Тогава множеството $L(\mathcal{A}) \subseteq \Sigma^*$, дефинирано като

$$L(\mathcal{A}) = \{\sigma \in \Sigma^* \,|\, !\mu^*(s, \sigma) \;\&\; \mu^*(s, \sigma) \in F\},$$

ще наричаме език на автомата $\mathcal{A}$ или език, разпознаван от автомата $\mathcal{A}$.

Нека с Σ^ε означим $\Sigma \cup \{\varepsilon\}$. Понятието недетерминиран краен автомат (с ε-преходи) ще дефинираме по следния начин:

Дефиниция 3. Недетерминиран краен автомат ще наричаме петорката $\mathcal{N} = \langle \Sigma, S, s, F, \mu \rangle$, където:

- Σ е крайна азбука;
- S е крайно множество от състояния;
- $s \in S$ е началното състояние;
- $F \subseteq S$ е множеството от крайни състояния;
- $\mu : S \times \Sigma^\varepsilon \to 2^S$ е функция на преходите.

Функцията μ се разширява до функция над $S \times \Sigma^*$ по следния начин:

$$\begin{aligned}
\mu^*(p, \sigma) \;\; = \;\; & \{r \in S \,|\, \exists m \in \mathbb{N}\; \exists a_1, a_2, ..., a_m \in \Sigma^\varepsilon\; \exists t_0, t_1, ..., t_m \in S \\
& ((\sigma = a_1 a_2 ... a_m) \;\&\; (t_0 = p) \;\&\; (t_m = r) \;\& \\
& (\forall i \in \{1, 2, ..., m\}\; (t_i \in \mu(t_{i-1}, a_i))))\},
\end{aligned}$$

за всеки $p \in S, \sigma \in \Sigma^*$. В случая, когато $\sigma = \varepsilon$ при $m = 0$, получаваме, че $p \in \mu^*(p, \varepsilon)$.

Дефиниция 4. Нека е даден недетерминираният краен автомат $\mathcal{N} = \langle \Sigma, S, s, F, \mu \rangle$. Тогава множеството $L(\mathcal{N}) \subseteq \Sigma^*$, дефинирано като

$$L(\mathcal{N}) = \{\sigma \in \Sigma^* \,|\, \mu^*(s, \sigma) \cap F \neq \emptyset\},$$

ще наричаме език на автомата $\mathcal{N}$ или език, разпознаван от автомата $\mathcal{N}$.

Един детерминиран краен автомат ще наричаме ацикличен, ако е изпълнено условието $\forall r \in S\ \forall \sigma \in \Sigma^+\ (\mu^*(r, \sigma) \not\cong r)$. Аналогично, един недетерминиран краен автомат ще наричаме ацикличен, ако $\forall r \in S\ \forall \sigma \in \Sigma^+\ (\mu^*(r, \sigma) \cap \{r\} = \emptyset)$. Езикът на всеки ацикличен краен автомат е краен.

Два крайни автомата ще наричаме еквивалентни, ако езиците им съвпадат. Ясно е, че всеки детерминиран краен автомат може да се разглежда като недетерминиран. По-интересна е добре известната теорема за еквивалентност на детерминираните и недетерминираните крайни автомати.

Теорема 5. *За всеки недетерминиран краен автомат съществува еквивалентен на него детерминиран краен автомат.*

Също така съществува алгоритъм, който по даден недетерминиран краен автомат построява еквивалентен на него детерминиран краен автомат.

Дефиниция 6. Нека $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ е детерминиран краен автомат.

1. Състоянието $r \in S$ ще наричаме достижимо от $t \in S$, когато $\exists \sigma \in \Sigma^*\ (\mu^*(t, \sigma) \cong r)$.

2. Подавтомат с начало $s' \in S$ дефинираме като: $\mathcal{A}|_{s'} = \langle \Sigma, S', s', F \cap S', \mu|_{S' \times \Sigma} \rangle$, където $S' = \{r \in S \mid r \text{ е достижимо от } s'\}$.

3. Две състояния $s_1, s_2 \in S$ наричаме еквивалентни, когато $\mathcal{A}|_{s_1}$ е еквивалентен на $\mathcal{A}|_{s_2}$;

Дефиниция 7. Детерминираният краен автомат $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ с език $L(\mathcal{A})$ ще наричаме минимален (с език $L(\mathcal{A})$), ако за всеки детерминиран автомат $\mathcal{A}' = \langle \Sigma, S', s', F', \mu' \rangle$ с език $L(\mathcal{A}') = L(\mathcal{A})$ е изпълнено $|S| \leq |S'|$.

Важна за практически приложения е следната теорема:

Теорема 8. *Един детерминиран краен автомат с непразен език е минимален, тогава и само тогава когато всяко негово състояние е достижимо от началното, от всяко състояние е достижимо някое крайно състояние и няма различни еквивалентни състояния. За всеки език на краен автомат съществува единствен (с точност до изоморфизъм) минимален краен автомат, разпознаващ този език.*

Съществува алгоритъм, който по даден детерминиран краен автомат построява еквивалентен на него минимален краен автомат.

Регулярни изрази и езици

Дефиниция 9. Нека е дадена крайна азбука Σ. Регулярен израз над Σ представлява дума над $\Sigma \cup \{(,),*,+,\cdot,\emptyset\}$, дефинирана индуктивно:

1. $\emptyset$ е регулярен израз;
2. ако $\sigma \in \Sigma^*$, то σ е регулярен израз;
3. ако E_1 и E_2 са регулярни изрази, то
   - $(E_1 + E_2)$ е регулярен израз,
   - $(E_1 \cdot E_2)$ е регулярен израз,
   - E_1^* е регулярен израз.

Регулярен език дефинираме по подобен индуктивен начин.

Дефиниция 10. Нека е дадена крайна азбука Σ. Тогава дефинираме регулярен език над Σ индуктивно:

1. $\emptyset$ е регулярен език;
2. ако $\sigma \in \Sigma^*$ то $\{\sigma\}$ е регулярен език;
3. ако L_1 и L_2 са регулярни езици, то
   - $L_1 \cup L_2$ е регулярен език (обединение),
   - $L_1.L_2 = \{\sigma_1\sigma_2 \,|\, \sigma_1 \in L_1, \sigma_2 \in L_2\}$ е регулярен език (конкатенация),
   - $L_1^* = \bigcup_{i=0}^{\infty} L_1^i$ е регулярен език (затваряне на Клини).

По естествен начин може да се съпостави на всеки регулярен израз съответен регулярен език и обратно. По-дълбока е връзката между регулярните и автоматните езици. Тя се показва в теоремата на Клини.

Теорема 11 Kleene. *Един език е регулярен, тогава и само тогава когато е език на краен автомат.*

Съществува също така алгоритъм, който по даден регулярен израз построява краен автомат, разпознаващ регулярния език, съответстващ на входния регулярен израз.

Използвайки тази теорема по-нататък можем да отъждествяваме понятията регулярен език и език на краен автомат.

Най-интересните свойства на регулярните езици са следните:

Свойство 12.

1. *Допълнение на регулярен език е регулярен език;*

2. *Крайно сечение на регулярни езици е регулярен език.*

Доказателствата на горните теореми и свойства, както и представяне на споменатите алгоритми могат да се намерят например в [Br84, ДПД84, Ma96].

Крайни автомати с етикети на крайните състояния

Дефиниция 13. Детерминиран краен автомат с етикети на крайните състояния ще наричаме седморката $\mathcal{A} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$, където Σ, S, s, F, μ са дефинирани както при класическите детерминирани крайни автомати, а

- E е (крайно) множество от етикети;

- $\psi : F \to E$ е функция, присвояваща на крайно състояние етикет.

Език на автомат с етикети на крайните състояния се дефинира аналогично на класическия случай. Новото понятие е етикет на разпозната дума от автомат.

Дефиниция 14. Нека $\mathcal{A} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$ е детерминиран краен автомат с етикети на крайните състояния. Дефинираме функция, връщаща етикет на дума $e_{\mathcal{A}} : L(\mathcal{A}) \to E$, такава, че $e_{\mathcal{A}}(\sigma) = \psi(\mu(s, \sigma))$ за $\sigma \in L(\mathcal{A})$. Функцията $e_{\mathcal{A}}$ ще наричаме класификация на езика $L(\mathcal{A})$.

Върху автоматите с етикети могат да се приложат някои от методите, приложими за класическите автомати. За тази цел обаче трябва да се обобщят някои от понятията и резултатите, дадени за детерминираните крайни автомати.

По-долу са дадени съответните дефиниции за краен автомат с етикети.

Дефиниция 15. Нека $\mathcal{A} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$ е краен автомат с етикети на крайните състояния.

1. Автоматът $\mathcal{A}$ е еквивалентен на автомата с етикети $\mathcal{A}'$, ако $L(\mathcal{A}) = L(\mathcal{A}')$ и $e_{\mathcal{A}} = e_{\mathcal{A}'}$.

2. Подавтомат с начало $s' \in S$ дефинираме като:

$$\mathcal{A}|_{s'} = \langle \Sigma, E, S', s', F \cap S', \mu|_{S' \times \Sigma}, \psi|_{F \cap S'} \rangle ,$$

където $S' = \{r \in S \mid r \text{ е достижимо от } s'\}$.

3. Две състояния $s_1, s_2 \in S$ наричаме еквивалентни, когато $\mathcal{A}|_{s_1}$ е еквивалентен на $\mathcal{A}|_{s_2}$.

Дефиницията за достижимост на състояния на краен автомат с етикети съвпада с тази, дадена в дефиниция 6.

Дефиниция 16. Краен автомат с етикети на крайните състояния $\mathcal{A} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$ ще наричаме минимален, ако за всеки автомат с етикети $\mathcal{A}' = \langle \Sigma, E, S', s', F', \mu', \psi' \rangle$, еквивалентен на $\mathcal{A}$, е изпълнено $|S| \leq |S'|$.

В този случай отново е в сила теоремата за минималност на автомат.

Теорема 17. *Един краен автомат с етикети на крайните състояния с непразен език е минимален, тогава и само тогава когато всяко негово състояние е достижимо от началното, от всяко състояние е достижимо някое крайно състояние и няма различни еквивалентни състояния. За всеки език на краен автомат с дадена класификация съществува единствен (с точност до изоморфизъм) минимален краен автомат с етикети, разпознаващ този език, като го класифицира в съответствие на дадената класификация.*

Забележка. За да докажем горните свойства, можем да сведем крайните автомати с етикети до детерминирани крайни автомати по следния начин. Нека е даден автомат с етикети $\mathcal{A} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$. Разглеждаме детерминирания краен автомат $\mathcal{A}' = \langle \Sigma \cup E, S \cup \{f\}, s, \{f\}, \mu' \rangle$, където f е ново състояние, а $\mu' = \mu \cup \{\langle p, \psi(p), f \rangle \mid p \in F\}$.

Главното приложение на тези автомати се състои в използването им за класифициране на разпознаваните думи.

1.2 n-лентови крайни автомати

Интересно обобщение на крайните автомати са многолентовите крайни автомати. Класическите крайни автомати могат да се разглеждат като еднолентови. За да се дефинират n-лентови крайни автомати, е необходимо да се дефинира конкатенация на n-торки от думи.

Нека $A = \langle a_1, a_2, ..., a_n \rangle \in (\Sigma^*)^n$, $B = \langle b_1, b_2, ..., b_n \rangle \in (\Sigma^*)^n$, тогава:

$$A.B =_{df} \langle a_1 b_1, a_2 b_2, ..., a_n b_n \rangle .$$

Нека с ε_n означим n-торката $\langle \varepsilon, \varepsilon, ..., \varepsilon \rangle$.

Дефиницията на недетерминиран n-лентов краен автомат е следната:

Дефиниция 18. Недетерминиран n-лентов краен автомат ще наричаме петорката $\mathcal{N} = \langle \Sigma, S, s, F, \mu \rangle$, където:

- Σ е крайна азбука;
- S е крайно множество от състояния;
- $s \in S$ е началното състояние;
- $F \subseteq S$ е множеството от крайни състояния;
- $\mu : S \times (\Sigma^\varepsilon)^n \to 2^S$ е функция на преходите.

Функцията μ се разширява до функция над $S \times (\Sigma^*)^n$ по следния начин:

$$\begin{aligned}\mu^*(p,\sigma) \;\; = \;\; & \{r \in S \,|\, \exists m \in \mathbb{N}\; \exists a_1, a_2, ..., a_m \in (\Sigma^\varepsilon)^n \; \exists t_0, t_1, ..., t_m \in S \\ & ((\sigma = a_1 a_2 ... a_m) \;\&\; (t_0 = p) \;\&\; (t_m = r) \;\& \\ & (\forall i \in \{1, 2, ..., m\} \; (t_i \in \mu(t_{i-1}, a_i))))\},\end{aligned}$$

за всеки $p \in S, \sigma \in (\Sigma^*)^n$.

Дефиниция 19. Нека е даден n-лентовият недетерминиран краен автомат $\mathcal{N} = \langle \Sigma, S, s, F, \mu \rangle$. Тогава множеството $L(\mathcal{N}) \subseteq (\Sigma^*)^n$, дефинирано като

$$L(\mathcal{N}) = \{\sigma \in (\Sigma^*)^n \,|\, \mu^*(s,\sigma) \cap F \neq \emptyset\},$$

ще наричаме релация на автомата $\mathcal{A}$ или релация, разпознавана (описана) от автомата $\mathcal{A}$.

Дефинирането на детерминиран n-лентов краен автомат в нашия случая не е оправдано, тъй като в n-лентовия случай няма еквивалентност между детерминираните и недетерминираните крайни автомати. Регулярните изрази и езици обаче могат естествено да се разширят до регулярни n-изрази и n-релации.

Нека е дадена крайна азбука Σ.

Дефиниция 20. Регулярен n-израз над Σ ще наричаме дума над азбуката $(\Sigma^*)^n \cup \{(,), *, +, \cdot, \emptyset\}$, дефинирана индуктивно:

1. $\emptyset$ е регулярен n-израз;
2. ако $\sigma \in (\Sigma^*)^n$ то σ е регулярен n-израз;
3. ако E_1 и E_2 са регулярни n-изрази, то
   - $(E_1 + E_2)$ е регулярен n-израз,
   - $(E_1 \cdot E_2)$ е регулярен n-израз,
   - E_1^* е регулярен n-израз.

Дефиниция 21. Регулярна n-релация над Σ дефинираме индуктивно:

1. $\emptyset$ е регулярна n-релация;

2. ако $\sigma \in (\Sigma^*)^n$, то $\{\sigma\}$ е регулярна n-релация;

3. ако R_1 и R_2 са регулярни n-релации, то

   - $R_1 \cup R_2$ е регулярна n-релация (обединение),
   - $R_1.R_2 = \{\sigma_1\sigma_2 \,|\, \sigma_1 \in R_1, \sigma_2 \in R_2\}$ е регулярна n-релация (конкатенация),
   - $R_1^* = \bigcup_{i=0}^{\infty} R_1^i$ е регулярна n-релация (затваряне на Клини).

По естествен начин може да се съпостави на всеки регулярен n-израз съответна регулярна n-релация и обратно. Съществува и еквивалент на теоремата на Клини в n-лентовия случай.

Теорема 22 Kleene. *Една n-релация е регулярна, тогава и само тогава когато е релация на недетерминиран n-лентов краен автомат.*

Доказателството на тази теорема повторя схемата, използвана за доказването на класическия вариант. Единствената разлика е използването на новия вариант на конкатенация.

Основни свойства на крайните преобразуватели

От n-лентовите автомати особен интерес представляват двулентовите. Двулентовите крайни автомати се наричат крайни преобразуватели. Това наименование се дължи на факта, че представлявайки бинарна релация, те могат да се използват за преобразуване на думи и езици. Това може да стане по следния неформално описан начин: Входната дума се записва на първата лента и се търси път в автомата до крайно състояние, което изчита думата от първата лента. При това на втората лента се получава дума в релация с първата. Разбира се може да има повече от един такъв път. Като се изброят всички такива пътища в автомата (може и да не са краен брой), се намират всички думи в релация с входната. Поради съображения за симетричност това може да се направи и в обратна посока.

Някой от свойствата, изброени по-долу, могат да се пренесат върху n-лентовите автомати, но тъй като по-нататък ще използваме основно преобразуватели, ще се съсредоточим върху тях. Под регулярна релация ще разбираме регулярна 2-релация.

Свойство 23. *За всеки краен преобразувател* $\mathcal{T} = \langle \Sigma, S, s, F, \mu \rangle$ *можем да построим еквивалентен на него краен преобразувател* $\mathcal{T}' = \langle \Sigma, S', s', F', \mu' \rangle$*, така че* $\forall r \in S \ (\mu'(r, \varepsilon_2) = \emptyset)$.

Това може да се направи, като се използва класическият алгоритъм за отстраняване на ε-преходи от недетерминиран краен автомат.

Забележка. Възможно е да се приложи класическият алгоритъм за детерминиране и минимизиране на автомат, в случая на преобразувател, като се разглежда n-торка от букви и ε като една буква. При това обаче в общия случай не се получава детерминиран или минимален преобразувател, поради наличието на ε-и в n-торките на преходите.

Свойство 24.

1. *Ако* L *е регулярен език, то* $Id(L) = \{\langle \sigma, \sigma \rangle \mid \sigma \in L\}$ *е регулярна релация.*

2. *Ако* R *е регулярна релация, то* $R^{-1} = \{\langle \sigma, \tau \rangle \mid \langle \tau, \sigma \rangle \in R\}$ *е регулярна релация.*

3. *Доменът и образът регулярна релация* R*, означавани като* $dom(R) = \{\sigma \mid \exists \tau \in \Sigma^* \ (\langle \sigma, \tau \rangle \in R)\}$*,* $range(R) = \{\tau \mid \exists \sigma \in \Sigma^* \ (\langle \sigma, \tau \rangle \in R)\}$*, са регулярни езици.*

4. *Декартово произведение на два регулярни езика* L_1, L_2 *е регулярна релация* $R = L_1 \times L_2 = \{\langle \sigma, \tau \rangle \mid \sigma \in L_1, \tau \in L_2\}$.

5. *Ако* R_1, R_2 *са регулярни релации, то композицията им* $R = R_1 \bullet R_2 = \{\langle \sigma, \tau \rangle \mid \exists \psi \in \Sigma^* \ (\langle \sigma, \psi \rangle \in R_1 \ \& \ \langle \psi, \tau \rangle \in R_2)\}$ *е регулярна релация.*

6. *Образ на регулярен език* L *през регулярна релация* R*, означаван като* $L/R = \{\sigma \mid \exists \tau \in L \ (\langle \tau, \sigma \rangle \in R)\}$*, е регулярен език. От симетричността следва, че и* $R/L = \{\sigma \mid \exists \tau \in L \ (\langle \sigma, \tau \rangle \in R)\}$ *е регулярен език.*

Доказателство.

1. Нека $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ е детерминиран краен автомат, разпознаващ езика L. Тогава крайният преобразувател $\mathcal{T} = \langle \Sigma, S, s, F, \mu' \rangle$, където

   $$\mu'(q, A) = \begin{cases} r \in S, \text{ ако } A = \langle a, a \rangle \ \& \ a \in \Sigma \ \& \ \mu(q, a) \cong r \\ \text{не е дефинирана в противен случай,} \end{cases}$$

   за $q \in S, A \in (\Sigma^\varepsilon)^2$, разпознава релацията $\mathrm{Id}(L)$.

2. Нека крайният преобразувател $\mathcal{T} = \langle \Sigma, S, s, F, \mu \rangle$ разпознава релацията R. Тогава крайният преобразувател $\mathcal{T}' = \langle \Sigma, S, s, F, \mu' \rangle$, където

   $$\mu'(q, \langle a, b \rangle) = \mu(q, \langle b, a \rangle)$$

   за $q \in S, a, b \in \Sigma^\varepsilon$, разпознава релацията R^{-1}.

3. Нека крайният преобразувател $\mathcal{T} = \langle \Sigma, S, s, F, \mu \rangle$ разпознава релацията R. Тогава недетерминираният краен автомат $\mathcal{N} = \langle \Sigma, S, s, F, \mu' \rangle$, където

   $$\mu'(q, a) = \{ r \in S \mid \exists b \in \Sigma^\varepsilon \ (r \in \mu(q, \langle a, b \rangle)) \}$$

   за $q \in S, a \in \Sigma^\varepsilon$, разпознава езикът $\mathrm{dom}(R)$. Образът се получава от равенството: $\mathrm{range}(R) = \mathrm{dom}(R^{-1})$.

4. Нека $\mathcal{N}_1 = \langle \Sigma, S_1, s_1, F_1, \mu_1 \rangle$ и $\mathcal{N}_2 = \langle \Sigma, S_2, s_2, F_2, \mu_2 \rangle$ са недетерминирани крайни автомати, разпознаващи съответно езиците L_1 и L_2. Без ограничение на общността можем да предполагаме, че $\forall r \in S_1 \ (r \in \mu_1(r, \varepsilon))$ и $\forall r \in S_2 \ (r \in \mu_2(r, \varepsilon))$. Тогава разглеждаме крайния преобразувател $\mathcal{T} = \langle \Sigma, S_1 \times S_2, \langle s_1, s_2 \rangle, F_1 \times F_2, \mu \rangle$, където

   $$\mu(\langle q_1, q_2 \rangle, \langle a_1, a_2 \rangle) = \mu_1(q_1, a) \times \mu_2(q_2, a_2)$$

   за $q_1 \in S_1, q_2 \in S_2, a_1, a_2 \in \Sigma^\varepsilon$. Чрез индукция се показва, че

   $$\mu^*(\langle q_1, q_2 \rangle, \langle \sigma_1, \sigma_2 \rangle) = \mu_1^*(q_1, \sigma_1) \times \mu_2^*(q_2, \sigma_2)$$

   за $q_1 \in S_1, q_2 \in S_2, \sigma_1, \sigma_2 \in \Sigma^*$. От това следва, че $\mathcal{T}$ разпознава релацията $L_1 \times L_2$.

5. Нека $\mathcal{T}_1 = \langle \Sigma, S_1, s_1, F_1, \mu_1 \rangle$ и $\mathcal{T}_2 = \langle \Sigma, S_2, s_2, F_2, \mu_2 \rangle$ са крайни преобразуватели, разпознаващи съответно релациите R_1 и R_2. Без ограничение на общността можем да предполагаме, че $\forall r \in S_1$ $(r \in \mu_1(r, \varepsilon_2))$ и $\forall r \in S_2$ $(r \in \mu_2(r, \varepsilon_2))$. Тогава разглеждаме крайния преобразувател:

$$\mathcal{T} = \langle \Sigma, S_1 \times S_2, \langle s_1, s_2 \rangle, F_1 \times F_2, \mu \rangle,$$

където

$$\mu(\langle q_1, q_2 \rangle, \langle a_1, a_2 \rangle) = \{\langle t_1, t_2 \rangle \mid \exists b \in \Sigma^\varepsilon \ (t_1 \in \mu_1(q_1, \langle a_1, b \rangle) \ \& \ t_2 \in \mu_2(q_2, \langle b, a_2 \rangle))\}$$

за $q_1 \in S_1, q_2 \in S_2, a_1, a_2 \in \Sigma^\varepsilon$. Чрез индукция се показва, че

$$\mu^*(\langle q_1, q_2 \rangle, \langle \sigma_1, \sigma_2 \rangle) = \{\langle t_1, t_2 \rangle \mid \exists \tau \in \Sigma^* \ (t_1 \in \mu_1^*(q_1, \langle \sigma_1, \tau \rangle) \ \& \ t_2 \in \mu_2^*(q_2, \langle \tau, \sigma_2 \rangle))\}$$

за $q_1 \in S_1, q_2 \in S_2, \sigma_1, \sigma_2 \in \Sigma^*$. От това следва, че $\mathcal{T}$ разпознава релацията $R_1 \bullet R_2$.

6. Регулярността на езиците L/R и R/L следва от равенствата:

$$L/R = \mathrm{range}(\mathrm{Id}(L) \bullet R)$$

$$R/L = \mathrm{dom}(R \bullet \mathrm{Id}(L)).$$

$\square$

Свойство 25. *Сечение на регулярни релации не винаги е регулярна релация.*

Доказателство. Да разгледаме регулярните релации: $R_1 = \langle a, b \rangle^* \cdot \langle \varepsilon, c \rangle^*$ и $R_2 = \langle \varepsilon, b \rangle^* \cdot \langle a, c \rangle^*$. Ясно е, че $R_1 = \{\langle a^n, b^n c^* \rangle \mid n \geq 0\}, R_2 = \{\langle a^n, b^* c^n \rangle \mid n \geq 0\}$. Сечението на R_1 и R_2 е релацията $R = \{\langle a^n, b^n c^n \rangle \mid n \geq 0\}$, която не е регулярна, защото нейният образ е контекстно свободният език $b^n c^n$. $\square$

От горното свойство и закона на Де Морган следва, че допълнение на регулярна релация в общия случай не винаги е регулярна релация. От това следва, че не за всеки недетерминиран краен преобразувател съществува еквивалентен на него детерминиран. Защото в противен случай бихме могли да приложим класическия метод за конструиране на автомат, разпознаващ допълнение на езика (релацията) на друг автомат.

Интерес представляват регулярните релации между думи с еднаква дължина.

Дефиниция 26. Една регулярна релация R ще наричаме регулярна релация между думи с еднаква дължина, ако изпълнява следното условие:

$$\forall \langle \sigma, \tau \rangle \in R \, (|\sigma| = |\tau|).$$

Ще казваме, че краен преобразувател $\mathcal{T} = \langle \Sigma, S, s, F, \mu \rangle$ е свободен от ε-и, ако е изпълнено условието:

$$\forall r \in S \; \forall a, b \in \Sigma^{\varepsilon} \; ((a = \varepsilon \;\vee\; b = \varepsilon) \rightarrow \mu(r, \langle a, b \rangle) = \emptyset).$$

В сила са следните важни свойства за регулярните релации между думи с еднаква дължина.

Лема 27. *Една регулярна релация е релация между думи с еднаква дължина, тогава и само тогава когато е релация на краен преобразувател, свободен от ε-и.*

Доказателство. Обратната посока е тривиална — ако краен преобразувател е свободен от ε-и, то той при всеки преход съпоставя на символ от едната лента символ от другата и следователно релацията, която разпознава, е релация между думи с еднаква дължина. Ще представим схема на доказателството за правата посока. Нека е даден краен преобразувател, разпознаващ регулярна релация между думи с еднаква дължина. Без ограничение на общността можем да предполагаме, че всяко състояние е достижимо от началното и от всяко състояние е достижимо някое крайно. Чрез стандартния алгоритъм за премахване на ε-преходи можем да получим еквивалентен преобразувател, за който $\forall r \in S \; (\mu(r, \varepsilon_2) = \emptyset)$. По-нататък систематично ще правим преобразувания на преобразувателя, които запазват еквивалентността с първоначалния и премахват ε съдържащи преходи. Път от началното състояние до дадено състояние ще съдържа някакъв брой $\langle a, \varepsilon \rangle$ и някакъв брой $\langle \varepsilon, b \rangle$ преходи, като тези две числа може да не са равни, ако състоянието не е крайно. Но за всички пътища от началното състояние до даденото разликата между тези две числа трябва да бъде константна, тъй като тя трябва да бъде компенсирана от всеки път, който води от това състояние до някое крайно. Нека наречем тази разлика *дисбаланс* на състоянието. Тъй като всяко състояние е достижимо от ацикличен път, то абсолютната стойност на дисбаланса на състоянието е ограничена от броя на състоянията в преобразувателя. На всяка стъпка нашата процедура ще премахва състоянията с максимален дисбаланс. Ясно е, че всички преходи от вида $\langle a, b \rangle$ $(a, b \neq \varepsilon)$ ще свързват състояния с еднакъв дисбаланс. Такъв преход можем да премахнем чрез еквивалентен път $\langle \varepsilon, b \rangle \langle a, \varepsilon \rangle$ през ново състояние, което има дисбаланс с единица по-малък от началните две състояния. Нека предположим, че $k > 0$ е максималният дисбаланс в преобразувателя и че всички преходи от вида $\langle a, b \rangle$ между състояния с дисбаланс k са премахнати. Ако q е сътояние с дисбаланс k, то към него ще се преминава само чрез $\langle a, \varepsilon \rangle$ преходи от състояния с дисбаланс $k-1$ и от него ще излизат само $\langle \varepsilon, b \rangle$ преходи към състояния с дисбаланс $k-1$. За всеки преход $\langle a, \varepsilon \rangle$ от състояние p към q и за всеки преход $\langle \varepsilon, b \rangle$ от q към състояние r построяваме преход $\langle a, b \rangle$ от p към r. След това премахваме състоянието q заедно с всички преходи към и от него. Тази манипулация запазва еквивалентността на преобразувателя, като намалява броя на състоянията с дисбаланс k. Повтаряме процедурата за всички състояния с дисбаланс k. След това итеративно преминаваме към $k-1$ и така, докато получим преобразувател, в който няма състояния с положителен дисбаланс. Чрез симетричната на тази процедура премахваме и всички състояния с отрицателен дисбаланс. В получения накрая преобразувател няма състояния с дисбаланс, различен от 0 и следователно е свободен от ε-и. $\square$

От горната лема се получава следното следствие.

Следствие 28. *Сечение и разлика на регулярни релации между думи с еднаква дължина е регулярна релация между думи с еднаква дължина.*

Доказателство. Нека $\mathcal{T}_1 = \langle \Sigma, S_1, s_1, F_1, \mu_1 \rangle$ и $\mathcal{T}_2 = \langle \Sigma, S_2, s_2, F_2, \mu_2 \rangle$ са крайни преобразуватели, свободни от ε-и разпознаващи съответно релациите R_1 и R_2. Тогава разглеждаме крайния преобразувател $\mathcal{T} = \langle \Sigma, S_1 \times S_2, \langle s_1, s_2 \rangle, F_1 \times F_2, \mu \rangle$, където

$$\mu(\langle q_1, q_2 \rangle, \langle a_1, a_2 \rangle) = \mu_1(q_1, \langle a_1, a_2 \rangle) \times \mu_2(q_2, \langle a_1, a_2 \rangle)$$

за $q_1 \in S_1, q_2 \in S_2, a_1, a_2 \in \Sigma^\varepsilon$. Ясно е, че преобразувателят $\mathcal{T}$ е свободен от ε-и. Като се използва съществено този факт, можем да покажем чрез индукция, че

$$\mu^*(\langle q_1, q_2 \rangle, \langle \sigma_1, \sigma_2 \rangle) = \mu_1^*(q_1, \langle \sigma_1, \sigma_2 \rangle) \times \mu_2^*(q_2, \langle \sigma_1, \sigma_2 \rangle) \quad (1.1)$$

за $q_1 \in S_1, q_2 \in S_2, \sigma_1, \sigma_2 \in \Sigma^*$. Оттук получаваме, че $\mathcal{T}$ разпознава релацията $R_1 \cap R_2$.

Да разгледаме сега преобразувателя $\mathcal{D} = \langle \Sigma, S_1 \times S_2, \langle s_1, s_2 \rangle, F_1 \times (S_2 \setminus F_2), \mu \rangle$. Функцията μ е дефинирана както при преобразувателя $\mathcal{T}$ и следователно $\mathcal{D}$ е свободен от ε-и и е в сила условието (1.1). От това получаваме, че релацията на преобразувателя $\mathcal{D}$ е $R_1 \setminus R_2$. $\square$

Всички дадени по-горе доказателства са конструктивни и директно могат да се използват за получаването алгоритми за построяване на съответните автомати и преобразуватели.

По-подробно разглеждане на свойствата на крайните преобразуватели може да се намери в [KK94].

1.3 Моноидни преобразуватели

В тази секция ще разгледаме моноидните преобразуватели. Моноидния преобразувател можем да разглеждаме като детерминиран краен автомат, който при всеки преход извежда елемент от моноида. При разпознаване на думата се извежда елементът на моноида, който е моноидното произведение от елементите, извеждани при преминаване през преходите. Грубо казано по този начин моноидният преобразувател преобразува език на краен автомат в подмножество на моноида, наричано изходен език.

Дефиниция 29. Моноиден преобразувател ще наричаме седморката $\mathcal{M} = \langle \Sigma, M, S, s, F, \mu, \lambda \rangle$, където:

- Σ е крайна азбука;
- $M = \langle M, \circ, \mathbf{e} \rangle$ е моноид;
- S е крайно множество от състояния;
- $s \in S$ е началното състояние;
- $F \subseteq S$ е множеството от крайни състояния;

- $\mu : S \times \Sigma \to S$ е частична функция на преходите;

- $\lambda : S \times \Sigma \to M$ е частична функция, такава, че $\mathrm{dom}(\lambda) = \mathrm{dom}(\mu)$. По-нататък ще наричаме λ изходна функция.

Функциите μ и λ се разширяват над $S \times \Sigma^*$ индуктивно:

$$\begin{cases} \mu^*(r, \varepsilon) = r \\ \mu^*(r, \sigma a) = \begin{cases} \mu(\mu^*(r,\sigma), a) \text{ , в случай че } \mu^*(r,\sigma) \text{ и } \mu(\mu^*(r,\sigma), a) \text{ са дефинирани} \\ \text{не е дефинирана в противен случай} \end{cases} \end{cases},$$

$$\begin{cases} \lambda^*(r, \varepsilon) = \mathbf{e} \\ \lambda^*(r, \sigma a) = \begin{cases} \lambda^*(r,\sigma) \circ \lambda(\mu^*(r,\sigma), a) \text{ , в случай че } !\mu^*(r,\sigma) \;\&\; !\lambda(\mu^*(r,\sigma), a) \\ \text{не е дефинирана в противен случай} \end{cases} \end{cases},$$

където $r \in S, \sigma \in \Sigma^*, a \in \Sigma$.

Дефиниция 30.

1. Входният език на моноидния преобразувател $\mathcal{M}$ дефинираме като:

$$L_I(\mathcal{M}) = \{\sigma \in \Sigma^* \,|\, !\mu^*(s,\sigma) \;\&\; \mu^*(s,\sigma) \in F\}.$$

2. Изходният език на моноидния преобразувател $\mathcal{M}$ дефинираме като:

$$L_O(\mathcal{M}) = \{m \in M \,|\, \exists \sigma \in L_I(\mathcal{M}) \; (\lambda^*(s,\sigma) = m)\}.$$

Корегулярни изрази и езици можем да дефинираме подобно на регулярните. Нека е даден моноид $M = \langle M, \circ, \mathbf{e} \rangle$.

Дефиниция 31. Корегулярен израз над M представлява дума над азбуката $M \cup \{(,), *, +, \cdot, \emptyset\}$, дефинирана индуктивно:

1. $\emptyset$ е корегулярен израз;

2. ако $m \in M$ то m е корегулярен израз;

3. ако E_1 и E_2 са корегулярни изрази, то

- $(E_1 + E_2)$ е корегулярен израз,
- $(E_1 \cdot E_2)$ е корегулярен израз,
- E_1^* е корегулярен израз.

Дефиниция 32. Корегулярен език над моноида M дефинираме индуктивно:

1. $\emptyset$ е корегулярен език;
2. ако $m \in M$, то $\{m\}$ е корегулярен език;
3. ако R_1 и R_2 са корегулярни езици, то
   - $R_1 \cup R_2$ е корегулярен език (обединение),
   - $R_1 \circ R_2 = \{m_1 \circ m_2 \,|\, m_1 \in R_1, m_2 \in R_2\}$ е корегулярен език,
   - $R_1^* = \bigcup_{i=0}^{\infty} R_1^i$ е корегулярен език (затваряне на Клини).

В [Br84, Ми93] се показва, че хомоморфен образ (при моноиден хомоморфизъм) на регулярен език е корегулярен език. И обратно — всеки корегулярен език е хомоморфен образ на някой регулярен език. Също така хомоморфен образ на език на краен автомат е изходен език на моноиден преобразувател. И обратно — всеки изходен език на моноиден преобразувател е хомоморфен образ на някой език на краен автомат. Като се използват тези свойства и класическия вариант на теоремата на Клини, може да се покаже следният вариант на теоремата на Клини за моноидните преобразуватели и корегулярните езици.

Теорема 33 Kleene. *Подмножество на даден моноид е корегулярен език, тогава и само тогава когато е изходен език на моноиден преобразувател.*

Тук няма да приложим доказателства на горните свойства. (Те няма да се използват по-нататък.) По-подробно разглеждане на моноидните преобразуватели може да се намери в [Br84, Ми93].

По-нататък ще се интересуваме главно от случая, когато изходният моноид е $\mathbb{N} = \langle \mathbb{N}, +, 0 \rangle$. В този случай при разпознаване на дадена дума от автомата на изхода се получава сумата от числата, връщани от функцията λ при преминаване през преходите.

Ацикличен моноиден преобразувател, връщащ индекс на дума в списък с лексикографска подредба

Ацикличният детерминиран краен автомат описва краен списък от думи. Интересно е приложението на автоматите за индексиране на списъка от думи. Нека е дадена линейна наредба в азбуката Σ. Тя индуцира лексикографска подредба в Σ^*. Тогава обхождайки автомата първо в дълбочина, като избираме преходите по реда на буквите в азбуката, можем да изведем списъка от думи, които автоматът разпознава в лексикографска подредба.

Под индекс на дума в списък с лексикографска подредба ще разбираме поредния номер на думата в списъка, започвайки от 0.

В приложенията често възниква въпросът за построяване на автомат, връщащ индекса на разпозната дума в списъка, разпознаван от автомата. Едно възможно решение е използването на моноиден преобразувател. Следващата теорема ни дава начин за конструиране на такъв преобразувател.

Дефиниция 34. Нека е даден детерминиран краен автомат $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$. Нека дефинираме помощната функция $\mathit{fin} : S \to \mathbb{N}$ по следния начин:

$$\mathit{fin}(r) = \begin{cases} 1 \text{ ако } r \in F \\ 0 \text{ ако } r \notin F \end{cases}$$

Функцията fin се дефинира по същия начин и за моноиден преобразувател.

Теорема 35. *Нека е даден детерминиран краен автомат $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$. Тогава съществува моноиден преобразувател $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$, така че за всяка дума $\sigma \in L(\mathcal{A})$, $\lambda^*(\sigma)$ е нейният индекс в списъка на думите, описан от автомата $\mathcal{A}$.*

Доказателство. За удобство да положим $|L(\mathcal{A}|_{\mu(r,x)})| = 0$, в случай че $\mu(r, x)$ не е дефинирана. Построяваме функцията λ по следния начин:

$$\lambda(r, a) = \begin{cases} \mathit{fin}(r) + \sum_{a' < a} |L(\mathcal{A}|_{\mu(r,a')})|, \text{ ако } !\mu(r, a); \\ \text{не е дефинирана в противен случай} \end{cases} \tag{1.2}$$

Горната конструкция ни осигурява следното свойство: ако сме достигнали в даден възел с поддума σ, то има точно $\lambda^*(s, \sigma)$ думи пред поддумата σ в лексикографски подредения списък, описан от автомата. От това наблюдение директно следва теоремата. Доказателството на свойството ще проведем чрез индукция по дължината на думата σ.

Ако $|\sigma| = 0$, то $\sigma = \varepsilon$. Следователно $\lambda^*(s, \sigma) = 0$ и условието е тривиално изпълнено.

Допускаме, че за всяко $|\sigma| = n$ условието е изпълнено.

Нека $|\sigma| = n + 1$. Тогава $\sigma = \tau a$, където $\tau \in \Sigma^*, a \in \Sigma$ и $|\tau| = n$. От допускането получаваме, че в списъка L има точно $\lambda^*(s, \tau)$ думи пред τ. Нека $r = \mu^*(s, \tau)$. Тогава броят на думите в списъка, които са пред σ, ще бъде точно броят на думите пред τ плюс броят на думите от вида $\tau a'\psi$, където $a' \in \Sigma, a' < a, \psi \in \Sigma^*$ и евентуално плюс едно, ако $\tau \in L$. Но броят на думите, разпознавани от автомат от вида $\tau a'\psi$, е точно $|L(\mathcal{A}|_{\mu(r,a')})|$. По дефиниция имаме, че $\lambda^*(s, \sigma) = \lambda^*(s, \tau) + \lambda(r, a)$. В зависимост от това, дали $r \in F$, т.е. $\tau \in L$ от условие 1.2, получаваме $\lambda^*(s, \sigma) = \lambda^*(s, \tau) + \sum_{a'<a} |L(\mathcal{A}|_{\mu(r,a')})|$ или $\lambda^*(s, \sigma) = \lambda^*(s, \tau) + \sum_{a'<a} |L(\mathcal{A}|_{\mu(r,a')})| + 1$. Това точно отговаря на броя на думите пред σ в списъка L. $\square$

Един от възможните начини за построяване на моноиден преобразувател, връщащ индекс, е първо да се построи детерминиран краен автомат (по възможност минимален), след което рекурсивно да се дефинира функцията λ, така че да отговаря на условието 1.2 от горната теорема. По-нататък ще покажем директен метод за построяването на минимален автомат, връщащ индекса на разпознатите думи в списъка.

1.4 Минимален освен в дадена дума автомат

В тази секция ще въведем понятието минимален освен в дума автомат, като ще изложим най-важните му свойства. Това понятие е основната математическа концепция за представения в следващата секция метод за директно построяване на минимален автомат по даден списък. За първи път минималните освен в дадена дума автомати се разглеждат в [Mi98].

Ще предполагаме, че е дадена крайна азбука Σ с линейна наредба. По-нататък под наредба на думите в Σ^* ще разбираме лексикографската наредба, индуцирана от линейната наредба в Σ.

Дефиниция 36. Нека $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ е ацикличен детерминиран краен автомат с език $L(\mathcal{A})$. Тогава автомата $\mathcal{A}$ ще наричаме *минимален освен в думата* $\omega \in \Sigma^*$, когато са изпълнени следните условия:

1. Всяко състояние на автомата е достижимо от началното и от всяко състояние е достижимо някое крайно;

2. ω е префикс на последната дума в лексикографската подредба на $L(\mathcal{A})$;

   В този случай можем да въведем следната нотация:

$$\omega = w_1^{\mathcal{A}} w_2^{\mathcal{A}} \dots w_k^{\mathcal{A}} \text{ , където } w_i^{\mathcal{A}} \in \Sigma \text{ , за } i = 1, 2, \dots, k \tag{1.3}$$

$$t_0^{\mathcal{A}} = s \text{ ; } t_1^{\mathcal{A}} = \mu(t_0^{\mathcal{A}}, w_1^{\mathcal{A}}) \text{ ; } t_2^{\mathcal{A}} = \mu(t_1^{\mathcal{A}}, w_2^{\mathcal{A}}) \text{ ; } \dots \text{ ; } t_k^{\mathcal{A}} = \mu(t_{k-1}^{\mathcal{A}}, w_k^{\mathcal{A}}) \tag{1.4}$$

$$T = \{t_0^{\mathcal{A}}, t_1^{\mathcal{A}}, \dots, t_k^{\mathcal{A}}\} \tag{1.5}$$

3. В множеството $S \setminus T$ няма различни еквивалентни състояния;

4. $\forall r \in S \; \forall i \in \{1, 2, \dots, k\} \; \forall a \in \Sigma \; (\mu(r, a) \cong t_i \leftrightarrow (i > 0 \;\&\; r = t_{i-1} \;\&\; a = w_i^{\mathcal{A}}))$.

По-нататък когато работим с минимален освен в дадена дума автомат, ще използваме нотациите (1.3),(1.4) и (1.5), въведени в горната дефиниция. В случай, че това не води до двусмислие, ще пишем t_i, w_i вместо $t_i^{\mathcal{A}}, w_i^{\mathcal{A}}$.

Ясно е, че ако автомат е минимален освен в две различни думи, то едната е префикс на другата.

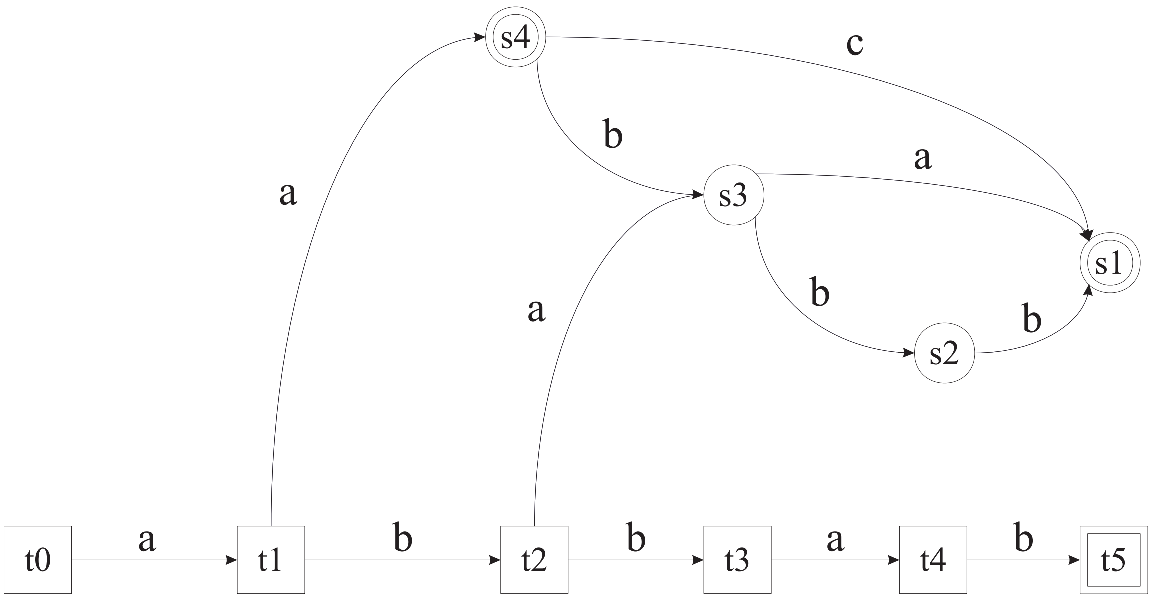
Свойство 37. *Нека автоматът $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ е минимален освен в ω. Тогава:*

1. $\forall r \in S \setminus T \; \forall a \in \Sigma \; (!\mu(r, a) \to \mu(r, a) \in S \setminus T)$;

2. $\mu^*(s, \sigma) \cong t_i \leftrightarrow \sigma = w_1 w_2 \dots w_i$.

Доказателството на горното свойство е директно следствие на дефиницията 36.

Пример 1. На фигура 1.1 е даден ацикличен краен автомат над азбуката $\{a, b, c\}$. Езикът на автомата е $\{aa, aac, aaba, aabbb, abaa, ababb, abbab\}$. Този автомат е минимален освен в $abbab$ (при използване на обичайната наредба на азбуката).



Фигура 1.1: Краен автомат минимален освен в $abbab$.

Свойство 38. *Ако един автомат е минимален освен в празната дума ε, то той е минимален.*

Доказателство. От дефиницията за минимален освен в празната дума автомат получаваме, че всяко състояние е достижимо от началното, от всяко състояние е достижимо някое крайно и в множеството $S \setminus \{s\}$ няма различни еквивалентни състояния. За да изпълним условието на теорема 8, трябва да покажем, че в автомата няма друго състояние еквивалентно на s.

Да допуснем, че $r \in S$ и s са еквивалентни и $r \neq s$. Нека ω е най-дълга дума, разпознавана от автомата. Съществуването ѝ следва от това, че езикът на автомата е краен. От допускането, че r и s са еквивалентни следва, че $\omega \in L(\mathcal{A}|_r)$. Състоянието r е достижимо от s. Следователно съществува $\sigma \in \Sigma^+$ и $\mu^*(s, \sigma) \cong r$. Тогава думата $\sigma\omega \in L(\mathcal{A})$ и от $\sigma \neq \varepsilon$ получаваме $|\sigma\omega| > |\omega|$. Това противоречи на факта, че ω е най-дълга дума в езика. □

Лема 39. *Нека автоматът $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ е минимален освен в думата $\omega = w_1 w_2 \ldots w_k$, $\omega \neq \varepsilon$. Нека няма състояние еквивалентно на t_k в множеството $S \setminus T$. Тогава $\mathcal{A}$ е минимален също освен в думата $\omega' = w_1 w_2 \ldots w_{k-1}$.*

Доказателство. Трябва да проверим, че са изпълнени условията на дефиниция 36 за минималност освен в ω'. Условията 1 и 2 са тривиално изпълнени. Условието 3 следва от условието на лемата, че в $S \setminus \{t_0, t_1, \ldots, t_k\}$ няма състояние еквивалентно на t_k. Условие 4 е изпълнено, защото само стеснява областта на квантора от условието 4 от дефиницията за минималност освен в ω. □

Лема 40. *Нека автоматът* $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ *е минимален освен в думата* $\omega = w_1 w_2 \ldots w_k$, $\omega \neq \varepsilon$. *Нека състоянието* $p \in S \setminus T$ *е еквивалентно на състоянието* t_k. *Нека автоматът* $\mathcal{A}' = \langle \Sigma, S', s, F', \mu' \rangle$ *е дефиниран както следва:*

$$S' = S \setminus \{t_k\}$$

$$F' = F \setminus \{t_k\}$$

$$\mu'(r, a) = \begin{cases} \mu(r, a) & \text{, в случай, че } r \neq t_{k-1} \vee a \neq w_k \text{ и } !\mu(r, a) \\ p & \text{, в случай, че } r = t_{k-1}, a = w_k \\ \text{не е дефинирано} & \text{в противен случай} \end{cases}.$$

Тогава $\mathcal{A}'$ *е еквивалентен на автомата* $\mathcal{A}$ *и е минимален освен в* $\omega' = w_1 w_2 \ldots w_{k-1}$.

Доказателство. Първо трябва да покажем, че $\mathrm{range}(\mu') \subset S'$. Това следва от това, че единственият преход към t_k в $\mathcal{A}$ е от състоянието t_{k-1} с буквата w_k (виж условие 4 на дефиницията 36). В дефиницията на μ' този преход е пренасочен към $p \in S'$.

Автоматът $\mathcal{A}$ е еквивалентен на $\mathcal{A}'$, защото новият автомат се получава чрез премахване на състоянието t_k и единственият преход към t_k се пренасочва към преход към еквивалентно състояние. Условията 1 и 2 от дефиницията за минималност освен в ω' на $\mathcal{A}'$ са тривиално изпълнени. Условието 3 следва от равенството:

$$S' \setminus \{t_0, t_1, \ldots, t_{k-1}\} = (S \setminus \{t_k\}) \setminus \{t_0, t_1, \ldots, t_{k-1}\} = S \setminus \{t_0, t_1, \ldots, t_k\}.$$

За да проверим условието 4, трябва да използваме условие 4 от дефиницията за минималност освен в ω и условието $\mu'(t_{k-1}, w_k) \cong p \in S \setminus T$. □

Теорема 41. *Нека автоматът $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ е минимален освен в $\omega' = w_1 w_2 \dots w_m$. Нека $\psi \in L(\mathcal{A})$ е последната дума в лексикографската подредба на езика на автомата. Нека $\omega \in \Sigma^*$ е дума, която е по-голяма по лексикографски ред от думата ψ. Нека ω' е най-дългият общ префикс на думите ψ и ω. В този случай да означим $\omega = w_1 w_2 \dots w_m w_{m+1} \dots w_k; k > m$. Нека автоматът $\mathcal{A}' = \langle \Sigma, S', s, F', \mu' \rangle$ е дефиниран както следва:*

$$t_{m+1}, t_{m+2}, \dots, t_k \text{ са нови състояния такива, че } S \cap \{t_{m+1}, t_{m+2}, \dots, t_k\} = \emptyset$$

$$S' = S \cup \{t_{m+1}, t_{m+2}, \dots, t_k\}$$

$$F' = F \cup \{t_k\}$$

$$\mu'(r,a) = \begin{cases} t_{i+1} & \text{, в случай, че } r = t_i, m \leq i \leq k-1, a = w_{i+1} \\ \mu(r,a) & \text{, в случай, че } r \in S \text{ и } !\mu(r,a) \text{и } r \neq t_m \vee a \neq w_{m+1} \\ \text{не е дефинирано} & \text{в противен случай} \end{cases} .$$

Тогава $\mathcal{A}'$ е минимален освен в ω и разпознава езика $L(\mathcal{A}) \cup \{\omega\}$.

Доказателство. Първо ще покажем, че $L(\mathcal{A}') = L(\mathcal{A}) \cup \{\omega\}$. Трябва да покажем, че автоматът $\mathcal{A}'$ съдържа автомата $\mathcal{A}$. Ясно е, че $S' \supset S$ и $F' \supset F$. Остава да покажем, че $\mu' \supset \mu$. От дефиницията на μ' е ясно, че единствено случаят $\mu'(t_m, w_{m+1})$ е нетривиален. Но $\mu(t_m, w_{m+1})$ не е дефинирано, защото ω' е най-дълъг общ префикс на ψ и ω и ω по-голям по лексикографски ред от ψ – последната дума по лексикографски ред на езика на автомата $\mathcal{A}$. На това място използването на лексикографския ред на ω е съществено.

Следователно получаваме, че $L(\mathcal{A}') \supset L(\mathcal{A})$. Нови думи могат да се получат само при преминаването през новите състояния. От дефиницията на μ' получаваме, че те са достижими единствено от t_m. Единствената дума в езика $L(\mathcal{A}'|_{t_m})$ е $w_{m+1} w_{m+2} \dots w_k$. От свойство 37 имаме, че t_m е достижимо от s само чрез думата ω'. Следователно единствената нова дума в $L(\mathcal{A}')$ е думата $\omega' w_{m+1} w_{m+2} \dots w_k = \omega$.

Трябва да проверим, че $\mathcal{A}'$ е минимален освен в ω. Нека разгледаме условията на дефиниция 36 за минималност на $\mathcal{A}'$ освен в ω. Условията 1, 2 и 3 са тривиално изпълнени. Условието 4 е изпълнено за $t_0, t_1, \dots, t_{m-1}$ според условие 4 от дефиницията за минималност освен в ω' на автомата $\mathcal{A}$, за състоянията $t_m, t_{m+1}, \dots, t_k$ условието е директно изпълнено от дефиницията на μ'. □

Лема 42. *Нека детерминираният краен автомат $\mathcal{A} = \langle \Sigma, S, s, F, \mu \rangle$ е минимален освен в $\omega = w_1 w_2 \dots w_k$. Тогава за t_k (виж нотацията, въведена в дефиниция 36) и произволно $r \in S \setminus T$ е в сила следното твърдение*[1]*:*

$$
\begin{gathered}
t_k \text{ е еквивалентно на } r \leftrightarrow \\
((t_k \in F \leftrightarrow r \in F)\ \& \\
\forall a \in \Sigma\ ((\neg !\mu(t_k, a)\ \&\ \neg !\mu(r, a)) \vee (!\mu(t_k, a)\ \&\ !\mu(r, a)\ \&\ \mu(t_k, a) = \mu(r, a))).
\end{gathered}
$$

Доказателство. Състоянията t_k и r са еквивалентни, тогава и само тогава когато и двете са крайни или и двете са некрайни и чрез еднакви букви от тях се преминава към еквивалентни състояния, или не са дефинирани. От факта, че всички преходи от t_k са към състояния в $S \setminus T$, всички преходи от $r \in S \setminus T$ са към състояния в $S \setminus T$ и че в $S \setminus T$ няма еквивалентни състояния следва, че две състояния, към които се преминава от t_k и r, са еквивалентни, тогава и само тогава когато съвпадат. □

1.5 Директно построяване на минимален автомат за даден списък

Решаването на задачата за създаване на минимален автомат е важно за приложението на крайните автомати. До момента за построяване на минимален автомат, разпознаващ даден списък от думи, най-ефективен е методът на Revuz, представен в [Re91, Re92, Ro92]. При този метод се строи първо детерминиран краен автомат с дървовидна структура, който след това се минимизира. Тук ще представим един директен метод, изложен за първи път в [Mi98], който се основава на показаните в предишната секция свойства и строи директно минималния автомат, представящ дадения списък. Ще илюстрираме действието му с един пример.

[1] Това твърдение означава, че за да проверим еквивалентността между t_k и някое състояние в $S \setminus T$ е достатъчно да проверим че и двете имат еднаква крайност и от тя се преминава с еднакви букви към едни и същи състояния.

Нека е даден непразен списък от думи L в лексикографска подредба. Нека с $\omega^{(i)}$ означим i-тата дума от списъка. Ще започнем с минималния автомат, разпознаващ точно първата дума от списъка. Този автомат се строи тривиално и е също минимален освен в $\omega^{(1)}$. Използвайки го за база, провеждаме индукция по думите в списъка. Нека предположим, че автоматът $\mathcal{A}^{(n)} = \langle \Sigma, S, s, F, \mu \rangle$ с език $L^{(n)} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n\}$ е построен и че $\mathcal{A}^{(n)}$ е минимален освен в $\omega^{(n)}$. Трябва да построим автомат $\mathcal{A}^{(n+1)}$ с език $L^{(n+1)} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n+1\}$, който е минимален освен в $\omega^{(n+1)}$.

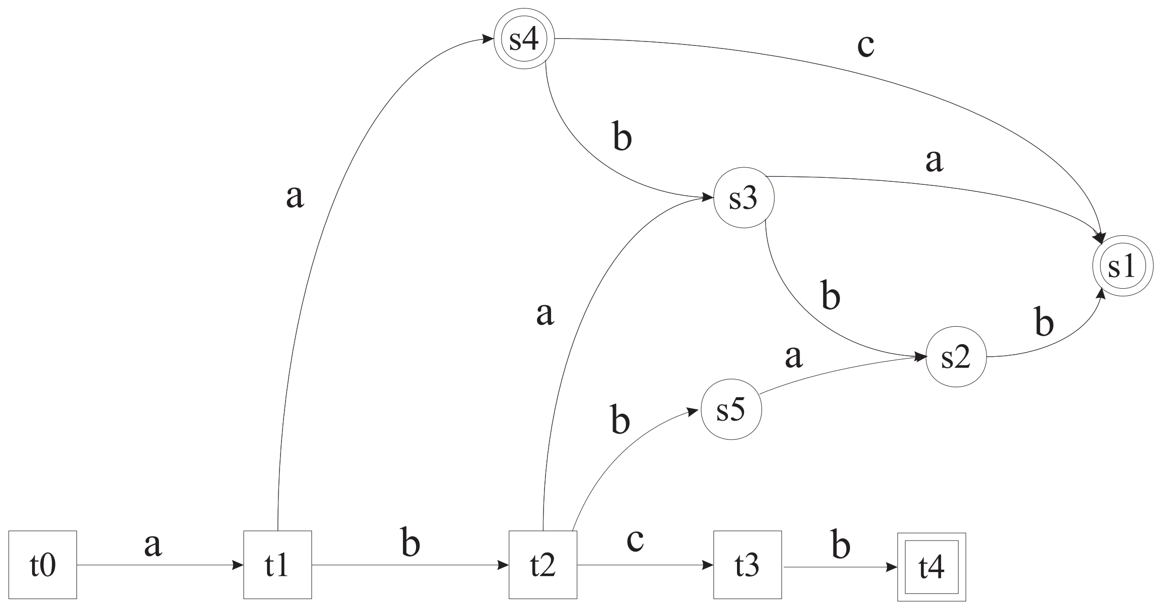
Нека ω' е най-дългият общ префикс на думите $\omega^{(n)}$ и $\omega^{(n+1)}$. Използвайки няколко пъти лема 39 и лема 40 (в съответствие със случая) построяваме автомата $\mathcal{A}' = \langle \Sigma, S', s, F', \mu' \rangle$, който е еквивалентен на $\mathcal{A}^{(n)}$ и е минимален освен в ω'. Сега можем да използваме теорема 41 и да построим автомата $\mathcal{A}^{(n+1)}$ с език $L^{(n+1)} = L^{(n)} \cup \{\omega^{(n+1)}\} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n+1\}$, който е минимален освен в $\omega^{(n+1)}$.

По този начин индуктивно построяваме минималния освен в последната дума от списъка автомат, който разпознава списъка L. Накрая отново, използвайки неколкократно лема 39 и лема 40, построяваме еквивалентен автомат, минимален освен в ε. От свойство 38 имаме, че той е минималният автомат за списъка L. За да разграничаваме условията за прилагането на лема 39 и лема 40, трябва да търсим състояние, еквивалентно на t_k в $S \setminus T$. Това става ефективно чрез използването на лема 42. $\square$

При построяването на автомата броят на състоянията на междинния автомат е не по-голям от броя на състоянията на резултатния минимален автомат плюс дължината на най-дългата дума в списъка. Това е най-същественото предимство на нашия метод спрямо прилаганите досега.

Забележка. Тук е уместен въпросът защо използваме автомат минимален освен в дадена дума. Не е ли възможно да процедираме по следния по-прост начин – ако на всяка стъпка сме построили автомата, разпознаващ списъка до тук: (i) чрез следващата дума от списъка да преминаваме през преходите на автомата, докъдето това е възможно; (ii) от достигнатото състояние да се добави верига от състояния, за да се разпознае новата дума; (iii) накрая като се търсят еквивалентни на новите състояния в обратен ред, да се построи минимален автомат.

Тази процедура в общия случай няма да построи коректния минимален автомат, защото дадено състояние по пътя може да е достижимо с различни думи. По този начин добавянето на веригата от състояния към даденото състояние на минималния автомата може да доведе до автомат, който разпознава повече от една нова дума. Следователно резултатният автомат ще разпознава и думи, които не са от списъка.



Фигура 1.2: Краен автомат минимален освен в $abcb$.

Пример 2. За да илюстрираме нашия метод, да разгледаме следния пример. На фигура 1.1 е даден автоматът, разпознаващ езика $\{aa, aac, aaba, aabbb, abaa, ababb, abbab\}$. Този автомат е минимален освен в последната дума от списъка — $abbab$. Нека следващата дума е $abcb$. Най-дългият общ префикс на двете думи е ab. Трябва първо да построим автомат, еквивалентен на този, показан на фигура 1.1, който е минимален освен в ab, като използваме лема 39 и лема 40. Първо прилагаме два пъти лема 40, след което веднъж лема 39. Накрая използваме теорема 41 и конструираме автомата минимален освен в $abcb$ даден на фигура 1.2. По този начин добавихме следващата дума от списъка към езика на автомата.

В секция 2.3 е представен алгоритъмът, прилагащ този метод.

Директно построяване на минимален краен автомат с етикети на крайните състояния, разпознаващ и класифициращ даден списък

Както вече споменахме, върху крайните автомати с етикети на крайните състояния могат да се приложат варианти на някои от класическите методи, приложими върху крайните автомати. Тук ще покажем как директният метод за построяване на краен автомат за даден списък може да се приложи за построяване на краен автомат с етикети на крайните състояния. За целта трябва да разгледаме как се обобщават понятията и резултатите за минимален освен в дадена дума автомат в случая – на автомат с етикети. По-долу излагаме съответните дефиниции и резултати в случая на автомат с етикети.

В случая на автомат с етикети дефиниция 36 остава непроменена, като трябва да се вземе предвид обобщаването на понятието еквивалентни състояния (виж дефиниция 15).

Дефиниция 43. Нека $\mathcal{A} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$ е ацикличен краен автомат с етикети на крайните състояния с език $L(\mathcal{A})$. Тогава автомата $\mathcal{A}$ ще наричаме *минимален освен в думата* $\omega \in \Sigma^*$, когато са изпълнени условия 1, 2, 3 и 4 от дефиниция 36.

Свойство 44. *Ако един автомат с етикети е минимален освен в празната дума* ε, *то той е минимален.*

Лема 45. *Нека автоматът с етикети* $\mathcal{A} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$ *е минимален освен в* $\omega = w_1 w_2 \dots w_k$, $\omega \neq \varepsilon$. *Нека няма състояние еквивалентно на* t_k *в множеството* $S \setminus T$. *Тогава* $\mathcal{A}$ *е минимален също освен в думата* $\omega' = w_1 w_2 \dots w_{k-1}$.

Лема 46. *Нека автоматът с етикети* $\mathcal{A} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$ *е минимален освен в* $\omega = w_1 w_2 \dots w_k$, $\omega \neq \varepsilon$. *Нека състоянието* $p \in S \setminus T$ *е еквивалентно на състоянието* t_k. *Тогава автоматът с етикети* $\mathcal{A}' = \langle \Sigma, E, S', s, F', \mu', \psi' \rangle$, *дефиниран както следва:*

$$S' = S \setminus \{t_k\}$$

$$F' = F \setminus \{t_k\}$$

$$\mu'(r,a)=\begin{cases}\mu(r,a) & \text{, в случай, че } r\neq t_{k-1}\vee a\neq w_k \text{ и } !\mu(r,a)\\ p & \text{, в случай, че } r=t_{k-1}, a=w_k\\ \text{не е дефинирано} & \text{в противен случай}\end{cases},$$

$$\psi'=\psi|_{F'}$$

е еквивалентен на автомата $\mathcal{A}$ *и е минимален освен в* $\omega'=w_1w_2\dots w_{k-1}$.

Теорема 47. *Нека автоматът е етикети* $\mathcal{A}=\langle\Sigma,E,S,s,F,\mu,\psi\rangle$ *е минимален освен в* $\omega'=w_1w_2\dots w_m$. *Нека* $\tau\in L(\mathcal{A})$ *е последната дума в лексикографската подредба на езика на автомата. Нека* $\omega\in\Sigma^*$ *е дума, която е по-голяма по лексикографски ред от думата* τ. *Нека* ω' *е най-дългият общ префикс на думите* τ *и* ω. *В този случай да означим* $\omega=w_1w_2\dots w_mw_{m+1}\dots w_k; k>m$. *Нека думата* ω *се класифицира в* $l\in E$ *Тогава автоматът с етикети* $\mathcal{A}'=\langle\Sigma,E,S',s,F',\mu',\psi'\rangle$, *дефиниран както следва:*

$$t_{m+1},t_{m+2},\dots,t_k \text{ са нови състояния такива, че } S\cap\{t_{m+1},t_{m+2},\dots,t_k\}=\emptyset$$

$$S'=S\cup\{t_{m+1},t_{m+2},\dots,t_k\}$$

$$F'=F\cup\{t_k\}$$

$$\mu'(r,a)=\begin{cases}t_{i+1} & \text{, в случай, че } r=t_i, m\leq i\leq k-1, a=w_{i+1}\\ \mu(r,a) & \text{, в случай, че } r\in S \text{ и } !\mu(r,a) \text{ и } r\neq t_m\vee a\neq w_{m+1}\\ \text{не е дефинирано} & \text{в противен случай}\end{cases}$$

$$\psi'(r)=\begin{cases}\psi(r) & \text{, в случай, че } r\in F\\ l & \text{, в случай, че } r=t_k\\ \text{не е дефинирана} & \text{в противен случай}\end{cases},$$

е минимален освен в ω *и разпознава езика* $L(\mathcal{A})\cup\{\omega\}$, $e_{\mathcal{A}'}|F=e_{\mathcal{A}}$ *и* $e_{\mathcal{A}'}(\omega)=l$.

Лема 48. *Нека крайният автомат с етикети* $\mathcal{A}=\langle\Sigma,E,S,s,F,\mu,\psi\rangle$ *е минимален освен в* $\omega=w_1w_2\dots w_k$. *Тогава за* t_k *(виж нотацията въведена в дефиниция 43) е в сила следното твърдение:*

$$t_k \text{ е еквивалентно на } r\in S\setminus T\leftrightarrow$$

$$((t_k\in F\leftrightarrow r\in F)\ \&\ (t_k\in F\rightarrow\psi(t_k)=\psi(r))\ \&$$

$$\forall a\in\Sigma\ ((\neg!\mu(t_k,a)\ \&\ \neg!\mu(r,a))\vee(!\mu(t_k,a)\ \&\ !\mu(r,a)\ \&\ \mu(t_k,a)=\mu(r,a))).$$

Доказателствата на горните резултати се получават от доказателствата на съответните резултати в предишните секции чрез минимални допълнения, като трябва да се държи сметка за класификационната функция.

Нека сега представим метода за директно построяване на краен автомат с етикети по даден списък.

Нека е даден непразен списък от думи L в лексикографска подредба. Нека е дадена класификация на думите от L. Нека с $\omega^{(i)}$ означим i-тата дума от списъка. Ще започнем с минималния автомат с етикети, разпознаващ и класифициращ точно първата дума от списъка. Този автомат с етикети се строи тривиално и е също минимален освен в $\omega^{(1)}$. Използвайки го за база, провеждаме индукция по думите в списъка. Нека предположим, че автоматът с етикети $\mathcal{A}^{(n)} = \langle \Sigma, E, S, s, F, \mu, \psi \rangle$ с език $L^{(n)} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n\}$, класифициращ думите според дадената класификация, е построен и че $\mathcal{A}^{(n)}$ е минимален освен в $\omega^{(n)}$. Трябва да построим автомат $\mathcal{A}^{(n+1)}$ с език $L^{(n+1)} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n+1\}$, класифициращ правилно думите, който е минимален освен в $\omega^{(n+1)}$.

Нека ω' е най-дългият общ префикс на думите $\omega^{(n)}$ и $\omega^{(n+1)}$. Използвайки няколко пъти лема 45 и лема 46 (в съответствие със случая), построяваме автомата с етикети $\mathcal{A}' = \langle \Sigma, E, S', s, F', \mu', \psi' \rangle$, който е еквивалентен на $\mathcal{A}^{(n)}$ и е минимален освен в ω'. Сега можем да използваме теорема 47 и да построим автомата $\mathcal{A}^{(n+1)}$ с език $L^{(n+1)} = L^{(n)} \cup \{\omega^{(n+1)}\} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n+1\}$, който е минимален освен в $\omega^{(n+1)}$ и класифицира $\omega^{(n+1)}$ според дадената класификация.

По този начин индуктивно построяваме минималния освен в последната дума от списъка автомат с етикети, който разпознава и класифицира списъка L според дадената класификация. Накрая, отново използвайки неколкократно лема 45 и лема 46, построяваме еквивалентен автомат, минимален освен в ε. От свойство 44 имаме, че той е минималният автомат с етикети за списъка L с дадената класификация. За да разграничаваме условията за прилагането на лема 45 и лема 46, трябва да търсим състояние, еквивалентно на t_k в $S \setminus T$. Това става ефективно чрез използването на лема 48. $\square$

Алгоритъмът, съответстващ на този метод, е представен в секция 2.2.

Директно построяване на минимален моноиден преобразувател, връщащ индекса на думите в даден списък

Моноидният преобразувател представлява детерминиран краен автомат, в който е дефинирана функцията λ, съпоставяща на всеки преход елемент от моноида. Поради това за построяването на преобразувателя, връщащ индекса на дума в списък, можем първо да построим минималния краен автомат, който разпознава списъка от думи, използвайки директния метод, представен в началото на тази секция. След това трябва отново да обходим автомата, като се движим от крайните състояния към началното и да определим функцията λ според даденото в теорема 35 условие (1.2).

Съществува също така и по-директен метод, който ще изложим по-долу. Идеята се състои в това да дефинираме функцията λ по време на построяването на минималния автомат. По този начин не се налага допълнителното обхождане на автомата.

За целта отново ще използваме представения по-горе метод, като към него ще направим добавки.

Дефиниция 49. Нека е даден моноиден преобразувател $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$. Тогава:

1. Преобразувателя $\mathcal{M}$ ще наричаме минимален, ако детерминираният краен автомат $\mathcal{A}_{\mathcal{M}} = \langle \Sigma, S, s, F, \mu \rangle$ е минимален.

2. Състоянията $s_1, s_2 \in S$ на моноидния преобразувател наричаме еквивалентни, когато $\mathcal{A}_{\mathcal{M}}|_{s_1}$ е еквивалентен на $\mathcal{A}_{\mathcal{M}}|_{s_2}$;

Дефиницията за ацикличност и достижимост на състоянията на моноиден преобразувател съвпада с тази при краен автомат. Нека за облекчение на изложението да положим $\lambda(r, a) = 0$, ако $\neg ! \mu(r, a)$.

Дефиниция 50. Нека $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$ е ацикличен моноиден преобразувател с входен език $L_I(\mathcal{A})$. Тогава преобразувателя $\mathcal{M}$ ще наричаме *минимален освен в думата* $\omega \in \Sigma^*$ връщащ индекс, когато са изпълнени условията 1, 2, 3 и 4 от дефиниция 36 и още следните допълнителни условия:

5. За всяко състояние $r \in S \setminus T$ е изпълнено условието (1.2) от теорема 35 т.е.:
$$\lambda(r,a) = \begin{cases} fin(r) + \sum_{a'<a} |L(\mathcal{A}|_{\mu(r,a')})|, \text{ ако } !\mu(r,a); \\ \text{не е дефинирана в противен случай.} \end{cases}$$

6. За състоянията $t_i \in T$ е изпълнено условието:
$$\lambda(t_i,a) = \begin{cases} |L(\mathcal{A}|_{\mu(t_i,a)})|, \text{ ако } \mu(t_i,a) \in S \setminus T; \\ \text{не е дефинирана в противен случай.} \end{cases}$$

Свойство 51. *Ако един преобразувател, връщащ индекс $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$, е минимален освен в празната дума ε, то преобразувателят, връщащ индекс, $\mathcal{M}' = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda' \rangle$, където:*
$$\lambda'(r,a) = \begin{cases} \lambda(r,a), \text{ ако } r \in S \setminus \{s\} \\ fin(s) + \sum_{a'<a} \lambda(s,a'), \text{ ако } r = s \ \& \ !\mu(s,a); \\ \text{не е дефинирана в противен случай} \end{cases},$$
е минимален и връща индекса на думите в списъка, който разпознава.

Лема 52. *Нека преобразувателят, връщащ индекс $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$ е минимален освен в $\omega = w_1 w_2 \dots w_k$, $\omega \neq \varepsilon$. Нека няма състояние, еквивалентно на t_k в множеството $S \setminus T$. Тогава преобразувателят $\mathcal{M}' = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda' \rangle$, където:*
$$\lambda'(r,a) = \begin{cases} fin(t_k) + \sum_{a'<a} \lambda(t_k,a'), \text{ ако } r = t_k \text{ и } !\mu(s,a); \\ fin(t_k) + \sum_{a'\in\Sigma} \lambda(t_k,a'), \text{ ако } r = t_{k-1} \text{ и } a = w_k; \\ \lambda(r,a) \text{ в противен случай} \end{cases},$$
е минимален освен в думата $\omega' = w_1 w_2 \dots w_{k-1}$ връщащ индекс.

Лема 53. *Нека преобразувателят, връщащ индекс $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$ е минимален освен в $\omega = w_1 w_2 \dots w_k$, $\omega \neq \varepsilon$. Нека състоянието $p \in S \setminus T$ е еквивалентно на състоянието t_k. Тогава преобразувателят $\mathcal{M}' = \langle \Sigma, \mathbb{N}, S', s, F', \mu', \lambda' \rangle$, дефиниран както следва:*
$$S' = S \setminus \{t_k\}$$
$$F' = F \setminus \{t_k\}$$

$$\mu'(r,a) = \begin{cases} \mu(r,a) & \text{, в случай, че } r \neq t_{k-1} \vee a \neq w_k \text{ и } !\mu(r,a) \\ p & \text{, в случай, че } r = t_{k-1}, a = w_k \\ \text{не е дефинирано} & \text{в противен случай} \end{cases}$$

$$\lambda'(r,a) = \begin{cases} fin(t_k) + \sum_{a' \in \Sigma} \lambda(t_k, a'), & \text{ако } r = t_k - 1 \text{ и } a = w_k; \\ \lambda(r,a) \text{ в противен случай} \end{cases},$$

е еквивалентен на преобразувателя $\mathcal{M}$ *и е минимален освен в думата* $\omega' = w_1 w_2 \dots w_{k-1}$ *връщащ индекс преобразувател.*

Теорема 54. *Нека преобразувателят, връщащ индекс* $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$, *е минимален освен в* $\omega' = w_1 w_2 \dots w_m$. *Нека* $\psi \in L_I(\mathcal{A})$ *е последната дума в лексикографската подредба на езика на автомата. Нека* $\omega \in \Sigma^*$ *е дума, която е по-голяма по лексикографски ред от думата* ψ. *Нека* ω' *е най-дългият общ префикс на думите* ψ *и* ω. *В този случай да означим* $\omega = w_1 w_2 \dots w_m w_{m+1} \dots w_k$; $k > m$. *Тогава преобразувателят* $\mathcal{M}' = \langle \Sigma, \mathbb{N}, S', s, F', \mu', \lambda \rangle$, *дефиниран както следва:*

$$t_{m+1}, t_{m+2}, \dots, t_k \text{ са нови състояния такива, че } S \cap \{t_{m+1}, t_{m+2}, \dots, t_k\} = \emptyset$$

$$S' = S \cup \{t_{m+1}, t_{m+2}, \dots, t_k\}$$

$$F' = F \cup \{t_k\}$$

$$\mu'(r,a) = \begin{cases} t_{i+1} & \text{, в случай, че } r = t_i, m \leq i \leq k-1, a = w_{i+1} \\ \mu(r,a) & \text{, в случай, че } r \in S \text{ и } !\mu(r,a) \text{ и } r \neq t_m \vee a \neq w_{m+1} \\ \text{не е дефинирано} & \text{в противен случай} \end{cases},$$

е минимален освен в ω *връщащ индекс преобразувател, разпознаващ езика* $L_I(\mathcal{A}) \cup \{\omega\}$.

Лема 55. *Нека преобразувателят, връщащ индекс* $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$, *е минимален освен в* $\omega = w_1 w_2 \dots w_k$ *връщащ индекс. Тогава за* t_k *е в сила следното твърдение:*

$$t_k \text{ е еквивалентно на } r \in S \setminus T \leftrightarrow$$

$$((t_k \in F \leftrightarrow r \in F)\ \&$$

$$\forall a \in \Sigma\ ((\neg!\mu(t_k,a)\ \&\ \neg!\mu(r,a)) \vee (!\mu(t_k,a)\ \&\ !\mu(r,a)\ \&\ \mu(t_k,a) = \mu(r,a))).$$

Горните свойства на моноидните преобразуватели се доказват подобно на съответните при крайните автомати, като трябва да се спазват условията за функцията λ.

Методът за директно построяване на минимален преобразувател, връщащ индекс на думите, е повторение на схемата, изложена за краен автомат, като разликата се състои в използването на новите свойства.

Нека е даден непразен списък от думи L в лексикографска подредба. Нека с $\omega^{(i)}$ означим i-тата дума от списъка. Ще започнем с минималния преобразувател, разпознаващ точно първата дума от списъка, като в него функцията λ е дефинирана произволно. Този автомат се строи тривиално и е също минимален освен в $\omega^{(1)}$ връщащ индекс преобразувател. Използвайки го за база, провеждаме индукция по думите в списъка. Нека предположим, че преобразувателят $\mathcal{M}^{(n)} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$ с език $L_I^{(n)} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n\}$ е построен и че $\mathcal{M}^{(n)}$ е минимален освен в $\omega^{(n)}$. Трябва да построим преобразувател, връщащ индекс $\mathcal{M}^{(n+1)}$ с език $L_I^{(n+1)} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n+1\}$, който е минимален освен в $\omega^{(n+1)}$.

Нека ω' е най-дългият общ префикс на думите $\omega^{(n)}$ и $\omega^{(n+1)}$. Използвайки няколко пъти лема 52 и лема 53 (в съответствие със случая), построяваме преобразувател, връщащ индекс $\mathcal{M}' = \langle \Sigma, \mathbb{N}, S', s, F', \mu', \lambda' \rangle$, който е еквивалентен на $\mathcal{A}^{(n)}$ и е минимален освен в ω'. Сега можем да използваме теорема 54 и да построим преобразувателя, връщащ индекс $\mathcal{M}^{(n+1)}$ с език $L_I^{(n+1)} = L^{(n)} \cup \{\omega^{(n+1)}\} = \{\omega^{(i)} \,|\, i = 1, 2, \ldots, n+1\}$, който е минимален освен в $\omega^{(n+1)}$.

По този начин индуктивно построяваме минималния освен в последната дума от списъка преобразувател, връщащ индекс, който разпознава списъка L. Накрая отново използвайки неколкократно лема 52 и лема 53, построяваме еквивалентен преобразувател минимален освен в ε. Използвайки свойство 51, построяваме минималния преобразувател за списъка L, връщащ индекса на думите. За да разграничаваме условията за прилагането на лема 52 и лема 53, трябва да търсим състояние, еквивалентно на t_k в $S \setminus T$. Това става ефективно чрез използването на лема 55. $\square$

Алгоритъмът, съответстващ на този метод, е даден в секция 2.3.

1.6 Директно построяване на минимален автомат, представящ сечение, обединение или разлика на ациклични детерминирани крайни автомати

В тази секция ще разгледаме приложение на метода за директно построяване на минимален автомат по даден списък за директно построяване на минимален автомат, представящ сечение, обединение или разлика на ациклични крайни автомати.

Стандартните методи за построяване на обединение, сечение или разлика на автомати се основават на построяването на междинен автомат, чиито състояния са декартово произведение на състоянията на изходните автомати. Този автомат в повечето случай е огромен в сравнение с минималния автомат. Поради това е целесъобразно създаването на по-ефективни методи. По-долу ще представим нашия метод, който е приложим за ациклични крайни автомати и построява директно минималния автомат. Първото излагане на този метод е в [Mi99].

Както вече отбелязахме на страница 33, обхождайки един ацикличен краен автомат първо в дълбочина, като избираме преходите от дадено състояние по лексикографския ред на буквите, можем да изведем списъка от думи в лексикографски ред. Използвайки това свойство, чрез комбинирането списъците от думи, извеждани от изходните автомати, можем да създадем списък от думи, който е обединение, сечение или разлика на думите от изходните списъци. При това можем да запазим лексикографската подредба. Чрез използването на новополучения списък като входен при метода за директно построяване на минимален автомат по даден списък получаваме минималния автомат, представящ обединение, сечение или разлика на изходните автомати. При това не се построяват други междинни автомати. Освен това на практика може да се приложи методът, като се използва само поредната дума от списъка, без да се налага експлицитното построяване на списъка от думи. Формално съответният алгоритъм е представен в секция 2.5.

1.7 Директно построяване на минимален нормализиран ацикличен n-лентов краен автомат

Тук ще разгледаме как методите за директно построяване на минимални ациклични автомати могат да се прилагат за построяване на минимални ациклични n-лентови крайни автомати.

За целта ще използваме следния подход. Елементите на $(\Sigma^\varepsilon)^n$ разглеждаме като отделни символи и върху тях дефинираме произволна линейна наредба. Това можем да направим, тъй като Σ е крайна азбука. Тогава всеки n-лентов краен автомат ще можем да разглеждаме като класически недетерминиран краен автомат. Нека представим по-формално определенията. За целта до края на тази секция с $\circ$ ще обозначаваме конкатенацията в свободния моноид, а с $\bullet$ ще обозначаваме конкатенацията на n-торки от думи, дефинирана в началото на секция 1.2.

По-нататък с $\Pi_k(\sigma)$ (за $k \in \{1, 2, ..., n\}$) ще обозначаваме k-тата проекция на σ, където $\sigma \in A^n$ за дадено множество A. Също така с $\sigma^{(i)} \in \Sigma$ ще бележим i-тата поредна буква в думата $\sigma \in \Sigma^*$.

Нека първо дефинираме следното изображение.

Дефиниция 56. Изображението $\phi : ((\Sigma^\varepsilon)^n)^* \to (\Sigma^*)^n$ дефинираме както следва:

$$\begin{aligned}
&\phi(\varepsilon) = \varepsilon_n \\
&\phi(A) = A, \text{ за } A \in (\Sigma^\varepsilon)^n \\
&\phi(\alpha \circ B) = \phi(\alpha) \bullet B, \text{ за } \alpha \in ((\Sigma^\varepsilon)^n)^*, B \in (\Sigma^\varepsilon)^n
\end{aligned}$$

От дефиницията става ясно, че ϕ е моноиден хомоморфизъм. Нека отбележим, че изображението ϕ е сюрекция, но не е инекция.

Сега ще дефинираме едно изображение, което на недетерминиран краен автомат над $(\Sigma^\varepsilon)^n$ без ε преходи съпоставя n-лентов краен автомат. За целта нека с $\mathcal{A}_\Sigma$ означим класа на всички недетерминирани крайни автомати над азбуката Σ. Нека с $\mathcal{A}_\Sigma^{-\varepsilon}$ означим класа на всички недетерминирани крайни автомати над азбуката Σ без ε преходи. Нека с $\mathcal{K}_\Sigma^n$ означим класа на всички n-лентови крайни автомати над азбуката Σ.

Дефиниция 57. Дефинираме изображение $\psi : \mathcal{A}^{-\varepsilon}_{(\Sigma^\varepsilon)^n} \rightarrow \mathcal{K}^n_\Sigma$ по следния начин. Нека е даден недетерминираният краен автомат без ε-преходи $\mathcal{N} = \langle (\Sigma^\varepsilon)^n, S, s, F, \mu \rangle$. Тогава $\psi(\mathcal{N})$ е n-лентов краен автомат $\langle \Sigma, S, s, F, \mu' \rangle$, където:

$$\mu'(r, A) = \mu(r, A), \text{ за } A \in (\Sigma^\varepsilon)^n$$

Изображението ψ е биекция между класовете $\mathcal{A}^{-\varepsilon}_{(\Sigma^\varepsilon)^n}$ и $\mathcal{K}^n_\Sigma$. С $\psi^{-1} : \mathcal{K}^n_\Sigma \rightarrow \mathcal{A}^{-\varepsilon}_{(\Sigma^\varepsilon)^n}$ ще означим обратното изображение. Формално то се дефинира по следния начин.

Дефиниция 58. Дефинираме изображение $\psi^{-1} : \mathcal{K}^n_\Sigma \rightarrow \mathcal{A}^{-\varepsilon}_{(\Sigma^\varepsilon)^n}$ като за даден n-лентов краен автомат $\mathcal{T} = \langle \Sigma, S, s, F, \mu \rangle$ съпоставяме $\psi^{-1}(\mathcal{T})$ недетерминирания краен автомат $\langle (\Sigma^\varepsilon)^n, S, s, F, \mu' \rangle$, където:

$$\mu'(r, \varepsilon) = \emptyset$$
$$\mu'(r, A) = \mu(r, A), \text{ за } A \in (\Sigma^\varepsilon)^n$$

Освен че е биекция, изображението ψ притежава още и следното важно свойство:

Свойство 59. *За всеки автомат $\mathcal{N} \in \mathcal{A}^{-\varepsilon}_{(\Sigma^\varepsilon)^n}$ имаме, че:*

$$\phi(L(\mathcal{N})) = L(\psi(\mathcal{N})).$$

Доказателство. Нека е даден недетерминираният краен автомат без ε-преходи $\mathcal{N} = \langle (\Sigma^\varepsilon)^n, S, s, F, \mu \rangle$. Тогава $\psi(\mathcal{N})$ е n-лентов краен автомат $\langle \Sigma, S, s, F, \mu' \rangle$, където $\mu'(r, A) = \mu(r, A)$, за $A \in (\Sigma^\varepsilon)^n$.

Нека $\sigma \in L(\mathcal{N})$, тогава $\sigma = A_1 \circ A_2 \circ ... \circ A_k, A_i \in (\Sigma^\varepsilon)^n$ и $r_i \in \mu(r_{i-1}, A_i), i = 1, 2, ..., k$ и $r_0 = s, r_k \in F$. Тогава получаваме, че думата $\phi(\sigma) = A_1 \bullet A_2 \bullet ... \bullet A_k$ и от това, че $r_i \in \mu'(r_{i-1}, A_i), i = 1, 2, ..., k$ и $r_0 = s, r_k \in F$, следва $\phi(\sigma) \in L(\psi(\mathcal{N}))$ и получаваме, че $\phi(L(\mathcal{N})) \subseteq L(\psi(\mathcal{N}))$.

Обратно: Нека имаме, че $d \in (\Sigma^*)^n, d \in L(\psi(\mathcal{N}))$. Тогава получаваме, че $d = A_1 \bullet A_2 \bullet ... \bullet A_k$ и $r_i \in \mu'(r_{i-1}, A_i), i = 1, 2, ..., k$ и $r_0 = s, r_k \in F$. Но тогава за думата $\sigma = A_1 \circ A_2 \circ ... \circ A_k$ ще получим: $r_i \in \mu(r_{i-1}, A_i), i = 1, 2, ..., k$ и $r_0 = s, r_k \in F$. Следователно $\sigma \in L(\mathcal{N})$ и $\phi(\sigma) = d$ и получаваме $L(\psi(\mathcal{N})) \subseteq \phi(L(\mathcal{N}))$. □

От това свойство и свойствата на биекцията получаваме и следното:

$$\forall \mathcal{T} \in \mathcal{K}^n_\Sigma \; (\phi(L(\psi^{-1}(\mathcal{T}))) = L(\mathcal{T})).$$

От свойството получаваме следното следствие:

Следствие 60.

$$\forall \mathcal{N}_1, \mathcal{N}_1 \in \mathcal{A}^{-\varepsilon}_{(\Sigma^\varepsilon)^n} \; (L(\mathcal{N}_1) = L(\mathcal{N}_2) \to L(\psi(\mathcal{N}_1)) = L(\psi(\mathcal{N}_2))$$

В класа $\mathcal{A}^{-\varepsilon}_{(\Sigma^\varepsilon)^n}$ имаме понятие минимален автомат както и методи за получаването му. За да получим подобно понятие за класа $\mathcal{K}_\Sigma$, можем да подходим по следния начин.

Дефиниция 61. Изображението $\phi' : (\Sigma^*)^n \to ((\Sigma^\varepsilon)^n)^*$ ще наричаме нормализатор, ако за него е изпълнено следното условие:

$$\forall \sigma \in (\Sigma^*)^n \; (\phi(\phi'(\sigma)) = \sigma).$$

Пример 3. Изображението $\phi' : (\Sigma^*)^n \to ((\Sigma^\varepsilon)^n)^*$ дефинираме по следния начин:

$$\begin{aligned}
&\phi'(\varepsilon_n) = \varepsilon \\
&\phi'(\sigma) = A_1 \circ A_2 \circ \ldots \circ A_m, \text{ където} \quad && m = \max\{|\Pi_i(\sigma)| \mid i = 1, 2, ..., n\} \\
& && A_j = \langle b_{j,1}, b_{j,2}, ..., b_{j,n} \rangle \in (\Sigma^\varepsilon)^n, \text{ за } j = 1, 2, ..., m, \\
& && b_{i,j} = \begin{cases} \Pi_j(\sigma)^i & \text{, в случай, че } i \leq |\Pi_j(\sigma)| \\ \varepsilon & \text{в противен случай} \end{cases}
\end{aligned}$$

за $\sigma \in (\Sigma^*)^n$.

Чрез непосредствена проверка остановяваме, че за него е изпълнено $\forall \sigma \in (\Sigma^*)^n \; (\phi(\phi'(\sigma)) = \sigma)$ и следователно е нормализатор.

Забележка. Не съществува нормализатор, който е хомоморфизъм. Това се показва по следния начин. Нека $a \in \Sigma, a \neq \varepsilon$. Тогава

$$\langle a, a \rangle = \langle a, \varepsilon \rangle \bullet \langle \varepsilon, a \rangle = \langle \varepsilon, a \rangle \bullet \langle a, \varepsilon \rangle$$

следователно ако ϕ' е хомоморфизъм, получаваме:

$$\phi'(\langle a, \varepsilon \rangle) \circ \phi'(\langle \varepsilon, a \rangle) = \phi'(\langle \varepsilon, a \rangle) \circ \phi'(\langle a, \varepsilon \rangle).$$

От това равенство чрез допълнителни разсъждения, използвайки, че ϕ' е обратно на ϕ, може да стигнем до противоречие.

По нататък ще предполагаме, че е даден нормализатор $\phi' : (\Sigma^*)^n \to ((\Sigma^\varepsilon)^n)^*$. Тогава получаваме, че ϕ е биекция между множествата $\phi'((\Sigma^*)^n)$ и $(\Sigma^*)^n$ и ϕ' е неговото обратно изображение.

Дефиниция 62. Един недетерминиран краен автомат без ε-преходи $\mathcal{N} = \langle (\Sigma^\varepsilon)^n, S, s, F, \mu \rangle$ ще наричаме нормализиран спрямо нормализатор ϕ', ако са изпълнени следните условия:

1. $\forall r \in S \; \forall A \in (\Sigma^\varepsilon)^n \; (|\mu(r, A)| \leq 1)$
2. $L(\mathcal{N}) \subset \phi'((\Sigma^*)^n)$

Ясно е, че нормализиран спрямо ϕ' недетерминиран краен автомат може да се разглежда като детерминиран и следователно за него е дефинирано свойството минималност.

Свойство 63. *Нека $\mathcal{N} = \langle (\Sigma^\varepsilon)^n, S, s, F, \mu \rangle$ е нормализиран спрямо нормализатор ϕ' недетерминиран краен автомат без ε-преходи. Тогава:*

$$\forall \sigma_1, \sigma_2 \in L(\mathcal{N}) \; (\sigma_1 \neq \sigma_2 \to \phi(\sigma_1) \neq \phi(\sigma_2)).$$

Доказателството на това свойство се получава непосредствено, като се съобрази, че ϕ' е обратно на ϕ и че $\mathcal{N}$ е нормализиран спрямо ϕ'.

Дефиниция 64. Един n-лентов краен автомат $\mathcal{T}$ ще наричаме нормализиран спрямо нормализатор ϕ', когато $\psi^{-1}(\mathcal{T})$ е нормализиран спрямо ϕ'.

Сега вече можем да дефинираме понятието минимален нормализиран спрямо нормализатор ϕ' n-лентов краен автомат.

Дефиниция 65. Един нормализиран спрямо нормализатор ϕ' n-лентов краен автомат $\mathcal{T}$ ще наричаме минимален, когато $\psi^{-1}(\mathcal{T})$ е минимален.

Свойство 66. *Нека са дадени регулярна n-релация R и нормализатор ϕ', такива че $\phi'(R)$ е регулярен език. Тогава съществува нормализиран спрямо ϕ' n-лентов краен автомат, който я разпознава.*

Доказателство. От теоремата на Клини следва, че съществува детерминиран краен автомат $\mathcal{N}$, който рапознава езика $\phi'(R)$. Тогава е ясно, че $\mathcal{N}$ е нормализиран спрямо ϕ'. Тогава имаме, че $\psi(\mathcal{N})$ е нормализиран спрямо ϕ' и $L(\psi(\mathcal{N})) = R$. □

Забележка. Лесно може да се покаже, че съществуват регулярна n-релация и нормализатор, при които образът на релацията през нормализатора не е регулярен език.

Пример 4. Да разгледаме регулярните релации: $R_1 = \langle a, b\rangle^* \bullet \langle \varepsilon, c\rangle^*$ и $R_2 = \langle \varepsilon, b\rangle^* \bullet \langle a, c\rangle^*$. Нека разгледаме произволен нормализатор ϕ'. Да допуснем, че $\phi'(R_1)$ и $\phi'(R_2)$ са регулярни езици. Тогава и $\phi'(R_1) \cap \phi'(R_2)$ е регулярен език. Лесно се вижда, че $R_1 \cap R_2 = \phi(\phi'(R_1) \cap \phi'(R_2))$. ϕ е моноиден хомоморфизъм, а хомоморфен образ на регулярно множество е регулярно [Ми93]. Следователно $R_1 \cap R_2$ е регулярна релация, което е противоречие със свойство 25. От тук получаваме, че за произволен нормализатор поне един от езиците $\phi'(R_1)$ и $\phi'(R_2)$ не е регулярен.

Забележка. В случай, че регулярната n-релация R е крайна, независимо от избора на нормализатор ϕ', получаваме, че $\phi'(R)$ е краен език и следователно е регулярен.

Теорема 67. *Нека са дадени регулярна n-релация R и нормализатор ϕ', такива че $\phi'(R)$ е регулярен език. Тогава съществува единствен (с точност до изоморфизъм) минимален нормализиран спрямо ϕ' n-лентов краен автомат, който разпознава R.*

Доказателството на тази теорема следва непосредствено от предишното свойство и от теорема 8.

По този начин можем да представим следния директен метод за построяване на нормализиран минимален n-лентов краен автомат по дадена крайна n-релация.

Директно построяване на минимален нормализиран n-лентов краен автомат по дадена крайна n-релация

Нека е дадена крайна n-релация R и произволен нормализатор ϕ'. На първата стъпка построяваме езика $\phi'(R)$. Можем да разглеждаме $\phi'(R)$ като списък в лексикографска наредба, индуцирана от линейна наредба върху $(\Sigma^\varepsilon)^n$. В този случай на втората стъпка можем да използваме метода за директно построяване на минимален автомат по даден списък в лексикографска наредба, описан в секция 1.5. Така получаваме минимален нормализиран спрямо ϕ' краен автомат $\mathcal{N}$ с език $\phi'(R)$. По този начин директно получаваме $\psi(\mathcal{N})$, който представлява минимален нормализиран спрямо ψ' n-лентов краен автомат, разпознаващ релацията R. Коректността на метода следва от коректността на метода за директно построяване на минимален автомат по даден списък в лексикографска наредба, описан в секция 1.5 и свойство 66. $\square$

Директно построяване на минимален нормализиран n-лентов краен автомат, представящ сечение, обединение или разлика на ациклични нормализирани n-лентов крайни автомати

Нека е даден произволен нормализатор ϕ'. От това, че ϕ и ϕ' са обратни за произволни n-релации $R_1, R_2 \subset (\Sigma^*)^n$, имаме следните свойства:

$$R_1 \cup R_2 = \phi(\phi'(R_1) \cup \phi'(R_2))$$
$$R_1 \cap R_2 = \phi(\phi'(R_1) \cap \phi'(R_2))$$
$$R_1 \setminus R_2 = \phi(\phi'(R_1) \setminus \phi'(R_2)).$$

Това обуславя коректността на следния метод за директно построяване на сечение, обединение или разлика на ациклични нормализирани n-лентови крайни автомати.

Нека са дадени ацикличните нормализирани спрямо ϕ' n-лентови крайни автомати $\mathcal{T}_1$ и $\mathcal{T}_2$. Тогава $\psi^{-1}(\mathcal{T}_1)$ и $\psi^{-1}(\mathcal{T}_2)$ са нормализирани спрямо ψ' ациклични детерминирани крайни автомати над азбуката $(\Sigma^\varepsilon)^n$. Тогава можем да приложим методите за директно построяване на сечение, обединение или разлика на ациклични крайни автомати, представени в предишната секция. По този начин получаваме минимален нормализиран спрамо ϕ' детерминиран краен автомат $\mathcal{N}$, представящ сечението, обединението или разликата на автоматите $\psi^{-1}(\mathcal{T}_1)$ и $\psi^{-1}(\mathcal{T}_2)$. Тогава $\psi(\mathcal{N})$ е минимален нормализиран спрямо ψ' n-лентов краен автомат и в следствие на горното свойство представя съответно сечението, обединението или разликата на автоматите $\mathcal{T}_1$ и $\mathcal{T}_2$. $\square$

Забележка. Като се използват свойствата на нормализираните автомати, може да се покаже, че и тук в случайте на сечение и допълнение е достатъчно само първият автомат да бъде ацикличен.

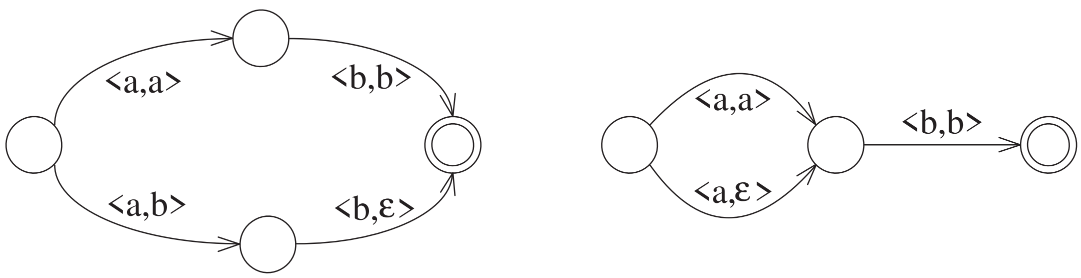
Избор на нормализатор

Както показахме за нашите цели нормализатор може да бъде произволно изображение, за което е в сила свойството $\forall\sigma \in (\Sigma^*)^n \ (\phi(\phi'(\sigma)) = \sigma)$. Едно такова изображение е дадено в пример 3.

За практически цели е естествено изискването нормализираният n-лентов краен автомат за n-релация между думи с еднаква дължина да бъде свободен от ε-и. Условието за това се дава от следното свойство.

Свойство 68. *Нека ϕ' е нормализатор, за което е изпълнено $\forall\sigma \in (\Sigma^\varepsilon)^n \ (|\phi'(\sigma)| = \max\{|\Pi_i(\sigma)| \mid i = 1, 2, ..., n\}$. Тогава ако имаме крайна релация между думи с еднаква дължина, то автоматът, който я разпознава и е свободен от ε-и, е нормализиран спрямо този нормализатор.*

Ясно е, че нормализаторът, даден в пример 3, удовлетворява условието на горното свойство.



Фигура 1.3: Минимални нормализирани преобразователи за релацията $\{\langle ab, ab\rangle, \langle ab, b\rangle\}$.

Забележка. За дадена регулярна n-релация минималните нормализирани спрямо различни функции автомати могат да съдържат различен брой състояния.

Пример 5. На фигура 1.3 са дадени еквивалентните минимални нормализирани преобразователи за релацията $\{\langle ab, ab\rangle, \langle ab, b\rangle\}$. Автоматът, даден от лявата страна, е нормализиран спрямо дефинираният в пример 3 нормализатор ϕ'. На дясната страна на фигурата е даден еквивалентен преобразовател, който е минимален и нормализиран спрямо нормализатор, който запълва с ε-и отпред думите.

От примера се вижда, че изборът на нормализатор е от значение за броя на състоянията на минималния нормализиран n-лентов автомат. Интересен е въпросът за намирането на такъв нормализитор, при който съответният минимален автомат е с най-малък брой състояния. Този проблем остава открит.

Глава 2

Алгоритми

В тази глава ще представим основните оригинални алгоритми, разработени от автора за построяване на минимални крайни автомати, както и за опериране с тях.

Основен оригинален резултат, представен в тази глава, е алгоритъмът за директно построяване на минимален ацикличен краен автомат по даден списък, както и обобщенията му за построяване на минимален ацикличен моноиден преобразувател, връщащ индекс на думите и алгоритъмът за построяване на минимален краен автомат с етикети, класифициращ думите. Също така е важен ефективният алгоритъм за директно построяване на минимален автомат, представящ обединение, сечение и разлика на крайни автомати. Тези алгоритми са изложени в [Mi99].

Формалното описание на алгоритмите ще бъде дадено чрез псевдо-код на език подобен на Pascal. Ще ползваме нотацията за алгоритми, въведена в [AHU83]. Ще представяме също така доказателства за коректност и оценка на сложността на някой от алгоритмите.

2.1 Алгоритъм за директно построяване на минимален автомат за даден списък

Ще предполагаме, че са дадени имплементации за абстрактни типове данни (АТД), представящи състояние на автомат и речник на състоянията. По-нататък ще предполагаме, че NULL е нулевата константа за произволен абстрактен тип данни.

Върху автоматно състояние ще са ни необходими следните типове и операции:

1. STATE е тип указател към състояние на автомат;
2. FIRST_CHAR, LAST_CHAR са съответно първата и последната буква от азбуката на автомата. Ще предполагаме, че кодовете на буквите от автоматната азбуката са поредни;
3. **function** NEW_STATE : STATE връща ново празно състояние;
4. **function** FINAL(STATE) : boolean връща истина, ако състоянието е крайно, и лъжа в противен случай;
5. **procedure** SET_FINAL(STATE,boolean) задава крайността на състоянието;
6. **function** TRANSITION(STATE,char) : STATE връща състоянието, към което се преминава при преход от състоянието, дадено като параметър чрез параметъра-буква;
7. **procedure** SET_TRANSITION(STATE,char,STATE) насочва прехода от състоянието, дадено като първи параметър, чрез параметъра-буква към състоянието, дадено като последен параметър;
8. **procedure** PRINT_AUTOMATON(file,STATE) извежда автомата с начало състоянието, дадено като параметър на файл.

Забележка. Трябва да отбележим, че има съществена разлика между понятието „състояние“ като АТД и понятието „състояние“ според математическата дефиниция на краен автомат от предната глава. В математическата дефиниция състоянието е елемент от множеството на състоянията за даден автомат. В модела, който сега разглеждаме, състоянието само за себе си определя дали е крайно, както и функцията на преходите върху него. Имайки състояние – член на АТД, ние можем да определим с точност до изоморфизъм подавтомата с начало това състояние. Едно възможно математическо представяне на състоянието като член на АТД е наредена двойка от състояние и автомат. По нататък ще използваме съществено това свойство на АТД „автоматно състояние“, което дава възможност при предаване на дадено състояние като аргумент на дадена процедура да избегнем предаването като допълнителен аргумент на самия автомат.

Използвайки горните операции, можем да дефинираме следните помощни функции и процедури:

```
function COPY_STATE ( s : STATE) : STATE;
{ Копира s в ново състояние }
var
   r : STATE;
   c : char;
begin
   r := NEW_STATE;
   SET_FINAL(r,FINAL(s));
   for c := FIRST_CHAR to LAST_CHAR do
      SET_TRANSITION(r,c,TRANSITION(s,c));
   return(r);
end; {COPY_STATE}

procedure CLEAR_STATE ( s : STATE);
 { Изтрива всички преходи от s и го прави некрайно }
var c : char;
begin
   SET_FINAL(s,false);
   for c := FIRST_CHAR to LAST_CHAR do
      SET_TRANSITION(s,c,NULL);
end; {CLEAR_STATE}

function COMPARE_STATES ( s1,s2 : STATE) : integer;
```

```
{ Сравнява две състояния }
var c : char;
begin
   if FINAL(s1) < FINAL(s2) then return(-1)
   else if FINAL(s1) > FINAL(s2) then return(1);
   for c := FIRST_CHAR to LAST_CHAR do
      if TRANSITION(s1,c) < TRANSITION(s2,c) then return(-1)
      else if TRANSITION(s1,c) > TRANSITION(s2,c) then return(1);
      { Тук сравняваме само указателите }
   return(0);
end; {COMPARE_STATES}
```

Абстрактният тип данни, представящ речник на състоянията, трябва да използва дефинираната по-горе функция COMPARE_STATES за сравняване на състояния. За речника ще са необходими следните операции:

1. **function** NEW_DICTIONARY : DICTIONARY връща нов празен речник;
2. **function** MEMBER(DICTIONARY,STATE) : STATE връща състояние, намиращо се в речника, което е еквивалентно на състоянието, дадено като параметър;
3. **procedure** INSERT(DICTIONARY,STATE) вмъква състояние в речника.

Имплементация на гореизброените АТД е представена например в [AHU83]. По-нататък ще предполагаме, че сложността на процедурата PRINT_AUTOMATON е пропорционална на размера на автомата, а всички останали изброени по-горе операции АТД за автоматните състояния, включително COPY_STATE, CLEAR_STATE и COMPARE_STATES, се изпълняват за константно време при фиксирана азбука на автомата. В зависимост от конкретната имплементация на АТД за речника може да имаме различна сложност за операциите върху него. Например при използване на AVL или 2-3 балансирани дървета (виж [AHU83]) сложността на операциите MEMBER и INSERT ще бъде логаритъм от броя на елементите в речника, а необходимата памет ще бъде пропорционална на броя на елементите в речника.

Сега сме готови да представим псевдо-кода на нашия алгоритъм.

Алгоритъм 69. За директно построяване на минимален краен автомат, разпознаващ входен списък от думи в лексикографска подредба.

```
program Create_Minimal_FS_Automaton_for_Given_List ( input, output);
var
   MinimalAutomatonStatesDictionary : DICTIONARY;
   TempStates : array [0..MAX_WORD_SIZE] of STATE;
   InitialState : STATE;
   PreviousWord, CurrentWord : string;
   i, PrefixLengthPlus1 : integer;

   function FindMinimized ( s : STATE) : STATE;
   { връща еквивалентно на това състояние от речника. Ако не съществува,
     добавя към речника копие на състоянието и връща това копие.}
    var r : STATE:
    begin
       r := MEMBER(MinimalAutomatonStatesDictionary,s);
       if r = NULL then begin
          r := COPY_STATE(s);
          INSERT(r);
       end;
       return(r);
    end; {FindMinimized}

 begin
    MinimalAutomatonStatesDictionary := NEW_DICTIONARY;
    for i := 0 to MAX_WORD_SIZE do
       TempState[i] := NEW_STATE;
    PreviousWord := '';
```

```
CLEAR_STATE(TempState[0]);

while not eof(input) do begin
{ цикъл по думите в списъка }
   readln(input,CurrentWord);

   { следващият цикъл изчислява дължината на най-дългия общ
    префикс на CurrentWord и PreviousWord }
   i := 1;
   while (i<length(CurrentWord)) and (i<length(PreviousWord)) and
       (PreviousWord[i] = CurrentWord[i]) do
      i := i+1;
   PrefixLength Plus1 := i;

   { тук минимизираме състоянията от суфикса на предишната дума }
   for i := length(PreviousWord) downto PrefixLengthPlus1 do
      SET_TRANSITION(TempStates[i-1],PreviousWord[i],
                                      FindMinimized(TempStates[i]));
   { тук се построява веригата от нови състояния за текущата дума }
   for i := PrefixLengthPlus1 to length(CurrentWord) do begin
      CLEAR_STATE(TempStates[i]);
      SET_TRANSITION(TempStates[i-1],CurrentWord[i],TempStates[i]);
   end;
   SET_FINAL(TempStates[length(CurrentWord)], true);

   PreviousWord := CurrentWord;
end; {while}

{ тук минимизираме състоянията от последната дума }
for i := length(CurrentWord) downto 1 do
   SET_TRANSITION(TempStates[i-1],PreviousWord[i],
                                   FindMinimized(TempStates[i]));
```

```
47      InitialState := FindMinimized(TempStates[0]);
48      PRINT_AUTOMATON(output,InitialState);
49   end.
```

По-долу ще докажем коректността на алгоритъма и ще представим оценка за сложността му.

Теорема 70. *Нека във входния файл е даден списък от думи в лексикографска подредба. Тогава алгоритъм 69 извежда минималния детерминиран краен автомат, разпознаващ входния списък на изходния файл.*

Доказателство. За да докажем теоремата, ще проведем индукция по думите във входния списък.

При стартиране в редовете 20-24 алгоритъмът инициализира речника от състоянията на минималния автомат и масива от временни състояния. В ред 24 *Tempstate[0]* се инициализира, като се дефинира като некрайно и без преходи. Това съответства на автомата за празния език. Ред 27 чете първата дума от входа. От инициализацията в ред 23 в този момент имаме, че *PreviousWord* е празният низ. Следователно на *PrefixLengthPlus1* ще се присвои 1 в редовете 29-32. Цикълът в редове 33-35 няма да се активира и тогава цикълът в редове 36-40 ще построи верига от състояния, разпознаващи първата дума. По този начин алгоритъмът строи минималния автомат за езика, състоящ се от първата дума в списъка. Ясно е, че този автомат е минимален освен в първата дума $\omega^{(1)}$, като състоянията $t_0, t_1, \ldots, t_k$ са представени в *Tempstate*. В ред 42 на низа *PreviousWord* се присвоява първата дума.

Сега ще покажем, че цикълът в редове 25-43 добавя следващата дума от списъка към езика на автомата и построява минимален освен в тази дума автомат.

Нека допуснем, че на стъпка j алгоритъмът е построил автомат, разпознаващ $\{\omega^{(i)} \,|\, i = 1, 2, \ldots, j-1\}$, който е минимален освен в $\omega^{(j-1)}$. Състоянията $t_0, t_1, \ldots, t_k$ са представени в *TempStates*, състоянията от $S \setminus T$ са в речника *MinimalAutomatonStatesDictionary* и на *PreviousWord* е присвоена думата $\omega^{(j-1)}$.

В ред 27, $\omega^{(j)}$ се прочита от входния файл и се присвоява на *CurrentWord*. Цикълът 28-32 изчислява най-дългия общ префикс на *PreviousWord* и *CurrentWord*, на който в този момент са присвоени съответно $\omega^{(j-1)}$ и $\omega^{(j)}$. Ще покажем, че цикълът 33-35 строи еквивалентен на дадения автомат минимален освен в на-дългия общ префикс на $\omega^{(j-1)}$ и $\omega^{(j)}$. В обратен ред преходите от състоянието t_i се заменят с преходи към състоянията, връщани от функцията *FindMinimized*, където i пробягва от дължината на *PreviosWord* до *PrefixLengthPlus1* в намаляващ ред. Функцията *FindMinimized* търси състояние, еквивалентно на даденото в *MinimalAutomatonStatesDictionary*. Тук са изпълнени условията на лема 42, поради което при използването на COMPARE_STATE във функцията MEMBER ще се търси състояние, еквивалентно на t_i. Ако съществува такова състояние, то се връща като резултат. Това съответства на условията на лема 40. В другия случай функцията копира състоянието в ново и добавя новото състояние към *MinimalAutomatonStatesDictionary*. Новото състояние се връща като резултат. Това съответства на лема 39. Според тези леми и в двата случая полученият автомат ще бъде еквивалентен на дадения и минимален освен в префикса с една буква по-къс. След края на този цикъл ще имаме автомат, разпознаващ $\{\omega^{(i)} \mid i = 1, 2, \ldots, j-1\}$, който е минимален освен в най-дългия общ префикс на $\omega^{(j-1)}$ и $\omega^{(j)}$.

Цикълът 36-40 просто построява верига от състояния в масива *TempStates*, така че да се разпознава и *CurrentWord*. В ред 41 последното от тези нови състояния се маркира като крайно. Това точно отговаря на условията на теорема 41. Следователно построихме автомата с език $\{\omega^{(i)} \mid i = 1, 2, \ldots, j\}$, който е минимален освен в думата $\omega^{(j)}$. В ред 42 присвояваме на *PreviousWord* $\omega^{(j)}$ и затваряме основния цикъл.

От индукцията получаваме, че при излизане от цикъла 25-43 алгоритъмът ще построи автомат, разпознаващ входния списък, който е минимален освен в последната дума. В редовете 44-47 по същия начин както в цикъла 33-35 се построява автомат, еквивалентен на дадения, който е минимален освен в ε. От свойство 38 имаме, че това е минималният автомат за входния списък. В ред 48 този автомат се извежда на изходния файл. □

Теорема 71. *Сложността на алгоритъм 69 по отношение на време е*

$O(n\log(m))$, където n е общият брой символи във входния списък и m е размерът (брой състояния) на резултатния минимален автомат. Сложността по отношение на паметта на алгоритм 69 е $O(m)$.

Доказателство. За всеки символ от входния списък алгоритъмът преминава или през ред 31, или през редове 38-39. Всяка от инструкциите в тези редове се изпълнява за константно време. В случай че сме преминали през редове 38-39, по-нататък ще трябва да преминем през ред 35 или 46. Изпълнението на редовете 35, 46 зависи от сложността на *FindMinimized.* При използване на имплементация за речника от състоянията чрез балансирано дърво сложността (по отношение на времето) на функцията *FindMinimized* е логаритъм от броя на елементите в речника. Речникът съдържа не повече от m елемента, където m е броят на състоянията на минималния автомат за списъка. Следователно сложността (по отношение на времето) на алгоритъма като цяло е $O(n\log(m))$.

Ясно е, че паметта, необходима за изпълнението на алгоритъма, е равна на размера на речника на състоянията на минималния автомат плюс размера на масива *TempStates* плюс размера на останалите фиксирани променливи, които са с константни размери. Размерът на *TempStates* е пропорционален на дължината на най-дългата дума в списъка, а тя е винаги по-малка от броя на състоянията на минималния автомат за този списък. При имплементация на речника на състоянията чрез балансирани дървета размерът на речника е пропорционален на броя на елементите му, които са състоянията на минималния автомат. Следователно получаваме, че сложността на алгоритъма по отношение на памет е $O(m)$. □

Забележка. Когато предварително може да се оцени броят на състоянията на резултатния минимален автомат, на практика е по-целесъобразно да се използва „хеширане“ (виж [AHU83]) за представянето на речника от състояния. В този случай времето, необходимо за операциите MEMBER и INSERT, е почти константно.

Основното преимущество на нашия алгоритъм е изключително доброто съотношение между сложността за памет и време.

За пълнота ще представим и алгоритъма за разпознаване на дума от краен автомат.

Алгоритъм 72. За разпознаване на дума от краен автомат.
Нека s е началното състояние на крайния автомат. Тогава функцията RECOGNIZE_WORD връща истина в случай, че думата w се разпознава от автомата, и лъжа в противен случай.

```
function RECOGNIZE_WORD (w: string; s: STATE) : boolean;
var i : integer;
begin
   i := 1;
   while i<=length(w) do begin
      if TRANSITION(s,w[i])=NULL then return(false);
      s := TRANSITION(s,w[i]);
      i := i+1;
   end;
   return(FINAL(s));
end;
```

Действието на горната функция не се нуждае от обяснения. Единствено трябва да отбележим, че времето за разпознаване на дадена дума чрез горната функция е пропорционално на дължината на думата и тъй като не е рекурсивна, не използва допълнителна памет.

2.2 Алгоритъм за директно построяване на краен автомат с етикети на крайните състояния, разпознаващ и класифициращ даден списък

Нека сега разгледаме какви изменения трябва да се проведат в алгоритъма 69, за да се приложи методът за директно построяване на краен автомат с етикети на крайните състояния, разпознаващ и класифициращ даден списък.

На първо място трябва да се разширят АТД, представящи състояние на автомат, с цел да се присвоява етикет на крайно състояние. Без ограничение на общността можем по-нататък да предполагаме, че множеството на етикетите E съвпада с $\mathbb{N}$. В такъв случай трябва да направим следните промени в операциите на АТД за състоянията на автомата:

- **function** FINAL(STATE) : integer връща -1, ако състоянието не е крайно, в противен случай връща етикета на състоянието;
- **procedure** SET_FINAL(STATE,integer) ако вторият параметър е -1, задава, че състоянието, дадено като първи параметър, не е крайно, в противен случай задава етикета на състоянието;

Във процедурата CLEAR_STATE, дадена на страница 60 редът:
SET_FINAL(s,false);
трябва да се промени на:
SET_FINAL(s,-1);

За изпълнението на алгоритъма 69 трябва да се предполага, че във входния файл е даден списъкът от думи заедно с класовете, към които принадлежат. За правилното изпълнение е необходимо да се направят следните промени:

1. Към ред 7 трябва да се добави нова променлива *CurrentWordClass* от тип integer:
   i, PrefixLengthPlus1, CurrentWordClass : integer;

2. Ред 27 се изменя на:
   readln(*input,CurrentWord,CurrentWordClass*);

3. Ред 41 се изменя на:
   SET_FINAL(*TempStates*[length(*CurrentWord*)], *CurrentWordClass*);

Тъй като всички горепосочени промени са незначителни и почти не изменят действието на алгоритъма, няма да представяме подробно доказателство за коректността му, както и оценка за сложност. Това може да се направи по схемата, представена в предишната секция.

В тази секцията представихме алгоритъм за директното построяване на автомат с етикети на крайните състояния, който се характеризира с многократно по-добра ефективност от предлаганите до сега.

За пълнота ще представим и алгоритъма за класифициране на дума от краен автомат с етикети.

Алгоритъм 73. За класифициране на дума от краен автомат с етикети.
Нека s е началното състояние на крайния автомат с етикети. Тогава функцията CLASSIFY_WORD връща -1 в случай, че думата w не се разпознава от автомата, или връща класа на думата, в случай че тя принадлежи на езика на автомата.

```
function CLASSIFY_WORD (w: string; s: STATE) : integer;
var i : integer;
begin
   i := 1;
   while i<=length(w) do begin
      if TRANSITION(s,w[i])=NULL then return(-1);
      s := TRANSITION(s,w[i]);
      i := i+1;
   end;
   return(FINAL(s));
end;
```

Действието на горната функция е почти аналогично на функцията RECOGNIZE_WORD от алгоритъм 72. И тук трябва да отбележим, че времето за класифициране на дадена дума чрез горната функция е пропорционално на дължината на думата и тъй като не е рекурсивна, не използва допълнителна памет.

2.3 Алгоритъм за директно построяване на моноиден преобразувател за даден списък, връщащ индекса на думите

В този случай отново ще използваме като база алгоритъма 69.

Ще ни трябват някой разширения на АТД, представящи състояние на автомат. Първо да дефинираме тип двойка състояние - число.

State_int_pair = **record**
  s : STATE;
  i : integer;
end

Ще използваме следните допълнения и изменения на АТД, представящ състояние на автомата:

- **function** State_int_pair(STATE,integer) : State_int_pair връща двойката от състояние и число, дадени като аргументи;
- **function** TRANSITION_MASS(STATE,char) : integer връща изхода на функцията λ при прехода от състоянието, дадено като параметър чрез параметъра буква;
- **procedure** SET_TRANSITION(STATE,char,State_int_pair) насочва прехода от състоянието, дадено като първи параметър, чрез параметъра буква към състоянието от двойката, дадена като последен параметър, и установява функцията λ за този преход на числото от двойката;

Помощната функция COMPARE_STATES остава без изменения, а по-долу са дадени новите варианти на функциите COPY_STATE и CLEAR_STATE.

```
function COPY_STATE ( s : STATE) : STATE;
{ Копира s в ново състояние, като в случай че се приложи
лема 52, преизчислява функцията λ за състоянието }
var
   r : STATE;
   c : char;
   i : integer;
begin
   r := NEW_STATE;
   SET_FINAL(r,FINAL(s));
   if FINAL(s) then i := 1 else i := 0;
   for c := FIRST_CHAR to LAST_CHAR do
      if TRANSITION(s,c) = NULL then
         SET_TRANSITION(r,c,State_int_pair(NULL,0))
      else begin
         SET_TRANSITION(r,c,State_int_pair(TRANSITION(s,c),i));
         i := i + TRANSITION_MASS(s,c);
      end; {if}
   return(r);
end; {COPY_STATE}

procedure CLEAR_STATE ( s : STATE);
{ Изтрива всички преходи от s и го прави некрайно }
var c : char;
begin
   SET_FINAL(s,false);
   for c := FIRST_CHAR to LAST_CHAR do
      SET_TRANSITION(s,c, State_int_pair(NULL,0));
end; {CLEAR_STATE}
```

Ще ни е необходима и още една помощна функция:

function STATE_MASS (s : STATE) : integer;
var
  i : integer;
begin
  if FINAL(s) **then** $i := 1$ **else** $i := 0$;
  for c := FIRST_CHAR **to** LAST_CHAR **do**
    $i := i +$ TRANSITION_MASS(s,c);
  return(i);
end; {STATE_MASS}

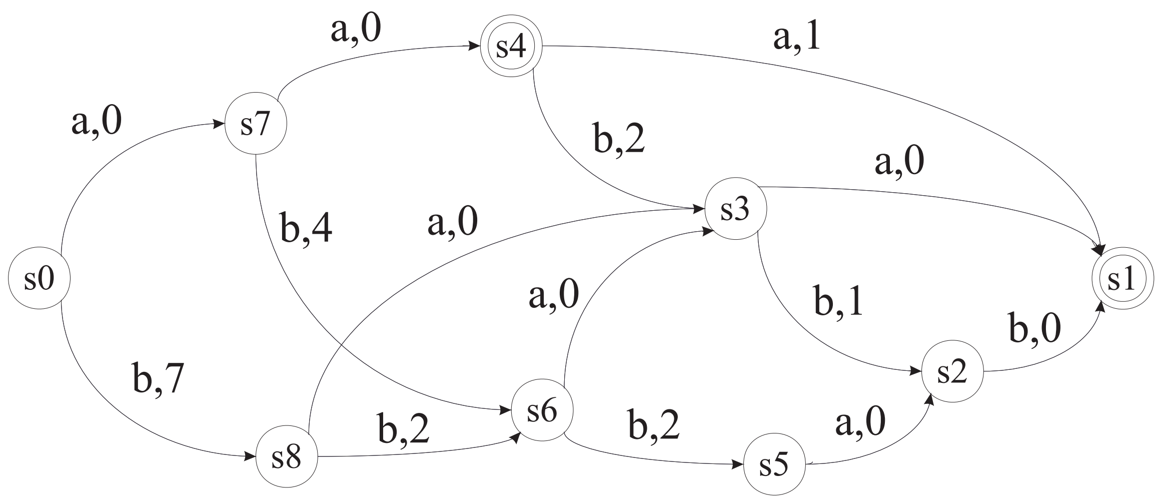
Върху алгоритъма 69 ще трябва да направим следните поправки:

1. Ред 17 се променя по следния начин:
   return(State_int_pair(r, STATE_MASS(r));

2. Ред 47 се променя по следния начин:
   InitialState := *FindMinimized*(*TempStates*[0]).s;

Разглеждайки алгоритъма установяваме, че промените точно отразяват допълненията, необходими за изпълнение на условията на свойствата 50-55. По основния алгоритъм промените са незначителни и не изменят съществено действието на програмата. Доказателствата за коректност и сложност се правят по схемата, приложена при теореми 70, 71.

Забележка. Показаните в тази и предишната секция разширения на алгоритъма за построяване на минимален автомат за даден списък са независими. Това позволява и разширяването на алгоритъма едновременно в двете посоки — за построяване на минимален моноиден преобразувател по даден списък, който едновременно връща индекса на разпознатата дума и я класифицира.



Фигура 2.1: Минимален моноиден преобразувател, връщащ индекс на дума.

На фигура 2.1 е даден минималният моноиден преобразувател, връщащ индекса на думите. Стойностите на функцията λ са дадени към съответните преходи.

За пълнота ще представим и алгоритъма за намиране на индекса на дума от моноиден преобразувател, връщащ индекс.

Алгоритъм 74. За намиране на индекса на дума от моноиден преобразувател, връщащ индекс.
Некаs е началното състояние на моноиден преобразувател, връщащ индекс. Тогава функцията INDEX_OF_WORD връща -1 в случай, че думата w не принадлежи на езика на преобразувателя, или връща индекса на думата в противен случай.

function INDEX_OF_WORD (w: string; s: STATE) : integer;
var i, m : integer;
begin
  $i := 1$;
  $m := 0$;
  while i<=length(w) **do begin**
    if TRANSITION(s,w[i])=NULL **then return**(-1);
    $m := m$+TRANSITION_MASS(s,w[i]);

```
        s := TRANSITION(s,w[i]);
        i := i+1;
    end;
    if FINAL(s) then return(m) else return(-1);
end;
```

Действието на горната функция е почти аналогично на функцията RECOGNIZE_WORD от алгоритъм 72. И тук трябва да отбележим, че времето за индексиране на дадена дума чрез горната функция е пропорционално на дължината на думата и тъй като не е рекурсивна, не използва допълнителна памет.

2.4 Алгоритъм за получаване на дума по индекса ѝ от моноиден преобразувател, връщащ индекса на думите

В тази секция ще представим алгоритъм за намиране на дума по индекса ѝ в даден списък. Основното преимущество на този алгоритъм се състои в това, че за целта се използва моноидният преобразувател за списъка, връщащ индекс, което позволява времето за намирането на думата да е пропорционално на дължината на думата независимо от големината на списъка. Това позволява да се използват тези структури за перфектно хеширане. Добрата компресия на думите и изключително доброто време за достъп на този алгоритъм в комбинация с алгоритмите 73 и алгоритъма за директно построяване на моноиден преобразувател за даден списък позволяват широкото им използване за решаване на разнообразни практически задачи.

Алгоритъм 75. За намиране на дума по нейния индекс в моноиден преобразувател, връщащ индекс.

```
function FIND_WORD ( n : integer; r : STATE ) : string;
var
```

```
    c : char;
    w : string;
    i : integer;
begin
    w := '';
    i := 0;
    while (i <> n) or not FINAL(r) do begin
        c := LAST_CHAR;
        while (TRANSITION(r,c)=NULL) or (i+TRANSITION_MASS(r,c)>n) do
            c := pred(c);
        i := i + TRANSITION_MASS(r,c);
        r := TRANSITION(r,c);
        w := concat(w,c);
    end;
    return(w);
end; {FIND_WORD}
```

Теорема 76. *Нека е даден моноиден преобразувател $\mathcal{M} = \langle \Sigma, \mathbb{N}, S, s, F, \mu, \lambda \rangle$, отговарящ на условието (1.2) от теорема 35. Тогава алгоритъм 75 при даден входен индекс $n < |L_I(\mathcal{M})|$ и параметър началното състояние s връща думата, която е с индекс n в лексикографската подредба на входния език на автомата $\mathcal{M}$. Сложността на алгоритъма е $O(l)$, където l е дължината на думата, която се извежда.*

Доказателство. За доказателството на коректността на алгоритъма ще използваме помощното твърдение в доказателството на теорема 35 — ако сме достигнали в даден възел с поддума σ, то има точно $\lambda^*(s, \sigma)$ думи пред поддумата σ в азбучния списък, описан от автомата. Нека сме достигнали възел r с дума w. Като анализираме алгоритъма, виждаме, (чрез тривиална индукция), че в този момент $i = \lambda^*(s, w), r = \mu^*(s, w)$. Ако в този момент $i = n, r \in F$, то е ясно, че думата w има индекс n и $w \in L(\mathcal{A})$. В противен случай трябва да намерим подходящ преход, за да продължим по-нататък. Ясно е, че преходите, за които $\lambda(r, c_i) + i > n$, няма да доведат до необходимия индекс. Нека $c = max\{c_i \in \Sigma \,|\, \lambda(r, c_i) + i \leq n\}$. Тогава от условието 1.2 следва, че преминавайки през преходи с букви $c_i < c$, ще достигнем до думи с индекси, по-малки от $\lambda(r, c) + i$. Тъй като $\lambda(r, c) + i < n$, то единственият начин за достигане на дума с индекс n е да се премине през преход с буква c. По този начин задължително ще достигнем до удовлетворяване на условието в стъпка 1. Това се основава на факта, че моноидният преобразувател удовлетворява условие 1.2 на теорема 35, откъдето следва, че на всеки индекс отговаря дума от входния език на автомата.

Броят на итерациите на вътрешния цикъл е по-малък или равен на броя на буквите в азбуката на автомата, а операциите при всяка от итерациите се изпълняват за константно време. Следователно при фиксирана азбука вътрешният цикъл се изпълнява за константно време. Външният цикъл добавя при всяка итерация по една буква към думата, която се извежда. Всички операции във външния цикъл се изпълняват за константно време. Следователно времето за изпълнение на външния цикъл, а и на алгоритъма е пропорционално на дължината на думата, която се извежда. $\square$

2.5 Алгоритъм за директно построяване на минимален автомат, представящ обединение, сечение или разлика на ациклични детерминирани крайни автомати

В тази секция ще представим алгоритъм, който използвайки алгоритъма за директно построяване на минимален автомат по даден списък, построява директно минимален автомат, представящ обединение, сечение или разлика на ациклични детерминирани крайни автомати. За целта е необходимо първо да представим алгоритъм, който по даден ацикличен краен автомат извежда списъка от думите, разпознавани от автомат в лексикографски ред. При това от съображения за ефективност трябва списъкът да се извежда дума по дума, без да е необходимо генерирането на целия списък в паметта.

Отново ще използваме АТД за представяне на състояние на автомат, описан в секция 2.1. Ще ни е необходима още следната декларация:
type States_Stack = **array** [1..MAX_WORD_SIZE+1] of STATE тип масив от автоматни състояния.

Масив от състояния ще използваме за представяне на път в автомат, т.е. ако е дадена дума w : string, ще имаме, че:
$S[i+1]$ = TRANSITION($S[i]$,$w[i]$), $i = 1,2,...,$lenght(w), а $S[0]$ е началното състояние на автомата.

За извеждането на списъка дума по дума ще използваме функция, която по дадена дума и път в автомат намира следващата дума по лексикографски ред, разпознавана от автомата.

Алгоритъм 77. За извеждане на следващата дума по лексикографски ред, разпознавана от ацикличен детерминиран краен автомата по даден път и дума.

Ще предполагаме, че в автомата от всяко състояние, достижимо от началното, е достижимо някое крайно и че празната дума не принадлежи на езика на автомата.

```
function NEXT_AUTOM_WORD(var S : States_Stack; var w : string) : boolean;
var
  c : char;
  sp : integer;

  function FIND_FORWARD_WORD : boolean;
  begin
    c := FIRST_CHAR;
    while (c<=LAST_CHAR) and (TRANSITION(S[sp],c)=NULL) do c:=succ(c);
    if c>LAST_CHAR then return(false);
    S[sp+1] := TRANSITION(S[sp],c);
    sp := sp+1;
    w := concat(w,c);
    while not FINAL(S[sp]) do begin
      c := FIRST_CHAR;
```

```
        while TRANSITION(S[sp],c)=NULL do c:=succ(c);
        S[sp+1] := TRANSITION(S[sp],c);
        sp := sp+1;
        w := concat(w,c);
      end;
      return(true);
    end;

begin
  sp := length(w);
  if FIND_FORWARD_WORD then return(true);
  repeat
    if sp=1 then return(false);
    sp := sp-1;
    c := w[length(w)];
    delete(w,length(w),1);
    while (c<=LAST_CHAR) and (TRANSITION(S[sp],c)=NULL) do c:=succ(c);
  until c<=LAST_CHAR;
  S[sp+1] := TRANSITION(S[sp],c);
  sp := sp+1;
  w := concat(w,c);
  if not FINAL(S[sp]) then return(FIND_FORWARD_WORD) else return(true);
end;
```

Теорема 78. *Нека на алгоритъм 77 са подадени дума и съответен път от автомат. В случай, че не съществува дума в езика на автомата по-голяма по лексикографски ред от дадената, алгоритъмът връща лъжа. В противен случай алгоритъмът намира следващата дума по лексикографски ред и връща истина.*

Доказателство на горната теорема може да се направи чрез индукция по думите, разпознавани от автомата.

Чрез горния алгоритъм е лесно да се представи алгоритъм, даващ дума по дума лексикографски подреден списък – обединение, сечение или разлика на езиците на крайни автомати.

Алгоритъм 79. За извеждане на следващата дума по лексикографски ред от обединението на езиците на два ациклични детерминирани крайни автомата.

Ще са ни необходими следните глобални променливи:

```
var
  p1,p2,f1,f2 : boolean;
  s1,s2 : States_Stack;
  w1,w2 : string;
```

Ще предполагаме, че те са инициализирани чрез следната процедура:

```
procedure INIT_NEXT_WORD;
begin
  s1[1] := init_state1;
  s2[1] := init_state2;
  w1 := '';
  w2 := '';
  p1 := true;
  p2 := true;
  f1 := true;
  f2 := true;
end;
```

init_state1, init_state2 са началните състояния на съответно първия и втория автомат. Тогава функцията NEXT_WORD дава в променливата w следващата дума по лексикографсия ред в обединението или връща лъжа, ако няма повече думи.

```
function NEXT_WORD(var w: string) : boolean;
```

```
begin
  if not f1 and not f2 then return(false);
  if p1 then f1 := NEXT_AUTOM_WORD(s1,w1);
  if p2 then f2 := NEXT_AUTOM_WORD(s2,w2);
  if not f1 and not f2 then
    return(false)
  else if (f1 and f2) and (w1=w2) then begin
    w := w1;
    p1 := true;
    p2 := true;
  end else if not f1 or ((f1 and f2) and (w1>w2)) then begin
    w := w2;
    p1 := false;
    p2 := true;
  end else if not f2 or ((f1 and f2) and (w1<w2)) then begin
    w := w1;
    p1 := true;
    p2 := false;
  end;
  return(true);
end;
```

Действието на функцията NEXT_WORD е следното: в променливите *p1, p2* се маркира необходимостта от четене на следваща дума съответно от първия или втория автомат. В промелнивите *f1, f2* се маркира това, дали първият или вторият списък е свършил. Ако и двата списъка са свършили, се връща лъжа. В противен случай се сравняват поредните две думи от двата автоматни списъка. Ако те са равни, то се извежда едната от тях и чрез *p1, p2* се маркира, че при следващото повикване трябва да се прочетат нови думи и от двата списъка. Ако едната дума е по-малка по лексикографски ред от другата, то тя се извежда. Съответният списък се маркира, за да се прочете следващата дума от него. А ако единият от списъците е свършил, то се изчерпва другия.

Да отбележим, че чрез функцията NEXT_WORD се извежда дума по дума списъкът от думите в обединението на езиците на двата изходни автомата, без да се налага използване на памет за построяването на целия списък. При това времето за изчерпване на всички думи от обединението пропорционално е по-малко или равно на сбора на дължините на двата изходни езика в брой символи. Това се показва чрез следните разсъждения: за да се изведат всички думи, е необходимо да се изведат всички думи от изходните автомати. Това става чрез обхождането на всички пътища, които водят от началното до някое крайно състояние. Сборът от дължините на пътищата е равен на броя на символите в езика на автомата. □

За да изведем списък, явяващ се сечение или разлика на автоматни езици, бихме могли да постъпим подобно на случая на обединение. Но ние ще покажем един по-ефективен метод, който е приложим и в случай, когато единият от автоматите е цикличен. Ще използваме функцията RECOGNIZE_WORD, дадена в алгоритъм 72.

Алгоритъм 80. За извеждане на следващата дума по лексикографски ред от сечението на езиците на ацикличен детерминиран краен автомат и детерминиран краен автомат.

Ще са ни необходими следните глобални променливи:

var
  f1 : boolean;
  s1 : States_Stack;
  w1 : string;

Ще предполагаме, че те са инициализирани чрез следната процедура:

procedure INIT_NEXT_WORD;
begin
  s1[1] := *init_state1*;
  w1 := '';
  f1 := true;

```
end;
```

init_state1, init_state2 са началните състояния съответно на първия и втория автомат. Тогава функцията NEXT_WORD дава в променливата w следващата дума по лексикографси ред в сечението или връща лъжа, ако няма повече думи.

```
function NEXT_WORD(var w: string) : boolean;
begin
   if not f1 then return(false);
   repeat
      f1 := NEXT_AUTOM_WORD(s1,w1);
   until not f1 or RECOGNIZE_WORD(w1,init_state2);
   if not f1 then return(false);
   w := w1;
   return(true);
end;
```

В този случай функцията NEXT_WORD действа по следния начин: дума по дума се изчерва езикът на първия автомат по азбучен ред, като за всяка дума се проверява дали принадлежи на езика на втория автомат. Ако принадлежи, тя се извежда като дума от сечението.

Тук отново можем да отбележим, че чрез функцията NEXT_WORD се извежда дума по дума списъкът от думите в сечението на езиците на двата изходни автомата, без да се налага използване на памет за построяването на целия списък. При това времето за изчерпването на всички думи от сечението е пропорционално на дължината на първия автоматен език в брой символи, тъй като както изчерпването на думите от първия автомат, така и разпознаването им от втория по време е пропорцонално на броя на символите в езика на първия автомат. □

За да получим алгоритъм за извеждане на следващата дума по лексикографски ред от разликата на езиците на ацикличен детерминиран краен автомат и детерминиран краен автомат, е необходимо в горния алгоритъм да се направи следното изменение:

until not *f1* **or** RECOGNIZE_WORD(*w1*,*init_state2*);

да се замени с:

until not *f1* **or not** RECOGNIZE_WORD(*w1*,*init_state2*);

При това действието на алгоритъма почти не се изменя и отново времето, необходимо за извеждането на списъка, ще бъде пропорционално на броя на символите в езика на първия автомат.

При използване на програмни езици, позволяващи дефинирането на итератори, или чрез обектно-ориентирано програмиране би могло да се изложат по един по-прегледен начин горните алгоритми. За целта функциите NEXT_WORD, NEXT_AUTOM_WORD биха могли да се представят като итератори.

Нека сега представим алгоритъма за директното построяване на минимален автомат.

Алгоритъм 81. За директно построяване на минимален краен автомат, разпознаващ обединението, сечението или разликата на два ациклични детерминирани крайни автомата. В случай на сечение и разлика е достатъчно само първият автомат да бъде ацикличен.

Ще използваме за база отново алгоритъм 69. Ще предполагаме, че са дадени глобалните променливи init_state1, init_state2, представящи началното състояние съответно на първия и втория автомат. Също така – че са дефинирани глобалните променливи според алгоритъм 79 или съответно алгоритъм 80 и съответните функции NEXT_AUTOM_WORD, NEXT_WORD, INIT_NEXT_WORD, RECOGNIZE_WORD. Необходими са още и следните изменения:

1. Между редове 24 и 25 е необходимо да се извика инициализиращата процедура:
   INIT_NEXT_WORD;

2. Ред 25 е необходимо да се измени на:
   while NEXT_WORD(*CurrentWord*) **do begin**

3. Ред 27 е необходимо да се изтрие.

Единствената разлика между горния алгоритъм и алгоритъм 69 е използването на входен списък, който е съответно обединение, сечение или разлика на изходните автоматни езици. Оттук се получава директно и доказателството за коректността на представения алгоритъм. Използвайки разсъжденията, представени в тази секция, както и оценките за сложност на алгоритъм 69, получаваме следните оценки за сложност. В случай на обединение сложността по отношение на време на алгоритъм 81 е $O((n_1 + n_2) + n\log(m))$, където n_1, n_2 са общият брой символи съответно на първия и втория автоматен език, n е общият брой символи в резултатния език, а m е размерът (брой състояния) на резултатния минимален автомат. Тъй като $n_1 + n_2 < 2n$ получаваме, че сложността е $O(n\log(m))$. При сечение и разлика сложността по отношение на време на алгоритъм 81 е $O(n_1 + n\log(m))$. По отношение на памет сложността на алгоритъм 81 е $O(m)$. □

Забележка. Както вече отбелязахме, стандартните методи за обединение, сечение или разлика на автомати се основават на построяването на автомат, чиито състояния са декартово произведение на състоянията на изходните автомати. Поради огромния обем на този междинен автомат на практика този метод е трудно да се прилага за големи автомати. За да се отделят недостижимите състояния на резултатния автомат, трябва необходимо да се обходят всички състояния от новия автомат. След това е необходимо допълнително минимизиране. Поради тези причини представеният от нас метод значително подобрява ефективността както по отношение на памет, така и по отношение на време спрямо стандартните алгоритми.

Алгоритъм 81 може да се приложи също за директно построяване на минимален ацикличен краен автомат с етикети, представящ сечение, обединение и разлика на ациклични крайни автомати с етикети. За целта трябва да се направят някой незначителни изменения подобни на тези в секция 2.2. Също е необходимо с случайте наобединение и сечение класификационната функция да е съгласувана върху двата изходни автомата.

Чрез използването на преобразуванията представени в секция 1.7 алгоритъм 81 може да се приложи за директно построяване на минимален нормализиран ацикличен краен n-лентов автомат, представящ сечение, обединение и разлика на нормализирани, ациклични крайни n-лентови автомати.

Глава 3

Прилагане на методите и алгоритмите за построяване на ациклични крайни автомати

В тази глава ще представим някои приложения на методите и алгоритмите за директно построяване на минимален краен автомат. Ще се спрем главно на задачи от областта на компютърната лингвистика. Първо ще покажем как се използват ациклични крайни автомати за решаването на някои конкретни задачи. След това ще бъдат представени компютърни системи, в които са имплементирани нашите методи.

В тази глава ще се въздържим от разглеждането на въпроси от езикознанието, поради тяхната специфична дълбочина и отдалеченост от основната насока на разглежданата тематика. Лингвистичните примери са само с илюстративна цел и поради това оставяме настрана въпросите, свързани с детайли на пълната лингвистична коректност.

3.1 Моделиране на граматичен речник чрез ациклични крайни автомати с етикети на крайните състояния

Граматичният речник трябва да съдържа информация за думите в езика, техните граматични характеристики, както и формите, които се образуват. Основните трудности при това са свързани със следните особености на езика:

- Голямо количество словоформи — от 60 000 основни форми се генерират около 900 000 словоформи;
- Сложна класификация на думите и голямо разнообразие от граматични характеристики на словоформи;
- Много изключения при образуването на формите.

За български език са създадени следните компютърни системи за моделиране на морфология — в ЦЛПОИ [PSDS93, Mi96] и в Пловдивския университет [TK97]. Също така в момента се работи по създаването на морфологичен речник към Института по български език към БАН. Тези системи използват модел, при който експлицитно се описват измененията на основната форма за получаването на даден член на парадигмата.

В следващите две секции ще се спрем на модели, основаващи се на използването на крайни автомати. Те се характеризират със значително по-добро бързодействие, както и с това, че са езиково независими.

Един удобен начин за моделиране на граматичен речник е чрез използването на крайни автомати с етикети на крайните състояния. Това е подходът, изложен в [Re91, Ro92, Si93], за граматичен речник на френския език. Такова представяне позволява много бърз анализ на думите, както и сравнителна добра компресия. Ще илюстрираме този модел чрез следния пример.

Нека ни е даден речник във вида:

```
ваза,ваза.N:fs
вазата,ваза.N:fsd
```

```
вази,ваза.N:fp
вазите,ваза.N:fpd
вода,вода.N:fs
водата,вода.N:fsd
води,вода.N:fp,водя.V+t+IPF:P3s:A2s:A3s:Z2s
водите,вода.N:fpd,водя.V+t+IPF:P2p
маса,маса.N:fs
масата,маса.N:fsd
маси,маса.N:fp
масите,маса.N:fpd
```

Нека разгледаме какво означава изразът `водите,вода.N:fpd,водя.V+t+IPF:P2p`. Първата дума е словоформата `водите`. След запетайката следва основната форма, от която се получава словоформата. В случая това е думата `вода`. След точката следва граматичната информация `N:fs`, в която са кодирани граматичните характеристики: съществително, женски род, единствено число. Следва запетайка и нова основна форма. Това показва, че думата `водите` може да се получи и от основна форма `водя`. След това следва граматичната информация за второто четене: глагол, преходен, несвършен вид, сегашно време, второ лице, множествено число. При представянето на граматичната информация е използван стандартът за кодиране на граматични речници DELAF [Si93].

Първата стъпка се състои в това да съставим указание за получаване на основната форма от съответната словоформа. Например от думата `вазите` основната форма `ваза` се получава, като се изтрият последните три букви и се добави буквата `а`. Това ще се кодира като `3а`. Преобразуван по този начин, речникът изглежда така:

```
ваза,0.N:fs
вазата,2.N:fsd
вази,1а.N:fp
вазите,3а.N:fpd
вода,0.N:fs
```

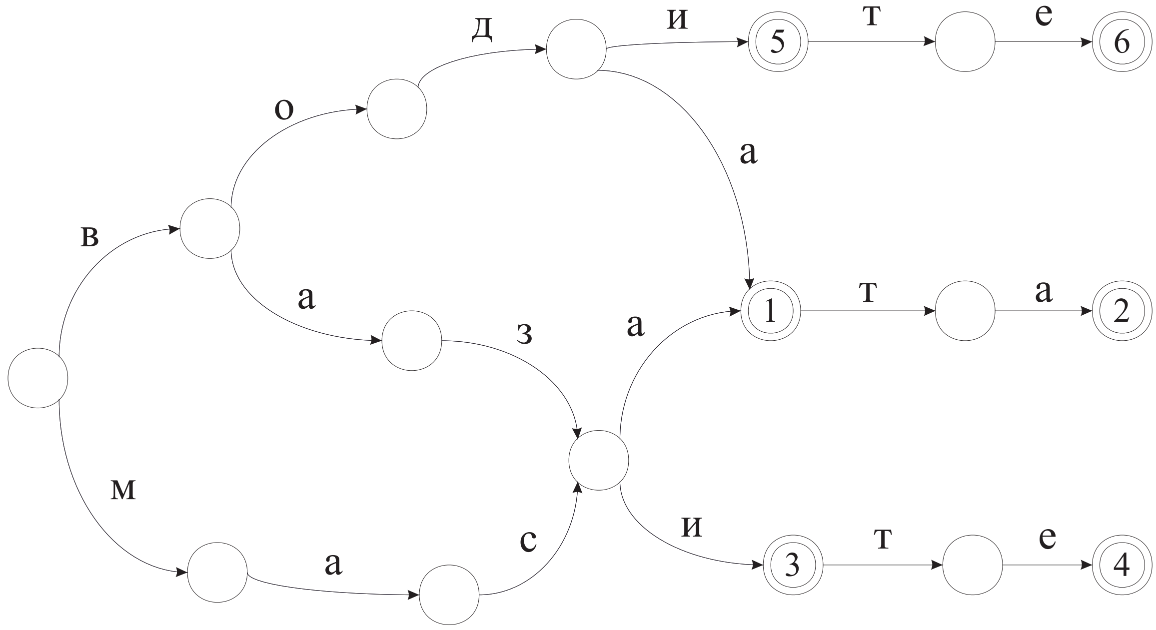
```
водата,2.N:fsd
води,1а.N:fp,1я.V+t+IPF:P3s:A2s:A3s:Z2s
водите,3а.N:fpd,3я.V+t+IPF:P2p
маса,0.N:fs
масата,2.N:fsd
маси,1а.N:fp
масите,3а.N:fpd
```

Нека отделим информациите с цел да опростим етикетите:

```
1.  0.N:fs
2.  2.N:fsd
3.  1а.N:fp
4.  3а.N:fpd
5.  1а.N:fp,1я.V+t+IPF:P3s:A2s:A3s:Z2s
6.  3а.N:fpd,3я.V+t+IPF:P2p
```

```
ваза,1
вазата,2
вази,3
вазите,4
вода,1
водата,2
води,5
водите,6
маса,1
масата,2
маси,3
масите,4
```

По този начин се получава класификация на думите, като всеки клас се свързва с информация за начина на получаване на основната форма от словоформата и допълнителна граматична информация за думата.



Фигура 3.1: Краен автомат с етикети на крайните състояния, моделиращ граматичен речник.

В този вид речникът е удобен за представяне чрез ацикличен краен автомат с етикети на крайните състояния за моделирането на речника. Примерът за това е показан на фигура 3.1.

Алгоритъмът за директно построяване на ацикличен краен автомат с етикети, представен в секция 2.2, е много ефективен за създаването на голям граматичен речник. При анализ на думите е подходящо да се използва алгоритъм 73. Алгоритмите от секция 2.5 са приложими за сечение, обединение и разлика на граматични речници. С тяхна помощ ефективно се решава и проблемът с модифицирането на речника. Например ако имаме речник G, от който множеството от думи S трябва да се замени с множеството от думи N, може да се постъпи по следния начин: $G' = (G \setminus S) \cup N$.

На таблица 3.1 е дадена по-подробна количествена информация за автомата, моделиращ български граматичен речник.

словоформи	893313
състояния	47536
преходи	110105
класове	6244

Таблица 3.1: Количествени данни за автомата, моделиращ български граматичен речник.

3.2 Моделиране на речници чрез ациклични крайни преобразуватели

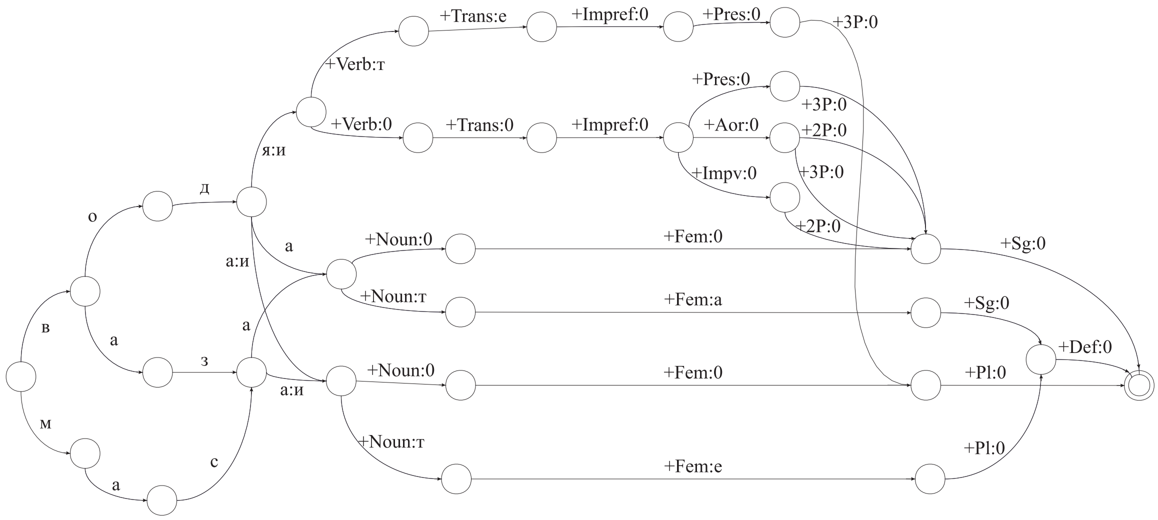
Крайните преобразуватели също са подходящи за представяне на граматични речници. Чрез модели с крайни преобразуватели се постига сравнително добра скорост при анализа на думите. Компресирането на думите, което се получава при крайните преобразуватели, също е добро. Предимството на крайните преобразуватели е, че представят речника като релация. По този начин може да се извлича информацията ефективно и в двете посоки. Трябва да се отбележи, че поради недетерминираността на преходите анализът на думите е значително по-бавен и сложен отколкото при модела с крайни автомати с етикети на крайните състояния.

Нека отново разгледаме примерния речник от предишната секция. Този път ще използваме модела и нотацията, представени в [Ка93, Ка94].

```
ваза+Noun+Fem+Sg :  ваза
ваза+Noun+Fem+Sg+Def :  вазата
ваза+Noun+Fem+Pl :  вази
ваза+Noun+Fem+Pl+Def :  вазите
вода+Noun+Fem+Sg :  вода
вода+Noun+Fem+Sg+Def :  водата
вода+Noun+Fem+Pl :  води
водя+Verb+Trans+Imperf+Aor+2P+Sg :  води
```

```
водя+Verb+Trans+Imperf+Aor+3P+Sg :  води
водя+Verb+Trans+Imperf+Impv+2P+Sg :  води
водя+Verb+Trans+Imperf+Pres+3P+Sg :  води
вода+Noun+Fem+Pl+Def :  водите
водя+Verb+Trans+Imperf+Pres+2P+Pl :  водите
маса+Noun+Fem+Sg :  маса
маса+Noun+Fem+Sg+Def :  масата
маса+Noun+Fem+Pl :  маси
маса+Noun+Fem+Pl+Def :  масите
```

Нека разгледаме даден ред от речника. Например `вода+Noun+Fem+Sg+Def : водата`. Това представяне означава, че думите `вода+Noun+Fem+Sg+Def` и `водата` са в релация. Етикетите `+Noun, +Fem, +Sg, +Def` трябва да се разглеждат като отделни символи от азбуката на автомата. Те кодират граматичната информация: съществително, женски род, единствено число, членувано. Представянето на речника като релация прави възможно, от една страна, да се получи по дадена словоформа нейната основна форма и граматична информация, а от друга страна, по основна форма и граматична информация – словоформата.



Фигура 3.2: Краен преобразувател, моделиращ граматичен речник.

На фигура 3.2 е даден крайният преобразувател, моделиращ примерния реч-

словоформи	512997
двойки в релация	618828
състояния	25883
преходи	69329

Таблица 3.2: Количествени данни за минимален нормализиран краен преобразувател, моделиращ средноголям български граматичен речник.

ник. За удобство двойките от символи при преходите са разделени с двоеточие и са без ъглови скоби. Ако символите в двойката са еднакви, е даден само единият. Преминавайки през преходите, символите от горната лента образуват основната форма и граматическите характеристики, а от долната лента – образуват словоформата.

Забележка. Крайният преобразувател на фигура 3.2 е минимален нормализиран спрямо нормализатора, даден в пример 3 от секция 1.7.

За построяването на преобразувател, представящ речник, е удобно да се приложи алгоритъм 69, като се използва преобразуванието, представено в секция 1.7. По този начин се получава минимален нормализиран ацикличен краен преобразувател. Чрез преобразуванието, представено в секция 1.7, алгоритмите от секция 2.5 могат да се приложат за сечение, обединение и разлика на граматични речници. С тяхна помощ ефективно се решава и проблемът с модифицирането на речника.

Чрез този подход се моделират удобно също така словообразувателни, правоговорни и други речници. Следва пример за представянето на словообразувателен речник:

```
възможност :  въз+мога+ен+ост
заверка :  за+вяра+ка
използване :  из+полза+ва+не
подписан :  под+пиша+ан
```

Така речникът се представя като релация между думите и тяхното разбиване на морфеми. Чрез описаните вече методи тази крайна релация се представя чрез минимален нормализиран краен преобразувател.

3.3 Индексиране на текст чрез използване на моноиден преобразувател, връщащ индекс

За обработка на корпуси от текстове се налага индексиране на текста. Индексирането се прави, като се създава речник на всички думи, които се срещат в текста, и за всяка дума се записват позициите, на които тя се среща. Целта е да се ускори изпълнението на задачите за намиране на срещанията на дадена дума, думите, които са в контекста на дадената и други. Тук ще представим подход, при който се използва моноиден преобразувател, връщащ индекс на думите. Той е разработен от автора, като се използва за база работата по индексиране на текстове, представена в [Mas96, Mas97]. Ще илюстрираме нашия метода чрез конкретен пример.

Нека ни е даден следният текст[1]:

А ти си в моя сън
и никога не си бил вън от него,
и никога не си поглеждал
отблизо в моите очи,
не си докосвал своята ръка
до моята, а се пренесе
направо в моя сън.

Речникът на думите от текста е:

[1]Използван е куплет от стихотворението на Дора Габе „Два свята“.

0	,	14	не
1	.	15	него
2	А	16	никога
3	а	17	от
4	бил	18	отблизо
5	в	19	очи
6	вън	20	поглеждал
7	до	21	пренесе
8	докосвал	22	ръка
9	и	23	своята
10	моите	24	се
11	моя	25	си
12	моята	26	сън
13	направо	27	ти

Като използваме речника на текста, можем да представим самия текст по следния начин:

позиция	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
дума	2	27	25	5	11	26	9	16	14	25	4	6	17	15	0
позиция	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
дума	9	16	14	25	20	18	5	10	19	0	14	25	8	23	22
позиция	30	31	32	33	34	35	36	37	38	39	40				
дума	7	12	0	3	24	21	13	5	11	26	1				

Срещанията на думите представяме по следния начин:

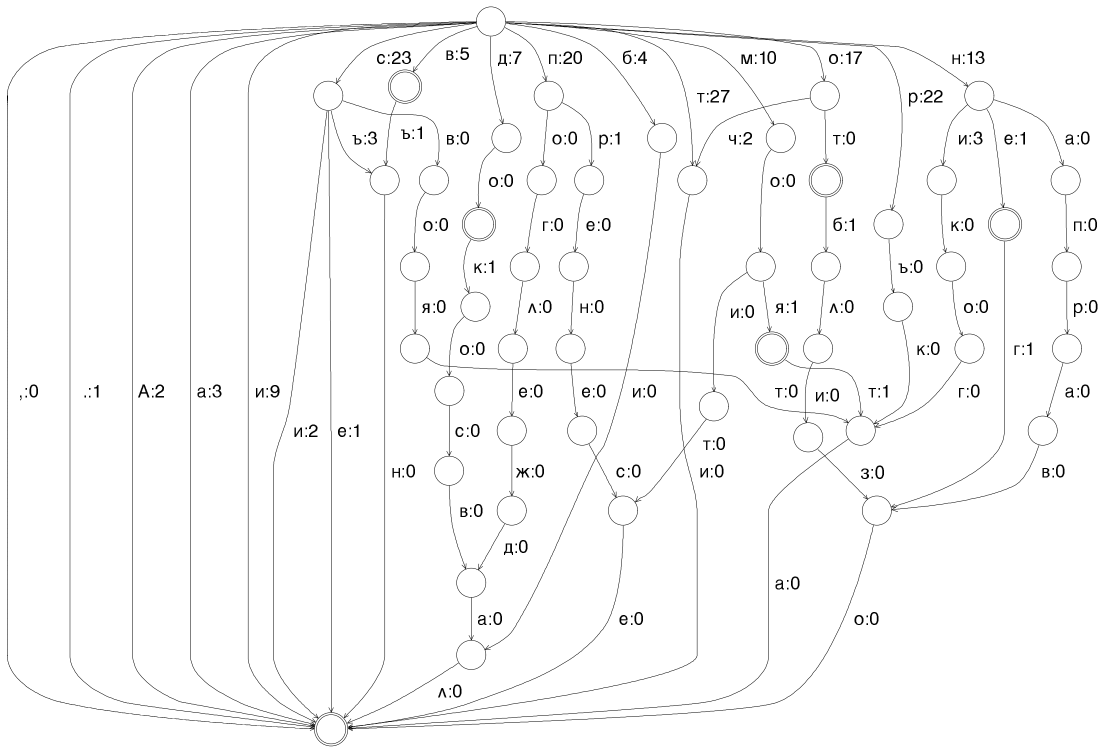
Дума	Позиции на срещанията	Дума	Позиции на срещанията
0	14,24,32	14	8,17,25
1	40	15	13
2	0	16	7,16
3	33	17	12
4	10	18	20
5	3,21,37	19	23
6	11	20	19
7	30	21	35
8	27	22	29
9	6,15	23	28
10	22	24	34
11	4,38	25	2,9,18,26
12	31	26	5,39
13	36	27	1

За представянето на речника на текста е подходящо използването на моноиден преобразувател, връщащ индекса на думите в речника. За нашия пример съответстващият преобразувател е даден на фигура 3.3[2].

Моноидният преобразувател, връщащ индекс, представящ речника на текста, се построява директно чрез алгоритъма, даден в секция 2.3. За намиране на индекса на думата в речника е подходящо да се използва алгоритъм 74, а за намирането на думата по нейния индекс – алгоритъм 75.

Прилагането на този метод върху големи корпуси от думи води, от една страна, до добра компресия, а от друга, до много добра скорост на намиране на думите. Обемът на обработения по посочения начин текст заедно с речника и информацията за срещанията става по-малък от самия текст в изходен вид. Скоростта за намирането на срещанията на дадена дума е пропорционален на дължината на думата и при добри имплементации е в порядъка на хиляди думи

[2]Автомата и диаграмата му са генерирани автоматично чрез използването на библиотеката MAFSA (виж следващата секция) и системата Graphviz на AT&T Laboratories.



Фигура 3.3: Моноиден преобразувател, връщащ индекса на думите в речника.

за секунда.

3.4 Библиотеката MAFSA

Библиотека MAFSA (Minimal Acyclic Finite State Automata) е разработена от автора за работа с крайни автомати и преобразуватели. Използван е програмният език JAVA, който позволява обектен подход.

В библиотеката MAFSA са имплементирани следните алгоритми:

- Алгоритмите за директно построяване на минимален ацикличен краен автомат (секция 2.1), минимален ацикличен автомат с етикети (секция 2.2), минимален ацикличен моноиден преобразувател, връщащ индекс (секция 2.3).
- Алгоритмите за разпознаване на дума от автомат (алгоритъм 72), за класифициране на дума от автомат с етикети (алгоритъм 73), за получаване

на индекс на дума от моноиден преобразувател, връщащ индекс (алгоритъм 74) и за получаване на дума по индекса ѝ от моноиден преобразувател (алгоритъм 75).

- Алгоритмите за сечение, обединение и разлика на автомати от секция 2.5.
- Алгоритми за работа с краен преобразувател.

Чрез тази библиотека могат да се осъществят всички модели, разгледани в секциите 3.1, 3.2 и 3.3.

Чрез използването на програмният език JAVA се осъществява пълна платформена независимост на системата. Недостатък представлява значително по--слабото бързодействие в сравнение с написаните на езика C програми. Това се дължи на факта, че сегашните реализации на JAVA отстъпват с около един порядък по скорост на C реализациите.

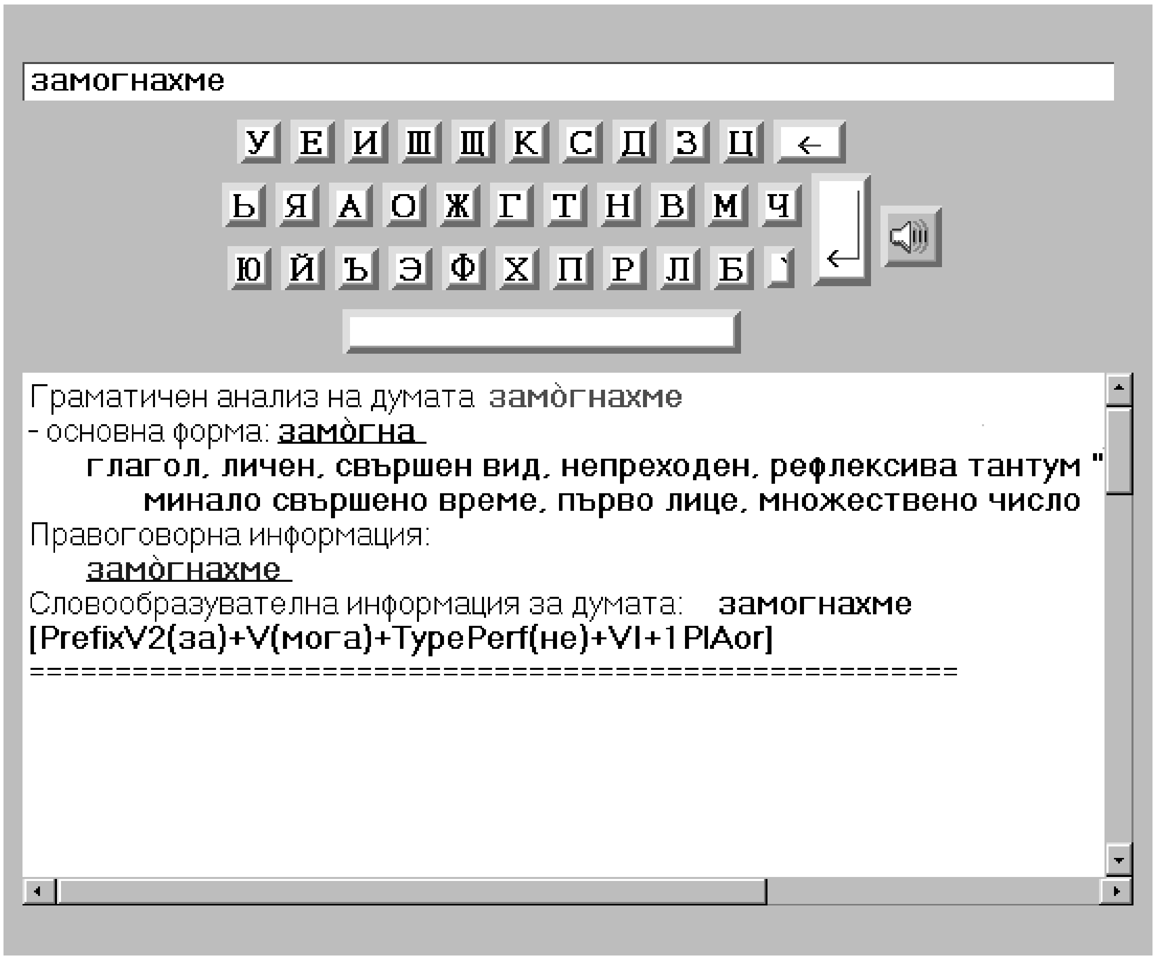
Системата Граматичен WWW сървър

Граматичният сървър е разработен при Централната лаборатория по паралелна обработка на информация (ЦЛПОИ) към БАН, като е подпомогнат от проект МУ-МИ-121/96 на фонда за научни изследвания към Министерството на образованието, науката и технологиите, ръководен от автора.

Системата предлага диалогово консултиране на широк кръг потребители с големия Граматичен, акцентуален и правоговорен речник на българския език, както и с разработващия се словообразувателен речник. Интернет ползвателите от цял свят – изучаващи, преподаващи или изследващи българския език, получават възможността да правят различни специализирани справки за българската граматика. Адресът за достъп е: **http://origin2000.bas.bg**.

Граматичната информация, която може да бъде получена, е структурирана по следния начин:

- Граматичен речник на българския език, съдържащ над 60 000 основни форми на думите от речниковия фонд със съответната им граматична и акцентологична информация. Речникът позволява автоматично анализиране (и синтезиране) на словоформите, чийто брой е около един милион.



Фигура 3.4: Граматична справка за думата *замогнахме* от граматичния WWW сървър. В долната част на екрана е дадена акцентуална, граматична, правоговорна и словообразувателна справка за думата.

Този речник е моделиран чрез минимален ацикличен краен автомат с етикети на крайните състояния.

- Словообразувателен речник на българския език, който в момента обхваща около 1350 морфеми, с помощта на които могат да се генерират около 80 000 от словоформите в Граматичния речник на българския език. Този речник е моделиран чрез минимален нормализиран ацикличен краен преобразувател.

- Система за транскрипция на думите, която преобразува правописната форма на думите в съответния правоговорен вариант. Този речник е моделиран чрез краен преобразувател.

- Система за определяне на местата за пренасяне на думите. Тази система използва краен преобразувател.
- Система за определяне на ударението, която автоматично поставя ударението на произволна словоформа, включена в Граматичния речник на българския език. Тази система използва минимален ацикличен краен преобразувател с етикети на крайните състояния.

Основните модули на системата са:

1. Модул за построяване и компресиране на граматичните данни. Този модул използва създадения метод за директно построяване на минимален краен автомат по даден списък. Този модул е реализиран чрез библиотеката MAFSA.

2. Модул за високоскоростна обработка на граматични заявки. Модулът е програмиран на JAVA и поддържа Servlet интерфейс, което позволява вграждането му в WWW сървър. Този модул осъществява връзката между WWW сървър и граматичните речници чрез използването на MAFSA библиотеката за достъп до речниците.

3. WWW сървър. Използван е програмираният изцяло на JAVA сървър „Jigsaw“, предоставен от организацията World Wide Web Consortium. Този високотехнологичен продукт позволява допълнително вграждане на Servlet модули и дава отлични скорости за достъп до ресурсите.

4. Модул за клиентите, написан на JAVA, и поддържащ Applet интерфейса. Основните функции на модула са:

   - Чрез използването на Applet интерфейса се осъществява възможността за интерпретацията му от всеки WWW броузер, поддържащ JAVA.
   - Модулът има вградени функции за визуализация на шрифтове на кирилица, поради което визуализацията на българските думи не зависи от настройките на компютрите на клиента.

- Модулът позволява въвеждането на текст на кирилица да се осъществява чрез натискането с мишка на бутони, подредени в прозорец. По този начин клиентът независимо от клавиатурните настройки на своя компютър може да използва кирилица.
- Модулът осъществява комуникацията със сървъра по HTTP протокол, използвайки GET и POST методи. Така се избягват проблемите, възникващи при използването на Proxy.
- Модулът визуализира резултата от заявката чрез графично възпроизвеждане, което позволява да се извеждат графики, снимки, схеми и др.
- Модулът предоставя възможност за аудио възпроизводство, чрез вградените в JAVA средства.

Основно предимство на използването на автоматните методи в тази система е това, че става възможно, въпреки използването на сравнително бавните (за момента) реализации на езика JAVA, да се обработват хиляди граматични заявки в секунда. Друго предимство на тази технология е пълната платформена независимост както на сървъра, така и на клиентската част.

3.5 Библиотека за обработка на корпуси от текстове

Библиотеката за обработка на корпуси от текстове, разработена в ЦЛПОИ е създадена от Богдан Александров и автора за изпълнение на задачи по COPERNICUS проектите BILEDITA, GLOSSER и TELRI. Тя е написана изцяло на GNU C/C++ и е изградена на модулен принцип. Основните модули в системата са:

- `dcomp` — Модул за построяване на граматичен речник във вид на минимален ацикличен краен автомат с етикети на крайните състояния. Използва се алгоритъмът от секция 2.2 и методите за представяне на граматичен речник от секция 3.2.

- `tindx` — Модул за създаване на речници на основните форми и словоформите на думите от текста и за индексиране на срещанията на думите. Този модул използва речник генериран чрез модула `dcomp` и алгоритъм 73.

- `ttag` — Модул за добавяне на граматична информация към всяка дума от текста. И този модул използва речник, генериран чрез модула `dcomp` и алгоритъм 73.

- `pmtu` — Модул за предварителна обработка на текста.

- `align` — Модул за подравняване на паралелен двуезичен текст.

- `trans` — Модул за намиране на преводни еквиваленти на фрази в двуезичен текст. Този модул използва резултатите от индексирането на текстовете от модула `tindx`, както и подравняването на текстовете от модула `align`.

Библиотеката е разработена главно с цел да бъде вграждана в системи за обработка на текстове. Обменът на данните става чрез файлове. При разработването на библиотеката е обърнато специално внимание върху бързодействието и възможността да работи с големи обеми от текстове. За представянето на крайните автомати в паметта е използвана техниката от [TY79]. Така се постига максимално скорост и добро компресиране едновременно.

На таблица 3.3 са дадени някои експериментални данни, представящи сравнение на ефективността на модула `dcomp` със съответния модул `compacter` за построяване на минимален автомат, вграден в системата INTEX [Si93]. Модулът `compacter` използва алгоритъма на Revuz [Re91, Ro92].

За експеримента са използвани голям граматичен речник за руски език и средноголям граматичен речник за български език. Използван е стандартът DELAF за представянето на граматичната информация.

Всички измервания са направени на компютър с 32MB RAM с процесор Pentium 180MHz, работещ под операционна система NEXTSTEP. Голямото време, необходимо за построяването на автомата, представящ руския речник за модула `compacter`, се обяснява с необходимостта от използването на виртуална памет. За българския речник разликата във времената е по-малка, но и при още

Таблица 3.3: Експериментално сравнение между модулите `compacter` и `dcomp`.

		compacter	dcomp
български	Размер (словоформи)	524473	
граматичен	Необходима памет	33450 KB	1660 KB
речник	Време	2:07 мин.	0:29 мин.
руски	Размер (словоформи)	1486552	
граматичен	Необходима памет	129000 KB	4400 KB
речник	Време	17 мин.	2:04 мин.

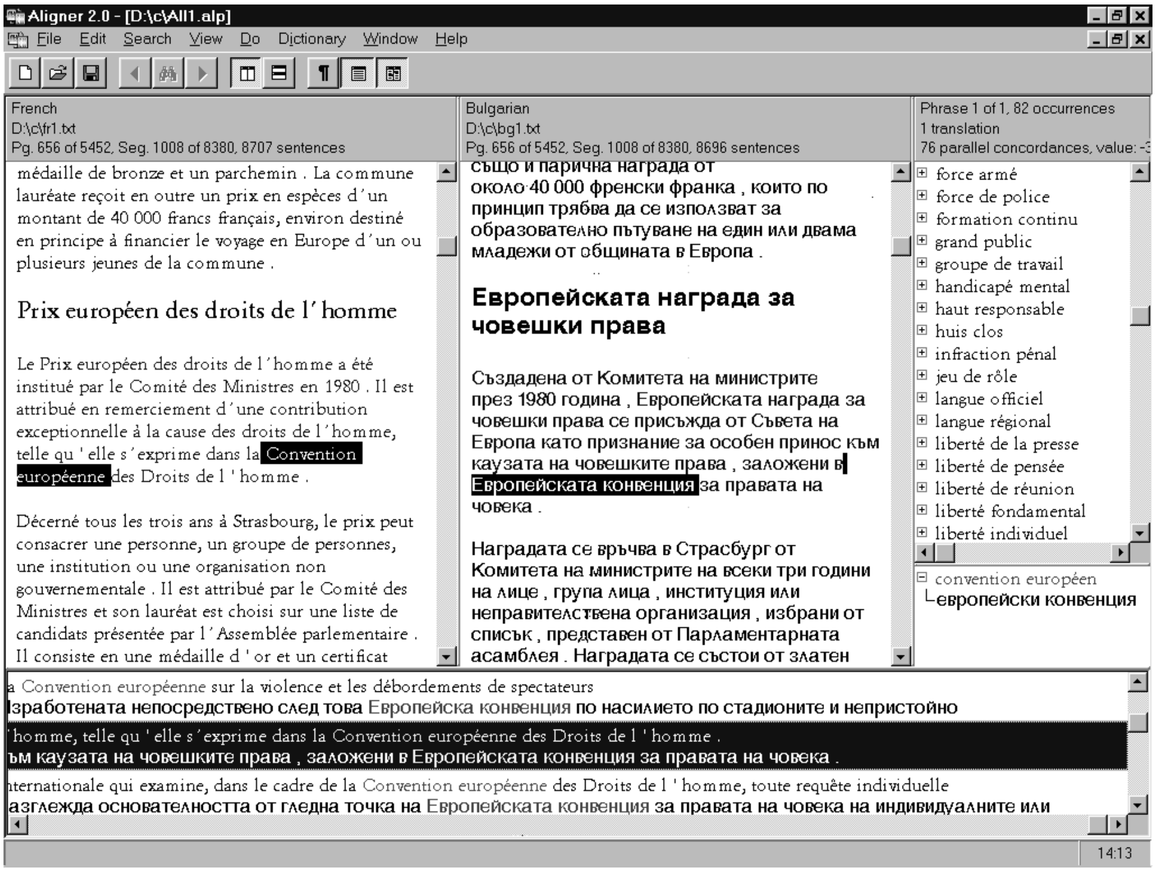
по-малки речници, когато всичко може да се обработи изцяло в оперативната памет, нашият модул остава по-бърз от модула `compacter`.

Модулът `ttag` обработва на същата машина над 30000 думи за секунда. При това скоростта е ограничена главно от времето, необходимо за осъществяване на дисковите операции. При експеримент, правен върху текстове, намиращи се изцяло в оперативната памет, модулът `ttag` обработва около 120000 думи за секунда. По тези показатели нашия модул превъзхожда всички подобни продукти, познати на автора.

Няма да се спираме подробно върху останалите модули от тази библиотека. По-нататък в тази секция ще представим две програмни системи за обработване на двуезични корпуси от текстове, които използват представените модули.

Системата MARK ALISTeR EXTRA

MARK-ALISTeR е система за обработка на паралелни двуезични текстове, създадена в ЦЛПОИ по проектите GLOSSER и TELRI. Първоначалният вариант на системата бе предназначен главно за автоматизирано подравняване на двуезичните текстове. Предварителната обработка на текста се извършва чрез модула `pmtu`, а автоматичното подравняване между двата езика – чрез модула `align`. Разширението на системата — MARK-ALISTeR EXTRA допълнително позволява да се извършва и автоматично извличане на преводни еквиваленти на терминологични фрази и изрази. Съществeният нов момент е провеждането на граматичен анализ на думите от двата текста чрез модула `tindx`. На базата на това, чрез модула `trans` се извършва статистическа обработка, която се основава на статистически анализ на съвместното появяване на търсените лексеми в подравнените сегменти на текстовете.



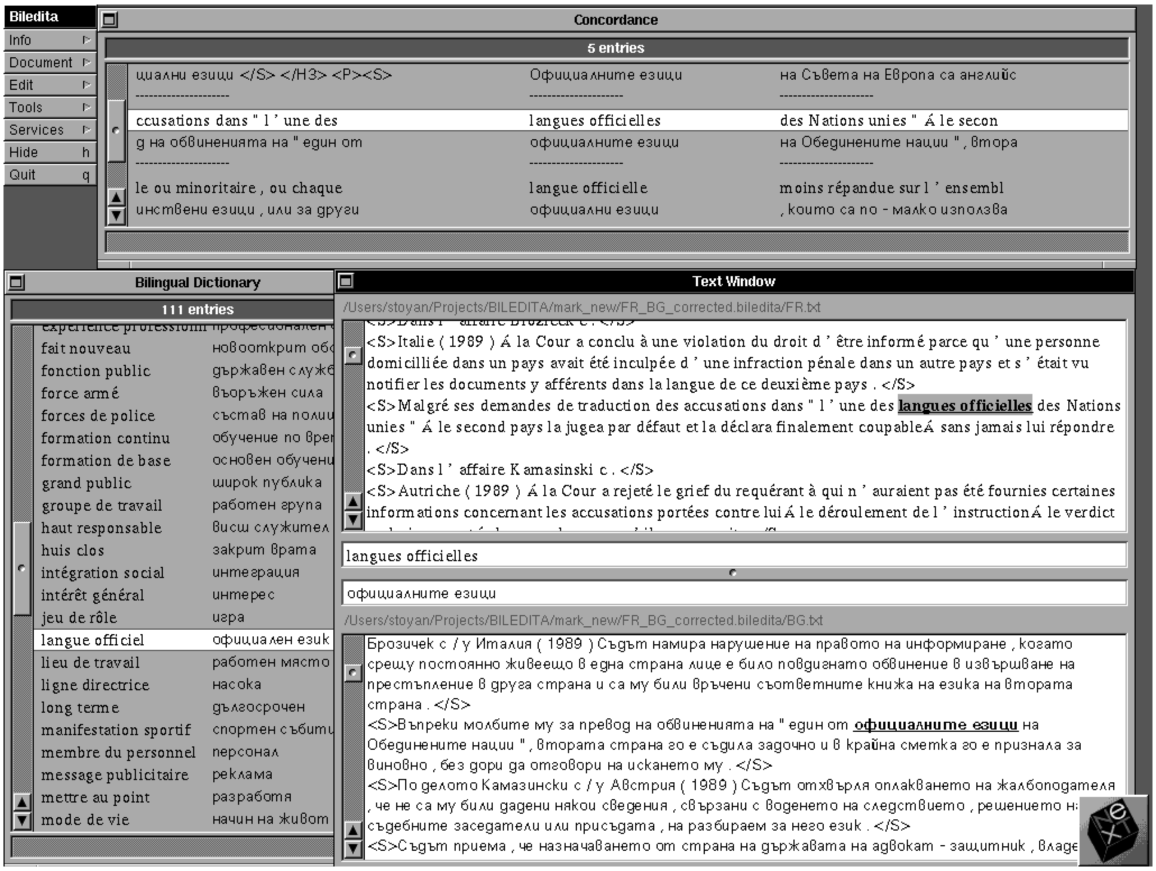
Фигура 3.5: Системата MARK-ALISTeR EXTRA.

На фигура 3.5 е даден екран от MARK-ALISTeR EXTRA. В десния прозорец е показан създаденият терминологичен двуезичен речник. Избраният израз е маркиран. В долния прозорец се намира паралелен конкорданс на превода на този израз. В прозорците вляво горе са дадени двата текста, като са маркирани сегментите, в които се среща търсеният израз.

По-подробно описание на възможностите на системата MARK-ALISTeR EXTRA може да се намери в [PM97, Mi97a, Mi97b].

Системата BILEDITA

BILEDITA е система за обработка на паралелни двуезични текстове, създадена по едноименния проект. В нея също са интегрирани модулите `tindx`, `pmtu`, `align` и `trans`. Основната разлика в сравнение със системата MARK-ALISTeR EXTRA се състои в използването на системата INTEX за някои предварителни обработки на текста, както и за извличането на терминологични фрази чрез граматични шаблони от изходния текст.



Фигура 3.6: Системата BILEDITA.

На фигура 3.6 е даден екран от BILEDITA. В левия долен прозорец е представен вече създадения терминологичен двуезичен речник. Избраният израз от речника е маркиран. В горния прозорец е даден паралелен конкорданс на превода на този израз. В прозорците вдясно долу са показани двата текста, като са маркирани сегментите, в които се среща търсеният израз.

Описание на възможностите на системата BILEDITA е дадено в [BI97].

Библиография

[Бу78] Анатолий Буда: *Абстрактные машины программ.* Академия наук СССР Сибирское отделение, Препринт 108, Новосибирск 1978.

[ДПД84] Йордан Денев, Радослав Павлов, Янош Деметрович: *Дискретна математика.* Наука и изкуство. София 1984.

[Ма96] Красимир Манев: *Увод в дискретната математика.* Нов български университет. София 1996.

[Ми93] Стоян Михов: *Унификация на корегулярни множества.* Дипломна работа, Софииски университет „св. Климент Охридски", Факултет по математика и информатика, Декември 1993.

[ТК97] Георги Тотков, Христо Крушков: *Математическо моделиране и изграждане на система от машинни речници.* Автоматика и информатика, бр.5-6, 1997.

[AHU83] A. Aho, J. Hopcroft, J. Ullman: *Data Structures and Algorithms.* Addison-Wesley, Reading, Massachutes, 1983.

[An90] Evan Antworth: *PC-KIMMO: a two-level processor for morphological analysis.* Summer Institute of Linguistics, Dallas, 1990.

[An94] Evan Antworth: *Morphological Parsing with a Unification-based Word Grammar.* North Texas Natural Language Processing Workshop, 1994.

[BBHECS85] A. Blummer, J. Blummer, D. Haussler, A. Ehrenfeucht, M. T. Chen, J. Seiferas: *The Smallest Automaton Recognizing the Subwords of a Text.* Theoretical Computer Science, Vol. 40, 1985.

[BI97] EC Project BILEDITA Copernicus #790 *Final Report*. July 1997.

[Br84] Wilfried Brauer: *Automatentheorie*. B. G. Teubner Stuttgart, 1984.

[CV97a] M. Crochemore and R. Vérin: *Direct construction of Compact Directed Acyclic Word Graphs*. In CPM97, A. Apostolico and J. Hein, eds., LNCS 1264, Springer-Verlag, 1997.

[CV97b] M. Crochemore and R. Vérin: *On Compact Directed Acyclic Word Graphs*. In Structures in Logic and Computer Science, a selection of essays in honor of A. Ehrenfeucht, J. Mycielski, G. Rozenberg and A. Salomaa, eds., LNCS 1261, Springer-Verlag, 1997.

[DWW98] Jan Daciuk, Bruce Watson, Richard Watson, *Incremental Construction of Minimal Acyclic Finite State Automata and Transducers*. Proceedings of Finite State Methods in Natural Language Processing, Ankara, July, 1998.

[GT94] Gregory Grefenstette and Pasi Tapanainen: *What is a Word, What is a Sentence? Problems of Tokenization.* Proceedings of The 3rd International Conference on Computational Lexicography (COMPLEX'94), Budapest, 1994.

[Ho71] J. Hopcroft: *An $n \log n$ algorithm for minimizing the states in a finite automaton*. The Theory of Machines and Computations (Z.Kohavi, ed.), Academic Press, New York, 1971.

[HU79] John Hopcroft and Jeffrey Ullman: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[Ka93] Lauri Karttunen: *Finite-State Lexicon Compiler*. Technical Report. ISTL-NLTT-1993-04-02. Xerox PARC. Palo Alto, California. April 1993.

[Ka94] Lauri Karttunen: *Constructing Lexical Transducers*. Proceedings of COLING'94, 1994.

[Ka95] Lauri Karttunen: *The Replace Operator*. Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL-95), Boston, 1995

[Ka96] Lauri Karttunen: *Directed Replacement*. ACL-96, Santa Cruz, California, June 1996.

[KB92] Lauri Karttunen, Kenneth R. Beesley: *Two-level Rule Compiler*. Technical Report. ISTL-92-2. Xerox PARC. Palo Alto, California. June 1992.

[KCGS96] Lauri Karttunen, Jean-Pier Chanod, Gregory Grefenstette, Anne Schiller: *Regular Expressions For Language Engineering*. Natural Language Engineering 2 (4), Cambridge University Press, 1996.

[KGK97] Lauri Karttunen, Tamás Gaál, André Kempe: *Xerox Finite-State Tool*. Technical Report. Xerox Research Centre Europe, Grenoble. June 1997.

[KK94] Ronald Kaplan, Martin Kay: *Regular Models of Phonological Rule*. Computational Linguistics, Vol. 20(3), 1994.

[KK96] André Kempe, Lauri Karttunen: *Parallel Replacement in Finite-State Calculus*. Proceedings of COLING-96, Copenhagen, 1996.

[KKZ92] Lauri Karttunen, Ronald Kaplan, Annie Zaenen: *Two-level Morphology with Composition*. Proceedings of COLING'92, 1992.

[Ko83] Kimmo Koskenniemi: *Two-level morphology: A general computational model for word-form recognition and production*. Publication No. 11, Department of General Linguistics, University of Helsinki, 1983.

[Mas96] Oliver Mason: *Corpus Access Software: The CUE System*. TEXT Technology, Vol 6, No 4, 1996.

[Mas97] Oliver Mason: *CUE - A Software System for Corpus Analysis*. Proceedings of the Second European Seminar "Language Applications for a Multilingual Europe", Kaunas, April 1997.

[Mi96] Stoyan Mihov: *A Constraint Based System for Lexical Knowledge Retrieval.* Annuaire de l'Universite de Sofia “St. Kl. Ohridski”, Faculte de Mathematique et Informatique, volume 90, livre 2, 1996.

[Mi97a] Stoyan Mihov: *MARK ALISTeR: MARKing, ALIgning and Searching TRanslation equivalents.* Proceedings of the Second European Seminar “Language Applications for a Multilingual Europe”, Kaunas, April 1997.

[Mi97b] Stoyan Mihov: *Automatic Extraction of Translation Equivalents from Aligned Corpora of Legal Texts.* Proceedings of the Third European Seminar “Translation Equivalence – Theory and Practice”, Montecatini, October 1997.

[Mi98] Stoyan Mihov: *Direct Building of Minimal Automaton for Given List.* Annuaire de l'Universite de Sofia “St. Kl. Ohridski”, Faculte de Mathematique et Informatique, volume 91, livre 1 (В печат).

[Mi99] Stoyan Mihov: *Direct Construction of Minimal Acyclic Finite States Automata.* Annuaire de l'Universite de Sofia “St. Kl. Ohridski”, Faculte de Mathematique et Informatique, volume 92 (В печат).

[Mo95] Mehryar Mohri: *On Some Applications of Finite-State Automata Theory to Natural Language Processing.* Natural Language Engineering 1 (1), 1995.

[Mo97] Mehryar Mohri: *Finite-state transducers in language and speech processing.* Computational Linguistics, Vol. 23 (2), 1997.

[PM97] Elena Paskaleva, Stoyan Mihov: *Second Language acquisition from Alignedned Corpora.* Proceedings of the International Conference “Language Technology and Language Teaching”, Groningen, April 1997.

[PSDS93] Elena Paskaleva, Kiril Simov, Mariana Damova, Milena Slavcheva: *The long journey from the core to the real size of a large LDB.* In Acquisition of Lexical Knowledge from Text (eds. Boguraev, Pustejowski), Columbus, Ohio, 1993, 161 - 169.

[Re91] Dominique Revuz: *Dictionaires et lexiques, méthodes et algorithmes.* Ph.D. dissertation, Université Paris 7, Paris, 1991.

[Re92] Dominique Revuz: *Minimisation of acyclic deterministic automata in linear time.* Theoretical Computer Science, Vol. 92 (1), 1992.

[Ro92] Emmanuel Roche: *Dictionary Compression Experiments.* Technical report. LADL, Université Paris 7, Paris, 1992.

[RS95] Emmanuel Roche and Yves Schabes: *Deterministic Part-of-Speech Tagging with Finite-State Transducers.* Computational Linguistics, Vol. 21 (2), 1995.

[Sa81] A. Salomaa: *Jewels of Formal Language Theory.* Computer Science Press, Rockville Md., Springer-Verlag, Berlin, 1981.

[Sc96] Anne Schiller: *Multilingual Finite-State Noun Phrase Extraction.* ECAI '96 Workshop on "Extended Finite State Models of Language". August 11-12, Budapest, 1996.

[Si93] Max Silberztein: *Dictionnaires électroniques et analyse automatique de textes: le système INTEX.* Masson, Paris, 1993.

[TY79] Robert Tarjan and Andrew Yao: *Storing a sparse table.* Communication of the ACM, 22(11), 1997.

[Wa93a] Bruce W. Watson: *A Taxonomy of Finite Automata Construction Algorithms.* Computing Science Report 93/43, Faculty of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1993.

[Wa93b] Bruce W. Watson: *A Taxonomy of Deterministic Finite Automata Minimization Algorithms.* Computing Science Report 93/44, Faculty of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1993.

[Wa95] Bruce W. Watson: *Taxonomies and Toolkits of Regular Language Algorithms*. Ph.D. dissertation, Faculty of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, September 1995.