STEM Approach to Build Competences from Automata and Languages Theory

Juliana Dochkova-Todorova
"St. Cyril and St. Methodius" University of Veliko Tarnovo, Bulgaria
doskova@ts.uni-vt.bg

Ivaylo Donchev
"St. Cyril and St. Methodius" University of Veliko Tarnovo, Bulgaria
i.donchev@ts.uni-vt.bg

STEM подход за изграждане на компетентности от теория на автоматите и езиците

Abstract

The paper presents a STEM approach to teaching undergraduate computer science students. This approach relies on interdisciplinary connections between mathematics and informatics and uses programming as a didactic tool. STEM educational methods applicable to computer science majors are explored and analyzed. A system of learning tasks for algorithms dealing with finite automata, stack automata, and nondeterministic automata is presented. Attention is paid to the formation of lasting knowledge and skills, increasing the motivation of the students and the application of theoretical statements in practice.

Keywords: *Cross-curricular Connections; STEM; Computer Science; Automata; Formal Languages; Programming.*

1. Увод

Една от най-важните способности, отличаващи добрия програмист, е способността му да мисли. В научната литература се изследват различни видове мислене – математическо, логическо, аналитично, абстрактно, алгоритмично, изчислително или компютърно.

Изграждането на математическо мислене е цел на обучението и добре изследвана област по отношение на стимулирането му, процесите в него и изградените умения и способности [1]. То предполага използване на математически принципи при решаване на реални проблеми. Логическото мислене произтича от способността да се анализират ситуации и проблеми и да се свързват факти и събития с цел достигане на заключения и решения. Абстрактното мислене е форма на логическото мислене, при която подробностите се пренебрегват, а се вземат предвид важните в конкретната ситуация характеристики на обектите. От гледна точка на информатиката информацията се анализира и синтезира и се разпознават модели и връзки при сложни системи и данни, за да се развият стратегии за решение. От математическа гледна точка абстракциите се идентифицират с класове на еквивалентност.

Алгоритмичното мислене е умението даден проблем да се разбие на последователност от стъпки, за всяка от които се намира отделно решение, а крайното решение на първоначалния проблем се базира на тези отделни решения. Изчислителното мислене (computational thinking) е концепция, която наподобява алгоритмичното, логическото и абстрактното мислене. В нея са включени формализирането на изчислителните проблеми и формалното описание на алгоритъма за решаването им [2]. От друга страна е важно също да се мисли и говори прецизно – да се формулират

недвусмислено и точно мисли и твърдения, да се различава вярно от възможно вярно. Информатиците, както и математиците, трябва да могат да използват точни и еднозначни формулировки за дефиниции и твърдения и да доказват тяхната истинност. Доказателствата на твърдения имат обаче по-различно значение за информатиците [3] – важно е по-скоро доколко доказателствата допринасят за конструиране на нужния алгоритъм и за решаването на даден проблем.

При проектирането на алгоритми се използват общи подходи, които трябва да се адаптират към конкретния проблем. Общите подходи се изучават в отделните учебни дисциплини, но проектирането на алгоритми идва с практиката и, както при всички умения, колкото повече се практикува, толкова по-успешно става [2].

Изграждането на модел, съответстващ на даден проблем, и намирането на решение са задачи, с които програмистите се сблъскват ежедневно в своята работа. При това се изискват познания за математически понятия, концепции и твърдения, както и основни модели и алгоритми, които да бъдат приложени за решението на разглеждания проблем. Така създаването на адекватен модел се опира от една страна на математическото моделиране, определящо подходящ математически апарат за проектирането на модела, а от друга – на компютърното моделиране, насочено към алгоритмите и изчислимите функции. През 1936 г. английският математик Алън Тюринг дефинира изчислителен модел, наречен машина на Тюринг, който по своята същност представлява формална дефиниция на алгоритъм. Изградените модели са от значение за изчислителното мислене, защото решаването на проблема се извършва в рамките на модела. Те трябва да са достатъчно прости за разбиране, но и достатъчно сложни, за да описват съществените аспекти на проблема. Освен това абстрактните модели дават универсално решение за повече практически проблеми.

Дискретната математика и нейните основополагащи принципи имат своето приложение в софтуерното инженерство, но също така са разработени специализирани математически теории и методи [4] за изграждането на тези модели, които са необходими в програмирането. Такива са например крайните автомати, които се използват за моделиране на изпълнението на програма, и графите, полезни при моделиране на данни в структура, компютърни мрежи, карти и др.

Построяването на модел започва с теоретично представяне на изследвания обект [5] чрез концептуален модел, доразвит след това чрез формалния език на математиката в математически модел и чрез алгоритмите и програмите в алгоритмичен модел. Формализацията на модела се подчинява на строгите математически методи и го представя чрез еднозначно описание.

В описанието на изчислителен проблем може да бъдат включени различни ограничения по отношение на характеристиките на модела или на решението, свързани с паметта или използвания инструментариум от основни алгоритми и структури [6], [7] и [8].

Сред основните цели [9] на обучението на студенти по дисциплини от областта на теоретичната информатика са:

- запознаване с теоретичните постановки и значението им за практиката
- избор на математически модели и методи за решаването на конкретни проблеми от практиката
- логически анализ на обекти и процедури
- умение за оценка на полученото решение и неговите параметри съобразно изходния проблем
- развитие на логическото и алгоритмичното мислене.

Студентите срещат немалко трудности в разбирането и усвояването на теоретичните постановки, които произхождат най-често от недобре развито абстрактно

мислене, пропуски в математическите знания и недостатъчно време, отделено за самостоятелна работа. Липсата на достатъчно опит с теоретичния подход към решаването на проблемите ги затруднява при изграждането на модели и адаптирането на общите алгоритми към конкретни задачи. При теоретичния подход за конструиране на алгоритми има „проблеми, които изглеждат трудни, но всъщност са много лесни за решаване, и проблеми, които изглеждат лесни, но всъщност са доста трудни за решаване" [10]. От своя страна преподавателската работа в известна степен преминава на ниво метамислене, т.е. мислене за мисленето, защото преподавателите трябва да изпълняват функции по контрола, наблюдението и оценката на процесите на разсъждение, водещи до вземането на решения от страна на обучаемите.

В настоящата статия следва да бъде представена система от задачи за крайни автомати и машини, която се основава преди всичко на надграждане в алгоритмичните подходи при различните автоматни модели и има за цел по-ясно разбиране на използваните методи, плавно повишаване на сложността, акцентиране на взаимовръзките и прилаганите логически принципи и постигане на трайни знания по компютърно моделиране.

2. STEM МЕТОДИ НА ОБУЧЕНИЕ, ПРИЛОЖИМИ ЗА КОМПЮТЪРНИ СПЕЦИАЛНОСТИ

Когато се говори за STEM обучение, обикновено акцентът е върху училищното образование в прогимназиален и гимназиален етап. Трябва обаче да се има предвид, че терминът „STEM образование" обхваща „обучителни дейности във всички нива на образованието – от предучилищна до постдокторантска степен – както във формална (напр. класна стая), така и в неформална (напр. извънучилищна) форма" [11]. Ясно е, че не всички разработени и апробирани методи за обучение в училищен етап са приложими и към по-горните степени на обучение или поне изискват сериозна адаптация и модификация. В тази статия се описва един такъв STEM метод, който изгражда интердисциплинарни връзки между математиката и информатиката, в частност между дискретната математика и програмирането.

Сред учените няма консенсус по отношение на това какво точно представлява STEM в обучението – едни акцентират върху превръщането на науката и математиката в атрактивни за учениците дисциплини; други залагат на интегрирането на дигиталните технологии в учебните предмети; трети промотират развитието на „STEM умения", съобразени с подготовка на учениците за бързо променящия се свят. За нуждите на настоящата статия най-точно отговаря определението „Интердисциплинарна проектна работа или учебни предмети, които комбинират две или повече от STEM дисциплините, основани на „автентични" контексти и фокусирани върху решаването на проблеми" [12].

В STEM обучението „компютърното програмиране се позиционира като жизненоважна дисциплина, обединяваща междупредметни знания и насърчаваща развитието на капацитета за решаване на задачи от реалния живот чрез фундаментални концепции и логически методологии, присъщи на компютърните науки" [13].

Отчитайки важността на програмирането за повишаване на конкурентоспособността на учениците и развиване на STEM умения, много държави го наложиха като задължителна дисциплина в гимназиалното образование. В България учениците се сблъскват с форми на програмиране още в общообразователната подготовка в прогимназиален етап в учебните програми по предметите „Компютърно моделиране" и „Компютърно моделиране и информационни технологии". Основните принципи на програмирането се изучават в 8. клас по предмета „Информатика" (усвоява се обектно

ориентиран език), а по „Информационни технологии" в 8., 9. и 10. клас учениците се докосват до уеб програмирането (ползва се скриптов език).

На ниво висше образование в професионално направление 4.6 програмирането е широко застъпено в много дисциплини и се счита по-скоро за умение, отколкото за учебен предмет. Традиционните подходите за обучение го използват като дидактически инструмент за усвояване на понятията от информатика още в уводните курсове.

В STEM практиката са се доказали няколко ефективни метода (подхода) за преподаване на програмиране. Те са приложими както за училищно, така и за университетско ниво.

Обучение, базирано на игри (Game-Based Learning, GBL)

Този метод се счита за най-ефективен по отношение на ангажиране на вниманието и запазване на интереса на учениците. Тук стратегиите са две – учене чрез игра и учене чрез създаване на игра. И при двете включването на програмиране в процеса позволява на учениците/студентите да усвоят в ангажираща вниманието интерактивна среда сложни понятия, свързани с алгоритми, структури от данни и софтуерно тестване. Също така методът помага да се преодолее границата между теорията и практиката, позволявайки учениците да приложат своите знания в реална среда. За децата платформи като Scratch и Blockly са подходящи за изграждане на умения за изчислително мислене и основни програмни умения чрез визуален интерфейс и интерактивни игри. Новото поколение студенти, също като децата, е свикнало с технологиите от най-ранна възраст и среща трудности с концентрацията по време на лекции и упражнения, ако те се провеждат по традиционния начин, при който преподавателят обяснява понятията и демонстрира прилагането им в части програмен код. Затова се търсят нови подходи, които се базират на технологиите, за преподаване на програмиране. В [14] е направен добър обзор на обучителни игри за STEM обучение по програмиране. Изводът е, че „образователните игри са ефективен механизъм за повишаване на мотивацията и ангажираността на учениците, подобряване на резултатите им и постигане на по-добро потребителско изживяване". Пак там [15] са представени две сериозни обучителни игри за курсове по програмиране, част от проекта NEWTON, които са насочени към понятията „променлива" и „цикъл".

Като недостатък на GBL подхода може да се посочи, че създаването на добра обучителна игра е продължителен и скъпоструващ процес, изискващ внимателно планиране и множество умения, които не всеки притежава. Освен това продължителната игра може да доведе и до неприятни последици като пристрастяване.

Проектно базирано и проблемно базирано обучение (Project-Based and Problem-Based Learning)

Някъде могат да се срещнат като отделни подходи, но понеже често се прилагат заедно в STEM обучението, тук са разгледани като един метод. Подходът, описан в настоящата статия също попада в тази група. Проектно базираното учене ангажира студентите да работят по конкретен проект (групово или индивидуално) продължително време. Това помага за свързване на теоретичните знания с приложението им в практиката. Предимство при груповата работа е, че учениците получават нови роли и се учат да комуникират помежду си, придобиват важни за работата в екип качества. Проблемно базираното обучение от своя страна залага на решаването на задачи от реалния живот, поставени от самите студенти или такива, за които няма директно интуитивно решение. Студентите трябва да проучват задачата, да си сътрудничат и да прилагат знанията си, за

да достигнат до решението. Така се развиват критично мислене и умения за решаване на проблеми, задълбочава се разбирането на принципите и понятията чрез реализирането им в практически и сложни условия. С помощта на тези подходи студентите виждат как програмирането и другите STEM умения могат да се прилагат извън класната стая. Подходът е приложим както за студенти, така и за ученици. Основно предимство е, че с него, освен че се развиват STEM уменията, се правят и силни междупредметни връзки. Пример за такава от вида физика – информатика откриваме в [16], където ученици от 9. клас в рамките на цялостен STEM проект изграждат и програмират собствена метеостанция. Интересна връзка висша математика – информатика е описана в [17]. Там предмет е програмното решаване на определени интеграли.

Като препоръка, добра идея е в учебните планове на компютърните специалности да се предвиди така наречения "capstone project" – работа по заключителен проект, която да обхваща цял семестър (последен или предпоследен). Заключителният проект изисква студентите да интегрират и приложат натрупаните знания към изграждането на цялостно решение. Тази работа може да включва задачи като разработка на софтуерни приложения, симулации или роботизирани системи. Обикновено изисква много програмиране. Може да бъде и част от стажантската практика, за да се получи реален опит и потенциална възможност за привличане на работа.

Обучение, базирано на проучване (Inquiry-Based Learning, IBL)

Този подход насърчава студентите да проучват и задълбочено да изследват понятията чрез цикличен процес, включващ фази на ангажираност, изследване, обяснение, разработка и оценка [14]. Акцентът е върху проучването и решаването на проблеми. Затова най-подходящи са задачи, които изискват критично и творческо мислене. Така учениците изграждат STEM умения да задават правилните въпроси, да организират изследване, да интерпретират фактите, да се аргументират и да представят проектите си. Същността на подхода е вместо да се дава готово решение, учителят да премине през набор от сценарии, въпроси и задачи, които насочват учениците към решението. Подходът доказано носи предимства и развива критично и креативно мислене [18]. Приложим е както за лабораторна работа, така и за лекции. IBL може да варира от дейности, насочени към учителя, до дейности, насочени към ученика, което позволява различни степени на независимост в процеса на обучение.

Съвместно обучение (Collaborative Learning)

Когато става въпрос за обучение по програмиране на университетско ниво, типичен представител на съвместните подходи, който често прилагаме в нашата работа, е програмирането по двойки (pair programming). Двама студенти работят заедно на един компютър, като единият пише кода, а другият го следи, дава идеи и коригира евентуални грешки. Ролите на студентите в двойката се сменят периодично. Този метод насърчава съвместното учене, изгражда умения за решаване на проблеми, подобрява комуникативните умения и уменията за работа в екип. Обучението в колаборативна среда създава подкрепяща атмосфера, в която учениците могат да споделят знания и колективно да се борят с проблемите.

В следствие на ковид пандемията и развитието на информационните и комуникационни технологии се разпространи модификация на този метод – разпределено програмиране по двойки – distributed pair programming (DPP), при което двойките работят на отделни машини, но комуникират пълноценно в електронна среда. Това е добро решение за прилагане на подхода в хибридна обучаваща среда. Разработиха се и

специални инструменти за DPP, като този, описан в [19], които допълват възможностите и преодоляват ограниченията на съществуващите интегрирани среди за разработка и уеб базирани редактори.

Изкуствен интелект (ИИ) в STEM обучението

ИИ технологиите все повече се интегрират в STEM обучението. Целта е да се предостави персонализирано учебното съдържание и да се осъществи „интелигентно" преподаване. Конкретно в обучението по програмиране ИИ може да предостави адаптивни обучаващи платформи, автоматизирана обратна връзка и персонализирани препоръки. Тези технологии помагат да се следва индивидуално за всеки ученик темпо и стил на учене и да се подобри на цялостната атмосфера на протичане на учебния процес.

Добър анализ и обобщение на постигнатите до момента резултати в интегрирането на ИИ в обучението с акцент върху изследванията, практиката и технологичните промени на ниво парадигма може да се открие в [20].

3. ФОРМАЛНИТЕ ОСНОВИ НА ПРОГРАМИРАНЕТО

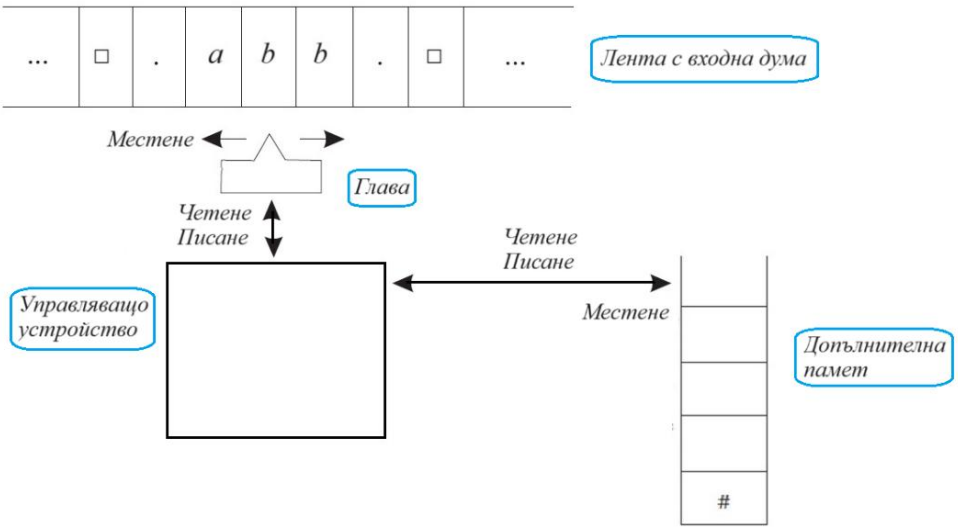
Теорията на формалните езици и автомати се основава на разработките на Алън Тюринг и идеите на Ноам Чомски и има съществена роля в създаването и използването на езиците за програмиране. През последните тридесет години се наблюдават по-активни теоретични изследвания като следствие от подема в области на приложение като изкуствен интелект и компютърна обработка на естествените езици.

Понятията „дума" и „език" са основни в информатиката. Думите са последователности от букви. Азбука наричаме множество от знаци – латински букви, гръцки букви, цифри, скоби и др. Формалният език представлява произволно множество от думи. По този начин реалният свят на езиците е опростен, защото думите и изреченията са равнопоставени – изреченията са думи, състоящи се от слепени чрез операцията конкатенация думи. От гледна точка на тази теория променливите, числата и аритметичните изрази, които се използват в една програма, представляват думи над зададена азбука.

Ще разгледаме четири основни абстрактни изчислителни модела. Чрез тях се описват алгоритми, а следователно и програми, като те се моделират формално. Тук дидактическите цели са свързани с това, как по един по-разбираем и прост начин да се създаде модел на изчислителния алгоритъм и да се въведат понятия като изчисления, симулация, детерминизъм и др. [21].

При задаване на изчислителен модел трябва да се отговори на следните основни въпроси: какви елементарни операции ще се прилагат; с каква памет ще се разполага; как ще се обработват входните данни и как ще се извеждат изходните данни [21]. На фиг. 1 е дадена обща схема на изчислителен модел, при който част от елементите и характеристиките не са задължителни: допълнителна памет, възможност за писане, четене или местене.

В изложението по-нататък няма да бъдат посочени формалните дефиниции на разглежданите четири машини, а ще се акцентира най-вече на начина им на работа и алгоритмите за обработка на входните думи, които те задават. Точните дефиниции са налични във всеки учебник по теория на автоматите, например в [21], [22] и [23].



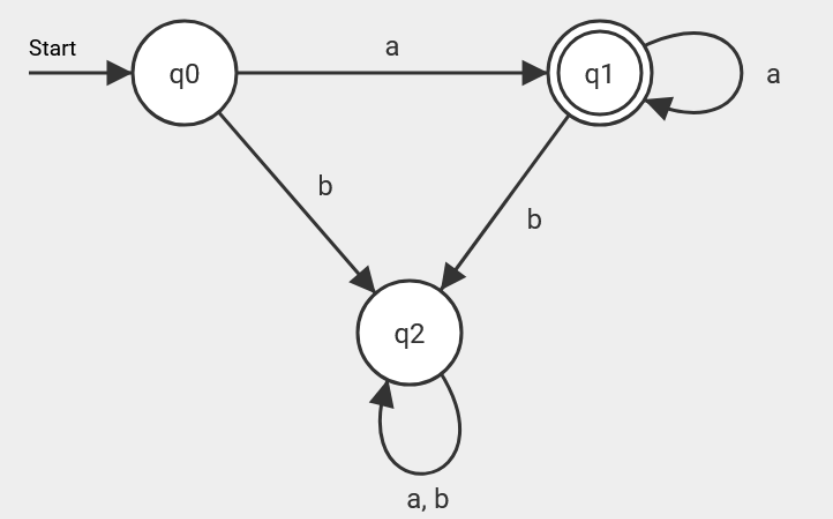
Фиг. 1. Схема на изчислителен модел

Крайните автомати са най-елементарният изчислителен модел, използван в информатиката. При тях е необходима съвсем малко памет и чисто алгоритмично те съответстват на специални програми [21], които решават определени проблеми без да използват променливи. Освен това изходният резултат се получава веднага след прочитане на последната входна буква. Единствената използвана памет при тях е тази, в която се записва указателят към достигнатата за момента команда. Не е нужно да се запомня друга информация в самата програма – указателят е единственият, който се променя и има значение за изхода. Всяка команда в програмата е отбелязана с указател и всяка команда представлява команда за многовариантен избор, в която в зависимост от входа се преминава към даден указател на следващата команда. Вместо многовариантен избор може да се използва и условен оператор, ако възможните входни букви, а с това и изборите, са малък брой. Програмата винаги започва с първата команда и чете входните букви последователно. Част от командите са наречени заключителни и ако програмата завърши в някой от тях, се извежда изход „Да, програмата разпознава тази входна дума", в противен случай изходът е „Не се разпознава".

Крайният детерминиран автомат (КДА) представлява абстрактна математическа машина, която се състои от краен брой състояния – това са командите от еквивалентната на автомата програма. Едно от състоянията е начално състояние и от него започва работата на автомата, а част от състоянията са заключителни. Освен това е налична входна лента с клетки, които съдържат входната дума, и глава, която чете съдържанието на клетките в посока от ляво надясно. Работата на автомата зависи от функция на преходите, определяща следващото състояние в зависимост от два аргумента: текущото състояние и текущата входна буква. Функцията на прехода се нарича още програма на автомата, защото определя действията му.

Работата започва с четене на най-левия символ, записан на лентата, а текущо е началното състояние. След всяка прочетена буква от входната лента главата се премества с една клетка надясно, а автоматът преминава в ново състояние, определено от функцията на прехода. Текущото състояние и текущата буква на входа определят следващото състояние. Крайният автомат е изчислително устройство със силно ограничена памет, което след прочитане на входната дума, я разпознава или не я разпознава в зависимост от състоянието, в което се намира автоматът – дали то е заключително. Автоматите се представят най-често чрез ориентирани графи, наречени диаграми.

Пример 1. На фиг. 2 е представен чрез диаграма един КДА. При задаване на входна дума *aa* автоматът преминава през състоянията q_0, q_1, q_1. Състоянието q_1 е заключително и затова КДА разпознава тази входна дума. За думата *ba* работата завършва в незаключителното състояние q_2 и затова тази дума не се разпознава. Езикът на всички думи, които един КДА разпознава, се нарича език, разпознаван от автомата, и в този случай това е $\{a^n | n \geq 1\}$. Езикът се определя от всички пътища в графа, започващи от началното състояние и завършващи в някое заключително.



Фиг. 2. КДА от Пример 1

Според описаната по-горе връзка между програма и модел даденият КДА съответства на следната програма:

```
0: if symbol = a then goto 1 else goto 2
1: if symbol = a then goto 1 else goto 2
2: if symbol = a then goto 2 else goto 2
```

Тук заключителна команда е отбелязана с указател 1. Такива програми не се използват в практиката, но целта тук е да се покаже, че разпознаването в този случай не се нуждае от повече памет и други команди, а работата на всеки КДА би могла да се реализира толкова опростено.

Функцията на преходите може да не е дефинирана за всяка двойка аргументи. Достигането на неопределеност за следващото състояние по време на работата на автомата води до неразпознаване на входната дума. Друга особеност е, че за един формален език може да съществуват повече КДА, които го разпознават. Те се наричат еквивалентни и представят различни алгоритми по разпознаването на думите.

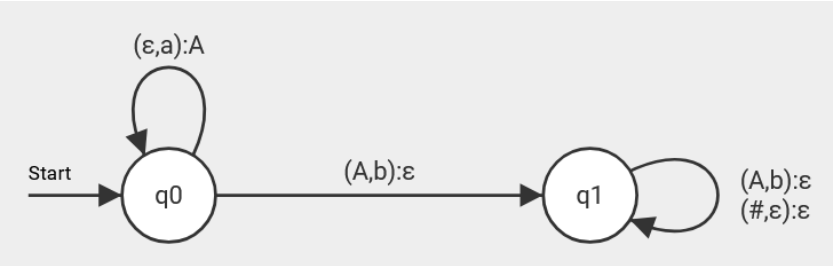
Едно от приложенията на автоматите в програмирането са регулярните изрази, които са еквивалентни на КДА, но компактният им синтаксис и трудната проверка за еквивалентност на изрази създават затруднения на програмистите. Използвайки някои алгоритми за КДА, тези практически проблеми лесно могат да бъдат решени. Важно е още да се подчертае, че макар регулярните езици да се представят обикновено като генератори на думи, а крайните автомати като разпознаватели, тези две средства за описание са еквивалентни и позволяват крайни представяния на безкрайни езици [24].

Паметта, с която оперират КДА, е съвсем минимална – във всеки един момент в нея се помни само текущото състояние. Тя обаче е недостатъчна за някои други езици като например $\{a^n b^n | n \geq 1\}$ и в този случай са необходими машини с повече памет.

Стековият детерминиран автомат (СДА) е КДА с добавена памет във формата на стек, т.е. на принципа Last In First Out. В този случай функцията на преходите има още

един аргумент – това е буквата на върха на стека, а стойността на функцията е двойката, образувана от следващо състояние и новия запис (дума) в стека, т.е. тя отразява още промяната в стека за всеки такт. Допустим е също така преход към ново състояние без четене от входната лента – така главата остава на старата си позиция и не се измества надясно. Разпознаване на входна дума има само след прочитане на цялата дума на входа и достигане до празен стек, без да се отчитат заключителни състояния. Дъното на стека в началото е отбелязано със специален символ, например #, за да може да се установи кога стекът е празен. СДА не може да работи с празен стек. СДА също се представят с диаграми, отразяващи функцията на прехода.

Пример 2. На фиг. 3 е представен чрез диаграма един СДА. При задаване на входна дума *aaabbb* автоматът прехвърля в стека буква *A* за всяка от първите букви *a*, а текущо е състоянието q_0, след което с всяка следваща буква *b* на входа от върха на стека се премахва една буква *a*. Накрая стекът е празен, а всички входни букви са прочетени и затова СДА разпознава тази входна дума. За думата *aaabb* след прочитане на входните думи в стека остава още една буква *A* и затова тази дума не се разпознава. Езикът, който даденият СДА разпознава, е $\{a^n b^n | n \geq 1\}$. Тук автоматът използва стека, за да запомни входните букви a в началото на думата и да може после да сравни техния брой с буквите *b* накрая.



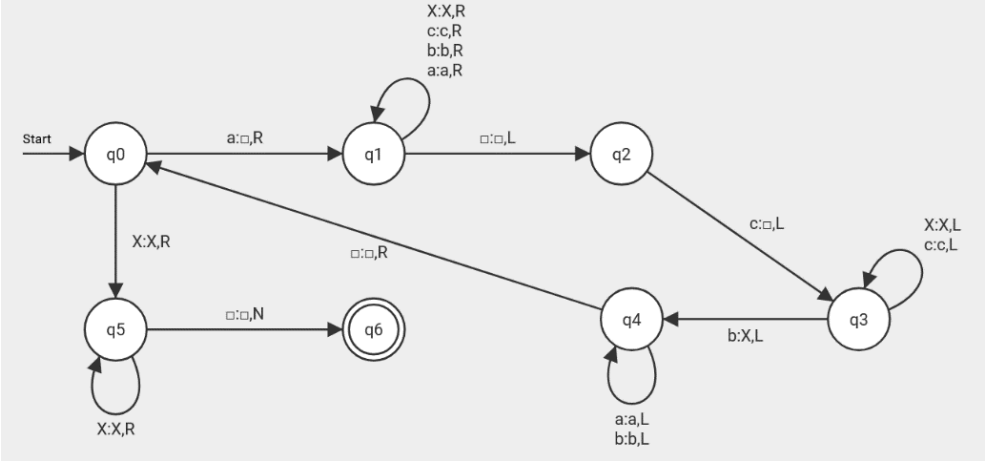
Фиг. 3. СДА от Пример 2

Изборът на структурата стек не е случаен – това е структурата от данни, с която се моделира рекурсията в информатиката. Приложението на стековите автомати е свързано със синтаксиса на езиците за програмиране и синтактичния анализ на програми по време на компилацията.

Паметта на СДА е повече от паметта на КДА, защото има добавен стек, но тя все пак е недостатъчна за някои други езици като например $\{a^n b^n c^n | n \geq 1\}$. В този случай стекът би позволил проверка за равенство на буквите a и b, но за да има проверка за буквите c е нужна още памет – например още един стек. Стеков автомат с два стека е еквивалентен на машината на Тюринг от следващия разгледан автоматен модел.

При Машината на Тюринг (МТ) главата може не само да чете входни букви, но и да пише в клетките на лентата. Лентата е безкрайна, а главата във всеки такт чете буква, пише буква и се придвижва – с една клетка надясно или наляво или остава на същата клетка. Всички празни клетки са отбелязани със знак □. Функцията на преходите определя кое е новото състояние, какво движение да се извърши и какъв знак да се запише в клетката, а нейните аргументи са текущото състояние и текущия символ в клетката. Работата се прекратява, ако се достигне до заключително състояние. Допълнително МТ превръща входната дума в изходна дума. МТ може бъде представена чрез диаграма, която задава алгоритъма на обработка, а за по-лесно разбиране се добавя словесно описание на този алгоритъм.

Пример 3. На фиг. 4 е представена чрез диаграма една МТ. При задаване на входна дума *aabbcc* машината работи по следния начин: първата буква *a* се изтрива от лентата и главата се придвижва надясно до края на думата, след това се изтрива последната буква *c* и главата се движи отново наляво до последната за думата буква *b*, която се заменя с буква *X*, след това главата се придвижва до началото на думата и описаните действия се повтарят. Така на всяка изтрита буква *a* съответства изтрита буква *c*, а една буква *b* е заменена с *X*. Когато свършат буквите *a* в началото на лентата и на нея са останали само букви *X*, то във входната дума броят на отделните букви е бил равен и те са били подредени така: първо са всички букви *a*, после са буквите *b* и накрая *c*. Тогава работата ще завърши в заключителното състояние q_6 и езикът, който МТ разпознава, в този случай е $\{a^n b^n c^n | n \geq 1\}$.

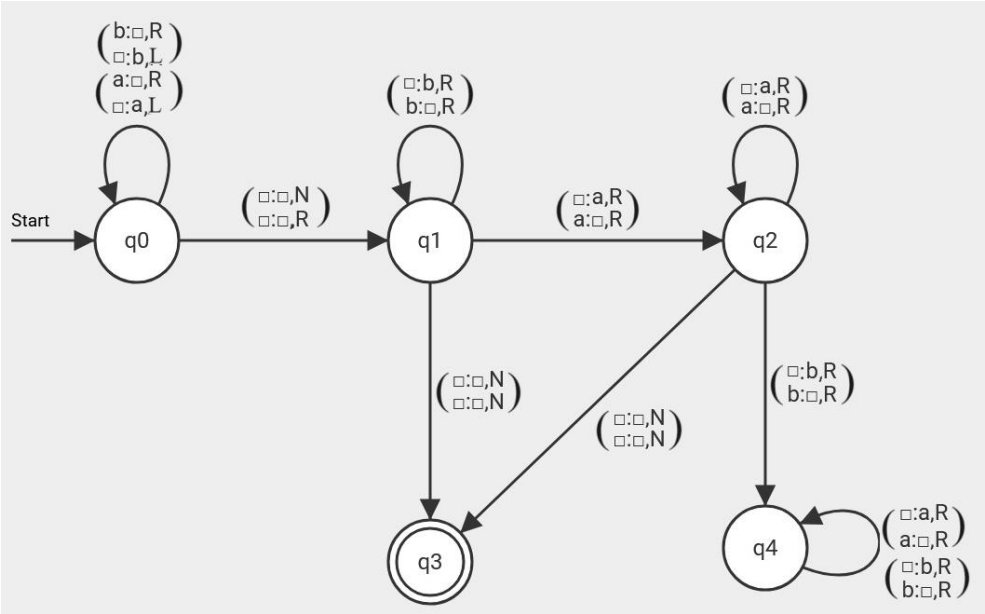


Фиг. 4. МТ от Пример 3

МТ е еквивалентна на задаване на алгоритъм, както и съответно на програма. За целите на изследванията на свойствата и взаимовръзките в теоретичен аспект обаче тя е по-подходяща, защото при нея има по-малко операции в сравнения с езиците за програмиране, запазвайки същевременно мощността. Паметта на МТ задава по-голяма мощност в сравнение със стековите автомати, но и тя не е достатъчна за разпознаване на всички формални езици.

Към паметта на МТ може да се добави допълнително памет, т.е. още ленти, всяка от които има собствена независима глава за четене и писане. Ще се спрем на МТ с две ленти, означени като първа и втора. В началото на работата на първата лента е записана входната дума, а втората лента е празна и ще играе помощна роля. Дефиницията на машината се различава от класическите МТ само във функцията на прехода, към която е добавен аргумент за буквата на втората лента, а към стойността са включени и промените върху втората лента – нова буква и движение на главата. И тези МТ се представят с диаграма, а словесното описание на алгоритъма улеснява проследяването на работата. За всеки преход се записват промените на двете ленти една под друга.

Пример 4. На фиг. 5 е представена чрез диаграма МТ с две ленти. При задаване на произволна входна дума машината работи по следния начин: чете буквите от първата лента от ляво надясно и ги записва на втората лента отдясно наляво, след това прочитане те се изтриват от първата лента. Следва четене на буквите от втората лента в посока от ляво надясно и записването им на първата лента в същата посока. Ако при това буквите *b* предхождат буквите *a* или няма букви *a*, то се достига до заключителното състояние q_3. Езикът, който МТ разпознава, е $\{a^n b^m | n \geq 0, m \geq 0\}$ и в края на работата всяка входна дума се записва в обратен ред на първата лента.



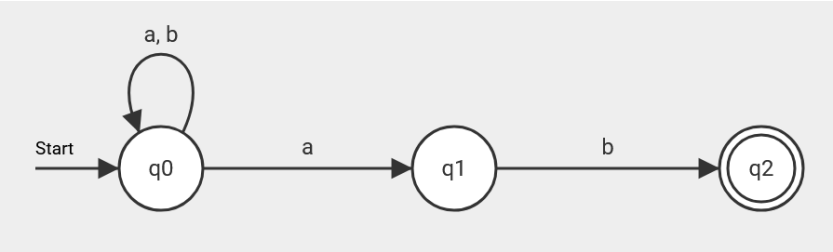
Фиг. 5. МТ от Пример 4

Доказано е, че МТ с повече ленти са еквивалентни на МТ с една лента, но повечето ленти дават по-висока ефективност на работата по отношение на времето за обработка, както и позволяват физически да се раздели работната памет от входните данни на две различни ленти. Така се намалява и броят на извършените операции и това превръща МТ с повече ленти в основен модел в теорията за сложността на алгоритмите [21].

Крайният автомат представлява опростен модел на компютър, а машината на Тюринг е точен модел на съвременните компютри и има същите възможности. В научната литература са изследвани и редица други автомати, но те по-рядко са част от основното обучение по теоретична информатика.

Изчислителните машини от разглежданите типове могат да бъдат детерминирани или недетерминирани. При детерминираните машини е еднозначно определено каква е следващата стъпка, а при недетерминираните – функцията на прехода за поне един набор от аргументи има повече стойности. Например при КДА за едно състояние и една входна буква има най-много едно следващо състояние, докато при крайните недетерминирани автомати (КНА) за една входна буква и едно състояние може да се дефинират повече следващи състояния. Аналогично и за другите машини може на дадена стъпка от работата да се допусне избор от повече възможности за преход и да се разглеждат стеков недетерминиран автомат (СНА) и недетерминирана МТ. Разпознаването на дума изисква да има поне един вариант на работа върху входната дума, който води до разпознаване. Това в значителна степен променя алгоритъма на обработка и стъпките в него, защото разпознаването не е единствената възможност за входната дума и в алгоритъма са включени и други варианти, които дори могат да не водят до разпознаване.

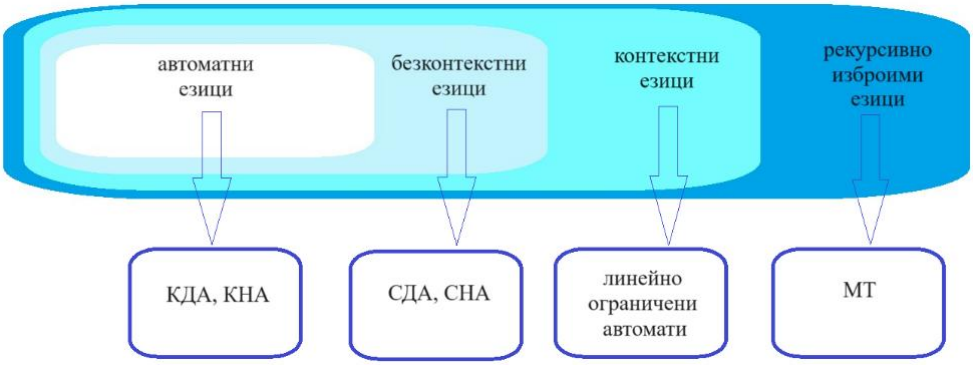
Недетерминираният подход при създаването на алгоритми и автоматни модели съответства на повече пътища в граф за една входна дума, от които поне един завършва в заключително състояние, за да се разпознае думата. Докато при детерминирания подход има едно следващо състояние и следователно обработката на думата е в единствена точно определена последователност от състояния. Пример за краен недетерминиран автомат (КНА) е даден на фиг. 6 и за него от състоянието q_0 има два възможни прехода с буква a. Ако на входа се подаде думата ab, само едно от тези две възможни решения – да остане в състояние q_0 или да премине в състояние q_1 ще доведе до разпознаване. Този автомат разпознава всички думи, които завършват на ab.



Фиг. 6. КНА

Интуитивно изглежда, че недетерминираните автомати са по-мощни по отношение на разпознаваните езици отколкото са детерминираните. Но това е вярно само за някои модели – за крайните автомати и за МТ. Стековите детерминирани автомати обаче наистина разпознават само част от езиците, разпознавани от стековите недетерминирани автомати.

Разгледахме четири изчислителни модела, реализирани с различни абстрактни машини, различаващи се по отношение на ограниченията на паметта, с която разполагат. На фиг. 7 е представена връзката между тях и съответствието на разпознаваните езици според Йерархията на Чомски. МТ с една лента и МТ с две ленти са еднакво мощни относно разпознаваните езици, а на контекстните езици съответстват МТ с ограничение на дължината на лентата, наречени линейно ограничени автомати.



Фиг. 7. Йерархия на Чомски и съответствие с изчислителните модели

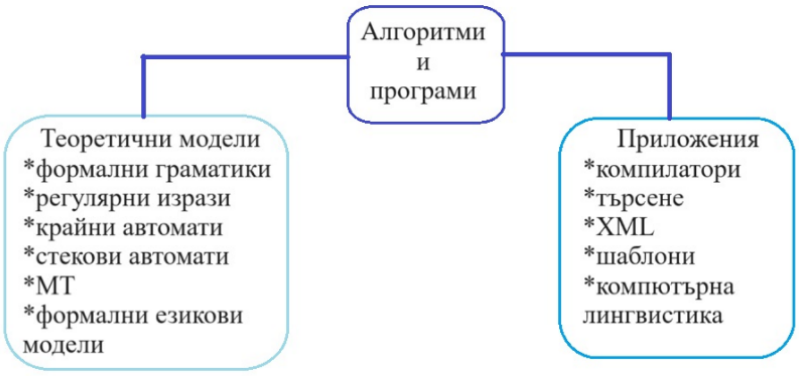
Обикновено в научната литература и в лекционните курсове по дисциплини от областта на теоретичната информатика в началото се дефинират понятията „формален език" (ФЕ) и „формална граматика" (ФГ), след това се въвеждат автоматите и някои типове абстрактни машини, а накрая се доказват еквивалентности и взаимовръзки. На схемата на фиг. 7 са илюстрирани тези взаимовръзки като разликите между тях произтичат от различната налична памет. На тази схема при изчислителните модели в посока от ляво надясно се увеличават възможностите за използване на паметта. При обучението би трябвало да се обръща повече внимание на алгоритмичната страна на проблемите и съответните ограничения в паметта и разликите и особеностите на детерминираната и недетерминираната работа на автомат.

Разпознаването на думи от изчислителни машини е свързано с изследванията за т. нар. Decision problem – при него за дадена дума и даден език се търси отговор на въпроса дали тя е от езика, т.е. дали има алгоритъм, който да отговори на този въпрос. От теоретична и практическа гледна точка други важни въпроси, свързани с това, са разрешимост, изчислимост и сложност на алгоритми.

Приложението на автоматните модели в информатика (фиг. 8) се свързва най-вече с фазите лексически и синтактичен анализ при компилаторите, където се разпознават токените във входната програма и се проверява дали тя съответства на синтаксиса на

езика. Те играят важна роля също при алгоритмите за търсене в текст, съвпадение на шаблони, регулярни изрази в програми, валидиране на XML документи, обработка на XPath и XQuery изрази, системи за обработка на изображения, обработка на реч, системи за обработка на сигнали, криптосистеми със симетричен ключ.

Формалните езикови модели се свързват още с изчислимост и сложност на алгоритми, изкуствен интелект, биоинформатика, компютърна лингвистика. Крайните автомати са най-ефективният начин за конструиране на езиков речник, като това се дължи най-вече на появата на алгоритми за бързото им изграждане и ефективно съхранение и компресиране. В този случай те имат редица предимства пред структури като списъци, масиви и дървета. Крайните автомати участват в обработката на естествен език (Natural Language Processing) и по-точно при морфологичния анализ и синтез, корекция на правописа и извличане на информация.



Фиг. 8. Приложения на теоретичните модели в програмирането

Дисциплини от областта на теоретичната информатика са включени в учебните планове на специалност Информатика в почти всички университети. В някои държави като Германия и Япония се наблюдава и тенденция за изучаване на тези теоретични постановки в профилираните курсове в последните години от гимназиалното обучение.

Поради големия брой на студентите и учениците, изучаващи дисциплини от областта на теоретичната информатика, са разработени онлайн среди [25] и [26] за обучение. Те предоставят инструменти за създаване, тестване, моделиране и генериране на ниво компилатор за различни формални езици, както и симулации, трансформации и конвертиране, а конструираните модели могат да се използват в други приложения. От друга страна са създадени и програми [27] и онлайн редактори [28] за изчертаване на автомати и абстрактни машини, а това е улеснено и с различни специализирани пакети за автомати към софтуера за LaTeX.

4. Система от задачи

Важна част от професионалната подготовка на студентите е практическата подготовка, която трябва да се опира на система от научни знания и да е насочена към формиране на практически и изследователски умения и компетентности. По време на обучението след запознаване с основните дефиниции и твърдения следва решаването на системи от задачи, чрез които се изграждат аналитични умения, култура на мисленето и умения за използването на терминологията.

Системите от задачи могат да включват задачи от различните типове, като част от тях са предназначени за решаване по време на семинарните упражнения, а друга част могат да бъдат задачи за самостоятелна работа или задачи за самоконтрол и подготовка

за изпит. Биха могли да се добавят и по-сложни проектни задачи, чрез които да се направи връзка с практически проблеми от областта на информатиката.

В представената на Табл. 1, Табл. 2, Табл. 3 и Табл. 4 система от задачи при даден формален език трябва да се построи разпознавател от посочения тип. Решението на първата основна задача от всяка група беше представено в примерите по-горе. Останалите основни задачи във всяка група на системата се решават чрез модифициране и адаптиране на алгоритъма по разпознаване, заложен в решението на основната задача. Към системата са включени и допълнителни задачи за самостоятелна работа, чието решение също е свързано с основните задачи.

Таблица 1. Задачи за крайни автомати

Модел	**Основни задачи**		**Допълнителни задачи**
КДА/КНА Памет: текущо състояние на автомата	1.0	$\{a^n \mid n \geq 1\}$	$\{a^n \mid n \geq 2\}$
	1.1	$\{a^n b^k \mid n \geq 1, k \geq 1\}$	$\{a^n \mid n \geq 0\}$
	1.2	$\{a^n b^k c^m \mid n \geq 1, k \geq 1, m \geq 1\}$	$\{a^n b^k a^m \mid n \geq 1, k \in \{1,2\}, m \geq 0\}$
	1.3	$\{(ab)^n \mid n \geq 1\}$	$\{(ab)^n b^m \mid n \geq 1, m \geq 1\}$
	1.4	$\{(abc)^n \mid n \geq 1\}$	$\{a^n b^m \mid n$ и m са четни$\}$
	1.5	$\{a^n \mid n \geq 1, n$ е четно$\}$	$\{a^n b^k \mid n \equiv 1(mod\ 3), k \geq 1\}$
	1.6	$\{a^n \mid n \equiv 0(mod\ 3)\}$	$\{w \in \{a,b\}^* \mid \#a$ и $\#b$ са четни$\}$
	1.7	$\{ab^n a \mid n \geq 1\}$	$\{w \in \{a,b\}^* \mid \|w\|$ е нечетно$\}$
	1.8	$\{a(bb)^n a \mid n \geq 1\}$	
	1.9	$\{w \in \{a,b\}^* \mid \#a$ е четно$\}$	
	1.10	$\{w \in \{a,b\}^* \mid \|w\|$ е четно$\}$	
	1.11	$\{w \in \{a,b\}^* \mid w$ съдържа $ab\}$	
	1.12	$\{w \in \{a,b\}^* \mid w$ не съдържа $ab\}$	

Таблица 2. Задачи за стекови автомати

Модел	**Основни задачи**		**Допълнителни задачи**
СДА/СНА Памет 1: текущо състояние на автомата Памет 2: стек	2.0	$\{a^n b^n \mid n \geq 1\}$	$\{a^n b^n \mid n \geq 2\}$
	2.1	$\{a^n b^{n+1} \mid n \geq 1\}$	$\{a^n b^m \mid n < m, n \geq 0\}$
	2.2	$\{a^n b^m \mid n > m, m \geq 1\}$	$\{a^n b^{n+2} \mid n \geq 1\}$
	2.3	$\{a^n b^m \mid n \neq m, n \geq 1, m \geq 1\}$	$\{a^{n+1} b^n \mid n \geq 1\}$
	2.4	$\{a^n b^n c^m \mid n \geq 1, m \geq 1\}$	$\{w \in \{a,b\}^* \mid \#a < \#b\}$
	2.5	$\{a^n b^{2n} \mid n \geq 1\}$	$\{w \in \{a,b\}^* \mid \#a = \#b$ или $\#b = \#c\}$
	2.6	$\{w \in \{a,b\}^* \mid \#a = \#b\}$	$\{a^n b^m c^m \mid n \geq 1, m \geq 1\}$
	2.7	$\{wa^n \in \{a,b\}^* \mid \|w\| = n\}$	$\{a^n b^m a^m b^n \mid n \geq 1, m \geq 1\}$
	2.8	$\{w\$w^R \mid w \in \{a,b\}^*\}$	$\{a^n (bc)^{n-1} \mid n \geq 1\}$
	2.9	$\{ww^R \mid w \in \{a,b\}^*\}$	$\{a^n b^{2m} c^m \mid n \geq 1, m \geq 1\}$
	2.10	$\{a^n b^m c^{n+m} \mid n \geq 1, m \geq 1\}$	$\{a^n bc^n \mid n \geq 1, m \geq 1\}$
	2.11	$\{a^n b^m c^k \mid n + k \geq\ m, n, m, k \geq 1\}$	$\{ww^R \mid w \in \{a,b\}^*, \|w\| = 2n, n \geq 1\}$
	2.12	$\{a^n (bc)^{n+1} \mid n \geq 1\}$	$\{w \in \{a,b\}^* \mid w = w^R\}$

Таблица 3. Задачи за МТ с една лента

Модел	**Основни задачи**		**Допълнителни задачи**
МТ с една лента Памет 1: текущо състояние на автомата Памет 2: лента и глава	3.0	$\{a^n b^n c^n \mid n \geq 1\}$	$\{a^n b^n c^k \mid n \geq 1, n \geq k\}$
	3.1	$\{a^n b^{n+1} c^n \mid n \geq 1\}$	$\{a^n b^{n+1} c^{n+1} \mid n \geq 1\}$
	3.2	$\{a^n b^{2n} c^n \mid n \geq 1\}$	$\{a^n b^k c^m \mid n \geq 1, n \geq k, n \geq m\}$
	3.3	$\{a^n b^k c^n \mid k \geq 1, n \geq k\}$	$\{(ab)^n b^k (ab)^m \mid n \geq k \geq m\}$
	3.4	$\{a^n b^k c^m \mid n \geq k \geq m\}$	$\{a^n b^{n-1} c^{n-2} \mid n \geq 3\}$
	3.5	$\{w \in \{a,b,c\}^* \mid \#a = \#b = \#c\}$	$\{a^n b^{n+1} c^{2n} \mid n \geq 1\}$
	3.6	$\{a^n b^k c^k d^m \mid n \geq k\}$	$\{a^n b^{2n} c^{3n} \mid n \geq 1\}$
	3.7	$\{a^n b^k c^k d^m \mid n \geq k \geq m\}$	
	3.8	$\{(ab)^n c^n (ab)^n \mid n \geq 1\}$	

Таблица 4. Задачи за МТ с две ленти

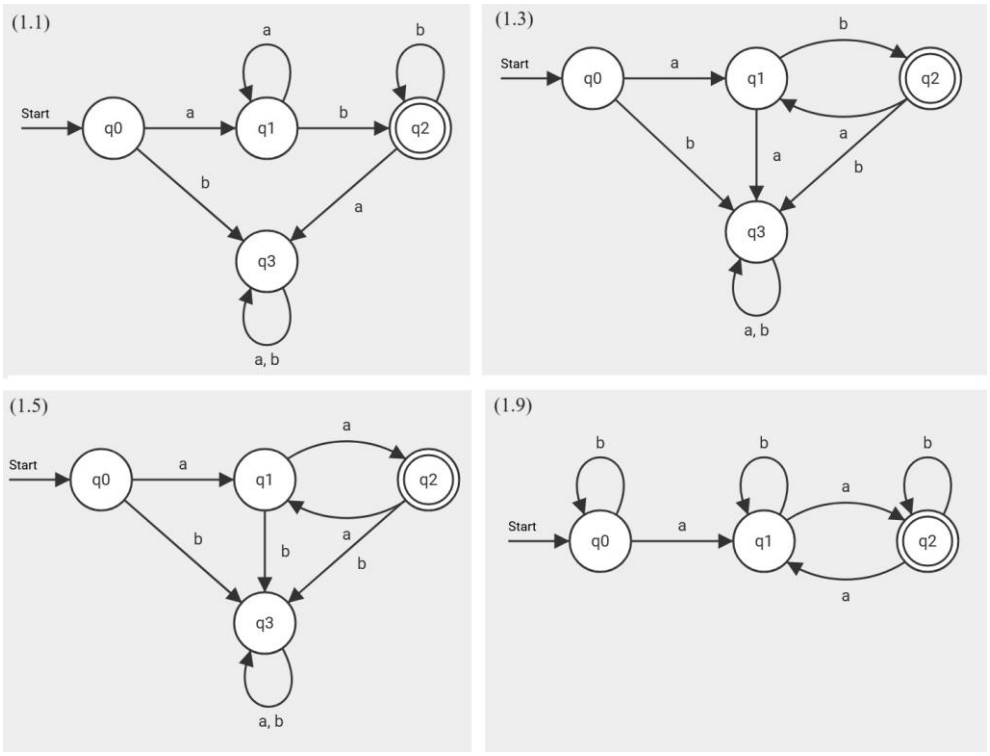
Модел	**Основни задачи**		**Допълнителни задачи**
МТ с две ленти Памет 1: текущо състояние на автомата Памет 2: първа лента Памет 3: втора лента	4.0	$\{a^n b^m \mid n \geq 0, m \geq 0\}$ и обръщане на входната дума	$\{a^n b^m \mid n \neq m\}$
	4.1	$\{a^n b^n \mid n \geq 0\}$	$\{a^n b^m \mid n < m\}$
	4.2	$\{a^n b^n c^n \mid n \geq 0\}$	$\{a^n b^k c^m \mid n < k < m\}$
	4.3	$\{a^n b^k \mid n \geq k\}$	$\{a^n b^k c^m \mid n > k, n > m\}$
	4.4	$\{a^n b^k c^m \mid n > k > m\}$	$\{a^n b^{2n} \mid n > 0\}$
	4.5	$\{a^n b^k c^k d^m \mid n \geq k > m\}$	$\{(ab)^n c^m \mid n < m\}$
	4.6	$\{(ab)^n c^n \mid n \geq 1\}$	$\{(ab)^n c^m \mid n \neq m\}$

На фиг. 9 са дадени КДА за задачи 1.1, 1.3, 1.5 и 1.9, а намирането на техните решения се основава на следните разсъждения:

- Задача 1.1 – В дадения език след произволния брой букви *a* следват букви *b* и това се постига чрез добавяне на ново заключително състояние и преход към него с буква *b*.
- Задача 1.3 – Повтарящата се буква *a* от Задача 1.0 тук е заменена с дума *ab* и затова заключителното състояние q_2 ще се достига само след прочитане на такава входна поддума. Преходът от q_2 към q_1 с входна буква *a* позволява прочитане на следващи поддуми *ab*.
- Задача 1.5 – Към условията за езика от задача 1.0 е добавено изискването броят на буквите *a* да е четно число, затова в автомата се добавя ново заключително състояние q_2 и достигането до него е възможно единствено чрез прочитане на поредна дума *aa*.
- Задача 1.9 – Тук броят на буквите *a* отново е четен, но се допуска да има и букви *b* на произволно място в думите. Изхождайки от автомата от Задача 1.5, за всяко състояние се добавят преходи към същото състояние с входна буква *b*.

КДА не са универсални разпознаватели, те могат да разпознават силно ограничено множество от формални езици. Причината за това е ограничението в паметта, която се използва от автомата по време на работата му – той помни само кое е текущото състояние. Например разпознаването на езика $\{a^n b^n \mid n \geq 1\}$ изисква от алгоритмична гледна точка запомняне на броя на буквите a от началото на думата и сравняването му с броя на

буквите b накрая. На КДА не му достига памет за разпознаването на този език. Необходимо е да се разшири паметта и това се постига със стековите автомати, които освен памет за текущото състояние имат и втора памет във формата на стек. Там биха могли да се запишат всички прочетени от началото на думата букви a и след това да се сравнят по брой с входните букви b. Това е направено в решението на Задача 2.0 в Пример 2.



Фиг. 9. КДА за задачи 1.1, 1.3, 1.5 и 1.9

Допълнителните задачи за стекови автомати в настоящата система представят алгоритми, чието конструиране се опира на основната идея:

- Задача 2.1 – Промяната в алгоритъма е в добавянето на последна допълнителна проверка, дали след изчерпване на буквите a от стека има още една буква b на входа.
- Задача 2.3 – Алгоритъмът трябва да осигурява разпознаване при всеки от следните случаи: дума само от букви a, дума само от букви b, при сравняване на входните букви b с буквите a от стека се достига до изчерпване на стека и наличие на още букви b на входа, а последната ситуация е изчерпване на буквите b на входа при наличие на още букви a в стека.
- Задача 2.8 – Всички букви до знака $ се прехвърлят в стека, а след него започва сравнение на поредната входна буква с буквата от върха на стека.
- Задача 2.9 – Тази задача не може да бъде решена със СДА, защото не е отбелязана средата на входната дума (както е в Задача 2.8). Алгоритъмът на СНА трябва да предлага на всеки такт избор дали да продължи прехвърлянето на букви в стека или това да бъде моментът, в който се спира прехвърлянето на букви в стека и се продължава със сравнение.
- Задача 2.10 – Първоначално в стека се записват всички входни букви a, следват и буквите b, а след това се сравняват по брой с буквите с на входа.

Както беше отбелязано по-горе, езикът $\{a^n b^n c^n | n \geq 1\}$ не може да се разпознае от СДА или СНА, т.е. не съществува алгоритъм по този изчислителен модел. След

проверката за равенството на степените на буквите *a* и *b* стекът е празен и няма как да се установи дали следващите букви *c* са същия брой. Разпознаването на този език е включено в системата като Задача 3.0 и се осъществява с МТ, която разполага с повече памет за извършването и на тази проверка. Решението на тази основна задача е представено в Пример 1 и на фиг. 4. Алгоритмите на допълнителните задачи от Табл. 3 модифицират това решение по следния начин:

- Задача 3.1 – Когато по време на работата се изключат всички букви *a*, на лентата преди буквите *X* трябва да стои още една буква *b*.
- Задача 3.3 – Когато първата и последната буква във входната дума, това са съответно *a* и *c*, се заменят със символа □, главата се придвижва наляво до последната буква *b* и тя се заменя с *X*. В един момент е възможно е да няма букви *b* и тогава това действие не се извършва.
- Задача 3.8 – От началото на лентата се изтрива първата буква *a* и втората буква *b*. Главата се мести в края на думата и оттам се изтрива последната буква *b* и предпоследната буква *a*. Следва замяна на последното *c* с буква *X*.

Задачите от системата, съответстващи на модела МТ с две ленти, се основават на алгоритъма на Задача 4.0, представен в Пример 4. За допълнителната задача 4.1 алгоритъмът се модифицира по следния начин: при четене на буквите a от първата лента, те се прехвърлят върху втората лента. Когато се четат букви *b* от първата лента, те се изтриват от нея едновременно с буквите *a* от втората лента. Накрая двете ленти трябва да останат празни, за да има разпознаване.

Определянето на типа на даден език и на съответния изчислителен модел може да бъде труден въпрос. Йерархията на Чомски, представена на Фиг. 7, изисква съществуване на формална граматика от съответния тип за пораждане на думите в езика. Намирането на пораждаща граматика не означава, че това е непременно граматиката от най-високо ниво в йерархията. Тук може обаче да се мисли на ниво алгоритъм и минимална необходима памет за реализацията му, т. е. от кой тип е абстрактната машина и така да се определи типът на езика.

От друга страна в задачите, при които за даден формален език се търси абстрактна машина, която го разпознава, може да не се посочва нейният тип. Тогава е възможно да има решения с различни типове машини и от гледна точка на оптимизацията на алгоритъма по разпознаване, определен от машината, оптимална е тази машина, която изисква по-малко памет. В такъв случай за даден език се търси първо алгоритъм чрез автомат с памет само за текущото състояние, т.е. КДА, а ако не може да се конструира такъв, се минава към търсене на алгоритъм с повече памет, т.е. стеков автомат или МТ.

На фиг. 10 са обобщени взаимовръзките между практическите проблеми, изчислителните модели и очакваните резултати, на които се акцентира при обучението и използването на разработената система задачи.

Фиг. 10. Схема на взаимовръзките между проблеми и модели

5. Заключение

Обучението в компютърните специалности традиционно е съпътствано с много трудности и висок процент на отпадналите още след първата година. Проблемът е глобален. Затова непрекъснато се анализират причините и се търсят нови подходи и методи на обучение. В частност това важи и за обучението по програмиране, което винаги е било предизвикателство. Нашите наблюдения показват, че като резултат от обучението по уводните дисциплини „Основи на информатиката", „Увод в програмирането", „Алгоритми и структури от данни", „Обектно ориентирано програмиране", „Дискретни структури", които под различни наименования присъстват във всички учебни планове, студентите успяват да изградят в задоволителна степен умения за абстрактно, алгоритмично и изчислително мислене. Затрудняват се обаче при използването на точни и еднозначни формулировки за понятията, механизмите и езиковите конструкции, които са част от програмния код. Следствие на това са и трудностите при изграждането на модели и адаптирането на общите алгоритми към конкретни задачи. Студентите не могат убедително да доказват, че създаденият от тях модел съответства на поставената задача. Не могат да аргументират достатъчно използваните и разработените от тях самите алгоритми за решаване на конкретен проблем. Тоест математическото мислене не е достатъчно развито. Паралелно с уводните компютърни курсове студентите учат и немалко „чиста" математика, но явно не може да се получи силна междупредметна връзка. Такава връзка може съвсем естествено да се изгради в учебните курсове по „Автомати и теория на езиците", „Методи за транслация" и „Езикови процесори". Точно това цели предложената тук система от задачи за автомати и абстрактни машини – обръща повече внимание на алгоритмичната страна на проблемите и съответните ограничения в паметта и разликите и особеностите на детерминираната и недетерминираната работа на автоматите. Надгражда алгоритмичните подходи при различните автоматни модели и спомага за по-ясно разбиране на използваните методи чрез плавно повишаване на сложността, акцентиране на взаимовръзките и прилаганите логически принципи и постигане на трайни знания по компютърно моделиране. Тук залагаме и на мотивиращото действие на смия факт, че програмно се емулира работата на абстрактни математически машини. Някои студенти с изпреварващи знания по съвременните езици и библиотеки опитват да решават задачите с езиковите средства от високо ниво, като в резултат с един кратък и елегантен израз се получава отговорът. Това, разбира се, е полезно, доколкото могат да се сравняват различни решения, но не е целта на разглеждания тук метод. Разработвайки емулацията на работата на автомата с минимално множество от процедурни езикови конструкции, студентите получават трайни знания и по-пълно разбиране на математическите основи на компилаторите. Очевидна е и междупредметната връзка с дисциплината „Алгоритми и структури от данни". За някои от задачите трябва да се направи реализация на абстрактната структурата стек – много важна за информатиката структура, с която може да се моделира рекурсия. Не се насърчава използването на библиотечна версия на стека, а се залага на самостоятелната реализация на операциите с него.

Предложеният метод се прилага от учебната 2023/2024 във Факултет „Математика и информатика" на Великотърновски университет. Системата е включена в обучението на студенти от спец. „Софтуерно инженерство" по дисциплината „Методи за транслация", както и за студенти от спец. „Компютърни науки" по дисциплината „Автомати и теория на езиците". Тъй като броят на студентите, чиито резултати от работа по системата от задачи проследяваме не е голям, за да предложим статистически значими изводи, предстои да се съберат данни и от обучението през 2024/2025. На този етап все пак биха могли да бъдат отчетени интерес у обучаемите към използването на

обучителната среда FLACI [25], с която по-бързо могат да създават и да тестват своите модели, и повишаване на мотивацията им като следствие от подчертаните тесни връзки на задачите с практиката.

REFERENCES // ЛИТЕРАТУРА

1. Stacey, K. (2006). "What is mathematical thinking and why is it important?", University of Melbourne, Australia
2. Mailund, T. (2021). "Introduction to Computational Thinking", Apress, 2021. DOI: https://doi.org/10.1007/978-1-4842-7077-6
3. Wolf, K. (2017). "Präzises Denken für Informatiker", Springer Vieweg, 2017, DOI: https://doi.org/10.1007/978-3-662-54973-5
4. O'Regan, G. (2013). "Mathematics in Computing", Springer, 2013. DOI: https://doi.org/10.1007/978-1-4471-4534-9
5. Korolev, A. (2019). "Computer modeling", 2019.
6. Barua, R., Gupta. K. (2013). "Finite Automata", Handbook of Finite State Based Models and Applications, 2013.
7. Deiser, O.; Lasser, C.; Vogt, E.; Werner, D. (2016). "12 × 12 Schlüsselkonzepte zur Mathematik", Springer Spektrum, 2016. DOI: https://doi.org/10.1007/978-3-662-47077-0
8. Martín-Vide, C. (2011). "Scientific Applications of Language Methods", Imperial College Press, 2011.
9. Sedakova, V. (2018). "Methodology for solving mathematical problems", Surgut, 2018.
10. Ferragina, P.; Luccio F. (2018). "Computational Thinking: First Algorithms, Then Code", Springer, 2018. DOI: https://doi.org/10.1007/978-3-319-97940-3
11. Granovskiy, B. (2018). "Science, Technology, Engineering, and Mathematics (STEM) Education: An Overview", Congressional Research Service R45223, VERSION 4, June 12, 2018
12. Tytler, R., Swanson, D., Appelbaum, P. (2015). Subject matters of science, technology, engineering, and mathematics. In, M. F. He, B. D. Schultz, & W. H. Schubert (Eds), The Sage Guide to Curriculum in Education (Ch. 4, pp. 27-35), Sage Publications, 2015
13. Lee, HY., Lin, CJ., Wang, WS. et al. (2023). „Precision education via timely intervention in K-12 computer programming course to enhance programming skill and affective-domain learning objectives", IJ STEM Ed 10, 52 (2023). https://doi.org/10.1186/s40594-023-00444-5
14. Bybee, R.W., Taylor, J., Gardner, A., Scotter, P., Carlson, J., Westbrook, A., Landes, N. (2006). The BSCS 5E instructional model: Origins and effectiveness. Office of Science Education National Institutes of Health. 1-80
15. Zhao, D., Muntean, Cr., Chis, A., Rozinaj, G., Muntean, G. (2022). "Game-Based Learning: Enhancing Student Experience, Knowledge Gain, and Usability in Higher Education Programming Courses", IEEE Transactions on Education, Vol. 65, No. 4, November 2022, pp. 502-513
16. Bungum, B., Mogstad, E. (2024). "Building and programming a weather station: Teachers' views on values and challenges in a comprehensive STEM project", Research in Science & Technological Education, Volume 42, 2024 – Issue 2, pp. 450–466. https://doi.org/10.1080/02635143.2022.2103108
17. Hajiyeva, R. (2024). "Using elements of C++ programming language in teaching informatics with mathematics", XVI International Scientific-Practical Conference "Actual Problems of Improving Farming Productivity and Agroecology" (IPFA 2024), E3S Web Conf. Volume 538, 2024. https://doi.org/10.1051/e3sconf/202453802033
18. Corral-Barraza, L., Gomez-Elizalde, J., Moran-Soto, G., Godinez-Garcia, F., Pena, Omar I. Gonzalez (2024), "Transforming Electronics Engineering Classrooms: Fostering Students' Motivation to Design Technological Solutions Through Inquiry-Based Learning", IEEE Access, vol. 12, pp. 88585-88595, 2024, doi: https://doi.org/10.1109/ACCESS.2024.3413011
19. Colin, J., Hoarau, S., Declercq, Chr., Broisin, J. (2024). "Design and Evaluation of a Web-based Distributed Pair Programming Tool for Novice Programmers", In Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024). Association for Computing Machinery, New York, NY, USA, 527–533. https://doi.org/10.1145/3649217.3653571

20. Ouyang, F., Jiao, P., McLaren, B., Alavi, A. (2022). "Artificial Intelligence in STEM Education", (Chapman & Hall/CRC Artificial Intelligence and Robotics Series), CRC Press; 1st edition (December 29, 2022)
21. Hromkovič, J. (2014). "Theoretische Informatik", Springer Vieweg, 2014. DOI https://doi.org/10.1007/978-3-658-06433-4
22. Linz, P. (2012). "An Introduction to Formal Languages and Automata", Jones & Bartlett Learning, 2012.
23. Wagenknecht, C.; Hielscher, M. (2014). "Formale Sprachen, abstrakte Automaten und Compiler", Springer Vieweg, 2014
24. Esparza, J., Blondin, M. (2023). "Automata Theory – An Algorithmic Approach", The MIT Press, 2023.
25. FLACI – Formal Languages and Compilers and Interpreters, Available at: https://flaci.com/home/# (last view: 05-08-2024)
26. JFLAP, Available at: https://www.jflap.org/ (last view: 11-08-2024)
27. AtoCC – from Automation to Compiler Construction, Available at: https://atocc.de/cgi-bin/atocc/site.cgi?lang=de&site=main (last view: 05-08-2024)
28. Finite State Machine Designer, Available at: https://madebyevan.com/fsm/ (last view: 05-08-2024)

Received: 15-08-2024 Accepted: 25-09-2024 Published: 20-12-2024

Cite as:
Dochkova-Todorova, J., Donchev, I. (2024). "STEM Approach to Build Competences from Automata and Languages Theory", Science Series "Innovative STEM Education", volume 06, ISSN: 2683-1333, pp. 35-54, 2024. DOI: https://doi.org/10.55630/STEM.2024.0604