УНИВЕРЗИТЕТ У ПРИШТИНИ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
КОСОВСКА МИТРОВИЦА
ПРИМЉЕНО 17.05.2023
ОРГ. ЈЕДИН. ВРЕДНОСТ
613/2

УНИВЕРЗИТЕТ У ПРИШТИНИ
СА ПРИВРЕМЕНИМ СЕДИШТЕМ У
КОСОВСКОЈ МИТРОВИЦИ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

Срећко Н. Стаменковић

СОФТВЕРСКИ СИСТЕМ ЗА УЧЕЊЕ АЛГОРИТАМА КОЈИ СЕ ПРИМЕЊУЈУ У РАЗЛИЧИТИМ ФАЗАМА ПРОГРАМСКИХ ПРЕВОДИЛАЦА

Докторска дисертација

Косовска Митровица, 2023.

УНИВЕРЗИТЕТ У ПРИШТИНИ
СА ПРИВРЕМЕНИМ СЕДИШТЕМ У
КОСОВСКОЈ МИТРОВИЦИ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

Срећко Н. Стаменковић

СОФТВЕРСКИ СИСТЕМ ЗА УЧЕЊЕ АЛГОРИТАМА КОЈИ СЕ ПРИМЕЊУЈУ У РАЗЛИЧИТИМ ФАЗАМА ПРОГРАМСКИХ ПРЕВОДИЛАЦА

Докторска дисертација

Косовска Митровица, 2023.

UNIVERSITY OF PRISTINA
TEMPORARY SETTLED IN
KOSOVSKA MITROVICA

FACULTY OF TECHNICAL SCIENCES

Srećko N. Stamenković

SOFTWARE SYSTEM FOR LEARNING ALGORITHMS APPLIED IN DIFFERENT PHASES OF A COMPILER

Doctoral Dissertation

Kosovska Mitrovica, 2023.

Ментор:

др Ненад Јовановић, редовни професор

Универзитет у Приштини са привременим седиштем у Косовској Митровици
Факултет техничких наука

Чланови комисије:

_____

_____
_____

_____

_____
_____

_____

_____
_____

Датум одбране:

_____

Наслов докторске дисертације: *Софтверски систем за учење алгоритама који се примењују у различитим фазама програмских преводилаца*

Сажетак: У овој докторској дисертацији описан је нови веб базирани софтверски систем назван ComVis (енгл. *Compiler Visualization System*) који представља скуп алата намењених за учење и подучавање тема из области програмских преводилаца. И поред убрзаног развоја рачунарских наука и отварања нових поља проучавања, конструкција компајлера је и даље веома важно научно поље које се предаје на већини светских универзитета. Због високог нивоа апстракције процеса програмског превођења, предавачима је врло тешко да традиционалним начином подучавања објасне студентима теме у оквиру универзитетског предмета Програмски преводиоци. Праћењем стеченог знања студената на овом предмету, примећено је да имају потешкоћа са савлађивањем градива које обилује комплексним алгоритмима и теоријским конструкцијама, што се на крају манифестује лошим резултатима на испиту. Анализом различитих педагошких метода, дошло се до претпоставке да би увођење система за интерактивну визуелну репрезентацију и симулацију процеса компајлирања у оквиру лабораторијских вежби на предмету довело до дубљег разумевања функционисања програмских преводилаца од стране студената.

Методологија истраживања у оквиру докторске дисертације представља комбинацију теоријских, практичних и експерименталних метода које су реализоване кроз неколико фаза. Извршен је преглед фундаменталних техника и алгоритама који се примењују у конструкцији програмских преводилаца са акцентом на теоријске концепте из различитих фаза програмског превођења који се могу илустровати помоћу симулационог система. На основу систематичног проучавања литературе, направљен је генерални преглед области образовних софтверских система за помоћ у учењу уз осврт на неопходност интерактивне компоненте у образовном софтверу, али и на моћ визуелизације која може допринети ефективнијем преносу знања. Извршено је испитивање постојећих симулационих система погодних за учење тема из области програмских преводилаца, као и креирање

модела за оцену квалитета одабраних система на основу успостављених критеријума који су развијени као резултат опсежне анализе релевантних метода евалуације образовних софтвера. Проценом је утврђено да је сваки од анализираних система развијен да испуни специфичне захтеве (нпр. обрађује специфични алгоритам, једну тему или фазу компајлирања). Такође, у погледу интерактивности и визуелне репрезентације, анализирани алати нису задовољили постављене критеријуме. Сви ови аргументи указали су на реалну потребу за развојем новог система у којем би се усавршили постојећи и осмислили нови модули који обрађују теме које нису покривене тренутним системима, уз употребу савремених развојних технологија за унапређење функционалних карактеристика. Резултати претходно наведених анализа искоришћени су за формирање функционалних и педагошких захтева новог софтверског система, на основу којих је дефинисан генерализовани модел за развој интерактивног софтверског система за симулацију и визуелизацију алгоритама који се примењују у различитим фазама програмског превођења.

На основу дефинисаног модела извршена је имплементација новог софтверског система који је због веће доступности и боље визуелне репрезентације развијен као веб базирана апликација. Софтвер је модуларно пројектован тако да омогућава једноставну надоградњу додавањем нових алгоритама. У лексичком модулу система студенти могу да се упознају са основним појмовима коначних аутомата, врстама и различитим могућностима њихове конструкције, а обезбеђена је и могућност симулације рада аутомата за задати улаз, симулација Hopcroft-овог алгоритма за минимизацију детерминистичких коначних аутомата, као и визуелна репрезентација конверзије недетерминистичког у детерминистички коначни аутомат применом алгоритма за конструкцију подскупова. Такође, у овом модулу студенти могу да науче све о регуларним изразима и њиховој практичној примени визуелизацијом и симулацијом различитих конверзија (коначни аутомат у регуларни израз применом Brzozowski и McCluskey алгоритма и супротна конверзија применом Thompson-овог алгоритма). Симулациони систем садржи и модул за учење синтаксне анализе. Овај модул

обухвата већи број алгоритама који се примењују у процесу парсирања, као што су алгоритам рада нерекурзивног предиктивног парсера, LL (1) и LR (0) парсера. Студенти сами дефинишу регуларне језике уносом скупа правила и на тај начин стичу искуство у раду са граматикама. Систем помаже студентима да боље разумеју процес синтаксне анализа тако што обезбеђује могућност праћења симулација парсирања за различите задате улазе. Значајан допринос новог софтверског система је и решење за симулирање семантичке анализе, с обзиром да претрагом литературе није пронађен ниједан симулатор који у потпуности покрива ову тему. Применом различитих симулационих алата у семантичком модулу система, студенти уче о атрибутивним граматикама и семантичким правилима, а симулациони систем ComVis, за разлику од свих анализираних алата, садржи и део за процену знања студената. Имплементацијом овакве врсте модула испуњени су сви постављени педагошки циљеви.

На крају овог истраживања приказани су примери примене овог софтверског система у образовном процесу, као и резултати спроведене евалуације. Процена употребљивости и ефикасности система потврдила је претпоставку да његова примена у настави има велики значај и за предаваче и за студенте. Примена ComVis-а као помоћног наставног средства у оквиру лабораторијских вежби на предмету Програмски преводиоци директно је утицала на повећање мотивације и анагажовања студената, што је коначно довело до бољег разумевања функционисања компајлера.

Кључне речи: програмски преводиоци, коначни аутомати, *Thompson*-ов алгоритам, *Hopcroft*-ов алгоритам, *Brzozowski and McCluskey* алгоритам, алгоритам конструкције подскупова, алгоритми парсирања, системи за помоћ у учењу, образовна технологија, софтверско инжењерство, визуелне симулације.

Научна област: Електротехника и рачунарство

Ужа научна област: Рачунарска техника и информатика

Dissertation title: Software system for learning algorithms applied in different phases of a compiler

Abstract: In this doctoral dissertation, a new web-based software system called ComVis (Compiler Visualization System) is described, which represents a set of tools intended for learning and teaching topics in the compilers field. Despite the accelerated development of computer science, and creation of new fields of study, compiler construction is still a very important field that is taught at most world universities. Due to the high level of abstraction of the program translation process, it is very difficult for lecturers to explain to students the topics within the university course "Compilers" using the traditional ways of teaching. By monitoring the acquired knowledge of the students in this course, it was noticed that the students have problems mastering the material that is full of complex algorithms and theoretical constructions, which ultimately manifests itself in poor results on the exam. Various analysis of pedagogical methods led to the assumption that the introduction of a system for interactive visual representation and simulation of the compiling process, within course laboratory exercises, would lead to a deeper understanding of the compiler's functioning by students.

The research methodology within the doctoral dissertation represents a combination of theoretical, practical and experimental methods, which were realized through several stages. An overview of the fundamental techniques and algorithms applied in the construction of compilers was performed, with an emphasis on theoretical concepts from different phases of compilers, which can be illustrated using a simulation system. Based on a literature systematic study, a general overview of educational software systems which can help in learning was made, with reference to the necessity of an interactive components in educational software, but also to the power of visualization that can contribute to more effective knowledge transfer. An examination of existing simulation systems suitable for learning topics in the field of compilers was carried out. A model was created for evaluating the selected systems quality based on established criteria, which were developed as a result of extensive analysis of relevant educational software evaluation methods. The assessment concluded that, for the most part, each of the

analyzed systems was developed to meet specific requirements (eg, it handles a specific algorithm, a single topic, or a compilation phase). Also, in terms of interactivity and visual representation, the analyzed tools did not meet the set criteria. All these arguments pointed to a real need for the development of a new system in which the existing ones would be perfected and new modules would be developed that deal with topics not covered by the current systems, with the use of modern development technologies to improve functional characteristics. The results of the aforementioned analysis were used to form the functional and pedagogical requirements of the new software system, based on which a generalized model was defined for the interactive software system development for the algorithms simulation and visualization which are applied in different phases of compilers.

Based on the defined model, a new software system was implemented. Due to greater availability and better visual representation, the system was developed as a web-based application. The software is designed in a modular way, so it allows for a simple upgrade later by adding new algorithms. In the system's lexical module, students can get acquainted with the basic terms of finite automata, their types and different possibilities of their construction. In this module, the possibility of simulating the operation of the automaton for a given input is provided, the simulation of Hopcroft's algorithm for the minimization of deterministic finite automata, as well as a visual representation of the conversion of a non-deterministic to a deterministic finite automaton using the construction of subsets algorithm. Also, in this module, students can learn everything about regular expressions and their practical application by visualizing and simulating different conversions (finite automaton to regular expression using the Brzozowski and McCluskey algorithm, and the opposite conversion using the Thompson algorithm). The simulation system also contains a module for learning syntax analysis. This module includes a number of algorithms that are applied in the parsing process, such as the non-recursive predictive parser, LL(1) and LR(0) parser algorithms. Students themselves define regular languages by entering a set of rules, thus gaining experience in working with grammars. By following parsing simulations for various given inputs, the system

helps students gain a deeper understanding of the parsing process. A significant contribution of the new software system is the solution for simulating semantic analysis, given that a literature search did not found a single simulator that fully covers this topic. In the system's semantic module, students learn about attributive grammars and semantic rules using various simulation tools. Also, the ComVis simulation system, unlike all analyzed tools, also contains a part for assessing students' knowledge. With the implementation of this module type, all set of pedagogical goals have been met.

At the end of this research, application examples of this software system in the educational process are shown, and the results of the conducted evaluation are given. The evaluation of the system usability and efficiency confirmed the assumption that its application in teaching is of great importance for both lecturers and students. The application of ComVis as an auxiliary teaching tool within the laboratory exercises in the compilers course directly influenced the increase in student motivation and engagement, which ultimately led to a deeper understanding of the compiler functioning by the students.

Keywords: compilers, finite automata, Thompson's algorithm, Hopcroft's algorithm, Brzozowski and McCluskey algorithm, subset construction algorithm, parsing algorithms, e-learning tools, educational technology, software engineering, visual simulations.

Scientific area: Electrical Engineering and Computing

Scientific subarea: Computer Science and Informatics

Садржај

Списак слика

Списак табела

Списак програмског кода

1. УВОД

У ери информационо-комуникационих технологија рачунар је добио веома значајну улогу у свакодневном животу, на посредан или непосредан начин (многи кућни уређаји постали су „паметни“, путни рачунари су саставни делови нових аутомобила, мобилни телефонски апарати постали су прави џепни рачунари…). Директну употребу рачунара човек остварује кроз интеракцију са корисничким апликацијама. Због све већих захтева корисника апликације постају све комплексније. Самим тим потребно је унапређивати и хардвер како би рачунар могао да одговори захтевнијим апликацијама. Међутим, колико год рачунар био „моћан“ односно сложен он ће и даље обрађивати инструкције у машинском, то јест бинарном облику (Stanković, Stojković and Tošić, 2018). Због тога рачунарске апликације имају потребу за виртуелним системом који ће дату апликацију превести у форму која се може извршити на рачунару. Тај виртуелни алат, који представља мост између софтвера и хардвера, заправо је софтверски систем који преводи програме са изворног вишег програмског језика на одговарајући језик нижег нивоа који је разумљив циљној хардверској архитектури. Софтверски систем задужен за превођење програма са једног програмског језика на одговарајући други програмски језик назива се програмски преводилац[1] (енгл. *compiler*).

Програмски преводиоци почели су да се развијају педесетих година прошлог века и били су ограничени на превођење програмских језика вишег нивоа у машински код и на оптимизацију меморијског простора и времена извршавања програма (Hall, Padua and Pingali, 2009). Данас су се развили у

[1] У домаћој литератури се за програмске преводиоце користе и називи компајлери и компилатори.

озбиљне софтверске системе у оквиру којих функционише велики број различитих сложених техника. Програмски преводиоци су састављени од више подсистема са много унутрашњих компоненти и алгоритама и сложених интеракција између њих. Дизајн програмског преводиоца утиче на квалитет излазног кода, а самим тим и на перформансе апликације. Алгоритми и технике програмских преводилаца, омогућавају и откривање и спречавање разних дефеката у коду, као и злонамерног кода. Због свега наведеног, конструисање програмских преводилаца данас представља веома важну инжењерску дисциплину.

Програмски преводиоци заузимају значајно место у курикулуму студијских програма рачунарских наука на бројним светским универзитетима. Не постоји предмет који исцрпно обрађује све теме везане за дизајн и имплементацију програмских преводилаца јер добро дизајниран програмски преводилац садржи микрокосмос рачунарских наука (Cooper and Torczon, 2012). Компајлери представљају област рачунарских наука, у којој се математички модели и теоријски концепти, као што су теорија аутомата, теорија графова, формални језици и граматике, успешно примењују у практичним имплементацијама лексичких анализатора, парсера, семантичких анализатора, оптимизатора кода итд. Теме које се обрађују на курсу компајлера на различитим универзитетима се могу прилично разликовати. Већина постојећих курсева компајлера на светским универзитетима углавном дели наставни план и програм на модуле (теме) који одговарају фазама компајлирања. Основне фазе програмског превођења су: лексичка анализа, синтаксна анализа, семантичка анализа, генерисање међукода, оптимизација и генерисање циљног кода (Jovanović, 2023). Студенти би требало да савладају основне концепте овог предмета како би боље разумели и предмете уско повезане са овом тематиком јер се поље које се бави програмским преводиоцима све више преплиће са другим дисциплинама (архитектура и организација рачунара, програмски језици, теорија формалних језика и аутомата, теорија рачунарства, софтверски инжењеринг и рачунарска сигурност).

1.1 Предмет и циљ истраживања

Традиционалним подучавањем програмских преводилаца студенти кроз апстрактне теоријске материјале добијају само површно знање, док су вежбе углавном посвећене алгоритмима, структурама података и теоријским конструкцијама (Griswold, 2002). Такав приступ заснован на предавањима не припрема добро студенте инжењерства за „професионални свет“ (Kundra and Sureka, 2016). Праћењем активности студената у току семестра на предмету Програмски преводиоци на Факултету техничких наука Универзитета у Приштини, установљено је да имају проблема са разумевањем градива јер се први пут сусрећу са комплексним и апстрактним теоријским концептима програмских преводилаца што доводи до изостанка мотивације и ангажованости на овом предмету, а све то коначно резултира лошим успехом на испиту. Међутим, и предавачи су суочени са изазовом како да своје знање пренесу студентима на адекватан дидактички начин, из чега произилази закључак да процес изучавања програмских преводилаца није једноставан задатак ни за предаваче ни за студенте. Да би се студенти боље припремили за овај изазов, неопходан је нови приступ за подучавање и вежбање процеса програмског превођења који треба да буде комплементаран постојећим наставним плановима. У овим чињеницама лежи и мотивација за ово истраживање.

Детаљном анализом могућих решења за унапређење наставе програмских преводилаца, дошло се до претпоставке да би увођење система за интерактивну визуелну репрезентацију и симулацију процеса програмског превођења у оквиру лабораторијских вежби на предмету могло да помогне студентима да боље разумеју градиво. У циљу унапређења традиционалне наставе у многим областима рачунарских наука успешно се примењују алати за визуелизацију, симулатори, анимациони системи и други едукативни софтверски алати за илустрацију и визуализацију апстрактних концепата. Образовни системи се разликују у зависности од области у којој се користе. Уобичајено је да се на курсевима везаним за хардвер користе софтверски

симулатори (Jovanović, Popović and Jovanović, 2009) како би се избегла висока цена неопходне опреме за лабораторијске вежбе. Системи за анимацију и симулацију рада алгоритама предложени су у многим студијама за учење сложених алгоритама (Lawrence, Badre and Stasko, 1994). Симулатори и системи за визуелну репрезентацију алгоритама омогућавају предавачима да троше мање времена на предавања, остављајући им више времена за индивидуални рад са студентима (Woolf and Hall, 1995). Иако су одређена истраживања спроведена са намером да покажу да анимације у учењу могу донети лажни осећај задовољства негативно утичући на развој стратегија учења, визуелизације и симулације су ипак изузетно корисне за разумевање динамичких активности и могу дати више самопоуздања и мотивације студентима (Ainsworth, 2008).

У дисертацији је извршена детаљна анализа већег броја образовних система који су погодни за учење неких од тема из области програмских преводилаца. Направљена је анализа постојећих решења како би се утврдило које теме сваки од система у наставном предмету покрива и какве су му карактеристике, а на основу добијених резултата и процена које карактеристике би систем требало да има да би се успешно користио у настави. Тако је успостављен модел за оцену симулационих система са аспекта покривености тема, као и основних карактеристика и функционалности. Од анализираних система ниједан не покрива довољан број тема, углавном се ради о едукативним алатима који визуелизују неки специфични алгоритам или симулирају једну фазу компајлирања. Ови системи нису задовољили постављене критеријуме у погледу карактеристика и функционалности које нуде у процесу симулације. Због тога је донета одлука да се развије нови софтверски систем ComVis (Compiler Visualization System) којим би се превазишли уочени недостаци постојећих система.

У дисертацији је дефинисан ефикасан модел за развој интерактивног софтверског система за симулацију и визуелизацију алгоритама који се примењују у различитим фазама програмског превођења. Један од захтева

дефинисаног модела је и да се софтверски систем модуларно пројектује тако да омогући једноставну надоградњу додавањем нових алгоритама. Сваки модул покрива једну фазу програмског превођења и састоји се од више подмодула који обрађују одговарајућу технику или алгоритам који се примењују у датој фази компајлирања. С обзиром да се ради о образовном софтверу, симулациони систем мора да испуњава све педагошке норме и буде прилагођен како студентима тако и предавачима. Општи циљ овог рада је унапређење наставног процеса предмета Програмски преводиоци применом имплементираног едукативног алата. Експерименталним путем треба потврдити да предложена метода за учење програмских преводилаца побољшава успешност процеса учења и олакшава стицање знања студентима.

1.2 Полазне хипотезе

Сходно постављеном предмету и циљевима истраживања, основна хипотеза ове дисертације је:

Могуће је пронаћи ефикасно и ефективно решење за помоћ студентима и предавачима у учењу, односно подучавању тема из предмета Програмски преводиоци на Факултету техничких наука Универзитета у Приштини.

Из ове генерализоване хипотезе произилазе следеће:

- Поштујући све педагошке норме, могуће је развити модуларни софтверски систем који покрива већи број тема од тренутно доступних алата;
- Реализовани софтверски систем наставницима олакшава организацију лабораторијских вежби у смислу визуелизације теоријских основа, дефинисања задатака за лабораторијске вежбе и процене стеченог знања;
- Примена софтверског система у оквиру лабораторијских вежби у оквиру предмета Програмски преводиоци повећава мотивацију и ангажовање студената;

- Применом реализованог софтверског система студенти могу постићи боље разумевање теоријских концепата и алгоритама из области програмских преводилаца.

1.3 Методе истраживања

Методологија истраживања у оквиру докторске дисертације представља комбинацију теоријских, практичних и експерименталних метода, које су реализоване кроз неколико фаза:

- Преглед фундаменталних техника и алгоритама који се примењују у конструкцији програмских преводилаца са акцентом на теоријске концепте из различитих фаза програмског превођења који се могу илустровати помоћу симулационих система;
- Систематично проучавање литературе из области образовних софтверских симулационих система;
- Испитивање карактеристика одабраних симулационих система који су погодни за учење тема из области програмских преводилаца;
- Евалуација квалитета одабраних софтверских алата на основу скупа дефинисаних критеријума за оцену познатих карактеристика датих система;
- Дефинисање функционалних захтева новог софтверског система базираним на развијеним техничким, педагошким и циљевима употребљивости;
- Имплементација и тестирање софтверског система;
- Дефинисање адекватног начина примене имплементираног система у образовном процесу;
- Евалуација реализованог симулационог система у односу на новоуспостављени модел оцене квалитета симулационог система, као и са аспекта ефикасности приликом примене у наставном процесу.

1.4 Допринос

Очекивани резултат докторске дисертације је реализација потпуно новог веб базираног софтверског система заснованог на интерактивном графичком корисничком интерфејсу за приказ симулација и визуелизацију комплексних теоријских конструкција и алгоритама из области програмског превођења. Применом овог система у наставном процесу студенти би могли много лакше и брже да савладају предвиђено градиво. У оквиру докторске дисертације очекују се следећи научни доприноси:

- Генерални преглед области образовних симулационих система за помоћ у учењу уз осврт на неопходност интерактивне компоненте у образовном софтверу, али и на моћ визуелизације која може допринети ефективнијем преносу знања;
- Креиран модел за оцену квалитета постојећих система на основу развијених критеријума;
- Процена квалитета постојећих система погодних за учење тема програмских преводилаца на основу успостављеног модела за оцену квалитета;
- Генерализација функционалности и покривености тема постојећих система за учење програмских преводилаца;
- Дефинисање ефикасног модела за развој интерактивног софтверског система за симулацију и визуелизацију алгоритама који се примењују у различитим фазама програмског превођења.

Очекивани стручни доприноси докторске дисертације су:

- Имплементација софтверског система на основу дефинисаног модела;
- Формирање лабораторијских вежби за интерактивну демонстрацију помоћу симулација у оквиру имплементираног софтверског система;
- Унапређење наставног процеса применом реализованог едукативног алата.

1.5 Структура рада

Докторска дисертација је систематизована у осам поглавља. У другом поглављу представљене су теоријске основе програмских преводилаца како би се дефинисали основни појмови који се користе у раду. Укратко су описане фундаменталне технике и алгоритми који се користе за конструкцију програмских преводилаца. Треће поглавље представља образовне софтверске системе за симулацију и визуелну репрезентацију апстрактних теоријских конструкција. Овде је акценат стављен на пожељне карактеристике образовних софтвера (попут интерактивности и визуелизације) које могу допринети ефективнијем преносу знања. У четвртом поглављу представљен је већи број симулационих система погодних за учење тема из предмета Програмски преводиоци. Ово поглавље се бави дефинисањем модела за квалитативну оцену постојећих система. Резултати евалуације детаљно су приказани и дискутовани на крају тог поглавља. На основу добијених резултата евалуације и темељном претрагом литературе, у петом поглављу дефинисани су педагошки, функционални и циљеви употребљивости за развој новог образовног система. Опис ComVis система и детаљи имплементације презентовани су у шестом поглављу. Овде су описани алгоритми и теоријски концепти који се могу учити овим системом. Такође, у овом поглављу описан је и начин употребе ComVis система у оквиру лабораторијских вежби. Седмо поглавље се бави евалуацијом реализованог симулационог система са аспекта ефикасности и употребљивости приликом примене у настави. Анализа је извршена на основу спроведеног контролисаног експеримента и проценом корисничког искуства квантитативном анкетом студената. У осмом поглављу дат је закључак о главним доприносима дисертације, као и план за будућа истраживања.

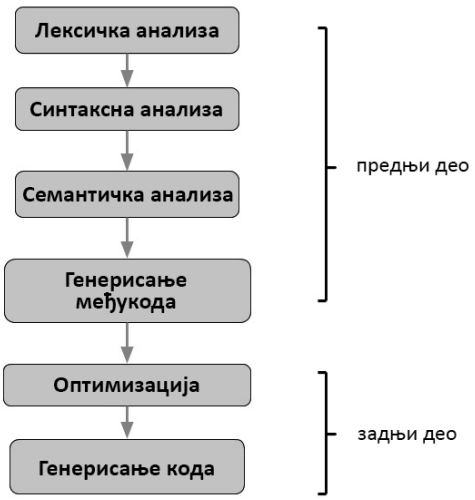
2. ТЕОРИЈСКЕ ОСНОВЕ ПРОГРАМСКИХ ПРЕВОДИЛАЦА

Чињеница да програмски преводилац има два основна задатака, да прихвати изворни програм као улаз и преслика његову функционалност на циљну хардверску архитектуру, упућује на то да се програмски преводиоци састоје од два основна дела, дела за анализу и дела за синтезу, односно предњег (енгл. *front end*) и задњег дела (енгл. *back end*) како се у неким литературама може наћи (Wirth, 2005). Оваква организација је добра због преносивости компајлера. Технологија компајлера, посебно дела за анализу, неопходна је за обраду свих формалних или природних језичких улаза било ког компјутерског система. Предњи део обрађује програмске језике високог нивоа и генерише међукод. Задњи део оптимизује међукод и задужен је за генерисање инструкција, односно кода на ниском нивоу који зависи од архитектуре циљне машине. Ове две етапе превођења састоје се од више фаза које су дефинисане у софтверским подсистемима који се извршавају секвенцијално. Реализовани софтверски систем ComVis покрива фазе и технике програмских преводилаца које су присутне у предњем делу. Предњи део компајлера одговоран је за лексичку, синтаксну и семантичку анализу. У овом поглављу укратко су описане фазе програмског превођења које су покривене софтверским системом ComVis. Објашњени су основни теоријски појмови који се користе у дисертацији.

2.1 СТРУКТУРА ПРОГРАМСКИХ ПРЕВОДИЛАЦА

Превођење изворног програма почиње у лексичком анализатору, такозваном скенеру. Задатак овог подсистема је да прочита улазни низ знакова који чине изворни програм и групише их у смислене секвенце које се називају лексеме. На основу добијених лексема генеришу се одговарајући токени, то јест симболи који идентификују елементе програмског језика (идентификаторе, операторе, кључне речи…). Добијени низ токена се даље

упућује у синтаксни анализатор где се проверава синтакса изворног програма, то јест да ли су идентификоване речи (лексичке јединице) у програму написане у складу са правилима која дефинишу дати програмски језик. Уколико не постоје грешке у синтакси, покреће се процес парсирања (рашчлањивање програма на елементарна правила). Парсирањем се формира структура у виду стабла која приказује рашчлањене наредбе и редослед којим се требају применити. Синтаксно стабло представља основу за генерисање међукода, машински независне репрезентације изворног програма. У делу за синтезу поред међукода прослеђује се и табела симбола у којој се чувају додатне информације о изворном коду (имена променљивих, тип, опсег…). У овом делу врши се оптимизација међукода, генерисање кода циљне машине и евентуална оптимизација кода. Код може бити генерисан директно на машински језик или на асемблерски језик циљне машине. Ово је најопштији опис процеса програмског превођења приказан блок шемом на слици 1.



Слика 1. Основне фазе програмског превођења

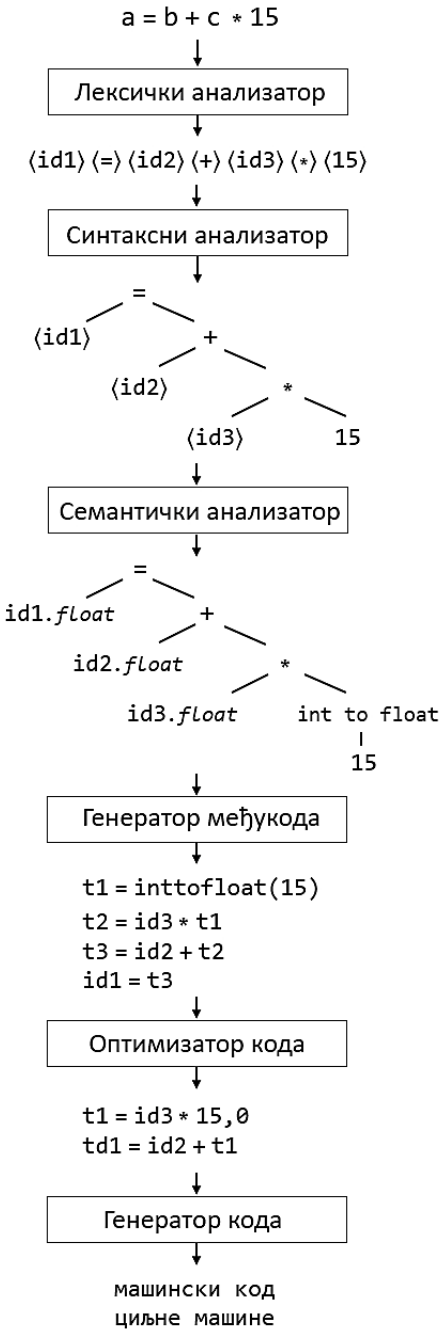
2.2 ПРОЦЕС ПРЕВОЂЕЊА ИЗВОРНОГ КОДА

Најбољи начин за детаљнију презентацију рада програмског преводиоца је демонстрација једноставног примера процеса превођења проласком кроз све фазе, од изворног вишег програмског кода до излазног машинског кода.

Графички приказ свих акција које се примењују приликом пресликавања улазног низа у циљни код је дат на слици 2. Као улазни низ коришћен је код који представља једноставну наредбу додељивања:

```
a = b + c * 15
```

Програмски код је написан на хипотетичком језику, и одговара форми већине виших програмских језика.



Слика 2. Пример процеса превођења задате наредбе додељивања

Изворни код се најпре доводи на улаз лексичког анализатора чији је основни задатак да у овој почетној фази превођења из низа карактера јасно идентификује све лексичке целине. За наведени пример скенер препознаје следеће лексеме: три идентификатора (*a*, *b* и *c*), аритметичке операторе (+, *, =) и једну константу (број 15). Препознате лексеме затим мапира у одговарајуће токене, па се на његовом излазу добија следећи резултат:

⟨id1⟩ ⟨=⟩ ⟨id2⟩ ⟨+⟩ ⟨id3⟩ ⟨*⟩ ⟨15⟩

Лексички анализатор генерише и специјалну табелу у којој се чувају додатне информације о препознатим елементима програмског кода. У овом примеру сваки идентификатор је добио свој редни број који представља везу са сачуваним подацима у табели. За идентификаторе је важно сачувати њихова имена и типове. Технички је могуће и за константу генерисати токен облика ⟨const1⟩, али ће због прегледности у овом примеру бити коришћена директно њена вредност.

Следећа фаза превођења је синтаксна анализа чији је задатак генерисање конструкције налик стаблу која осликава граматичку структуру низа токена. На оваквом стаблу сваки унутрашњи чвор представља операцију, а потомци чвора представљају аргументе операције. Ово стабло показује редослед операција у задатој наредби, што се може видети на слици 2. Распоред операција је у складу са уобичајеним аритметичким конвенцијама на основу којих множење има већи приоритет од сабирања.

Семантички анализатор користи стабло које је конструисао синтаксни анализатор и информације из табеле генерисане од стране лексичког анализатора да провери семантичку конзистентност изворног програма са дефиницијом програмског језика. Важан део семантичке анализе је провера типа где програмски преводилац проверава да ли сваки оператор има одговарајуће операнде. Спецификација програмског језика може дозволити конверзије неких типова, на пример бинарни аритметички оператор се може применити или на пар целих бројева или на пар децималних бројева. Ако се

оператор примени на децимални и цео број, компајлер може извршити конверзију целог броја у децимални. Ово може бити случај у датом примеру, ако су промењљиве *a*, *b* и *c* декларисане као децимални бројеви то ће имплицирати конверзију целобројне константе у децималну. На излазу из семантичког анализатора добија се структура стабла са додатим атрибутима. На слици 2 се може приметити да излаз семантичког анализатора има додатни чвор за оператор *inttofloat* који експлицитно конвертује целобројни аргумент у децимални.

Након процеса анализе изворног кода следи генерисање такозваног међукода који представља репрезентрацију кода на ниском нивоу и треба да омогући једноставно превођење на машински код циљне машине. На слици 2 се на излазу из генератора међукода могу уочити нове промењљиве (t1, t2, t3). У питању су промењљиве које генерише програмски преводилац за смештање међурезултата.

Фаза оптимизације је машински независна, али има за циљ да унапреди међукод тако да се из њега добије што бољи код циљне машине. Обично бољи код значи бржи, али може бити и краћи код или циљни код који троши мање ресурса. Пошто се промењљива t3 користи само једном да пренесе своју вредност у id1, алгоритам оптимизатора може да трансформише међукод у краћи низ, као што је приказано на слици 2 на излазу из оптимизатора.

Генератор кода узима као улаз оптимизовани међукод и пресликава га у циљни језик који може бити асемблерски или директно машински језик. Генерисањем циљног кода води се рачуна о рационалном додељивању регистара за привремено смештање промењљивих.

2.3 ЈЕЗИЦИ И ГРАМАТИКЕ

Језик је скуп речи (стрингова), реч је коначни низ симбола (Andrew and Jens, 2002). Природни језици развијали су се без претходно успостављене граматике. Граматике су настале како би помогле у проучавању и

нормализацији језика. Ниједна граматика не може у потпуности да дефинише феномене језика. Природни језици имају више особина које их чине тешким за машинску обраду, као што су велики степен комплексности, честе нејасноће и двосмислености у изражавању. Сваки програмски језик потребно је приликом пројектовања формално дефинисати због његове стандардизације и правилне примене. Формални систем састоји се од формалног језика и скупа правила извођења. Формални језик је скуп речи, тј. коначних низова који се састоје од симбола *азбуке* над којима је језик дефинисан и који се формирају у складу са граматичким правилима. Нека је X непразан скуп који називамо *азбуком*, а чије елементе називамо *словима* (*симболима*). *Реч* (*стринг*) над *азбуком X* дефинише се као коначан низ:

$$x_1x_2 \ldots x_n,$$

где су $x_1,x_2,\ldots x_n \in X$ симболи азбуке (Ignjatović and Ćirić, 2016). Скуп свих речи над *азбуком X*, укључујући и празну реч ε, означава се са X^*, док се са X^+ означава скуп свих непразних речи над том *азбуком*. И у случају када *азбука X* садржи један симбол $X = \{c\}$, скуп X^* је бесконачан, $X^* = \{\varepsilon, c, cc, ccc, cccc, \ldots\}$.

Граматика сваког програмског језика обухвата скуп терминалних симбола, скуп нетерминалних симбола, скуп правила и стартни симбол (Jovanović, 2023). Терминални симболи или терминали представљају скуп симбола, као што су кључне речи и специјални знаци, и они улазе у састав дефинисаног језика. Нетерминални симболи или нетерминали представљају скуп симбола који не улазе у састав дефинисаног језика, већ се користе као помоћни симболи за дефинисање правила. Обично се прво дефинише стартни симбол, а онда се тежи да се применом правила граматике од тог симбола генеришу речи језика. Таква правила називају се и продукције. Према лингвисти Ноаму Чомском (Chomsky, 1956) формална граматика је н-торка:

$$G = (N, \Sigma, P, S);$$

Где је:

- N коначан скуп нетерминалних симбола,
- Σ коначан скуп терминалних симбола,

- *P* коначан скуп смена облика $\alpha \rightarrow \beta$, где важи:

$$\alpha \in (N \cup \Sigma)^{*}N(N \cup \Sigma)^{*}, \beta \in (N \cup \Sigma)^{*},$$

при чему скупови N и Σ немају заједничких елемената $N \cap \Sigma = \emptyset$; низ α мора да садржи бар један нетерминални симбол,

- *S* почетни нетерминални симбол, $S \in N$.

Формални језик *L* дефинисан граматиком $G = (N, \Sigma, P, S)$:

$$L(G) = \{w \mid w \in \Sigma^{*}, S \overset{*}{\rightarrow} w\}$$

Према томе, реч *w* припада језику *L(G)* који дефинише граматика *G* ако за реч *w* важи:

- Реч *w* садржи искључиво терминалне симболе граматике,
- Реч *w* је могуће генерисати из почетног нетерминалног симбола S.

Правила којим се могу дефинисати изрази неког програмског језика могу бити задата на следећи начин:

```
izraz → id
izraz → const
izraz → izraz + izraz
izraz → izraz * izraz
izraz → (izraz)
```

Да би се дефинисале и наредбе датог програмског језика, потребно је горњи скуп правила проширити следећим:

```
naredba → id := izraz;
```

Avram Noam Chomsky је извршио категоризацију формалних граматика у односу на врсте елемената леве и десне секвенце продукционих правила, то јест правила *P* формалне граматике *G*:

- Тип 0 – Опште граматике (енгл. *Unrestricted grammar*),
- Тип 1 – Контексно-зависне граматике (енгл. *Context-sensitive*),
- Тип 2 – Контексно-независне (бесконтексне) граматике (енгл. *Context-free*),
- Тип 3 – Регуларне граматике (енгл. *Regular*).

За представљање граматика програмских језика најчешће се користи BNF облик (McCracken and Reilly, 2003), односно *Backus-Naur*-ова форма. Такође, постоји и проширени BNF облик (Extended Bacus-Naur Form) који уводи додатне елементе за изражавање граматика (Hajduković and Suvajdžin, 2004).

2.4 ЛЕКСИЧКА АНАЛИЗА

Лексичка анализа програмског језика подразумева анализу изворног кода, карактер по карактер, идентификујући целине које се називају лексеме. Слика 3 приказује примере типичних токена, њихове неформалне описе и одговарајуће лексеме.

ТОКЕН	ОПИС	ЛЕКСЕМА
id1	Идентификатор - реч састављена од слова и бројева, која не припада скупу кључних речи	A1; rezultat; prosek;
const1	Било која бројна вредност	5; 15,00; 0,3
if	Кључна реч	if
else	Кључна реч	else
assig	Додељивање	=

Слика 3. Примери токена

За сваку лексему, лексички анализатор као излаз производи токен облика:

⟨*token-name, attribute-value*⟩

који се преноси на следећу фазу, синтаксну анализу (Aho et al, 2007). Токен је, дакле, низ симбола (на пример идентификатор, константа, кључна реч и слично). Прва компонента токена *token-name* је апстрактни симбол који се користи током синтаксне анализе, а друга компонента *attribute-value* указује на унос у табелу симбола за овај токен. Табела симбола (енгл. *symbol table*) представља структуру података која се формира у току процеса анализе, и користи се за чување основних информација о идентификатору (Ajzenhamer and Bukurov, 2020). Информације из табеле симбола потребне су за семантичку анализу и генерисање кода. Процес лексичке анализе илустрован је блок шемом на слици 4.

Слика 4. Процес лексичке анализе

Модул за извештавање о грешкама позива се у случају када лексички анализатор наиђе на непознате улазне симболе или када се поткраде грешка у основним граматичким симболима (Waite and Goos, 1996). Анализатор води евиденцију о броју симбола за прелазак у нови ред (енгл. *newline*) што се користи за референцирање грешке. Поред свега тога, лексички анализатор има задатак да из улазног кода избацује делове који нису значајни за синтаксну анализу (коментаре, празнине, табулаторе и *newline* симболе).

2.4.1 Регуларни изрази

Спецификација, односно формални опис улазних низова за које се генерише одређени токен, врши се помоћу регуларних израза (алгебарска нотација за описивање скупова стрингова). Ова нотација укључује комбинацију низова симбола неке азбуке Σ, заграде и операторе |, ◦ и *. Оператор | се користи за означавање уније (операција алтернације), а оператор ◦ за надовезивање (операција конкатенације). У литератури се уместо оператора | може наћи +, а оператор ◦ се може изоставити. Оператор * означава да се уместо претходног симбола азбуке у изразу може користити празан низ или више понављања датог симбола. Овај оператор је у литератури познат као *Kleene star* или *Kleene closure*. У регуларним изразима оператор * има највиши приоритет, оператор надовезивања је следећи по приоритету, док је оператер | најнижег приоритета.

Нека је Σ азбука. Регуларни изрази над Σ дефинишу се рекурзивно на следећи начин (Su and Yan, 2011):

1) $\emptyset$ (празан скуп) је регуларни израз;
2) ε (празан низ) је регуларни израз;
3) x је регуларни израз ако је $x \in \Sigma$;
4) (r1 | r2) је регуларни израз ако су r1 и r2 регуларни изрази;
5) (r1 $\circ$ r2) је регуларни израз ако су r1 и r2 регуларни изрази;
6) r * је регуларни израз ако је и r регуларни израз.

На пример регуларни израз a|a*b описује речи језика { *a, b, ab, aab, aaab, ...* }.

Да би лексички анализатор препознао све лексеме, односно елементе које дати програмски језик садржи, оне морају бити прецизно описане одговарајућим регуларним изразима. Код већине програмских језика скуп валидних идентификатора се састоји од комбинације слова и цифара (неки језици дозвољавају и одређене специјалне карактере). Регуларни израз којим се дефинишу такви идентификатори је следећи:

```
slovo(slovo|cifra)*
```

Оваква дефиниција указује на то да идентификатор обавезно почиње словом. Заграда се користи за дефинисање подизраза и с обзиром на примењени оператор *, тај подизраз се може поновити нула или више пута. Подизраз у загради дозвољава комбинацију слова и бројева. Међутим, лексичком анализатору је потребна и дефиниција слова и цифара, па би комплетна дефиниција идентификатора била следећа:

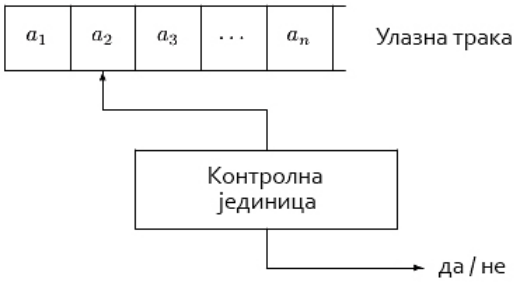
```
slovo → A|B|...|Z|a|b|...|z
cifra → 0|1|...|9
id → slovo(slovo|cifra)*
```

Конкретно, програмски језик C за имена променљивих дозвољава и употребу доње црте „ _“. У том случају довољно је да скуп *slovo* проширимо за овај знак на следећи начин:

```
slovo → A|B|...|Z|a|b|...|z|_
```

2.4.2 Коначни аутомати

Уређаји који се користе за утврђивање да ли нека реч припада језику који је описан задатом граматиком називају се аутоматима. Аутомат је апстрактни функционални рачунарски уређај који самостално извршава унапред дефинисану секвенцу операција (Linz, 2012). У општем случају, може се рећи да је аутомат апстрактни модел дигиталног рачунара и као такав има све основне карактеристике рачунара - може да чита улаз, обрађује улазне податке и даје излаз користећи контролну јединицу која врши обраду и привремену меморију.



Слика 5. Модел коначног аутомата

Коначни аутомат (енгл. *Finite Automaton*) прима свој улаз као стринг који му се испоручује на улазној траци, али не даје излаз осим индикације да ли се улаз сматра прихваћеним или не (Lewis and Papadimitriou, 1997) што га чини уређајем за препознавање језика, конкретно регуларних језика. Коначни аутомат је карактеристичан по томе што нема привремено складиште (слика 5). Ови аутомати се деле на детерминистичке (DFA - *Deterministic Finite Automata*) и недетерминистичке коначне аутомате (NFA - *Nondeterministic Finite Automata*). Детерминистички коначни аутомат је аутомат који прихвата коначне низове симбола тако што за свако стање и одређени улазни симбол једнозначно дефинише следеће стање аутомата. Код недетерминистичког коначног аутомата за сваки пар стања и улазног симбола може постојати више од једног следећег стања (Indu, 2016).

Недетерминистички коначни аутомат састоји се од:

- Коначног скупа стања Q,

- Скупа улазних симбола Σ, улазне азбуке (претпоставка да Ɛ никада није члан Σ),
- Функције прелаза (транзиције) δ, која за свако стање и за сваки симбол из Σ ∪ {Ɛ} даје скуп следећих стања,
- Скупа почетних односно иницијалних стања S ($S \subseteq Q$),
- Скупа стања F ($F \subseteq Q$), који представља скуп крајњих односно завршних стања прихватања.

Дакле, недетерминистички коначни аутомат M је уређена петорка:

$$M=(Q, \Sigma, \delta, S, F).$$

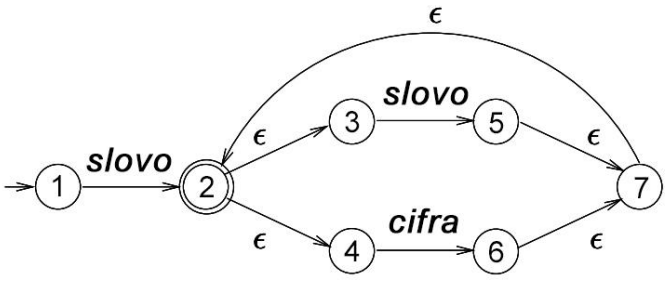
Детерминистички коначни аутомат је специјални случај недетерминистичког где:

- Прелази за улазни празан низ Ɛ (Ɛ -прелази) нису допуштени;
- За свако стање s и улазни симбол a, постоји тачно један прелаз;
- Постоји само једно почетно стање q_0 ($q_0 \in Q$).

Детерминистички коначни аутомат се дефинише као петорка:

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

За графичко представљање NFA и DFA користе се графови (дијаграми стања) где чворови представљају стања, а потези одговарају функцији прелаза. На пример, изглед графа за NFA који препознаје идентификатор дефинисан као *slovo*(*slovo*|*cifra*)*, приказан је на слици 6.



Слика 6. Недетерминистички коначни аутомат који препознаје идентификатор дефинисан регуларним изразом *slovo*(*slovo*|*cifra*)*

Такође, NFA и DFA се могу представити помоћу табеле прелаза, односно транзиција чији редови одговарају стањима, а колоне улазним симболима и Ɛ. У свако поље табеле за одговарајуће стање и улаз се уноси вредност функције

прелаза која се добија за дате аргументе. Ако функција прелаза нема информације о том пару стање-улаз, у табелу за тај пар се може ставити ∅. За горе наведени NFA табела прелаза је приказана на слици 7.

СТАЊЕ	*slovo*	*cifra*	*ε*
1	2	∅	∅
2	∅	∅	3,4
3	5	∅	∅
4	∅	6	∅
5	∅	∅	7
6	∅	∅	7
7	∅	∅	2

Слика 7. Табела прелаза за NFA дефинисан регуларним изразом *slovo*(*slovo*|*cifra*)*

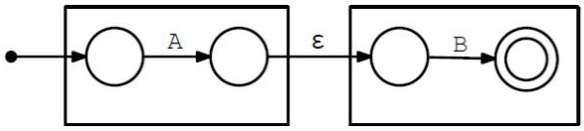
2.4.3 Конверзија регуларних израза у коначне аутомате (Томсонов алгоритам)

За сваки регуларни израз постоји одговарајући коначни аутомат и обрнуто (Sipser, 2013). Конвертовање регуларног израза у недетерминистички коначни аутомат може се извршити применом алгоритама који су најпре дефинисали McNaughton и Yamada (1960), а затим је на њему радио Thompson (1968). Овај алгоритам се у литератури може пронаћи под називом McNaughton–Yamada–Thompson или чешће као Thompson-ов алгоритам. Алгоритам ради рекурзивно раздвајањем израза на његове саставне подизразе од којих се NFA конструише применом скупа правила (Xing, 2004). Најпре се конструишу аутомати за издвојене подизразе, а затим се, користећи епсилон прелазе (ε), ови аутомати спајају у складу са правилима која комбинују регуларне изразе.

Скуп правила која користи Томсонов алгоритам објашњена су у наставку. Прво се конструишу аутомати који одговарају основним случајевима изведених подизраза. Аутомат који одговара регуларном изразу са једним симболом има само почетно и коначно стање, а прелаз који повезује стања означен је симболом који чини регуларни израз (епсилон или симбол из азбуке). Ова правила приказана су на слици 8.

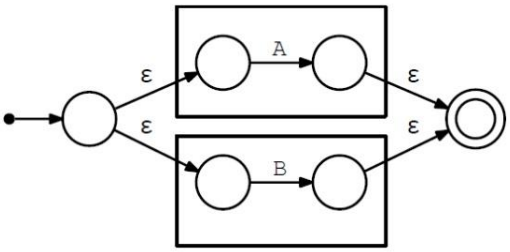
Слика 8. NFA за регуларни израз са једним симболом азбуке (лево) и NFA за регуларни израз са празаним низом ε (десно)

Уколико постоје конструисани NFA за регуларне изразе *A* и *B* онда се NFA за регуларни израз *AB*, односно операцију конкатенације добија једноставно повезивањем *A* и *B* епсилон транзицијом. Почетно стање *A* постаје почетно стање комбинације, а стање прихватања *B* постаје завршно стање комбинације (слика 9).



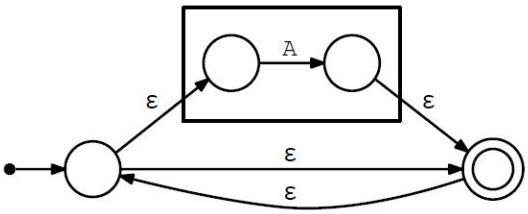
Слика 9. NFA за регуларни израз *AB* (операција конкатенације)

На сличан начин операција алтернације између *A* и *B* написана као *A* | *B* може се приказати као два NFA аутомата спојена заједничким почетним и заједничким стањима прихватања. Спајање се врши епсилон транзицијама што се може видети на слици 10.



Слика 10. NFA за регуларни израз *A* | *B* (операција алтернације)

Операција Kleene closure *A** може се претворити у NFA тако што се најпре врши додавање почетних и завршних чворова аутомату за регуларни израз *A*, а затим додавање епсилон прелаза да би се омогућило нула или више понављања (слика 11).



Слика 11. NFA за регуларни израз *A** (операција Kleene closure)

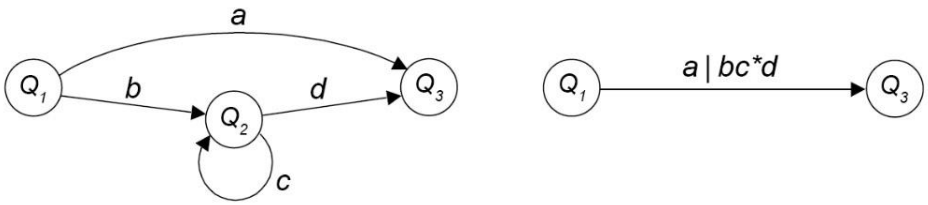
2.4.4 Конверзија коначних аутомата у регуларне изразе (Brzozowski and McCluskey алгоритам)

Једна од метода конверзије коначних аутомата у регуларне изразе позната је као метода елиминације стања или оригинално Brzozowski and McCluskey (1963) алгоритам. Састоји се у елиминацији стања аутомата, једно за другим, уз трансформацију транзиција тако да језик који резултујући аутомат прихвата остане непромењен (Wood, 1987). Коначном аутомату се додаје ново иницијално стање S_i које се повезује са постојећим иницијалним стањем, а свако стање прихватања коначног аутомата се повезује са новим финалним стањем S_f као што је приказано на слици 12.

Слика 12. Додавање новог иницијалног S_i и новог финалног стања S_f

Након тога започиње итеративни поступак елиминације стања коначног аутомата једно за другим уз формирање одговарајућих нових транзиција. На крају алгоритма остају само два стања која су додата на почетку S_i и S_f, као и једна транзиција између њих. За свако елиминисано стање производи се регуларни израз као ознака новој транзицији. Креирани регуларни израз представља улаз ка стању које треба следеће елиминисати. У елиминацији

стања регуларни израз се акумулира. Пример једног корака методе елиминације стања приказан је на слици 13.



Слика 13. Пример једног корака *state-elimination* алгоритма

Леви дијаграм приказује стање Q_2 које треба елиминисати и транзиције које треба трансформисати. Десни дијаграм приказује аутомат након елиминације датог стања и нову ознаку за прелаз са Q_1 на Q_3. Језици које је аутомат прихватао пре и после елиминисања стања Q_2 су једнаки.

2.4.5 Минимизација коначних аутомата (Hopcroft алгоритам)

Минимизација коначног аутомата је проблем који се проучава од педесетих година прошлог века (Huffman, 1954; Moore, 1956). Решавање овог проблема подразумева проналазак еквивалентног коначног аутомата са минималним бројем стања који прихвата исти језик као и коначни аутомат који се минимизира. За добијање минималног еквивалентног аутомата потребна је техника за откривање еквивалентних стања, то јест стања која се исто понашају за било који симбол улазног стринга (Cooper and Torczon, 2012). За дати DFA $A = (Q, \Sigma, \delta, q_0, F)$ кажемо да су два стања $q_1, q_2 \in Q$ еквивалентна и означавамо као $q_1 \approx q_2$, ако за сваки симбол $u \in \Sigma$ важи $\delta(q_1,u)=q_n \Leftrightarrow \delta(q_2,u)=q_n$, где је $q_n \in Q$.

Неки DFA могу да садрже такозвана недостижна и мртва стања. Недостижна стања су она до којих се не може доћи из почетног стања за било који улазни стринг. Нека је дат аутомат $A = (Q, \Sigma, \delta, q_0, F)$ са иницијалним стањем q_0 и скупом завршних стања F. За стање $q \in Q$ кажемо да је достижно стање ако постоји стринг $u \in \Sigma^*$ тако да је $\delta(q_0,u)=q$. У супротном, q је недостижно (сувишно) стање. Мртва стања су она из којих се не може доћи до

коначних стања. Јасно је да ни недостижна ни мртва стања немају утицај на језик који препознаје дати DFA тако да процес минимизације треба почети елиминацијом недостижних и мртвих стања.

Током година предложено је неколико алгоритама за решавање проблема минимизације од којих су већина неефикасни за коначне аутомате са великим бројем стања. Један од ефикаснијих са најбољим временом потребним за обављање процеса минимизације је Hopcroft-ов алгоритам (Watson, 1993). Hopcroft (1971) је дефинисао метод минимизације који је познат као техника уситњења партиција. Ова техника добро је позната не само у теорији аутомата, већ и у бројним областима рачунарских наука попут теорије графова, стрингова и Булових матрица.

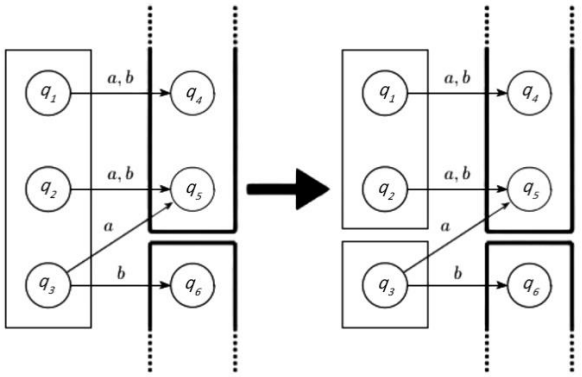
Ако су два стања q_1 и q_2 еквивалентна, онда се сви прелази на q_2 могу заменити прелазима на q_1, односно два стања се могу спојити у једној заједничкој групи, то јест партицији (Fischer, Cytron and LeBlanc Jr, 2010). Партиција је конзистентна ако су сва стања у њој еквивалентна. Основни кораци у Hopcroft-овом алгоритму минимизације су следећи:

1. Најпре се формирају иницијалне партиције: једну партицију чине сва коначна стања F, док другу сва преостала $Q\backslash F$. Ове две групе нису означене као конзистентне;

2. Бира се било која неозначена партиција G и проверава њена конзистентност за сваки симбол азбуке. Ако се утврди неконзистентност за неки симбол, група G се дели на подгрупе (слика 14). У супротном ова група се означава као конзистентна;

3. Ако више нема неозначених партиција, алгоритам се завршава, а формиране партиције представљају стања минималног DFA. У супротном, алгоритам се враћа на корак 2.

Почетно стање минималног DFA је партиција која садржи првобитно почетно стање, а све партиције које садрже првобитна стања прихватања представљају коначна стања у минималном DFA.



Слика 14. Пример поделе партиција код Hopcroft-овог алгоритма

2.4.6 Конверзија NFA у DFA применом алгоритма за конструкцију подскупова

Два аутомата су еквивалентна уколико препознају исти језик. За сваки недетерминистички постоји еквивалентан детерминистички коначни аутомат (Sipser, 2013). За одређени NFA може се конструисати еквивалентни DFA применом алгоритма конструкције подскупова (енгл. *subset construction*). Назив алгоритма произилази из примењене технике где свако стање DFA одговара скупу стања NFA. Овакав начин конструкције први су објавили Rabin и Scott (1959), па се у литератури ова техника може наћи и под називом Rabin-Scott-ова конструкција.

За детаљнији опис алгоритма конструкције подскупова неопходно је најпре дефинисати функцију *ε*-затварања (енгл. *ε-closure*). За дати скуп NFA стања *Q,* функција *ε*-затварања се дефинише као:

$$\varepsilon\text{-}closure(q_i) = \{\, q \mid q \in \delta\,(q_i, \varepsilon),\ q \in Q \,\}$$

где q представља сва стања скупа Q која су ε-прелазима доступна из стања q_i.

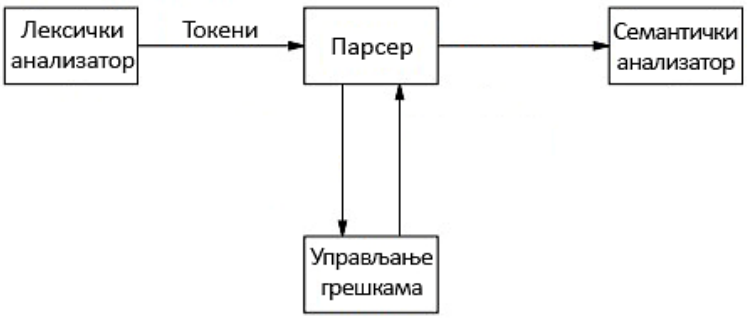
Нека је $N = (Q, \Sigma, \delta, q_0, F)$ недетерминистички коначни аутомат који препознаје језик A. За конструкцију еквивалентног детерминистичког коначног аутомата $D = (Q', \Sigma, \delta', q_0', F')$ који препознаје језик A треба предузети следеће кораке:

- Почетно стање DFA се добија применом функције ε-затварања на почетно стање NFA, $q_0' = \varepsilon\text{-}closure\ (q_0)$;
- Почетно стање DFA се додаје скупу стања Q'. Сада се прате све транзиције из овог стања;
- Ако се из датог стања врши прелазак у више стања за исти улазни знак азбуке, тај скуп стања треба третирати као једно стање DFA;
- Уколико су формирана нова стања, она се додају скупу Q' и понавља се претходни корак. У супротном, када нема више нових стања, итеративни поступак се завршава;
- Скуп стања прихватања DFA F' чине сва стања која садрже бар једно NFA стање прихватања.

2.5 СИНТАКСНА АНАЛИЗА

Лексичком анализом улазни низ се дели на одговарајуће токене, док је задатак синтаксне анализе рекомбиновање ових токена не назад у исти облик улазних карактера, већ у такозвано стабло парсирања које одражава структуру улазног низа симбола. Листови овог стабла су токени пронађени лексичком анализом и ако се листови читају с лева на десно секвенца је иста као у улазном тексту. Синтаксни анализатор или такозвани парсер (слика 15) врши проверу синтаксне усаглашености улазног низа са дефинисаном граматиком формалног језика. За улазни низ се може рећи да је синтаксно исправан само ако се може доказати да речи тог низа припадају датој граматици. Да би се доказало да одређена реченица припада неком језику, мора се показати да постоји колекција правила граматике тог језика чијом применом се од почетног симбола може доћи до жељене реченице. Таква

примена скупа одговарајућих правила граматике се назива извођење (енгл. *derivation*).



Слика 15. Позиција парсера у моделу програмског преводиоца

Улога парсера је двострука. С једне стране, одговоран је за пријављивање сваке синтаксне грешке на разумљив начин и у том контексту би требао да буде у стању да изврши опоравак од уобичајених грешака да би наставио са даљим радом. С друге стране, ако утврди да је изворни програм тачан у погледу синтаксне структуре, његов задатак је генерисање синтаксног стабла.

Постоје три општа типа синтаксних анализатора: *универзални*, *силазни* (енгл. *top-down*) и *узлазни* (енгл. *bottom-up*). Универзалне методе парсирања, као што су Cocke-Younger-Kasami алгоритам и Earley алгоритам, омогућавају анализирање било које бесконтексне граматике. Међутим, ове методе су сувише неефикасне да би се користиле у изради компајлера (Su and Yan, 2011). Методе које се обично користе у компајлерима су *top-down* која врши анализу одозго наниже и *bottom-up* којом се анализа врши одоздо навише. Као што њихова имена указују, *top-down* парсери граде стабла парсирања од врха (корена) до дна (лишћа), док *bottom-up* парсери граде стабла парсирања од лишћа до корена.

Конструкција синтаксног стабла врши се применом дефинисаних правила граматике. Код бесконтексних граматика у сваком кораку извођења по један нетерминал се замењује десном страном одговарајућег правила пресликавања у којем се дати нетерминал појављује на левој страни. Кораци

се понављају све док не преостану само терминали. Формално, релација извођења се врши применом следећих основних правила (Mogensen, 2009):

- $\alpha N \beta \implies \alpha\gamma\beta$, ако постоји продукција $N \to \gamma$
- $\alpha \implies \gamma$, ако постоји β тако да важи $\alpha \implies \beta$ и $\beta \implies \gamma$

где су α, β и γ симболи граматике (терминали или нетерминали), а N нетерминални симбол.

Редослед примене правила може бити:

- с десна – десно извођење (енгл. *rightmost derivation*) када се за замену бира крајњи десни нетерминал у свакој реченици,
- с лева – лево извођење (енгл. *leftmost derivation*) када се за замену бира крајњи леви нетерминал у свакој реченици.

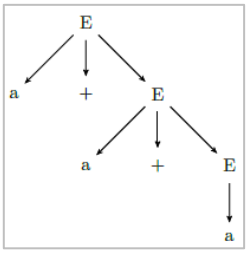
На пример, ако имамо граматику *G* са следећим правилима:

1. E → a,
2. E → a + E.

Десно извођење израза *a+a+a* се врши на следећи начин:

E ⟹ a+E ⟹ a+a+E ⟹ a+a+a.

Стабло парсирања за овај пример приказано је на слици 16.

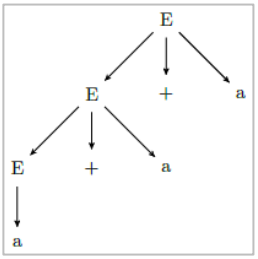


Слика 16. Пример стабла парсирања за десно извођење

Лево извођење израза *a+a+a* за иста граматичка правила, врши се на следећи начин:

E ⟹ E+a ⟹ E+a+a ⟹ a+a+a.

Стабло парсирања за пример левог извођења приказано је на слици 17.



Слика 17. Пример стабла парсирања за лево извођење

2.5.1 LL(1) анализатори

У току извођења може се десити да у неком кораку не постоји адекватно правило пресликавања и у том случају врши се враћање уназад (енгл. *backtracking*) све док се не пронађе адекватна алтернативна смена. Да би се обезбедила ефикасност анализе, тежи се избегавању враћања уназад, односно потребно је да процес иде само у једном правцу. Управо из те идеје су развијени LL(k) и LR(k) анализатори. Конкретно, LL(k) користи *top-down* методу анализе, док LR(k) *bottom-up* методу. Ови анализатори се заснивају на предиктивним алгоритмима парсирања што значи да се правило пресликавања не бира на основу првог нетерминала у правилу, већ избор зависи и од симбола у улазном стрингу. Генерално, анализатори се могу дефинисати као LL(k), где *k* представља дужину речи на основу које се врши предикција, али се најчешће у пракси користи LL(1). Да би предикција била могућа, потребно је да граматика којом је описан језик буде тако дефинисана да за један улазни симбол и један нетерминал постоји највише једно пресликавање.

Конструкцији предиктивних парсера помажу две функције, *FIRST* и *FOLLOW* које су повезане са одговарајућом граматиком (Mogensen, 2009). Током *top-down* парсирања *FIRST* и *FOLLOW* функције обезбеђују предикцију, односно омогућавају да се изабере одговарајуће пресликавање које ће се применити, на основу следећег улазног симбола. Током опоравка од грешке скупови токена произведени од стране *FOLLOW* могу се користити као синхронизујући токени.

FIRST(α) је скуп свих терминала који се налазе на почетку стрингова изведених из *α*, где је *α* било који стринг граматичких симбола. Симбол *c* припада скупу *FIRST(α)* ако и само ако постоји продукција $\alpha \Longrightarrow c\beta$ где је *β* нека секвенца граматичких симбола. *FOLLOW(A)* је скуп свих терминала који могу да следе иза нетерминала *A* у току извођења. Симбол *c* припада скупу *FOLLOW(A)* ако и само ако постоји извођење из почетног симбола граматике *S* такво да је $S \Longrightarrow \alpha A c \beta$, где су *α* и *β* секвенце граматичких симбола.

Алгоритам LL(1) парсера може се реализовати применом синтаксне табеле (енгл. *parsing table*) на којој се заснива процес одлучивања о примени одговарајућих правила извођења. За имплементацију парсера могу се користити рекурзивне функције које одражавају акције из синтаксне табеле или се може директно користити синтаксна табела и стек за представљање тренутног скупа нетерминала (Douglas, 2020). За креирање синтаксне табеле користе се информације из формираних *FIRST* и *FOLLOW* скупова. На слици 18 приказан је пример формирања синтаксне табеле на основу скупова за предикцију за задату граматику *G*.

Граматика G

$$
\begin{array}{lcl}
E & \rightarrow & TE' \\
E' & \rightarrow & +TE' \mid \epsilon \\
T & \rightarrow & FT' \\
T' & \rightarrow & *FT' \mid \epsilon \\
F & \rightarrow & (E) \mid \textbf{id}
\end{array}
$$

FIRST и FOLLOW скупови за граматику G

$$
\begin{array}{rcl}
\text{FIRST}(E) & = & \text{FIRST}(T) = \text{FIRST}(F) = \{(, \textbf{id}\} \\
\text{FIRST}(E') & = & \{+, \epsilon\} \\
\text{FIRST}(T') & = & \{*, \epsilon\} \\
\text{FOLLOW}(E) & = & \text{FOLLOW}(E') = \{), \$\} \\
\text{FOLLOW}(T) & = & \text{FOLLOW}(T') = \{+,), \$\} \\
\text{FOLLOW}(F) & = & \{*, +,), \$\}
\end{array}
$$

Синтаксна табела за граматику G

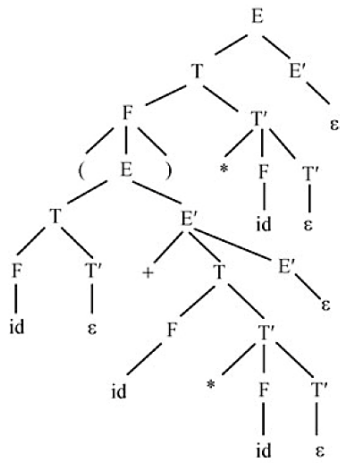
Нетерминали	Улазни симболи					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow \textbf{id}$			$F \rightarrow (E)$		

Слика 18. Пример формирања синтаксне табеле за задату граматику *G*

Граматика *G* дефинише аритметичке изразе са два оператора + и * при чему дозвољава и употребу заграда тако да је дозвољени скуп терминала следећи: *id*, +, *, *(* и *)*. Процес LL(1) парсирања може се демонстрирати на примеру извођења израза *(id + id * id) * id*. На слици 19 поред свих корака

извођења приказано је и синтаксно стабло како би се јасније разумео поступак LL(1) синтаксне анализе.

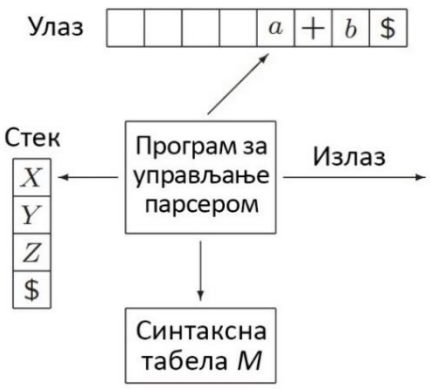
$$
\begin{aligned}
\mathrm{E} &\rightarrow \mathrm{TE}' \\
&\rightarrow \mathrm{FT}'\mathrm{E}' \\
&\rightarrow (\mathrm{E})\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{TE}')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{FT}'\mathrm{E}')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id}\mathrm{T}'\mathrm{E}')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id}\varepsilon\mathrm{E}')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{TE}')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{FT}'\mathrm{E}'')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{id} * \mathrm{FT}'\mathrm{E}')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{id} * \mathrm{id}\mathrm{T}'\mathrm{E}')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{id} * \mathrm{id}\varepsilon\mathrm{E}')\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{id} * \mathrm{id}\varepsilon)\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{id} * \mathrm{id}) * \mathrm{FT}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{id} * \mathrm{id}) * \mathrm{id}\mathrm{T}'\mathrm{E}' \\
&\rightarrow (\mathrm{id} + \mathrm{id} * \mathrm{id}) * \mathrm{id}.
\end{aligned}
$$



Слика 19. Поступак LL(1) синтаксне анализе за граматику *G* и задати израз *(id + id * id) * id*, са одговарајућим синтаксним стаблом

2.5.2 Нерекурзивни предиктивни парсер

Нерекурзивни парсер се може реализовати експлицитно коришћењем стека, а не имплицитно путем рекурзивних процедура (Aho et al, 2007). Нерекурзивни предиктивни парсер користи лево извођење. На слици 20 се може видети модел овог парсера. Састоји се од улазног бафера, стека који садржи секвенцу граматичких симбола, синтаксне табеле и излаза.

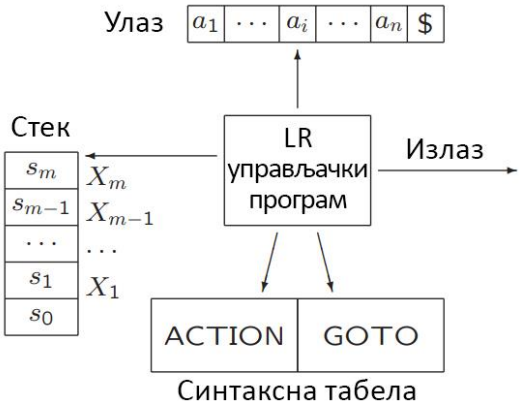


Слика 20. Предиктивни парсер заснован на синтаксној табели

Улазни бафер садржи низ који се парсира. Симбол *$* се користи да означи дно стека. Синтаксна табела се користи да одреди које правило треба применити за било коју комбинацију нетерминала на стеку и следећег симбола на улазу. Парсером управља програм који прво узима *X*, симбол на врху стека, и *a*, тренутни улазни симбол. Ако је *X* нетерминал, парсер бира продукцију за *X* из табеле парсирања *M* на позицији *M*[*X*, *a*]. У супротном, проверава се подударање између терминала *X* и тренутног улазног симбола *a*.

2.5.3 LR(0) анализатори

Концепт LR парсирања увео је Knuth (1965). Најчешће коришћени тип анализатора у комерцијалним решењима се заснива на концепту LR(k) анализе. У питању су ефикасни *bottom-up* анализатори којим се могу препознати практично све конструкције програмских језика које се могу написати применом бесконтексних граматика. Овим анализаторима се препознаје класа граматика која представља надскуп LL граматика. Модел LR анализатора може се видети на слици 21.



Слика 21. Модел LR анализатора

LR анализатор се реализује као аутомат који користи стек као меморију. Састоји се од улазног бафера, излаза, стека, управљачког програма и синтаксне табеле која има два дела – ACTION и GOTO (акције и прелази). Управљачки програм је исти за све типове LR парсера само се синтаксне

табеле разликују. Програм парсера чита знакове са улазног бафера један по један. Акције могу бити (Stanković, Stojković and Tošić, 2018):

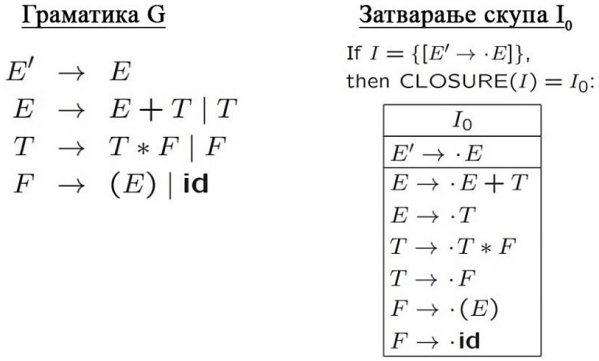
- *shift p* – са улазног бафера се узима текући знак и поставља на стек заједно са следећим стањем p након чега се прелази на анализу наредног знака са улаза;
- *reduce k* – врши се редукција употребом пресликавања k. За пресликавање $A \rightarrow \beta$ са стека треба уклонити укупно $2 \cdot \text{length}(\beta)$ елемената, на стек поставити нетерминал A и ново стање узети из GOTO дела синтаксне табеле;
- *accept* – стринг је препознат и анализа се прекида;
- *error* – грешка, улазни стринг се не може препознати и анализа се прекида.

ACTION део синтаксне табеле специфицира акцију парсера на основу датог стања парсера и следећег знака на улазу. То значи да се овај део синтаксне табеле састоји од колона које представљају терминале и редова који се односе на стања. Функција GOTO одређује које стање треба ставити на врх стека након редукције. Код GOTO дела синтаксне табеле редови представљају стања аутомата, а колоне нетерминале.

LR(0) аутомат се формира на основу LR(0) чланова (енгл. *LR(0) items*) који се генеришу применом граматичких правила при чему се води рачуна о тренутно препознатом делу речи (тренутно стање парсера). За раздвајање дела речи које су препознате од оних које још увек нису користи се тачка (•) код LR(0) чланова. На овај начин се назначава део улаза који је скениран до дате тачке у процесу парсирања (Sudkamp, 2007). На пример, на основу продукције $A \rightarrow XYZ$ могу се добити четири LR(0) члана:

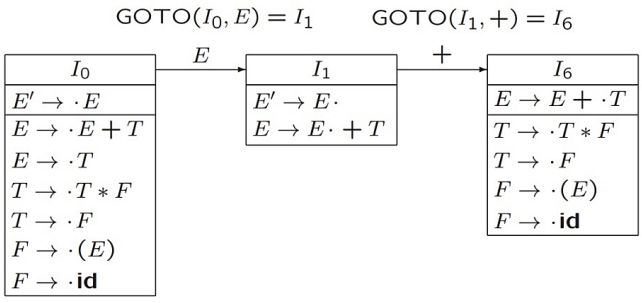
1. $A \rightarrow \bullet XYZ$
2. $A \rightarrow X \bullet YZ$
3. $A \rightarrow XY \bullet Z$
4. $A \rightarrow XYZ \bullet$

Конструисање аутомата почиње формирањем првог стања I_0, коришћењем продукције за почетни симбол (нпр. $S \rightarrow A$) и додавањем тачке на почетку десне стране ($S \rightarrow \bullet A$). Затим, применом операције затварања (eng. *closure*), формира се скуп LR(0) чланова за дато стање. За сваки члан у стању са нетерминалом, одмах десно од тачке додајемо сва правила граматике која имају дати нетерминал на левој страни продукције. Затварање овог стања је завршено тек када се примене све могуће смене на описани начин. На слици 22 приказана је граматика *G* и скуп LR(0) чланова који одговара стању I_0. Овај скуп се добија применом операције затварања на почетни члан.



Слика 22. Пример примене операције затварања (*closure*)

Формирање следећег стања аутомата врши се помоћу функције *goto(I, A)*, где се *I* односи на скуп чланова док *A* представља симбол граматике. Примена функције *goto* над скупом I_0 из претходног примера врши се за све граматичке симболе испред којих се налази тачка (*E*, *T*, *F*, *(*, *id*). Након тога функција *goto* се примењује и над новим скуповима. Овакав поступак затварања скупова понавља се све док је то могуће. Примена функције *goto*(I_0, *E*) и *goto*(I_1, *+*) за граматику из претходног примера може се видети на слици 23.



Слика 23. Пример функције *goto*

Пример LR(0) синтаксне табеле за задату граматику *G* приказан је на слици 24.

Граматика **G**

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to \mathbf{id}$

	ACTION						GOTO		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Слика 24. LR(0) синтаксна табела за задату граматику *G*

Ако се на улаз синтаксног анализатора, реализованог помоћу LR(0) табеле са слике 24, доведе израз облика *id * id + id* поступак парсирања по корацима се може пратити на слици 25.

	Стек	Симболи	Улаз	Акције
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	shift
(2)	0 5	**id**	$* \mathbf{id} + \mathbf{id}\,\$$	reduce $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reduce $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	shift
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	shift
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reduce $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reduce $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reduce $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	shift
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	shift
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reduce $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reduce $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reduce $E \to E + T$
(14)	0 1	E	$\$$	accept

Слика 25. Пример процеса LR(0) парсирања

У првом кораку, ред (1) табеле са слике 25, LR(0) парсер је у стању 0 (почетном стању) и на улазу има први симбол *id*. Акција у реду 0 и колони *id* синтаксне табеле са слике 24 је s5. Због тога се у другом кораку парсирања, ред (2), може видети да је на врху стека стање 5, док је симбол *id* уклоњен са улаза. Затим, следећи улазни симбол је *, па је акција за стање 5 и улазни симбол * reduce (*F→ id*). Једно стање са стека се уклања и тада на врху остаје стање 0. Пошто функција goto за стање 0 и симбол *F*, даје стање 3, сада у трећем

кораку, ред (3), на врху стека је 3. Преостали кораци парсирања се предузимају на сличан начин.

2.6 СЕМАНТИЧКА АНАЛИЗА

Најопштије речено, синтакса се тиче форме програмског кода, док се семантика односи на његово значење. Семантички анализатор користи синтаксно стабло и информације из табеле симбола да провери семантичку усклађеност изворног програма са дефиницијом језика. Такође, прикупља информације о типу и чува их у синтаксном стаблу или у табели симбола за даљу употребу приликом генерисања међукода. Провера типа је важан део семантичке анализе где програмски преводилац проверава да ли сваки оператор има одговарајуће операнде. На пример, многе дефиниције програмског језика захтевају да индекс низа буде цео број тако да програмски преводилац мора да пријави грешку ако се користи децимални број за индексирање низа. Све ове информације се не могу представити формалним описом језика, већ се симболима граматике додају одређени атрибути. На пример, у некој граматици можемо да повежемо атрибут *val* са одређеним граматичким симболом *X*. У том случају *X.val* представља аритметичку вредност. Међутим, потребан је и додатни скуп правила за сваку продукцију како би се представила релација између вредности различитих граматичких симбола. За придруживање семантичких рутина правилима граматике користе се синтаксно управљане дефиниције (енгл. *Syntax Directed Definitions*).

2.6.1 Табеле симбола

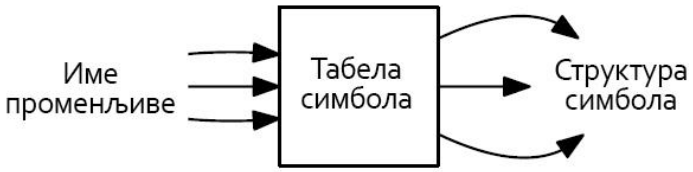
Табеле симбола су структуре података које програмски преводиоци користе за чување информација о конструкцијама изворног програма. Информације се прикупљају у фазама анализе компајлера и користе у фазама синтезе за генерисање циљног кода. Табеле симбола садрже информације о идентификатору, типу, позицији у складишту и друге релевантне

информације (Muchnick, 1997). Пример табеле симбола приказан је на слици 26.

ТОКЕН	СИМБОЛ	ТИП
id1	a	int
id2	b	real
id3	c	boolean
id4	opis	string
const1	55	int

Слика 26. Пример складиштења информација у табели симбола

Табеле симбола треба да подрже више декларација истог идентификатора у оквиру програма. У семантичкој анализи табеле симбола се користе за проверу типова података у изразима и за проверу опсега важења имена. Дакле, табела симбола се може посматрати као мапа између имена сваке променљиве и структуре симбола која је описује (Douglas, 2020), као што је илустровано на слици 27.



Слика 27. Табела симбола мапира име променљиве са припадајућом структуром симбола

2.6.2 Синтаксно управљане дефиниције

Атрибути се повезују са нетерминалима и терминалима, а правила се додају продукцијама граматике. Правила описују како се израчунавају вредности атрибута на одговарајућим чворовима синтаксног стабла. Синтаксно управљане дефиниције повезују:

- Уз сваки граматички симбол, скуп атрибута,
- Уз сваку продукцију, сет семантичких правила за израчунавање вредности атрибута повезаних са симболима који се појављују у продукцији.

Претпоставимо да је чвор *N* у синтаксном стаблу означен граматичким симболом *X*. Да би се означила вредност атрибута *a* за *X* на том чвору записује се као *X.a*. Синтаксно стабло које на сваком чвору приказује вредности атрибута назива се анотирано синтаксно стабло (енгл. *annotated parse tree*). Атрибути могу бити бројеви, типови, низови… Обично се разликују два типа атрибута (Jovanović, 2023):

- Синтетизовани атрибут (енгл. *synthesised attribute*) се дефинише у функцији вредности атрибута самог чвора и вредности атрибута потомака чвора. Синтетизовани атрибути се прослеђују навише кроз синтаксно стабло, од листова до корена.
- Наслеђени атрибут (енгл. *inherited attribute*) се дефинише у функцији вредности атрибута самог чвора, вредности атрибута осталих чворова на истом нивоу и вредности родитељских атрибута. Наслеђени атрибути се прослеђују наниже кроз синтаксно стабло.

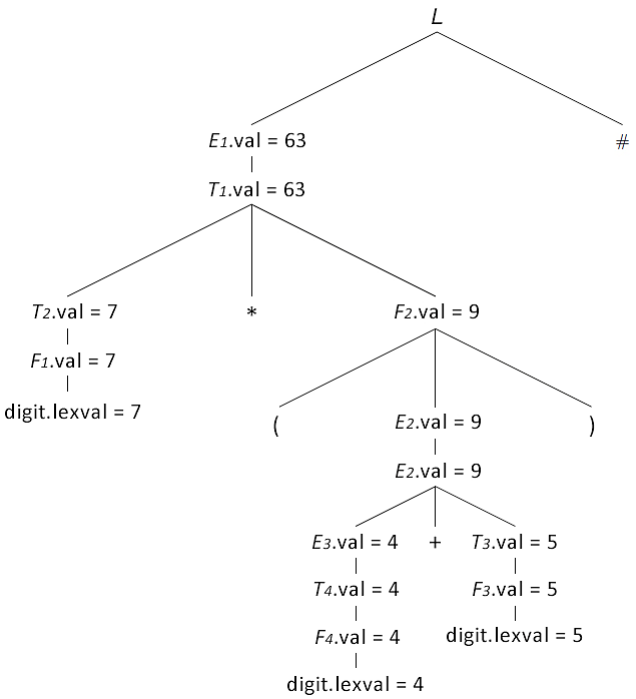
На слици 28 представљена је граматика за једноставне изразе са операторима сабирања и множења (+, ∗), пример израза на основу дефинисане граматике који ће бити коришћен за анализу је *digit∗(digit+digit)*. Вредности атрибута нетерминалних симбола израчунавају се семантичким рутинама које су придружене правилима пресликавања. Сваки нетерминални симбол има атрибут *val* преко којег се израчунавају вредности израза. Такође, терминал *digit* има атрибут *lexval* што је целобројна вредност коју враћа лексички анализатор.

Продукције (правила граматике)	Семантичке рутине
1. $L \rightarrow E\#$	print(*E*.val)
2. $E \rightarrow E + T$	*E*.val := *E*.val + *T*.val
3. $E \rightarrow T$	*E*.val := *T*.val
4. $T \rightarrow T * F$	*T*.val := *T*.val ∗ *F*.val
5. $T \rightarrow F$	*T*.val := *F*.val
6. $F \rightarrow (E)$	*F*.val := *E*.val
7. $F \rightarrow$ digit	*F*.val := digit.lexval

Слика 28. Пример синтаксно вођене дефиниције за једноставне изразе са операцијама сабирања и множења

Семантичка правила долазе у два облика (Scott, 2006). Правила у форми *E*.val := *T*.val су позната као правила копирања и наводе да један атрибут треба да буде копија другог. Остала правила позивају семантичке функције. У овом примеру семантичке функције су познате аритметичке операције. Генерално гледано, то могу бити произвољно сложене функције које је одредио дизајнер језика. Свака семантичка функција узима произвољан број аргумената и свака израчунава један резултат који такође мора бити додељен атрибуту граматичког симбола у тренутној продукцији.

На слици 29 представљено је анотирано синтаксно стабло за улазни низ *7∗(4+5)* конструисано на основу граматике са слике 28.



Слика 29. Анотирано синтаксно стабло за израз 7∗(4+5)

Атрибутивне граматике имају практичну употребу само уз одговарајуће евалуаторе који могу да тумаче правила аутоматски. У литератури су предложене многе технике евалуације атрибута (Cooper and Torczon, 2012). Евалуатори атрибута се могу поделити у две основне категорије: динамички и статички евалуатори (Alblas, 1991). На основу изворног програма и одговарајућег анотираног синтаксног стабла, динамички евалуатор

конструише граф зависности (енгл. *dependency graph*) и врши тополошко сортирање чворова да би одредио редослед евалуације инстанци атрибута. Док анотирано синтаксно стабло приказује вредности атрибута, граф зависности приказује ток информација међу инстанцама атрибута. Смер од једне инстанце атрибута до друге значи да је вредност прве потребна за израчунавање друге. Статички евалуатор не конструише граф зависности, уместо тога редослед евалуације је заснован на зависностима између појављивања атрибута у продукцијама атрибутивне граматике.

3. СОФТВЕРСКИ СИСТЕМИ ЗА СИМУЛАЦИЈУ И ВИЗУЕЛИЗАЦИЈУ АПСТРАКТНИХ ТЕОРИЈСКИХ КОНЦЕПАТА

На основу претходног теоријског прегледа може се закључити да је превођење програма веома комплексан процес који се састоји из више фаза које обилују апстрактним концептима. Апстрактна теорија представљена на традиционалан начин често изазива апатију код студената, док њено повезивање са нечим стварним и физичким обично доводи до већег интересовања и ентузијазма. У овом поглављу говори се о значају употребе образовних софтверских система у наставном процесу који представљају ефикасне помоћне алате за савладавање комплексних теоријских конструкција у инжењерском образовању.

3.1 СОФТВЕРСКИ СИСТЕМИ ЗА ПОМОЋ У УЧЕЊУ

Увођење и усвајање нових информационих технологија у учењу и настави брзо је еволуирало последњих година. Улога технологије у високом образовању је да подстакне студенте на размишљање о проблему проучавања и да унапреди образовни процес, а не да га сведе на скуп процедура за испоруку садржаја. Због тога се у овој секцији наводе битни аспекти софтверских система који су неопходни да би се систем могао окарактерисати као образовни, односно као систем за помоћ у учењу.

3.1.1 Интерактивни аспект образовања

У циљу постизања ефективнијих педагошких резултата, потребно је напустити одређене стереотипе традиционалне наставе која подразумева вербално и једносмерно преношење знања – наставник предаје, студент слуша. Савремене наставне методе подразумевају иновативне приступе. Интерактивност је кључ ефективног и ефикасног процеса подучавања и учења

где наставник може привући пажњу студента, а студенти могу научити више у поређењу са традиционалном методом. Sims (1997) у свом раду наводи да интерактивност игра пресудну улогу у стицању знања и развоју когнитивних вештина и да је интеракција својствена ефективној наставној пракси и индивидуалном откривању. Термин „интерактивни“ појављује се у два различита склопа образовног истраживачког дискурса, један се односи на педагогију, а други се тиче нових технологија у образовању (Beauchamp and Kennewell, 2010). Педагошка интерактивност се односи на интеракцију наставника и студената. Употреба технологије у образовању подразумева ефикасну интеграцију информационо-комуникационих технологија (ИКТ) у наставни процес.

Wang (2008) у свом истраживању предлаже генерички модел за ефикасну интеграцију ИКТ у процес подучавања и учења. По том моделу, педагогија, социјална интеракција и технологија представљају кључне компоненте технолошки унапређеног окружења за учење. Интерактивност у таквом окружењу може се графички представити као на слици 30.

Слика 30. Интерактивност на основу Wang (2008) модела

У дигиталној ери студенти желе већу активност на часу која се не своди само на седење, држање оловке и књиге (Pradono, Astriani, and Moniaga, 2013). Увођење интерактивних алата заснованих на ИКТ може подстаћи ученике да се сами мотивишу и усмеравају своје учење (Evans and Gibbons, 2007). Такође, ови алати могу побољшати учење јер омогућавају корисницима да сами

контролишу процес учења и пружају им могућност прегледања садржаја, прескакања и понављања по потреби (Wang, Vaughn and Liu, 2011).

Ако се конкретизује појам интерактивности на ИКТ, онда се може рећи да је најбитнији елемент интерактивног процеса одговарајући едукативни софтвер. Образовни софтвер се данас не може замислити без интеракције. Поред интерактивности, софтвер који се користи у образовању требало би да поседује још једну важну особину која може допринети ефективнијем преносу знања, а то је могућност визуелизације предмета проучавања. Визуализација и интерактивност представљају неопходне и очекиване компоненте када се говори о софтверу који се примењује у образовању.

3.1.2 Примена визуелизације у настави

У литератури се може наћи већи број експлицитних дефиниција визуелизације, а углавном се овај термин користи за описивање когнитивне активности имагинације визуелне репрезентације. Приликом дефинисања термина визуелизације, Phillips, Norris и Macnab (2010) издвајају три различита концепта:

- *Објекти визуелизације* су физички предмети које особа посматра и интерпретира ради разумевања нечег другог, а не датог предмета. Остали сензорни подаци попут звука могу бити саставни делови ових предмета и предмети се могу појавити на многим медијима, као што су папир, екран рачунара, слајдови;

- *Интроспективна визуелизација* је „имагинативна конструкција неког могућег визуелног искуства“ у одсуству објекта визуелизације. Интроспективна визуелизација фокусира се на „менталне објекте пројектоване у свести човека“;

- *Интерпретативна визуелизација* односи се на интерпретацију значења објеката визуелизације или интроспективне визуелизације у односу на „постојећа уверења, искустава и разумевања“.

Идеје у научном и инжењерском свету најчешће произилазе из визуелних и специфичних ситуација. Перцепција стварности је у основи визуелна и у многим случајевима човек користи симболичку обраду, визуелне дијаграме и друге облике имагинативних процеса како би стекао интуицију онога што у свом формалном аспекту има апстрактну структуру (DÍaz, Dormido and Rivera, 2015).

Костић (2017) у својој дисертацији наводи да когнитивно-визуелни приступ представља савремену дидактичко-методичку концепцију која промовише визуелизацију процеса учења у мултирепрезентативном окружењу. Такође, наводи да је за оптимално коришћење потенцијала визуелног мишљења студената потребно подстицајно визуелно окружење за учење, као и да у реализацији когнитивно-визуелног приступа посебно важну улогу имају визуелизовани задаци.

У последњој деценији, убрзаном прогресијом рачунарских капацитета и напретком технологија графичког дизајна, мултимедијална окружења за учење еволуирала су од секвенцијалног статичног оквира за текст и слике до високо софистициране визуелизације (Bétrancourt, 2005). Информационе технологије и посебно дизајнирани софтверски производи отварају нове могућности у развијању потребних вештина и способности студената (перцептивне, менталне, вербалне, практичне…) (Makarova, 2016). Дакле, може се рећи да визуелизација покреће машту студената и продубљује њихово разумевање одређених садржаја.

Добра визуализација није само статична слика или тродимензионално виртуелно окружење. Ware (2019) сматра да је добра визуелизација нешто што посматрачу омогућава да детаљно анализира и пронађе више података о било чему што се чини важним. У идеалном случају, сваки објект података на екрану ће бити активан, а не само мрља у боји. Биће способан да по потреби приказује више информација, нестаје кад није потребно и прихвата корисничке наредбе за помоћ у процесима размишљања. Интерактивна визуелизација је процес који се састоји од великог броја међусобно повезаних

петљи повратних информација. Графички приказ процеса визуализације података (Ware, 2019) обухвата четири основне фазе, комбиноване у већем броју повратних петљи (слика 31).

Слика 31. Процес визуелизације по Ware (2019)

3.1.3 Образовни софтвер

Удружење за образовне комуникације и технологију (*Association for Educational Communications and Technology – AECT*), највеће професионално друштво фокусирано искључиво на образовну технологију, под појмом „образовна технологија“ подразумева студију и етичку праксу која се односи на креирање и коришћење технолошких процеса и ресурса у циљу побољшања перформанси и олакшавања процеса учења (Rasmussen and Salkind, 2008). У пракси образовна технологија односи се на више области. Једна је планирање и инсталирање технологије за образовање која се од средине 1970-их углавном односи на рачунаре за подучавање и управљање учењем. Друга област на коју се фокусира је образовни софтвер, док је трећа усмерена на процесе и продукте који се користе за дизајнирање наставе. Ово се заправо односи на процес планирања, конструисања и развоја наставе, организовања

и контроле учења и примену когнитивне психологије на дизајн наставе, а не на физички хардвер или врсту софтвера. Ове активности су познате као инструкциони дизајн.

Образовним софтвером се генерално може назвати било који софтвер који се може користити у наставном процесу за подучавање и учење. Прецизнија дефиниција образовног софтвера се среће у (Nadrljanski, 2000) где је представљен као софтвер који обухвата програмске језике и помоћне алате, одређену организацију наставе и учења, и који се заснива на логици и педагогији. Овакав софтвер треба бити дидактички осмишљен тако да омогући студентима постепено напредовање у складу са њиховим могућностима (Odadžić, V., 2016). Образовни софтвер се данас не може замислити без интерактивних могућности које обезбеђују тренутно исправљање грешака, могућности враћања, понављања или задржавања на појединим елементима наставног градива, као и утврђивање стеченог знања.

Дизајн образовног софтвера подразумева имагинацију, идеју, елаборацију и дескрипцију рачунарског система у односу на неке педагошке циљеве и различита образовна ограничења која треба узети у обзир у вези са окружењем у којем ће се софтвер користити (Tchounikine, 2011). Ова техничка фаза програмирања може се извести од нуле или надоградити на постојеће софтверске компоненте.

Ако су образовни софтвери добро дизајнирани, обезбеђује се максимална индивидуализација наставног рада (Popov and Jukić, 2006). Радосав (2005) сматра да при дизајну образовног софтвера треба водити рачуна о следећим фазама:

- избор садржаја који ће се реализовати на рачунару,
- прикупљање потребне литературе и материјала у писаном и електронском облику,
- обрада материјала и дизајнирање,
- процес програмирања,

- провера образовног софтвера - тестирање,
- израда програмске документације,
- евалуација програма.

Постоје различите врсте образовног софтвера, а класификација се може извршити на основу различитих критеријума. Класификација на основу дидактичко-методичких критеријума може се наћи у (Nadrljanski, Nadrljanski and Soleša, 2008). Аутори по датом основу разликују више типова образовног софтвера где значајно место заузима *образовни софтвер симулација.* Овај тип софтвера омогућава да се неки реални или апстрактни теоријски системи представе помоћу модела на рачунару и да се симулирају процеси тих система. Помоћу симулације могу се сагледати структурне и функционалне карактеристике проучаваних система. Taher и Khan (2015) сматрају да је симулација изузетно моћно наставно средство када се користи заједно са традиционалном наставом. Образовни софтвери за симулацију и визуелизацију теоријских система управо су предмет истраживања овог рада.

3.2 УПОТРЕБА СОФТВЕРСКИХ СИСТЕМА У ОБРАЗОВАЊУ

Бројне студије показују да интерактивни софтверски алати за помоћ у учењу могу подстаћи мотивацију студената (Chesnevar, González and Maguitman, 2004; Chakraborty, Saxena and Katti, 2011). Да би се премостио јаз између теорије и њене практичне научне примене, Goyal и Sachdeva (2009) препоручују ефикасну стратегију за „оживљавање“ теоријских концепата која укључује коришћење софтверских алата за визуелизацију и интеракцију. Hamada и Hassan (2017) за унапређење процеса учења предлажу интерактивне алате, то јест рачунарска окружења која интегришу различите групе софтверских модула, омогућавајући студентима да на једноставан и експресан начин доживе теме учења. Образовни софтвери треба да омогуће мултимедијални приступ наставним садржајима, као и њихово лакше и успешније разумевање и усвајање. Процес наставе и учења биће ефикаснији и интересантнији подстицањем студената да буду проактивнији и креативнији

на часу. Интерактивни час подразумева веће ангажовање студената тако што се уз помоћ одговарајућих медија за учење, најчешће софтвера за симулацију или система за визуелну репрезентацију алгоритама, реализује као двосмерна комуникација наставника и студената. На овај начин визуелизација садржаја уџбеника привлачи интересовање студената тако да могу брже и лакше да савладају градиво и касније примене оно што су научили (Wang et al., 2012; Dewi et al., 2018).

Постоји више истраживања која се баве проучавањем система за визуелизацију алгоритама, опширнији преглед може се наћи у (Shaffer et al., 2010; Fouh et al., 2012). Велики број аутора у својим истраживачким радовима презентују резултате који потврђују хипотезу да се употребом оваквих система повећава ангажовање студената и позитивно утиче на ефикасност учења (Naps et al., 2002; Hundhausen et al., 2002; Urquiza-Fuentes et al., 2009). Hansen и сарадници (2002) сматрају да су у неким случајевима резултати коришћења анимације за подучавање понашања алгоритма били незадовољавајући, не због грешке у анимацији као алату за учење, већ због приступа који се користио за преношење анимација. Сматрају да је визуелизација заиста моћно средство за ефикасно преношење динамичког понашања алгоритама, али да је потребно добро дизајнирати алгоритамске анимације уз примену одговарајућих педагошких норми. Ослањајући се на истраживања о когнитивној науци и интеракцији човека и рачунара, развили су архитектуру за мултимедијалне презентације алгоритама HalVis и показали да је степен учења код студената који су користили HalVis био значајно већи него код оних који су користили средства традиционалне наставе. Velazquez-Iturbide и сарадници (2013) доказују хипотезу да примена експлицитних педагошких циљева повећава квалитет и ефективност система визуелизације, па се самим тим са становишта програмера и захтеви за визуелизацију и интеракцију могу лакше идентификовати. Из експицитних циљева произилазе и имплицитни који проширују спектар задатака које подржава систем визуелизације.

Симулатори као образовни алати за помоћ у учењу привукли су значајну пажњу, што је доказано бројним научним радовима (De Jong et al., 1998; Lindgren et al., 2009) који говоре о позитивним аспектима њихове употребе. Rutten са сарадницима (2012) разматра велики број експерименталних истраживања о ефектима софтверских симулација у настави која су објављена у временском периоду од једне деценије. Велики број студија које су разматране у том раду упоређују услове рада са и без симулација, показујући позитивне резултате у корист употребе симулација за побољшање традиционалних начина учења. Симулације нуде студентима прилику да прате последице својих избора путем графичке демонстрације апстрактних концепата, при чему се од њих може тражити да користе математичке вештине, вештине опсервације и вођења белешки (Bozeman, 1999). Дакле, софтверски алати за симулацију омогућавају студентима експериментисање са теоријским конструкцијама, дајући им тренутне информације о резултатима експеримента.

Већ је добро познато да се и игре могу користити у образовне сврхе. Pivec и сарадници (2004) описују учење засновано на игрицама као иновативну образовну парадигму која се сматра погодном за дату сврху. Релевантни аутори за ту област објавили су одличне чланке у којима описују појам „играња“ као природни начин на који људска бића уче, доказујући како се игре могу сматрати “озбиљним” дидактичким ресурсима за било коју фазу – контекст учења (Prensky, 2001; Gee, 2004).

Последњих година софтверски алати за помоћ у учењу постали су присутни у свим фазама образовања. Образовни софтверски системи данас су доступни скоро за сваку област науке. Велики број симулационих система користи се у области електронских наука (Zhang and Jie, 2018), обновљивих извора енергије (Witzig et al., 2016), микробиологије (Huppert, Lomask and Lazarowitz, 2002), генетике (Soderberg and Price, 2003), хирургије (Harders, 2005). Софтверски системи успешно се користе и за учење математичких алгоритама (Wolfram, 2003), теорије вероватноће (Koparan and Yilmaz, 2015),

статистике (Konold and Miller, 2004), теорије графова (Adar, 2006). Такође, системи за помоћ у учењу пронашли су примену и у области музичке уметности, на пример за свирање вилолине (Yin, Wang and Hsu, 2005), клавира (Oshima, Nishimoto and Suzuki, 2004), као и компоновање музике (Farbood, Pasztor and Jennings, 2004).

Образовни софтверски системи се убрзано развијају, нарочито у области рачунарског инжењерства (Branovic et al., 2014). Јовановић и сарадници (2012; 2016; 2018) представљају симулационе системе намењене проучавању рачунарских мрежа. Едукативни компјутерски систем са веб базираним симулатором, дизајниран да помогне у настави и учењу рачунарске архитектуре, представили су Ђорђевић и сарадници (2005). Визуелни симулатори су такође нашли примену за помоћ у учењу алгоритама за заштиту података (Thakur et al., 2011; Stanisavljević et al., 2014). Детаљна анализа већег броја система за визуелизацију алгоритама вештачке интелигенције представљена је у (Stamenković et al., 2023).

Закључак је да се образовни софтверски системи користе у готово свим научним областима где бележе позитивне резултате у погледу унапређења традиционалне наставе. Бројни аутори их предлажу и за учење тема из области програмских преводилаца како би помогли да се „оживе“ многе апстрактне теоријске конструкције (Naveed and Sarim, 2018; Chesnevar, Cobo and Yurcik, 2003).

4. АНАЛИЗА СОФТВЕРСКИХ СИСТЕМА ПОГОДНИХ ЗА УЧЕЊЕ ПРОГРАМСКИХ ПРЕВОДИЛАЦА

У овом поглављу презентован је већи број образовних система за помоћ у учењу програмских преводилаца различитих аутора, анализиране су њихове карактеристике и функционалности. Сваки од њих је специфичан на свој начин, у погледу могућности које нуди, тема које обрађује, дидактичких метода за успостављање интеракције са студентом, приступа за постављање и решавање проблема. На крају поглавља успостављају су критеријуми за евалуацију на основу којих је извршена процена квалитета ових система.

4.1 ПРЕГЛЕД ПОСТОЈЕЋИХ СОФТВЕРСКИХ СИСТЕМА

Због обима и комплексности тема у оквиру предмета Програмски преводиоци и на основу анализе позитивних искустава у коришћењу иновативне едукативне технологије (у претходном поглављу), може се закључити да је за унапређење наставе пожељно искористити потенцијал таквих технолошких решења. Прави потенцијал технологије у високом образовању, када се правилно користи, је да студентима прошири могућности учења и истраживања. Због тога су у овом поглављу приказана различита едукативна софтверска решења у виду помоћних наставних алата који се могу користити за учење тема из области програмских преводилаца. Претрагом литературе може се пронаћи већи број таквих алата развијених на различитим универзитетима широм света. За анализу је одабрано двадесет симулатора и алата за визуелизацију, неки од њих приказани су у прегледним радовима (Stamenković and Jovanović, 2019; Stamenković, Jovanović and Chakraborty, 2020). Одабрани софтверски системи приказани су у табели 1.

Табела 1 приказује основне податке о софтверским алатима. У првој колони дати су називи тих система, у другој се наводе универзитети на којима су развијени, трећа приказује државу из које потичу, четврта колона означава

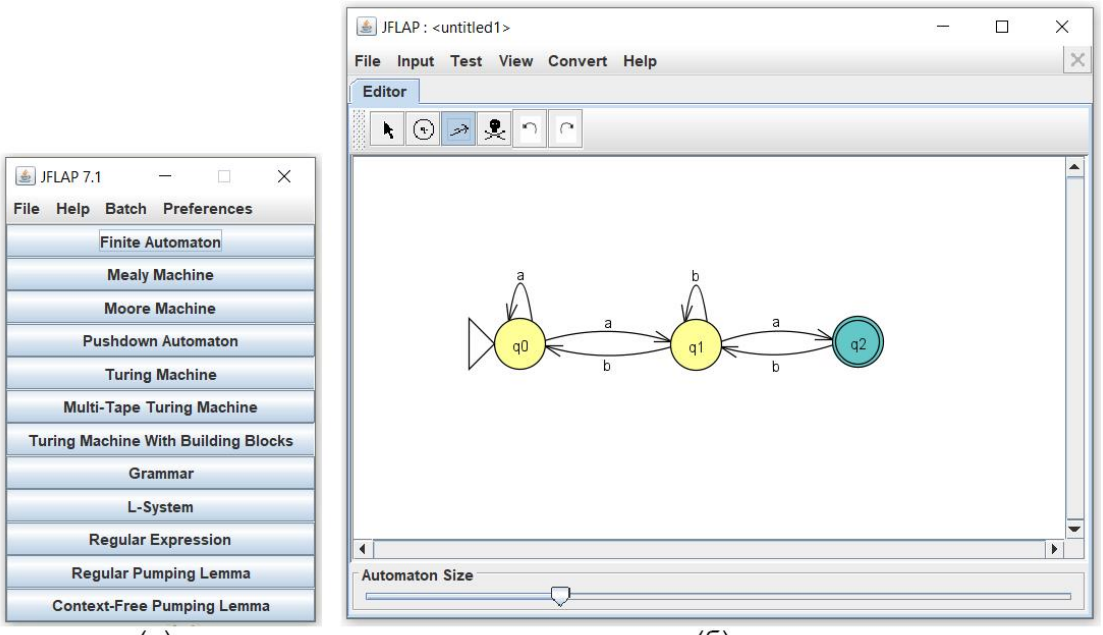
доступност алата у време писања дисертације (веб адресе на којима се могу пронаћи). Сви анализирани симулатори су у изворној литератури наведени као бесплатни алати, али за већину није наведена веб адреса за преузимање. Детаљном претрагом пронађено је тринаест важећих адреса, три симулатора су у изворној литератури имала наведене адресе које у време писања рада нису биле доступне, док четири симулатора нису имала наведену адресу, а ни претрагом веба нису пронађена.

Табела 1. Опште информације о одабраним софтверским системима погодним за учење програмских преводилаца

Софтверски систем	**Универзитет**	**Земља**	**Веб адреса / линк за преузимање**
JFLAP	Duke University	САД	http://www.jflap.org/
VAST	Rey Juan Carlos University	Шпанија	*http://www.lite.etsii.urjc.es/vast/
Automata Simulator	Netaji Subhas University of Technology	Индија	https://play.google.com/store/apps/details?id=com.singh.tuhina.automatasimulationcopy
Language Emulator	Federal University of Minas Gerais	Бразил	https://homepages.dcc.ufmg.br/~lfvieira/ftc.html
CMSimulator	Slovak University of Technology	Словачка	https://play.google.com/store/appa/details?id=fiitstu.gulis.cmsimulatcm
Seshat	University of Burgos	Шпанија	http://cgosorio.es/Seshat/
Robot-based Automata Simulator	University of Aizu	Јапан	Непозната
L-FLAT	University of Beira Interior	Португалија	https://code.google.com/archive/p/lflat/
DFA simulation tool	Vishwakarma Institute of Technology	Индија	Непозната
jFAST	Villanova University	САД	https://sourceforge.net/projects/jfast-fsm-sim/
RegExpert	Sveučilište u Zagrebu	Хрватска	Непозната
SOTA	Rey Juan Carlos University	Шпанија	*http://www.escet.urjc.es/~jurquiza/research-iticse.html#sota
LLparse & LRparse	Rensselaer Polytechnic Institute	САД	https://users.cs.duke.edu/~rodger/tools/
BURGRAM	University of Burgos	Шпанија	http://cgosorio.es/BURGRAM/
PAVT	Netaji Subhas University of Technology	Индија	https://sourceforge.net/projects/pavt/
JCT	SUNY Institute of Technology	САД	*http://turing.sunyit.edu/JCT
Webworks applets	Montana State University	САД	https://www.cs.montana.edu/webworks/
SELFA-Pro	University of Castilla-La Mancha	Шпанија	https://portal.esi.uclm.es/selfa/
LISA	Univerza v Mariboru	Словенија	https://labraj.feri.um.si/lisa/index.html
FADL toolkit	Jawaharlal Nehru University	Индија	Непозната

*веб адреса је наведена у литератури али у време писања дисертације није била доступна

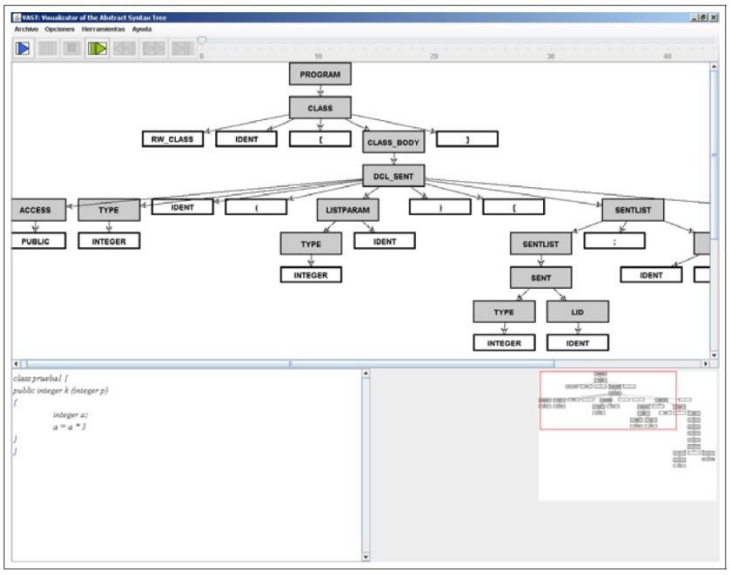
JFLAP (Java Formal Languages and Automata Package) је интерактивни графички симулациони алат написан на Јави и намењен учењу формалних језика и теорије аутома (Procopiuc, Procopiuc and Rodger, 1996; Rodger and Finley, 2006; Rodger et al., 2009). Овај систем се развија од 1990. године на Duke универзитету у Северној Каролини, а последње стабилно издање је 7.1 које је објављено 2018. године. У главном менију JFLAP-а (слика 32а) понуђене су основне функције које се односе на дефинисање различитих типова аутомата и граматике. Конструисање аутомата се врши помоћу графичког едитора за цртање (слика 32б) који омогућава рад са неколико различитих врста аутомата као што су: NFA, DFA, Милијева и Мурова машина, потисни аутомати, Тјурингова машина. Дефинисање језика врши се коришћењем регуларних израза или бесконтексних граматика. Поред основних функција, JFLAP обезбеђује и друге функционалности, на пример конверзију NFA у DFA, минимизацију аутомата, трансформацију аутомата у одговарајуће регуларне изразе и обрнуто. Резултати спроведене студије о употреби JFLAP-а у настави на више различитих универзитета (Rodger et al., 2009) показују позитивне ефекте у погледу ангажовања студената на часу и лакшег савладавања концепата курса, што је резултирало и бољим оценама на испиту.



(а) (б)

Слика 32. (а) Главни мени JFLAP-а и (б) графички едитор за цртање аутомата

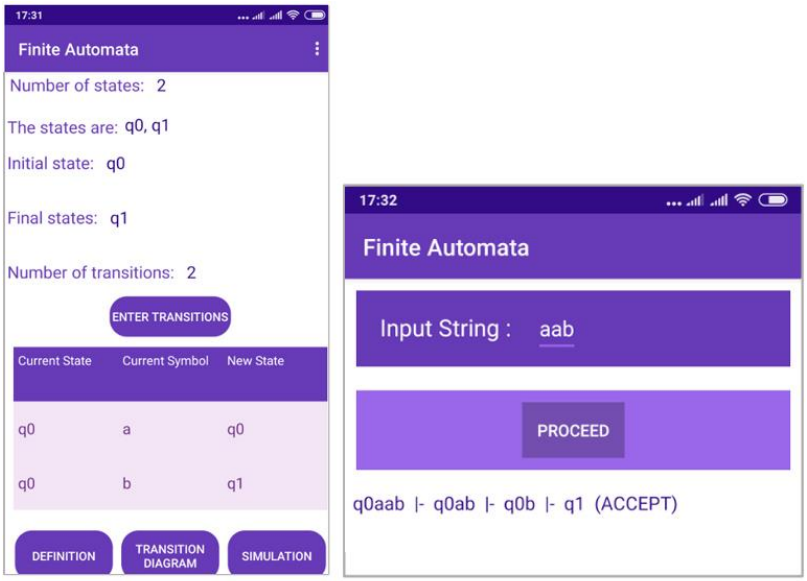
VAST је едукативни софтвер за визуелизацију синтаксног стабла развијен на Универзитету Rey Juan Carlos у Шпанији (Almeida-Martınez, Urquiza-Fuentes and Velázquez-Iturbide, 2009; Almeida-Martínez and Urquiza-Fuentes, 2009). У питању је десктоп апликација написана на Јава програмском језику. VAST чине два независна модула VASTapi и VASTview. VASTapi је задужен за конструкцију синтаксног стабла и као излаз производи XML датотеку. VASTview интерпретира XML фајл и визуелизује процес формирања синтаксног стабла. Анимација овог процеса се одвија у више међуфаза које контролише студент (слика 33). Резултати спроведене педагошке евалуације су позитивни, али је генерализација ових резултата ограничена због мале групе студената који су учествовали у експерименту.



Слика 33. VAST: праћење процеса конструкције синтаксног стабла

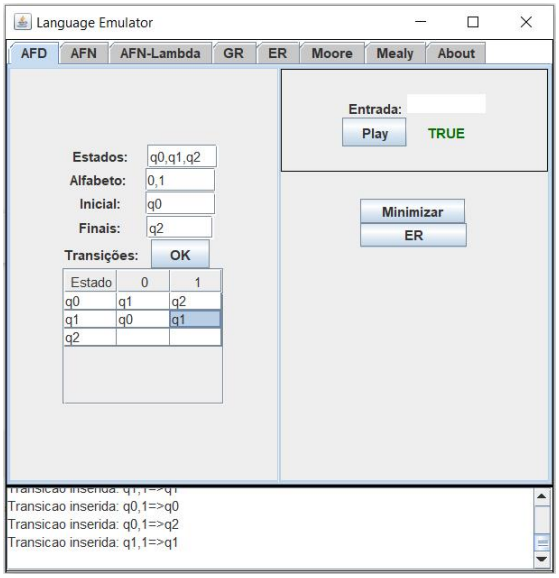
Automata Simulator је едукативни алат имплементиран као мобилна апликација како би студентима био приступачнији (Singh et al., 2019). Апликација је намењена студентима за пројектовање и симулацију аутомата током предавања. Студенти могу да конструишу и симулирају NFA, DFA, Мурову и Милијеву машину, потисне аутомате и Тјурингову машину. Дефинисање одабраног типа аутомата се врши помоћу табеле транзиција (слика 34а). Након конструисања аутомата могуће је покренути симулацију понашања аутомата за задати улазни стринг (слика 34б). Ефикасност овог

алата тестирана је у току једног семестра, а анализа резултата испита је показала да је овакав приступ помогао студентима да боље науче теоријске појмове и добију већи проценат бодова.



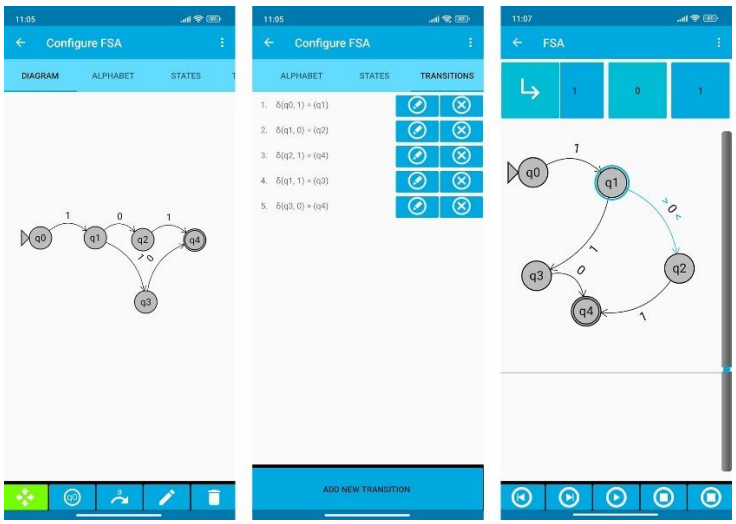
Слика 34. (а) Дефинисање коначног аутомата и (б) симулација рада аутомата у Automata Simulator-у

Language emulator је развијен на Универзитету Minas Gerais у Бразилу као помоћни алат за учење теорије аутомата и формалних језика на основним студијама (Vieira et al., 2004). У питању је десктоп апликација која је реализована на Јава програмском језику. Састоји се од седам независних интерфејса раздвојених табовима. Алат подржава рад са коначним аутоматима (DFA, NFA, NFA са епсилон прелазима), регуларним изразима и граматикама, Муровом и Милијевом машином. Додатне функционалности које су на располагању студентима односе се на минимизацију DFA, генерисање регуларног израза који одговара задатом DFA, трансформацију Мурове у Милијеву машину и обрнуто. Аутомати се дефинишу уносом функције прелаза у одговарајућим текстуалним пољима (слика 35). Евалуација алата је спроведена анкетом студената који су имали прилику да користе ову апликацију током учења, а скоро 95% одговорило је да им је употреба Language emulator-а помогла у процесу учења теорије аутомата.



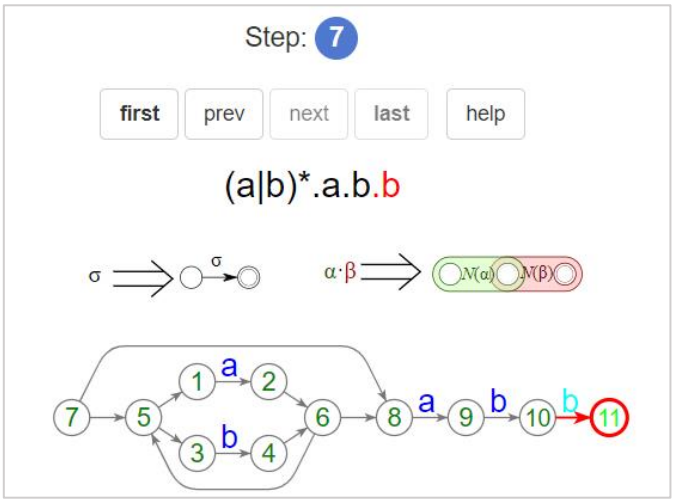
Слика 35. Дефинисање DFA у симулатору Language emulator

CMSimulator представља Андроид апликацију за симулацију рада аутомата (Chuda, Trizna and Kratky, 2015). Мобилна апликација укључује више екрана са логички подељеном функционалношћу: екран главног менија, екран симулације и екран за конфигурацију аутомата. Екран интерактивне симулације пружа кориснику преглед улазне траке, прелазних функција, дијаграм стања и укључује контролну дугмад за кретање кроз процес симулације (слика 36). На екрану за уређивање конфигурације аутомата корисник дефинише функцију транзиције. CMSimulator подржава четири типа аутомата: коначне аутомате, потисне аутомате, линеарно ограничене аутомате и Тјурингову машину.



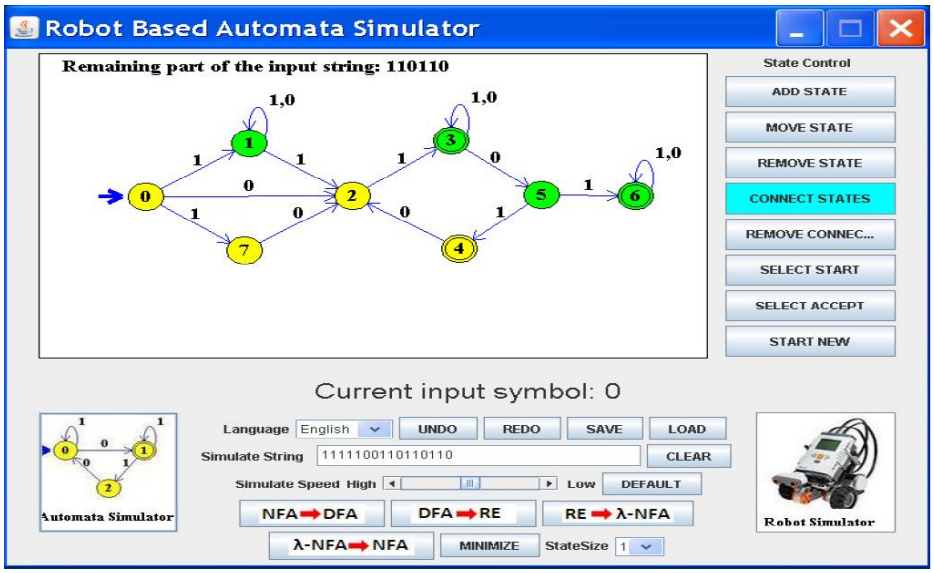
Слика 36. Конфигурација и симулација аутомата у систему CMSimulator

Seshat је веб базирани образовни алат који студентима пружа могућност интеракције и експериментирања са најчешћим алгоритмима лексичке анализе и теорије аутомата (Arnaiz-González et al., 2018). Систем обезбеђује визуелизацију четири алгоритма: конструисање NFA на основу задатог регуларног израза (слика 37), конверзија NFA у DFA, претварање регуларног израза у DFA и минимизацију аутомата. Seshat симулира рад одабраног алгоритма корак по корак, пружајући одговарајућа објашњења на сваком кораку. Процена употребљивости овог алата тестирана је квантитативном експериментом који је показао да су студенти уз овај едукативни алат постигли боље резултате на тесту у односу на студенте који су учили без помоћи алата.



Слика 37. Seshat: трансформација регуларног израза у NFA

Robot-based Automata Simulator представља симулациони систем за учење теорије аутомата, развијен комбинацијом Јаве и роботских технологија (Hamada and Sato, 2011; Hamada and Sato, 2012; Hamada, 2013). Симулатор може комуницирати са роботом који је дизајниран за ову сврху коришћењем Lego NXT сета. Комуникација робота и симулатора остварује се између Јава окружења и LeJOS софтвера. Студенти конструишу аутомат у оквиру графичког интерфејса система (слика 38), а затим га прослеђују роботу који може симулирати транзиције аутомата. Посматрањем кретања робота врши се учење теорије аутомата кроз игру.



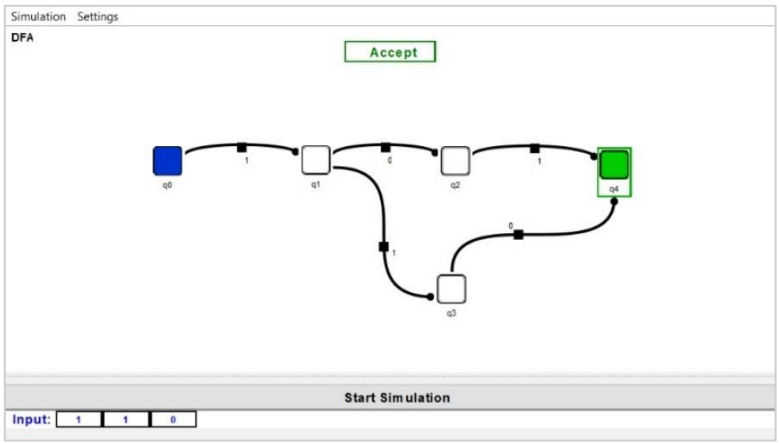
Слика 38. Robot-based Automata Simulator – изглед корисничког интерфејса (Hamada, 2013)

L-FLAT (Logtalk Toolkit for Formal Languages and Automata Theory) представља симулациони систем који имплементира алгоритме погодне за учење формалних језика и теорије аутомата (Moura and Dias, 2011). Овај симулатор укључује алгоритме за препознавање и генерисање речи, конвертовање различитих врста аутомата, испитивање различитих аспеката аутомата, минимизацију дефиниција аутомата. Може се користити за учење регуларних израза, потисних аутомата, NFA, DFA, бесконтексне граматике и Тјурингове машине. L-FLAT је имплементиран у Logtalk, објектно оријентисан логички програмски језик. Дизајниран је као образовни алат са интерфејсом командне линије.

DFA simulation tool је веб алат за визуелизацију DFA, засновану на JavaScript-у (Vayadande et al., 2022). Омогућава конструисање, отклањање грешака и тестирање DFA. Развијен је са циљем да помогне учењу теорије аутомата. Аутомати се конструишу помоћу графичког едитора, а тестирање се своди на испис информација о томе да ли је задати стринг прихваћен или не.

jFAST (Java Finite Automata Simulation Tool) је графички софтверски алат дизајниран за помоћ у учењу уводних тема из области теорије аутомата (White and Way, 2006). За имплементацију компоненти корисничког интерфејса

коришћена је Java библиотека Swing. Алат омогућава визуелни дизајн, истраживање и симулацију неколико врста аутомата (слика 39). Подржани су коначни аутомати (NFA, DFA), Тјурингова машина, потисни аутомат. Једноставан је за употребу и посебно намењен за мање напредне студенте. Евалуација је извршена проценом ефикасности од стране предавача и проценом употребљивости анкетирањем студената који су користили алат током учења. Опште повратне информације показују да је коришћење jFAST-а у комбинацији са теоријом значајно побољшало њихово разумевање.

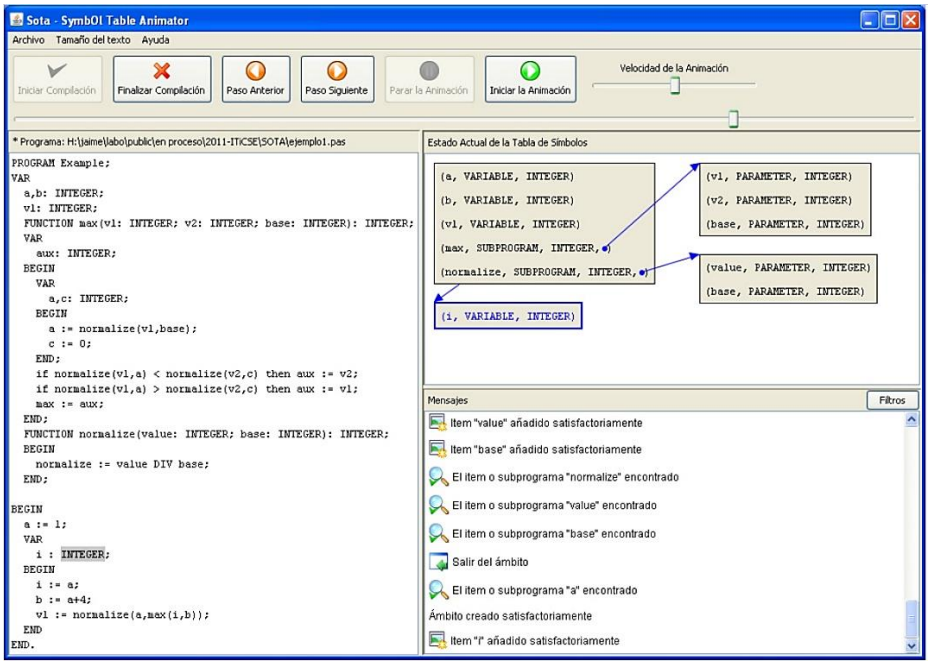


Слика 39. Симулациони интерфејс jFAST-а

RegExpert је образовни систем за интерактивно учење регуларних језика и граматика (Budiselic, Srbljic and Popovic, 2007). Развијен је као десктоп апликација за приказ конверзије регуларних израза у одговарајуће NFA са епсилон прелазима. Студенти имају могућност да сами задају регуларни израз или изаберу опцију за његово аутоматско генерисање. Формат и сложеност аутоматски генерисаних регуларних израза могу се контролисати помоћу посебног конфигурационог панела. Алат студентима користи за проверу решења задатака које раде на папиру, а предавачима помаже у припреми задатака.

SOTA (SymbOl Table Animation) је алат за визуелизацију табеле симбола намењен за употребу на курсу компајлера (Urquiza-Fuentes et al., 2011; Urquiza-Fuentes and Manso, 2011). Главни циљ алата је визуализација тренутног стања табеле симбола и операције које се на њој врше током анализе изворног кода.

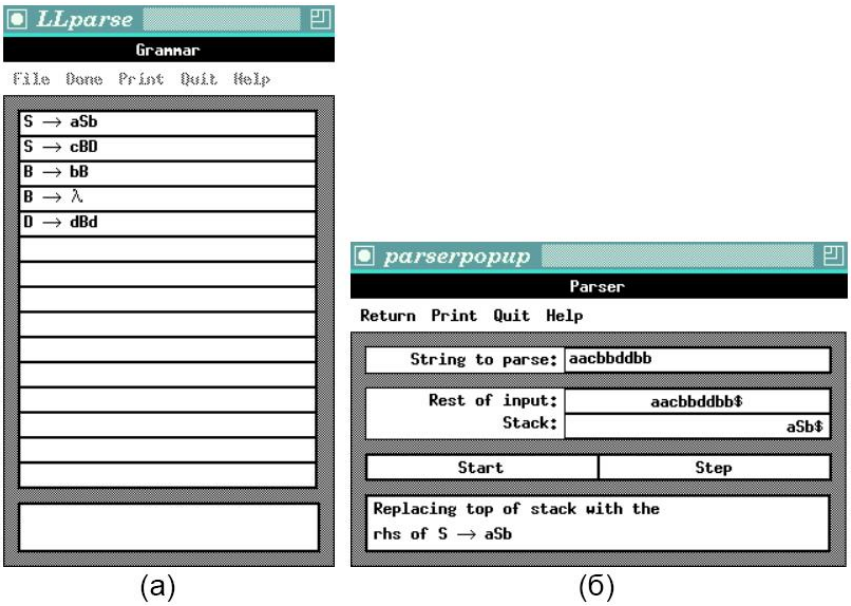
Интерфејс алата (слика 40) подељен је на три дела: програмски део, део за приказ тренутног стања и део за поруке. На левој страни интерфејса се налази програмски део који приказује изворни код програма који се анализира. Део за приказ тренутног стања у горњој десној зони интерфејса, пружа графички приказ тренутног стања табеле симбола и тренутне операције (убацивање идентификатора, креирање опсега) које се извршавају. Део за поруке у доњој десној зони интерфејса приказује кратке текстуалне описе свих операција које се изводе на табели симбола. Једна од спроведених евалуација овог алата подразумевала је оцењивање стеченог знања студената о процесу конструкције табеле симбола и о парсеру који врши конструкцију табеле симбола током процеса компајлирања. Студенти који су користили алат постигли су одличне резултате на тесту и надмашили студенте који су пратили стандардну методологију наставе за чак 22%.



Слика 40. Кориснички интерфејс образовног алата SOTA

LLparse и **LRparse** представљају два едукативна алата који студентима пружају визуелне инструкције процеса конструисања LL(1) и LR(1) табела (Blythe, James and Rodger, 1994). Ови алати су развијени на програмском језику C++. Оба алата се састоје од низа прозора који представљају кораке у изради табеле парсирања. У почетном прозору (слика 41а) студент треба да унесе

LL(1), односно LR(1) граматику која садржи највише петнаест правила. Након уноса граматике следе прозори у оквиру којих се израчунавају FIRST и FOLLOW скупови, а затим и прозор са табелом парсирања. Прозор за визуелизацију парсирања дозвољава кориснику да унесе улазни стринг и започне анимацију процеса парсирање корак по корак (слика 41б).

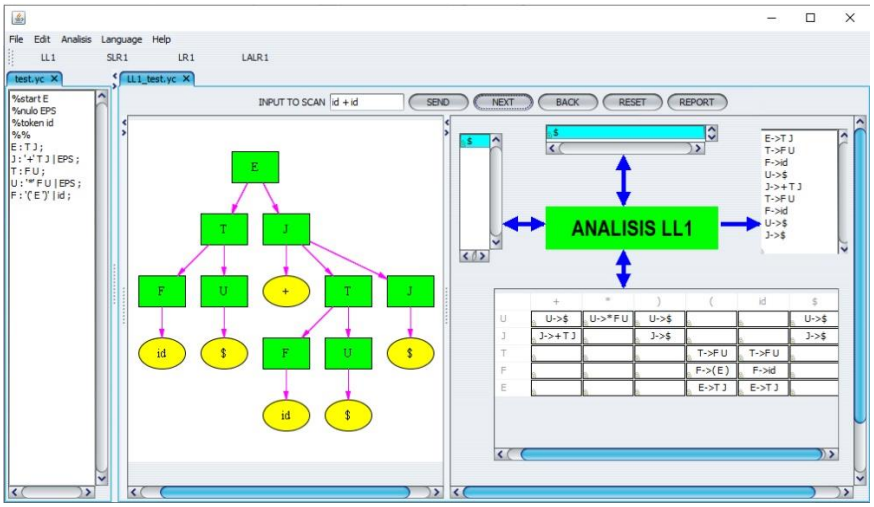


Слика 41. LLparse алат: (а) прозор за унос граматике и (б) пример једног корака у процесу парсирања (преузето са сајта Duke University[2])

BURGRAM је симулациони систем развијен са циљем да олакша наставу програмских преводиоца у фази синтаксне анализе (García-Osorio, Gómez-Palacios and García-Pedrajas, 2008). Симулира top-down и bottom-up алгоритме анализе: LL, SLR, LALR и LR. Током симулације за дефинисану граматику и задати улазни низ, студент може пратити корак по корак поступака парсирања уз креирање синтаксне табеле и синтаксног стабла (слика 42). Занимљива функционалност алата је генерисање извештаја целокупног

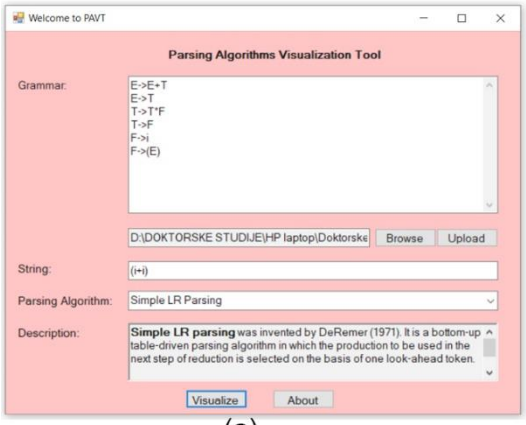
[2] https://users.cs.duke.edu/~rodger/tools/LLparse.html

процеса анализе у PDF формату. BURGRAM је развијен у Јави, за приказ процеса формирања синтаксног стабла коришћена је библиотека GRAPPA, а за генерисање извештаја библиотека iText.



Слика 42. Симулација процеса парсирања у BURGRAM-у

PAVT (Parsing Algorithm Visualizer Tool) је образовни алат који студентима помаже у учењу синтаксне анализе тако што визуелизује алгоритме парсирања (Sangal et al., 2018). Путем једноставаног корисничког интерфејса (слика 43а) студент уноси бесконтексну граматику или је учитава из текстуалне датотеке, уноси стринг који ће бити парсиран и бира један од шест понуђених алгоритама. PAVT визуелизује поступак парсирања за дати пар граматике и стринга и чува га у текстуалној датотеци (слика 43б).



(а)

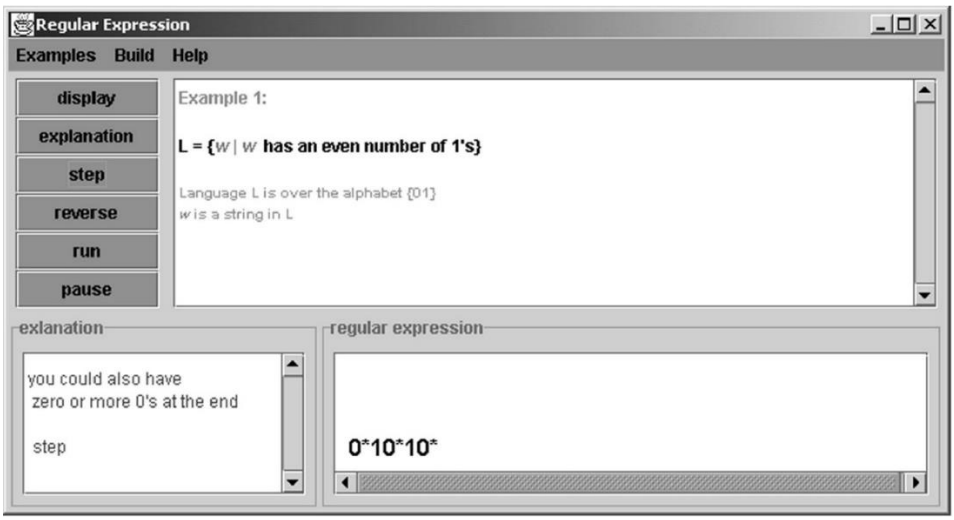
```
-------------------------TABLE DRIVEN PARSING-------------------------

STACK                    INPUT                  ACTION     PRODUCTION
0                        (i+i)$                   s5
0 5                      i+i)$                    s4
0 5 4                    +i)$                     r5       F->i
0 5 3                    +i)$                     r4       T->F
0 5 2                    +i)$                     r2       E->T
0 5 8                    +i)$                     s6
0 5 8 6                  i)$                      s4
0 5 8 6 4                )$                       r5       F->i
0 5 8 6 3                )$                       r4       T->F
0 5 8 6 9                )$                       r1       E->E+T
0 5 8                    )$                      s11
0 5 8 11                 $                        r6       F->(E)
0 3                      $                        r4       T->F
0 2                      $                        r2       E->T
0 1                      $                        acc      Accept
```

(б)

Слика 43. PAVT алат: (а) кориснички интерфејс и
(б) део резултата парсирања који се чува у текстуалној датотеци

JCT (Java Computability Toolkit) садржи два веб базирана окружења за конструкцију и симулацију аутомата (Robinson et al., 1999). Једно окружење се користи за учење теорије коначних аутомата док је друго намењено учењу Тјурингове машине. Окружење за учење коначних аутомата, поред симулације рада аутомата, омогућава и конверзију NFA у DFA, као и минимизацију DFA. JCT је добио позитивне повратне оцене од стране студената на факултету SUNY Institute of Technology где је интегрисан у оквиру курса основе рачунарства.

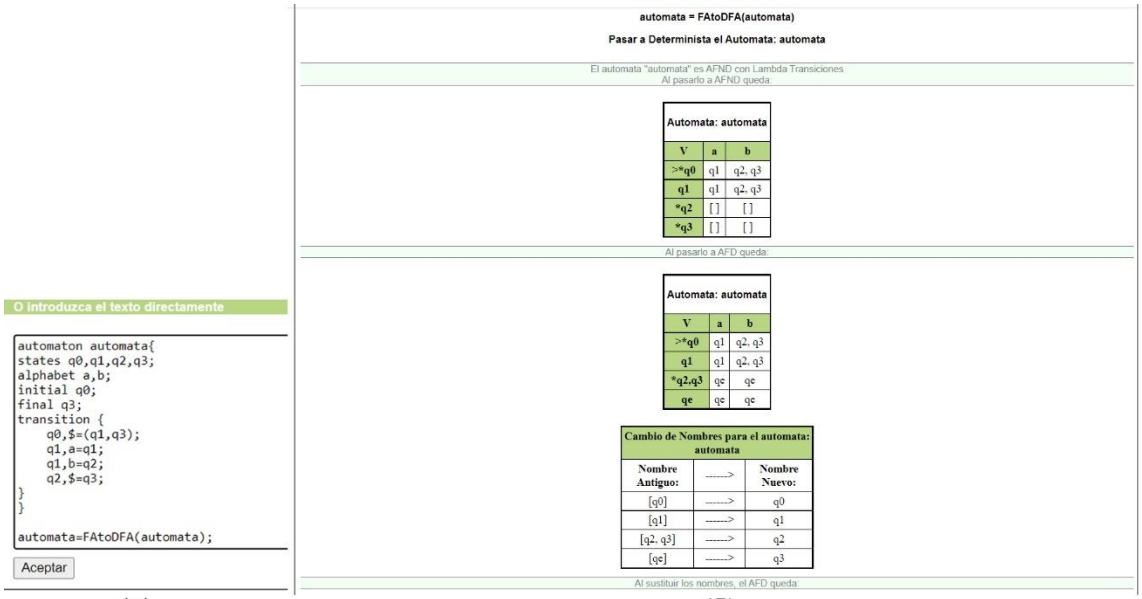
Webworks applets представљају решење лабораторије Webworks Универзитета Montana за активно учење тема из области формалних језика и аутомата (Grinder et al., 2002). У питању су три аплета и то: аплет за учење коначних аутомата, за учење регуларних израза и аплет за учење регуларне граматике. На слици 44 приказан је аплет за рад са регуларним изразима. У овим аплетима студентима се поред симулације нуде и одговарајуће демонстрације и вежбе. Симулатор има могућност претварања NFA у еквивалентне DFA, конверзију DFA у њихове минималне облике, конверзију регуларних израза или регуларних граматика у NFA и обрнуто.



Слика 44. Webworks аплет за рад са регуларним изразима (Grinder et al., 2002)

SELFA-Pro (SoftwarE for Learning Formal languages and Automata theory - Pro) је образовни софтверски систем развијен са циљем да студентима олакша

учење главних концепата теорије аутомата и формалних језика (Castro-Schez et al., 2009). Главне функционалности овог веб алата могу се сврстати у четири групе: коначни аутомати, регуларне граматике, потисни аутомати и регуларни изрази. SELFA-Pro омогућава конверзију коначних аутомата у DFA, минимизацију аутомата, претварање аутомата у регуларне изразе и обрнуто. Аутомати и граматике се задају у текстуалном облику у одговарајућем пољу (слика 45а). Пример приказа поступка конверзије коначног аутомата у DFA може се видети на слици 45б. Испитивање ефикасности алата обухватило је две академске године и подразумевало је праћење групе студената који су за учење користили овај алат. Резултати су показали да су захваљујући овом алату студенти стекли више знања, што су потврдили и резултати на испиту.

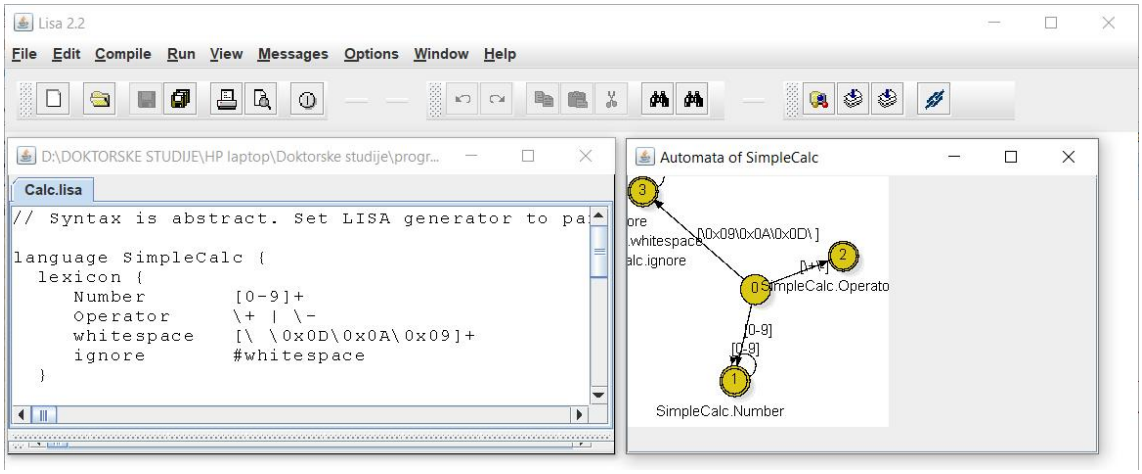


(а) (б)

Слика 45. SELFA-Pro: (а) начин дефинисања аутомата и граматике и (б) приказ конверзије коначног аутомата у DFA

LISA (Language Implementation System Based on Attribute Grammars) је софтверски алат развијен са циљем да студентима олакша учење и концептуално разумевање конструкције компајлера на ефикасан и директан начин (Mernik and Zumer, 2003). У овом алату студенти имају могућност да експериментишу, процењују и тестирају различите лексичке и синтаксне анализаторе, као и стратегије везане за употребу атрибута. За учење лексичке

анализе користе се регуларне дефиниције које се трансформишу у еквивалентни DFA. На слици 46 приказана је граматика једноставног калкулатора и одговарајући коначни аутомат. Синтаксна анализа демонстрира се конструкцијом синтаксног стабла, док се за тумачење значења програма, односно семантике, користе граматике засноване на атрибутима. Један од доказа да су студенти заиста стекли дубље разумевање конструкције компајлера захваљујући овом алату је да се просечна оцена на овом предмету повећала са 8,05 (на крају школске године са традиционалном наставом) на 8,6 (на крају школске године са алатом LISA).



Слика 46. Пример конструисања детерминистичког коначног аутомата у оквиру алата LISA

FADL (Finite Automaton Description Language) је језик за формално моделовање коначних аутомата на којем се заснива комплет алата за учење компајлера (Chakraborty, Saxena and Katti, 2013). Брзи компајлер са једним пролазом се користи за компајлирање коначног аутомата дефинисаног у FADL. Комплет алата омогућава симулацију рада дефинисаног аутомата за било који улазни стринг, минимизацију броја стања аутомата, конверзију NFA у DFA и претварање DFA у Тјурингову машину. Овај образовни алат заснива се на конзоли и развијен је на језику C++. Прелиминарна евалуација, на основу анкете студената који су користили овај пакет алата, показала је да је студентима FADL занимљиво наставно средство и да им је помогао да стекну бољу перцепцију о коначним аутоматима.

4.2 МЕТОД ЕВАЛУАЦИЈЕ

Евалуација софтвера за наставу/учење састоји се од два типа испитивања, формативног и сумативног (Geissinger, 1997). Формативне методе евалуације се користе када се започне рад на дизајну и развоју различитих делова софтвера, док се евалуација сумативним методама спроводи на готове софтверске производе. Евалуација образовних софтвера подразумева прикупљање, анализу и тумачење информација о било ком аспекту програма образовања и обуке као део процеса процене његове употребљивости и ефикасности (Thorpe, 1993). Дакле, за образовне софтверске системе битне су две карактеристике, употребљивост система и успешност коришћења система у настави. Ефикасност, односно успешност едукативних софтверских алата, се углавном испитује контролисаним експериментом. Ова метода подразумева праћење и упоређивање исхода учења две групе студената. Једну групу чине студенти који су у процесу учења користили одговарајући алат, а друга група је контролна и чине је студенти чије се учење заснивало на традиционалним методама без помоћног софтверског алата. За евалуацију употребљивости система користе се различите методологије од којих су неке дефинисане у (Price et al., 1997):

- неформална испитивања (студенти дају своје мишљење о образовном систему),
- хеуристичка испитивања (стручњак проверава карактеристике система),
- технике упитника (студенти попуњавају упитнике о систему),
- опсервационе студије (посматра се на који начин студенти користе систем).

Аутори представљених образовних система су углавном извршили неку од наведених врста евалуације, што је и наглашено код описа датих система. Ове врсте евалуације се односе на процену квалитета једног софтверског система, а за потребе дисертације потребно је извршити компаративну процену квалитета свих презентованих алата, па је неопходно најпре усвојити

одговарајуће критеријуме који су од значаја за ово истраживање. Да би одређени софтверски систем могао да се усвоји као помоћни алат на предмету Програмски преводиоци на Факултету техничких наука Универзитета у Приштини, он би морао да обухвата што већи број тема предмета и морао би да поседује што боље карактеристике и напредније функционалности. У том контексту предлаже се евалуација на основу две групе критеријума (Stamenković, Jovanović and Chakraborty, 2020). У прву групу спадају критеријуми који се односе на карактеристике и функционалности симулатора, док у другу спадају критеријуми који се односе на теме програмских преводилаца које су подржане од стране симулационих система, такозвани критеријуми покривености.

4.2.1 Дефинисање прве групе критеријума

Прва група критеријума анализира основне карактеристике симулатора и функционалности које могу бити од значаја приликом избора одговарајућег образовног система за примену у настави. На основу датог прегледа софтверских система погодних за учење програмских преводилаца, може се извршити генерализација њихових карактеристика и заступљених функција. Овом генерализацијом долази се до битних критеријума за процену квалитета одабраних софтверских система.

Иако у основне техничке карактеристике спада програмски језик на којем је развијен анализирани образовни алат, битнија карактеристика коју треба узети у обзир је врста платформе за коју је пројектован. Неки системи су имплементирани као десктоп апликације, неки као веб базиране апликације, а присутни су и образовни системи пројектовани за мобилне платформе. Критеријум који је усвојен да ово прикаже назван је *Платформа*. Резултати процене могу бити: *Десктоп*, *Мобилна* и *Веб*.

Многи од представљених софтверских система приликом прве интеракције остављају лош утисак и одбојност због лоше организованог корисничког интерфејса или недостатка инструкција које упућују корисника

на акције које треба да предузме да би остварио одређени циљ. Међутим, међу анализираним системима има и оних који су једноставни за коришћење и јасно и недвосмислено наводе корисника на акције и крајње циљеве. У складу са тим, битна карактеристика за процену квалитета образовног софтвера је *Интуитивност*. Резултат процене софтверских алата по овом критеријуму може бити *Да* и *Не*.

Већина система за учење програмских преводилаца процес симулације и резултате примењених алгоритама приказује у оквиру графичког корисничког окружења, неки их само извозе у текстуалном облику, док се код неких рад заснива на командној линији. Напреднији системи имају и графичке едиторе за конструисање аутомата. Критеријум који је за ову карактеристику усвојен је *Графичко окружење*. Резултати процене могу бити *Да*, уколико је за приказ симулације и резултата предвиђено графичко окружење посебно пројектовано за ту сврху, у супротном резултат процене је *Не*.

У претходним поглављима је интерактивност представљена као веома важна карактеристика образовних софтверских система. Софтвер се може сматрати интерактивним ако пружа било који вид интеракције који има за циљ да код корисника изазове размишљање. Ипак, неки од представљених система на основу задатих улазних параметара презентују само финалне резултате рада алгоритма, што на основу претходне дефиниције интерактивности софтвера имплицира одсуство исте. Критеријум који је усвојен за процену ове карактеристике је *Интерактивност*, а резултати процене могу бити *Да* и *Не*.

Једну групу образовних система чине алати који су развијени као вишенаменски софтверски пакети за проучавање више сродних концепата програмских преводилаца, док другу групу чине алати специјализовани за ужу област програмског превођења. На основу ових информација едукативне алате можемо поделити на *Вишенаменске* и *Специјализоване*. Критеријум за ову процену назива се *Тип алата*.

Неки од одабраних образовних система обрађују исте теме програмских преводилаца, али обим и квалитет презентованих информација о датој теми се може прилично разликовати, на пример принцип рада неког алгоритма код неких система се демонстрира само приказом коначних резултата рада, код неких површно кроз приказ основних концепата, док одређени системи пружају напредни ниво презентације са детаљнијим информацијама о обрађиваном алгоритму. Критеријум који је усвојен за процену ове карактеристике је *Садржај*. Софтверски систем на основу овог критеријума може бити *Основни* и *Напредни*.

Приказ тока симулације може се разликовати од система до система. Код неких се процес симулације врши интерактивно корак по корак омогућавајући кориснику покретање и заустављање симулације, враћање уназад или настављање процеса, док се код осталих процес симулације не може контролисати. Критеријум који је на основу тога усвојен је *Контрола визуелизације*. Резултат процене софтверских система по овом критеријуму може бити *Да* и *Не*.

Веома битна функција алата је и могућност чувања започетог или завршеног рада како би корисник у сваком тренутку могао да учита свој рад због додатних анализа и тестирања. Такође, неки системи дозвољавају задавање улазних параметара не само из форме корисничког интерфејса, већ и учитавањем из датотеке. Критеријум за ову анализу је *Чување/учитавање података*. Резултат процене може бити *Да*, уколико систем омогућава бар неку функцију од наведених, у супротном резултат процене ће бити *Не*.

Код неких алата постоји добар систем помоћи кориснику у виду упутства за коришћење, обавештења у случају погрешних уноса, понуђених примера за демонстрацију начина рада и слично. Неки од анализираних алата не пружају ниједан вид помоћи кориснику. Критеријум који је у овом контексту усвојен је *Помоћ*. Резултат процене алата по овом критеријуму може бити *Да* и *Не*.

Образовни алати би требали да поседују и могућност за процену стеченог знања како би студенти могли да провере своје напредовање у учењу или како би наставници могли да задају тестове и прате активности студената. Критеријум који се може успоставити за оцену ове карактеристике алата је *Процена знања*. Резултати на основу процене овог критеријума могу бити *Да* и *Не*.

Битна карактеристика образовних алата је и њихова јавна доступност. У табели 1, која садржи основне информације о анализираним алатима, наведене су и њихове веб адресе, односно линкови за преузимање. На основу тога, критеријум *Доступност алата* може имати вредности *Да* и *Не*.

Успостављени критеријуми за евалуацију употребљивости образовних софтверских система погодних за учење алгоритама и техника програмских преводилаца, који су засновани на карактеристикама и функцијама алата, приказани су у табели 2.

Табела 2. Група критеријума заснованих на катактеристикама и функцијама образовних алата

Бр.	**Категорија**	**Опис**
1	*Платформа*	Алат може бити имплементиран као апликација за веб, мобилну или платформу десктоп рачунара.
2	*Интуитивност*	Софтвер је једноставан за коришћење и кориснику пружа јасне акције које треба да изврши да би постигао циљ.
3	*Графичко окружење*	Процес симулације и резултати примењених алгоритама приказују се у оквиру графичког корисничког окружења.
4	*Интерактивност*	Било који вид интеракције који има за циљ да код корисника изазове размишљање.
5	*Тип алата*	Алати могу бити вишенаменски за учење више сродних концепата програмских преводилаца, или специјализовани за ужу област програмског превођења.
6	*Садржај*	Обим и квалитет презентованих информација одређују да ли се алат може користити за основни или напредни ниво учења.
7	*Контрола визуелизације*	Могућност контроле праћења визуелизације извршавања алгоритма, корак по корак, или контрола на неком другом нивоу.
8	*Чување/учитавање података*	Могућност чувања и поновног учитавања података који се могу односити на примере за вежбање или улазне параметре алгоритама.
9	*Помоћ*	Софтвер пружа обавештења кориснику путем одговарајућих и правовремених повратних информација.
10	*Процена знања*	Могућност решавања постављених задатака или питања уз пружање информација о исходу.
11	*Доступност алата*	Софтверски алат је јавно доступан.

4.2.2 Дефинисање друге групе критеријума

Друга група критеријума анализира степен покривености тема. Да би се прецизно дефинисали критеријуми покривености, потребно је утврдити списак наставних јединица са припадајућим темама. У ту сврху у раду (Stamenković, Jovanović and Chakraborty, 2020) је коришћен интернационални наставни план препоручен од стране заједничке радне групе ACM (Association for Computing Machinery) и IEEE Computer Society - Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (Computer Science Curricula, 2013). У наведеном документу је предложено да предмет *Topics in Compiler Construction* обухвата седам наставних јединица са одговарајућим темама и бројем часова. Препоручене наставне јединице биле су изабране за критеријуме покривености тема.

Табела 3. Наставне јединице и теме на предмету Програмски преводиоци

Редни бр.	**Наставна јединица**	**Тема**
1.	*Лексичка анализа*	Лексички анализатор
		Лексичка правила
		Регуларни изрази
2.	*Коначни аутомати*	Детерминистички коначни аутомати
		Минимизација аутомата
		Недетерминистички коначни аутомати
		Конверзија NFA у DFA
		Конверзија регуларног израза у NFA
		Конструкција DFA директно из регуларног израза
		Конверзија DFA у регуларни израз
		Потисни аутомати
3.	*Синтаксна анализа*	Граматика
		Стабло извођења
		Нерекурзивно парсирање
		Синтаксна анализа наниже (LL парсирање)
		Синтаксна анализа навише (LR парсирање)
4.	*Семантичка анализа*	Табеле симбола
		Синтаксно управљане дефиниције
		Атрибутивне граматике (синтетизовани и наслеђени атрибути)
		Конструкција анотираног синтаксног стабла
		Транслационе шеме
5.	*Међукод*	Апстрактно синтаксно стабло
		Троадресни код
		Оптимизација међукода
6.	*Генерисање кода*	Генератор кода
		Алокација регистра

Циљ докторске дисертације је проналазак ефикасног и ефективног решења за помоћ у учењу тема из предмета Програмски преводиоци на Факултету техничких наука Универзитета у Приштини. У складу са циљем намећу се и критеријуми засновани на покривеност тема. На акредитованом студијском програму основних академских студија Електротехничког и рачунарског инжењерства из 2021. године предмет Програмски преводиоци је обавезан предмет, и подразумева 2 часа теоријске и 2 часа практичне наставе. Теме се у великој мери поклапају са темама предмета Topics in Compiler Construction. Основне теме за чије изучавање је пожељна и препоручена употреба помоћних алата приказане су у табели 3. Наведене теме ће бити критеријуми за евалуацију алата са аспекта покривености.

4.3 ЕВАЛУАЦИЈА ОДАБРАНИХ СИМУЛАЦИОНИХ СИСТЕМА

Категорије успостављене методе евалуације јасно су дефинисане тако да се приликом процене недвосмислено може утврдити испуњеност одговарајућих критеријума. Резултати евалуације применом дефинисаног модела (на основу усвојених група критеријума) презентовани су у наредним секцијама.

4.3.1 Резултати евалуације на основу прве групе критеријума

Резултати спроведене евалуације на основу карактеристика и функционалности софтверских система приказани су у табели 4. У погледу платформе анализираних софтверских система може се приметити да се углавном ради о апликацијама намењеним десктоп рачунарима. Укупно је 13 таквих система. За развој десктоп система најчешће су коришћени програмски језици Јава и C++. Присутно је и 5 веб базираних апликација: Seshat, DFA simulation tool, JCT, Webworks applets и SELFA-Pro. У питању су клијент сервер апликације које се на клијентској страни извршавају у оквиру неког веб читача. Предност оваквих софтверских алата је што су независни од оперативног система и хардвера, и могу се извршавати на било ком уређају

који има одговарајући веб читач. Због тога се ови системи могу посматрати и као вишеплатформски. Од анализираних образовних система два су имплементирана као мобилне апликације за Андроид оперативни систем: Automata Simulator и CMSimulator.

Табела 4. Резултати евалуације на основу карактеристика и функционалности софтверских система

Софтверски систем	***Критеријум***										
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
JFLAP	Десктоп	Да	Да	Да	Вишенаменски	Напредни	Да	Да	Да	Не	Да
VAST	Десктоп	Да	Да	Да	Специјализовани	Основни	Да	Не	Да	Не	Не
Automata Simulator	Мобилна	Да	Да	Да	Специјализовани	Основни	Не	Не	Да	Не	Да
Language Emulator	Десктоп	Да	Да	Не	Специјализовани	Основни	Не	Не	Не	Не	Да
CMSimulator	Мобилна	Да	Да	Да	Специјализовани	Основни	Да	Да	Да	Не	Да
Seshat	Веб	Да	Да	Да	Вишенаменски	Напредни	Да	Да	Да	Не	Да
Robot-based Automata Simulator	Десктоп	Да	Да	Да	Специјализовани	Основни	Да	Да	Не	Не	Не
L-FLAT	Десктоп	Не	Не	Да	Специјализовани	Основни	Не	Не	Да	Не	Да
DFA simulation tool	Веб	Да	Да	Да	Специјализовани	Основни	Не	Не	Да	Не	Не
jFAST	Десктоп	Да	Да	Да	Специјализовани	Основни	Да	Да	Не	Не	Да
RegExpert	Десктоп	Да	Да	Не	Специјализовани	Основни	Не	Не	Не	Не	Не
SOTA	Десктоп	Да	Да	Да	Специјализовани	Напредни	Да	Да	Да	Не	Не
LLparse & LRparse	Десктоп	Да	Да	Да	Специјализовани	Напредни	Да	Да	Да	Не	Да
BURGRAM	Десктоп	Не	Да	Да	Специјализовани	Напредни	Да	Да	Да	Не	Да
PAVT	Десктоп	Да	Не	Не	Специјализовани	Напредни	Не	Да	Да	Не	Да
JCT	Веб	Да	Да	Да	Специјализовани	Напредни	Да	Да	Не	Не	Не
Webworks applets	Веб	Не	Да	Да	Вишенаменски	Напредни	Да	Не	Да	Не	Да
SELFA-Pro	Веб	Не	Да	Да	Вишенаменски	Напредни	Да	Да	Да	Не	Да
LISA	Десктоп	Не	Да	Да	Вишенаменски	Напредни	Не	Да	Да	Не	Да
FADL toolkit	Десктоп	Не	Не	Да	Специјализовани	Основни	Не	Не	Да	Не	Не

Већина софтверских алата испуњава критеријум који се односи на интуитивну ефикасност. Међутим, постоје и алати који нису једноставни за коришћење и приликом стартовања, код корисника који се први пут сусреће са његовим корисничким интерфејсом, изазивају збуњеност. Укупно је 6 таквих алата: L-FLAT, BURGRAM, Webworks applets, SELFA-Pro, LISA и FADL toolkit. Код неких од њих акције нису јасно дефинисане тако да корисници нису сигурни шта треба да предузму да би дошли до циља, док се код других алата акције могу предвидети, али не и начин на који их треба извршити. На пример, код BURGRAM-а је јасно да се очекује унос граматике за праћење одговарајућег алгоритма за парсирање, али није јасно у ком облику се задају граматике у предвиђеном текстуалном пољу.

Графичко корисничко окружење је веома важна карактеристика образовних система јер људи имају импресивну меморију за визуелне информације и просторну структуру, што значи да коришћење визуелних компоненти може имати само бенефит у корист едукације (Lindgren and Schwartz, 2009). L-FLAT и FADL toolkit су алати базирани на конзоли и свака функција се позива кратком наредбом у командној линији, а резултати се приказују у текстуалном облику. Специфичност се јавља код симулатора PAVT. Овај алат има графичко окружење за задавање улазних параметара за симулацију, док се резултати симулације приказују у текстуалном облику у фајлу са екстензијом txt. Осим ова три система, сви остали обезбеђују визуелну презентацију резултата симулација у оквиру својих графичких окружења.

Углавном сви од анализираних алата остварују неки вид интеракције са корисником. Међутим, код три алата ниво интерактивности је веома низак. Корисник са овим алатима врши интеракцију једино док уноси улазне параметре, након чега добија само приказ коначних резултата. То су алати: Language Emulator, RegExpert и PAVT.

Вишенаменски софтверски алати за учење програмских преводилаца имају за циљ интегрисање више различитих рачунарских симулатора у јединствени, обједињени пакет. У овој евалуацији идентификовано је пет

таквих софтверских алата: JFLAP, Seshat, Webworks applets, SELFA-Pro и LISA. Ови симулациони алати се могу користити за учење више различитих области програмских преводилаца нудећи широк спектар могућности, од експериментисања са регуларним изразима и граматикама, преко конструкције теоријских машина и аутомата до упознавања са процесом парсирања формирањем синтаксних табела и стабла парсирања. LISA обезбеђује и могућност симулације семантичке анализе. Остали симулатори се углавном фокусирају на одређене теме, односно на ужу област проучавања програмских преводилаца.

Половина софтверских система студентима обезбеђује добро конципиран и богат садржај, који им помаже да се детаљније упознају са презентованим теоријским концептом. На пример, и Language Emulator и SELFA-Pro подржавају конверзију DFA у регуларни израз. Међутим, разлика у садржају који презентују када приказују ову конверзију је у томе што Language Emulator приказује само добијени регуларни израз, док SELFA-Pro даје детаљан приказ поступка који доводи до тог коначног решења.

Веома важна карактеристика образовних система која директно утиче на процес учења је могућност контроле визуелизације, односно тока симулације. Контрола визуелизације подразумева извршавање процеса симулације парцијално у корацима где се при сваком наредном кораку кориснику презентују повратне информације о предузетим функцијама, односно правилима, која се у том кораку примењују. У најбољем случају, симулатори имају и могућност паузирања симулације и враћања неколико корака уназад. Најбољи примери за ову функционалност су JFLAT и Seshat. Симулатори који немају ову могућност су: Language Emulator, Automata Simulator, DFA simulation tool, L-FLAT, PAVT, RegExpert, LISA и FADL toolkit.

Функција чувања започетог или завршеног рада је пожељна код образовних алата и због тога што омогућава студентима да изводе вежбе код куће, а сачуване резултате вежби касније презентују наставнику. Већина анализираних симулатора поседује ову функцију. Специфичност постоји код

Seshat-а јер уместо чувања података нуди могућност експортовања коначног решења алгоритма у JSON формат и то не директним одабиром одговарајуће опције за чување, већ уносом посебног URI (Uniform Resource Identifier) за сваки алгоритам у пољу за претрагу веб читача. Што се тиче осталих критеријума, треба напоменути да пет софтверских система не пружају ниједан вид помоћи, док ниједан од анализираних алата не поседује могућност процене знања корисника.

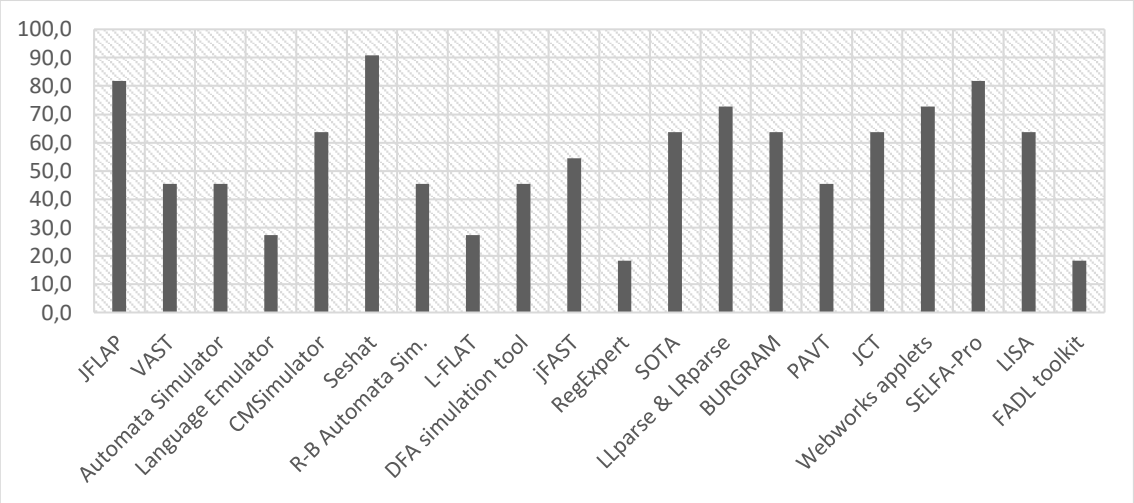
Иако је овде дата детаљнија дискусија о резултатима евалуације алата на основу њихових карактеристика и функција, било би пожељно квантификовати ове резултате како би се могла извршити директна компарација анализираних алата на основу прве групе критеријума. Јасно је да се код критеријума са могућим резултатима процене *Да* и *Не* могу увести оцене „1“ и „0“. Код критеријума *Платформа* веб системи добијају оцену „1“ због објашњене предности коју имају над осталим системима. По истој аналогији, *Вишенаменски* системи код критеријума *Тип алата* и *Напредни* системи код критеријума *Садржај* добијају оцену „1“. Применом ове методе процене формирана је табела 5 која приказује коначне оцене ове квантитативне евалуације, као и проценат испуњености критеријума.

Табела 5. Квантитативни резултати евалуације на бази прве групе критеријума

	JFLAP	**VAST**	**Automata Simulator**	**Language Emulator**	**CMSimulator**	**Seshat**	**R-B Automata Sim.**	**L-FLAT**	**DFA simulation tool**	**jFAST**	**RegExpert**	**SOTA**	**LLparse & LRparse**	**BURGRAM**	**PAVT**	**JCT**	**Webworks applets**	**SELFA-Pro**	**LISA**	**FADL toolkit**
Укупна оцена	9	5	5	3	7	10	5	3	5	6	2	7	8	7	5	7	8	9	7	2
%	81,8	45,5	45,5	27,3	63,6	90,9	45,5	27,3	45,5	54,5	18,2	63,6	72,7	63,6	45,5	63,6	72,7	81,8	63,6	18,2

Резултати евалуације показују да је софтверски алат Seshat најбоље оцењен (90,9% испуњености критеријума) у погледу карактеристика и функција које обезбеђује. Овај образовни алат не испуњава само један

критеријум који се односи на могућност провере знања студената. По укупној оцени следећи су JFLAP и SELFA-Pro. SELFA-Pro је у предности у односу на JFLAP у погледу платформе, али код овог алата изостаје важна карактеристика једног едукативног алата, а то је интуитивност. По резултату следе алат LLparse & LRparse који је уско специјализован за синтаксну анализу и Webworks applets као један веб базиран вишенаменски софтверски систем са преко 70% испуњености критеријума. Најслабије оцене у овој евалуацији имају алати RegExpert и FADL toolkit јер испуњавају само по два критеријума. Графички приказ резултата спроведене евалуације засноване на првој групи успостављених критеријума дат је на слици 47.



Слика 47. Графички приказ процента испуњености прве групе критеријума евалуације

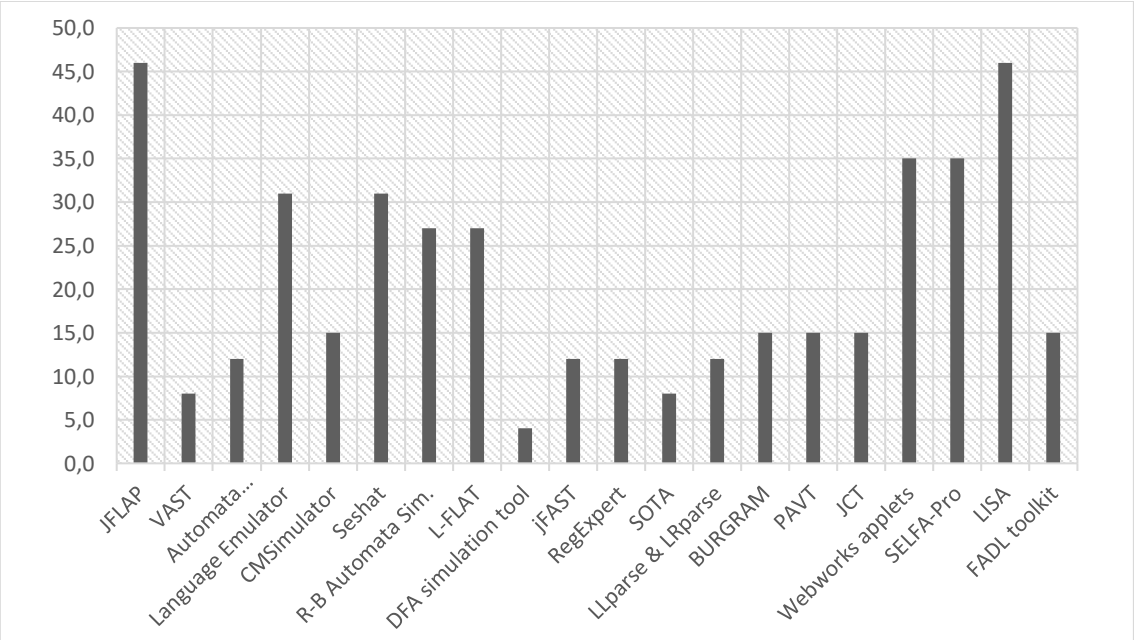
4.3.2 Резултати евалуације на основу друге групе критеријума

Резултати евалуације симулатора на основу критеријума покривености тема приказани су у табели 6. Знак „+“ означава да алат поседује функцију којом се може покрити одговарајућа тема наставне јединице, док знак „-“ значи да се алат не може користити за покривање дате теме. На дну табеле приказан је укупан број тема које се могу покрити анализираним системом, као и процентуални удео тих тема у односу на укупан број тема наставног предмета Програмски преводиоци.

Табела 6. Резултати евалуације алата на основу покривености тема

Наставне јединице	**Теме**	**JFLAP**	**VAST**	**Automata Simulator**	**Language Emulator**	**CMSimulator**	**Seshat**	**R-B Automata Sim.**	**L-FLAT**	**DFA simulation tool**	**jFAST**	**RegExpert**	**SOTA**	**LLparse & LRparse**	**BURGRAM**	**PAVT**	**JCT**	**Webworks applets**	**SELFA-Pro**	**LISA**	**FADL toolkit**
Лексичка анализа	Лексички анализатор	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
	Лексичка правила	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
	Регуларни изрази	+	-	-	+	-	+	+	+	-	-	+	-	-	-	-	-	+	+	+	-
Коначни аутомати	Детерминистички коначни аутомати	+	-	+	+	+	+	+	+	+	+	-	-	-	-	-	+	+	+	+	+
	Минимизација аутомата	+	-	-	+	-	+	+	+	-	-	-	-	-	-	-	+	+	+	-	+
	Недетерминистички коначни аутомати	+	-	+	+	+	+	+	+	-	+	+	-	-	-	-	+	+	+	-	+
	Конверзија NFA у DFA	+	-	-	+	-	+	+	-	-	-	-	-	-	-	-	+	+	+	-	+
	Конверзија регуларног израза у NFA	+	-	-	+	-	+	+	-	-	-	+	-	-	-	-	-	+	+	-	-
	Конструкција DFA директно из регуларног израза	-	-	-	-	-	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-
	Конверзија DFA у регуларни израз	+	-	-	+	-	-	+	-	-	-	-	-	-	-	-	-	+	+	-	-
	Потисни аутомати	+	-	+	-	+	-	-	+	-	+	-	-	-	-	-	-	-	+	-	-
Синтаксна анализа	Граматика	+	+	-	+	+	-	-	+	-	-	-	+	+	+	+	-	+	+	+	-
	Стабло извођења	+	+	-	-	-	+	-	-	-	-	-	-	-	+	+	-	+	-	+	-
	Нерекурзивно парсирање	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Синтаксна анализа наниже (LL парсирање)	+	-	-	-	-	-	-	-	-	-	-	-	+	+	+	-	-	-	+	-
	Синтаксна анализа навише (LR парсирање)	+	-	-	-	-	-	-	-	-	-	-	-	+	+	+	-	-	-	+	-
Семантичка анализа	Табеле симбола	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	+	-
	Синтаксно управљане дефиниције	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
	Атрибутивне граматике (синтетизовани и наслеђени атрибути)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
	Конструкција анотираног синтаксног стабла	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
	Транслационе шеме	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Међукод	Апстрактно синтаксно стабло	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Троадресни код	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Оптимизација међукода	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Генерисање кода	Генератор кода	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Алокација регистра	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Score	12	2	3	8	4	8	7	7	1	3	3	2	3	4	4	4	9	9	12	4
	%	46	8	12	31	15	31	27	27	4	12	12	8	12	15	15	15	35	35	46	15

На слици 48 дат је графички приказ резултата евалуације који приказују проценат покривености тема предмета Програмски преводиоци од стране анализираних алата. Укупни резултати евалуације показују да не постоји ниједан симулатор који може да покрије све теме предмета Програмски преводиоци. Веома је комплексно развити свеобухватни алат који би симулирао комплетан процес програмског превођења, од лексичке анализе до генерисања кода. Најбољу укупну покривеност тема (46%) имају симулациони системи JFLAP и LISA. JFLAP је широко прихваћен образовни систем који се користи на многим факултетима управо због тога што важи за један од најпотпунијих алата из ове области. Очекивано је да алати који су у евалуацији базираној на карактеристикама означени као вишенаменски имају најбољу покривеност тема, а поред алата JFLAP и LISA најбољу покривеност тема имају и SELFA-Pro и Webworks applets са по 35% и Seshat са 31%. Сви остали симулатори имају мању покривеност јер су специјализовани за уже области, односно само за одређене теме. Проценат покривености појединих наставних јединица анализираним софтверским алатима приказан је у табели 7.



Слика 48. Графички приказ резултата евалуације алата на основу друге групе критеријума

Табела 7. Приказ процентуалне покривеност наставних јединица

Наставна јединица	**JFLAP**	**VAST**	**Automata Simulator**	**Language Emulator**	**CMSimulator**	**Seshat**	**R-B Automata Sim.**	**L-FLAT**	**DFA simulation tool**	**jFAST**	**RegExpert**	**SOTA**	**LLparse & LRparse**	**BURGRAM**	**PAVT**	**JCT**	**Webworks applets**	**SELFA-Pro**	**LISA**	**FADL toolkit**
Лексичка анализа	33,3	0	0	33,3	0	33,3	33,3	33,3	0	0	33,3	0	0	0	0	0	33,3	33,3	100,0	0
Коначни аутомати	87,5	0	37,5	75,0	37,5	75,0	75,0	62,5	12,5	37,5	25,0	0	0	0	0	50,0	75,0	87,5	12,5	50,0
Синтаксна анализа	80,0	40,0	0	20,0	20,0	20,0	0	20,0	0	0	0	20,0	60,0	80,0	80,0	0	40,0	20,0	80,0	0
Семантичка анализа	0	0	0	0	0	0	0	0	0	0	0	20,0	0	0	0	0	0	0	80,0	0
Међукод	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Генерисање кода	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Теме у оквиру наставне јединице *Лексичка анализа* нису адекватно покривене анализираним алатима. Више од половину алата не подржава ниједну тему из ове области, део алата обрађује само тему која се односи на регуларне изразе, док само алат LISA покрива све теме у овој области.

Област *Коначни аутомати* је у великој мери покривена од стране анализираних софтверских система. Највећи број алата је уско специјализован за учење тама из ове области, али ипак ниједан алат је не покрива у потпуности. Најбоље покривена тема из ове области је *Детерминистички коначни аутомати*, коју обрађује чак 14 система. JFLAP и SELFA-Pro имају највећи проценат покривености ове области од 87,5%.

Наставна јединица *Синтаксна анализа* такође бележи добру покривеност тема. Највећи проценат покривености од 80,0% имају алати: JFLAP, BURGRAM, PAVT и LISA. Укупно седам система не покрива ниједну тему из ове наставне јединице. Најбоље покривене теме у овој области су *Граматика* и *Стабло извођења*.

Теме које се односе на *Семантичку анализу* су изразито лоше покривене анализираним симулаторима. Ову наставну јединицу једино покривају LISA и SOTA, при чему SOTA демонстрира само једну тему, док LISA покрива 80,0% ове области и ово је једини алат који студентима може пружити помоћ у учењу семантичких правила и атрибутне граматике.

Наставне јединице *Међукод* и *Генерисање кода* нису подржане од стране анализираних софтверских система. Већина анализираних алата углавном се фокусира на основне теме програмских преводилаца које се односе на теорију аутомата и формалних језика.

5. РАЗВОЈ НОВОГ СИСТЕМА ЗА УЧЕЊЕ ПРОГРАМСКИХ ПРЕВОДИЛАЦА

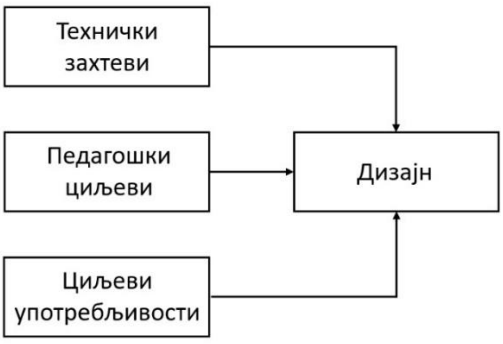
У овом поглављу биће наведени разлози за развој и имплементацију новог софтверског система као помоћног наставног средства на предмету Програмски преводиоци. На основу резултата анализе из претходног поглавља и прегледа релевантне литературе, истичу се и класификују циљеви, односно формира се таксономија циљева за дизајн образовног алата ComVis. У овом поглављу акценат се ставља на приступ дизајну образовног система вођеног јасно дефинисаним техничким, педагошким и циљевима употребљивости.

5.1 ЗАХТЕВИ ЗА РАЗВОЈ НОВОГ СОФТВЕРСКОГ СИСТЕМА

У претходном поглављу извршена је детаљна анализа и процена квалитета образовних софтверских система као потенцијалних помоћних алата за потребе предмета Програмски преводиоци. Показало се да иако постоји већи број таквих система релативно мали број њих поседује задовољавајуће карактеристике и функционалности, док међу анализираним системима не постоји ниједан систем који покрива потребан број тема предмета. Покривеност тема анализираним алатима је прилично лоша. Најбољи проценат покривености је 46%, код алата JFLAP и LISA који пак не испуњавају све неопходне карактеристике. Са друге стране, најбоље оцењени систем са аспекта карактеристика и функционалности је Seshat, али његов степен покривености тема је свега 31%. Узимајући све ово у обзир, може се предложити надоградња и унапређење неког од постојећих система или реализација потпуно новог образовног алата. Системи са најбољим оценама на спроведеним евалуацијама су добри кандидати за евентуални редизајн и надоградњу, али у тренутку истраживања ниједан од њих није био отвореног кода због чега је и донета одлука да се приступи развоју новог софтверског система.

Истраживање које је подразумевало проналазак ефикасног решења за развој образовног алата за учење програмских преводилаца започето је 2017. године, а аутор ове дисертације се прикључује истраживањима 2019. године. Како би се јасно истакли захтеви за нови софтверски систем, извршено је систематично проучавање литературе из области развоја и дизајна образовних софтверских система за визуелизацију алгоритама, а анализирани су и проблеми са којима се сусрећу овакви софтвери приликом примене у наставном процесу. Препреке за увођење интерактивних динамичких визуелизација у настави укључују и педагошке проблеме и проблеме употребљивости (Amershi et al., 2008; Tversky, Morrison and Betrancourt, 2002). Због тога се пре укључивања неког едукативног софтвера у теоријска предавања или лабораторијске вежбе морају извршити евалуације ефикасности у смислу стицања знања, разумевања, ангажовања и мотивације студената. Недостаци употребљивости односе се на време потребно за проналажење и примену одговарајућих функција алата, а затим и на уложени труд и време за учење начина коришћења алата (Naps et al., 2003).

Због свега наведеног, у фази пројектовања главни захтеви за дефинисање дизајна образовног алата односе се на остваривање техничких, педагошких и циљева употребљивости. У оквиру овако успостављене таксономије циљева биће каласификовани и критеријуми који су коришћени за евалуацију одабраних алата у поглављу 4. Концептуални модел за дефинисање дизајна софтверског система ComVis приказан је на слици 49.



Слика 49. Концептуални модел за дефинисање дизајна образовног софтверског система ComVis

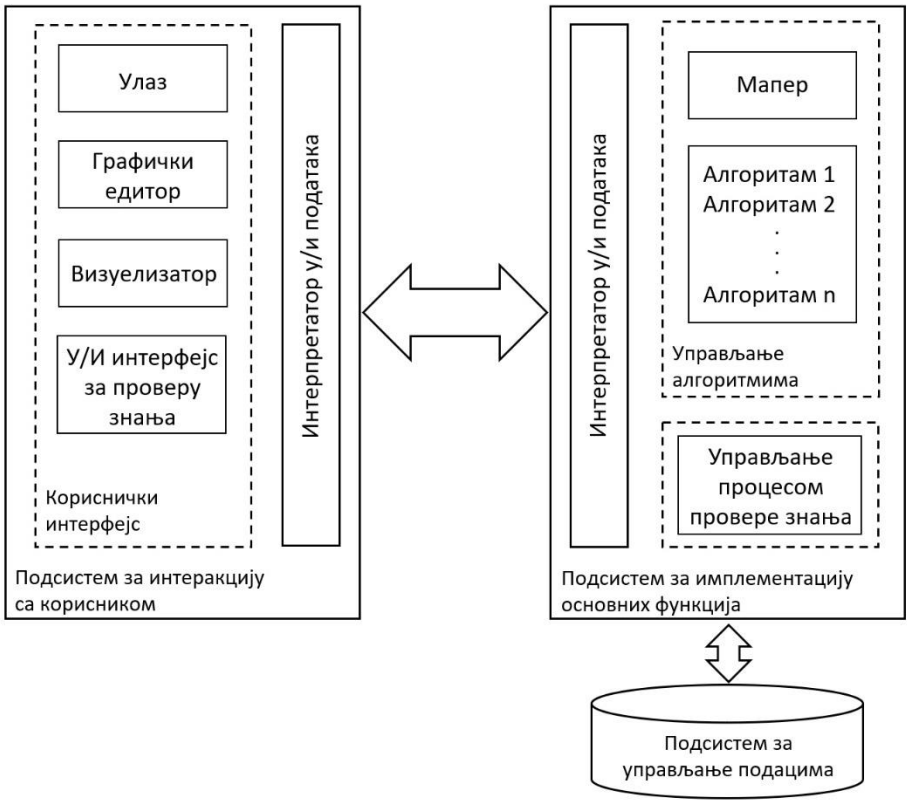
5.2 ТЕХНИЧКИ ЗАХТЕВИ И АРХИТЕКТУРА СИСТЕМА

За дефинисање архитектуре система потребно је сагледати све техничке захтеве, као и успостављене критеријуме у поглављу 4 који могу имати утицај на коначну структуру свих компоненти система. Критеријуми које треба узети у обзир код пројектовања архитектуре су: платформа, графичко окружење, контрола визуелизације, процена знања.

Претходном анализом указано је на потенцијални проблем доступности система који су реализовани као десктоп апликације. Такви системи су везани само за одређени оперативни систем, а често се јављају и проблеми дистрибуције њихових инсталационих пакета. Унапређивање функција система или исправљање уочених недостатака захтева обавештавање корисника, дистрибуирање нове верзије и ангажовање корисника приликом инсталације. Да би се ови проблеми превазишли, нови систем треба бити пројектован као веб апликација. На овај начин се обезбеђује *независност од хардвера и оперативног система*, а унапређење постојећих и додавање нових функција обавља се аутоматски и без ангажовања корисника. Резултате симулације алгоритама и техника програмског превођења не треба приказивати само у текстуалном облику, већ у одговарајућем графичком окружењу чији изглед ће коначно дефинисати педагошки и циљеви употребљивости. Уколико је то могуће, улазне податке треба уносити путем графичког едитора. Графички кориснички интерфејс треба да буде естетски допадљив уз *минималистички дизајн*, да би софтвер био привлачан не треба да приказује небитне или ретко коришћене информације. Функција контроле визуелизације указује на то да приликом пројектовања архитектуре система треба предвидети компоненту која ће бити задужена за издвајање података о специфичним међуфазама алгоритма који се симулира. Могућност провере знања студената подразумева да софтвер треба да обезбеди *пријаву на систем* са корисничким улогама студент и наставник тако да наставницима буде доступна опција за креирање и чување тестова, а студентима опција за решавање тестова и чување података о резултатима. Пријављени студент би

на тај начин на било ком уређају и у било ком тренутку имао приступ свом едукативном окружењу и својим резултатима рада. Ово имплицира обавезну употребу базе података коју треба предвидети у архитектури образовног софтверског система.

У фази дефинисања дизајна основни технички захтев је био да софтверски систем буде *лако надоградив*. Због тога је алат пројектован као модуларни софтверски систем који у свом саставу може имати скуп подмодула у виду независних интерактивних алата. У складу са свим наведеним техничким захтевима предложена је архитектура новог софтверског система ComVis која је приказана на слици 50.



Слика 50. Архитектура софтверског система ComVis

Овакав концепт архитектуре пружа могућност поделе програмског кода апликације на сегменте који су задужени за имплементацију конкретних функција система. У оквиру подсистема за интеракцију са корисником реализују се независни сегменти који обезбеђују улазно-излазне операције

путем графичког корисничког интерфејса. Оваква архитектура за унос улазних података предвиђа два различита начина, улазни параметри за симулацију алгоритама могу се задати у текстуалном облику или графички. Помоћу графичког едитора подаци се уносе конструкцијом дијаграма стања или попуњавањем аутоматски генерисаних табела. За унос података у текстуалном облику (на пример програмског кода за симулацију рада лексичког анализатора) задужен је улазни блок који имплементира различите облике текстуалних форми. Илустрацију рада алгоритама врши визуелизатор тако што за приказ резултата користи више графичких елемената. На корисничкој страни је предвиђен још један улазно-излазни блок, доступан само након пријаве корисника на систем и намењен за реализацију функције провере знања.

Језгро подсистема за имплементацију основних функција чине програмски кодови алгоритама чији се рад симулира путем софтверског система. Да би визуелизација алгоритма била могућа, потребни су и међурезултати у одговарајућим кључним тачкама. Управо томе служи мапер који за сваки алгоритам мапира битне позиције у коду у којима долази до значајних акција које треба приказати кориснику.

На страни оба подсистема постоје интерпретатори улазно-излазних података који представљају интерфејсе за комуникацију. Код подсистема за интеракцију са корисником, интерпретатор прихвата све улазне податке задате графички или текстуално и конвертује их у пакете погодне за слање. На страни подсистема за имплементацију основних функција интерпретатор из добијених пакета формира променљиве које се одговарајућем алгоритму прослеђују као параметри. Када алгоритам заврши са радом, разултате са свих мапираних тачака шаље назад интерпретатору. У зависности од алгоритма од којег добија податке, интерпретатор ређа резултате по тачно дефинисаном редоследу. На корисничкој страни интерпретатор обрађује добијене податке и структурира их у облику погодном за илустрацију од стране визуелизатора. Оваква организација софтвера обезбеђује модуларност систему и једноставну

надоградивост. За сваку нову функцију формира се посебан подмодул. Имплементација новог подмодула подразумева додавање одговарајућег алгоритма, дефинисање кључних тачака у маперу и конфигурисање интерпретатора улазно-излазних података.

5.3 ПЕДАГОШКИ ЦИЉЕВИ

Да би софтверски систем за помоћ у учењу допринео образовању, мора да пружа јасне и одређене педагошке бенефите. У наставку су наведени педагошки циљеви, као и препоручене карактеристике алата за остваривање датих циљева који представљају основу за дизајн ComVis-а.

Теорија наставе која се фокусира на конструктивистичке приступе указује на важност активног учења, односно активног ангажовања студента у образовном окружењу које треба да подстиче слободну интеракцију са информацијама (Elissavet and Economides, 2000). Истраживања су показала да студенти најбоље уче када су активно укључени у сам процес учења (Chickering and Gamson, 1987). Да би се постигао ефекат активног учења, образовно окружење треба да обезбеди активности које су смислене и које наводе студенте на размишљање о томе шта раде (Bonwell and Eison, 1991). Schweitzer и Brown (2007) сматрају да за *ангажовање* студената образовно окружење треба да обезбеди неке од следећих карактеристика:

- Интерактивност (подразумева постојање неке активности која укључује механизам повратне информације након предузете акције од стране студента);
- Једноставност за разумевање (ако активност захтева много објашњења или је сложена за разумевање она одвлачи пажњу од сврхе и троши драгоцено време);
- Кратак временски оквир (активности треба да буду релативно кратке тако да допуњују предавања, а не да их замењују);

- Креативност (активности које се сматрају досадним неће задржати интересовање студената и могу бити контрапродуктивне);
- Колаборативност (активности активног учења могу бити ефикасне као појединачне или групне активности);
- Релевантност (нерелевантна активност ван контекста теме проучавања може да разбије досаду, али она не доприноси образовном исходу часа).

Nahvi (1996) сматра да за *мотивацију* студената софтверски системи морају да буду интуитивни, односно једноставни за употребу и разумевање. Типичан начин мотивисања студената је да буду обавештени о томе шта ће постићи на крају наставе наводећи циљеве (Gagné, Briggs, Wager, 1988). У контексту образовног алата, студента пре сваког корака симулације треба обавестити о томе која се акција предузима и шта се њоме постиже. Гарантовање мерљивог *повећања разумевања* концепата и процеса предмета проучавања традиционално је у фокусу многих истраживача који се баве образовним софтверским системима (Hundhausen, 2002). Дубљем разумевању динамичких алгоритама може допринети релевантан садржај представљен употребом више различитих графичких елемената како би се студенту помогло да формира тачне менталне моделе алгоритама. Један од битнијих фактора који утиче на разлике у разумевању је *предзнање студената* (Adams et al., 1996). Од система за електронско учење се очекује да обезбеде персонализацију, односно интеракцију са студентима узимајући у обзир њихове разлике у нивоима знања, стиловима учења и приоритетима (Wade and Ashman, 2007). Зато приликом пројектовања софтвера треба узети у обзир чињеницу да студенти могу имати различите нивое предзнања. Алат треба да буде прилагођен како студентима без предзнања тако и напреднијим студентима, истовремено подржавајући *индивидуални темпо учења*. У хипермедијским системима учења важна је и контрола која је примарна у дизајну интерактивног учења јер омогућава студентима да прилагоде искуство учења својим индивидуалним потребама. Међутим, постоје

опасности у предаји превише контроле кориснику. Студенти са ниским способностима могу се збунити када контрола зависи од широког спектра опција (Litchfield, 1993). Висок ниво контроле студента може довести до дезоријентације и ометања. Контрола темпа учења од стране студента је ипак прикладнија од контроле која зависи од система. Иновативна технологија мора корисницима да пружи осећај да они владају системом. Студенти морају имати могућност да преузму саморегулишућу улогу у процесу учења тако да буду свесни да сами контролишу своје учење (Ferdig, 2006). Hattie и Jaeger (1998) тврде да *процену знања* треба посматрати као саставни део наставе и учења, а не као додатак томе. Неопходно је створити систем оцењивања студената који узима у обзир образовне циљеве и помаже студентим да развију своје вештине које ће дугорочно бити од користи (Ridgway, McCusker and Pead, 2004). Електронска процена знања унапређује мерење исхода студената и омогућава добијање тренутних и директних повратних информација (Alruwais, Wills and Wald, 2018). Након завршеног учења одређене области или теме образовни систем треба пружити студенту могућност процене стеченог знања.

5.4 ЦИЉЕВИ УПОТРЕБЉИВОСТИ

Многи аутори наводе различите дефиниције употребљивости. У већини случајева заједнички консензус је да софтверски производ прво мора да буде функционално исправан, а затим и *једноставан за употребу* (Belson and Ho, 2012). На основу истраживања из теоријске и практичне перспективе у области софтверског инжењерства, употребљивост се може дефинисати као лакоћа са којом корисник може да научи функционисање система, то јест да буде у могућности да припрема улазне податке и тумачи излазе система или компоненте (Feizi and Wong, 2012). Лакоћа коришћења је ипак субјективна ствар. Употребљивост као критеријум квалитета ISO (International Organization for Standardization) објашњена је у два актуелна стандарда: ISO/IEC 9126-1: 2004 (разумљивост, могућност учења, операбилност и

атрактивност) и ISO/IEC 9241-11: 2018 (ефективност, ефикасност и задовољство), као и нормама ISO/IEC 25010: 2011 (прикладност, препознатљивост, операбилност, заштита од грешака корисника, естетика корисничког интерфејса и приступачност). Из перспективе софтверског инжењерства употребљивост се односи на кориснички интерфејс и степен у којем он испуњава различите хеуристике употребљивости (Nielsen, 1993). Кориснички интерфејс треба да обезбеди једноставан дијалог између система и корисника који је јасно изражен прецизним терминима (Shneiderman and Plaisant, 2010). Треба бити *конзистентан*, а потреба за памћењем функција од стране корисника мора бити сведена на минимум (Norman and Draper, 1986). Корисничке грешке се спречавају добрим дизајном, али када се догоде *поруке о грешкама* прате претходно поменуте смернице за дијалог. Такође, систем обавештава кориснике о сваком предузетом кораку путем одговарајућих *повратних информација*. Употребљивост у образовним технологијама анализирана је у различитим студијама (Ardito, 2004; Storey et al., 2002; Hossain, 2015; Issa and Jusoh, 2019; Silius, Tervakari and Pohjolainen, 2003). Ове студије успостављају смернице које гарантују употребљивост образовне технологије (дизајн и развој) и критеријуме за верификацију нивоа употребљивости ових образовних система.

Табела 8. Битни критеријуми за употребљивост интерактивног едукативног софтвера (Barker and King, 1993)

Категорија	**Опис**
Квалитет дизајна корисничког интерфејса	Истраживања показују да дизајнери најбоље оцењених производа прате добро утврђена правила и смернице. Овај аспект дизајна утиче на перцепцију корисника о производу (шта могу да ураде са њим и колико их може ангажовати).
Ангажовање	Одговарајуће коришћење аудио и покретних визуелних сегмената може у великој мери допринети мотивацији корисника за рад са медијумом.
Интерактивност	Учешће у партиципативним задацима у софтверском производу наводи корисника на размишљање.
Прилагодљивост	Софтверски производи који омогућавају корисницима да их конфигуришу и мењају, како би задовољили одређене индивидуалне потребе, доприносе квалитету образовног искуства.

Постоје бројни радови који анализирају критеријуме употребљивости образовних алата. Barker и King (1993) развили су метод за евалуацију интерактивних мултимедијалних курсева. Они пружају четири фактора за које њихово истраживање сугерише да су од кључног значаја за употребљивост софтверског производа (табела 8). Сматрају да је дизајн корисничког интерфејса веома битан критеријум, а ангажовање корисника у партиципативним задацима доприноси да софтверски систем буде смислен и да изазове размишљање.

Nielsen и Mack (1994) дефинисали су хеуристичку евалуацију која се бави битним питањима употребљивости. Критеријуми њихове хеуристичке евалуације образовног софтвера приказани су у табели 9. Предложени критеријуми представљају једну од популарних дефиниција употребљивости. *Контрола и слободна воља* корисника су веома битни критеријуми, јер могућност да у сваком тренутку корисници могу напустити локацију и пониште грешке дају додатни осећај задовољства.

Табела 9. Критеријуми хеуристичке евалуације образовног софтвера (Nielsen and Mack, 1994)

Категорија	**Опис**
Приказ статуса система	Софтвер обавештава корисника о статусу путем одговарајућих и правовремених повратних информација.
Усклађеност система и стварног света	Софтвер користи језик корисника, а не жаргон. Информације се појављују природним и логичним редоследом.
Контрола и осећај слободе	Корисници могу напустити локације и поништити грешке.
Конзистентност и стандарди	Корисници не морају да се питају да ли познате наредбе, ситуације или радње у датом софтверу значе исто. Прате се уобичајени стандарди оперативног система.
Спречавање грешака	Дизајн пружа смернице које смањују ризик од грешака корисника.
Препознавање уместо памћења	Објекти, акције и опције су видљиви. Корисник се не мора ослањати на памћење. Информације су видљиве или лако доступне кад год је то потребно.
Флексибилност и ефикасност употребе	Софтвер омогућава искусним корисницима да користе пречице и прилагођавају подешавања како им одговарају.
Естетски и минималистички дизајн	Софтвер пружа привлачан општи дизајн и не приказује небитне или ретко коришћене информације.
Помоћ за препознавање, дијагностику и опоравак од грешака	Поруке о грешкама су изражене једноставним језиком, јасно указују на проблем и препоручују решење.
Помоћ и документација	Софтвер пружа одговарајућу помоћ и документацију којој је лако приступити и која је повезана са потребама корисника.

Quinn (1996) сматра да постоји могућност да образовни софтвер не буде употребљив или да има висок степен употребљивости, али да ефекат учења који се постиже није задовољавајући. Системи намењени учењу морају бити и употребљиви и образовни и на основу тога он је предложио метод хеуристичке евалуације којом се процењује да ли се ради о образовном и употребљивом софтверу (табела 10). Систем мора да пружа *јасне циљеве и задатке*, студент треба да разуме акције које предузима и да зна шта добија њиховом употребом.

Табела 10. Метод хеуристичке евалуације употребљивости образовног софтвера (Quinn, 1996)

Категорија	**Опис**
Јасни циљеви и задаци	Студент треба да разуме шта може да постигне употребом софтвера.
Контекст са доменом и студентом	Активност треба да буде повезана са вежбом и да заинтересује студента.
Садржај је јасно и вишеструко представљен, са вишеструком навигацијом	Порука треба да буде недвосмислена, да подржава различите афинитете студената, и да омогући студенту да пронађе релевантне информације док је ангажован на некој активности.
Вођена партиципација	Потребна је подршка студенту како би се омогућио рад у оквиру компетенција.
Провера разумевања код студената	Студенти треба да артикулишу своја концептуална разумевања кроз одговарајуће повратне информације.
Формативна евалуација	Студентима је потребна конструктивна повратна информација о њиховим напредовању.
Учинак треба бити мерљив	Резултати треба да буду јасни и мерљиви.
Подршка за самостално учење	Систем треба да подржи преношење знања и ван предвиђеног окружења за учење и да олакша студенту се самостално усавршава.

Употребљивост технологије оријентисане на подучавање и учење зависи и од нивоа образовања за који је намењена, карактеристика студената и специфичности предмета (Molina, Fuentes-Cancell and García-Hernández, 2022). Стога је од суштинске важности да образовни софтверски системи буду *усклађени са литературом* и другим ресурсима како би ефикасније утицали на боље разумевање студената (Naps, Eagan and Norton, 2000; Kehoe, Stasko and Taylor, 2001). Да би се постигла једноставна интеграција алата, предавачима мора бити лако да креирају нове пропратне материјале или да комбинују постојеће ресурсе.

5.5 ТАКСОНОМИЈА ЦИЉЕВА ЗА ДИЗАЈН ОБРАЗОВНОГ СИСТЕМА COMVIS

На основу опсежне анализе техничких, педагошких и циљева употребљивости образовних софтверских система, може се предложити таксономија циљева прилагођена домену, то јест дизајну алата погодног за учење програмских преводилаца. У табели 11 приказана је класификација експлицитних циљева који дефинишу дизајн софтверског система ComVis. У овој табели се заправо резимирају карактеристике дизајна које су описане циљевима разматраним у овом поглављу.

Табела 11. Таксономија циљева за дизајн образовног алата ComVis

ВРСТЕ ЦИЉЕВА	ЦИЉЕВИ	ОЗНАКА
Технички	- Независност од хардвера и оперативног система	Т1
	- Минималистички дизајн	Т2
	- Могућност пријаве на систем	Т3
	- Лако надоградив	Т4
Педагошки	- Веће ангажовање	П1
	- Повећање мотивације	П2
	- Дубље разумевање садржаја	П3
	- Подршка различитим нивоима предзнања	П4
	- Индивидуални темпо учења	П5
	- Процена знања	П6
Употребљивости	- Једноставан за употребу	У1
	- Јасни циљеви и задаци	У2
	- Контрола и слободна воља	У3
	- Конзистентан кориснички интерфејс	У4
	- Повратне информације и помоћ	У5
	- Спречавање грешака	У6
	- Усклађеност са литературом	У7

6. COMVIS ОБРАЗОВНИ СОФТВЕРСКИ СИСТЕМ

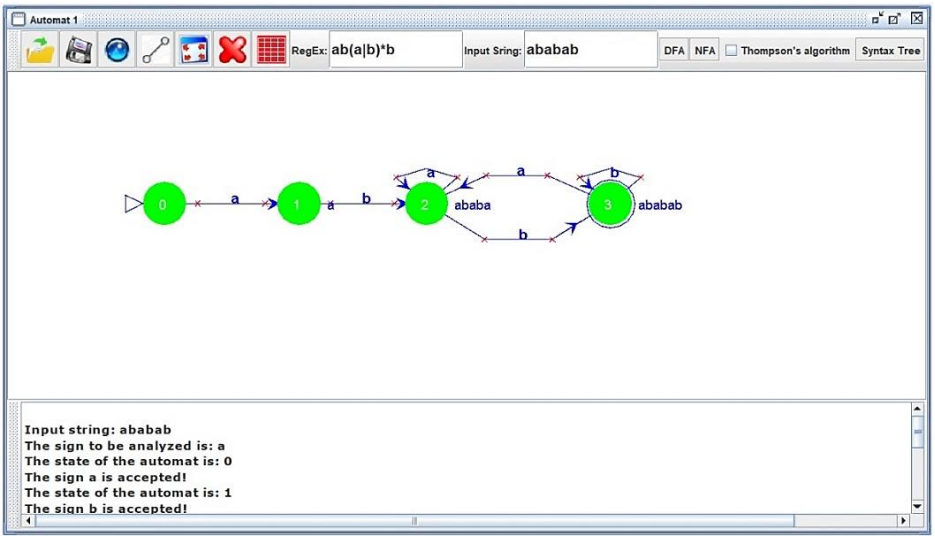
У овом поглављу је представљен образовни софтверски систем ComVis, развијен као алат за симулацију и визуелизацију алгоритама који се примењују у различитим фазама програмског превођења. Реализован је на основу предложене архитектуре и дизајна који заједно чине генерализовани модел за развој интерактивног образовног софтверског система. На почетку поглавља описан је почетак развоја првих алата у оквиру система ComVis, а затим је детаљније описана имплементација свих модула и функција система. С обзиром да аутори анализираних алата нису представили детаље имплементације подржаних функција, приликом развоја система ComVis нису се могла применити нека од добрих решења која су реализована у датим системима. Због тога су у увом поглављу приказани псеудо кодови имплементираних алгоритама у ComVis-у. Реализовани алгоритми су резултат рада аутора, као и препорука пронађених у литератури. На крају овог поглавља представљен је начин примене ComVis-а у настави.

6.1 ПОЧЕТАК РАЗВОЈА СИСТЕМА

Истраживање усмерено ка проналску ефикасног решења за унапређење наставе на предмету Програмски преводиоци почело је и пре успостављања коначног модела за развој новог система. Експериментисано је са постојећим системима који су анализирани у поглављу 4 и паралелно су развијани нови симулациони алати. Управо је развој тих првих алата и допринео формирању коначног модела.

Најпре је реализован алат ComVis Finite Automata (Jovanović, Stamenković and Miljković, 2020; Jovanović et al., 2020; Stamenković and Jovanović, 2021a) који представља едукативно окружење за учење основних принципа на којима се заснива рад коначних аутомата. Систем је реализован као интерактивна десктоп апликација на Јава програмском језику. Користећи овај софтвер, коначни аутомати се могу дефинисати цртањем у графичком уређивачу у

облику дијаграма стања или задавањем функције преласка помоћу табеле транзиција. Након конструисања могуће је покренути визуелну симулацију рада аутомата за произвољан улазни низ. Процес симулације коначног аутомата у овом едукативном окружењу врши се приказом стања за сваки карактер улазног низа (слика 51). Поред наведене функције ComVis Finite Automata студентима омогућава конверзију задатог регуларног израза у DFA или NFA и симулацију Томсоновог алгоритма.

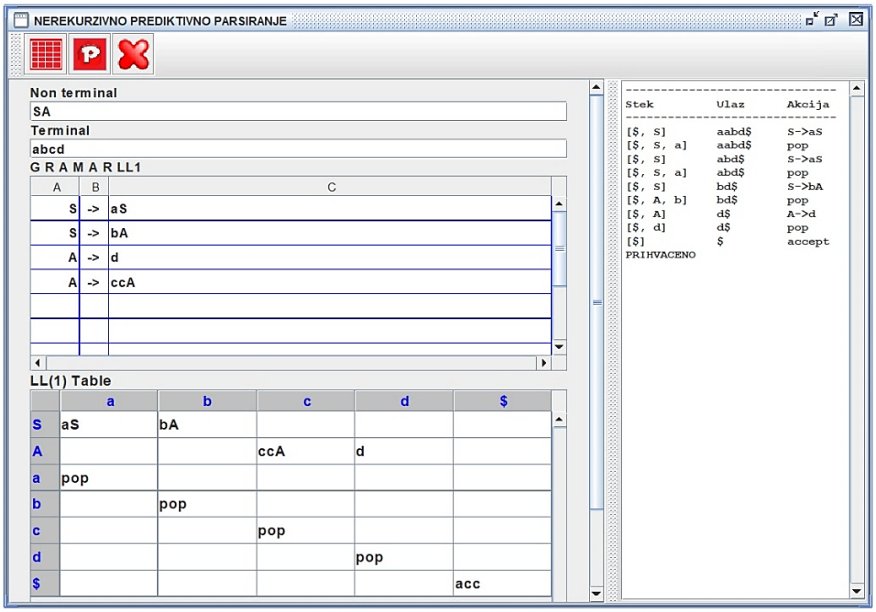


Слика 51. Пример симулације рада аутомата у симулационом софтверу ComVis Finite Automata (десктоп верзија)

Квантитативна анализа и евалуација ефикасности овог десктоп симулатора заснована на анкетирању студената основних студија на два универзитета, Факултет техничких наука Универзитета у Приштини и Department of Computer Science and Engineering, Netaji Subhas University of Technology New Delhi, показала је позитивне резултате (Jovanović et al., 2020). Студенти су открили да им је симулациони алат помогао да боље разумеју коначне аутомате и омогућио им да дизајнирају и компликованије DFA и NFA него што су то могли без њега, односно помоћу папира и оловке.

Због ових чињеница настављен је рад на развоју и надоградњи ComVis-а тако да је убрзо био проширен новим алатима: Scanner, NP Parser, LL(1) Parser, SR Parser, LR(0) Parser. Ови алати су представљени у (Jovanović et al., 2021).

Scanner има за циљ да студентима приближи процес лексичке анализе у процесу компајлирања. Кориснички интерфејс NP Parser-а са примером процеса парсирања за задату граматику и улазни стринг приказан је на слици 52. Симулационо окружење студентима омогућава израчунавање FIRST и FOLLOW скупова, као и симулирање процеса конструисања синтаксних табела LL(1) и LR(0).



Слика 52. Кориснички интерфејс NP Parser-а десктоп верзије ComVis-а, са примером процеса парсирања за задату граматику и улазни стринг

Процена ефикасности комплетног ComVis десктоп софтверског система презентована у (Jovanović et al., 2021) показује да употреба овог алата у наставном процесу директно утиче на смањење процента студената који у првом року не могу да положе испит. Предавачи су запазили да је код студената приметно повећана концентрација, као и интересовање за теме предмета. Коришћењем алата студенти су уочили и нека ограничења. Навели су као проблем то што је систем доступан само за Windows окружење, а као ограничење истакли су и недостатак модула семантичке анализе којим би се заокружио комплетан процес анализе. Узимајући у обзир повратне информације студената, али и предавача који су изнели слична запажања у погледу ограничења симулатора, дефинисани су циљеви за нову верзију система који су презентовани у претходном поглављу.

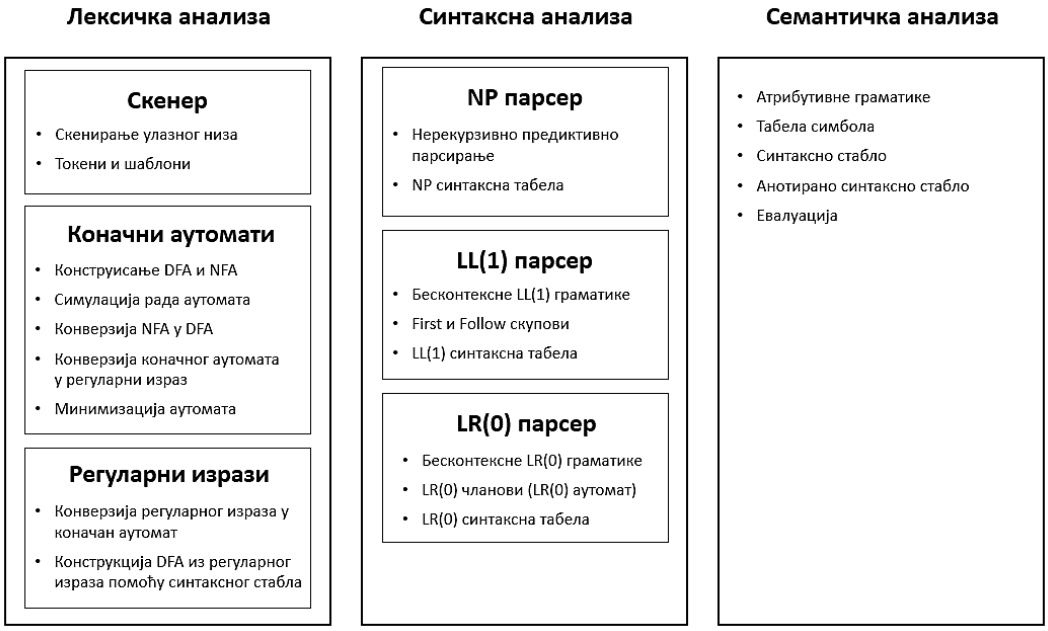
6.2 ОПИС СИСТЕМА COMVIS

Иако је процена покривености тема у (Jovanović et al., 2021) показала да је ComVis (десктоп верзија) већ достигао тренутно доступне алате, поред редизајна и унапређења функционалности за испуњавање постављених циљева тежња је била и развој додатних модула за покривање нових тема, пре свега модула семантичке анализе. Да би се испунио циљ T1 који се односи на независност система од платформе (хардвера и оперативног система), нова верзија ComVis-а је пројектована као веб базирана апликација. Да је овај циљ у потпуности оправдан говоре и претходне анализе које показују да су од свих алата (табела 4) само пет развијени као веб апликације: Seshat, DFA simulation tool, JCT, Webworks applets и SELFA-Pro. У време писања дисертације на званичним веб адресама ових система била су доступна само два Seshat и SELFA-Pro. И један и други систем имају мали степен покривености тема.

Пошто је ComVis десктоп верзија развијена на Јава програмском језику, за backend технологију веб верзије намеће се JSP (Java Server Pages). На овај начин већина алгоритама који су написани на Јави за десктоп апликацију искоришћени су и код веб базираног ComVis-а. Код десктоп апликације за представљање графичких елемената коришћена је Јава библиотека Swing. Графички кориснички интерфејс и сви елементи визуелизације су код веб апликације израђени помоћу следећих frontend технологија: HTML, JavaScript, CSS. Поред ових стандардних технологија, коришћен је и Grafviz – софтвер отвореног кода за визуелизацију графова (Gansner and North, 2000). Grafviz је омогућио да се структуирани текстуални резултати извршавања алгоритма представе прегледније у виду дијаграма и графова.

Графички приказ модула са припадајућим подмодулима и основним функционалностима дат је на слици 53. Детаљи имплементације свих модула описани су у наставку поглавља. Приликом описа модула и функција ComVis-а врши се и компарација са алатима JFLAP и Seshat. JFLAP је изабран као десктоп представник симулатора који је постигао широку примену на више светских универзитета (Rodger et al., 2009) и који је у погледу

функционалности и покривености показао најбоље резултате у спроведеној анализи (поглавље 4), а Seshat као један од најбољих представника веб базираних симулатора за учење компајлера, са највећом оценом код процене функционалности, који је доступан и у време писања дисертације.



Слика 53. Графички приказ модула са припадајућим подмодулима и основним функционалностима ComVis система

6.3 ИМПЛЕМЕНТАЦИЈА МОДУЛА ЗА УЧЕЊЕ ЛЕКСИЧКЕ АНАЛИЗЕ

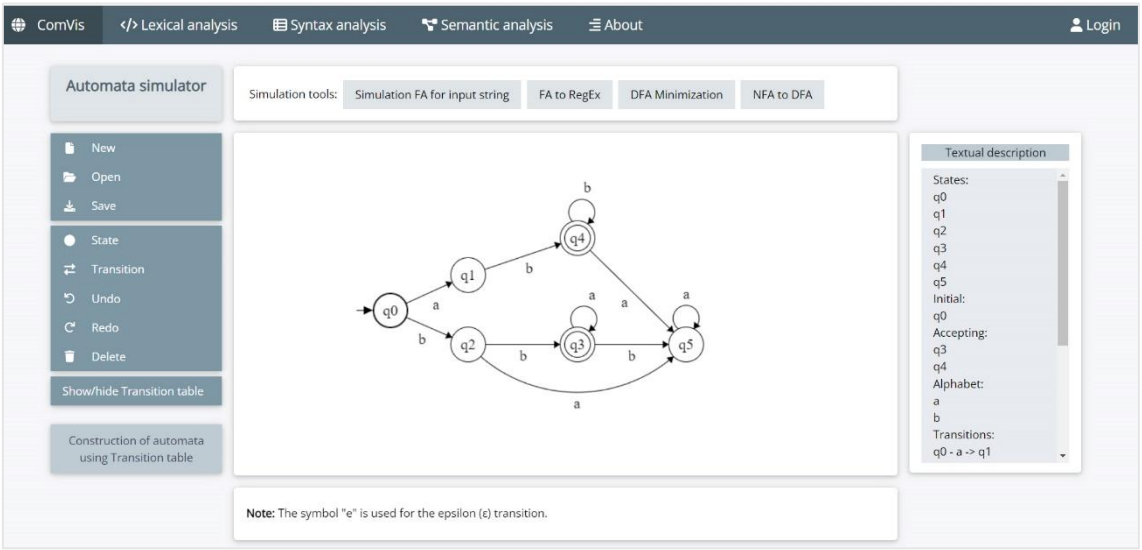
Модул за учење лексичке анализе састоји се од три независних подмодула: Finite Automata, Scanner и Regular Expression. Сваки од њих има своје корисничко окружење у оквиру посебне веб странице. Код претходне десктоп верзије за лексичку анализу су постојала два окружења, Scanner и Finite Automata.

6.3.1 Finite Automata (коначни аутомати)

Подмодул Finite Automata намењен је за учење теорије аутомата и регуларних граматика. Задржао је сличан графички интерфејс као код десктоп

верзије, али број функција, визуелних могућности и тема које се могу учити је сада знатно већи. Изглед корисничког интерфејса овог подмодула са примером једног конструисаног коначног аутомата приказан је на слици 54. Као што се може приметити испоштован је циљ Т2, дизајн интрфејса је минималистички без сувишних збуњујућих информација и елемената. На левој стрни налази се палета основних алата намењених конструисању коначних аутомата. Средњи и највећи део интерфејса заузима графички едитор, односно платно за цртање аутомата реализовано помоћу HTML Canvas елемента. На десној страни је монитор за приказ текстуалног описа аутомата који се аутоматски синхронизује са променама на платну. Изнад графичког едитора су постављене опције којима се покрећу одговарајуће симулације над конструисаним аутоматима. На овај начин улазни параметри за симулацију задају се графичким путем брзо и једноставно, што је у складу са постављеним циљевима У1, П1 и П2. Симулације које обезбеђује овај подмодул су следеће:

- симулација рада коначног аутомата за задати улазни стринг,
- симулација алгоритама за минимизацију DFA,
- симулација конверзије коначног аутомата у регуларни израз,
- симулација конверзије NFA у DFA.



Слика 54. Finite Automata: Кориснички интерфејс са примером конструисања коначног аутомата цртањем у графичком едитору

Поред сваке опције на палети алата постављен је и одговарајући симбол који ближе појашњава акцију (циљеви У1 и У2). У првом блоку палете налази се опција за отварање новог платна за цртање аутомата и опције за чување дизајнираних аутомата и отварање раније сачуваних. За имплементацију механизма који обезбеђује чување и учитавање коначних аутомата усвојен је JSON формат фајла који треба да садржи информације о чворовима и прелазима. Подаци о чвору су следећег облика:

```
{"x":294,"y":172,"text":"q1","isAcceptState":false,"isInitialState":true}
```

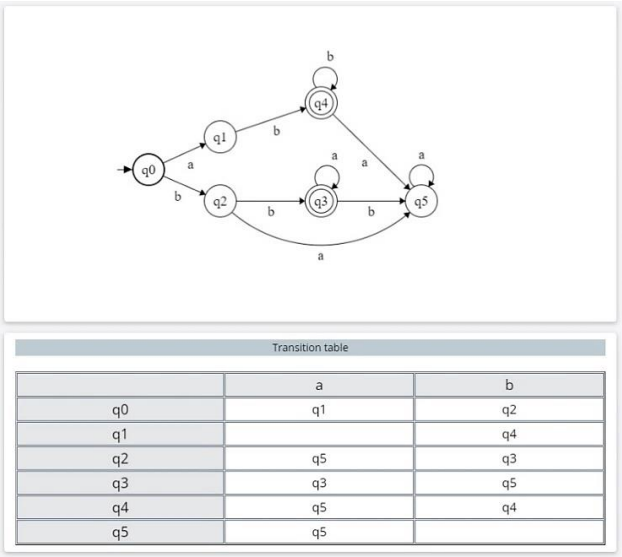
Чувају се координате чвора на Canvas-у, текст у оквиру чвора, односно ознака стања и информација о томе да ли дати чвор представља иницијално стање и/или стање прихватања. Основни подаци о транзицији су облика:

```
{"type":"Link","nodeA":0,"nodeB":1,"text":"a"}
```

Овде се чувају подаци о типу транзиције (Link или SelfLink), о чворовима који су повезани датим прелазом и лабели транзиције.

У другом блоку палете алата доступне су акције за цртање чворова и транзиција. Употребом HTML Canvas-а који представља контејнер за графичке објекте, чворови и трнзиције се додају у ходу применом одговарајућих функција написаних на JavaScript-у. Додатне опције у овом блоку су Undo, Redo и Delete. Помоћу ових опција корисник у сваком тренутку може да поништи грешке брисањем или враћањем на претходно стање. Начин рада при којем је сва контрола у рукама корисника, повећава задовољство и мотивацију, у складу са циљевима П2 и У3. Да би се нацртани чвор прогласио за иницијално или стање прихватања, користи се контекстни мени који се добија десним кликом на одговарајући објекат. Испод алата за цртање доступна је опција за аутоматско генерисање табеле прелаза. У сваком тренутку, овом опцијом може се приказати табела прелаза аутомата који се тренутно конструише на платну (слика 55). Приказ једног концепта на више различитих начина доприноси дубљем разумевању код студената (циљ П3).

Пошто је алат намењен студентима са различитим нивоима предзнања, опције за учитавање готових примера су од значаја за студенте са мањим предзнањем. На овај начин се пружа помоћ, приказ једноставних примера аутомата над којима се могу покренути симулације или се учитани аутомати могу додатно едитовати, што је у складу са постављеним циљевима П4 и У5.

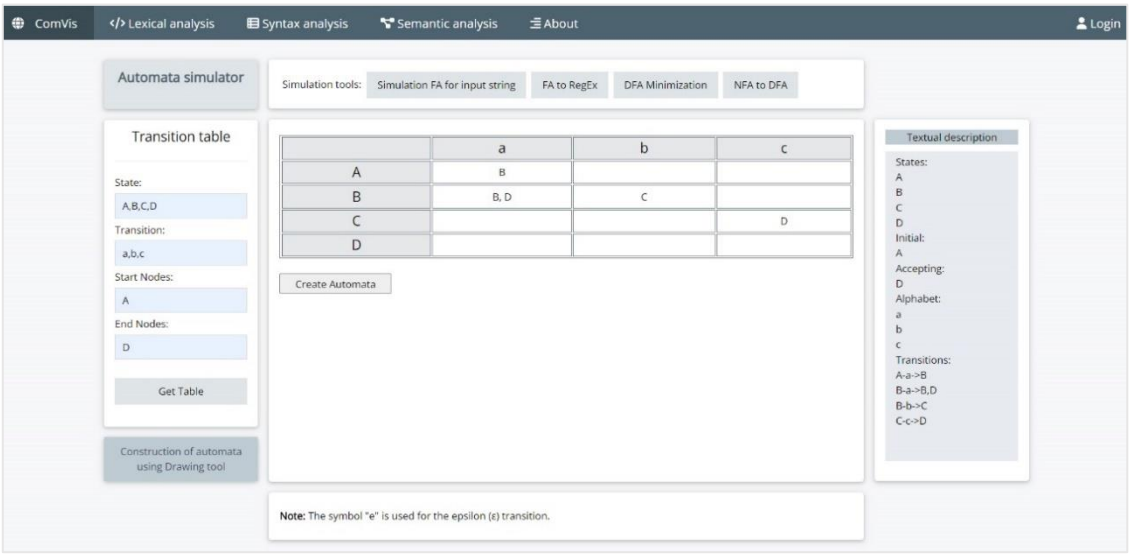


Transition table

	a	b
q0	q1	q2
q1		q4
q2	q5	q3
q3	q3	q5
q4	q5	q4
q5	q5	

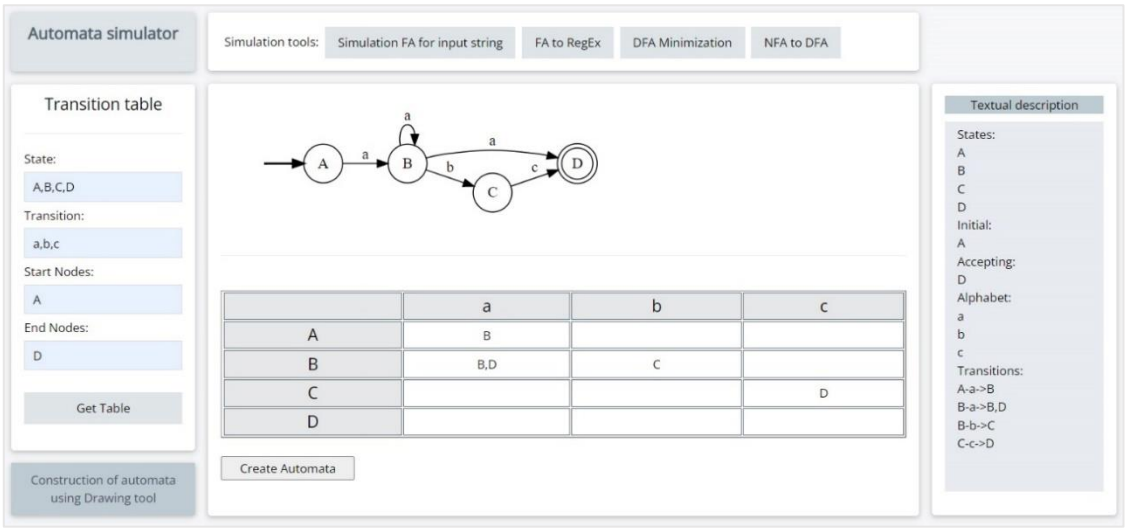
Слика 55. Приказ табеле прелаза коначног аутомата дефинисаног у оквиру графичког едитора

Поред могућности конструисања аутомата цртањем дијаграма стања на платну, доступан је и начин дефинисања путем табеле прелаза. Ова опција је јасно назначена и налази се испод палете алата. Избором оваквог начина за конструисање коначног аутомата отвара се ново радно окружење које је потпуно конзистентно са окружењем за цртање (циљ У4). На слици 56 приказан је аутомат који је дефинисан табелом прелаза. Као што се може видети, у окружењу за конструисање коначних аутомата путем табеле прелаза уместо палете алата за цртање на левој страни доступна је форма за унос основних података о аутомату (скуп стања, азбука, иницијално стање и стања прихватања). На основу задатих параметара, избором опције Get Table у пољу где се налазило платно за цртање појавиће се едитабилна табела (слика 56). Двоструким кликом на одговарајућем пољу табеле појављује се курсор и студент може да унесе стање аутомата, дефинишући на тај начин транзицију.



Слика 56. Finite Automata: Кориснички интерфејс са примером конструисања коначног аутомата дефинисањем табеле прелаза

Избором опције Create Automata која је доступна одмах испод табеле прелаза, биће приказан дијаграм стања дефинисаног аутомата (слика 57). Након завршене конструкције могу се покренути исте акције које су на располагању и у окружењу за цртање аутомата. Такође, на десној страни окружења и овде је присутан монитор за преглед текстуалног описа аутомата. Студент у сваком тренутку може да напусти ово окружење и врати се окружењу за цртање избором јасно обележене опције.

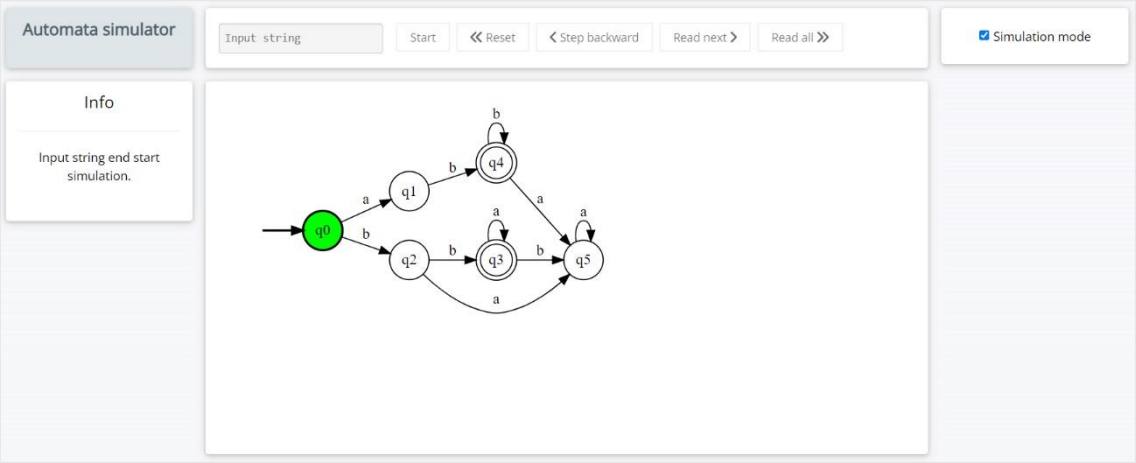


Слика 57. Приказ дијаграма стања коначног аутомата на основу дефинисане табеле прелаза

Поређењем ComVis система са алатима JFLAP и Seshat може се приметити да сва три обезбеђују могућност конструисања аутомата цртањем на платну, с тим што Seshat нуди оскудне могућности у односу на друга два. Конструкција коначних аутомата применом табеле транзиција је једино могућа код система ComVis.

6.3.1.1 Симулација рада коначног аутомата за задати улаз

Након конструисања аутомата ComVis обезбеђује студенту могућност да унесе жељени улазни стринг и прати рад аутомата. Код десктоп верзије ComVis-а симулација рада аутомата се приказивала корак по корак, али без могућности да на ток симулације утиче корисник. У новој веб верзији симулацију у потпуности контролише корисник. Изглед симулационог мода приказан је на слици 58.

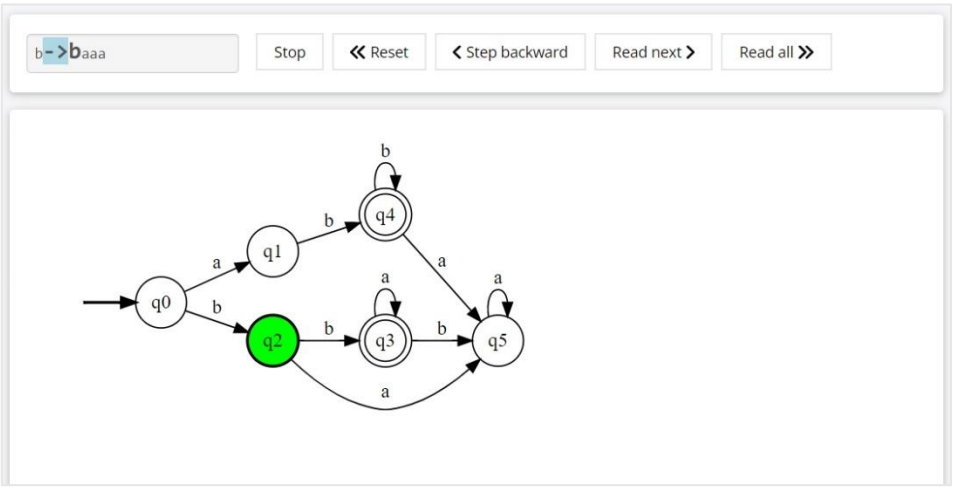


Слика 58. Finite Automata: Симулациони мод

На левој страни корисничког интерфејса у симулационом моду постављен је инфо блок (у складу са циљем У5) који најпре обавештава корисника да треба да унесе улазни стринг и онда започне процес симулације. У горњем делу интерфејса налази се поље за унос стринга и опције за контролу симулације (циљеви У1 и У3). Веома важно је да улазни стринг буде састављен од карактера из дефинисане азбуке. Пошто је као један од циљева употребљивости система наведена и могућност спречавања грешака (У6),

ComVis обезбеђује функцију која при уносу сваког карактера одмах врши проверу припадања азбуци. У случају погрешног уноса у инфо блоку се исписује порука о насталој грешци, као и информација о позицији на којој се налази погрешан карактер.

Симулација рада коначног аутомата се приказује у централном делу корисничког интерфејса. Опцијом Read all студент може одмах да испита да ли конструисани аутомат прихвата улазни стринг. За дубље разумевање препоручује се симулација у корацима. У овом случају за сваки карактер се прати покушај аутомата да обради улазни низ. Следећи карактер који се анализира се посебно истиче, а стање аутомата у датом кораку означава се зеленом бојом. На слици 59 може се видети да се аутомат налази у стању q2 и да је прочитан први карактер, док је следећи карактер b означен стрелицом и истакнут већим фонтом. Уколико је задњи карактер низа довео аутомат у финално стање, задати стринг је прихваћен. У оквиру инфо блока студент на крају симулације добија информацију о томе да ли је улазни низ препознат или не. Битно је да се нагласи да студент има потпуну контролу над симулацијом тако што, поред кретања унапред, постоји и могућност враћања симулације уназад или ресетовања на први карактер стринга. Корисник у сваком тренутку може да изађе из симулационог мода помоћу поља за чекирање које се налази на десној страни корисничког интерфејса (слика 58).

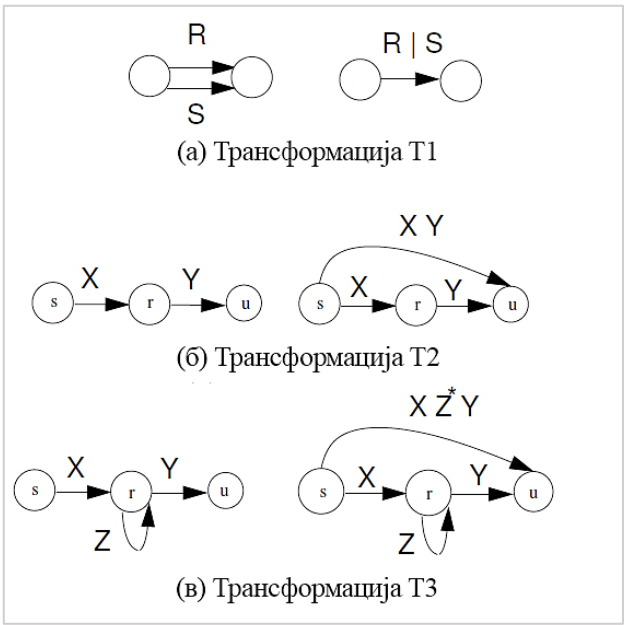


Слика 59. Пример симулације операције аутомата за задати улаз

Seshat не пружа могућност симулације рада коначног аутомата за одговарајући улазни стринг. JFLAP обезбеђује конструисање и симулацију рада аутомата на сличан начин као и ComVis.

6.3.1.2 Симулација процеса конверзије коначног аутомата у регуларни израз

ComVis за симулацију процеса конверзије коначног аутомата у регуларни израз имплементира алгоритам који се заснива на методи елиминације стања. Функционише тако што уклања једно по једно стање аутомата при чему у сваком кораку добијени коначни аутомат мора бити еквивалентан оригиналном. То се постиже формирањем нове транзиције акумулирањем прелаза који су водили до уклоњеног стања у одговарајући регуларни израз. Основне трансформације T1, T2 и T3 које алгоритам користи за прогресивно упрошћавање коначног аутомата приказане су на слици 60. Уклањање стања се наставља све док се не добије аутомат са једним прелазом из почетног стања у једно стање прихватања.



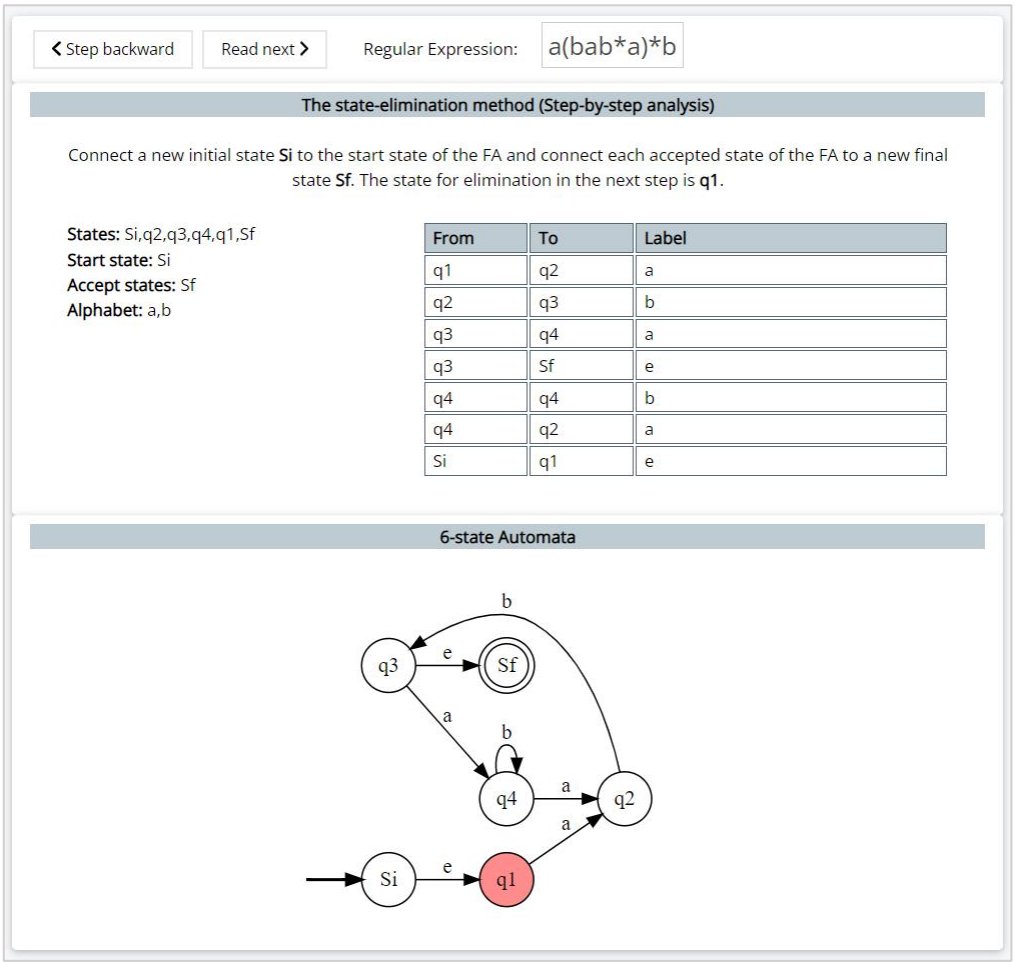
Слика 60. Трансформације T1, T2 и T3 које се примењују у методи елиминације стања (Fischer, Cytron and LeBlanc Jr, 2010)

Као што се са слике 60 може видети, сваки претходник и следбеник неког стања могу бити повезани коришћењем трансформација T1, T2 или T3. У том случају посматрано стање се заобилази новом транзицијом и једноставно се може уклонити. Код 1 приказује псеудо код функције FindRE() која имплементира ову методу конверзије коначног аутомата у регуларни израз. Управо се овај алгоритам користи у систему ComVis.

```
function FindRE(N) returns RegExpr
  OrigStates ← N.States
  call Augment(N)
  foreach s ∈ OrigStates do                                          // 1
    call Eliminate(s)
    N.States ← N.States − {s}                                        // 2
  /* vraća regularni izraz koji se odnosi na jedinu preostalu tranziciju */
end
procedure Eliminate(s)
  foreach (x,y)∈ N.States × N.States | CountTrans(x,y) > 1 do        // 3
  /* Primena transformacije T1 na x i y */
  foreach p ∈ Preds(s) | p ≠ s do
    foreach u ∈ Succs(s) | u ≠ s do
      if CountTrans(s,s) = 0
      then /* Primena transformacije T2 na p,s i u */
      else /* Primena transformacije T3 na p,s i u */
end
function CountTrans(x,y) returns Integer
  return (broj tranzicija sa x na y)
end
function Preds(s) returns Set
  return ({p|(∃a)(N.T(p,a)=s)})
end
function Succs(s) returns Set
  return ({u|(∃a)(N.T(s,a)=u)})
end
procedure Augment(N)
  OldStart ← N.StartState
  NewStart ← NewState()
  /* Definisati N.T(NewStart, λ)={OldStart} */
  N.StartState ← NewStart
  OldAccepts ← N.AcceptStates
  NewAccept ← NewState()
  foreach s ∈ OldAccepts do
  /* Definisati N.T(s,λ)={NewAccept} */
  N.AcceptStates ← {NewAccept}
end
```

Код 1. Псеудо код алгоритма који се заснива на методи елиминације стања (Fischer, Cytron and LeBlanc Jr, 2010)

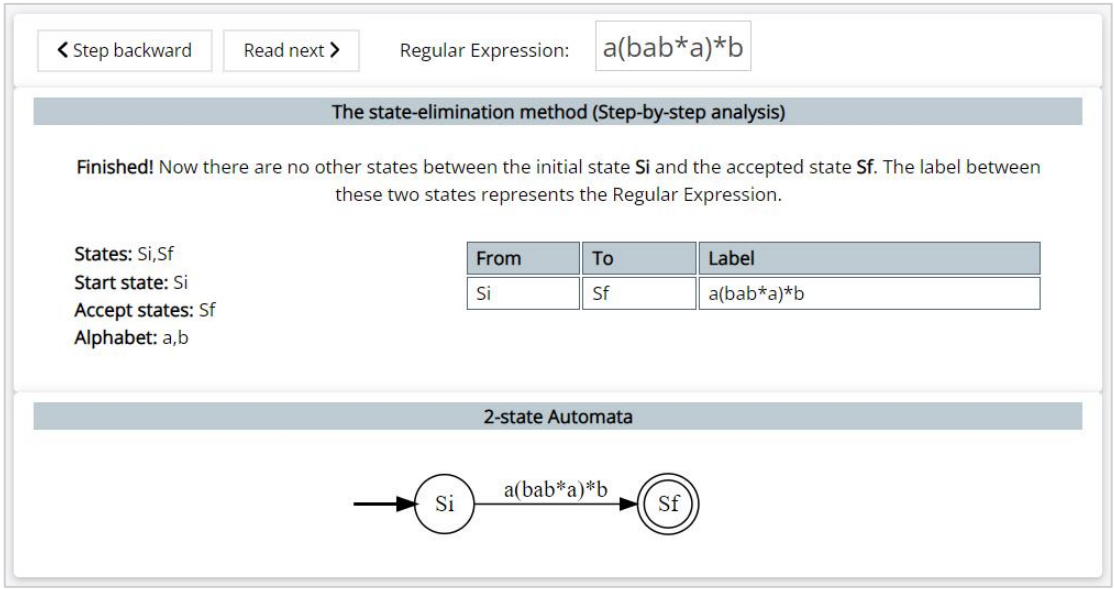
Дати алгоритам почиње позивањем функције Augment() која уводи ново иницијално и ново стање прихватања. Петља означена бројем 1 врши сукцесивно елиминацију сваког стања *s* коначног аутомата позивањем функције Eliminate(s). Трансформација T1 обезбеђује да сваки пар стања буде повезан са највише једним прелазом. Тај део кода је означен бројем 3. Затим се трансформишу сви прелази који воде од и ка стању *s* применом трансформација T2 или T3. Онда следи уклањање стања *s* у делу кода који је означен бројем 2. Када се алгоритам заврши, једина преостала стања су NewStart и NewAccept које је увела функција Augment(). Ознака транзиције између ова два стања представља коначни резултат рада алгоритма, у питању је регуларни израз који је еквивалентан полазном коначном аутомату.



Слика 61. FA to RegEx: Обележавање стања које се елиминише у следећем кораку симулације

За покретање симулације процеса конвертовања аутомата у регуларни израз у оквиру система ComVis бира се опција *FA to RegEx*. Уколико је у графичком едитору конструисан аутомат без иницијалног или стања прихватања, систем за спречавање грешака неће дозволити покретање симулације (циљ У6), већ ће о томе обавестити корисника (циљ У5). На почетку симулације се додају ново иницијално и ново стање прихватања коначном аутомату. Такође, у сваком кораку исписује се одговарајуће објашњење предузетих акција (циљ У5), као и најава следећег стања које ће бити елиминисано у наредном кораку (циљ У2), што се може видети на слици 61. Пошто примењени алгоритам врши елиминисање стања у итеративном поступку, симулација у систему ComVis подразумева приказ информација о сваком кораку петље означене бројем 1 у коду 1 (циљеви П3 и У3).

У задњем кораку петље алгоритма остају само два стања са једном транзицијом као на слици 62. На овај начин презентовања алгоритма студентима су на располагању све информације о свакој итерацији уз могућност контроле тока симулације, што може допринети дубљем разумевању презентоване методе (циљ П3), а самим тим и већој мотивацији за даље учење (циљ П2).



Слика 62. FA to RegEx: Приказ коначних резултата алгоритма за конверзију аутомата у регуларни израз

Треба напоменути да JFLAP обезбеђује симулацију процеса конверзије аутомата у регуларни израз, такође методом елиминације стања, док Seshat не подржава овакву могућност.

6.3.1.3 Симулација Hopcroft-овог алгоритма за минимизацију DFA

У софтверском систему ComVis имплементиран је метод минимизације коначних аутомата познат као техника уситњења партиција. Један од најпопуларнијих и најбржих алгоритама који минимизацију аутомата врши методом партиционисања је Hopcroft-ов алгоритам (Al-Zobaidi, 2014), детаљније описан у теоријском делу дисертације. За писање овог алгоритма на Јава програмском језику послужиле су смернице аутора Yingjie (2009) које су дате у виду псеудо кода (код 2).

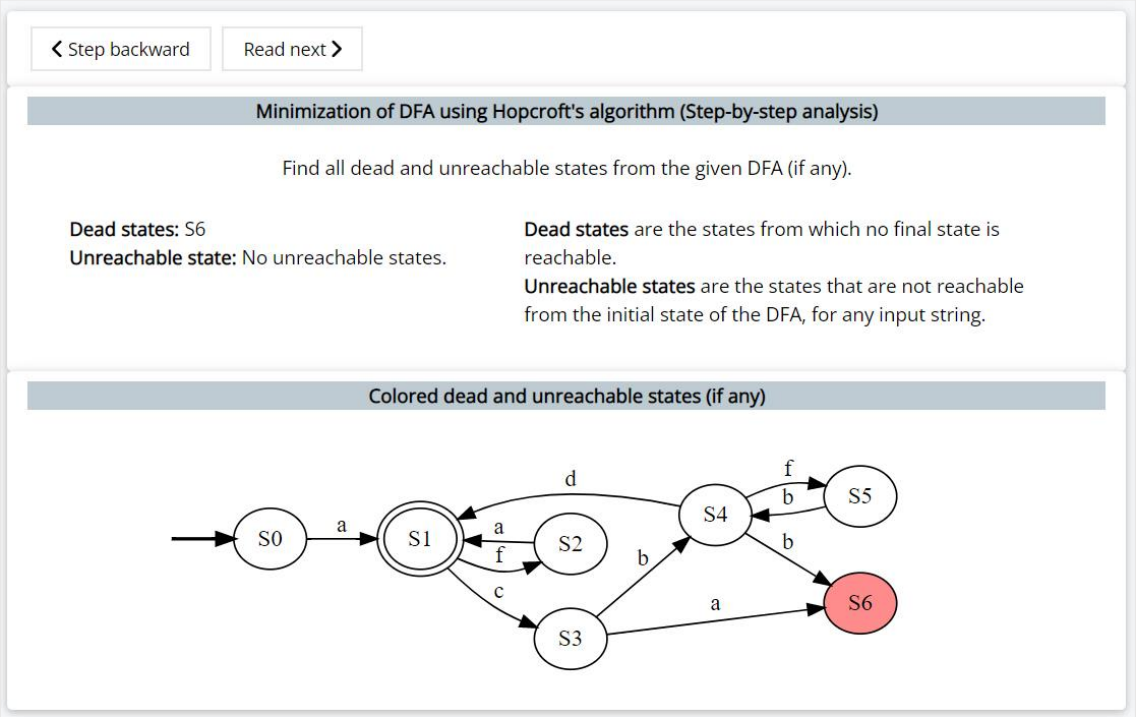
```
W ← {F,Q − F}
P ← {F,Q − F}
while W is not empty do
  select and remove S from W
  for all a ∈ Σ do
    l_a ← δ^-1(S,a)
    for all R in P such that R ∩ l_a ≠ ∅ and R ⊈ l_a do
      partition R into R1 and R2: R1 ← R ∩ l_a and R2 ← R − R1
      replace R in P with R1 and R2
      if R ∈ W then
        replace R in W with R1 and R2
      else
        if |R1| ≤ |R2| then
          add R1 to W
        else
          add R2 to W
        end if
      end if
    end for
  end for
end while
```

Код 2. Псеудо код Hopcroft-овог алгоритма (Yingjie, 2009)

Ако аутомат садржи два еквивалентна стања то значи да је једно од њих сувишно и да се може елиминисати. Као резултат тога, број стања аутомата се

смањује. Hopcroft-ов алгоритам се управо заснива на проналажењу класа еквиваленције. Стања су подељена у дисјунктне скупове (партиције) тако да сваки скуп садржи само стања која се понашају слично у кореспонденцији са улазима. На почетку алгоритма процес минимизације почиње само са две партиције, једна садржи сва коначна стања { F }, а друга се састоји од свих преосталих стања { $Q - F$ }. Да би се дошло до коначног броја партиција, алгоритам примењује итеративни поступак у више корака у којима се врши процес „пречишћавања“ партиција. У сваком кораку алгоритма постоји радна листа W у којој се смешта партиција која се испитује. Испитивање се врши тако што се прати понашање стања из партиције за сваки симбол азбуке. Уколико се установи да партиција садржи стања која нису еквивалентна, долази до њиховог одвајања у новој партицији. Поступак се понавља све док не преостану партиције састављене само од еквивалентних стања које се не могу више делити. Тада се операција минимизације завршава. Број стања у минимизираном DFA одговара коначном броју партиција. У резултујућем минимизираном аутомату сва стања која се налазе у једној партицији биће спојена у једно стање.

Када се заврши са конструкцијом коначног аутомата, симулација процеса минимизације се покреће из менија симулационих алата избором опције *DFA minimization*. У случају да конструисани аутомат није DFA типа, систем за спречавање грешака ће обавестити корисника о томе (циљ У6). Студенти имају могућност праћења симулације корак по корак, уз могућност враћања уназад (циљеви У3, П1 и П2). У сваком кораку постоји и одговарајуће објашњење акције која се у том тренутку спроводи (циљеви П3 и У5). У првом кораку студенту се пружају основне информације о процесу минимизације аутомата. У следећем кораку траже се сва недостижна и мртва стања која немају утицај на језик који дати DFA препознаје. Такође, студенту су у овом кораку на располагању дефиниције ових стања. Уколико постоје оваква стања она се обележавају посебном бојом (слика 63) и у наредном кораку симулације се бришу.

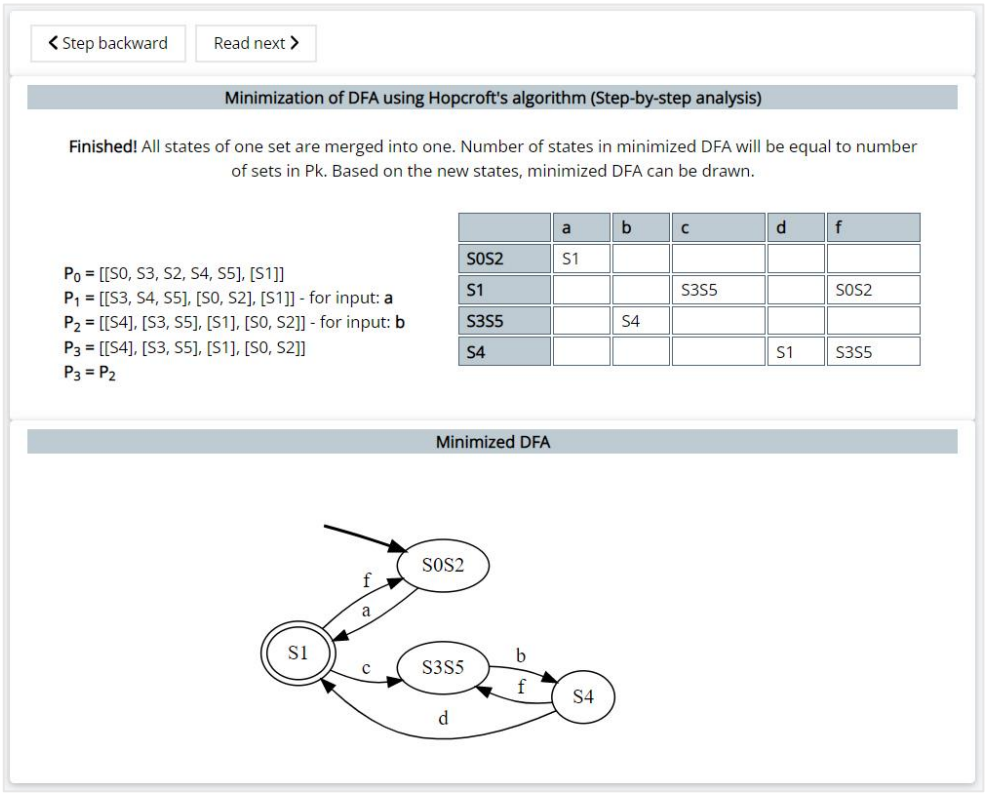


Слика 63. DFA minimization: Обележавање недостижних и мртвих стања

Пошто се Hopcroft-ов алгоритам извршава у итерацијама у којима се испитује еквиваленција стања у партицијама, од великог значаја за дубље разумевање процеса минимизације је и приказ предузетих акција у свакој итерацији (циљеви П3 и П4), па се у наредним корацима симулације студентима најпре приказује тебела прелаза како би се лакше пратио поступак формирања партиција, а затим се истиче свака партиција која се дели, као и симбол који је иницирао њену поделу. На крају овог процеса презентује се новоформирани аутомат са минималним бројем стања, као и његова табела прелаза (слика 64).

У складу са дефинисаном архитектуром ComVis система, резултати рада алгоритма шаљу се у текстуалном облику интерпретатору улазно-излазних података (слика 50) где се пакују у тачно дефинисаном облику и шаљу подсистему за интеракцију са корисником. На пример, пакет података за резултујући минимални аутомат са слике 64 је следећи:

```
minAutomata:S0-a->S1,S1-f->S0,S1-c->S3,S3-b->S4,S4-d->S1,S4-f->S3
```



Слика 64. DFA minimization: Приказ коначних резултата Hopcroft-овог алгоритма у задњем кораку симулације

Интерпретатор улазно-излазних података на корисничкој страни преузима информације из овог пакета и обликује их у формат погодан за приказ помоћу визуелизатора. Овај облик прописује GrafViz који улазне податке претвара у граф стања. Формат података који се шаљу визуелизатору може се видети у коду 3.

```
digraph finite_state_machine {
    rankdir=LR;
    node [width = 0.4];
    node [peripheries = 2]; S1;
    node [width = 0.5];
    node [peripheries = 1];
    secret_node [style=invis, shape=point];
    secret_node -> S0S2 [style=bold];
    S0S2 -> S1 [label = "a"];
    S1 -> S0S2 [label = "f"];
    S1 -> S3S5 [label = "c"];
    S3S5 -> S4 [label = "b"];
    S4 -> S1 [label = "d"];
    S4 -> S3S5 [label = "f"];
}
```

Код 3. Очекивани формат података од стране визуелизатора

Функцију минимизације коначних аутомата имају и JFLAP и Seshat с тим што се код Seshat-а не врши симулација у корацима, већ се одмах приказује табела транзиција минималног DFA без приказа дијаграма стања.

6.3.1.4 Симулација конверзије NFA у DFA применом алгоритма за конструкцију подскупова

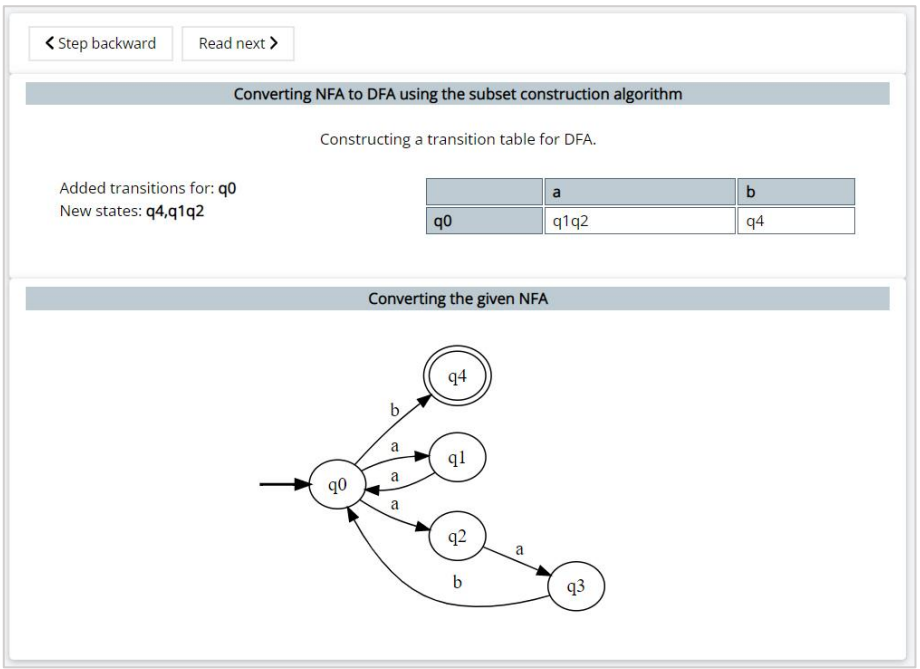
Трансформација NFA у еквивалентни DFA у образовном систему ComVis врши се применом алгоритма за конструкцију подскупова. Псеудо код имплементираног алгоритма дат је у коду 4. Конструкција подскупова узима као улаз NFA ($N, \Sigma, \delta_N, n_0, N_A$), а као резултат даје DFA ($D, \Sigma, \delta_D, d_0, D_A$). Оба аутомата, NFA и DFA, користе исту азбуку Σ. Конструкција подскупова подразумева извођење скупа DFA стања D из скупа NFA стања N и извођење DFA прелазних функција δ_D. Почетно стање DFA d_0 и стања прихватања D_A ће такође произаћи из конструкције.

```
q0 ← ε-closure({n0});
Q ← q0;
WorkList ← {q0};
while (WorkList ≠ ∅) do
   remove q from WorkList;
   for each character c ∈ Σ do
      t ← ε-closure(Delta(q,c));
      T[q,c] ← t;
      if t ∉ Q then
         add t to Q and to WorkList;
      end;
end;
```

Код 4. Псеудо код алгоритма за конструкцију подскупова (Cooper and Torczon, 2012)

Алгоритам формира скупове Q пратећи прелазе које NFA може да направи за сваки симбол азбуке. Сваки подскуп q садржи одговарајућа стања NFA. Алгоритам креће са почетним скупом q_0, који садржи n_0 и сва стања NFA која су доступна из почетног стања ε-прелазима. Та стања су еквивалентна јер се могу достићи без задавања улазног симбола. Да би конструисао q_0 из n_0, алгоритам израчунава ε-*closure*(n_0). Иницијално, Q има само једног члана q_0

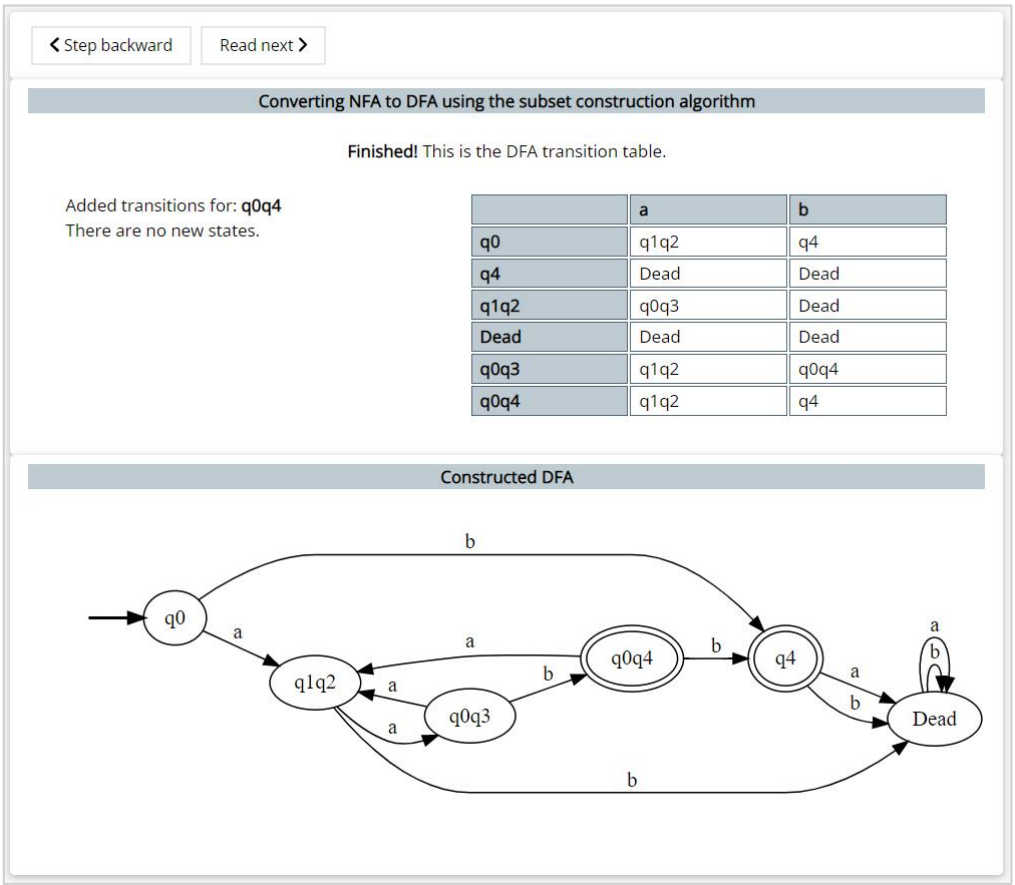
који се додаје радној листи (WorkList). Након тога алгоритам покреће *while* петљу у којој се најпре из радне листе уклања подскуп *q*. Конструисање нових подскупова се врши тако што се за сваки симбол *c* азбуке *Σ* прате стања која би NFA достигао из подскупа *q*. За ово израчунавање у алгоритму користи се функција *Delta*(*q,c*). Уколико се из подскупа *q* за исти симбол *c* достижу различита стања NFA, онда та стања формирају нови подскуп *t*. Транзиције између подскупа *q* и *t* се чувају у табели прелаза *T*. Новоформирани подскупови се додају скупу *Q* и радној листи. Петља *while* уклања следећи подскуп *q* из радне листе и прати његово понашање за симболе азбуке, на основу чега могу настати нови подскупови. Ова петља се понавља све док у радној листи има подскупова *q*. На крају алгоритма сачувани су сви формирани подскупови у *Q,* од којих q_0 представља иницијално стање DFA, а стања прихватања DFA су сви подскупови који садрже бар једно NFA стање прихватања. У систему ComVis симулација процеса трансформације NFA у DFA покреће се избором опције *NFA to DFA*. Интерфејс за праћење симулације приказан је на слици 65. Као што се може запазити, све врсте симулације доступне из модула Finite Automata обезбеђују конзистентан кориснички интерфејс чиме је испуњен циљ У4.



Слика 65. NFA to DFA: Конструисање подскупова из почетног стања

Покретањем симулације студенти ће на првом кораку добити објашњење алгоритма који се симулира. У следећем кораку се објашњава поступак креирања новог иницијалног DFA стања. Након тога приказују се информације из сваке итерације алгоритма. На слици 65 приказана је конструкција подскупова анализом из почетног стања. Студент контролише процес симулације опцијама у горњем делу корисничког интерфејса. У сваком следећем кораку врши се анализа из новоформираних подскупова стања. На слици 65 назначено је да се у следећем кораку анализирају прелази из скупова q4 и q1q2. Задњи корак симулације (слика 66) приказује табелу транзиција и дијаграм стања добијеног DFA.

Поређењем ComVis-а са референтним алатима JFLAP и Seshat може се закључити да сви подржавају симулацију конвертовања NFA у DFA, али са нешто другачијим приступом.

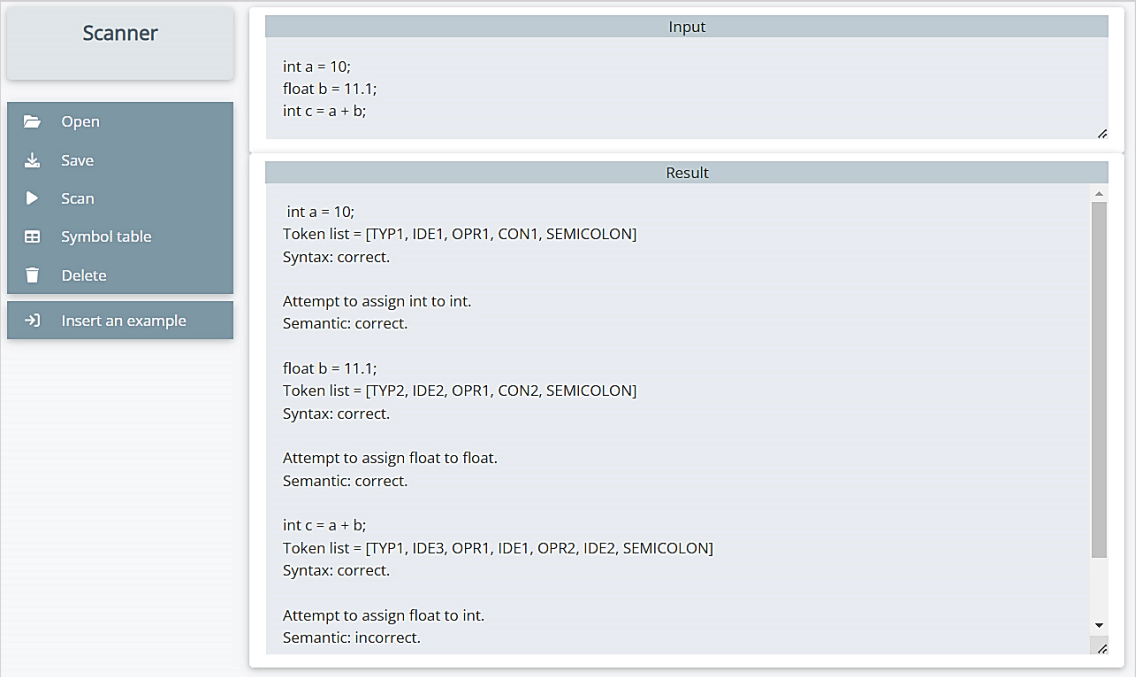


	a	b
q0	q1q2	q4
q4	Dead	Dead
q1q2	q0q3	Dead
Dead	Dead	Dead
q0q3	q1q2	q0q4
q0q4	q1q2	q4

Слика 66. NFA to DFA: Приказ коначних резултата алгоритма за конструкцију подскупова, задњи корак симулације

6.3.2 Scanner (лексички анализатор)

Развоју подмодула Scanner допринели су претходно развијени анализатори описани у (Stamenković and Jovanović, 2021b; Jovanović et al., 2022). У оквиру подмодула Scanner студент добија прилику да се детаљније упозна са процесом лексичке анализе. Кориснички интерфејс је једноставан са елементима који су конципирани на сличан начин као и код осталих подмодула, а у складу са циљем У4 (слика 67). Са леве стране је на располагању палета алата, горњи део централног интерфејса је намењен уносу програмских наредби које ће бити скениране лексичким анализатором, док је у доњем делу централног интерфејса смештен монитор за праћење резултата анализе. Поред директног укуцавања, студент улазни програмски код може да зада и учитавањем из текстуалног документа. Такође, откуцане наредбе кода у сваком тренутку може да извезе, односно сачува за каснију употребу. Дефиниција граматике која је подржана лексичким анализатором приказана је на слици 68.



Слика 67. Scanner: Лексичка анализа задатих наредби

```
linijaKoda -> LBRACE deklaracija RBRACE
linijaKoda -> TYP1 IDE OPR1 izraz SEMICOLON
linijaKoda -> TYP2 IDE OPR1 izraz SEMICOLON
linijaKoda -> PRINT IDE SEMICOLON
linijaKoda -> izraz SEMICOLON
deklaracija -> TYP1 identifikator
deklaracija -> TYP2 identifikator
identifikator -> IDE
identifikator -> IDE COMMA identifikator
izraz -> terminal
izraz -> izraz OPR1 izraz
izraz -> izraz OPR2 izraz
izraz -> izraz OPR3 izraz
izraz -> izraz OPR4 izraz
izraz -> izraz OPR5 izraz
izraz -> LPAREN izraz RPAREN
izraz -> LPAREN izraz RPAREN OPR2 izraz
izraz -> LPAREN izraz RPAREN OPR3 izraz
izraz -> LPAREN izraz RPAREN OPR4 izraz
izraz -> LPAREN izraz RPAREN OPR5 izraz
terminal -> IDE
terminal -> CON
```

Слика 68. Дефиниција граматике подржане лексичким анализатором

Након уноса програмских наредби за анализу и притиском на дугме Scan, почиње процес анализе. Scanner чита изворни код из текстуалног поља Input карактер по карактер, при чему врши идентификацију лексичких јединица - лексема. За сваку лексему се везује одговарајући токен. Токен се може дефинисати низом знакова, односно шаблоном. Шаблони се конструишу помоћу регуларних израза. Они дефинишу правила тако да свака лексема буде препозната као валидни токен, на пример шаблон за дефинисање идентификатора може бити следећи:

```
[A-Za-z_][A-Za-z0-9]*
```

Алгоритам који имплементира ову функцију у ComVis-у заснива се на приступу читања једног карактера унапред. Лексички анализатор чита унапред само када мора. На пример, оператер попут „*“ може се идентификовати без читања унапред. Лексички анализатор чита унапред док прикупља цифре за препознавање константи или знакове за препознавање идентификатора. Када се у улазном току појави низ цифара, лексички анализатор прослеђује парсеру токен CON, док се у табели симбола поред

токена чува вредност броја заједно са атрибутом. На пример, код 5 приказује псеудо код алгоритма који је задужен за препознавање целобројних вредности (*int*). Променљива *peek* чува следећи улазни карактер. Вредност целобројне константе се акумулира у променљивој *v* .

```
if ( peek holds a digit ) {
  v = 0;
  do {
    v = v * 10 + integer value of digit peek;
    peek = next input character;
  } while ( peek holds a digit ) ;
  return token [CON, v];
}
```

Код 5. Псеудо код алгоритма за препознавање целобројних вредности

Кључне речи генерално задовољавају правила за формирање идентификатора, па је потребан механизам за одлучивање када лексема формира кључну реч, а када идентификатор. Проблем се решава ако су кључне речи резервисане, то јест ако се не могу користити као идентификатори. Дакле, низ знакова формира идентификатор само ако није препознат као кључна реч. Ова функција се имплементира иницијализацијом табеле стрингова са резервисаним речима и њиховим одговарајућим токенима. Када лексички анализатор прочита лексему која би могла да формира идентификатор, прво проверава да ли се лексема налази у табели стрингова. Ако је тако, враћа токен из табеле, у супротном враћа токен IDE. Псеудо код за препознавање кључних речи и идентификатора приказан је у коду 6.

```
if ( peek holds a letter ) {
  collect letters or digits into a buffer b;
  s = string formed from the characters in b;
  w = token returned by words.get(s);
  if ( w is not null ) return w;
  else {
    return token [IDE, s];
  }
}
```

Код 6. Псеудо код за препознавање кључних речи и идентификатора

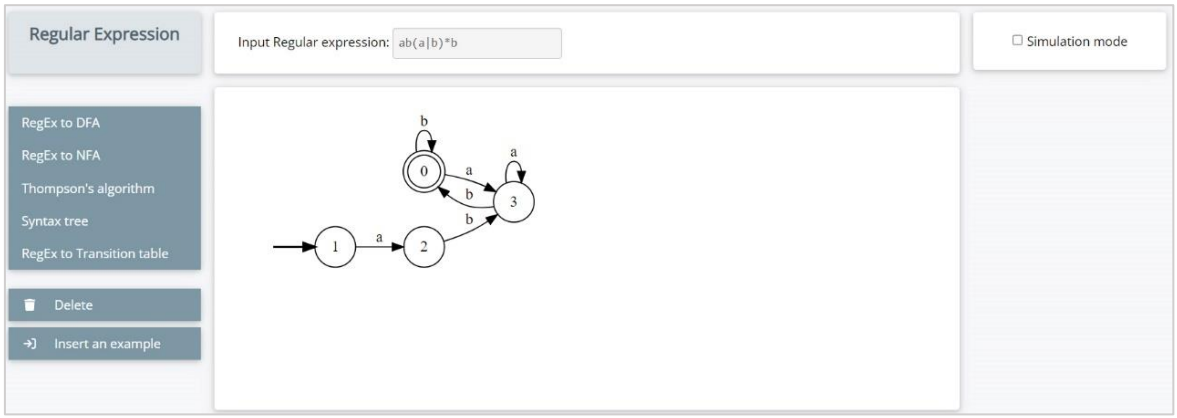
Овај алгоритам (код 6) узима са улаза стринг *s* који се састоји од слова и бројева и који почиње словом на основу горе дефинисаног шаблона. Лексички анализатор ће наставити да чита карактере са улаза све док наилази на слова и цифре. Када се наиђе на неки други карактер, на пример размак лексема се копира у бафер `b`. Ако у табели стрингова постоји *s*, онда се враћа токен који је преузет помоћу `words.get(s)`. У супротном, алгоритам враћа токен IDE и лексему *s*.

Код десктоп верзије ComVis-а студент је могао да види само генерисани низ токена. Код веб верзије овај модул је унапређен. Сваки ред улазног кода се посебно анализира. Препознати токени се даље шаљу у парсер на анализу синтаксе. Синтаксни анализатор, такозвани парсер, узима низ токена из лексичког анализатора и проверава да ли тај низ припада језику који је описан задатом граматиком. Корисник добија обавештење о томе да ли је исправна синтакса унетог кода и уколико није указује се на грешку. Поред праћења процеса лексичке и синтаксне анализе Scanner омогућава и креирање табеле симбола која садржи информације о препозатој лексеми, одговарајућем токену и атрибуту. Информације из табеле симбола потребне су за семантичку анализу и генерисање кода. На крају анализе студент добија и информацију о семантичкој усклађености изворног програма са дефиницијом језика. JFLAP и Seshat немају овакав тип скенера за праћење токенизације улазног кода.

6.3.3 Regular Expression (регуларни изрази)

У подмодулу Regular Expression могуће је извршити конверзију регуларног израза у NFA или DFA, пратити кораке Томсоновог алгоритама за конверзију регуларног израза у NFA, као и конструкцију синтаксног стабла за жељени регуларни израз. Поред могућности конструисања аутомата цртањем дијаграма стања или дефинисањем табеле прелаза, ComVis на овај начин, обезбеђује и трећи начин дефинисања коначног аутомата применом регуларних израза. Кориснички интерфејс и овог подмодула је конзистентан са осталим (циљ У4). На левој страни су доступне основне функције, док је

централни део резервисан за приказ резултата предузетих акција. Да би се извршила конверзија регуларног израза у детерминистички коначни аутомат, потребно је у предвиђено поље унети жељени регуларан израз и притиснути дугме DFA, на пример за регуларан израз *ab(a|b)*b* генерише се детерминистички коначан аутомат као на слици 69. Када се заврши конверзија регуларног израза у аутомат, на десној страни корисничког интерфејса појављује се и опција за покретање симулационог мода.



Слика 69. Regular Expression: Приказ конструисаног DFA из регуларног израза *ab(a|b)*b*

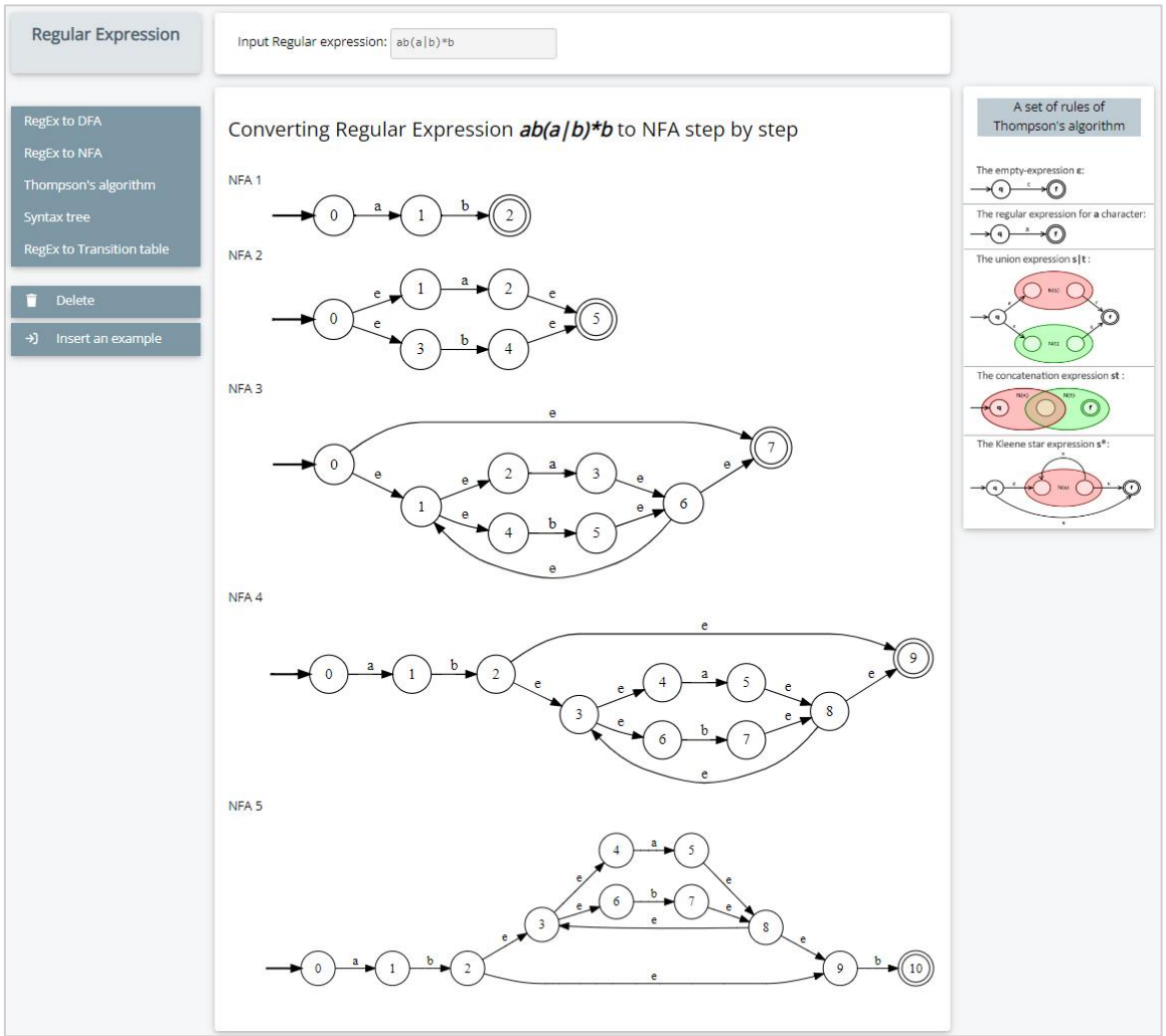
Уколико корисник покуша да унесе регуларни израз са погрешним оператором, систем генерише обавештење о томе и даје информацију о дозвољеним операторима са описом њихових функција (у складу са циљевима У5 и У6). Такође, за почетнике и мање напредне студенте и овде је на располагању пример који се у сваком тренутку може учитати (циљ П4).

6.3.3.1 Симулација Томсонове конструкције

Томсонова конструкција представља једноставан алгоритам за креирање NFA од еквивалентног регуларног израза. Алгоритам ради рекурзивно тако што дели израз на његове саставне подизразе, од којих ће NFA бити конструисан коришћењем скупа правила која су детаљније описана у теоријском делу дисертације. На слици 70 приказан је пример конструкције NFA на основу регуларног израза *ab(a|b)*b*. Са десне стране студентима су

приказана правила Томсонове конструкције, а централни део интерфејса приказује све кораке алгоритма. Алгоритам почиње са рашчлањивањем регуларног израза на подизразе од којих гради основне NFA који се даље применом индуктивних правила спајају у коначни NFA.

Први дијаграм стања на слици 70 одговара изразу *ab*, други дијаграм представља коначан аутомат за алтернацију *a|b*, на трећем је представљен регуларан израз *(a|b)**, четврти дијаграм одговара изразу *ab(a|b)** и последњи, пети дијаграм стања, представља полазни регуларни израз *ab(a|b)*b* и његову коначну конверзију у NFA.



Слика 70. Regular Expression: Симулација Томсонове конструкције за регуларни израз *ab(a|b)*b*

6.3.3.2 Конверзија регуларног израза у синтаксно стабло и DFA

Опција Syntax Tree омогућава кориснику да нацрта синтаксно стабло које одговара регуларном изразу, као и да израчуна функције: nullable, firstpos, lastpos и followpos. Регуларни израз се представља у облику стабла, тако да листови представљају терминалне симболе, а унутрашњи чворови су операције спајања (конкатенација) „·“, уније (алтернација) „∪“ и итерације „*“.

За чвор *n*, нека *L(n)* буде језик који генерише подстабло са кореном *n*:

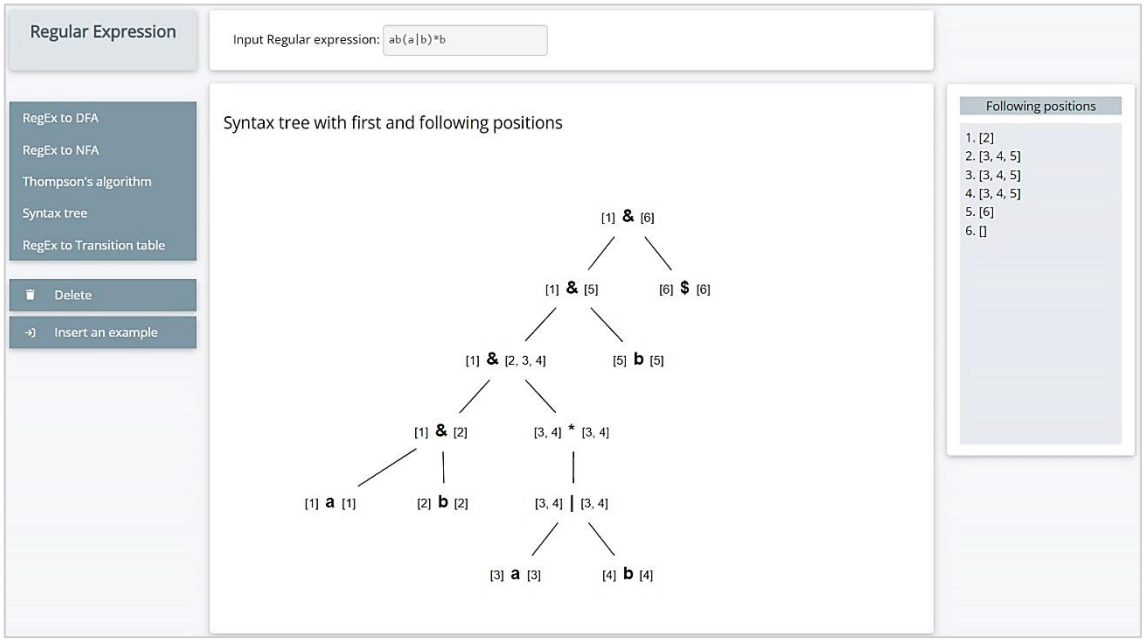
- nullable(n): L(n) садржи празан стринг ε;
- firstpos(n): функција даје скуп позиција које одговарају првим симболима у подстаблу са врхом *n* генерисаним подизразом L(n);
- lastpos(n): функција даје скуп позиција које одговарају последњим симболом у подстаблу са врхом *n* генерисаним подизразом L(n);
- followpos(i): функција даје скуп позиција које могу да прате позицију *i* у било ком генерисаном стрингу.

Скуп *firstpos* се у ComVis-у приказује лево, а скуп *lastpos* десно од сваког чвора. Скуп *followpos* се израчунава на основу алгоритма приказаног у коду 7.

```
for each node n in the tree do
  if n is a concatenation node with left child c1 and right child c2 then
    for each i in lastpos(c1) do
      followpos(i) := followpos(i) ∪ firstpos(c2)
    end do
  else if n is a star node
    for each i in lastpos(n) do
      followpos(i) := followpos(i) ∪ firstpos(n)
    end do
  end if
end do
```

Код 7. Алгоритам за израчунавање скупа *followpos*

Пример синтаксног стабла у систему ComVis за регуларни израз *ab(a|b)*b* приказан је на слици 71. Поред сваког чвора приказани су *firstpos* и *lastpos* скупови, а скуп *followpos* приказан је у посебном прозору на десној страни интерфејса.



Слика 71. Regular Expression: Синтаксно стабло за регуларни израз *ab(a|b)*b*

Алгоритам за директну конструкцију DFA из регуларног израза приказан је у коду 8.

```
s0 := firstpos(root) where root is the root of the syntax tree
Dstates := {s0} and is unmarked
while there is an unmarked state T in Dstates do
  mark T
  for each input symbol a ∈ Σ do
  let U be the set of positions that are in followpos(p)
    for some position p in T,
      such that the symbol at position p is 'a'
      if U is not empty and not in Dstates then
        add U as an unmarked state to Dstates
      end if
      Dtran[T,a] := U
  end do
end do
```

Код 8. Псеудо код алгоритма за директну конструкцију DFA из регуларног израза

У почетку се разматра скуп позиција *firstpos(root)*. Скуп позиција које представљају овај *firstpos* се сматрају једним стањем и ово је почетно стање DFA. Линија 1 и 2 алгоритма означавају ову процедуру где се променљива *Dstates* користи за памћење стања DFA. Линија 3 иницијализује *while* петљу

која генерише нова стања на основу прелаза из почетног стања. Почетно стање је маркирано у реду 4, што означава да се тренутно оно анализира. Линија 5 покреће *for* петљу да дефинише транзиције из почетног стања за све улазне симболе (знакове из регуларног израза). Ново стање ће чинити скуп позиција које се одређују као унија *followpos* позиција у почетном стању. Поступак се понавља све док се више не могу генерисати нова стања.

Поређењем ComVis-а са алатима JFLAP и Seshat може се приметити да оба алата имају опцију за конвертовање регуларног израза у NFA, док конвертовање регуларног израза директно у DFA подржава само Seshat.

6.4 ИМПЛЕМЕНТАЦИЈА МОДУЛА ЗА УЧЕЊЕ СИНТАКСНЕ АНАЛИЗЕ

Као што је у теоријском делу наглашено, задатак синтаксног анализатора је да прими низ симбола (токена) из лексичког анализатора и изврши проверу да ли тај низ припада језику који је описан задатом граматиком. Циљ синтаксног анализатора је да генерише синтаксно стабло за улазни низ симбола. Да би се генерисало синтаксно стабло, потребно је применити одговарајућа правила за пресликавање стартног симбола у анализирани низ. Како се за опис језика обично користе бесконтексне граматике, у сваком кораку извођења по један нетерминални симбол се пресликава у неку реч. У односу на то како обављају синтаксну анализу постоје два типа синтаксних анализатора:

- *top-down* анализатори који врше анализу одозго наниже и
- *bottom-up* анализатори који врше анализу одоздо навише.

Помоћу ComVis симулатора студенти могу да се упознају са алгоритмима који се примењују и код top-down и код bottom-up анализатора. Модул Synatax analiysis обухвата подмодуле који симулирају алгоритам рада нерекурзивног предиктивног парсера, LL(1) и LR(0) парсера.

6.4.1 Симулација нерекурзивног предиктивног парсера

Студенти у овом подмодулу добијају прилику да прате симулацију процеса нерекурзивног предиктивног парсирања заснованог на табели. Предиктивни парсер заснован на табели има један улазни бафер, стек, табелу парсирања и излазни низ (као што је представљено у теоријском делу дисертације). Улазни бафер садржи стринг који треба да се парсира уз додати симбол *$* на крају. Стек садржи секвенцу граматичких симбола са знаком *$* на дну. Табела парсирања је дводимензионални низ *M[A,a]*, где је *A* нетерминал, док *a* представља терминал или симбол *$*. Код 9 приказује алгоритам којим ComVis имплементира нерекурзивно предиктивно парсирање за улазни стринг *w* и табелу парсирања *M* за задату граматику *G*.

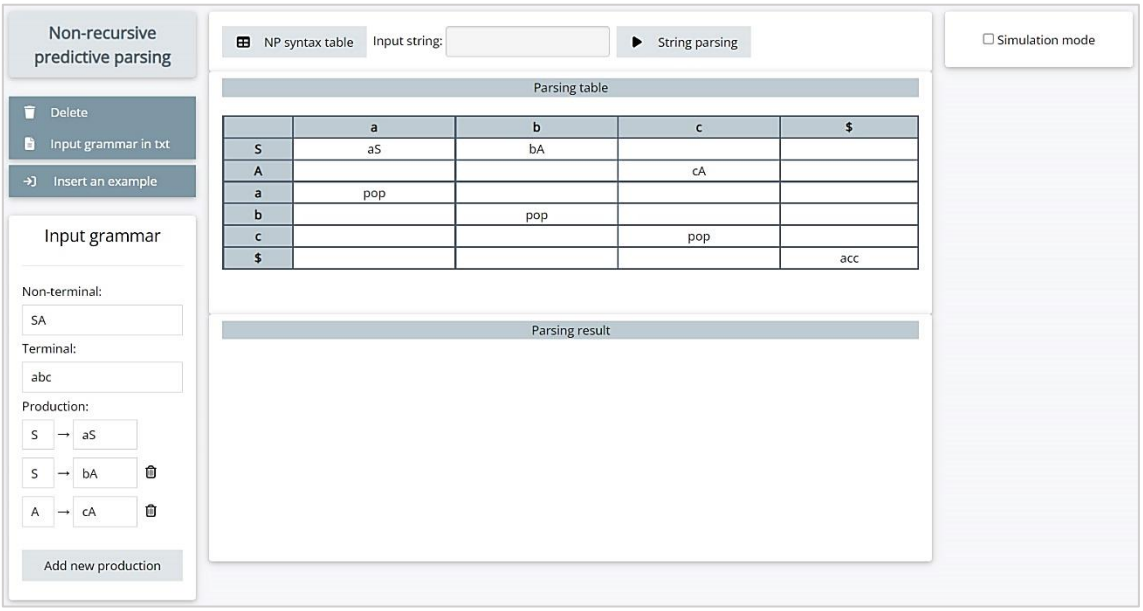
```
set ip to point to the first symbol of w;
set X to the top stack symbol;
while ( X ≠ $ ) {    /* stek nije prazan */
  if ( X is a ) pop the stack and advance ip;
  else if ( X is a terminal ) error();
  else if ( M[X, a] is an error entry ) error();
  else if ( M[X, a] = X → Y1 Y2... Yk ) {
    output the production X → Y1 Y2... Yk;
    pop the stack;
    push Yk, Yk-1,... , Y1 onto the stack, with Y1 on top;
  }
  set X to the top stack symbol;
}
```

Код 9. Алгоритам нерекурзивног предиктивног парсера (Aho et al, 2007)

Алгоритам узима *X*, симбол на врху стека, и *a*, тренутни улазни симбол стринга *w*. Ако је *X* нетерминал, парсер бира продукцију за *X* на основу уноса *M[X,a]* из табеле парсирања *M*. У супротном, проверава се подударање између терминала *X* и тренутног улазног симбола *a*. На почетку алгоритма парсер је у конфигурацији са првим симболом из стринга *w$* у улазном баферу и стартним симболом *S* граматике *G* на врху стека.

Кориснички интерфејс овог подмодула на левој страни садржи форму за дефинисање грамтике, у централном делу простор за приказ табеле

парсирања и резултата рада алгоритма за задати улазни стринг, а на десној страни је присутна уобичајена опција ComVis-а којом се покреће процес симулације (слика 72). Студенти имају могућност да сами дефинишу граматику на основу које ће пратити процес парсирања. У пољима Non-terminal и Terminal специфицирају се нетерминални и теминални симболи граматике. Продукције бесконтексне граматике студент уноси у посебно дизајнираној форми Production за једноставнији и прегледнији унос правила језика. У сваком тренутку може се додати нова продукција или избрисати нека од дефинисаних.



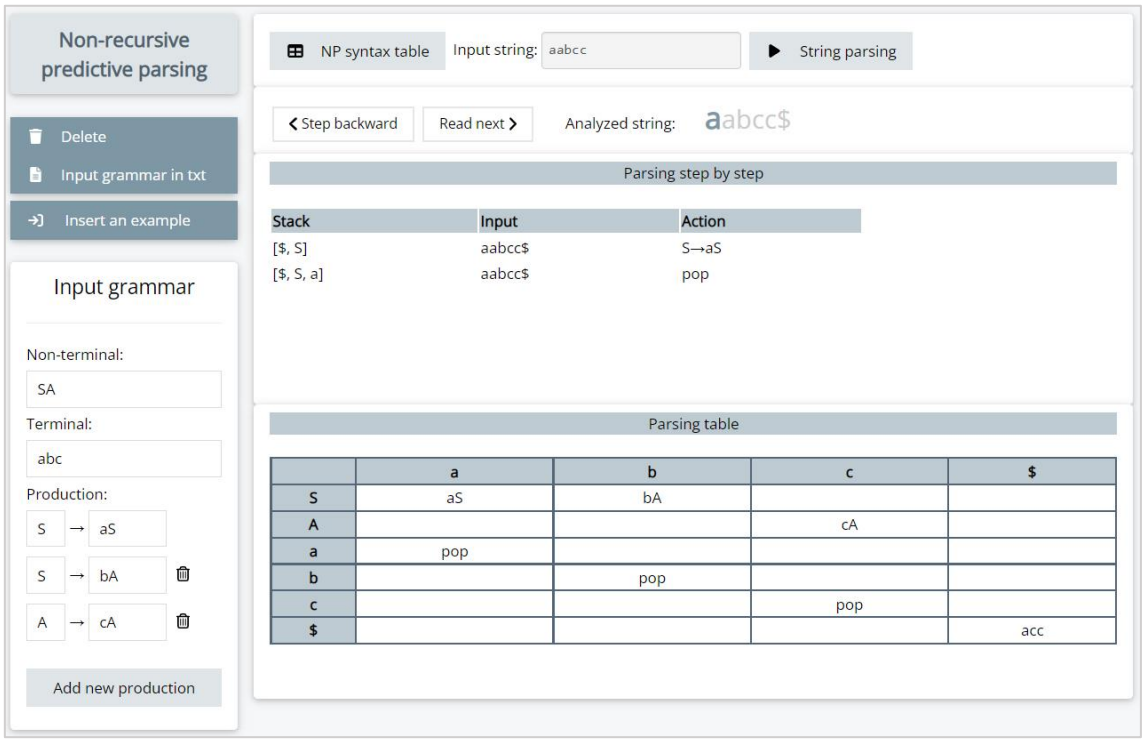
	a	b	c	$
S	aS	bA		
A			cA	
a	pop			
b		pop		
c			pop	
$				acc

Слика 72. NP Parser: Пример конструисања табеле парсирања за задату граматику

Због педагошког циља П4 на основу кога систем треба да пружи подршку различитим нивоима предзнања, ComVis студентима са мањим предзнањем обезбеђује могућност учитавања граматике из текстуалног фајла или могућност директног задавања готовог примера. Код учитавања граматике из текстуалног фајла, систем за спречавање грешака (циљ У6) проверава форму и упозорава корисника у случају погрешно написаних продукција. На слици 72 приказан је пример конструкције табеле парсирања која се након дефинисања граматике добија избором опције NP syntax table. Када је табела парсирања креирана, студент може задати жељени улазни стринг и покренути процес

парсирања избором опције String parsing. Пошто се алгоритам нерекурзивног предиктивног парсирања одвија у петљи, за сваки улазни знак у ComVis-у за визуелизацију сваког корака итерације искоришћена је та чињеница. Студенти могу пратити симулацију процеса парсирања у корацима које сами контролишу (циљеви ПЗ и УЗ). Због подршке различитим нивоима предзнања, ComVis може да прикаже и комплетан поступак парсирања без симулације.

Један корак симулације парсирања за улазни стринг *aabcc* приказан је на слици 73. Као што се може видети, у сваком кораку симулације студенти могу да прате тренутни симбол улазног стринга који се анализира, стање стека као и предузету акцију из табеле парсирања која је све време присутна због једноставнијег праћења читавог процеса. Овде је битно нагласити да референтни системи за поређење са ComVis-ом, JFLAP и Seshat не пружају могућност учења нерекурзивног предиктивног парсирања.



Слика 73. NP Parser: Симулација процеса нерекурзивног предиктивног парсирања за задату граматику и улазни стринг

6.4.2 Симулација LL(1) парсера

Ефикасан алгоритам за top-down анализу је LL(1) парсер код кога се предикција врши на основу једног знака у улазном стрингу. Најчешћа техника имплементације LL(1) парсера заснива се на синтаксној табели. Код 10 представља псеудо код алгоритама који ComVis користи за реализацију LL(1) парсера. Овакав парсер захтева граматику, табелу парсирања и стек за праћење тренутног скупа нетерминала. LL(1) табела парсирања се користи да одреди које правило треба применити за било коју комбинацију нетерминалног симбола на стеку и следећег токена на улазном току *ts*.

```
procedure LLparser(ts)
   call push(S)
   accepted ← false
     while not accepted do
       if TOS() ∈ Σ
       then
         call match(ts, TOS())
         if TOS() = $
         then accepted ← true
         call pop()
       else
         p ← LLtable[TOS(), ts.peek()]
         if p = 0
         then
           call error(Syntax error – no production applicable)
         else call apply(p)
end
procedure apply( p : A → X1 ...Xm )
   call pop()
   for i = m downto 1 do
     call push(Xi)
end
```

Код 10. Псеудо код алгоритма за имплементацију LL(1) парсера (Fischer, Cytron and LeBlanc Jr, 2010)

При свакој итерацији петље *while*, LL(1) парсер изводи једну од следећих акција:

- Методом *TOS( )* се проверава садржај на врху стека и уколико је у питању терминални симбол, онда се позива метода *MATCH*. Овом методом утврђује се да ли следећи токен улазног тока одговара

симболу на врху стека. Ако постоји подударање, прелази се на нови токен улазног тока, и методом *pop()* се скида симбол са врха стека;

- Ако је врх стека неки нетерминални симбол *A*, онда се тражи одговарајућа продукција $A \rightarrow X_1 \ldots Xm$ из табеле парсирања. Ако се пронађе одговарајућа смена, позива се примена тог правила да би се нетерминал *A* избацио са врха стека. Симболи из смене $X_1 \ldots Xm$ се затим гурају на стек почевши од X_1 тако да врх стека буде *Xm*.

Парсирање је завршено када се дође до симбола за крај уноса $. За креирање табеле парсирања користе се FIRST и FOLLOW скупови и она се израчунава на следећи начин:

- За продукцију $A \rightarrow B\alpha$ и сваки терминал *a* који припада скупу *FIRST(B)* додаје се $A \rightarrow B\alpha$ у *M[A, a]*;
- Ако *ε* припада скупу *FIRST(B)* тада за сваки елемент *b* из скупа *FOLLOW(A)* у поље табеле *[A, b]* уписује се $A \rightarrow B\alpha$.

Слика 74. LL(1) Parser: Симулација процеса LL(1) парсирања за задату граматику и улазни стринг

Редови и колоне табеле парсирања означени су нетерминалима и терминалима граматике респективно. Табела у процесу парсирања се користи тако што нетерминални симбол на врху стека одређује ред, а следећи улазни токен одређује колону која ће бити изабрана.

Кориснички интерфејс LL(1) парсера је исти као и код претходног подмодула синтаксне анализе (у складу са циљем У4). Једина ралика је у томе што је додата опција за израчунавање FIRST и FOLLOW скупова. На слици 74 приказан је пример једног корака симулације парсирања за улазни стринг *(a∗a)* и задату LL(1) граматику.

JFLAP омогућава симулацију LL(1) парсирања, док Seshat не покрива ову тему.

6.4.3 Симулација LR(0) парсера

LR анализаторима могу се препознати све програмске конструкције које се описују бесконтексним граматикама. Реализација LR(0) анализатора се заснива на дефницији LR(0) чланова и примени функције *closure* за затварање LR(0) чланова и функције *goto* за иницијализацију новог скупа LR(0) чланова. LR(0) чланови се генеришу на основу правила граматике и свако правило граматике даје неколико LR(0) чланова (детаљније објашњење је дато у теоријском делу дисертације). На основу добијене колекције ових чланова генерише се LR(0) аутомат, а као крајњи резултат добија се LR синтаксна табела на којој се заснива процес парсирања.

LR синтаксна табела се састоји од два дела, први део садржи ACTION функције, а други GOTO функције. LR(0) табела има онолико врста колико је могућих стања синтаксног анализатора. У делу за акције постоји онолико колона колико је терминалних симбола граматике, а у делу за прелазе број колона једнак је броју нетерминалних симбола граматике. Псеудо код алгоритма за конструисање LR(0) синтаксне табеле на основу задате граматике *G* приказан је у коду 11.

```
function generate-tbl(A,G) =
  <Q,δ,start> := generate-dfa(A,G)
  for each q → X q' ∈ δ
    if X ∈ Σ(G) then action(q,X):= shift q'
    if X ∈ N(G) then goto(q,X):= q'
  for each q ∈ Q
    if [A → α•eof] ∈ q then
      accept := accept ∪ {q}
    for each [A → α•] ∈ q
      for each a ∈ Σ(G)
        action(q,a) := reduce A → α
  return <Q,Σ(G),N(G),start,action,goto,accept>

function move(q,X) =
  return {[A → αX•β] | [A → α•Xβ] ∈ q}

function generate-dfa(A,G) =
  start := closure({[S' → •A eof]},G)
  Q := {start}, δ := ∅
  repeat until Q and δ do not change
    for each q ∈ Q
      for each X ∈ {Y | [B → α•Yβ] ∈ q}
        q' := closure(move(q,X),G)
        Q := Q ∪ {q'}
        δ := δ ∪ {q →X q'}
  return <Q,δ,start>

function closure(q,G) =
  repeat until q does not change
    for each [A → α•Bβ] ∈ q
      q := q ∪ {[B → •γ] | B → γ ∈ P(G)}
  return q
```

Код 11. Псеудо код за конструисање LR(0) синтаксне табеле на основу LR(0) аутомата (Bravenboer and Visser, 2008)

LR(0) табела се попуњава ACTION и GOTO функцијама праћењем стања и прелаза LR(0) аутомата. За свако стање *q* из које за *x* постоји прелаз према *q'*, ако *x* припада скупу терминала, онда се у пољу (*q, x*) синтаксне табеле уписује акција *shift q'*. Уколико је *x* нетерминални симбол, онда се у GOTO делу табеле у пољу (*q, X*) уписује прелаз *q'*. За стање прихватања *q* до које води симбол за крај улазног низа (eof или $) уноси се *accept*. За свако стање *q* које садржи продукцију облика *A→a•*, где је *a* терминал, у поље (*q, a*) акционог дела синтаксне табеле се уписује *reduce A→a*.

Алгоритам коришћен за имплементацију LR(0) парсера у систему ComVis приказан је у коду 12. Улазни параметри овог алгоритма су стартно стање S_S,

стање прихватања S_A, LR синтаксна табела T конструисана за задату граматику G и улазни стринг $a_1 \ldots a_n$.

```
push $ and then Ss on to the stack
  for i=0 to n do
    let l be the state on the top of the stack
    if sk ∈ T(l,ai+1) then
      push ai+1 and then k on to the stack
    else if rk ∈ T(l,ai+1) then
      find rule number k such that A ::= β
      pop 2x |β| symbols off the stack
      let t be the state on the top of the stack
      let u ∈ T(t,A)
      push A and then u on to the stack
    else if acc ∈ T(l,ai+1) then
      return success
    else return error
```

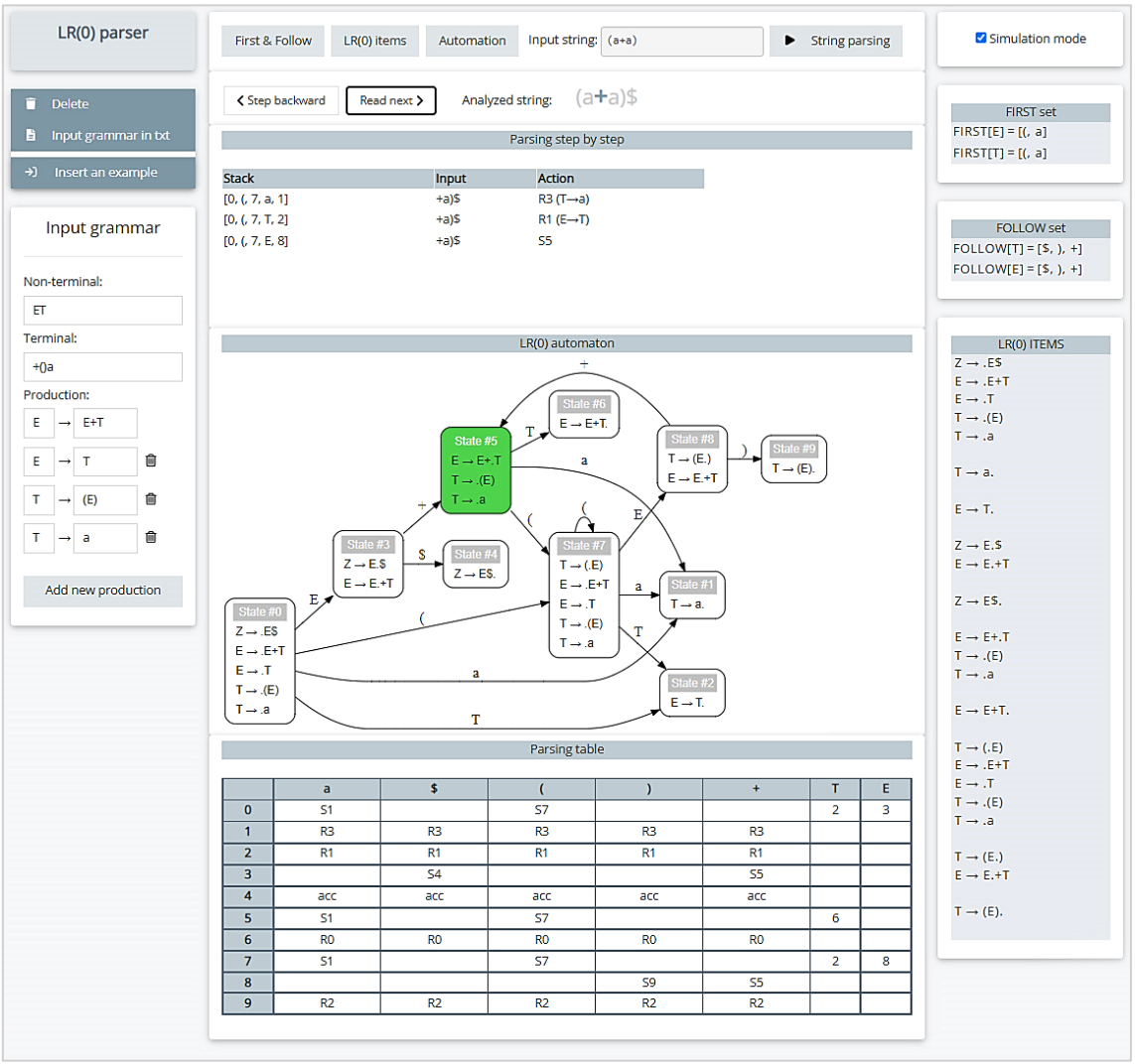
Код 12. Алгоритам за имплементацију LR(0) парсера (Economopoulos, 2006)

На основу тренутног улазног симбола a_i и стања на врху стека l, алгоритам одређује следећи потез парсера тако што покреће акцију дефинисану у пољу $ACTION[l,a_i]$ табеле парсирања T. Иницијално, парсер има S_s на врху стека и први симбол на улазном баферу. Ако је $T[l, a_i] = shift\ k$, са улазног бафера се узима текући знак и заједно са стањем k се поставља на стек тако да је k на врху. Након тога се прелази на анализу наредног знака са улаза a_{i+1}. Ако је $T[l, a_i] = reduce\ k$, врши се редукција употребом продукције k. За продукцију $A \rightarrow \beta$ са стека треба уклонити укупно $2x|\beta|$ елемената. На основу тренутног стања на врху стека t, из GOTO дела синтаксне табеле треба узети ново стање u и онда на стек поставити нетерминал A, па затим и u. Уколико је $T[l, a_i] = accept$, стринг је препознат и анализа се прекида. У свим осталим случајевима улазни стринг се не може препознати и анализа се прекида.

Као и код претходна два подмодула синтаксне анализе и овде је студентима омогућено праћење процеса парсирања у корацима. Најпре се у форми за унос продукција дефинишу правила граматике, а након тога студенти могу да покрену опцију за израчунавање FIRST и FOLLOW скупова и опцију за за генерисање LR(0) чланова. Следи приказ дијаграма стања LR(0)

аутомата на основу кога се формира и одговарајућа синтаксна табела. Након тога може се унети одговарајући улазни стринг за симулацију поступка LR(0) парсирања, као што је приказано на слици 75.

У току симулације рада парсера студенти у сваком тренутку могу пратити и тренутно стање LR(0) аутомата. Овакав начин визуелизације који подразумева више различитих графичких елемената за приказ једног теоријског концепта, помаже дубљем разумевању и повећању мотивације студената у складу са дефинисаним педагошким циљевима П2 и П3. Овакву могућност не пружају JFLAP и Seshat.



Слика 75. LR(0) Parser: Симулација процеса LP(0) парсирања за задату граматику и улазни стринг

6.5 ИМПЛЕМЕНТАЦИЈА МОДУЛА ЗА УЧЕЊЕ СЕМАНТИЧКЕ АНАЛИЗЕ

Након проучавања форме програмског кода, односно његове синтаксе, ComVis студентима пружа могућност да науче и како програмски преводиоци анализирају семантику, то јест начин на који се тумачи значење програмског кода. Алати JFLAP и Seshat не пружају могућност учења семантичке анализе. У томе је највећа предност ComVis-а јер заокружује комплетан процес који се обавља предњим делом компајлера.

Опис наредби језика преко бесконтексних граматика обично није довољан за превођење језика. Постоји много проблема који се не могу представити формалним описом језика и због тога се правилима којима се дефинише језик придружује скуп додатних информација тако што се симболима граматике додају одређени атрибути. Вредност атрибута одређена је семантичком рутином која је придружена правилу које се користи у одговарајућем чвору синтаксног стабла. Атрибути могу да буду синтетизовани и наслеђени. Вредности синтетизованих атрибута се израчунавају као функције атрибута потомака одговарајућег чвора, док се вредности наслеђених атрибута израчунавају као функције атрибута родитељских чворова и атрибута осталих чворова на истом нивоу (детаљније информације се могу наћи у теоријском делу дисертације). За приказ синтаксног стабла заједно са вредностима атрибута придружених чворовима користи се анотирано синтаксно стабло. Семантички анализатор користи синтаксно стабло и информације из табеле симбола да провери семантичку усклађеност задатих наредеби са дефиницијом језика. У овом поглављу, у секцији где је описан лексички анализатор, приказан је начин на који се препознаје лексема и како се са одговарајућим токеном и атрибутом додаје у табелу симбола. Поступак је једноставан када су у питању константе и кључне речи, док је код идентификатора ситуација другачија. Треба обратити пажњу код евалуације израза који садрже идентификаторе којима је већ додељена

вредност и тип. Алгоритам за препознавање типа чвора идентификатора дефинисан је кодом 13.

```
procedure processNode(node)
switch (kind(node))
  case Dcl
    call symtab.enterSymbol(node.name, node.type)
  case Expr
    call symtab.checkType()
    call symtab.enterSymbol(node.name, node.type)
end switch
checkType()
  foreach c ∈ node.getChildren()
  if c Ref
    c ← symtab.retrieveSymbol(node.name)
      if c = null
        then call error("Undeclared symbol : ", c)
      end if
  end if
  ResultExpr()
  type = ResultExpr.type
return type
end
```

Код 13. Алгоритам за препознавање типа чвора идентификатора

Алгоритам најпре испитује да ли се идентификатор, то јест променљива, декларише директно или путем израза. Уколико је у питању директна декларација неком константом, функцијом *enterSymbol(name, type)* се у табели симбола за дати идентификатор уноси име и тип који одговара типу константе. У случају да се променљивој додељује вредност израза, онда се позива функција *checkType()* за утврђивање типа добијеног резултата. Овом функцијом се испитује подстабло датог чвора тако што се проверавају типови и вредности деце чвора. Уколико у изразу постоји идентификатор који је већ декларисан, односно који у овом изразу представља референцу, онда се позива функција *retrieveSymbol(node.name)*. Овом функцијом тражи се вредност идентификатора у табели симбола на основу имена. Уколико се не пронађе, генерише се информација да дати симбол *c* није декларисан. Уколико грешка не постоји, позива се функција *ResultExpr()* којом се утврђује вредност израза. На крају се посматраном чвору додељује тип који одговара типу резултата.

На слици 76 приказан је скуп правила граматике којом се описује једноставан језик за задавање основних математичких операција. На основу ове дефиниције језика, студент у поље за унос изворног кода може да унесе математичке изразе који ће бити прослеђени семантичком анализатору. Као и код осталих модула, и код семантичког анализатора је кориснички интерфејс дизајниран на исти начин. На левој страни се налази палета алата подељена на три блока. У првом блоку се налазе опције за учитавање сачуваног програмског кода за анализу, чување тренутно задатих наредби и брисање садржаја поља за унос. Главни алати за анализу задатог програмског кода доступни су у другом блоку. У последњем блоку налази се само једна опција која омогућава унос готовог примера како би се студентима почетницима олакшала употреба овог модула.

```
expr_list := expr_list expr_part | expr_part
expr_part := IDENTIFIER ASSIG expr SEMI
expr := term term_add
term_add := add_op term term_add | e
term := factor factor_add
factor_add := mult_op factor factor_add | e
factor := ( expr ) | IDENTIFIER| NUMBER
add_op := + | -
mult_op := * | /
```

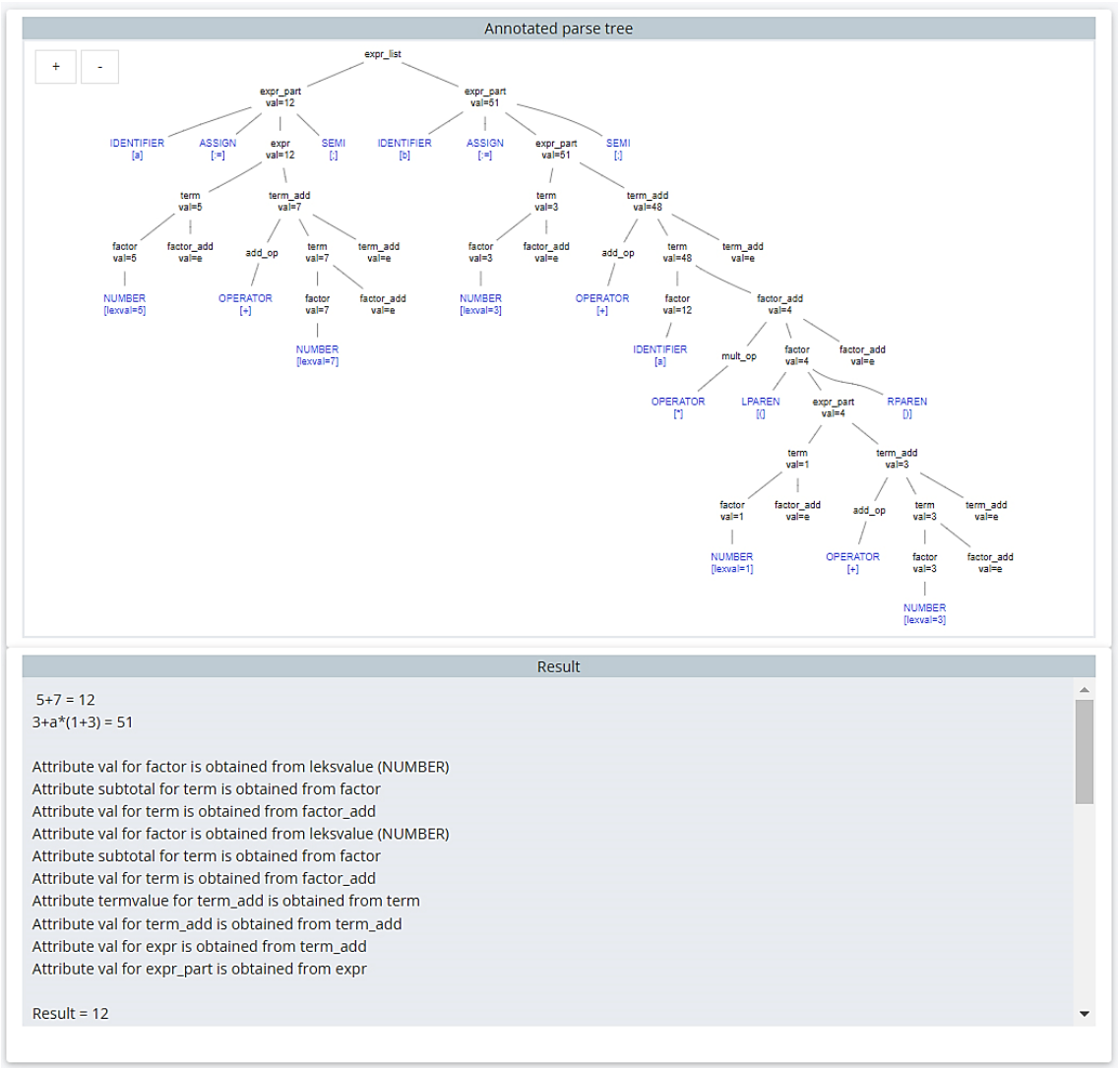
Слика 76. Пример правила граматике за опис језика који се користи за симулацију семантичке анализе

Приказана граматика је увек присутна и налази се поред поља за унос у горњем делу корисничког интерфејса како би студентима било лакше да у складу са датом граматиком формирају одговарајуће изразе за анализу. Резултати анализе се приказују у централном делу интерфејса.

Модул Semantic analysis обезбеђује већи број различитих анализа, па тако студент пре симулације семантичке анализе може да покрене лексичку анализу како би се уверио да су задати изрази лексички исправни, након чега може покренути процес парсирања и проверити синтаксу унетог кода. Такође, на располагању су му и прикази табеле симбола и синтаксног стабла. За дубље разумевање процеса семантичке анализе доступне су следеће опције:

приказ анотираног синтаксног стабла, евалуација задатих израза и симулација семантичке анализе. На слици 77 приказана је евалуација следећих израза:

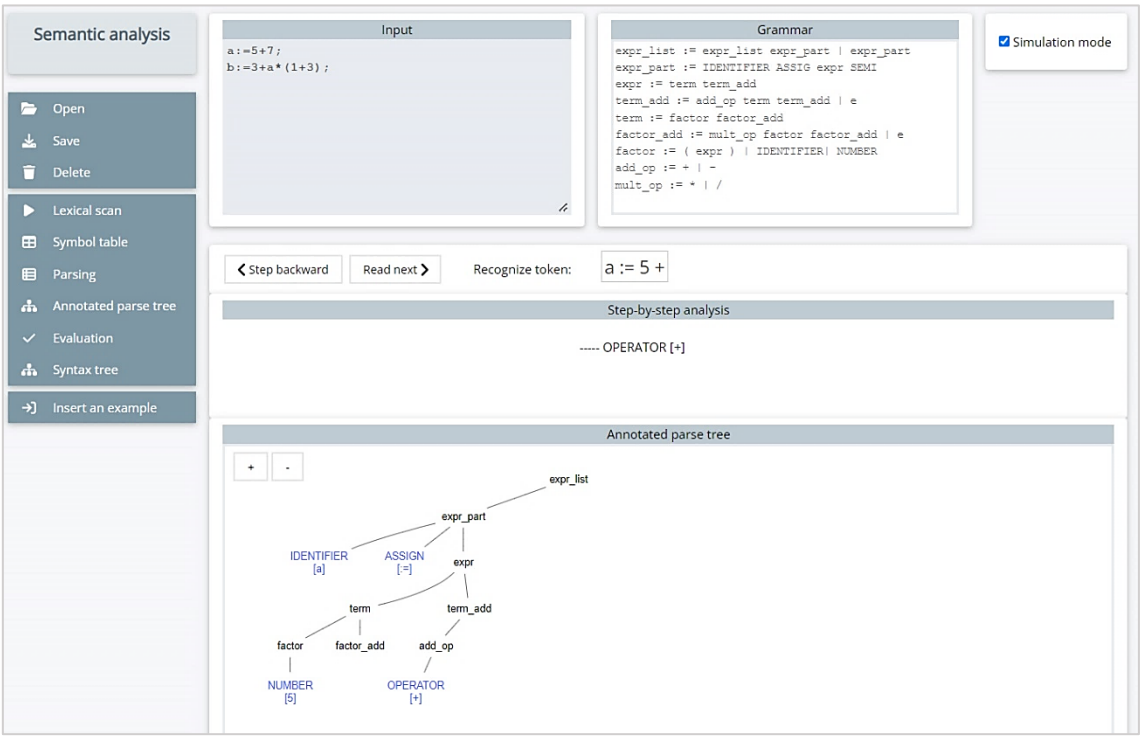
```
a:=5+7;
b:=3+a*(1+3);
```



Слика 77. Приказ евалуације задатих израза

Студент у овом случају добија график анотираног синтаксног стабла и текстуални опис резултата евалуације. Вредности израза израчунавају се преко атрибута *val*. Атрибут *lexval* преузима из табле симбола вредност цифре (константе) представљене преко токена *NUMBER*.

Семантичку анализу студент најбоље може да разуме праћењем анимације формирања семантичког стабла и проценом атрибута на чворовима стабла. Ова анимација је могућа у симулационом моду где студент у сваком кораку прати креирање семантичког стабла, добија информације о предузетим акцијама у складу са граматиком и информацију о тренутно препознатом токену (слика 78). Оквир за приказ стабла има опцију зумирања и померања како би студент због боље прегледности могао да увећа жељени део стабла, уколико је због сложеног израза стабло достигло велике размере.



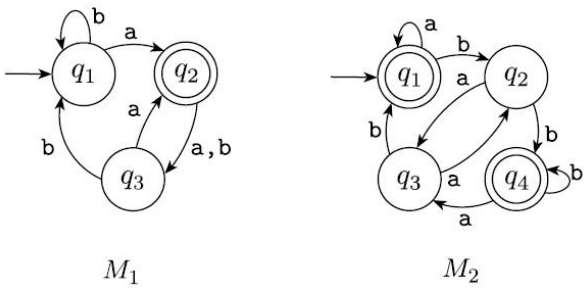
Слика 78. Пример једног корака симулације поступка семантичке анализе

6.6 ПРИМЕНА ОБРАЗОВНОГ СИСТЕМА COMVIS

Настава на предмету Програмски преводиоци заснива се на традиционалној парадигми теорија-примери-вежбе. За сваку тему се прво излажу теоријски концепти на часовима теоријске наставе. Лабораторијске вежбе предвиђају детаљније објашњавање тема обрађених на предавањима уз употребу симулатора за визуелно представљање изложених концепата.

Усклађеност ComVis-а са литературом која се користи на теоријској настави (на основу постављеног циља употребљивости У7), омогућава његову једноставнију интеграцију у оквиру лабораторијских вежби. Свака вежба се састоји од три компоненте: демонстрација примера, лабораторијски задатак и процена знања. Да би се припремили за вежбу, студенти треба да прочитају одговарајући материјал са предавања и уџбеника, као и пратећи лабораторијски материјал. На почетку лабораторијске вежбе предавач демонстрира примере у оквиру образовног алата. Предавач треба да искористи потенцијал система како би ангажовао студенте, јер би у супротном вежбе представљале само другачији приступ теоријске презентације. У складу са принципима активног учења, интерактивни приступ демонстрацији подразумева и ангажовање студента како не би били само пасивни посматрачи. Најлакши начин да студенти буду укључени у току демонстрације је често постављање питања „шта ако“ и навођење да предвиде резултате. На пример, у току демонстрације рада коначног аутомата у ComVis-у, може се додати нова транзиција или стање и онда од студената затражити одговор на питање шта изазивају извршене промене. Покретањем симулационог мода студентима се одмах може показати да ли су били у праву. Након завршене демонстрације, студенти добијају одговарајуће лабораторијске задатаке које самостално обављају. Пример прве лабораторијске вежбе приказан је у наставку.

1) На следећој слици су приказани дијаграми стања два коначна аутомата M_1 и M_2:



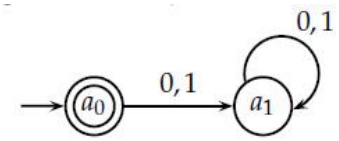
(а) Која стања су стартна код оба аутомата?

(б) Која стања чине скупове стања прихватања код оба аутомата?

(ц) Кроз која стања пролазе аутомати за улазни стринг *aabb*?

(д) Да ли аутомати прихватају стринг *aabb*?

2) Конструишите аутомат са слике $M=(A, \Sigma, \delta, a_0, F)$ и испитајте које речи препознаје над улазном азбуком $\Sigma= \{0,1\}$.



3) За аутомат $M=(A, \Sigma, \delta, a_0, F)$, где је $A = \{a_0, a_1, a_2, a_3\}$, $F = \{0,1\}$, $\Sigma = \{a_1, a_2\}$ и функција прелаза δ задата следећом табелом транзиција:

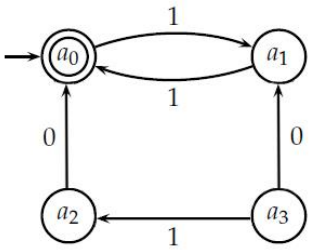
δ	a_0	a_1	a_2	a_3
0	a_1	a_2	a_2	a_3
1	a_3	a_3	a_2	a_3

(а) Конструишите дијаграм стања аутомата *M*.

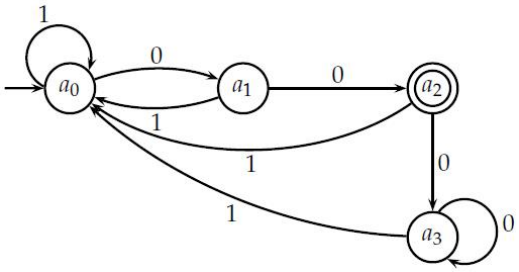
(б) Да ли овај аутомат препознаје стрингове 000 и 010?

4) Конструишите аутомат $M=(Q, \Sigma, \delta, q_0, F)$ који препознаје све могуће речи над азбуком $\Sigma = \{a, b\}$.

5) Додајте транзиције аутомату са слике тако да препознаје речи азбуке које садрже паран број нула и паран број јединица.



6) Да ли је аутомат са слике типа DFA или NFA? Објасните.



7) Нека је *M* детерминистички аутомат дефинисан на следећи начин:

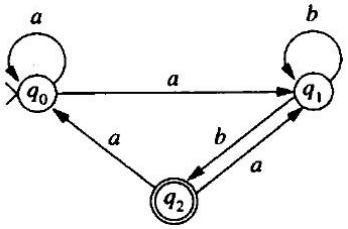
$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$F = \{q_0\}$

δ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

(а) Конструишите дијаграм стања.

(б) Пратите промену стања за улазни стринг *babaab*.

8) Нека је недетерминистички аутомат *M* задат дијаграмом стања:



(а) Конструишите табелу транзиција аутомата *M.*

(б) Проверите да ли *aaabb* припада језику *L(M)*?

9) Конструишите аутомат који може да препозна стринг *abababaab*.

10) Конструишите недетерминистички аутомат који над азбуком {*0, 1*} прихвата језик:

(а) {*w*| *w* садржи подстринг 0101}, (на пример, w = …0101…).

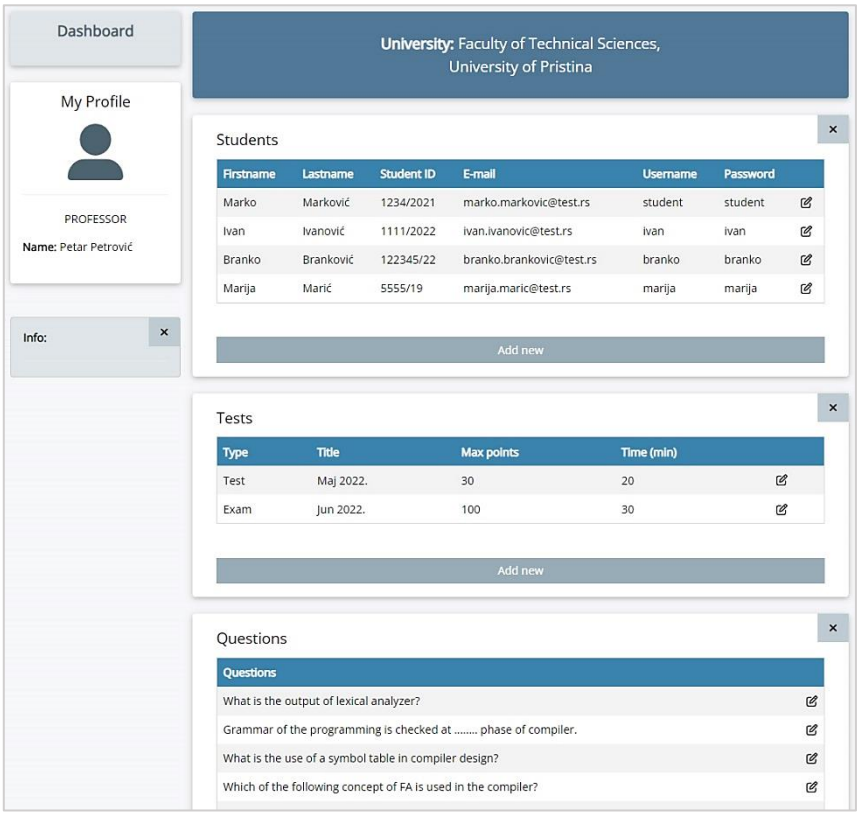
(б) {*w*| *w* садржи паран број 0 или садржи тачно две 1}.

Симулациони систем ComVis покрива већи део лабораторијских вежби предмета Програмски преводиоци. У табели 12 приказане су тематске јединице које се могу обрађивати на лабораторијским вежбама помоћу овог образовног окружења. Захваљујући доступности ComVis-а, студенти имају прилику да вежбају код куће и учење могу да прилагоде својим индивидуалним потребама јер систем има отворене све подмодуле без условљавања да се вежбање врши одређеним редоследом (што је у складу са педагошким циљем П5).

Табела 12. Тематске јединице које се могу обрађивати на лабораторијским вежбама помоћу софтверског система ComVis

Р.број	**Тематске јединице**
1.	Коначни аутомати - Дефинисање аутомата (дијаграм стања, табела прелаза) - DFA - NFA - Конверзија NFA у DFA
2.	Минимизација аутомата - Метода партиционисања (Hopcroft-ов алгоритам)
3.	Регуларни изрази - Конверзија регуларног израза у коначан аутомат (Томсонов алгоритам) - Конструкција DFA из регуларног израза помоћу синтаксног стабла регуларног израза
4.	Конверзија коначног аутомата у регуларни израз - Метода елиминације стања
5.	Конструкција лексичког скенера - Токени - Шаблони - Табела симбола (основе)
6.	Top-Down алгоритми, нерекурзивно предиктивно парсирање - NP синтаксна табела
7.	Top-Down алгоритми, LL(1) парсирање - Бесконтексне LL(k) граматике - First и Follow скупови - LL(1) синтаксна табела
8.	Bottom-Up алгоритми, LR(0) парсирање - Бесконтексне LR(k) граматике - LR(0) чланови (LR(0) аутомат) - LR(0) синтаксна табела
9.	Атрибутивне граматике - Синтетизовани атрибути - Наслеђени атрибути - Табела симбола (напредни ниво)
10.	Семантичка анализа - Провера типа - Анотирано синтаксно стабло

У лексичком делу алата студенти могу да се упознају са основним појмовима коначних аутомата, врстама и различитим могућностима њихове конструкције. Такође, у овом модулу студенти могу да науче све о регуларним изразима и њиховој практичној примени визуелизацијом и симулацијом различитих конверзија (FA у регуларни израз и обрнуто). У синтаксном делу алата уведене су различите врсте парсера. Студенти сами дефинишу језике уносом скупа правила чиме стичу искуство у раду са граматикама. Праћење симулација парсирања за различите задате улазе студентима помаже да дубље разумеју процес синтаксне анализе. У семантичком делу симулатора студенти применом различитих симулационих алата уче о атрибутивним граматикама и семантичким правилима.

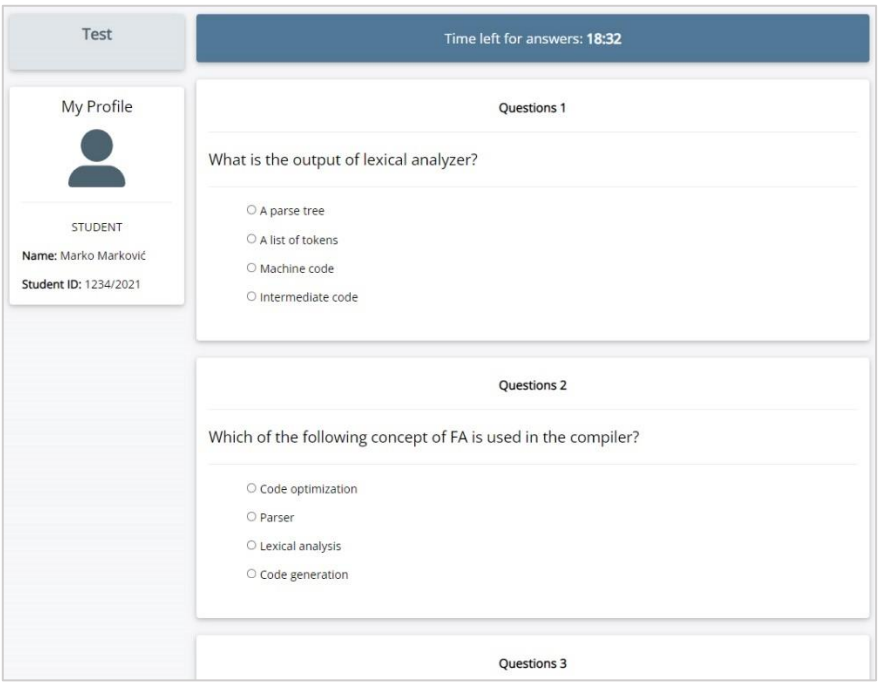


Слика 79. Администраторски панел ComVis-а за управљање тестовима

После сваке лабораторијске вежбе следи провера стеченог знања. На основу запажања током демонстрације предавача и стеченог искуства самосталним вежбањем постављених задатака, студенти одговарају на питања о обрађеној теми. Симулациони систем ComVis за разлику од свих

анализираних алата садржи и део за процену знања студената (циљ П6). Имплементацијом овог модула испуњени су сви постављени педагошки циљеви. До модула *Test* се долази тако што се студент пријављује на систем својим приступним подацима (циљ Т3). Наставници такође имају могућност да се пријаве на систем чиме добијају администраторске опције којима уређују тестове за проверу знања и прате активности студената (слика 79).

Након завршене лабораторијске вежбе, студенти се пријављују на систем и у делу *Test* на врху странице имају опцију за покретање актуелног теста. Сваки тест садржи квиз питања која су везана за обрађену тематску јединицу (слика 80). Тестови су временски ограничени. Број питања, њихов садржај, број поена, као и време трајања дефинише наставник у свом контролном панелу. Након завршеног теста, студент добија повратну информацију о броју тачних одговора, као и информације о тачним одговорима на питањима које је погрешио. Наставници добијају евиденцију о успеху свих студената.



Слика 80. Пример теста у оквиру система ComVis

Применом ComVis система предавачи су приметили да се студентима повећала мотивација и пажња током вежби, што се манифестовало њиховим активнијим учешћем. Током уводних инструкција о тематској јединици и

начину примене ComVis система за дату тему, запажено је да су студенти постављали различита питања о могућностима алата. Такође, током самих вежби студенти су имали жељу да сами предлажу и тестирају неке примере. Напреднији студенти су одмах прелазили на рад са комплекснијим примерима, док су остали морали да крену од једноставнијих.

7. ЕВАЛУАЦИЈА СИМУЛАЦИОНОГ СИСТЕМА COMVIS

У овом поглављу дисертације представљени су резултати квантитативне евалуације ефикасности система спроведене контролисаним експериментом и резултати квалитативне евалуације употребљивости која је реализована анкетом студената и хеуристичким испитивањима. Хеуристичка испитивања су извршена применом постојеће методе која је намењена образовном универзитетском софтверу, али и развијеном методом која је прилагођена домену и којом су процењени одабрани софтверски системи у поглављу 4.

7.1 КВАНТИТАТИВНА ПРОЦЕНА ЕФИКАСНОСТИ

Ефикасност односно успешност образовног софтверског алата је испитивана контролисаним експериментом. Ова метода подразумева праћење и упоређивање исхода, односно резултата две групе студената. Једну групу су чинили студенти који су у процесу учења тема програмских преводилаца користили веб базирани систем ComVis као помоћно наставно средство и то је била експериментална група. Друга група је била контролна, тако да се учење студената ове групе заснивало на традиционалним методама без помоћног софтверског алата. Сличне методе евалуације образовних алата коришћене су у (Amershi et al., 2008; Grivokostopoulou and Hatzilygeroudis, 2013; Pfahl et al., 2004).

Експеримент је спроведен у академској 2021-2022 години. Формиране су две групе од по 12 студената, при чему се водило рачуна о равномерној заступљености просечних и напредних студентата. Да би се пружио доказ да између ових група не постоји значајна разлика, коришћен је такозвани пре-тест који је садржао питања о основним појмовима из области компајлера. Пре-тест је садржао 25 питања, свако питање је носило 4 поена, па је тако максималан број поена био 100. Остварени резултати на крају овог теста приказани су у табели 13.

Табела 13. Резултати студената остварени на пре-тесту (макисималн број поена 100)

Група	**Број студената**	**Средња вредност**	**Стандардна девијација**
Експериментална	12	59.667	9.717
Контролна	12	61.333	7.499

За поуздану анализу добијених резултата коришћен је Студентов т-тест. За проверу варијанси је употребљен Ф-тест. Пошто је Ф-тест показао да су варијансе два скупа резултата еквивалентне, коришћен је т-тест са претпоставком једнаких варијанси (Two-Sample Assuming Equal Variances) са нултом хипотезом да између две групе студената не постоји значајна разлика. Резултати ових анализа приказани су у табели 14. Пошто је вредност *p-value* 0,643>0,05 може се закључити да је нулта хипотеза потврђена, то јест да се две групе студената нису значајно разликовале пре експеримента.

Табела 14. Дескриптивна статистика остварених резултата на пре-тесту

***F*-Test Two-Sample for Variances**			***t*-Test: Two-Sample Assuming Equal Variances**				
F	**P(F<=f) one-tail**	**F Critical one-tail**	**t Stat**	**P(T<=t) one-tail**	**t Critical one-tail**	**P(T<=t) two-tail**	**t Critical two-tail**
1,679	0,202	2,812	-0,4704	0,321	1,717	0,643	2,074

Обе групе су пратиле теоријска предавања формалних језика и граматика, теорије аутомата, лексичке, синтаксне и семантичке анализе. Предавачи и асистенти на лабораторијским вежбама су били исти и код једне и код друге групе студената. Након завршених предавања из наведених тематских јединица, обе групе добиле су колоквијум са потпуно истим задацима. Колоквијум из предмета Програмски преводиоци носи укупно 25 поена и садржи 25 питања, тако да свако питање вреди по један поен. Колоквијум је у овом случају представљао пост-тест чија је сврха процена

способности и успеха студената након спроведених различитих метода учења. Резултати колоквијума приказани су у табели 15.

Табела 15. Резултати студената остварени на пост-тесту (колоквијуму) (макисималн број поена 25)

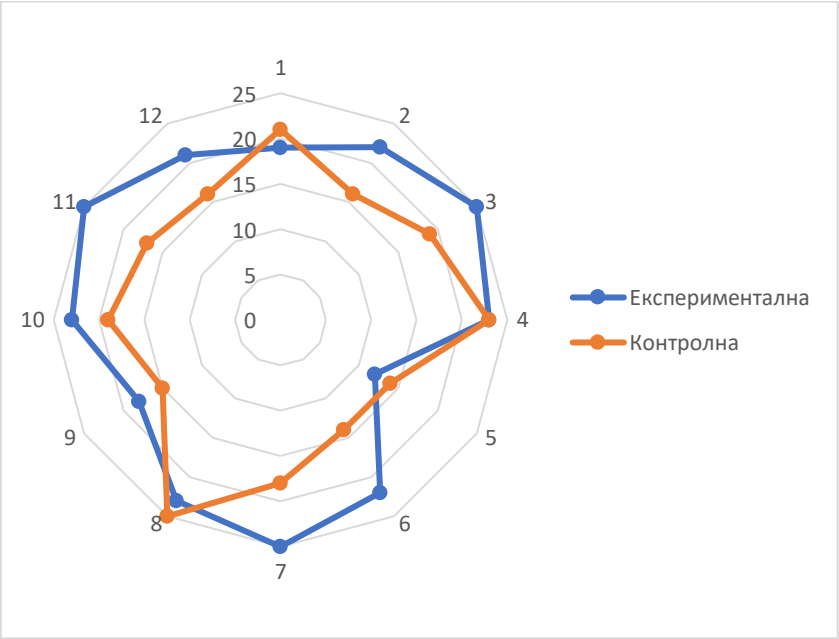
Група	**Број студената**	**Средња вредност**	**Стандардна девијација**
Експериментална	12	21.500	3.729
Контролна	12	18.083	3.502

Упоређивањем средњих вредности остварених резултата на колоквијуму, може се уочити напредак експерименталне групе. Дескриптивна статистика (Ф-тест и т-тест), погодна за анализу добијених резултата, приказана је у табели 16. У овом случају резултат т-теста је указао на одбацивање нулте хипотезе, то јест показао је значајну разлику између две групе (*p-value*=0,03<0,05). Дакле, то имплицира да су студенти у експерименталној групи стекли дубље разумевање теоријских концепата и алгоритама из области програмских преводилаца.

Табела 16. Дескриптивна статистика остварених резултата на колоквијуму (пост-тесту)

***F*-Test Two-Sample for Variances**			***t*-Test: Two-Sample Assuming Equal Variances**				
F	**P(F<=f) one-tail**	**F Critical one-tail**	**t Stat**	**P(T<=t) one-tail**	**t Critical one-tail**	**P(T<=t) two-tail**	**t Critical two-tail**
1,134	0,419	2,818	2,313	0,015	1,717	0,030	2,074

Претходно су разматрани резултати група, а на слици 81 дат је графички приказ индивидуалних резултата студената из обе групе. На графику се може јасно увидети предност експерименталне у односу на контролну групу. У експерименталној групи максималан број поена освојила су три студента, док у контролној само један. У експерименталној групи испод 20 поена имају три студента, док у контролној чак девет.



Слика 81. Графички приказ индивидуалних резултата студената из експерименталне и контролне групе

Ово су само прелиминарни резултати о ефикасности ComVis веб базираног симулационог система добијени на малој групи студената, али су свакако охрабрујући за наставак истраживања. У наредном периоду потребно је праћење резултата учења нових генерација студената којима ће употреба ComVis алата бити обавезна у оквиру лабораторијских вежби.

7.2 ПРОЦЕНА УПОТРЕБЉИВОСТИ НА ОСНОВУ КОРИСНИЧКОГ ИСКУСТВА

Поред процене ComVis система на основу мерљивих резултата, спроведена је и квалитативна процена применом анкете. Након завршеног колоквијума, од студената који су користили ComVis у процесу учења затражено је да попуне упитник о употребљивости симулатора. Анкета је била анонимна и састојала се од питања са слободним одговорима на основу Ликертове скале (уопште се не слажем, не слажем се, немам мишљење, слажем се, потпуно се слажем). Овај начин процене и питања слични су онима у раду

(Stanisavljević et al., 2014). Питања и сумирани резултати анкете приказани су у табели 17.

Табела 17. Резултати анкете студената (повратне информације о искуству са коришћењем ComVis система)

	Питања	**Резултати (%)**				
		1	2	3	4	5
1	ComVis систем ми је помогао да боље разумем материјал курса.	0	16.7	16.7	41.7	25.0
2	ComVis систем је допринео већем ангажовању на лабораторијским вежбама.	0	8.3	8.3	25.0	58.3
3	Кориснички интерфејс ComVis система је интуитиван и једноставан за коришћење.	8.3	0	33.3	41.7	16.7
4	Апстракције које се користе за представљање концепата су једноставне за разумевање.	0	16.7	8.3	33.3	41.7
5	ComVis систем је утицао на повећање мотивације.	0	8.3	16.7	25.0	50.0
6	Моје искуство коришћења ComVis система је било одлично.	8.3	8.3	0	58.3	25.0

Да би резултати анкете били прегледнији и једноставнији за анализу, коришћене су оцене од 1 до 5 које одговарају одговорима од „уопште се не слажем“ до „потпуно се слажем“. Просечне оцене за свако питање приказане су у табели 18.

Табела 18. Просечне оцене по сваком питању из анкете

Питање	**1**	**2**	**3**	**4**	**5**	**6**
Средња вредност	3,75	4,33	3,58	4,00	4,17	3,83

Питање број 2 и 5 имају најбоље оцене, што указује на то да је ComVis испунио и педагошки циљ који се односи на повећање пажње и мотивације студената. Питање у вези интуитивности система је добило добру оцену, али је свакако најмања у односу на остале. Студенти који су имали неко предзнање

о одговарајућим темама су високо оценили интуитивност алата. За остале студенте је била потребна мала помоћ приликом првог коришћења. У разговору са студентима установљено је да би предложени почетни пример у сваком од алата ComVis система био од велике помоћи. Због тога је након ове анкете додата опција за унос примера у сваком подмодулу. Студенти могу да отворе предложене примере или да одмах крену са својим. Оцене на питањима број 1 и 4 показују да је ComVis систем помогао студентима у разумевању материјала курса и да је примењена метода симулације била једноставна за праћење. На крају анкете студенти су имали и могућност да оставе коментар како би изнели неки уочени проблем или предлог за унапређење симулатора. Неки студенти су сматрали да симулатор треба да има вишејезични интерфејс. Пар студената је сматрало да симулатору недостаје детаљније упутство за коришћење.

7.3 ХЕУРИСТИЧКА ЕВАЛУАЦИЈА

Код хеуристичке методе процене употребљивости и функционалности образовног алата, више евалуатора врши независну инспекцију софтвера на основу унапред дефинисане листе критеријума (Nielsen and Mack, 1994). Претрагом литературе пронађен је већи број радова који предлажу критеријуме за хеуристичку евалуацију образовних софтвера (Nielsen and Mack, 1994; Barker and King, 1993; Quinn, 1996; Iglesias, Paniagua and Pessacq, 1997). За евалуацију веб базираног ComVis система коришћена је метода предложена у (Iglesias, Paniagua and Pessacq, 1997). Метода се заснива на општем образацу за тестирање универзитетског образовног софтвера. Анкетни образац се састоји од 24 питања класификована у три категорије:

- Садржај и методологија;
- Карактеристике дизајна;
- Реакција корисника.

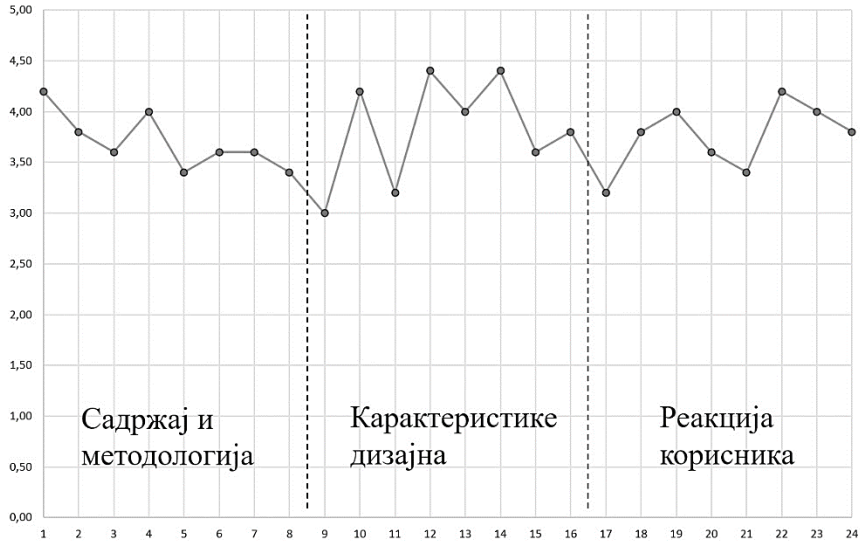
За сваку категорију се израчунава просечна оцена и стандардна девијација. Свако питање се оцењује оценом од 1 до 5. Комплетна листа хеуристика приказана је у (Iglesias, Paniagua and Pessacq, 1997).

Хеуристичка евалуација образовног система ComVis је подразумевала процену евалуатора на основу предложене листе хеуристика. Процену је вршило пет стручних евалуатора из две различите високошколске установе. У раду (Nielsen and Mack, 1994) анализом се дошло до закључка да је 3-5 евалуатора довољно за идентификовање корисних резултата у процени употребљивости. Сваки евалуатор је добио прилику да уради два или више пролаза кроз сваки алат симулационог система са циљем да испита ток симулација од екрана до екрана, као и специфичне карактеристике сваког појединачног алата. Резултати примењене хеуристичке методе, на основу узорка од пет евалуатора, приказани су у табели 19.

Табела 19. Резултати хеуристичке евалуације на основу методе представљене у (Iglesias, Paniagua and Pessacq, 1997)

Категорија евалуације	**Средња вредност**	**Стандардна девијација**
Садржај и методологија	3.70	0.714
Карактеристике дизајна	3.83	0.647
Реакција корисника	3.75	0.704

На основу сумарних резултата из табеле 19, може се закључити да је ComVis систем добио веома добре оцене које су се кретале у распону од 3 до 5. Ако се посматрају просечне оцене и стандардне девијације категорија евалуације, може се приметити да је категорија која се односи на карактеристике и функционалности софтверског система најбоље оцењена. Поред ових генералних закључака, вредна је и провера најлошијих и најбољих оцена за свако питање јер открива предности и, што је још важније, недостатке у софтверу који се тестира. Графички приказ резултата на основу просечних оцена по сваком питању може се видети на слици 82.



Слика 82. Графички приказ резултата хеуристичке евалуације по моделу (Iglesias, Paniagua and Pessacq, 1997)

Из категорије *Садржај и методологија* најбоље је оцењено питање Q1 које се односи на оправданост употребе рачунарског софтвера као помоћног наставног средства у домену програмских преводилаца. Питања Q12 и Q14 која се односе на квалитет презентације софтверског система и ефикасност коришћења ресурса (графика, табела, текстова…) су најбоље оцењена у другој категорији, али и генерално. У трећој категорији евалуатори су били задовољни степеном интеракције између корисника и софтвера (Q22). Генерално гледано, најлошије резултате имају питања Q9 и Q17 која се односе на смернице за коришћење и упутства за употребу софтвера, као и питање Q11 које се односи на потребно предзнање. Слични закључци су произашли и из претходне евалуације.

На крају је неопходно извршити и компарацију ComVis-а са осталим софтверским алатима погодним за учење програмских преводилаца. За ту сврху је најбоље употребити методу која је представљена у поглављу 4 и којом су процењени одабрани софтверски системи. У табели 20 приказана је процена ComVis-а на основу карактеристика и функционалности, док је у табели 21 приказана процена на основу покривености тема на предмету Програмски преводиоци методом успостављеном у поглављу 4.

Табела 20. Резултати евалуације ComVis-а на основу карактеристика и функционалности методом успостављеном у поглављу 4

Софтверски систем	***Критеријум***										
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
ComVis	Веб	Да	Да	Да	Вишенаменски	Напредни	Да	Да	Да	Да	Да

Табела 21. Резултати евалуације ComVis-а алата на основу покривености тема методом успостављеном у поглављу 4

Наставне јединице	**Теме**	**ComVis**
Лексичка анализа	Лексички анализатор	+
	Лексичка правила	+
	Регуларни изрази	+
Коначни аутомати	Детерминистички коначни аутомати	+
	Минимизација аутомата	+
	Недетерминистички коначни аутомати	+
	Конверзија NFA у DFA	+
	Конверзија регуларног израза у NFA	+
	Конструкција DFA директно из регуларног израза	+
	Конверзија DFA у регуларни израз	+
	Потисни аутомати	-
Синтаксна анализа	Граматика	+
	Стабло извођења	+
	Нерекурзивно парсирање	+
	Синтаксна анализа наниже (LL парсирање)	+
	Синтаксна анализа навише (LR парсирање)	+
Семантичка анализа	Табеле симбола	+
	Синтаксно управљане дефиниције	+
	Атрибутивне граматике (синтетизовани и наслеђени атрибути)	+
	Конструкција анотираног синтаксног стабла	+
	Транслационе шеме	-
Међукод	Апстрактно синтаксно стабло	-
	Троадресни код	-
	Оптимизација међукода	-
Генерисање кода	Генератор кода	-
	Алокација регистра	-
	Rezultat	19
	%	73,07

У погледу критеријума који се односе на карактеристике и функционалности система може се констатовати да ComVis испуњава све постављене захтеве и да је ComVis функционално најнапреднији образовни систем за учење програмских преводилаца. Када је у питању евалуација на основу критеријума покривености, најбоље оцењени алати покривају 46% тема предмета. Евидентно је да ComVis и у погледу покривености тема превазилази све остале алата. ComVis покрива четири наставне јединице од укупно шест и остварује 73% покривености тема на предмету Програмски преводиоци.

8. ЗАКЉУЧАК

У овој дисертацији представљен је веб базирани софтверски систем ComVis за симулацију и визуелизацију алгоритама и теоријских концепата из области програмских преводилаца. Традиционалним подучавањем поступка програмског превођења студенти добијају само површно знање кроз апстрактне теоријске материјале, па студенти на предмету Програмски преводиоци имају проблема са разумевањем и праћењем наставног плана, што се касније негативно одражава на њихове резултате на испиту. Програмски језици су студентима интересантна дисциплина јер бројна развојна окружења омогућавају брзу и једноставну проверу исправности написаног програмског кода. Међутим, код учења конструкције програмских преводилаца, повратна информација о примењеним техникама и алгоритмима је управо то што недостаје. Главни циљ ове дисертације је био проналазак адекватног решења у виду софтверског система које ће студентима пружити помоћ при учењу на предмету Програмски преводиоци. Систем треба да студентима поједностави процес учења апстрактне теорије компајлера, уз повећање мотивације и анагажовања на лабораторијским вежбама.

Бројне студије показују да интерактивни софтверски алати за помоћ у учењу могу подстаћи мотивацију студената. Детаљном претрагом литературе установљено је да постоји већи број софтверских система који могу симулирати одређене теме из области програмских преводилаца. Направљена је анализа постојећих решења како би се утврдило које теме сваки од система у наставном предмету покрива и какве су му карактеристике, а на основу добијених резултата и процена које карактеристике би нови систем требало да има да би се успешно користио у настави. Тако је успостављен модел за оцену квалитета симулационих система са аспекта покривености тема, као и основних карактеристика и функционалности. Анализом је утврђено да иако постоји већи број таквих система, ниједан не поседује одговарајуће карактеристике и функционалности, као ни довољану покривеност тема

предмета Програмски преводиоци. Највише су заступљени симулациони системи који симулирају рад коначних аутомата. Постоји и мањи број система који презентују процес парсирања.

На основу резултата анализе и прегледа релевантне литературе, истакнути су и класификовани циљеви за дизајн образовног алата ComVis. Дефинисани су технички, педагошки као и циљеви употребљивости који су представљали модел за дизајн новог образовног система. На основу успостављеног модела развијен је веб базирани софтверски систем ComVis који подржава већи број алгоритама који се примењују у различитим фазама програмског превођења. Симулација и визуелизација алгоритама врши се у оквиру истог интуитивног корисничког интерфејса. Систем је модуларано пројектован тако да омогућава једноставну надоградњу додавањем нових алгоритама. ComVis, као образовни софтвер, испунио је све педагошке норме и прилагођен је како студентима тако и предавачима.

Допринос и значај развијеног ComVis система огледа се у унапређењу наставног процеса, његовом применом као помоћног наставног средства у оквиру лабораторијских вежби на предмету Програмски преводиоци. Резултати спроведених евалуација управо то и потврђују. Експерименталним путем је потврђено да предложена метода за учење програмских преводилаца побољшава успешност студената и олакшава им стицање знања. Квантитативна евалуација ефикасности система контролисаним експериментом показала је да је употреба ComVis-а у процесу учења довела до дубљег разумевања функционисања компајлера од стране студената експерименталне групе. Квалитативне евалуације су указале на неке недостатке које треба отклонити у наредном периоду. Првенствено треба радити на формирању детаљног корисничког упутства, као и додатним смерницама које помажу самосталном коришћењу алата. Анализом је такође утврђено да је ComVis најкомплетнији образовни систем у погледу покривености тема које се изучавају на предмету Програмски преводиоци, као и са аспекта карактеристика и функционалности. Систем може да подржи

лабораторијске вежбе које покривају теме из области регуларних језика и граматика, коначних аутомата, лексичке, синтаксне и семантичке анализе.

У оквиру докторске дисертације остварени су следећи научни доприноси:

- Генерални преглед области образовних софтверских система за помоћ у учењу, уз осврт на неопходност интерактивне компоненте у образовном софтверу, али и на моћ визуелизације која може допринети ефективнијем преносу знања;
- Креиран модел за оцену квалитета постојећих система на основу развијених критеријума;
- Процена квалитета постојећих система погодних за учење тема програмских преводилаца на основу успостављеног модела за оцену квалитета;
- Генерализација функционалности и покривености тема постојећих система за учење програмских преводилаца;
- Дефинисање ефикасног модела за дизајн и развој интерактивног софтверског система за симулацију и визуелизацију алгоритама који се примењују у различитим фазама програмског превођења;
- Имплементација новог софтверског система и предлог начина примене;
- Квантитативна евалуација ефикасности система спроведена контролисаним експериментом и квалитативна евалуација употребљивости спроведена анкетом студената и хеуристичким испитивањима.

Позитивни резултати досадашњег истраживања обавезују на наставак развоја и усавршавања ComVis система, како у погледу функционалности, тако и у погледу покривености нових тема. Систему недостају модули за учење тема које се односе на генерисање међукода и циљног кода како би се заокружио комплетан процес програмског превођења.

ЛИТЕРАТУРА

1. Adams, E.S., Carswell, L., Kumar, A., Meyer, J., Ellis, A., Hall, P. and Motil, J., (1996). Interactive multimedia pedagogies: report of the working group on interactive multimedia pedagogy. *ACM SIGCSE Bulletin*, 28(SI), pp.182-191.
2. Adar, E., (2006). Guess: A language and interface for graph exploration. *Proceedings of the 2006 Conference on Human Factors in Computing Systems*, pp. 347–363.
3. Aho, A.V., Lam, M.S., Sethi, R. and Ullman, J.D., (2007). *Compilers: Principles, techniques, and tools second edition*. Boston: Pearson Education, Inc.
4. Ainsworth, S., (2008). How Do Animations Influence Learning. *Current Perspectives on Cognition, Learning, and Instruction: Recent Innovations in Educational Technology That Facilitate Student Learning*, pp. 37–67.
5. Ajzenhamer, N. and Bukurov, A., (2020). *Prevođenje programskih jezika*, drugo izdanje. Beograd: Matematički fakultet.
6. Alblas, H., (1991). Attribute evaluation methods. In *Attribute Grammars, Applications and Systems, vol 545,* Springer, pp. 48-113.
7. Almeida-Martínez, F.J. and Urquiza-Fuentes, J., (2009). Syntax trees visualization in language processing courses. In *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, pp. 597-601.
8. Almeida-Martınez, F.J., Urquiza-Fuentes, J. and Velázquez-Iturbide, J., (2009). Visualization of syntax trees for language processing courses. *Journal of Universal Computer Science*, 15(7), pp.1546-1561.
9. Alruwais, N., Wills, G. and Wald, M., (2018). Advantages and challenges of using e-assessment. *International Journal of Information and Education Technology*, 8(1), pp. 34-37.

10. Al-Zobaidi, Z., (2014). *Coherent minimisation: aggressive optimisation for symbolic finite state transducers.* Doctoral dissertation, University of Birmingham.

11. Amershi, S., Carenini, G., Conati, C., Mackworth, A.K. and Poole, D., (2008). Pedagogy and usability in interactive algorithm visualizations: Designing and evaluating CIspace. *Interacting with Computers*, 20(1), pp.64-96.

12. Andrew, W.A. and Jens, P., (2002). *Modern compiler implementation in Java*, second edition. Cambridge University Press.

13. Ardito, C., De Marsico, M., Lanzilotti, R., Levialdi, S., Roselli, T., Rossano, V. and Tersigni, M., (2004). Usability of e-learning tools. In *Proceedings of the working conference on Advanced visual interfaces*, pp. 80-84.

14. Arnaiz-González, Á., Díez-Pastor, J. F., Ramos-Pérez, I. and García-Osorio, C. (2018). Seshat—a web-based educational resource for teaching the most common algorithms of lexical analysis. *Computer Applications in Engineering Education*, 26(6), 2255-2265.

15. Barker, P. and King, T. (1993). Evaluating interactive multimedia courseware - a methodology. *Computers in Education* 21 (4), 307-319.

16. Beauchamp, G. and Kennewell, S., (2010). Interactivity in the classroom and its impact on learning. *Computers and Education*, 54, pp. 759–766.

17. Belson, H. and Ho, J., 2012. Usability. *A fresh graduate's guide to software development tools and technologies*. Singapore: School of Computing, National University of Singapore.

18. Bétrancourt, M., (2005). The animation and interactivity principles in multimedia learning. *The Cambridge handbook of multimedia learning*, pp.287-296.

19. Blythe, S. A., James, M. C. and Rodger, S. H. (1994). LLparse and LRparse: visual and interactive tools for parsing. *ACM SIGCSE Bulletin*, 26(1), pp.208-212.

20. Bonwell, C.C. and Eison, J.A., (1991). *Active learning: Creating excitement in the classroom*. ASHE-ERIC higher education reports. Washington: The George Washington University.

21. Bozeman, W. C., (1999). *Educational Technology : Best Practices From America's Schools*. NY : Electronic & Database Publishing, Inc.

22. Branovic, I., Popovic, R., Jovanovic, N., Giorgi, R., Nikolic, B. and Zivkovic, M. (2014). Integration of simulators in virtual 3D computer science classroom. In *2014 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1-4.

23. Bravenboer, M. and Visser, E., (2008). Parse table composition. In *International Conference on Software Language Engineering*, Springer, Berlin, Heidelberg, pp. 74-94.

24. Brzozowski, J.A. and McCluskey, E.J. (1963). Signal flow graph techniques for sequential circuit state diagrams. *IEEE Transactions on Electronic Computers*, (2), pp.67-76.

25. Budiselic, I., Srbljic, S., and Popovic, M. (2007). Regexpert: A tool for visualization of regular expressions. *In EUROCON 2007-The International Conference on "Computer as a Tool"*, pp. 2387-2389.

26. Castro-Schez, J. J., Del Castillo, E., Hortolano, J. and Rodriguez, A. (2009). Designing and using software tools for educational purposes: FLAT, a case study. *IEEE Transactions on Education*, 52(1), pp. 66-74.

27. Chakraborty, P., Saxena, P. C. and Katti, C. P. (2013). A compiler-based toolkit to teach and learn finite automata. *Computer Applications in Engineering Education*, 21(3), pp. 467-474.

28. Chakraborty, P., Saxena, P. C., and Katti, C. P., (2011). Fifty years of automata simulation: a review. *ACM Inroads*, 2(4), pp. 59-70.

29. Chesnevar, C. I., Cobo, M. L., and Yurcik, W., (2003). Using theoretical computer simulators for formal languages and automata theory. *ACM SIGCSE Bulletin*, 35(2), pp. 33-37.

30. Chesnevar, C. I., González, M. P. and Maguitman, A. G., (2004). Didactic strategies for promoting significant learning in formal languages and automata theory. *ACM SIGCSE Bulletin*, 36(3), pp. 7-11.

31. Chickering, A. and Gamson, Z., (1987). Seven principles of good practice in undergraduate education. *AAHE Bulletin*, 39, pp. 3-7.

32. Chomsky, N., (1956). Three models for the description of language. *IRE Trans Inf Theory* 2(3), pp.113 – 124.

33. Chuda, D., Trizna, J. and Kratky, P., (2015). Android automata simulator. In *International Conference on e-Learning*, Vol. 15, p. 80.

34. Cooper, K. D. and Torczon, L., (2012). *Engineering a compiler*. Burlington: Elsevier, Inc.

35. De Jong, T., & Van Joolingen, W. R. (1998). Scientific discovery learning with computer simulations of conceptual domains. *Review of educational research*, 68(2), pp. 179-201.

36. Dewi, T., Risma, P., & Oktarina, Y. (2018). An fuzzy logic simulation as a teaching-learning media for artificial intelligence class. *Journal of Automation Mobile Robotics and Intelligent Systems*, 12(3), 3-9.

37. DÍaz, J.M., Dormido, S. and Rivera, D.E., (2015). *Interactive Education for Time-Domain Time Series Analysis using ITTSAE*. IFAC-PapersOnLine, 48(28), pp.751-756.

38. Djordjevic, J., Nikolic, B. and Milenkovic, A., (2005). FlexibleWeb-Based Educational System for Teaching Computer Architecture and Organization. *IEEE Transactions on Education*, 48(2), pp. 264-273.

39. Douglas, T., (2020). *Introduction to Compilers and Language Design*, Second Edition. Indiana: University of Notre Dame.

40. Economopoulos, G.R., (2006). *Generalised LR parsing algorithms*. Doctoral dissertation, Royal Holloway, University of London, UK.

41. Elissavet, G. and Economides, A.A., (2000), December. Evaluation factors of educational software. In *Proceedings International Workshop on Advanced Learning Technologies, Advanced Learning Technology: Design and Development Issues*, pp. 113-116. IEEE.

42. Evans, C. and Gibbons, N.J., (2007). The interactivity effect in multimedia learning. *Computers & Education*, 49(4), pp.1147-1160.

43. Farbood, M. M., Pasztor, E., Jennings, K. (2004). Hyperscore: a graphical sketchpad for novice composers. *IEEE Computer Graphics and Applications*, 24(1), pp. 50-54.

44. Feizi, A. and Wong, C.Y., (2012). Usability of user interface styles for learning a graphical software application. In *2012 International Conference on Computer & Information Science (ICCIS)*, Vol. 2, pp. 1089-1094. IEEE.

45. Ferdig, R.E., (2006). Assessing technologies for teaching and learning: understanding the importance of technological pedagogical content knowledge. *British journal of educational technology*, 37(5), pp. 749-760.

46. Fischer, C.N., Cytron R.K. and LeBlanc Jr, R.J., (2010). *Crafting a Compiler*. Boston: Pearson Education., Inc.

47. Fouh, E., Akbar, M., & Shaffer, C. A. (2012). The role of visualization in computer science education. *Computers in the Schools*, 29(1-2), 95-117.

48. Gagné, R.M., Briggs, L.J, Wager W.W., (1988). *Principles of Instructional Design*, 3rd edition. N.Y: Thomson Learning.

49. Gansner, E.R. and North, S.C., (2000). An open graph visualization system and its applications to software engineering. *Software: practice and experience*, 30(11), pp.1203-1233.

50. García-Osorio, C. I., Gómez-Palacios, C., and García-Pedrajas, N. (2008). A tool for teaching LL and LR parsing algorithms. *In Annual Conference on Innovation and Technology in Computer Science Education ITiCSE'08*, pp. 317-317.

51. Gee, J. P. (2004). *What video games have to teach us about learning and literacy*. New York: Palgrave Macmillan.

52. Geissinger, H. (1997). Educational software: Criteria for evaluation. In *Proceedings of ASCILITE*, vol. 97, pp. 219-225.

53. Goyal, M., and Sachdeva, S. (2009). Enhancing Theory of Computation teaching through integration with other courses. *International Journal of Recent Trends in Engineering*, 1(2), pp. 137-140.

54. Grinder, M. T., Kim, S. B., Lutey, T. L., Ross, R. J., and Walsh, K. F. (2002). Loving to learn theory: active learning modules for the theory of computing. *In ACM SIGCSE Bulletin*, 34(1), pp. 371-375.

55. Griswold, W.G., (2002). Teaching software engineering in a compiler project course. *Journal on Educational Resources in Computing (JERIC)*, *2*(4).

56. Grivokostopoulou, F. and Hatzilygeroudis, I., (2013). Teaching AI Search Algorithms in a Web-Based Educational System. *International Association for Development of the Information Society*, 83-90.

57. Hajduković, M. and Suvajdžin, Z., (2004). *Programski prevodioci*. Katedra za računarske nauke i informatiku, FTN, Novi Sad.

58. Hall, M., Padua, D. and Pingali, K., (2009). Compiler research: the next 50 years. *Communications of the ACM*, *52*(2), pp.60-67.

59. Hamada, M. and Sato, S. (2011). A game-based learning system for theory of computation using Lego NXT Robot. *Procedia Computer Science*, 4, 1944-1952.

60. Hamada, M. and Sato, S. (2012). A learning system for a computational science related topic. *Procedia Computer Science*, 9, 1763-1772.

61. Hamada, M., (2013). Turing machine and automata simulators. *Procedia Computer Science*, 18, pp.1466-1474.

62. Hamada, M., and Hassan, M. (2017). An Interactive Learning Environment for Information and Communication Theory. *Eurasia Journal of Mathematics, Science & Technology Education*, 13(1), pp. 35-59.

63. Hansen, S., Narayanan, N. H., & Hegarty, M. (2002). Designing educationally effective algorithm visualizations. *Journal of Visual Languages & Computing*, 13(3), pp. 291-317.

64. Harders, M., Bajka, M., Spaelter, U., Tuchschmid, S., Bleuler, H., Szekely, G. (2005). Highly-realistic, immersive training environment for hysteroscopy. *Studies in health technology and informatics*, vol. 119, pp. 176-181.

65. Hattie, J. and Jaeger, R., (1998). Assessment and classroom learning: A deductive approach. *Assessment in Education: principles, policy & practice*, 5(1), pp. 111-122.

66. Hopcroft, J., (1971). An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pp. 189-196.

67. Hossain, A.A.M.S., (2015). Evaluating and testing user interfaces for e-learning system: blackboard usability testing. *Journal of Information Engineering and Applications*, 5(1), p.23.

68. Huffman, D.A., (1954). The synthesis of sequential switching circuits. *Journal of the franklin Institute*, 257(3), pp.161-190.

69. Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3), 259-290.

70. Hundhausen, C.D., (2002). Integrating algorithm visualization technology into an Undergraduate Algorithms Course: ethnographic studies of a social constructivist approach. *Computers and Education*, 39 (3), pp.237–260.

71. Huppert, J., Lomask, S. M. and Lazarowitz, R., (2002). Computer simulations in the high school: Students' cognitive stages, science process skills and academic achievement in microbiology. *International Journal of Science Education*. 24(8), pp. 803-821.

72. Iglesias, O.A., Paniagua, C.N. and Pessacq, R.A., (1997). Evaluation of university educational software. *Computer Applications in Engineering Education*, *5*(3), pp.181-188.

73. Ignjatović, J. and Ćirić, M., (2016). *Automati i formalni jezici*. Niš: Prirodno-matematički fakultet.

74. Indu, J., (2016). Technique for Conversion of Regular Expression to and from Finite Automata. *International Journal of Recent Research Aspects*, Vol. 3(2), pp 62-64.

75. Issa, L. and Jusoh, S., (2019). Usability evaluation on gamified e-learning platforms. In *Proceedings of the Second International Conference on Data Science, E-Learning and Information Systems*, pp. 1-6.

76. Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, [online]. Available at: https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf

77. Jovanović, N. and Zakić, A., (2018). Network simulation tools and spectral graph theory in teaching computer network. *Computer Applications in Engineering Education*, 26(6), pp. 2084-2091.

78. Jovanović, N., (2023). *Programski prevodioci*. Fakultet tehničkih nauka, Kosovska Mitrovica.

79. Jovanović, N., Miljković, D., Stamenković, S., Jovanović, Z. and Chakraborty, P., (2020). Teaching concepts related to finite automata using ComVis. *Computer Applications in Engineering Education*.

80. Jovanović, N., Popović, R. and Jovanović, Z., (2009). WNetSim: a web-based computer network simulator. *International Journal of Electrical Engineering Education*, 46(4), pp. 383-396.

81. Jovanović, N., Popović, R., Marković, S. and Jovanovic, Z., (2012). Web laboratory for computer network. *Computer Applications in Engineering Education*, 20(3), pp. 493-502.

82. Jovanović, N., Stamenković, S. and Miljković, D., (2020). Vizuelni i interaktivni alat za učenje konačnih automata. In *19th International Symposium INFOTEH-JAHORINA*, pp. 18-20.

83. Jovanović, N., Stamenković, S., Cvjetković, M. and Jovanović, Z., (2022). Softverski alat kao pomoćno sredstvo za učenje leksičke i sintaksne analize. In *21st International Symposium INFOTEH-JAHORINA*, pp. 249-253.

84. Jovanović, N., Stamenković, S., Miljković, D. and Chakraborty, P., (2021). ComVIS—Interactive simulation environment for compiler learning. *Computer Applications in Engineering Education*, 30(1), pp.275-291.

85. Jovanović, N., Zakić, A., and Veinović, M., (2016). VirtualMeshLab: Virtual laboratory for teaching wireless mesh network. *Computer Applications in Engineering Education*, 24(4), pp. 567-576.

86. Kehoe, C., Stasko, J. and Taylor, A., (2001). Rethinking the evaluation of algorithm animations as learning aids: an observational study. *International Journal of Human-Computer Studies*, 54(2), pp.265-284.

87. Knuth, D.E., (1965). On the translation of languages from left to right. *Information and control*, 8(6), pp.607-639.

88. Konold, C., and Miller, C. D., (2005). *TinkerPlots: Dynamic data exploration*. Computer software. Emeryville, CA: Key Curriculum Press.

89. Koparan, T., and Yilmaz, G. K., (2015). The Effect of Simulation-Based Learning on Prospective Teachers' Inference Skills in Teaching Probability. *Universal Journal of Educational Research*, 3(11), pp. 775-786.

90. Kostić, V., (2017). *Kognitivno-vizuelni pristup zasnovan na grafičkom prikazu funkcije u rešavanju matematičkih problema*. Disertacija. Novi Sad: Prirodno matematički fakultet.

91. Kundra, D. and Sureka, A., (2016). Application of Case-Based Teaching and Learning in Compiler Design Course. *arXiv preprint arXiv:1611.00271*.

92. Kuroda, S.Y., (1964). Classes of languages and linear-bounded automata. *Information and control*, 7(2), pp.207-223.

93. Lawrence, A.W., Badre, A.M. and Stasko, J.T., (1994), October. Empirically evaluating the use of animations to teach algorithms. In *Proceedings of 1994 IEEE Symposium on Visual Languages*, pp. 48-54.

94. Lewis, H. R. and Papadimitriou, C. H., (1997). *Elements of the Theory of Computation,* Second Edition. New Jersey: Prentice-Hall. lnc.

95. Lindgren, R., and Schwartz, D., (2009). Spatial Learning and computer simulations in science. *International Journal of Science Education*, 31(3), pp. 419–438.

96. Linz, P., (2006). *An introduction to formal languages and automata*, 5th ed. London: Jones & Bartlett Learning.

97. Litchfield, B.C., (1993). Design Factors in Multimedia Environments: Research Findings and Implications for Instructional Design. *Annual Meeting of the American Ed. Res. Ass*., pp. 1-10.

98. Makarova, E.A., (2016). Visual culture in educational environment and innovative teaching technologies. *Universal Journal of Management*, 4(11), pp.621-627.

99. McCracken, D.D., & Reilly, E.D. (2003). Backus-naur form (bnf). In *Encyclopedia of Computer Science*, pp. 129-131.

100. McNaughton, R., & Yamada, H. (1960). Regular expressions and state graphs for automata. *IRE transactions on Electronic Computers*, 83(1), 39-47.

101. Mernik, M. and Zumer, V. (2003). An educational tool for teaching compiler construction. *IEEE Transactions on Education*, 46(1), pp. 61-68.

102. Mogensen, T.Æ., (2009). *Basics of compiler design*. Copenhagen: Department of Computer Science, University of Copenhagen.

103. Molina, O.E., Fuentes-Cancell, D.R. and García-Hernández, A., (2022). Evaluating usability in educational technology: A systematic review from the

teaching of mathematics*. LUMAT: International Journal on Math, Science and Technology Education*, 10(1), pp.65-88.

104. Moore, E.F., (1956). Gedanken-experiments on sequential machines. *Automata studies, 34*, pp.129-153.
105. Moura, P. and Dias, A. M. (2011). L-FLAT: Logtalk toolkit for formal languages and automata theory. *arXiv preprint arXiv:1112.3783*.
106. Muchnick, S., (1997). *Advanced compiler design and implementation*. San Francisco: Morgan kaufmann.
107. Nadrljanski, Đ. (2000). *Obrazovni softver – hipermedijalni sistemi*. Zrenjanin: Tehnički fakultet "Mihajlo Pupin", Univerzitet u Novom Sadu.
108. Nadrljanski, Đ., Nadrljanski, M., Soleša, D. (2008): *Digitalni mediji – obrazovni softver*, Univerzitet u Novom sadu, Pedagoški fakultet u Somboru.
109. Nahvi, M. (1996). Dynamics of student-computer interaction in a simulation environment: Reflections on curricular issues. *Proceedings of the IEEE Frontiers in Education*, USA, pp. 1383-1386.
110. Naps, T.L., Cooper, S., Koldehofe, B., Leska, C., Ro¨ ßling, G., Dann, W., Korhonen, A., Malmi, L., Rantakokko, J., Ross, R.J., Anderson, J., Fleischer, R., Kuittinen, M., McNally, M., (2003). Evaluating the educational impact of visualization. *ITiCSE*, pp. 124–136.
111. Naps, T.L., Eagan, J.R. and Norton, L.L., (2000). JHAVÉ—an environment to actively engage students in Web-based algorithm visualizations. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pp. 109-113.
112. Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. and Velázquez-Iturbide, J.A. (2002). Exploring the role of visualization and engagement in computer science education. In *Working group reports from ITiCSE on Innovation and technology in computer science education* (pp. 131-152).

113. Naveed, M. S. and Sarim, M., (2018). Didactic Strategy for Learning Theory of Automata & Formal Languages. *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences*, 55(2), pp. 55-67.

114. Nielsen, J. (1993). *Usability engineering*, Boston: Academic Press.

115. Nielsen, J. and Mack, R.L., (1994). *Usability Inspection Methods*. New York: John Wiley.

116. Norman, D.A. and Draper, S.W., (1986). *User centered system design: New perspectives on human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

117. Odadžić, V., (2016). *Efikasnost primene obrazovno računarskog softvera u nastavi biologije u gimnaziji*. Disertacija. Novi Sad: Prirodno matematički fakultet.

118. Oshima, C., Nishimoto, K., Suzuki, M. (2004). Family ensemble: a collaborative musical edutainment system for children and parents. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pp. 556-563.

119. Pfahl, D., Laitenberger, O., Ruhe, G., Dorsch, J. and Krivobokova, T., (2004). Evaluating the learning effectiveness of using simulations in software project management education: results from a twice replicated experiment. *Information and software technology*, *46*(2), pp.127-147.

120. Phillips, L.M., Norris, S.P. and Macnab, J.S., (2010). *Visualization in mathematics, reading and science education* (Vol. 5). Netherlands: Springer Science & Business Media.

121. Popov, S., Jukić, S. (2006). *Pedagogija*. Novi Sad: Centar za razvoj i primenu nauke, tehnologije i informatike.

122. Pradono, S., Astriani, M.S. and Moniaga, J., (2013). A method for interactive learning. *International Journal of Communication & Information Technology*, 7(2), pp. 46-48.

123. Prensky, M. (2001). *Digital game-based learning*. New York: McGraw Hill.

124. Price, B., Baecker, R., Small, I. (1997). *An introduction to software visualization*, In J. Stasko, J. Domingue, M. Brown, B. Price (Eds.) Software Visualization, Cambridge, MA: MIT Press, pp. 3–27.

125. Procopiuc, M., Procopiuc, O. and Rodger, S. H. (1996). Visualization and interaction in the computer science formal languages course with JFLAP. *In Frontiers in Education FIE'96 26th Annual Conference*, pp. 121-125.

126. Quinn, C. N. (1996). Pragmatic evaluation: lessons from usability. In *13th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education*, pp. 15-22.

127. Rabin, M.O. and Scott, D., (1959). Finite automata and their decision problems. *IBM journal of research and development*, *3*(2), pp.114-125.

128. Radosav, D., (2005). *Obrazovni računarski softver i autorski sistemi*. Zrenjanin: Tehnički fakultet „Mihajlo Pupin“ u Zrenjaninu, Univerzitet u Novom Sadu.

129. Rasmussen, K., and Salkind, N. J. (2008). *Encyclopedia of Educational Psychology*. SAGE Publications, Inc.

130. Ridgway, J., McCusker, S. and Pead, D., (2004). Literature review of e-assessment. *Futurelab, Bristol*.

131. Robinson, M. B., Hamshar, J. A., Novillo, J. E. and Duchowski, A. T. (1999). A Java-based tool for reasoning about models of computation through simulating finite automata and Turing machines. *In ACM SIGCSE Bulletin*, 31(1), pp. 105-109.

132. Rodger, S. H. and Finley, T. W. (2006). *JFLAP: an interactive formal languages and automata package*. London: Jones & Bartlett Publishers.

133. Rodger, S. H., Wiebe, E., Lee, K. M., Morgan, C., Omar, K. and Su, J. (2009). Increasing engagement in automata theory with JFLAP. *In ACM SIGCSE Bulletin*, 41(1), pp. 403-407.

134. Rutten, N., Van Joolingen, W. R., and Van der Veen, J. T., (2012). The learning effects of computer simulations in science education. *Computers & Education, Elsevier*, 58(2012), pp. 136-153.

135. Sangal, S., Kataria, S., Tyagi, T., Gupta, N., Kirtani, Y., Agrawal, S. and Chakraborty, P. (2018). PAVT: a tool to visualize and teach parsing algorithms. *Education and Information Technologies*, 23(6), pp. 2737-2764.

136. Schweitzer, D. and Brown, W., (2007). Interactive visualization for the active learning classroom. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pp. 208-212.

137. Scott, M.L., (2006). *Programming language pragmatics*, second edition. San Francisco: Morgan Kaufmann.

138. Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., & Edwards, S. H. (2010). Algorithm visualization: The state of the field. *ACM Transactions on Computing Education (TOCE)*, 10(3), 1-22.

139. Shneiderman, B. and Plaisant, C., (2010). *Designing the user interface: Strategies for effective human-computer interaction*. Pearson Education India.

140. Silius, K., Tervakari, A.M. and Pohjolainen, S., (2003). A multidisciplinary tool for the evaluation of usability, pedagogical usability, accessibility and informational quality of web-based courses. In *The Eleventh International PEG Conference: Powerful ICT for Teaching and Learning*, Vol. 28, pp. 1-10.

141. Sims, R., (1997). Interactivity: A forgotten art?. *Computers in human behavior*, 13(2), pp.157-180.

142. Singh, T., Afreen, S., Chakraborty, P., Raj, R., Yadav, S. and Jain, D., (2019). Automata Simulator: A mobile app to teach theory of computation. *Computer Applications in Engineering Education*, 27(5), pp.1064-1072.

143. Sipser, M., (2013). *Introduction to the Theory of Computation*, third edition. Boston: Cengage Learning.

144. Soderberg, P., and Price, F., (2003). An examination of problem-based teaching and learning in population genetics and evolution using EVOLVE, a computer simulation. *International Journal of Science Education*, 25(1), pp. 35-55.

145. Stamenković, S. and Jovanović, N. (2019). Comparative analysis of simulation system for teaching compilers. *BizInfo (Blace)*, 10(2).

146. Stamenković, S. and Jovanović, N., (2021a). Improving Participation and Learning of Compiler Theory Using Educational Simulators. In *2021 25th International Conference on Information Technology (IT)*, pp. 1-4. IEEE.

147. Stamenković, S. and Jovanović, N., (2021b). Obrazovni interaktivni alat za vizuelnu reprezentaciju leksičke analize. In *20th International Symposium INFOTEH-JAHORINA*, pp. 163-168.

148. Stamenković, S., Jovanović, N. and Chakraborty, P., (2020). Evaluation of simulation systems suitable for teaching compiler construction courses. *Computer Applications in Engineering Education*, 28(3), pp.606-625.

149. Stamenković, S., Jovanović, N., Vasović, B., Cvjetković, M. and Jovanović, Z., (2023). Software tools for learning artificial intelligence algorithms. *Artificial Intelligence Review*, pp.1-30.

150. Stanisavljević, Z., Stanisavljević, J., Vuletić, P., & Jovanović, Z. (2014). COALA-System for visual representation of cryptography algorithms. *IEEE Transactions on Learning Technologies*, 7(2), 178-190.

151. Stanković, M., Stojković, S. and Tošić, Ž., (2018). *Programski prevodioci*. Niš: Elektronski fakultet.

152. Storey, M.A., Phillips, B., Maczewski, M. and Wang, M., (2002). Evaluating the usability of Web-based learning tools. *Journal of educational technology & society*, 5(3), pp.91-100.

153. Su, Y. and Yan, S.Y., (2011). *Principles of Compilers*. London: Springer.

154. Sudkamp, T.A., (2007). *Languages And Machines: An Introduction To The Theory Of Computer Science*, third edition. India: Pearson Education.

155. Taher, M., and Khan, A., (2014). Impact of Simulation-based and Hands-on Teaching Methodologies on Students' Learning in an Engineering Technology Program. *Proceedings of the ASEE Annual Conference & Exposition*, pp. 1-22.

156. Tchounikine, P., (2011). *Computer science and educational software design: A resource for multidisciplinary work in technology enhanced learning*. Springer Science & Business Media.

157. Thakur, J., and Kumar, N. (2011). DES, AES and Blowfish: Symmetric key cryptography algorithms simulation based performance analysis. *International journal of emerging technology and advanced engineering*, 1(2), pp. 6-12.

158. Thompson, K., (1968). Programming techniques: Regular expression search algorithm. *Communications of the ACM*. 11(6), pp.419-422.

159. Thorpe, M. (1993). *Evaluating open and distance learning*. Harlow: Longman Open Learning.

160. Turing, A.M., (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1), pp.230-265.

161. Tversky, B., Morrison, J.B., Betrancourt, M., (2002). Animation: can it facilitate? *International Journal of Human–Computer Studies* 57, pp.247–262.

162. Urquiza-Fuentes, J. and Manso, F., (2011). *A second evaluation of SOTA*. Spain: Universidad Rey Juan Carlos.

163. Urquiza-Fuentes, J., & Velázquez-Iturbide, J. Á. (2009). A survey of successful evaluations of program visualization and algorithm animation systems. *ACM Transactions on Computing Education (TOCE)*, 9(2), 1-21.

164. Urquiza-Fuentes, J., Manso, F., Velázquez-Iturbide, J.Á. and Rubio-Sánchez, M., (2011). Improving compilers education through symbol tables animations.

In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pp. 203-207.

165. Vayadande, K.B., Sheth, P., Shelke, A., Patil, V., Shevate, S. and Sawakare, C., (2022). Simulation and Testing of Deterministic Finite Automata Machine. *International Journal of Computer Sciences and Engineering*, 10(1), pp.13-17.

166. Velazquez-Iturbide, J. A., Debdi, O., Esteban-Sanchez, N., & Pizarro, C. (2013). GreedEx: A visualization tool for experimentation and discovery learning of greedy algorithms. *IEEE Transactions on Learning Technologies*, 6(2), pp. 130-143.

167. Vieira, L. F. M., Vieira, M. A. M. and Vieira, N. J. (2004). Language emulator, a helpful toolkit in the learning process of computer theory. *In ACM SIGCSE Bulletin*, 36(1), pp. 135-139.

168. Wade, V.P. and Ashman, H., (2007). Guest editors' introduction: Evolving the infrastructure for technology-enhanced distance learning. *IEEE Internet Computing*, 11(3), pp. 16-18.

169. Waite, W.M. and Goos, G., (1996). *Compiler construction*. Springer-Verlag, Berlin, New York Inc.

170. Wang, J. T., Hung, L. C., Hsieh, H. M., Tsai, J. T., & Lin, I. H. (2012). Computer technology integration and multimedia application for teacher professional development: The use of instructional technology in the classroom settings. *IERI Procedia*, 2, 616-622.

171. Wang, P.Y., Vaughn, B.K. and Liu, M., (2011). The impact of animation interactivity on novices' learning of introductory statistics. *Computers & Education*, 56(1), pp.300-311.

172. Wang, Q., (2008). A generic model for guiding the integration of ICT into teaching and learning. *Innovations in education and teaching international,* 45(4), pp.411-419.

173. Ware, C., (2019). *Information visualization: perception for design*, 3rd Edition. San Francisco: Morgan Kaufmann Publishers is an imprint of Elsevier.

174. Watson, B.W., (1993). *A taxonomy of finite automata minimization algorithms*. Eindhoven University of Technology, Department of Mathematics and Computing Science, Netherlands.

175. White, T. M. and Way, T. P. (2006). jFAST: A java finite automata simulator. In *ACM SIGCSE Bulletin*, 38(1), 384-388.

176. Wirth, N., (2005). *Compiler Construction*. Zürich: Addison-Wesley.

177. Witzig, A., Prandini, M., Wolf, A., and Kunath, L. (2016). Teaching Renewable Energy Systems by Use of Simulation Software: Experience at Universities of Applied Sciences, in In-Service Training, and from International Know-How Transfer. *Presented at and in the proceedings of Eurosun 2016: International Conference on Solar Energy and Buildings*, Spain, pp. 1591-1600.

178. Wolfram, S., (2003). *The Mathematica Book*, 5th ed. Wolfram Media, Inc.

179. Wood D., (1987). *Theory of Computation*. New York: John Wiley.

180. Woolf, B.P. and Hall, W., (1995). Multimedia Pedagogues: Interactive Systems for Teaching and Learning. *IEEE Computer, Special Issue on Multimedia*, 28(5), pp.74-80.

181. Xing, G., (2004). Minimized thompson NFA. *International Journal of Computer Mathematics*, 81(9), pp.1097-1106.

182. Yin, J., Wang, Y., Hsu, D. (2005). Digital violin tutor: an integrated system for beginning violin learners. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pp. 976-985.

183. Yingjie, X.U., 2009. Describing an nlog n algorithm for minimizing states in deterministic finite automaton.

184. Zhang, W. and Jie, L., (2018). Application of Simulation Software in Analog Electronic Technology Teaching. *DEStech Transactions on Social Science, Education and Human Science*, (ICESSH 2018), pp. 6-10.