МАТЕМАТИЧКИ ОМНИБУС, 7 (2020), 107 – 122
http://im-pmf.weebly.com/matematicki-omnibus-kniga-7.html

ФОРМАЛНИ ГРАМАТИКИ И НИВНА ПРИМЕНА ВО ДИЗАЈН НА ПРОГРАМСКИ ЈАЗИЦИ

Марија Михова [1]

Програмските јазици се средство за комуникација помеѓу луѓето и машините. Тие се всушност нотација за пишување програми со кои можат да се вршат пресметки или да се извршуваат алгоритми. Затоа некои автори го ограничуваат терминот „програмски јазик“ на оние јазици со кои што можат да се изразат сите можни алгоритми, [5]. Како што во еден говорен јазик се користат граматички правила и азбука за негово испишување, така и програмските јазици се состојат од азбука, речник од зборови и збир на граматички правила кои треба да се следат при формирање на „реченици“. За разлика од граматиките на говорните јазици, граматиките на програмските јазици се поедноставни и можат да се опишат со прецизни математички модели, познати како формални јазици.

Теоријата на формални јазици е дел од поширокаата математичка теорија за пресметливост, која обезбедува систематска терминологија и збир од конвенции за опишување на граматичките правила и структури кои се генерираат со нив, заедно со богата теорија од откритија и теореми [4, 6, 8]. Нејзините основи почнуваат со апстрактните машини познати како машини на Тјуринг [10] и Пост [7]. Со нив се воведува прецизна математичка теорија која опишува што е можно да биде пресметано. Модерната форма на оваа теорија е воведена десетина години подоцна, од лингвистот Ноам Чомски [1, 2], кој се обидувал да даде прецизна карактеризација на структурата на природните јазици. Имено, тој сакал да ја дефинира синтаксата на јазиците користејќи едноставни и прецизни математички правила и вовел класификација и градација на јазиците според нивната комплексност, која е позната како хиерархија на Чомски. Подоцна било утврдено дека една од класите кои ги вовел, класата на контекстно слободни јазици, е доволна за да се опише синтаксата на програмските јазици. Тоа го поттикнува развојот на програмирањето. Тешката инженерска работа за која биле способни само

многу мал број луѓе, претежно оние кои ги дизајнирале машините, се прилагодила до ниво сфатливо за многумина.

Инаку, самата теорија на јазици има примена во широк ранг на дисциплини. Освен во лингвистиката, оваа теорија се користи и во психологијата за истражување на способноста за учење кај луѓето и кај животни, во неврологијата се користи при експериментирање со невронски слики, во биологијата се користи за анализирање на структурата на РНК. Сепак, нејзината најзначајна примена е во компјутеските науки, бидејќи овде ја дава теоретската основа на програмските јазици и дизајнот на компајлерите како преведувачи на програмскиот јазик. Нејзината примена овде не е ограничена само на програмските јазици, туку таа се користи како модел или идеја за модел во секој дел од компјутеските науки.

Идејата на овој труд е да даде повеќе интуитивна претстава за поимот граматика, отколку да навлегува во прецизната теорија. Целта е преку примери да се долови како се креираат граматичките правила со цел истите да можат да се применат во дизајн на програмски јазици и компајлери.

1. КАКО РАБОТИ КОМПАЈЛЕРОТ

За двајца да можат да комуницираат меѓу себе, треба да го зборуваат и разбираат истиот јазик. Но, и ако еден од нив зборува јазик што е разбирлив за другиот, тогаш нему може да му пренесе порака и тој ќе ја разбере. Ако вториот човек не го разбира јазикот на којшто зборува првиот, тогаш комуникацијата може да се изведе преку посредник кој од јазикот со кој зборува едниот ќе преведува во јазик разбирлив за другиот. Буквално ваква е комуникацијата меѓу човекот и компјутерот, при што посредникот или преведувачот е посебна програма која може да биде или компајлер или интерпретер. Компјутерските програми обично се пишуваат во така наречен виш јазик, разбирлив за нас луѓето. Ваквиот код се нарекува изворен код. Компјутерот не разбира виш јазик, па компајлерот го преведува изворниот код во машински код, т.е. програма напишана со нули и единици, разбирлива за компјутерот. Вообичаено, програмата не се преведува директно во

бинарна низа, туку во некој меѓујазик, наречен асемблер, што е ниж јазик, па тој код потоа се преведува понатаму. Асембли јазикот или нижиот јазик луѓето потешко го разбираат, но добрите компјутерски инженери ги разбираат и ваквите јазици.

Кога инсталираме некој програмски јазик, како C++, Јава или Пајтон, всушност го инсталираме неговиот компајлер или неговиот интерпретер. Суштинската разлика меѓу овие два преведувачи е тоа што компајлерот го преведува изворниот код целосно, па дури потоа ја извршува програмата, додека интерпретерот го извршува наредба по наредба. Некои програмски јазици како Пајтон користат интерпретер, а други, како Јава и C++ користат компајлер. Нема потреба детално да навлегуваме во суштината на разликата меѓу овие два преведувачи, затоа што граматиките немаат голема одговорност за таквите детали, но можеме да дадеме едноставна илустрација. На пример, ако го имаме изразот „5+3“ напишан во симболички математички јазик, него можеме да го преведеме на македонски како „пет плус три“ или на англиски како „five plus three“. Вака би сработил компајлерот. Интерпретерот тоа веднаш би го извршил и би одговорил „8“, но тоа не значи дека го нема преведено во јазик разбирлив за него, туку дека освен што го превел, дополнително го пресметал и како одговор го зачувал само резултатот, а не целиот превод.

Слика 1. Илустрација на превод од изворен во асембли код.

На Слика 1 е претставено како изгледа програма напишана во виш јазик, дизајниран врз основа на македонскиот јазик и нејзиниот превод во ниж јазик. И двата јазика претставени овде се дизајнирани од

студентите на предметот Компајлери на Институтот за информатика на ПМФ, пред десетина години, [3]. Од илустрацијата може да се види дека текст напишан во еден јазик треба да се преведе во текст напишан во друг јазик, исто како што тоа се прави со говорните јазици. Ајде да анализираме што ни е потребно за превод од еден во друг говорен јазик! Прво, потребно е да ги разбереме речениците напишани во првиот јазик, а за да ги разбереме речениците, треба да го знаеме значењето на зборовите од една страна, но и начинот на подредување на тие зборови, затоа што различен редослед дава различна смисла на текстот. За да ги разбереме зборовите, потребно да ги знаеме буквите или да ги разбираме знаците од тоа писмо. За да го направиме преводот во вториот јазик потребно е сите овие работи да ги познаваме и за вториот јазик. Во овој текст ќе се фокусираме само на првиот дел односно како да го разбереме текстот, без да се задржуваме на техниките за превод во целниот јазик. Имено, граматиката на вишиот јазик е одговорна само за разбирање на текстот, па треба да биде дизајнирана така да може точно да ја разбере секоја реченица, која овде се нарекува наредба.

За разлика од говорните јазици, коишто прво биле создадени, па потоа се анализирале нивните граматики, програмските јазици вештачки ги креираме, па можеме граматиката да ја направиме доволно едноставна за врз неа да може да се направи автоматски превод. Интересно е, колку поедноставна граматика имаме, толку јазикот е потежок за човекот. За човекот интуитивно некои работи се потешки од други, па потешко му е да процесира реченица во програмски јазик отколку во природен јазик, како што му е потешко да игра шах отколку да вози. Од друга страна, за машината е обратно – многу е полесно да се направи машина која игра шах на многу високо ниво, отколку машина која ќе вози автомобил, [9]. Така, оваа едноставност е од гледна точка на машината, но сепак во голема мера се води сметка јазикот да биде што е можно поразбирлив за човекот, од една страна, но и поточно и попрецизно да ги наоѓа грешките кои се направени при програмирањето, од друга страна. Постојано се дизајнираат нови јазици во кои авторите даваат сè подобри решенија на двата претходно посочени аспекти.

2. ГРАМАТИКИ

Програмата што го преведува јазикот треба текстот да го чита карактер по карактер и да ги разбира зборовите. Дозволените карактери, или букви, се азбуката над која се креира јазикот. Зборовите можат да бидат броеви, имиња на функции или променливи, некои клучни зборови, на пример, cout или print кои што се користат за печатење, или знаци како што се + и =. Првиот чекор во преводот е препознавање на зборовите, што се нарекува жетонизирање, за што се користат регуларни граматики, наједноставниот тип на граматики претставени од Чомски. Овие жетони се аналог на речникот кај говорен јазик, а граматиките се правила со кои може да се формираат зборовите. На некој начин одговара на морфологија и фонетика во лингвистиката, бидејќи за секој од зборовите треба да се определи што значи, на пример 59 да се препознае како природен број или 23.5 како реален број. Во теоријата на компајлери ова е познато како лексичка анализа.

Некој правилен распоред на зборови образува наредба, која ја дава наредбата што треба да ја изврши компјутерот. За разбирањето на наредбите се користат контекстно слободни граматики. Овие контекстно слободни граматики всушност одговараат на она што го познаваме како граматика во говорниот јазик, т.е. начин на редење на зборовите во реченицата, за да се добие смислата на исказот. Значајната разлика меѓу говорните и програмските јазици е во тоа што граматиките на говорните јазици се контекстно зависни, што е највисокото скалило во хиерархијата на Чомски. Нема едноставно математичко решение за тие автоматски да се разберат. Во теоријата на компајлери, исто како и во лингвистиката, ова е познато како синтаксичка анализа.

Прво да ја дадеме формалната дефиниција за граматика, па ќе дадеме пример како тоа може да се примени на едноставен пример од македонскиот јазик.

Дефиниција 1. *Формална граматика* е подредена четворка G = (V, Σ, S, R) каде што:

Σ е азбука од која се земаат карактерите во јазикот;

V е конечно множество нетереминали, $V \cap \Sigma = \varnothing$;

$S \in V$ е почетен нетерминал.

R е конечно множество на правила од облик $v \rightarrow w$ каде што v,w се низи од карактери од $V \cup \Sigma \cup \{\lambda\}$, што се бележи со $(V \cup \Sigma)^*$, каде што λ е симбол што не е во $V \cup \Sigma$, а во v има барем еден симбол од V.

Правилата најчесто ги запишуваме во покомпактна форма, таканаречена БНФ форма (Backus-Naur form). Имено, ако имаме повеќе правила во кои се заменува *v*, тогаш десните страни од правилото ги редиме едно по друго одделени со симболот „|“ што означува „или“, и наместо стрелка користиме две точки.

Да разгледаме една едноставна граматика над македонската азбука и V={rec, im1, im2, kor, nast, pred} и S=rec, во која правилата во R се:

rec: im1 kornast pred im2,

im1: Автомобилите

im2 : тротоарот|улицата

kor : воз|свир

nast:еа|ат

pred: по|на|над

Со оваа граматика можеме да генерираме повеќе правилни реченици, почнувајќи од почетниот симбол rec, како: „Автомобилите возат по улицата“ или „Автомобилите свиреа над улицата“. Како би ја генерирале првата реченица. Почнуваме од почетниот симбол и ја правиме замената за него, а потоа секој нетерминал го заменуваме со некое од дадените правила:

rec→ im1 kornast pred im2→ Автомобилите kornast pred im2→ Автомобилите bosnast pred im2 → Автомобилите возат pred im2 → Автомобилите возат по im2 → Автомобилите возат по улицата.

Гледаме дека идејата на граматиката е да генерира реченици од јазикот при што ќе почне од почетниот нетерминал и сите нетерминали ќе ги замени со букви од азбуката. Секое правило има за задача да замени барем еден нетерминал.

Дефиниција 2. Јазикот генериран од граматиката G = (V, Σ, S, R) е јазкот од сите зборови од Σ* кои се добиваат со конечен број на чекори следејќи ја граматиката.

Ако правилото од R е такво што од левата страна имаме само еден нетерминал, т.е. е од облик $A \rightarrow w$, тогаш тоа се нарекува безконтекстно правило, а во секој друг случај правилото е контекстно правило. Ако граматиката се состои само од безконтекстни правила, се нарекува контекстно слободна граматика.

Јасно е дека првата форма на замена на реченица, именка, па заменка, па предлог и на крај именка, е можна форма на реченица од македонскиот јазик, но не секоја именка може да се замени на местото на првата или на втората именка. Ако наместо „автомобилите“ ставиме „автомобилот“, или, пак, некоја друга именка, едноставно реченицата ќе нема смисла. Тоа значи дека контекстот при замената е битен, па јазикот може да се опише само со контекстни правила. Програмските јазици сакаме да ги ослободиме од притисокот на контекстот и за нив бараме контекстот при замената во правилата да не биде битен.

3. ПРИМЕНА НА КОНТЕКСТНО-СЛОБОДНИТЕ ГРАМАТИКИ

Имаме голема среќа што се измислени контекстно-слободните граматики, бидејќи без нив програмирањето би било многу покомплицирано. Всушност, ќе требало да се навикнеме на нов начин на запишување на аритметичките изрази, кои како што биле усвоени отсекогаш, се препознатливи со ваков тип на граматики. Во овој дел ќе ја дадеме идејата за изградба на граматика за препознавање на математички изрази, што е и основата на првите сметачи кои можеле единствено да ги прават основните математички операции. Граматиката која се користи при ова се базира на познатите дефиниции за аритметичка или логичка формула. Ќе дадеме една парафраза на таква дефиниција, на која се базира наједноставната граматика.

Дефиниција за аритмерички израз:

1. Секој природен број е аритметички израз;
2. Ако a и b се аритметички изрази, тогаш и $a+b$, $a*b$ и (a) се аритметички изрази;
3. Секоја аритметички израз се добива со конечна примена на чекорите 1 и 2.

Оттука, ако еден аритметички израз се обележи со нетерминалот E, тогаш од вториот чекор E+E, E*E и (E) се аритметички изрази, па тоа ни ја дава едната релација во граматиката:

E:E+E|E*E|(E)

Останува да се дефинира што е природен број. Тој е составен од низа од цифри, па за него можеме да дадеме рекурзивна дефиниција на сличен начин: Секоја цифра е пророден број и ако на природен број на крајот му се долепи природен број, пак ќе се добие природен број. Со користење на нетерминалот B за бележење на број, ова формално може да се запише:

B:0|1|2|3|4|5|6|7|8|9|BB

На крај, двете работи се сврзуваат со

E:B

Сите правила на замена во оваа граматика која ќе ја наречеме Граматика 1 се:

E:E+E|E*E|(E)|B

B:0|1|2|3|4|5|6|7|8|9|BB

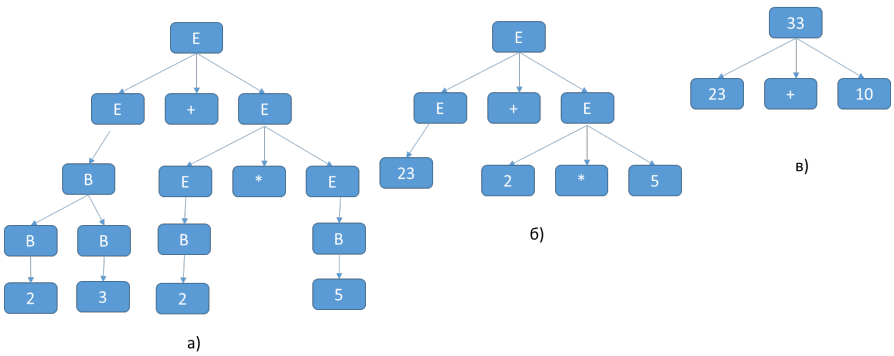
Да видиме како со оваа граматика може да се генерира изразот: 23+2*5.

$$E \rightarrow E+E \rightarrow B+E \rightarrow BB+E \rightarrow 2B+E \rightarrow 23+E \rightarrow 23+E*E \rightarrow 23+B*E \rightarrow 23+2*E \rightarrow 23+2*B \rightarrow 23+2*5.$$

Се покажува дека еден збор припаѓа во јазикот генериран со контекстно-слободна граматика G ако и само ако може да се добие така што секогаш ќе се заменува најлевиот нетерминал. Тоа значи дека сè што може да се генерира така што кој било нетерминал ќе се замени во даден момент, може да се генерира и со замена на најлевиот нетерминал во секој чекор. Граматиката дадена погоре генерира само добро напишан израз, па може да се искористи да препознава таква формула. Но, како што ќе видиме во наредниот дел, таа не води сметка дека множењето има предност пред собирањето, што не ја прави доволно добра, па ќе дадеме начин како да се преуреди за да ја има и оваа функционалност.

4. ПАРСИРАЧКО ДРВО

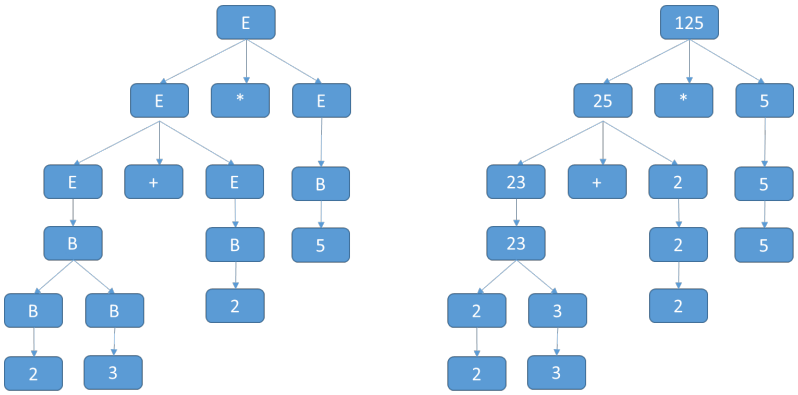
Во овој дел ќе зборите за начинот на кој се разбира текстот од страна на преведувачот. Таа техника се нарекува парсирање. При оваа постапка се прави така наречено парсирачко дрво, што е кореново дрво во кое листовите се карактерите во изразот, додека внатрешните јазли ги преставуваат деловите од формулата, а некои од нив ги содржат резултатите од операциите во формулата. Такво дрво е дадено на Слика 2. Дрвото под а) го претставува почетното парсирачко дрво, според парсирањето дадено во претходниот дел. Второто дрво ги покажува замените на броевите на местата на нетерминалите B, додека последното дрво ја дава целосната евалуација на изразот, која го дава точниот резултат 33.



Слика 2. Парсирачко дрво и негова евалуација.

За истиот израз 23+2*5 може да се направи и друго парсирачко дрво, ако во првиот чекор наместо замената E+E ја направиме замената E*E, кое е дадено на Слика 3 а). Изразот добро ќе се испарсира, односно ќе се добие потполно истиот израз, но пресметката на вредноста ќе биде различна, како што е дадено на Слика 3 б), каде што евалуираниот израз има вредност 125.

Според дадената граматика и двата начини на евалуација се добри, затоа што и двата го генерираат истиот израз како низа од стрингови, но имаат различно парсирачко дрво. Ваквите граматики кои за барем еден стринг имаат различни парсирачки дрва се нарекуваат двосмислени граматики.



Слика 3. Парсирачко дрво ако првата замена на E е E*E.

Двосмисленоста е својство на граматиките, а не на јазиците. За некои јазици може да има повеќе граматики, од кои некои се двосмислени, а некои не, а за други едноставно може да не постои недвосмислена граматика.

5. РАЗРЕШУВАЊЕ НА НЕДВОСМИСЛЕНОСТА

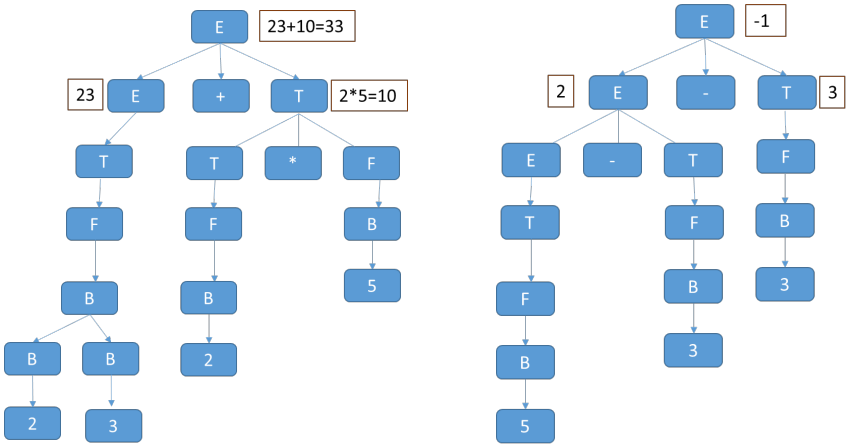
Дизајнирањето на недвосмислена граматика е „штосно“ и бара планирање од самиот почеток. Ќе покажеме како може ваква граматика да се дизајнира за нашиот јазик од аритметички изрази, така што секогаш кога ќе го парсираме изразот 23+3*5 да го добиваме дрвото од Слика 2, а не дрвото од Слика 3. За да го постигнеме тоа ќе воведеме нетрминали за секоја од операциите собирање, множење и заграда, кои имаат функција да „водат сметка“ за предноста на операциите.

Бидејќи во аритметички израз прво се пресметуваат изразите во заграда, ќе воведеме нетерминал F кој не може да се доведе ниту во израз со собирање, ниту во израз со множење, туку или до број или до израз во заграда. Слично ќе воведеме нетерминал T кој ќе може да се доведе само до израз со множење или загради, т.е. од него нема да може да се стигне до операцијата собирање (освен ако таа не е во заграда). Правилата во оваа граматика, која ќе ја наречеме Граматика 2 се:

E:E+T|T

T:T*F|F
F:(E)|B
B:BB|0|1|2|3|4|5|6|7|8|9

Парсирачкото дрво за изразот 23+5*2 е дадено на Слика 4.



Слика 4. Парсирачко дрво за граматиката со 4 нетерминали.

Уште повеќе, можеме да увидиме дека оваа граматика е и лево рекурзивна, што значи кога ќе имаме израз со повеќе исти операции, прво ја извршува првата операција. Таква граматика се користи за операции кои се лево асоцијативни, па неа можеме едноставно да ја проширime и со операциите одземање и делење, со додавање на правилата E: E – T и T: T/F. Со оваа граматика изразот 5 – 3 – 3 ќе се пресмета како (5 – 3) – 3, како што е правилно, Слика 4. Ако ги свртиме местата на нетерминалите, односно ако ставиме E: T – E и T: F/T, тогаш за овие операции ќе добиеме погрешно парсирачко дрво.

6. ПОСТАПКА НА ПАРСИРАЊЕ

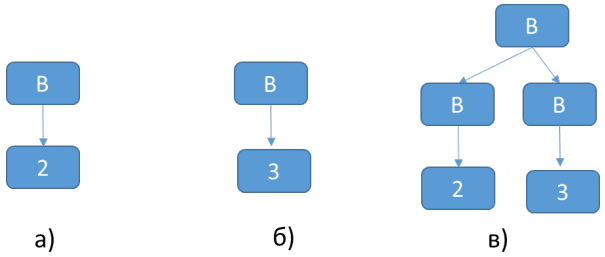
Начиот на работа на парсерот, делот од компајлерот кој е задолжен да го направи парсирачкото дрво, е таква што тој чита симбол по симбол од текстот кој му е даден и врз основа на наредниот симбол кој го чита треба да определи кое правило треба да го употреби. Граматиката 2 се користи за таканаречено „одоздола нагоре“ парсирање, во кое од дадениот израз се труди да го добие нетерминалот E. Всушност оваа постапка го гради парсирачкото дрво од листовите нагоре,

применувајќи ги правилата на граматиката во обратен редослед. Можеме да го гледаме како постапката на генерирање на изразот да ја прави во обратен редослед, од назад на напред, знаејќи во секој чекор кој е наредниот карактер кој го има во текстот. Ќе илустириаме како работи ова на нашиот пример 23+2*5.

На влез го имаме 23+2*5 и сакаме да провериме дали ова може да се редуцира до E. Редуцирањето го памтиме во стек и работиме така што секогаш кога нешто во стекот може да се редуцира, го редуцираме, но само во ситуација кога по тоа редуцирање смееме да го имаме наредниот симбол. Притоа, сакаме што е можно повеќе симболи од стекот да се редуцираат во еден симбол. Во првата фаза гледаме по кои нетерминали кои симболи се дозволени. Од граматиката после E е дозволено да има + и). По T е дозволено да има *, но бидејќи го имаме правилото E:T, по T може да се јави и +. По F може да се јави кој било од знаците +, * и) , а по B може да се најде кој било од овие знаци, но и која било цифра.

Редуцирањето претпоставуваме дека го правиме во стекот. Сметаме дека првиот карактер од влезот го префрламе во стекот (податочна структура која има ограничен пристап - елементот кој е ставен последен, излегува прв) и потоа правиме редуцирања. Целата постапка е прикажана во Табела 1.

Во првиот чекор на влез имаме 23+2*5 и првиот карактер кој го читаме е 2. Овој карактер се става во стекот и се брише од влезот. Имајќи предвид дека 2 не е нетерминал, веднаш го редуцираме во нетерминал. Единствено правило кое го вклучува симболот 2 е B:2, па го користиме во обратна насока и сега во стекот имаме B. Со овој дел се прави дрвото на Слика 5 а). Имаме правило F:B, но не можеме да го искористиме затоа што нареден карактер е 3, а после F не може да има 3, па застануваме овде и во стекот го ставаме симболот 3. Веднаш 3 го редуцираме до B со правилото B:3, како што е прикажано во ред 3 и 4 во Табела 1. Со ова се прави ново дрво, дрвото на Слика 5 б). Во стекот сега имаме BB и користејќи го правилото B:BB како и тоа дека по B може да има +, BB во стекот го заменуваме со B. Затоа сега двете дрва што ги направивме ги спојуваме во едно дрво, дрвото на Слика 5 в).



Слика 5. Градење на парсирачко дрво „одоздола нагоре“.

Табела 1. Чекори при парсирање „одоздола нагоре“.

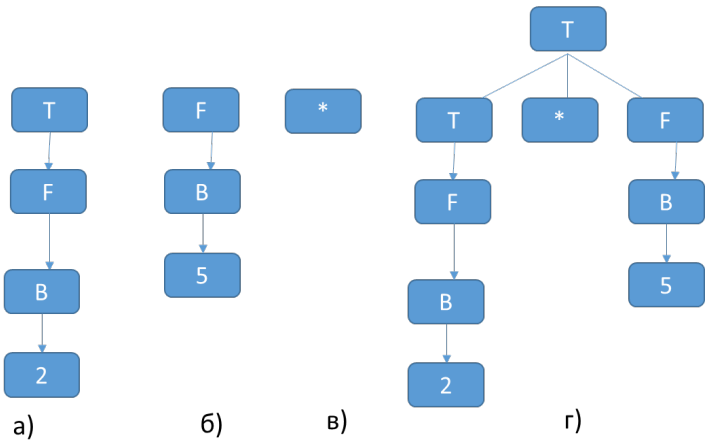
Чекор	Влез	Стек	Правило	Нареден симбол
0	23+2*5			
1	3+2*5	2	B:2	3
2	3+2*5	B	Нареден карактер	3
3	+2*5	B3	B:3	+
4	+2*5	BB	B:BB	+
5	+2*5	B	F:B	+
6	+2*5	F	T:F	+
7	+2*5	T	E:T	+
8	+2*5	E	Нареден карактер	+
9	2*5	E+	Нареден карактер	2
10	*5	E+2	B:2	*
11	*5	E+B	F:B	*
12	*5	E+F	T:F	*
13	*5	E+T	Нареден карактер	*
14	5	E+T*	Нареден карактер	5
15		E+T*5	B:5	λ
16		E+T*B	F:B	λ
17		E+T*F	T:T*F	λ
18		E+T	E:E+T	λ
19		E		λ

Сега B може да се редуцира во F, бидејќи по F е дозволен знакот +. Слично F може да се редуцира до T, а T до E. Со ова ја добиваме

најлевата гранка од дрвото на Слика 4. Бидејќи E не може понатаму да се редуцира, застануваме овде и во стекот го внесуваме наредниот симбол +, како што е даден во редот 9 во Табела 1.

E+ не може да се редуцира, бидејќи таков стринг немаме од десната страна. Затоа го внесуваме наредниот симбол 2 и како и претходно, него го редуцираме до B, B до F, а F до T и тука застануваме, затоа што ако T го редуцираме до E, тогаш по E не можеме да го имаме знакот *. Со ова се добива дрвото на Слика 6 а). Имено, во овој момент во стекот имаме E+T што може да се редуцира во E, но повторно, бидејќи по E не е дозволено *, тоа не го правиме.

После ова во стекот се внесува * и бидејќи повторно не е можна редукција го внесуваме и 5. Слично како претходно, 5 се редуцира прво до B, а потоа до F и овде застануваме, Слика 6 б). Имено, F може да се редуцира до T, но забележуваме дека во стекот имаме поголем редуцибилен израз, а ние сакаме да направиме редукција на најголемиот можен израз, па го редуцираме T*F во T, ред 18 во Табела 1. Со оваа редукција во парсирачкото дрво се спојуваат дрвата од Слика 6 а), б) и в) во едно дрво, дрвото г).



Слика 6. Градење на парсирачко дрво „од долу нагоре“.

Во последниот чекор на стекот имаме само E+T што може да се редуцира во E, при што се добива саканото парсирачко дрво од Слика 4. Бидејќи веќе немаме ништо на влез, а во стекот добивме само E, можеме да заклучиме дека на влез сме имале правилно запишан израз.

Ако не го добиеме ова што го добивме, тоа значи дека изразот што го имаме не е добар аритметички израз. Убаво дизајниран компајлер ќе ја открие и ќе ја репортира грешката и би застанал со работа. Ако тоа не е разрешено со компајлерот, тогаш програмата ќе се блокира, нема да извади резултат, може бескрајно да работи и слично.

7. ЗАКЛУЧОК

Формалните граматики несомнено претставуваат револуционен изум кој довел до развој на компјутерските науки, посебно развојот на програмските јазици како средство за комуникација помеѓу човекот и машината. Овој труд ја прикажува логичката страна, односно математичкиот концепт, за конструкција на една таква граматика. Неговата основна цел е накратко да ја објасни идејата зад оваа специфична математичка теорија, која како една од најновите математички теории е поразлична од другите и многу помалку застапена во математичкото образование. Разработката е направена преку илустрација на градењето на една од најосновните граматики, граматиката за формирање на правилни математички изрази, која е подмножество на секоја граматика за програмски јазик. Во трудот се потрудивме да ја објаснеме идејата за воведување на секое од граматичките правила, како и неговата улога во јазикот од една страна од гледна точка на човекот, кој сака да ги запишува изразите на начин на кој е навикнат, а од друга страна од аспект на компјутерот кој треба тој запис правилно да го разбере.

ЛИТЕРАТУРА

[1] N. Chomsky, *Three models for the description of language*, IRE Trans. Inf. Theory, 1956.

[2] N. Chomsky, *On certain formal properties of grammars*, Inf. Control 2, 1959.

[3] V. Kirandziska, M. Jovanov, M. Mihova, M. Gusev, *Lab assessments in undergraduate course in Compilers for students with no prior knowledge in assembly*, 2014 37th International Convention on

Information and Communication Technology, Electronics and Microelectronics (MIPRO), 738 – 743.

[4] P. Linz, *An Introduction to Formal Languages and Automata*, Jones and Bartlett Publishers, 2001.

[5] B. J. MacLennan, *In mathematical terms, this means the programming language is Turing-complete. Principles of Programming Languages*, Oxford University Press. P. 1., 1987.

[6] A. Meduna, *Formal Languages and Computation: Models and Their Applications*, CRC Press, p. 233, 2014.

[7] E. L. Post, *Recursively enumerable sets of positive integers and their decision problems*, Bull. Am. Math. Soc. 50, 1944.

[8] E. Rich, *Automata, Computability and Complexity, Theory and applications*, Pearson Education, Inc, 2008.

[9] W. Tecumseh Fitch and Angela D. Friederici, *Artificial grammar learning meets formal language theory: an overview*, Philos Trans R Soc Lond B Biol Sci., 367(1598): 1933 – 1955, 2012.

[10] A. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. Lond. Math. Soc. 42, 230 – 265, 1936.

1 Факултет за информатички науки и компјутерско инженерство, Универзитет Св. Кирил и Методиј, Скопје
ул. „Руѓер Бошковиќ“ 16, 1000 Скопје, Р. Северна Македонија
e-mail: marija.mihova@finki.ukim.edu.mk

Примен: 8.5.2020
Поправен: 30.7.2020
Одобрен: 1.8.2020
Објавен на интернет: 4.8.2020