КОНТЕКСТНО СЛОБОДНИ ГРАМАТИКИ И ПАРСИРАЊЕ

Скопје, 2002 година

Содржина:

1. Граматики и Pushdown автомати

1.1 Граматики, јазици генерирани од граматики

Да се замислиме како јазични процесори. Можеме да препознаеме точна реченица на македонски јазик кога би ја слушнале. “Рибата е во кокошарникот”, е во најмала рака синтаксно точна реченица, додека “Кокошарникот во во е рибата риба” воопшто не е. Без разлика како, ние успеваме да откриеме дали реченицата која сме ја чуле е составена според прифатените синтаксни правила на нашиот јазик. Ние играме улога на **јазични препознавачи** - направи кои прифаќаат точни зборовни конструкции. Конечните автомати се формализирани типови на јазични препознавачи.

Ние, исто така, можеме и да *произведеме* исправна реченица на македонски. Во тој случај ние претставуваме **јазични генератори**. Тоа е направа која во одреден момент започнува со креирање на јазична констукција која однапред не е наполно определена. Меѓутоа, машината се води според ограничено множество од правила. По некое време процесот запира со комплетиран збор (стринг) како излез. Јазикот кој е дефиниран од оваа машина е множество од сите зборови (реченици, стрингови) кои може да бидат произведени.

Воопшто не е лесно да произведеме јазичен генератор или јазичен препознавач за македонскиот или за кој друг било природен јазик. Од формална гледна точка, оваа задача може да е дури и невозможна. Поважна за нас е теоријата за генератори на формални, „вештачки” јазици. Така доаѓаме до поимот за „формална граматика” кој најопшто може да се интерпретира како алгоритам кој овозможува да се определи јазик над дадена азбука. Конкретно зборуваме за граматика која означува можност за даден збор од јазикот да постои таков начин на работа на алгоритмот што тој на крај на неговото работење го генерира зборот. Оваа граматика ја нарекуваме *генеративна граматика*. Значи:

Дефиниција 1.1.1

Формална граматика или само *граматика* G е подредена четворка $G = (V, \Sigma, S, R)$, каде што

- $V \cap \Sigma = \varnothing$ и V и Σ се конечни множества;
- $S \in V$ е почетен нетерминал;
- R е конечно множество зборови од облик $v \rightarrow w$, каде што $v, w \in (V \cup \Sigma)^*$, $\rightarrow$ е симбол што не е во $V \cup \Sigma$.

За Σ велиме дека е *основна азбука* или азбука од терминални симболи, а за V дека е *помошна азбука* на граматиката G, или азбука од нетерминални симболи, S е *почетен симбол* од G, R е множество од правила за генерирање на

зборови и се вика *шема* или *програма* на граматиката *G*. Секој член од *R* се вика *правило, v* се вика *лева страна*, а *w десна страна* на правилото $v \rightarrow w$. Множеството $V \cup \Sigma$ се вика *потполна азбука* на граматиката *G*.

Нека $r = v \rightarrow w$ е правило и нека $\alpha * v * \beta$ е настапување на *v* во зборот $s = \alpha v \beta$ од азбуката $V \cup \Sigma$. Во тој случај за зборот $t = \alpha * w * \beta$ велиме дека е *добиен од s со примена на правилото r*.

Ако *t* се добива со примена на едно правило од *G,* пишуваме $s \Rightarrow_G t$ или само $s \Rightarrow t$.

За низата зборови s_0 , s_1 , s_2 ,. . . . s_n велиме дека е *извод на* s_n *од* s_0 во граматиката *G* ако за секој $0 < i < n+1$ важи $s_{i-1} \Rightarrow s_i$. Бројот *n* се вика *должина* на изводот на s_n од s_0 во *G,* и притоа пишуваме $s_0 \Rightarrow_G^* s_n$ или само $s_0 \Rightarrow^* s_n$. За изводот s_0 , s_1 , s_2 ,. . . . s_n велиме дека е *потполн* ако $s_0 = S$ е почетниот симбол на *G*, а $s_n \in \Sigma^*$.

За еден извод во *G* велиме дека е *лев извод* ако во секој чекор замена се изведува на првиот од лево нетерминален симбол во зборот.

Формално, ќе пишуваме $s_0 \Rightarrow^{*L} s_n$ ако е даден лев извод на s_n од s_0 во *G*.

Дефиниција 1.1.2

Јазикот L(G) генериран од граматиката G се состои од сите зборови од Σ^* кои имаат потполн извод во G, т.е.

$L(G) = \{ w \in \Sigma^* : S \Rightarrow^* w \}$.

Ако празниот збор е елемент од јазикот, ќе го означуваме со *e* или λ.

Дефиниција 1.1.3

За граматиката G се вели дека е *контекстно осетлива* или *контекстно зависна* ако секое нејзино правило е од облик $\alpha A \beta \rightarrow \alpha w \beta$, каде *што* $\alpha, \beta \in (V \cup \Sigma)^*$, $A \in V$, $w \in (V \cup \Sigma)^*$.

Во горната дефиниција се гледа дека замената на А зависи од нејзиниот контекст (околината, соседните букви) , па оттука доаѓа и името на овој вид граматики.

Ако во правилото $\alpha A \beta \rightarrow \alpha w \beta$, α, $\beta = e$ за него велиме дека е *безконтекстно* правило.

Дефиниција 1.1.4

Ако сите правила во граматиката се безконтекстни тогаш станува збор за ***контекстно слободна граматика*** *(КС граматика).*

Пример 1.1.1

Разгледуваме граматика $G = (V, \Sigma, S, R)$ каде $V=\{S\}$, $\Sigma=\{a, b\}$, $R = \{ S \rightarrow aSb, S \rightarrow e\}$. Можен извод е:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Овде, во првите два чекора е применето правилото $S \rightarrow aSb$, а во третиот чекор $S \rightarrow e$. Всушност, лесно се гледа дека $L(G) = \{ a^n b^n : n \geq 0 \}$. |||

Пример 1.1.2

Нека G е граматиката (V, Σ, S, R) каде
$V=\{S, P, I, G, F\}$,
$\Sigma=\{dete, golemo, belo, sirenje, jade\}$,
$R = \{ F \rightarrow I ;$
$F \rightarrow PF;$
$S \rightarrow FGF;$
$P \rightarrow golemo;$
$P \rightarrow belo;$
$I \rightarrow sirenje;$
$I \rightarrow dete;$
$G \rightarrow jade \}$.

Тука се обидуваме да направиме граматика за дел од македонскиот јазик. S означува реченица, P значи придавка, I - именка, G - глагол и F значи фраза. Еве неколку производи од граматиката:

dete jade sirenje
golemo dete jade belo sirenje
belo dete jade golemo belo sirenje

но за жал зборови во јазикот се и:

golemo belo golemo dete jade golemo belo belo dete
golemo sirenje jade golemo belo sirenje. |||

Пример 1.1.3

Компјутерската програма мора да задоволува одредени строги правила за да биде синтаксно точна, а со тоа е и во форма да може да се интерпретира механички. За среќа, синтаксата на повеќето програмски јазици може да се покрие со контестно слободна граматика Понатаму ќе видиме дека својството на контекстно слободност многу помага кога треба да се парсира. Граматиката во овој пример генерира фрагмент од повеќе познати програмски јазици. Всушност таа ни дава синтаксно коректни изрази со променливите *x1* и *x2* и операциите + и *.

Па нека

$G = (V, \Sigma, E, R)$

каде

$V=\{T, F, E\},$

$\Sigma =\{x, 1, 2, +, *, (,) \},$

$R = \{ E \rightarrow E + T;$

$E \rightarrow T;$

$T \rightarrow T * F;$

$T \rightarrow F;$

$F \rightarrow (E);$

$F \rightarrow x1;$

$F \rightarrow x2 \}.$

Буквите *E, T* и *F* се однесуваат на *expression, term* и *factor*.
Граматикава го генерира *(x1 * x2 + x1) * x2* со следниов извод:

$E \Rightarrow T$ ...R2

$\Rightarrow T * F$ ...R3

$\Rightarrow T * x2$ ...R7

$\Rightarrow F * x2$ ...R4

$\Rightarrow (E) * x2$ ...R5

$\Rightarrow (E + T) * x2$ ...R1

$\Rightarrow (E + F) * x2$ ...R4

$\Rightarrow (E + x1) * x2$ ...R6

$\Rightarrow (T + x1) * x2$ ...R2

$\Rightarrow (T * F + x1) * x2$ ...R3

$\Rightarrow (F* F + x1) * x2$ ...R4

$\Rightarrow (F * x2 + x1) * x2$ ...R7

$\Rightarrow (x1* x2 + x1) * x2$ ...R6

|||

На крај ќе наведеме едно својство:

Lemma 1.1.1 За секоја КС граматика G и секој збор $w \in \Sigma^*$, $S \Rightarrow^* w$ акко $S \Rightarrow^{*L} w$.

1.2 Pushdown автомати

Не секој контестно слободен јазик може да биде препознаен од страна на конечен автомат. Тоа е јасно и следи од фактот дека регуларните јазици (јазиците кои се опишуваат со регуларни изрази) се поткласа од класата на контекстно слободни јазици.

Во обид да се најде „автомат“ кој би ги препознавал КС - јазиците се тргнува од конечните автомати. Најпростиот пример е јазикот $\{ a^n b^n : a, b \in \Sigma \}$, за кој конечниот автомат не може да запамти колку пати изел (прочитал) a, за да потоа допушти точно толку b-овци. Прва идеја која ни паѓа на памет е своевиден бројач. Меѓутоа ако тргнеме од јазикот $\{ ww^R : w \in \Sigma^* \}$ кој исто така е КС- јазик, ќе забележиме дека всушност треба да најдеме начин нашиот автомат да памти се што е исчитано до одреден момент и тоа потоа да може да го спореди со остатокот од зборот во обратен редослед. Значи, ќе дозволиме меморирање на збор, кој ќе може во даден момент да се чита од крајот, буква по буква. Нормално, воведуваме склад (магацин, stack, pushdown store) како дополнителен елемент. Вака замисленият автомат формално го дефинираме:

Дефиниција 1.2.1

Pushdown автомат е подредена шесторка $M = (K, \Sigma, \Gamma, \Delta, s, F)$, каде што:

- K е конечно множество состојби;
- Σ е азбука од влезни симболи;
- Γ е азбука од симболи на складот;
- $s \in K$ е почетна состојба;
- $F \subseteq K$ е множество од завршни состојби;
- Δ е релација на премин, т.е. конечно подмножество од $(K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$.

Ако $((p, u, \beta), (q, \gamma)) \in \Delta$, тогаш секогаш кога M е во состојба p, го гледа зборот u, преминува во состојба q, оди едно место на десно во складот и го заменува β со γ.

Парот $((p, u, \beta), (q, \gamma))$ се вика премин на M.

Вака дефинираните автомати овозможуваат симбол да се додаде или да се извади од складот. Имено, преминот $((p, u, e), (q, a))$ додава симбол a, додека преминот $((p, u, a), (q, e))$ го вади симболот a од складот.

Pushdown автоматите можат само да препознаваат дали некој збор припаѓа на даден јазик и почетниот дел од зборот (веќе прочитаниот) не влијае на понатамошната работа на автоматот. Така, *конфигурација на pushdown автомат* е елемент од $K \times \Sigma^* \times \Gamma^*$; притоа, првиот елемент е состојбата во која се наоѓа автоматот во даденот момент, вториот е зборот кој допрва треба да го прочита, а третиот е содржината на складот.

Дефинираме релација $|--_M$ во $K \times \Sigma^* \times \Gamma^*$ на следниов начин:

$$(p, ux, \beta\alpha) |--_M ((q, x, \gamma\alpha)$$

за секој $x \in \Sigma^*$, $\alpha \in \Gamma^*$, $((p, u, \beta), (q, \gamma)) \in \Delta$.

Рефлексивното и транзитивно затворање на $|--_M$ го означуваме со $|=_M$.

Велиме дека M го препознава зборот $w \in \Sigma^*$ако и само ако $(s, w, e) |= (p, e, e)$, каде што $p \in F$.

Значи, јазикот $L(M)$ препознаен од автоматот M е дефиниран со: $L(M) = \{w \in \Sigma^* : (s, w, e) |= (p, e, e), p \in F\}$.

Пример 1.2.1

Автоматот кој го препознава јазикот $L=\{ wcw^R : w \in \{a, b\}^*\}$ е $M = (K, \Sigma, \Gamma, \Delta, s, F)$ каде што:

- $K = \{s, f\}$,
- $\Sigma = \{a, b, c\}$,
- $\Gamma = \{a, b\}$,
- $F = \{f\}$,
- $\Delta = \{ ((s, a, e), (s, a)); ((s, b, e), (s, b)); ((s, c, e), (f, e)); ((f, a, a), (f, e); ((f, b, b), (f, e)) \}$.

Како работи автоматов?

На почетокот автоматот ги јаде сите a и b, притоа ставајќи ги на складот редоследно како што доаѓаат. Тој е во почетна состојба s, а потоа кога во оваа состојба ќе прочита (изеде) c се префрла во состојба f и ниту запишува нешто на складот, ниту пак вади нешто од него. Во оваа состојба јаде еден по еден симбол, и секој го споредува со оној на складот и ако се совпаѓаат го вади од складот. Ако зборот припаѓа на јазикот, ќе може да бидат прочитани сите букви и на крај да остане празен склад. Во спротивно, т.е. ако зборот не припаѓа на јазикот може да се случи или да не се премине во состојба f или автоматот да премине во таа состојба меѓутоа да не се прочита целиот збор или да не се испразни целиот склад. На пример: $(s, abbcbba, e)|-- (s, bbcbba, a) |-- (s, bcbba, ba) |-- (s, cbba, bba) |-- (f, bba, bba) |-- (f, ba, ba) |-- (f, a, a) |-- (f, e, e)$. |||

Пример 1.2.2

Автоматот кој го препознава јазикот $L=\{ ww^R : w \in \{a, b\}^*\}$ е $M = (K, \Sigma, \Gamma, \Delta, s, F)$ каде што:

- $K = \{s, f\}$,
- $\Sigma = \{a, b\}$,
- $\Gamma = \{a, b\}$,
- $F = \{f\}$,
- $\Delta = \{ ((s, a, e), (s, a)); ((s, b, e), (s, b)); ((s, e, e), (f, e)); ((f, a, a), (f, e); ((f, b, b), (f, e)) \}$.

Како што може да се забележи, автоматот е речиси еднаков на претходниот. Промената е само во азбуката од влезни симболи и во едно од правилата. Старото правило *((s, c, e), (f, e))* е заменето со *((s, e, e), (f, e))*. Ова правило (премин) може да се примени во секој момент и кога би го примениле точно во моментот кога завршува *w* тогаш таквата низа на премини ќе го препознае зборот. Меѓутоа секоја друга низа на премини не доведува до препознавање на зборот. Сепак, да автоматот препознава некој збор од даден јазик значи да постои низа од премини која од почетната состојба и празен склад, по читањето на зборот, ќе ја доведат машината во завршна состојба и празен склад. Токму заради преминот *((s, e, e), (f, e))* се јавува недетерминизам и овој автомат е недетерминистички. |||

1.3 Врска помеѓу Контекстно слободни граматики и Pushdown автомати

Врската помеѓу контекстно слободните граматики и pushdown автоматите ни ја дава следнава теорема:

Теорема 1.3.1:
Класта јазици препознаена од pushdown автомати е точно класата контекстно слободни јазици.

Теоремата нема да ја докажуваме. Ќе го разгледаме само начинот на конструкција на pushdown автомат, ако ни е дадена контекстно слободна граматика.

Нека е дадена контекстно слободната граматика $G = (V, \Sigma, S, R)$. Конструираме pushdown автомат M, така што $L(M) = L(G)$. Автоматот ќе има само две состојби p и q и ќе останува преманентно во состојбата q по првиот чекор. Така,

$M=(\{p, q\}, \Sigma, V \cup \Sigma, \Delta, p, \{q\})$,

каде што Δ ги содржи следните релации:

- $((p, e, e), (q, S))$,
- $((q, e, A), (q, x))$ за секое правило од облик $A \rightarrow x$ во R,
- $((q, a, a), (q, e))$ за секој $a \in \Sigma$.

Овој автомат започнува со сместување на S во складот кој што е празен и преминува во состојба q. Во секој нареден чекор, или го заменува најгорниот симбол A од складот со десната страна од правилото $A \rightarrow x$ од R или го вади најгорниот симбол од складот доколку тој е терминален симбол и е еднаков со следниот влезен симбол.

1.4 Својства на Контекстно Слободните Јазици

Својствата на контекстно слободните јазици ќе ги наведеме преку неколку теореми, а ќе се задржиме на алгоритамските својства. На почетокот ќе дефинираме графичко претставување на извод кај контекстно слободните јазици кој ќе го нарекуваме *парсирачко дрво.* Точките (крукчињата) се викаат *темиња*, најгорното теме *корен,* додека најдолните темиња *листови* на парсирачкото дрво. Со конкатенација на ознаките на листовите од лево на десно, се добива генерираниот збор, кој се нарекува *резултат* на парсирачкото дрво.

Дефиниција 1.4.1

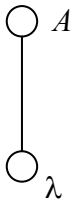
За произволна контекстно слободна граматика $G = (V, \Sigma, S, R)$, парсирачко дрво, неговиот корен листови и производ се дефинираат како:

1.

Слика 1.4.1 Парсирачко дрво со резултат *A*

е парсирачко дрво за секој $A \in (V \cup \Sigma)$. Единствено теме на ова парсирачко дрво е неговиот корен и неговиот лист. Резултат е зборот A.

2. Ако $A \rightarrow \lambda$ е правило од R, тогаш



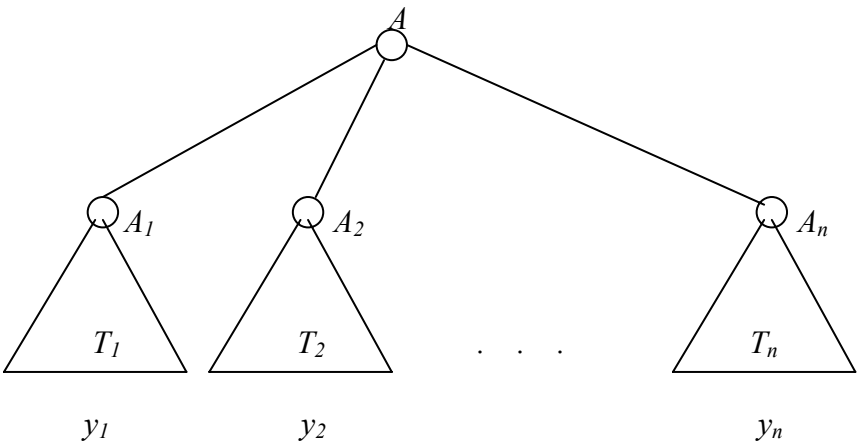
Слика 1.4.2 Парсирачко дрво со резултат празен збор

е парсирачко дрво со корен означен со A, листот е е означен со λ, а резултат е празниот збор λ.

3. Ако

Слика 1.4.3 Повеќе парсирачки дрва

се парсирачки дрва со корени означени со $A_1, \ldots, A_n$, и со резултати $y_1, \ldots, y_n$, соодветно, а притоа правило во R е $A \rightarrow A_1A_2...A_n$, тогаш



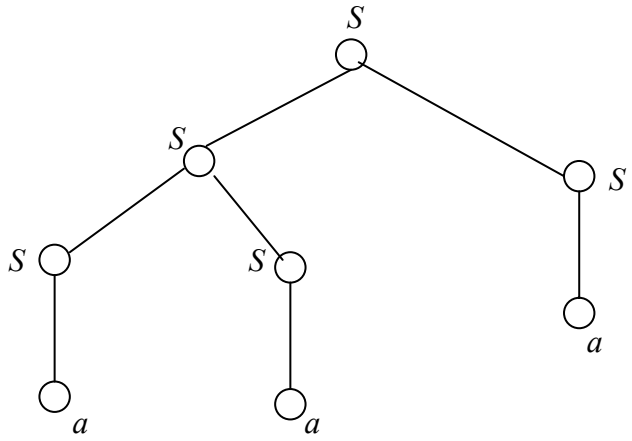
Слика 1.4.4 Резултантно парсирачко дрво

е парсирачко дрво со нов корен означен со A, листови се листовите на $T_1, T_2, \ldots, T_n$, а резултат е $y_1 y_2 \ldots y_n$.

4. Ништо друго не е парсирачко дрво.

Пример 1.4.1

За $G=(\{S\}, \{a\}, S, \{S \rightarrow SS, S \rightarrow a\})$ изводот $S \Rightarrow SS \Rightarrow Sa \Rightarrow SSa \Rightarrow aSa \Rightarrow aaa$ може да се претстави како на сликата 1.4.5 :



Слика 1.4.5 Парсирачко дрво за *aaa* од граматика *G*.

|||

Пат во парсирачкото дрво е низа од различни темиња, секое поврзано со претходното со отсечка. Притоа, првото теме е коренот, а последното, лист на парсирачкото дрво. *Должина на патот* е бројот на отсечки, којшто е за еден помалку од бројот на темиња. *Висина* на парсирачкото дрво е должината на најдолгиот пат во него.

Теорема 1.4.1

Контекстно слободните јазици се затворени во однос на унија, конкатенација и Клиниева ѕвезда.

Теорема 1.4.2

(Теорема за пумпање) Нека G е контекстно слободна граматика. Тогаш постои број K што зависи од G, таков што секој збор w од L(G) со должина поголема од K може да се запише во облик w=uvxyz на таков начин што или v или y се непразни зборови и за секој $n \geq 0$, uv^nxy^nz *е исто така во L(G).*

Теорема 1.4.3

Контекстно слободните јазици не се затворени во однос на операциите пресек и комплемент.

1.4.1 Алгоритамски својства

Во проучувањето на контекстно слободните јазици и граматики и нивните својства сеуште не дадовме одговор на некои прашања. Едно од тие прашања е: Дали даден збор припаѓа на даден јазик? Или да парафразираме, како ако ни е дадена граматика и ни е даден збор ние ќе можеме да кажеме дека зборот е во јазикот генериран од граматиката?

Осврнувајќи се на начинот на конструкција на pushdown автомат, од дадена граматика можеме да дадеме ваков одговор на горното прашање: За дадената граматика ќе конструираме pushdown автомат и ќе го пуштиме да работи за дадениот збор. За жал, дадениот пристап не функционира, барем не директно. Pushdown автоматот може да работи долго време без воопшто да не изеде ниту една буква од зборот. Така, не можеме да бидеме сигурни дека сме дозволиле доволно долго да работи автоматот пред да донесеме заклучок дека зборот не е прифатен.

Наместо ваквиот пристап, се прибегнува кон решавање на проблемот со анализа на парсирачки дрва.

Теорема 1.4.4

Постојат алгоритми за одговарање на следниве прашања за контекстно слободните граматики:

За дадена граматика G и збор w, дали $w \in L(G)$?

Дали $L(G) = \varnothing$?

Доказ: Основна идеја за да одредиме дали одреден збор w припаѓа во јазик генериран од дадена граматика G, е да ги провериме сите можни изводи до одредена должина или евентуално сите можни парсирачки дрва до одредена голем, во потрага на w. Ако тој е наден тогаш јасно $w \in L(G)$, а ако таков не се најде, ќе може да се заклучи дека $w \notin L(G)$. Важно е само да се знае кои парсирачки дрва е доволно да се проверат.

Сега, ако секое правило на G е од вид $A \rightarrow u$ каде u е терминал или $|u| \geq 2$, тогаш проблемот во многу се поедноставува. Во тој случај, при секој извод (деривација) зборот ќе станува подолг, освен во случаите кога ќе се применува правило со кое нетерминал се заменува со само еден терминал. Затоа доволно е да се проверат изводите не подолги од $2|w|-1$ чекори за да знаеме дали $w \in L(G)$. Или, преку парсирачки дрва: било кое дрво со висина h мора да има разултат со должина барем h. Па сега, ако сакаме да одредиме дали даден збор w е во јазикот $L(G)$, можеме едноставно да ги провериме сите парсирачки дрва со висина најмногу $|w|$. Бројот на овие дрва може да се ограничи со проверка на G.

Така, првиот дел може да се одговори ако успееме да ја претвориме било која граматика во еквивалентна граматика, без правила од вид $A \rightarrow u$, каде u е празниот збор или нетерминал. Ова практично не е возможно, бидејќи ако празниот збор е во јазикот тој може да биде генериран единствено со правило од вид $A \rightarrow e$ за некој нетерминал A. Но, ќе се обидеме да дојдеме до облик доволно добар за нашите потреби.

Прво, ги елиминираме сите правила од вид $A \rightarrow u$, *освен* правилото $S \rightarrow e$ кое ќе треба да остане ако празниот збор е во јазикот. Постапуваме вака:

Нека $G=(V, \Sigma, R, S)$. За секое правило од вид $A \rightarrow e$ кога постои и правило $B \rightarrow uAv$ $(u,v \in (V \cup \Sigma)^*)$ додаваме $B \rightarrow uv$ на множеството од правила. Ова го повторуваме се дури може. Процесов завршува, бидејќи десната страна од секое ново правило е подзбор од десната страна на некое веќе постоечко правило. Освен тоа, јазикот останува ист од причина што секое ново правило едноставно премостува едно изведување, т.е. е еквивалентно на две едноподруго изведени веќе постоечки правила. Потоа, се елиминираат сите правила од вид $A \rightarrow e$, освен правилото $S \rightarrow e$, ако тоа воопшто постои, од резултантните правила. И ова не го менува јазикот, оти употребата на правило $A \rightarrow e$ ќе имплицира дека претходно за да се добие A е употребено правило $B \rightarrow uAv$, па правилото $B \rightarrow uv$ може да се користи наместо претходните две. Така, дури и ако правилото $S \rightarrow e$ остане, потребно е да се користи само во изведувањето (изводот) $S \Rightarrow e$ и никаде на друго место каде што би се појавило S. Па добиената граматика G_1 ги има следниве својства:

$L(G_1) = L(G)$

Ако $e \in L(G_1)$ тогаш $S \rightarrow e$ е правило во G_1.

Секој збор од $L(G_1)$ освен e има извод и соодветно парсирачко дрво, во кое не е употребено ниедно правило од вид $A \rightarrow e$.

Останува да се елиминираат правилата од вид $A \rightarrow B$. Да забележиме дека за дадени два нетерминала A и B можеме да одредиме дали $A \Rightarrow^*_{G1} B$; бидејќи

ако е така, тогаш постои извод на B од A со најмногу $|V|$ чекори. Секогаш кога $B \rightarrow u$ е правило од G_1 и $A \Rightarrow^*_{G1} B$ се додава правило $A \rightarrow u$; и повторно, овој процес не го менува генерираниот јазик. На крај се елиминираат сите правила $A \rightarrow B$. Тие не се потребни земајќи го во предвид фактот дека секојпат кога секвенцата $A \Rightarrow^* B \Rightarrow u$ се употребува во изводот при што $u \notin V$ правилото $A \rightarrow u$ може да се искористи за замена. Со тоа е завршен доказот на првиот дел.

За одговор на второто прашање само една идеја. Треба да се забележи дека ако јазикот не е празен тогаш мора да постои парсирачко дрво со резултат кој е терминален збор, такво што тоа има висина најмногу $|V|$. Бидејќи сите такви парсирачки дрва може да се проверат, ќе може да се одговори дали е пронајден збор како резултат т.е. дали јазикот е празен или не. □

На крајот, една забелешка. Бидејќи секој pushdown автомат може да биде заменет со еквивалентна контекстно слободна граматика, овие алгоритми може да се користат за одговарање на истите прашања за pushdown автоматите.

2. Детерминизам и парсирање

Во доказот на теоремата 1.4.4, ние покажавме метод за одредување дали одреден збор е генериран од дадена контекстно слободна граматика. Накратко, методот беше да се измени оваа граматика за да се елиминираат некои видови на „непродуктивни“ правила и тогаш да се проверат сите парсирачки дрва до одредено ниво, во зависност од должината на зборот. Овој метод не е направен за да биде практичен метод, тој има само теориска важност. Како во практиката ќе го одговориме оваа прашање?

Овој проблем бил тема на поголем број истражувања и има повеќе можни приоди. Оние што овде се разгледуваат се вкоренети во идејата за pushdown автоматите. Значи, овде ја користиме еквивалентноста на pushdown автоматите и контекстно слободните граматики. Јасно е дека pushdown автоматот добиен по алгоритам од одредена контектно слободна граматика не може веднаш практично да се користи за одредување дали некој збор е од граматиката, бидејќи може да се случи автоматот да работи бескрајно без да изеде ниту дел од влезот. Исто така има друг проблем, самиот начин на кој pushdown автоматот беше дефиниран е недетерминистички. Бидејќи секој јазик генериран од контекстно слободна граматика може да биде препознаен од соодветен pushdown автомат (конструиран по алгоритмот даден во прилог на теорема 1.3.1), се поставува прашање дали секогаш може да направиме pushdown автоматите да работат детерминистички?

Наша прва цел во овој дел е да го проучиме прашањето на детерминистичките pushdown автомати. Ќе се види дека постојат контекстно слободни јазици кои *не може* да бидат прифатени од детерминистички pushdown автомати. Ова е малку разочарувачки затоа што сугерира дека конверзијата на граматика во автомат може да не биде база за било кој практичен метод, на пример: за компајлирање на програми напишани во контекстно слободен програмски јазик како што е предложено со пример 1.1.3. Но сепак, не е се изгубено. Се покажува дека за повеќето програмски јазици може да се конструираат детерминистички pushdown автомати кои ги прифаќаат сите синтаксно точни програми. Овие детерминистички машини се употребуваат во праксата не само за одбивање на програми со синтаксни грешки, туку и како синтаксни анализатори на програми, што е битна фаза во процесот на компајлирањето. Подолу ќе се презентираат две хевристички правила- тие се основни правила - корисни за конструирање детерминистички pushdown автомати од соодветна контекстно слободна граматика. Овие правила нема непроменливо да дадат pushdown автомат од секоја контекстно слободна граматика; веќе е кажано дека тоа би било невозможно. Но тие се типични за методите кои всушност се користат при конструкцијата на компајлери за програмски јазици.

2.1 Детерминистички pushdown автомати и контекстно - слободни јазици

Pushdown автоматот е *детерминистички*, интуитивно кажано, ако има најмногу една транзиција која може да ја изведиме за секоја конфигурација. Формално M е детерминистички ако за (p, w, γ) како конфигурација на M и за некои префикси w_1 и w_2 од w и γ_1 и γ_2 од γ двете $((p, w_1, \gamma_1), (q_1, \delta_1))$ и $((p, w_2, \gamma_2), (q_2, \delta_2))$ се транзиции на M, тогаш тие се идентични промени т.е. $w_1 = w_2$, $\gamma_1 = \gamma_2$, $q_1 = q_2$ и $\delta_1 = \delta_2$. Овој услов може да се парафразира на еквивалентен начин. Нека, два збора се нарекуваат **конзистентни** ако првиот е префикс на вториот или обратно. Две транзиции $((p_1, w_1, \gamma_1), (q_1, \delta_1))$ и $((p_2, w_2, \gamma_2), (q_2, \delta_2))$ се нарекуваат **компатибилни** ако $p_1 = p_2$, w_1 и w_2 се конзистентни и γ_1 и γ_2 се конзистентни. Тогаш, M е **детерминистички** ако нема две компатибилни транзиции.

На пример: машината која што е конструирана во пример 1.2.1 за прифаќање на $\{wcw^R: w \in \{a,b\}^*\}$ е детерминистичка, - таа за секој избор на состојба и влезен симбол има само една можна транзиција. Од друга страна машината која ја конструиравме во пример: 1.2.2 за прифаќање на $\{ww^R: w \in \{a,b\}^*\}$ не е детерминистичка. Третата транзиција е компатибилна со двете транзиции 1 и 2. Детерминистички контекстно слободни јазици се во суштина оние кои се прифатени од детерминистички pushdown автомат. Како и да е, од причини кое ќе се разјаснат подолу, мора малку да ја модифицираме прифатената конвенција. Ќе велиме дека јазикот е детерминистички контекстно-слободен ако е препознаен од детерминистички pushdown автомат кој има и дополнителна способност да го чувствува крајот на влезниот збор. Формално можеме да кажеме дека јазикот $L \subseteq \Sigma^*$ е **детерминистички контекстно слободен** ако $L\$ = L(M)$ за некој детерминистички pushdown автомат M. Овде $\$$ е нов симбол, кој не во Σ, додаден на секој влезен збор со цел да се обележи неговиот крај.

Секој детерминистички контекстно слободен јазик, како што е дефинирано, е контекстно слободен јазик. За да се види ова, се претпоставува дека детерминистичкиот pushdown автомат M го прифаќа $L\$$. Можеме да конструираме pushdown автомат M', кој може да е недетерминистички, што го прифаќа L. Во било која точка M' може да „замисли“ $\$$ во влезот и да скокне на ново множество состојби во кои што престанува со читање на влезни симболи.

Формално, ако $M=(K, \Sigma \cup \{\$\}, \Gamma, \Delta, s, F)$ тогаш $M' = (K', \Sigma, \Gamma, \Delta', s', F')$, каде

$$K' = \{ \langle p, i \rangle : p \in K, i = 0, 1\}$$
$$s' = \langle s, 0 \rangle$$
$$F' = \{ \langle f, 1 \rangle : f \in F\}$$

и Δ' го содржи следново

$((\langle p, 0\rangle, x, \alpha), (\langle q, 0\rangle, \beta))$ за секои $((p, x, \alpha), (q, \beta)) \in \Delta$, $x \in \Sigma^*$

$((<p, 0>, x, \alpha),(<q, 1>, \beta))$ за секои $((p, x\$, \alpha),(q, \beta)) \in \Delta$, $x \in \Sigma^*$
$((<p, 1>, e, \alpha), (<q, 1>, \beta))$ за секои $((p, e, \alpha), (q, \beta)) \in \Delta$

Може да се покаже дека *M'*, го прифаќа *L* кога *M* го прифаќа *L$*.

Од друга страна, доколку не се усвои овој договор „за прифаќање“, тогаш многу контекстно слободни јазици кои се интуитивно детерминистички нема да бидат детерминистички според нашата дефиниција. На пример, во случај кога $L = a^* \cup \{a^n b^n : n \geq 0\}$ детерминистичкиот pushdown автомат не може да запомни колку *a*-овци има видено, со цел да го провери подзборот од *b*-овци кој можеби следи и во исто време да биде спремен да го прифати зборот со празен скалд во случај да нема *b*-овци што следуваат. Но, може лесно да направиме детерминистички pushdown автомат што го прифаќа *L$*. Ако се наиде на *$* дури машината сеуште складира *a*-овци, значи дека влезот бил збор од a^*. Тогаш складот може да се испразни и влезот да се прифати.

Го променивме договорот „за прифаќање“ за да ја проширие класата на контекстно слободни јазици кои можат да бидат прифатени од детерминистички pushdown автомат. Природно се поставува прашањето: дали секој контекстно слободен јазик е детерминистички - како што секој регуларен јазик е прифатен од детерминистички конечен автомат. Би било изненадувачки доколку би било така. Имајки го во предвид, како пример, јазикот

$L_0 = \{a^{m(1)} b a^{m(2)} b \ldots a^{m(n)} : n \geq 2, m_{(1)}, \ldots, m_{(n)} \geq 0$ и $m_{(i)} \neq m_{(j)}$ за некои $i, j\}$.

Изгледа дека pushdown автоматот би можел да го прифати овој јазик само ако погоди кои два блока од *a*-овци да ги спореди. Ако нема погодок, по се изгледа дека автоматот нема начин да ги исчита двата блока, да ги спореди, а истовремено да ја запамти нивната должина за во случај тие да излезат дека се со еднаква должина да може да спореди со секој од наредниот блокови. Но, да се докаже дека L_0 не е детерминистички потребен е попрецизен аргумент.

Теорема 2.1.1

Нека $L \subseteq \Sigma^*$ е контекстно слободен јазик таков што

(a) за секој $u \in \Sigma^*$ постои $v \in \Sigma^*$ така што $uv \in L$;

(b) $\Sigma^* \setminus L$ не е контекстно слободен јазик

Тогаш L не е детерминистички контекстно слободен јазик.

Пред доказот, треба да се забележи дека теоремата 2.1.1 веднаш покажува дека јазикот L_0, горе дефиниран, не е детерминистички контекстно слободен, оти L_0 го има својството *(a)* и $\Sigma \setminus L_0 = a^* \cup \{a^m b a^m b \ldots a^m : m \geq 0\}$ е јасно дека не е контекстно слободен јазик.

Доказ*: Нека $M = (K, \Sigma, \Gamma, \Delta, s, F)$ е детерминистички pushdown автомат која што го прифаќа *L$*.

Можеме да претпоставиме дека во еден чекор *M* го проверува, со можност да го отстрани, единствено најгорниот симбол во неговиот склад. Исто така можеме да претпоставиме дека *K* се состои од две дисјунктни

подмножества, K_1 и K_2. K_1 се состои од состојби во кои што M може да влезе пред и само пред да биде прочитан $\$$, и K_2 кој што се состои од состојби во кои што M може да влезе откако и само откако $\$$ е прочитан; транзициите помеѓу состојбите во K_2 не вклучуваат читање на влез. Ова својство може да биде гарантирано со правење две копии од множеството состојби на M, така транзициите што го вклучуваат $\$$ како влез, преминуваат од првата копија во втората. Понатаму можеме да претпоставиме дека $F \subseteq K_2$, бидејќи M не прифаќа влез кој што не содржи $\$$.

Сега мораме да го анализираме однесувањето на M откако го чита $\$$, влегува во состојбите на K_2 и не продолжува со читање на влез. Конкретно, за $p \in K_2$, го разгледуваме множеството

$$X_p = \{\gamma \in \Gamma^* : (p, e, \gamma) \vdash_M^* (f, e, e) \text{ за некои } f \in F\}$$

Тврдиме дека секое X_p е регуларен јазик. За да го покажеме ова, нека земеме дека $\Theta = \{(r, a, q): r, q \in K_2, a \in \Gamma \text{ и } (r, e, a) \vdash_M^* (q, e, e)\}$

Множеството Θ, всушност, може да биде конструирано со проверка на M, но за овој доказ ни треба само заклучокот дека Θ е добро дефинирано множество од тројки. Тврдиме дека X_p е множеството од зборови прифатени од конечниот автомат $(K_2, \Gamma, \Theta, p, F)$. Ова е, затоа што, било кои операции да ги изведува M на складот додека е во K_2 и не чита ништо на влез, M може да го испразни складот само со серија од потпресметување, каде што секое од нив има краен ефект - исфрлање на единечен симбол од складот. Сега, бидејќи X_p е регуларен, таков е и $\Gamma^* \setminus X_p$. Нека $(K'_p, \Gamma, \Delta'_p, p, F'_p)$ е конечен автомат кој што го прифаќа $\Gamma^* \setminus X_p$. Да претпоставиме, без губење на општоста, дека K'_p и K'_q се дисјунктни за $p \neq q$, и дека $K'_p \cap K = \{p\}$ за секое $p \in K_2$. Нека Δ_2 биде множество од транзиции на M помеѓу состојбите во K_2 т.е.

$$\Delta_2 = \{ ((p, e, \gamma_1), (q, \gamma_2)) \in \Delta: p, q \in K_2, \ \gamma_1, \gamma_2 \in \Gamma^*\}.$$

Тогаш нека M' биде pushdown автомат $(K', \Sigma, \Gamma, \Delta', s, F')$ каде

$$K' = K \cup \bigcup \{K'_p : p \in K_2\}$$

$$\Delta' = \Delta \setminus \Delta_2 \cup \{ ((q_1, e, \gamma), (q_2, e)): (q_1, \gamma, q_2) \in \Delta'_p \text{ за некое } p \in K_2\}$$

$$F' = \bigcup \{F'_p : p \in K_2\}$$

Тоа значи, M' работи идентично со M се додека на влез не се појави \$; M' тогаш го испразнува својот склад и завршува во завршна состојба ако и само ако M не успеал да го испразни својот склад, или го испразнил но не успеал да влезе во завршна состојба.

Конечно, тврдиме дека M' го прифаќа $\Sigma^*\$ \setminus L\$$; ова ќе го комплетира доказот бидејќи и' противречи на хипотезата (b) дека, $\Sigma^*\$ \setminus L$ не е контекстно слободен. Го разгледуваме секој збор $w\$ \in \Sigma^*\$$. M, и соодветно M', всушност мора да го прочитаат целиот збор, па да влезат во состојба од K_2 откако ќе прочитаат $\$$, оти ако M откаже при читањето после одреден префикс u од w, тогаш според (а) ќе постои збор $uv\$ \in L\$$ кој M не го прифаќа. Но бидејќи влегол во состојба од K_2, M', нема да чита понатамошен влез и ќе го прифати само ако M не го прифаќа.

2.2 Top-down парсирање (Парсирање од_врв-надолу)

Откако покажавме дека не може секој контекстно слободен јазик да биде прифатен од детерминистички pushdown автомат, да разгледаме некои од оние кои можат. Нашата конечна цел е да ги проучиме случаите во кои контектно слободните граматики може да бидат претворени во детерминистички pushdown автомати кои што практично би можеле да се искористат во препознавање на јазиците. Наместо теореми и докази се презентираат таканаречени "хевристички правила" - што нема да бидат корисни во сите случаи, дури и без специфицирање *кога* тие *ќе бидат* корисни. Значи, целта е да се претстават некои примени на теоријата развиена погоре во овој труд.

Да почнеме со пример. Јазикот $L = \{ a^n b^n : n \geq 0 \}$ е генериран од формалната граматика $G = (\{S\}, \{a, b\}, R, S)$, каде R ги содржи двете правила $S \rightarrow aSb$ и $S \rightarrow e$. Знаеме како да конструираме pushdown автомат којшто го прифаќа L со употреба на граматиката G. Резултатот е

$$M_1 = (\{p, q\}, \{a, b\}, \{a, b, S\}, \Delta_1, p, \{q\}), \text{ каде}$$
$$\Delta_1 = \{((p, e, e), (q, S)), ((q, e, S), (q, aSb)), ((q, e, S), (q, e))$$
$$((q, a, a), (q, e)), ((q, b, b), (q, e))\}.$$

Бидејќи M_1 има два различни премини со идентични први компоненти - соодветни на двете правила од G кои имаат идентични леви страни - тој не е детерминистички.

Сепак L е детерминистички контекстно слободен јазик и M_1 може да биде модифициран да стане детерминистички pushdown автомат M_2 кој го прифаќа $L\$$. Интуитивно, цела информација која му е потребна на M_1 во секој момент за да одлучи која од двете транзиции ќе следи е *наредниот влезен симбол*. Ако тој симбол е a, тогаш M_1 треба да го замени S со aSb на неговиот склад. Од друга страна, ако наредниот влезен симбол е b, тогаш машината мора од складот да извади S. M_2 го остварува ова барано предвидување или **гледање нанапред** со јадење (читање) на влезен симбол *пред време* и вклопување на таа информација во својата состојба. Формално,

$$M_2 = (\{p, q, q_a, q_b, q_\$\}, \{a, b, \$\}, \{a, b, S\}, \Delta_2, p, \{q_\$\})$$

каде Δ_2 ги содржи следниве транзиции:

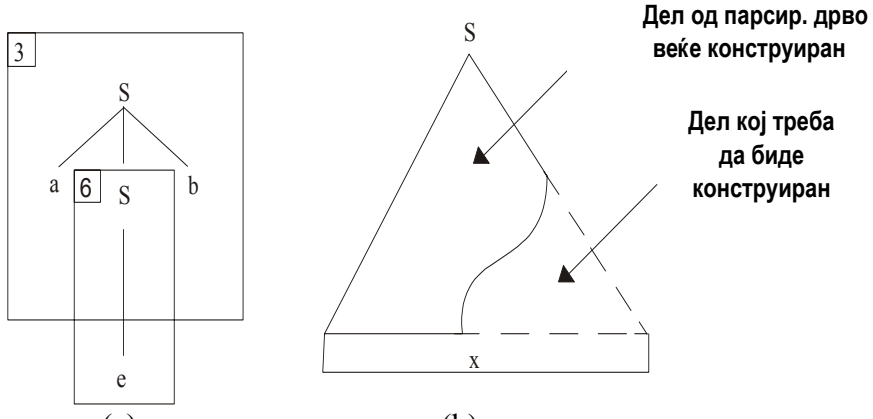
1. *((p, e, e), (q, S))*
2. *((q, a, e), (q_a, e))*
3. *((q_a, e, a), (q, e))*
4. *((q, b, e), (q_b, e))*
5. *((q_b, e, b), (q, e))*
6. *((q, $, e), ($q_\$$, e))*
7. *((q_a, e, S), (q_a, aSb))*
8. *((q_b, e, S),(q_b, e))*
9. *(($q_\$$, e, S),($q_\$$, e))*

Од состојба q, M_2 чита еден влезен симбол и без промена на складот, преминува во една од трите нови положби q_a, q_b или $q_\$$. Потоа ја употребува таа информација за да ги диференцира двете компитабилни транзиции *((q,e,S),(q,aSb))* и *((q,e,S),(q,e))*: Првата транзиција се изведува само од положбата q_a, и втората само од положбата q_b. Така што M_2 е детерминистички. Влезот *ab$* е прифатен на следниот начин:

чекор	сосотојба	влез	склад	употребена транзиција	Правило на G
0	*p*	*ab$*	*e*	--	
1	*q*	*ab$*	*S*	*1*	
2	q_a	*b$*	*S*	*2*	
3	q_a	*b$*	*aSb*	*7*	$S \rightarrow aSb$
4	*q*	*b$*	*Sb*	*3*	
5	q_b	*$*	*Sb*	*4*	
6	q_b	*$*	*b*	*8*	$S \rightarrow e$
7	*q*	*$*	*e*	*5*	
8	$q_\$$	*e*	*e*	*6*	

Така M_2 може да служи како детерминистичка машина за препознавање на зборови од облик $a^n b^n$. Освен тоа, со помнењето кои транзици од M_2, биле произлезени од кои правила на граматиката, ние би можеле да употребиме белег од операцијата M_2, со цел да реконструираме најлев извод на влезниот збор. Чекорите во пресметувањето, каде нетерминалот е заменет на врвот на складот (чекори 3 и 6 во примерот) одговараат на конструкцијата на парсирачкото дрво од коренот кон листовите. (види слика 2.2.1(a)).

Задачата за препознавање на контекстно слободен јазик на детерминистички начин кој исто така го реконструира парсирачкото дрво на зборот е исклучително важна и предизвикувачка. Направите како M_2 се наречени **парсери (анализатори).** Поточно**,** M_2, претставува **top-down парсер** бидејќи, следејќи ја неговата работа кај чекорите каде нетерминалите се заменети на складот го реконструира парсирачкото дрво на начин: горе-долу, лево-кон-десно. (види слика 3.2.1(b)).



Слика 2.2.1 Приказ на формирањето на парсирачко дрво

Јасно, сите контекстно слободни јазици немаат детерминистички препознавачи кои можат да бидат добиени од стандардниот недерминистички преку идејата за гледање нанапред. На пример, видовме дека некои контекстно слободни јазици воопшто не се детеминистички. Дури и за одредени детерминистички контекстно слободни јазици гледањето на еден симбол можеби ќе биде недоволно да ги реши сите несигурности. Некои јазици не подлегнуваат на „парсирањето со поглед нанапред“ од причини кои се површни и можат да бидат отстранети со мала модификација на граматиката.

Да се потсетиме на граматиката G која генерира аритметички изрази со операции + и * и променливи *x1* и *x2* (пример 1.1.3). Да пробаме да конструираме top-down парсер за оваа граматика. Нашата конструкција od делот 1.3 ке го даде pushdown автоматот

$$M_3 = (\{p, q\}, \Sigma, \Gamma, \Delta, p, \{q\})$$

со

$$\Sigma = \{(,), +, *, x, 1, 2\}$$
$$\Gamma = \Sigma \cup \{E, T, F\}$$

и Δ како што е дадено подолу.

0. *((p, e, e), (q, E))*
1. *((q, e, E), (q, E+T))*
2. *((q, e, E), (q, T))*
3. *((q, e, T), (q, T*F))*
4. *((q, e, T), (q, F))*
5. *((q, e, F), (q, (E)))*
6. *((q, e, F), (q, x1))*
7. *((q, e, F), (q, x2))*

Конечно, $((q, \sigma, \sigma), (q, e) \in \Delta$ за сите $\sigma \in \Delta$. Недетерминизмот кај M_3 е манифестиран со множеството транзиции 1-2, 3-4 и 5-6-7 кои имаат идентични први компоненти. Што е полошо, овде одлучувањето не може да биде базирано врз следниот влезен симбол. Да погледнеме од поблиску зошто е тоа така.

Транзиции 6 и 7. Да претпоставиме дека конфигурацијата на M_3 е (q, x_1, F). Во ваков момент, M_3 може да дејствува според било која од транзициите 5, 6 и 7. Со согледување на следниот влезен симбол -*x*- M_3, може да ја исклучи транзицијата 5, оти оваа транзиција бара следниот симбол да биде "(". Сеуште, M_3, нема да може да одлучи помеѓу транзициите 6 и 7, бидејќи тие и двете продуцираат врв на складот кој може да се совпадне со следниот влезен симбол -*x*. Проблемот се појавува поради правилата $F \rightarrow x1$ и $F \rightarrow x2$ од *G* кои не само што имаат идентична лева страна, туку имаат ист прв симбол на нивните десни страни.

Постои многу лесен начин за надминување на овој проблем: Само заменете ги правилата $F \rightarrow x1$ и $F \rightarrow x2$ во *G* со правилата $F \rightarrow xN$, $N \rightarrow 1$ и $N \rightarrow 2$,

каде N е нов нетерминал. Ова има ефект на одложување на одлуката помеѓу правилата $F \rightarrow x1$ и $F \rightarrow x2$ од G додека сите потребни информации се достапни. Модифицираниот pushdown автомат M_3 сега резултира од оваа модифицирана граматика, во која транзициите 6 и 7 се заменети со следново:

6'. ((q, e, F), (q, xN))
7'. ((q, e, N), (q, 1))
8'. ((q, e, N), (q, 2))

Сега гледајќи еден симбол е доволно да се одлучи точната активност. На пример, конфигурацијата *(q, x_1, F)* ќе резултира со *(q, x_1, xN), (q, 1, N), (q, 1, 1)* и конечно *(q, e, e)*.

Оваа техника на избегнување на нетерминизам е позната како **лево разложување**. Може да биде парафразирана на следниот начин:

Хевристичко правило 1:

Секогаш кога $A \rightarrow \alpha\beta$ и $A \rightarrow \alpha\gamma$ се правила т.ш. $\alpha \neq e$, заменете ги со правилата $A \rightarrow \alpha A'$, $A' \rightarrow \beta$ и $A' \rightarrow \gamma$, каде A' е нов нетерминал.

Лесно е да се види дека вклучувањето на Хевристичкото правило 1 не го менува јазикот генериран од граматиката. Сега ќе преминеме на разгледување на вториот вид на аномалија кои не спречува од трансформирање на M_3 во детерминистички парсер.

Транзиција 1 и 2. Овие транзиции ни укажуваат на посериозен проблем. Ако автоматот го има *x* за следен влезен симбол и содржината на складот е само *E*, може да превземе повеќе дејства. Може да ја изведе транзицијата 2, заменувајќи го *E* со *T* (ова ќе биде оправдано во случај кога влезот е, да речеме, *x1*). Или тој може да го промени *E* со *E+T* (транзиција 1), па потоа да го промени *E*-то на врвот со *T* (ова треба да биде направено ако влезот е *x1+x1*). Или, тој може да ја изведе транзиција 2 двапати и транзиција 1 еднаш (влез *x1+x1+x1*), и така натаму. Се чини дека не постои граница до каде автоматот мора да гледа (sирка) со цел да се одлучи на вистинскиот чекор. Проблемот во овој случај е правилото $E \rightarrow E+T$, во кој нетерминалот на левата страна е повторен како прв симбол на десната страна. Оваа појава е наречена **лева рекурзија** и може да биде отстранета со натамошно изменување на граматиката.

За да се отстрани левата рекурзија од правилото $E \rightarrow E+T$, го заменуваме со правилата $E \rightarrow TE'$, $E' \rightarrow +TE'$, и $E' \rightarrow e$, каде E' е нов нетерминал. Вакви трансформации не го менуваат јазикот, произведен од граматиката. Истиот метод мора да биде применет на другото лево рекурзивно правило на *G*, имено $T \rightarrow T*F$. Така ја добиваме граматиката $G' = (V', \Sigma, R, E)$ каде $V' = \{E, E', T, T', F, N\}$ и правилата се како што следуваат:

1. $E \rightarrow TE'$
2. $E' \rightarrow +TE'$
3. $E' \rightarrow e$
4. $T \rightarrow FT'$
5. $T' \rightarrow *FT$
6. $T' \rightarrow e$
7. $F \rightarrow (E)$
8. $F \rightarrow xN$
9. $N \rightarrow 1$
10. $N \rightarrow 2$

Горната техника за отстранување на левата рекурзија може да биде изразена како што следи. (Се претпоставува дека не постојат правила од вид $A \rightarrow A$.)

Хевристичко правило 2:

Нека $A \rightarrow A\alpha_1,\ldots,A \rightarrow A\alpha_n$ и $A \rightarrow \beta_1,\ldots,A \rightarrow \beta_m$ бидат сите правила со A на левата страна, каде β-овците не почнуваат со A и $n > 0$. Тогаш замени ги овие правила со $A \rightarrow \beta_1 A', \ldots, A \rightarrow \beta_m A', A' \rightarrow \alpha_1 A', \ldots, A' \rightarrow \alpha_n A'$ и $A' \rightarrow e$, каде A' претставува нов нетерминал.

Lemma 2.2.1.

Нека G биде контекстно-слободна граматика и нека G' биде граматика добиена со примена на хевристичкото правило 2 на G. Тогаш L(G)=L(G'). □

Сеуште граматиката G' од нашиот пример има правила со идентични леви страни, само што сега сите недостатоци можат да бидат решени со гледање кон следниот влезен симбол. Ние можеме да го конструираме следниот детерминистички pushdown автомат M_4 кој прифаќа $L(G)\$$.

$$M_4=(K, \Sigma \cup \{\$\}, V', \Delta, p, \{q_\$\})$$

каде

$$K=\{p, q, q_x, q_1, q_2, q_+, q_*, q_), q_(, q_\$\}$$

и Δ дадена со:

$((p, e, e), (q, E))$

$((q, \sigma, e), (q_\sigma, e))$ за секое $\sigma \in \Sigma \cup \{\$\}$

$((q_\sigma, e, \sigma), (q, e))$ за секое $\sigma \in \Sigma$

$((q_\sigma, e, E), (q_\sigma, TE'))$ за секое $\sigma \in \Sigma \cup \{\$\}$

$((q+, e, E'), (q+, +TE'))$

$((q_\sigma, e, E'), (q_\sigma, e))$ за секое $\sigma \in \Sigma \setminus \{+\} \cup \{\$\}$

$((q_\sigma, e, T), (q_\sigma, FT'))$ за секое $\sigma \in \Sigma \cup \{\$\}$

$((q_*, e, T'), (q_*, *FT'))$

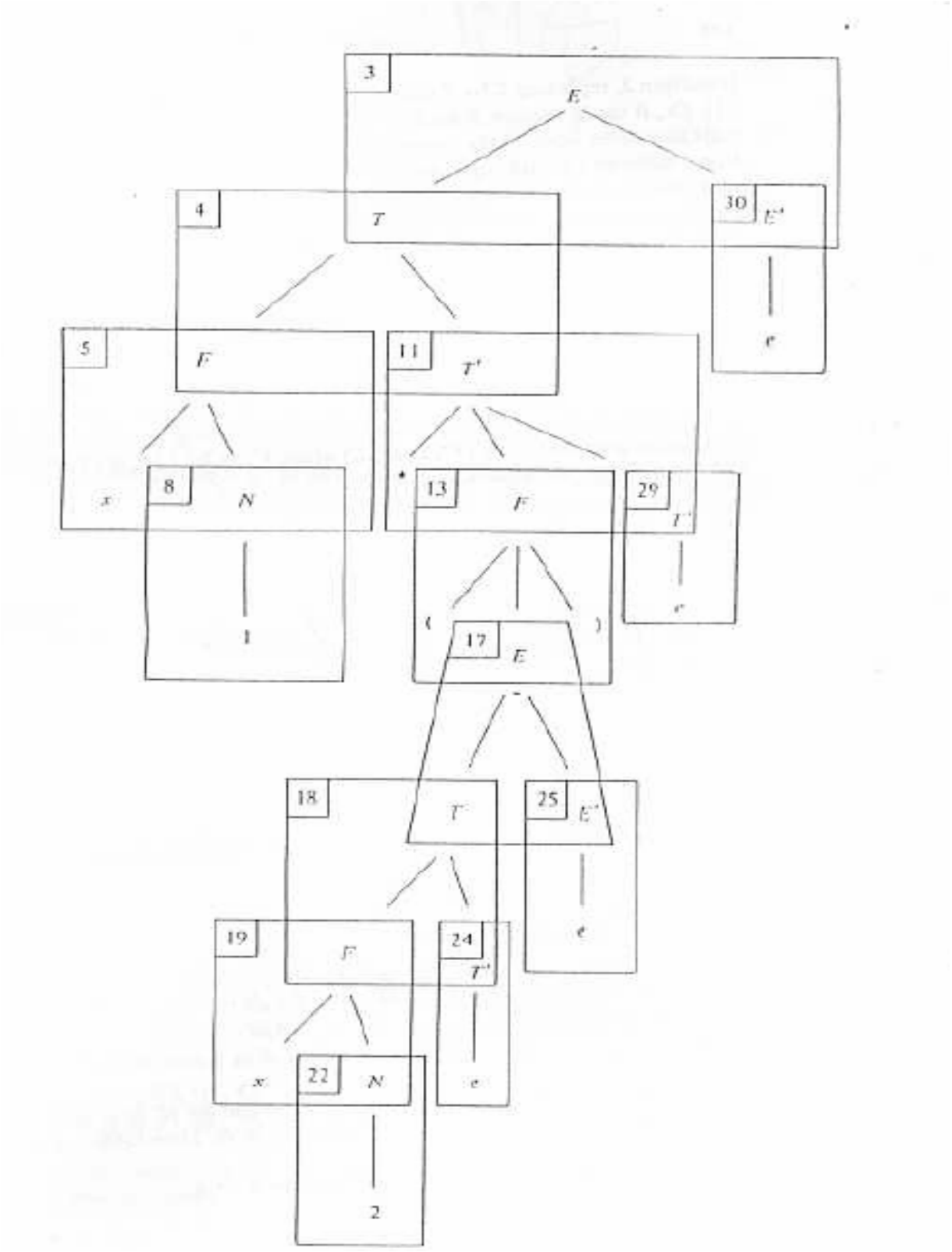
$((q_\sigma, e, T'), (q_\sigma, e))$ за секое $\sigma \in \Sigma \setminus \{*\} \cup \{\$\}$
$((q(, e, F), (q(, (E)))$
$((q_x, e, F), (q_x, xN))$
$((q_1, e, N), (q_1, 1))$
$((q_2, e, N), (q_2, 2))$

Тогаш M_4 е парсер за Γ'. На пример, влезниот збор *x1*(x2)$* ќе биде прифатен како што следува.

Чекор	Состојба	Непрочитан влез	Склад	Правило на G'
0	p	*x1*(x2)$*	e	
1	q	*x1*(x2)$*	E	
2	q_x	*1*(x2)$*	E	
3	q_x	*1*(x2)$*	TE'	1
4	q_x	*1*(x2)$*	$FT'E'$	4
5	q_x	*1*(x2)$*	$xNT'E'$	8
6	q	*1*(x2)$*	$NT'E'$	
7	q_1	**(x2)$*	$NT'E'$	
8	q_1	**(x2)$*	$1T'E'$	9
9	q	**(x2)$*	$T'E'$	
10	q_*	*(x2)$*	$T'E'$	
11	q_*	*(x2)$*	$*FT'E'$	5
12	q	*(x2)$*	$FT'E'$	
13	$q_($	*x2)$*	$FT'E'$	
14	$q_($	*x2)$*	$(E)T'E'$	7
15	q	*x2)$*	$E)T'E'$	
16	q_x	*2)$*	$E)T'E'$	
17	q_x	*2)$*	$TE')T'E'$	1
18	q_x	*2)$*	$FT'E')T'E'$	4
19	q_x	*2)$*	$xNT'E')T'E'$	8
20	q	*2)$*	$NT'E')T'E'$	
21	q_2	*)$*	$NT'E')T'E'$	
22	q_2	*)$*	$2T'E')T'E'$	10
23	q	*)$*	$T'E')T'E'$	
24	$q_)$	*$*	$T'E')T'E'$	
25	$q_)$	*$*	$E')T'E'$	6
26	$q_)$	*$*	$)T'E'$	3
27	q	*$*	$T'E'$	
28	$q_\$$	e	$T'E'$	
29	$q_\$$	e	E'	6
30	$q_\$$	e	e	3

Овде е укажано на чекорите во пресметувањето каде нетерминал е заменет на складот во согласност со правило од G'. Со примена на овие правила

од *G'* во секвенца, изводот на влезот може да биде реконструиран; соодветното парсирачко дрво е прикажано на Слика 2.2.2. Броевите ги поврзуваат чекорите во пресметувањето со деловите од парсирачкото дрво.



Слика 2.2.2 Парсирачко дрво за *x1*(x2)$*

Така, за дадена граматика *G*, може да се направи обид да се конструира top-down парсер за *G* на следниов начин:

Прво, се елиминира левата рекурзија во *G* со повеќекратна употреба на хевристичкото правило 2 на сите лево рекурзивни нетерминали *A* од *G*. Во продолжение се употребува хевристичко правило 1 за лево разложување на *G*, секогаш кога е потребно. Потоа се проверува дали резултантната граматика може да се одлучи за едно од правилата со иста лева страна, со гледање на следниот влезен симбол. Граматиките со ова својство се наречени LL(1). И покрај тоа што не прецизиравме точно како се одредува дали граматиката е всушност LL(1), ниту како да се конструира соодветен детерминистички парсер ако граматиката е LL(1), сепак постојат систематски методи за извршување на истото.

2.3 Bottom – up парсирање (Парсирање од_дно – нагоре)

Како што беше дискутирано во претходната глава не постои најдобар начин за парсирање на контекстно слободен јазик. Практично понекогаш се преферираат различни методи за различни граматики. Сега ќе разгледаме методи кои се различни од оние со top-down Сепак, исто така, нивната генеза може да се најде во конструкцијата на pushdown автомат.

Како дополнување на конструкцијата на pushdown автомат дадена во 1.3, постои сосема различен начин на конструкција на pushdown автомат која го прифаќа јазикот генериран од даден контекстно слободен јазик. Автоматите од таа конструкција (од која што се изведени top-down парсерите) работат со изведба на најлев извод на складот; како што се генерираат терминалните симболи така се споредуваат со влезниот збор. Во конструкцијата дадена подолу, автоматот се обидува првин да го прочита влезот и врз основа на влезот кој е всушност прочитан, да одреди кој извод треба да се обиде да го изврши. Идејата е да се реконструира парсирачкото дрво од листовите кон коренот, а не обратно, па така оваа класа на методи се нарекува **bottom-up** (од_дно – нагоре).

Bottom-up pushdown автомат се конструира на следниот начин:

Нека $G = (V, \Sigma, R, S)$ биде било која контекстно слободна граматика. Тогаш нека $M = (K, \Sigma, \Gamma, \Delta, s, F)$, каде $K = \{p, q\}$, $\Gamma = V$, $s = p$, $F = \{q\}$, и Δ се состои од следново:

1. $((p, \sigma, e), (p, \sigma))$ за секое $\sigma \in \Sigma$.
2. $((p, e, \alpha^R), (p, A))$ за секое правило $A \rightarrow \alpha$ во R.
3. $((p, \sigma, S), (q, e))$

Пред да преминеме на докажување дека јазиците генерирани од G и M се еднакви, да ги споредиме овие типови на транзиции со оние кај автоматот конструиран во 1.3. Транзициите од тип 1 овде ги префрлаат влезните симболи на складот; транзициите од тип 3 во конструкцијата во 1.3 ги исфрлаат терминалните симболи од складот кога тие се еднакви со влезните симболи. Транзициите од тип 2, овде, ја заменуваат десната страна од правилото, на складот, со соодветната лева страна, кога десната страна од правилото е најдена „обратно“ на складот. (Транзициите од тип 2 во конструкцијата во 1.3 ја заменуваа левата страна од правилото со соодветната негова десна страна). Транзициите од тип 3 овде го завршуваат препознавањето (рабоата на автоматот) со префрлање во крајна состојба кога само почетниот симбол останува на складот (транзициите од тип 1 во 1.3 го почнуваа препознавањето со ставање на почетниот симбол на иницијалнио празениот склад.

Значи, машината на оваа конструкција е во суштина ортогонална во однос на конструираната во 1.3.

Lemma 2.3.1. Нека G и M се дадени како погоре. Тогаш $L(M) = L(G)$

***Доказ**:*

Дефинираме дека **најдесен извод** кај контекстно слободна граматика е извод кај кој во секој чекор се заменува најдесниот нетерминал.

Дефиницијата е аналогна на дефиницијата за лев извод и аналогно може да тврдиме дека секој збор од јазикот има најдесен извод од почетниот симбол.

Поради тоа доказот на следното тврдење ќе биде доволно да се докаже Lemma 3.3.1**.**

Тврдење. За било кој $x \in \Sigma^*$ и $\gamma \in \Gamma^*$, (p, x, γ) |-- $^*_M$ (p, e, S) ако и само ако $\gamma^R x$ се добива од S преку нула или повеќе најдесни изводи.

Ако x е влез на M и $\gamma = e$, тогаш поради тоа што q е единствената крајна состојба и во неа може да се влезе само преку транзиција 3, тврдењето имплицира дека M го прифаќа x ако и само ако G го генерира x.

Доказот на тврдењето нема да го дадеме. Се изведува и во двете насоки со индукција по должина на влезниот збор. |||

Да ја разгледаме уште еднаш граматиката за аритметички изрази (пример 1.1.3) анализирана во претходниот дел. Правилата на оваа граматика, G, се следниве:

$E \rightarrow E + T$	(R1)
$E \rightarrow T$	(R2)
$T \rightarrow T * F$	(R3)
$T \rightarrow F$	(R4)
$F \rightarrow (E)$	(R5)
$F \rightarrow x1$	(R6)
$F \rightarrow x2$	(R7)

Ако на оваа граматика се примени конструкцијата на bottom-up pushdown автомат, се добива следната група на транзиции.

$((p, \sigma, e), (p, \sigma))$	за секој $\sigma \in \Sigma$	$(\Delta 0)$
$((p, e, T + E), (p, E))$		$(\Delta 1)$
$((p, e, T), (p, E))$		$(\Delta 2)$
$((p, e, F * T), (p, T))$		$(\Delta 3)$
$((p, e, F), (p, T))$		$(\Delta 4)$
$((p, e,)E( ), (p, F))$		$(\Delta 5)$
$((p, e, 1x), (p, F))$		$(\Delta 6)$
$((p, e, 2x), (p, F))$		$(\Delta 7)$
$((p, e, E), (q, e))$		$(\Delta 8)$

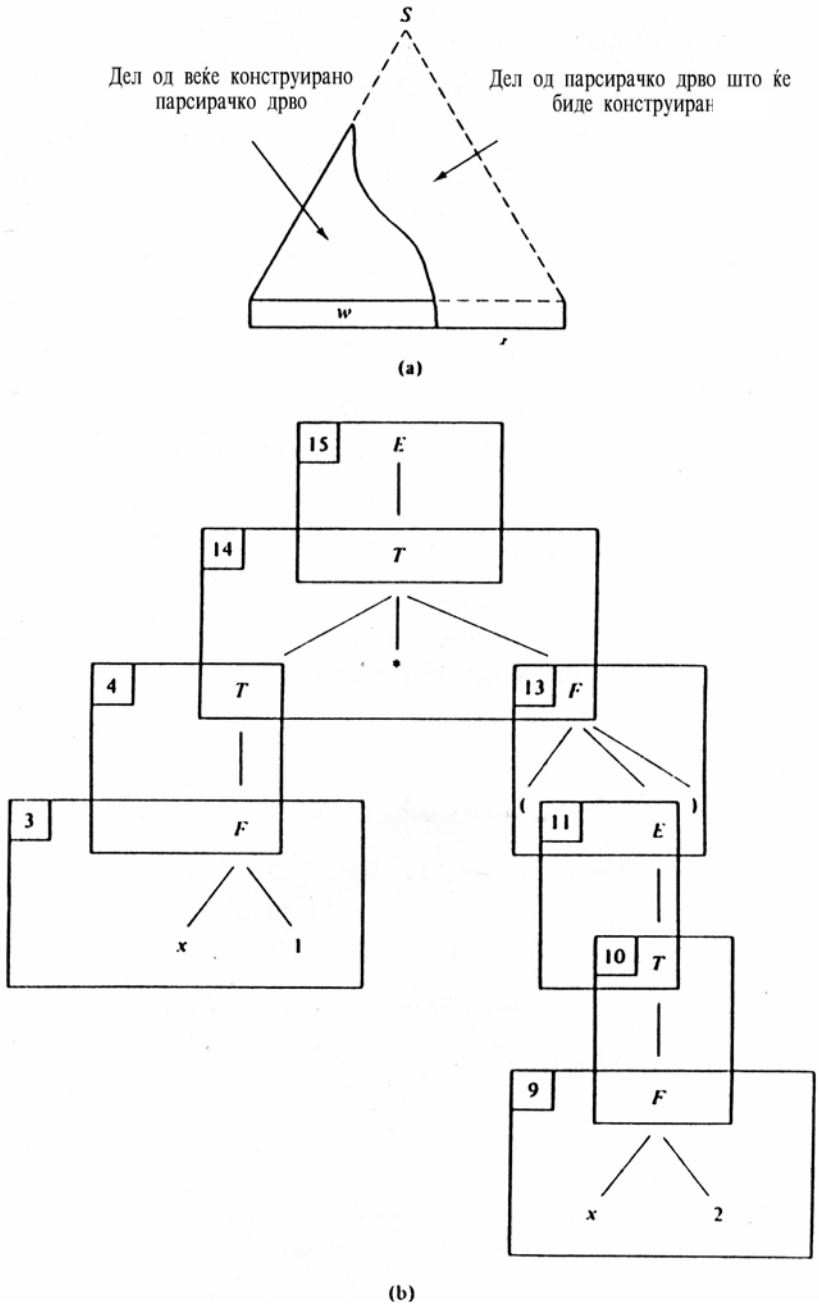
Да го наречеме овој pushdown автомат M. Влезот *x1*(x2)* е прифатен од M на следниов начин:

Чекор	Состојба	Непрочитан влез	Склад	Користена транзиција	Правило на G'
0	*P*	*x1*(x2)*	*e*		
1	*P*	*1*(x2)*	*x*	Δ0	
2	*P*	**(x2)*	*1x*	Δ0	
3	*P*	**(x2)*	*F*	Δ6	R6
4	*P*	**(x2)*	*T*	Δ4	R4
5	*P*	*(x2)*	**T*	Δ0	
6	*P*	*x2)*	*(*T*	Δ0	
7	*P*	*2)*	*x (*T*	Δ0	
8	*P*	*)*	*2x (*T*	Δ0	
9	*P*	*)*	*F(*T*	Δ7	R7
10	*P*	*)*	*T(*T*	Δ4	R4
11	*P*	*)*	*E(*T*	Δ2	R2
12	*P*	*e*	*)E(*T*	Δ0	
13	*P*	*e*	*F*T*	Δ5	R5
14	*P*	*e*	*T*	Δ3	R3
15	*P*	*e*	*E*	Δ2	R2
16	*Q*	*e*	*e*	Δ8	

Од ова може да се види дека *M* не е детерминистички. Транзициите од типот Δ0 се компатибилни со транзициите од типот Δ1 до Δ3. Сепак, може да се согледа основниот концепт на работа на *M*. Во секој момент *M* може да пренесе терминален симбол од влезот на врвот на складот (транзициите од типот Δ 0, користени во примерот во чекорите 1, 2, 5, 6, 7, 8 и 12). Од друга страна, од време на време, може да ги препознае горните неколку симболи на складот како „обратна“ десна страна од правило од *G*, и тогаш може да го **сведе** овој подзбор со соодветната лева страна (транзициите од типот Δ2 до Δ7, употребени во примерот каде на „правилото од G “ е укажано во најдесната колона). Низата од правила соодветна на **сведувачките чекори** се добива дека точно го отсликува, во обратен редослед, најдесниот извод на влезниот збор. Во нашиот пример, имплицираниот најдесен извод е:

$$
\begin{aligned}
E &\Rightarrow T \\
&\Rightarrow T * F \\
&\Rightarrow T * (E) \\
&\Rightarrow T * (T) \\
&\Rightarrow T * (F) \\
&\Rightarrow T * (x2) \\
&\Rightarrow F * (x2) \\
&\Rightarrow x1 * (x2)
\end{aligned}
$$

Овој извод може да се прочита од препознавањето, погоре, со примена на правилата на кои е укажано во десната колона, од последно кон прво, секогаш на најдесниот нетерминал. Еквивалентно, овој процес може да се разгледува како реконструкција од дното-кон-врвот, од лево-кон десно, на парсирачко дрво (Види Слика 2.3.1 (a); споредена со Слика 2.2.1 (a)). Во нашиот пример,



Слика 2.3.1 (a) Приказ на формирањето на парсирачко дрво
(b) Парсирачко дрво за *x1*(x2)*

реконструираното парсирачко дрво е покажано на Слика 2.3.1 (b); чекорите на пресметување (препознавање) се покажани горе лево во секое квадратче.

За да конструираме практично употребливо парсирачко дрво од *L(G)*, мораме *M* да го претвориме во детерминистички автомат што ќе го прифаќа *L(G)$*. Овде нема да дадеме систематска процедура, како при

обработката на top-down парсерите. Наместо тоа, низ примерот за G, ќе се укаже на основните хевристички принципи според кои се изведува оваа конструкција.

Прво, ни треба начин да одлучуваме помеѓу двата основни типови на потези - *преместување* на наредниот влезен симбол во складот и *сведување* на неколкуте најгорни симболи од складот во еден нетерминал според правило од граматиката. Ова го разрешуваме со разгледување на две информации: наредниот влезен симбол – да го наречеме b - и симболот на врвот на скалдот – да речеме a. (Симболот a може да биде и нетерминал.) Одлучувањето помеѓу преместувањето и сведувањето е направено со релацијата $P \subseteq (V \cup \Sigma \cup \{\$\}) \times (\Sigma \cup \{\$\})$, наречена **релација на претходење**. Ако a и b се дефинирани како погоре и $(a, b) \in P$ тогаш сведуваме; инаку го преместуваме b. Точната релација на претходење од граматиката во нашиот пример е дадена на слика 2.3.2.

	x	(	)	1	2	+	*	$
x								
(								
)			X			X	X	X
1			X			X	X	X
2			X			X	X	X
+								
*								
E								
T			X			X		X
F			X			X	X	X

Слика 2.3.2 Табеларен приказ на релацијата на претходење

Интуитивно $(a, b) \in P$ значи дека постојат најдесни деривации од облик $S \Rightarrow^{*R} \beta Abx \Rightarrow^{R} \beta\gamma abx$, т.е. постои најдесен извод во нула или повеќе чекори на βAbx од S и најдесен извод во еден чекор на $\beta\gamma abx$ од βAbx. Бидејќи ги реконстурираме најдесните изводи наназад, тогаш има смисла да го откажеме („однаправиме“) правилото $A \rightarrow \gamma a$ секогаш кога забележиме дека a му претходи на b. Постојат систематски начини за пресметување на релациите на претходење, како и начини за откривање кога, како во нашиот пример, релацијата на претходење е доволна за да се одлучи помеѓу преместување и сведување.

Но, мора да се соочиме и со другиот извор на недетерминизам: кога ќе одлучиме да сведуваме, како ќе одлучиме кој од префиксите од pushdown складот ќе го замениме со нетерминал? На пример, ако pushdown складот го содржи подзборот $F * T + E$ и мораме да сведуваме, имаме избор помеѓу сведување на F со T (правило R4) или сведување $F * T$ во T (правило R3). За нашата граматика доволно е да се избере *најдолгиот* префикс од складот што се совпаѓа со обратниот од десната страна на некое правило и да се сведе со

левата страна на тоа правило. Според тоа во овој случај ние треба да ја земеме втората опција и да сведеме $F * T$ во T.

Со овие две набљудувања можеме да конструираме детерминистички pushdown автомат M′ што го парсира L(G)$. Мора да се прецизира уште нешто. За да се изведе нашето второ хевристичко правило (сведување на најдолгиот можен префикс од складот), автоматот мора да го „чувствува" дното од складот. Ова е остварено со ставање на специјален симбол # во складот на почетокот на секоја операција.

Формално, нека $M' = (K', \Sigma', \Gamma', \Delta', s, F')$ каде што

$$K' = \{s, p\} \cup \{ p_\sigma : \sigma \in \Sigma' \}$$
$$\Sigma' = \Sigma \cup \{\$\}$$
$$\Gamma' = V \cup \Sigma \cup \{\$, \#\}$$
$$F' = \{ p_\$ \}$$

и Δ' е наведена долу.

$\Delta'1$:	$((s, e, e), (p, \#))$	
$\Delta'2$:	$((p, \sigma, e), (p_\sigma, e))$	за $(\forall \sigma \in \Sigma')$
$\Delta'3$:	$((p_\sigma, e, a), (p, \sigma a))$	за $(\forall (a,\sigma) \in (V \cup \Sigma \cup \{\$\}) \times \Sigma' \setminus P$ (транзициите од типот $\Delta'3$ одговараат на преместувањето)
$\Delta'4$:	$((p_\sigma, e, T+E), (p_\sigma, E))$	за $(\forall \sigma \in \Sigma')$ т.ш. $(T, \sigma) \in P$
$\Delta'5$:	$((p_\sigma, e, Ta), (p_\sigma, Ea))$	за $(\forall \sigma \in \Sigma')$ т.ш. $(T, \sigma) \in P$ и $a \in \Gamma' \setminus \{+\}$
$\Delta'6$:	$((p_\sigma, e, F*T), (p_\sigma, T))$	за $(\forall \sigma \in \Sigma')$ т.ш. $(F, \sigma) \in P$
$\Delta'7$:	$((p_\sigma, e, Fa), (p_\sigma, Ta))$	за $(\forall \sigma \in \Sigma')$ т.ш. $(T, \sigma) \in P$ и $a \in \Gamma' \setminus \{*\}$
$\Delta'8$:	$((p_\sigma, e,)E( ), (p_\sigma, F))$	за $(\forall \sigma \in \Sigma')$ т.ш. $( ), \sigma) \in P$
$\Delta'9$:	$((p_\sigma, e, 1x), (p_\sigma, F))$	за $(\forall \sigma \in \Sigma')$ т.ш. $(1, \sigma) \in P$
$\Delta'10$:	$((p_\sigma, e, 2x), (p_\sigma, F))$	за $(\forall \sigma \in \Sigma')$ т.ш. $(2, \sigma) \in P$
$\Delta'11$:	$((p_\$, e, E\#), (p_\$, e))$	

Овде M' е детерминистички. Освен тоа, може да се покаже дека тој го прифаќа $L(G)\$$, што всушност значи дека тој е bottom-up парсер за $L(G)\$$.

На крај, уште еднаш да нагласиме дека дадените хевристички правила не даваат резултати во сите ситуации. Граматиките за кои тие даваат резултати се нарекуваат **граматики со *слабо* претходење**; во практика некои граматики се или можат да бидат конвертирани во граматика со слабо претходење.

Користена литература:

[1] Биљана Јанева: *„Алгоритми и автомати“* . Предавања 2002.

[2] H. R. Lewis, C. H. Papadimitriou *„Elements of theory of computation“*. Prentice-Hall International, Inc, 1981

[3] J. E. Hopcroft, J. D. Ullman: *„Formal languages and their relation to automata“*. Addison-Wesley Publishing Company, 1969

[4] R. Durante, J. Ryden: „ Humanoid Robotics“ Machine Intelligence: University of Newcastle

[5] Катерина Здравкова: „Програмски Системи“ Предавања, 2001-2002

[6] Hahn, Udo. Schacht, Susanne. Bröker, Norbert. *Concurrent, Object-Orientated Natural Language Parsing: **The ParseTalk** Model.* Arbeitsgruppe Linguistische Informatik/Computerlinhguistik. Freiburg: 1995.

[7] M. J. Weiss „Zing went the spring ...“

[8] Ѓорѓи Јованчевски: „Програмски Јазици“ Предавања, 2001

[9] Generation5: „ An Introduction to Robotics“ 1998-2002