МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ”ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Навчальний посібник
з дисципліни

”ФОРМАЛЬНІ МОВИ, ГРАМАТИКИ ТА АВТОМАТИ”

для студентів напрямку ”Комп‘ютерні науки” та «Системний аналіз»

Затверджено
на засіданні кафедри
інформаційних систем та мереж
Протокол № 11 від 16.02.2016р.

Львів-2016

Формальні мови, граматики та автомати. Навчальний посібник до вивчення дисципліни для студентів напрямку ”Комп‘ютерні науки” та «Системний аналіз» Укл.: Л.М. Захарія, М.М. Заяць. – Львів: Видавництво Національного університету «Львівська політехніка», 2016. – 210с.

Укладачі: Захарія Л.М., кан. фіз.-мат. наук, доцент
Заяць М.М., асист.

Відповідальний за випуск: Литвин В.В., докт. тех. наук, професор

Рецензенти: Шаховська Н.Б., докт. техн. наук, професор каф. ІСМ,
Цегелик Г.Г., докт. фіз-мат. наук, професор, зав. каф. математичного моделювання соціально-економічних процесів ЛНУ ім.І.Франка,
Попович В.С., докт. техн. наук, професор, заступник директора ІППММ

ЗМІСТ

ПЕРЕДМОВА

Мета цього курсу - познайомити читача з деякими основними моделями і результатами, використовуваними в теоретичній інформатиці. Не дивно, що вони відносяться до математики, а не до будь-якої іншої галузі знань - адже в науці про комп'ютери саме математичні абстракції є найбільш плідними.

Розглянуті тут ідеї і результати належать до теорії формальних мов, граматик і автоматів. По суті, ця теорія описує деякі обмежені абстрактні машини, здатні виконувати певні операції з рядками. Наприклад, кінцевий автомат може з'ясувати, чи містить деякий файл певне слово, а автомат з магазинної пам'яттю здатний визначити, чи правильна система вкладених круглих, квадратних і фігурних дужок.

Як підказує сама назва курсу, основним об'єктом розгляду є формальна мова - довільна множина скінченних послідовностей символів, взятих з деякої скінченної множини (такі послідовності називаються словами).

У першому розділі дається класифікація формальних мов відповідно до ієрархії Хомського: автоматні мови, контекстно-вільні (або безконтексні) мови, контекстні (або контекстно-залежні) мови, мови типу 0. Якщо виключити мови, що містять порожнє слово, то названі класи вкладені один в одного (тут вони перераховані в порядку зростання).

У розділах 2-3 розглядається найвужчий з цих класів - клас автоматних (або праволінійних, або регулярних) мов, що володіє багатьма властивостями, важливими для програмних додатків. Саме автоматні мови лежать в основі лексичних аналізаторів, що входять в структуру компіляторів і інтерпретаторів мов програмування. Лексичному аналізу присвячено розділ 4.

У розділах 5-6 розглядаються контекстно-вільні мови, різні їх форми представлення та способи еквівалентного перетворення з однієї форми в іншу. Контексно-вільні мови використовуються при визначенні сучасних мов програмування (а також мов пошукових запитів і т. п.) Основна частина синтаксису описується в формалізмі, відомому як форма Бекуса-Наура. По суті, це спосіб записувати контекстно-вільні граматики. Інша важлива область використання контекстно-вільних граматик - розширювана мова розмітки (XML) і мови опису структури XML-документа (наприклад, мова DTD).

Два класи з ієрархії Хомського, що залишилися - контекстні мови і мови типу 0 - в даному курсі вивчаються менш докладно, оскільки вони відносяться швидше до теорії складності обчислень і до теорії алгоритмів відповідно. Також тут наводяться еквівалентні визначення цих класів в термінах автоматів.

Розділи 7 та 9 містять основні результати теорії детермінованих контекстно-вільних мов і формальні визначення, що демонструють зв'язок цієї теорії зі спадними та висхідними синтаксичними аналізаторами.

Говорячи «формальна мова», ми будемо говорити в першу чергу про штучні мови, створені людьми для певних задач, наприклад мови програмування. Однак між штучними та природніми мовами фундаментальних розбіжностей не існує: природні мови часто характеризуються доволі складними граматичними правилами, що жорстко формалізують їх, тоді як в найдосконаліших штучних мовах іноді виникають такі непередбачувані аспекти поведінки, однозначне розуміння яких стає проблемою.

Вивчаючи мови, ми повинні мати на увазі наступні три аспекти.

Перший аспект – *синтаксис* мови. Мова – це деяка множина «слів», де слово – це скінченна послідовність «букв» - символів деякого зафіксованого алфавіту. Терміни «слово», «буква» можна визначати по різному. Так «буквами» можуть бути окремі символи української мови чи паскаль-програми, а «слова» - відповідно їх скінченні послідовності, наприклад, «крокодил» - в українській мові, «integer» - в паскалі. Такі слова називають лексемами. В той же час «буквами» можна вважати окремі слова, тоді «словами» стають речення української мови чи програми на мові паскаль, наприклад. Якщо зафіксована послідовність «букв», то не кожне таке слово буде лексемою мови, наприклад «кракодил» - не є лексемою української мови, а iff – не є лексемою мови паскаль. Тому в мовах існує множина правил правильної побудови лексем мови, яка називається *синтаксисом* мови (слово синтаксис походить від давньогрецького syn – разом, та taxis – порядок, тобто розглядалось як порядок складання разом). Оскільки кожне «слово» характеризується певною структурою, специфічною саме для цієї мови, то необхідно розробити механізми перечисленні або породження «слів», а з іншого боку розробити механізми, які б дозволяли визначити чи являється «слово» лексемою, тобто чи належить воно даній мові. Саме механізми породження та розпізнавання «слів» і вивчає класична теорія формальних мов.

Другий аспект – *семантика* мови. Семантика передбачає співставлення «словам» певного значення, змісту (від давньогрецького sema – «знак, позначення» та semanticos – «означає»). Наприклад, записуючи математичну формулу, ми повинні дотримуватись певних синтаксичних правил (розстановка дужок та знаків операцій, тощо), але, крім цього, формула має визначений для нас зміст.

Мова – це засіб спілкування та передачі інформації. Для того, щоб нас розуміли слова мови повинні бути правильно побудовані та нести потрібний зміст – мати правильний синтаксис та семантику. Додамо, що і для опису семантики мови розроблені формальні системи, проте семантику формалізувати значно важче, ніж синтаксис.

Третій аспект – *прагматика* мови. Прагматика (від давньогрецького pragma –діло, дія та pragmateia – діяльність) пов'язана з цілями, які ставить перед собою носій мови, наприклад, отримання грошей за виступ. Прагматика пов'язана з соціально-психологічними аспектами мови та не входить в розгляд теорії, викладеної в даному посібнику.

Основними поняттями, що розглядаються в даному посібнику є

- формальні мови та породжуючі граматики,
- скінченні автомати та автомати з магазинною пам'яттю, їх роль в синтезі та аналізі мови
- класи граматик і мов, алгоритми синтезу та аналізу для різних класів мов
- процес трансляції формальних мов.

Розділ 1. МОВИ І ПОРОДЖУВАЛЬНІ ГРАМАТИКИ

1.1.ОСНОВНІ ПОНЯТТЯ ФОРМАЛЬНИХ МОВ

Розглянемо найпростіші поняття формальних мов.

Визначення 1.1.1.

Алфавіт – це скінченна непорожня множина неподільних елементів. Елементи алфавіту називаються **символами (буквами)**.

Визначення 1.1.2.

Словом (ланцюжком, рядком) в алфавіті Σ називається скінченна послідовність елементів з Σ. Слова будемо записувати у вигляді послідовності символів, наприклад, $baaa$ є словом в алфавіті $\Sigma = \{a, b, c\}$. Позначаються слова маленькими грецькими буквами, наприклад α, β, η.

Визначення 1.1.3.

Порожнє слово – це ланцюжок, який не містить жодного символа і позначається через ε. Порожнє слово не є символом, тобто $\varepsilon \notin \Sigma$ для будь-якого алфавіту Σ.

Довжина слова α дорівнює кількості символів у ньому і позначається $|\alpha|$. Якщо $\alpha = baaa$, то $|\alpha| = 4$. Довжина порожнього слова дорівнює нулю: $|\varepsilon| = 0$.

Визначення 1.1.4.

Конкатенацією слів або злиттям ланцюжків α і β називається ланцюжок $\alpha\beta$, що утворюється приписуванням слова β в кінець слова α. Наприклад, конкатенацією слів ca і bba є слово $cabba$. Вважається, що $\alpha\varepsilon = \varepsilon\alpha = \alpha$

Запис α^n означає $\underbrace{\alpha\alpha \ldots \alpha}_{n \text{ раз}}$, $\alpha^1 = \alpha$, $\alpha^0 = \varepsilon$. Наприклад, $0^2 1^3 = 00111$.

Через $|\alpha|_a$ позначимо кількість входжень букви a в слово α. Якщо $\Sigma = \{a, b, c\}$, то $|baaa|_a = 3, |baaa|_b = 1, |baaa|_c = 0$.

Множина всіх слів в алфавіті Σ позначається через Σ^*. Множина всіх непустих слів в алфавіті Σ позначається через Σ^+.

Визначення 1.1.5.

Мова над алфавітом Σ – це підмножина Σ^*, тобто це множина ланцюжків скінченної довжини у заданому скінченному алфавіті. Наприклад, множина $L = \{a, abb\}$ є мовою над алфавітом $\Sigma = \{a, b\}$

Оскільки кожна мова являється множиною, можна розглядати теоретико-множинні операції **об'єднання, перетину, різниці мов** (позначення $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$), заданих над одним і тим же алфавітом.

Визначення 1.1.6.

Нехай $L \subseteq \Sigma^*$. Тоді мова $\Sigma^* - L$ являється **доповненням** мови L відносно алфавіту Σ. Коли з контексту зрозуміло, про який алфавіт іде мова, говорять просто, що мова $\Sigma^* - L$ являється доповненням мови L.

Наприклад, для
$L_1 = \{ (ab)^n \mid n \geq 0 \}$ та $L_2 = \{ a^m b^m \mid m \geq 0 \}$
перетином $L_1 \cap L_2$ буде мова, що складається з єдиного ланцюжка ab.

Визначимо скільки слів містить мова $L_1 - L_2$, якщо вона задана над одним і тим же алфавітом $\Sigma = \{a, b, c\}$ і

$L_1 = \{w \in \Sigma^* \mid |w| = 4\}$ – *мова всіх слів дожини* 4
а мова L_2 складається зі слів з однією буквою c $L_2 = \{w \in \Sigma^* \mid |w|_c = 1\}$.

Різницею мов $L_1 - L_2$ є мова, що складається з довільних ланцюжків, складених з букв a, b, c довжиною 4, які або не містять входжень букви c або цих входжень є 2 та більше.

Зауважимо, що описати формально таку мову важко, тоді як використання операції різниці над мовами вирішує цю проблему.

1.2.ОПЕРАЦІЇ НАД МОВАМИ

Визначення 1.2.1.

Нехай $L_1, L_2 \subseteq \sum^*$. **Конкатенацією мов** L_1, L_2 є мова, що складається з усіх конкатенацій пар ланцюжків, перший з яких належить мові L_1, а другий – мові L_2

$$L_1 \cdot L_2 = \{x, y \,|x \in L_1, \qquad y \in L_2\}$$

Наприклад, якщо $L_1 = \{a, abb\}$ та $L_2 = \{bbc, c\}$, то

$$L_1 \cdot L_2 = \{ac, abbc, abbbbc\}.$$

Приклад 1.2.1.

Розглянемо наступну задачу:

при яких додатніх цілих числах k, l, m, n існують алфавіт Σ, мова $L_1 \subseteq \sum^*$ та мова $L_2 \subseteq \sum^*$, що задовольняють умовам

$$|\Sigma| = k,\ |L_1| = 1,\ |L_2| = m,\ |L_1 \cdot L_2| = n\ .$$

Зрозуміло, що за визначенням операції конкатенації мов, до мови $L_1 \cdot L_2$ належать ланцюжки, що починаються словом з мови L_1, а закінчуються словом з мови L_2.

Оскільки L_1 - множина, то з $|L_1| = 1$ випливає, що вона складається з одного слова ($|L_1|$ – потужність множини, кількість елементів в множині L_1), а мова L_2 складається з m слів. Тоді з'єнати одне слово з L_1 та m слів з L_2 можна m способами. Тому $m = n$.

Визначення 1.2.2.

Нехай $L \subseteq \Sigma^*$. Тоді **степенем** мови є

$$L^0 = \{\varepsilon\},$$

$$L^n = \underbrace{L \cdot L \cdot \ldots \cdot L}_{n-\text{разів}}\ \text{, якщо } n > 0.$$

Приклад 1.2.2.

Нехай $\Sigma = \{a, b\},\ L = (aa, ab\}$. Знайти L^3.
Визначимо спочатку $L^2 = L \cdot L =$$\{\ aaaa, aaab, abaa, abab\}$.
Далі

$$L^3 = L^2\ \cdot L =$$

$\{aaaaaa, aaaaab, aaabaa, aaabdb, abaaaa, abaaab, ababaa, ababab.$

Визначення 1.2.2.
Ітерацією мови (Kleene closure) *L* (позначення L^*) являється мова

$$\bigcup_{n \in N} L^n$$

Ця операція називається також **зірочкою Кліні** (Kleene star, star operation).

Приклад 1.2.3.

Якщо $\Sigma = \{a, b\},\ i\ L = (aa, ab, ba, bb\}$, то
$L^* = \{w\ \in\ \Sigma^*\ \ |\ |w|\ \vdots\ 2\}$, тобто є мовою з слів, що складається з довільної послідовності букв a, b , довжина яких кратна 2.

Приклад 1.2.4.

Нехай $\Sigma = \{a, b, c, d\}$ і
$L = \{w\ \in\ \Sigma^*\ \mid\ |w|_a = 1, |w|_c = 1.$

Чи вірно, що $abcdcacdcabbacba \in\ L^*$.

Ні, дане слово не буде належати L^*, оскільки при спробі розбити його на складові слова, з яких воно утворене застосуванням операції ітерації ми отримуємо слово, що не задовільняє умові входження однієї букви a та однієї букви c : наприклад

$$abcd\ \ ca\ \ \boldsymbol{cdca}\ bbac\ \ \boldsymbol{ba.}$$

Визначення 1.2.3.

Оберненим словом або **дзеркальним відображенням** слова *w* (позначається w^R) називається слово, в якому символи, які складають слово *w*, йдуть в зворотньому порядку.

Приклад 1.2.5.

Якщо $w = baaca$, то $w^R = acaab$.

Визначення 1.2.4.

Нехай $L \subseteq \Sigma^*$. Тоді

$$L^R = \{w^R \mid w \in L$$

Мова L^R являється **оберненою** до мови L.

Визначення 1.2. 5.

Кажуть, що слово *x* - **префікс** (**початок**) слова *y* (позначення $x \sqsubset y$), якщо $y = xu$ для деякого слова *u*.

Наприклад, $\varepsilon \sqsubset baa, b \sqsubset baa,\ ba \sqsubset baa, baa \sqsubset baa$.

Визначення 1.2.6.

Нехай $L \subseteq \Sigma^*$. Тоді через $pref(L)$ позначається множина, яка складається з всіх префіксів слів мови L:

$$pref(L) = \{x \mid (\exists\, y \in L), x \sqsubset y\}$$

Множина $pref(L)$ являється **множиною префіксів** мови L.

Визначення 1.2.7.

Говорять, що слово x - **суфікс** (**кінець**) слова y (позначення $x \sqsupset y$), якщо $y = ux$ для деякого слова *u*.

Визначення 1.2.8.

Нехай $L \subseteq \Sigma^*$. Тоді через $suf(L)$ позначається множина, що складається з усіх суфіксів слів мови L:

$$suf(L) = \{x \mid (\exists\, y\ \in L\,), x\ \sqsupset y\}$$

Множина $suf(L)$ являється **множиною суфіксів** мови *L*.

Визначення 1.2.9.

Кажуть, що слово *x* - **підслово** (substring) слова y, якщо $y = uxv$ для деяких слів *u* та *v*.

Визначення 1.2.10.

Нехай $L \subseteq \Sigma^*$. Тоді через Subw(L) позначається множина, що складається з усіх підслів слів мови L. Множина Subw(L) являється **множиною підслів** мови L.

Визначення 1.2.11.

Слово $a_1\, a_2\ \ldots\ a_n$ (довжини $n \geq 0$) являється **підпослідовністю** (subsequence) слова *y*, якщо існують такі слова $u_0\,, u_1, u_2\ , \ldots\ , u_n$, що

$$u_0\, a_1 u_1\, a_2\ \ldots\ a_n\, u_n - \ y.$$

Зауваження. Всі підслова слова *y* являються також підпослідовністю слова *y*.

Визначення 1.2.12.

Нехай $L \subseteq \Sigma^*$. Тоді через Subseq(L) позначається множина, що складається з усіх підпослідовностей слів мови L. Множина Subseq(L) являється **множиною підпослідовностей** мови *L*.

Приклад 1.2.6.

Розглянемо мову $L = \{cba, c\}$ над алфавітом $\{a, b, c\}$. Очевидно, що

$$Subseq(L) = \{\varepsilon, a, b, c, ba, ca, cb, cba\}.$$

1.3. ПОРОДЖУВАЛЬНІ ГРАМАТИКИ

Мови можуть бути задані різними способами. Один з них – задати всі слова мови. Інший – означити критерій, якому повинні задовольняти слова, щоб належати мові.

Розглянемо ще один важливий спосіб задати мову – через використання породжувальних граматик.

Визначення 1.3.1.

Формальна породжувальна граматика G – це формальна система, визначена четвіркою об‘єктів

$G = (T, N, S, P)$,

де T – скінченна множина термінальних символів;

N – скінченна множина нетермінальних символів, $N \cap T = \varnothing$;

P – скінченна множина продукцій або правил підстановки;

S – початковий символ, $S \in N$.

Граматика складається з множини символів різного типу (термінальних і нетермінальних) та множини правил побудови ланцюжків мови.

Термінальні символи використовуються для побудови слів мови. Вони не можуть бути замінені іншими символами. Це означає, що множина термінальних символів складає алфавіт цієї мови.

Нетермінальні символи використовуються як допоміжні (змінні) символи у процедурах побудови рядків мови.

Для позначення термінальних символів будемо використовувати малі латинські букви та цифри, нетермінальні символи позначатимемо великими латинськими буквами.

Визначення 1.3.2.

Продукціями граматики називаються правила виду $\alpha \to \beta$, які визначають, що ми можемо замінити ланцюжок α іншим ланцюжком β. Кожна продукція з P повинна містити принаймні один нетермінальний елемент у лівій частині.

Породження ланцюжків у граматиках починається з правила, що містить початковий символ S (називають **аксіомою**) у лівій частині.

Приклад 1.3.1.

Нехай дано множини

$N = \{A, B, S\}$,

$T = \{a, b\}$,

$P = \{S \to ABa,\ A \to BB,\ B \to ab,\ Ba \to ba\}$.

Тоді $G = (T, N, P, S)$ - породжувальна граматика.

Як вже зазначалося, класична теорія формальних мов вивчає насамперед синтаксис мови. Вона вводить математичну модель синтаксису, яка описує механізми породження і розпізнавання "правильно побудованих" ланцюжків. У цьому розділі ми розглядаємо перший з цих механізмів механізм породження. Задати такий механізм - значить описати якусь процедуру, що дозволяє вивести (або, як зазвичай говорять, породити) довільний ланцюжок мови згідно з визначенням скінченної множини правил. Останнє дуже важливо: мова може бути нескінченною, але множина правил, за допомогою яких виводяться її ланцюжки, обов'язково скінчена. Зауважимо, що не кожна мову може бути представлена таким чином, але ми в рамках цього підручника будемо розглядати тільки такі мови, тобто мови, синтаксис яких задається скінченною множиною правил. Ці мови, взагалі кажучи нескінченні, але допускають скінченний опис, їх називають зчисленними мовами.

Класичним способом визначення мов через породжувальну процедуру є визначення їх за допомогою породжувальних граматик, або граматик Хомського.

Породжувальна граматика дозволяє виводити (породжувати) ланцюжки мови з деякого початкового ланцюжка за допомогою певних правил заміни (або правил підстановки). Породження є покроковий процес, в якому на кожному кроці з ланцюжка, вже отриманого на попередніх кроках (зокрема, з початкового), можна шляхом застосування до нього правил заміни отримати новий ланцюжок.

Визначення граматики вказує поки тільки на те, що слід фіксувати, щоб задати якусь граматику. А саме слід фіксувати два алфавіти, що не перетинаються, виділивши в одному з них символ, названий аксіомою, а також скінченну множину правил виводу. Але ми ще не знаємо, яким чином "працює" граматика як інструмент породження (і перетворення) слів. Щоб це зрозуміти, потрібно визначити найважливіше поняття виводимості в даній граматиці.

Неформально кожне правило виводу граматики трактується як "правило заміни", що дозволяє довільне входження лівої частини правила в деякий ланцюжок в об'єднаному нетермінальному та термінальному алфавіті замінити його правою частиною, отримавши (або вивівши) тим самим новий ланцюжок.

Приклад 1.3.2.

Розглянута нижче граматика моделює найпростіший фрагмент природньої (української) мови: її термінали - це деякі слова української мови, нетерміналами є "граматичні категорії", а саме: поняття "речення", "підмет", "присудок" (вони задані як слова, вкладені в кутові дужки):

G_1 = (T={*кіт, пес, крокодил, м'явкає, гавкає, плаче*},
N={ < *речення*>, <*підмет*>, <*присудок*>},
S={< *речення*>}, P).

Множина правил P має вигляд

< *речення*> → <*підмет*> <*присудок*>,
<підмет> → *кіт* | *пес* | *крокодил*,
<присудок> → *м'явкає* | *гавкає* | *плаче*.

Тут знак | використовується для переліку правих частин правил виводу, у яких ліва частина правил однакова.

Кожне правило виведення граматики можна трактувати як визначення тієї чи іншої граматичної категорії: наприклад, перше правило є запис такого визначення: "речення - це підмет, за яким йде присудок". Оскільки нас цікавить

тільки синтаксис мови, то це визначення є вимогою того, що записуючи речення, спочатку потрібно поставити підмет, а після нього - присудок. Інші правила граматики визначають аналогічним чином "підмет" та "присудок". Але тут нових граматичних категорій не виникає - просто перераховуються ті слова, які можуть бути підметом і присудком.

Побудуємо яке-небудь виведення в граматиці G:

< речення> $\Rightarrow$ <підмет> <присудок> $\Rightarrow$ *кіт* <присудок> $\Rightarrow$ кіт *гавкає*.

Зауважимо, що "зміст" (семантика) виведеного ланцюжка нас ніяк не цікавить. Ми взагалі поки не знаємо, що таке "зміст". Так що кіт може гавкати, крокодил м‘явкати і т.д.

Наведений приклад, незважаючи на його простоту, дозволяє нам дати ще одну змістовну інтерпретацію поняття граматики. Граматику можна розглядати як "систему визначень деяких термінів, понять". Виділяється саме загальне поняття (в даному прикладі поняття "речення", воно зводиться до менш загального "поняття" доти, поки ми не прийдемо до "конкретних об'єктів" (в даному випадку до ланцюжків в якомусь термінальному алфавіті), які підпадають під визначення "конкретне поняття". Найбільш загальне "поняття" - це аксіома, інші "поняття" - нетермінали, "конкретні об'екти" - термінальні ланцюжки.

Нехай дано граматику $G=(N,T,S,P)$, і нехай $\alpha = \sigma\xi\tau$, $\xi \neq \eta$ та $\beta = \sigma\eta\tau$ – ланцюжки над алфавітом $N \cup T$.

Якщо $\xi \neq \eta$ є продукцією граматики *G*, то кажуть, що β **безпосередньо виводиться** з α і записують

$$\alpha \Rightarrow \beta .$$

Якщо $\alpha_0, \alpha_1, ..., \alpha_n$ – ланцюжки над алфавітом такі, що $\alpha_0 \Rightarrow \alpha_1 \Rightarrow ... \Rightarrow \alpha_n$, то кажуть, що ланцюжок α_n **виводиться з ланцюжка** α_0 або α_0 **породжує** α_n та використовують запис $\alpha_0 \overset{n}{\Rightarrow} \alpha_n$ (інколи позначають $\alpha_0 \overset{*}{\Rightarrow} \alpha_n$, якщо кількість застосованих правил виведення не суттєва).

Визначення 1.3.3.

Послідовність кроків для отримання α_n з α_0 називається **виводом**. Число *n* називають **довжиною (кількістю кроків)** цього виводу.

Приклад 1.3.3.

Ланцюжок $Aaba$ безпосередньо виводиться з ABa ($ABa \Rightarrow Aaba$) у граматиці з прикладу 1.3.1, оскільки $Aaba$ отримуємо, якщо згідно продукції граматики $B \to ab$ в ланцюжку ABa замість B підставити ab.

Ланцюжок $ababaaba$ породжується ланцюжком ABa, оскільки

$$ABa \Rightarrow Aaba \Rightarrow BBaba \Rightarrow Bababa \Rightarrow abababa$$

з допомогою застосування послідовно наступних продукцій

$$B \to ab, A \to BB, B \to ab, B \to ab .$$

Визначення 1.3.4.

Мовою $L(G)$, що породжується граматикою *G*, називають множину всіх ланцюжків термінальних символів, які виводяться з початкового символу *S*, тобто

$$L(G) = \left\{ \omega \in T^* \;\middle|\; S \overset{*}{\Rightarrow} \omega \right\}.$$

Приклад 1.3.4.

Знайти мову, яка породжується граматикою

$G_2 = (N = \{S, A\},$

$\mathrm{T} = \{a, b\},$

$\mathrm{P} = \{ S \to aA,$

$S \to b,$

$A \to aa\},$

аксіома граматики S).

Для цього запишемо вивід всіх можливих ланцюжків з початкового символа S, послідовно заміняючи кожен нетермінал з допомогою існуючих продукцій, поки не отримаємо ланцюжок з терміналів:

$S \Rightarrow aA \Rightarrow aaa$;

$S \Rightarrow b$.

Отже $L(G) = \{aaa, b\}$.

Приклад 1.3.5. Знайти мову, породжену граматикою $G_3 = (\mathrm{T} = \{0,1\},\ N = \{S\},\ \mathrm{P} = \{S \to 11S, S \to 0\}, S)$. Запишемо виведення ланцюжків.

$S \Rightarrow 0$;

$S \Rightarrow 11S \Rightarrow 110$;

$S \Rightarrow 11S \Rightarrow 1111\ S \Rightarrow 11110$.

Як бачимо, після кожного кроку виводу ми або додаємо дві одиниці в кінець ланцюжка або закінчуємо ланцюжок нулем. Тобто $L(G) = \{0,110,11110,1111110,...\} = \{1^{2n}0 \mid n = 0,1,2,...\}$ – це множина всіх ланцюжків з парною кількістю 1, після яких (у кінці) один 0.

Зауваження.

Ми отримали нескінченну мову ($L(G)$ складається з нескінченної кількості ланцюжків). Щоб граматика G породжувала нескінченну мову, в множині продукцій повинно бути принаймні одне рекурсивне правило (у прикладі 4 це правило $S \to 11S$) або рекурсивний вивід, який через певну

множину кроків породжує в ланцюжку повторну появу того ж нетерміналу або того ж підланцюжка.

Важливою є проблема побудови граматики для заданої мови.

Приклад 1.3.6.

Знайти граматику, яка породжує мову $L(G) = \{0^n 1^n \mid n = 0,1,2,...\}$.

Слова цієї мови складаються з однакової кількості нулів, за яким слідує така ж кількість одиниць. Запишем продукцію, що додає один 0 на початок і одну 1 у кінець ланцюжка. Друга продукція замінює S на порожній ланцюжок ε . Розв'язком є граматика

$G_4 = (T, N, P, S)$,
$T = \{0,1\}$,
$N = \{S\}$,
$P = \{S \to 0S1, S \to \varepsilon\}$, S – початковий символ.

Приклад 1.3.7.

Знайти граматику, яка породжує мову $L(G) = \{0^n 1^m \mid n, m = 0,1,2,...\}$.

Це нескінченна мова. Слова цієї мови складаються з довільної кількості нулів, за якими є довільна кількість одиниць, причому кількість нулів в загальному випадку не співпадає з кількістю одиниць. Таких граматик наведемо дві:

$G_5 = (\{S\}, \{0,1\},$
$\{S \to 0S, S \to S1, S \to \varepsilon\}, S)$

$G_6 = (\{S, A\}, \{0,1\},$
$\{S \to 0S, S \to 1A, S \to 1, A \to 1A, A \to 1, S \to \varepsilon\}, S)$.

Цей приклад свідчить, що дві різні граматики можуть породжувати одну мову.

Приклад 1.3.8.

Граматика

$$G_7 = (\{a, b, 0,1\}, \{I, D\}, P_2, I)$$

має множину правил P_2:

$$I \to aD \mid bD \mid a \mid b \,,$$

$$D \to aD \mid bD \mid 0D \mid 1D \mid a \mid b \mid 0 \mid 1$$

Неважко побачити, що дана граматика породжує найпростіші "ідентифікатори" в алфавіті, що складається з двох "букв" a, b і двох "цифр" $0,1$, тобто ланцюжки, що починаються обов'язково з "букви", за якою слідують довільні "букви" та "цифри" . Наведемо виводи в граматиці G_7:

$$I \Rightarrow aD \Rightarrow abD \Rightarrow ab0D \Rightarrow ab0bD \Rightarrow ab0b1$$

Приклад 1.3.9.

Граматика

$G_8 = (T = \{\,(\,,\,)\,,\ N = \{S\},\ P = \{S \to ()|(\,S) \mid SS\}\,, S)$

породжує множину так званих "правильних дужкових структур", наприклад

$$S \Rightarrow \ \mathrm{SS} \ \Rightarrow \ (S)S \Rightarrow \ (())S \Rightarrow \ (())()$$

Корисно зіставити з цією граматикою індуктивне визначення множини правильних дужкових структур: ланцюжок () є правильною дужковою структурою; якщо відомо, що x і y - правильні дужкові структури, то ланцюжок xy також вважаємо правильною дужковою структурою; якщо відомо, що x - правильна дужкова структура, то і ланцюжок (x) є правильною дужковою структурою; ніяких правильних дужкових структур, крім визначених вище, не

існує.

Видно, що граматика G_8 є не чим іншим, як формою запису сформульованого тільки що індуктивного визначення "правильна дужкова структура" (поняття, позначене аксіомою S) є або ланцюжок (), або "правильна дужкова структура" в дужках - (S), або дві "правильні дужкові структури", записані одна після одної - SS.

Приклад 1.3.10.

Розглянемо граматику

$$G_9 = (\{a, b, c\}, \quad \{S, A, B, C\}, \quad P_9, \quad S)$$

з множиною правил виводу P_9:

Правила 1,2	$S \to aSBC \mid abC$
Правило 3	$CB \to BC$
Правило 4	$bB \to bb$
Правило 5	$bC \to bc$
Правило 6	$cC \to cc$

Розберемо деякі приклади виводів: під символом $\Rightarrow$ будемо вказувати номер застосовуваного на даному кроці правила, а над символом - кількість повторень цього правила:

$S \underset{2}{\Rightarrow} abC \underset{5}{\Rightarrow} abc$;

$S \underset{1}{\Rightarrow} aSBC \underset{2}{\Rightarrow} aabCBC \underset{3}{\Rightarrow} aabBCC \underset{4}{\Rightarrow} aabbCC \underset{5}{\Rightarrow} aabbcC \underset{6}{\Rightarrow} aabbcc$;

$S \underset{1}{\Rightarrow} aSBC \underset{1}{\Rightarrow} aaSBCBC \underset{2}{\Rightarrow} aaabCBCBC \underset{3}{\Rightarrow} aaabBCCBC \underset{3}{\Rightarrow} aaabBCBCC \underset{3}{\Rightarrow}$

$\underset{3}{\Rightarrow} aaabBBCCC \underset{4}{\Rightarrow} aaabbbCCC \underset{5}{\Rightarrow} aaabbbcCC \underset{6}{\Rightarrow}^{2} aaabbbccc$;

Представимо тепер вивід довільного ланцюжка $a^n b^n c^n$

$$S \underset{1}{\Rightarrow}^{n-1} a^{n-1}S(BC)^{n-1} \underset{2}{\Rightarrow} a^{n-1}abC\underbrace{BCBC\ldots BC}_{n-1} \underset{3}{\Rightarrow} a^n bBCC\underbrace{BCBC\ldots BC}_{n-2} \underset{3}{\Rightarrow} \ldots \underset{3}{\Rightarrow}$$

$$\underset{3}{\Rightarrow} a^n bB^{n-1}C^n \underset{4}{\Rightarrow} a^n bb(B)^{n-2}C^n \underset{4}{\Rightarrow} \ldots \underset{4}{\Rightarrow} a^n b^n C^n \underset{5}{\Rightarrow} a^n b^n cC^{n-1} \underset{6}{\Rightarrow}^{n-1} a^n b^n c^n \quad (1)$$

Можна помітити, що за допомогою багаторазового застосування правила 3 у виводі (1) відбувається "перегонка" всіх букв В вліво від всіх букв С, тобто з ланцюжка

$$a^n bC\underbrace{BCBC\ldots BC}_{n-1}$$ виводиться ланцюжок $a^n bB^{n-1}C^n$.

Якщо вважати, що правило (3) на кожному кроці відповідного виводу застосовується так, що відбувається заміна першого входження ланцюжка СВ ланцюжком ВС, то перший (самий лівий) символ В в ланцюжку $a^n bC\underbrace{BCBC\ldots BC}_{n-1}$ "проходить" через один символ С, наступному символу В в ланцюжку $a^n bBCC\underbrace{BCBC\ldots BC}_{n-2}$ потрібно пройти вже два символи С і т.д. Звідси випливає, що правило (3) у вказаному фрагменті виводу (1) застосовується

$$1 + 2 + 3 + \cdots + n - 1 = \frac{n(n-1)}{2}$$ разів.

Тоді послідовність номерів застосовуваних правил (протокол виводу) може бути записана так:

$$(1)^{n-1}(2)(3)^{\frac{n(n-1)}{2}}4^{n-1}(5)6^{n-1}, n \geq 1$$

Набагато важче довести, що дана граматика породжує тільки ланцюжки зазначеного виду. Для доведення цього факту, проаналізуємо інші варіанти проведення виводу після застосування правила (2) у виводі (1).

1. Після застосування правила (2) застосовуємо правило (5) і після багаторазового застосування правила (3) отримуємо

$$a^n bC(BC)^{n-1} \underset{5}{\Rightarrow} a^n bc(BC)^{n-1} = a^n bc\underbrace{BCBC\ldots BC}_{n-1} \underset{3}{\Rightarrow} \ldots \underset{3}{\Rightarrow} a^n bc\, B^{n-1}C^{n-1} \quad (2)$$

Ланцюжок, записаний останнім, такий, що, по-перше, він не є термінальним, а по-друге, до нього не застосовується жодне правило граматики. Такий ланцюжок називається тупиковим.

2. Вивід (1) можна модифікувати, перервавши на певному кроці застосування правила (3) та застосувати правило (4):

$$a^n bB^m C^{m+1}(BC)^{n-1-m} \underset{4}{\Rightarrow}$$
$$a^n bbB^{m-1}C^{m+1}(BC)^{n-1-m} = a^n bbB^{m-1}C^m C\underbrace{BCBC\ldots BC}_{n-1-m}$$

при цьому $m < n-1$/

Подальше застосування правила (3) СВ → ВС "переганяє" всі символи В вліво, що дозволить отримати ланцюжок $a^n bb\,B^{n-2}C^n$, з якого легко виводиться ланцюжок $a^n b^n c^n$. Таким чином, ми отримали інший варіант виведення цього ланцюжка.

Але якщо ми будемо застосовувати правило (4), починаючи з ланцюжка

$$a^n bB^m C^{m+1}(BC)^{n-1-m},$$

до тих пір, поки це можливо, то отримаємо ланцюжок

$$a^n b^{m+1}C^{m+1}(BC)^{n-1-m}$$

З нього, якщо не застосовувати відразу правило (5), виводиться ланцюжок $a^n b^n c^n$. Застосування ж правила (5) призведе до тупикового ланцюжка.

За аналогією можна розглянути довільне чергування застосування правил (3) і (4).

Всі варіанти виведення термінального ланцюжка тим самим ми розібрали.

У загальному ж випадку строге доведення того, що вибрана нами певна граматика не породжує ніяких інших ланцюжків, крім ланцюжків певного виду, може бути нетривіальним.

Визначення 1.3.5.
Дві **граматики еквівалентні**, якщо вони породжують одну і ту ж мову.

Наприклад, граматика

$$G_{10} = (\{a, b, \varepsilon\}, \{S\}, S, P = \{S \rightarrow abS,\ \ S \rightarrow a\})$$

і граматика

$$G_{11} = (\{a, b, \varepsilon\}, \{T, U\}, T, P = \{T \rightarrow aU,\ \ U \rightarrow baU,\ \ U \rightarrow \varepsilon\})$$

еквівалентні, оскільки породжують одну і ту ж мову

$$L(G_{10}) = (ab)^n a = L(G_{11}) = a\,(ba)^n, \qquad n \geq 0$$

1.4.КЛАСИ ГРАМАТИК ТА ФОРМАЛЬНИХ МОВ.

Граматики можна класифікувати за виглядом їх продукцій. Така класифікація називається ієрархією Хомського.

Нагадаємо, що єдине обмеження, що накладається на правила виводу в будь-яких граматиках, полягає в тому, що в ліву частину правила повинен входити хоча б один нетермінал. Залежно від додаткових обмежень, які накладаються на правила виводу граматики, розрізняють наступні основні класи граматик.

1. Граматики ***типу 0***, або граматики ***загального виду.***

   Тут на правила виводу не накладається жодних обмежень.

2. Граматики ***типу 1***, або ***невкорочувальні*** граматики.
   Кожне правило такої граматики має вигляд

$$\alpha \rightarrow \beta \text{ , де } |\alpha| \leq |\beta| \text{ або } \alpha \rightarrow \varepsilon \text{ ,}$$

де ε - порожній ланцюжок. Таким чином, довжина правої частини правила не менше довжини лівої.

Граматика G_9 із прикладу 1.3.10 є невкорочувальною граматикою.

3. ***Контекстно-залежні граматики*** (КЗ-граматики). Граматику називають контекстно-залежною граматикою (КЗ-граматикою), якщо будь-яке її правило виводу має вигляд

$$l\,\boldsymbol{A}\,r \;\rightarrow l\,\boldsymbol{w_2}\,r,$$

при цьому хоча б один з ланцюжків l, r відмінний від ε, тобто нетермінал A може бути замінений ланцюжком w_2 лише в оточенні l та r, а отже - у відповідному контексті, звідси і назва – контексно-залежні. Іноді ланцюжок l називають лівим контекстом, а ланцюжок r - правим контекстом даного КЗ- правила. З визначення видно, що кожна КЗ-граматика є невкорочувальною.

Граматика G_9 з прикладу 1.3.10 не є КЗ-граматикою, оскільки правила

$$CB \rightarrow BC, \qquad S \rightarrow aSBC \mid abC,$$

не є контексно-залежними, хоча всі решта правила

$$bB \rightarrow bb,\ bC \rightarrow bc, cC \rightarrow cc$$

є контекстно-залежними.

4. ***Контекстно-вільні граматики*** (КВ-граматики). Кожне правило такої граматики має вигляд

$$A \rightarrow \beta \text{, де } A \in N,\ \beta \in (N \cup T)^*,$$

тобто ліва частина кожного правила виводу складається з єдиного нетерміналу, а права є довільним (може бути і порожнім) ланцюжком в об'єднаному нетермінальному та термінальному алфавіті.

З практичної точки зору це найбільш важливий клас граматик, оскільки саме в термінах КВ-граматик описується синтаксис мов програмування, тому цим граматикам буде присвячено окремий розділ посібника.

Граматика G_8 правильних дужкових виразів з прикладу 1.3.9 є КВ-граматикою.

5. ***Лінійні граматики***. Кожне правило такої граматики має вигляд

$$A \to u \text{ або } A \to uBv \text{ , де } A, B \in N \text{ , } u, v \in T^* ,$$

тобто в правій частині правила може міститися не більше одного входження нетерміналу.

Граматика називається **праволінійною**, якщо кожна продукція з P має вигляд

$$A \to u \text{ або } A \to uB \text{ , де } A, B \in N \text{ , } u \in T^* ,$$

відповідно **ліволінійною,** якщо кожна продукція з P має вигляд

$$A \to u \text{ або } A \to Bu .$$

Приклад 1.4.1. Лінійною є граматика

$$G_{12} = V = \{a_1, a_2, \dots , a_n\}, \{S\}, S ,$$
$$P = \{ S \to a_1 S a_1 \mid a_2 S a_2 \mid \dots \mid a_n S a_n \mid a_1 \mid a_2 \mid \dots \mid a_n \mid \varepsilon \}$$

Можна довести, що ця граматика породжує всі паліндроми в алфавіті V, тобто всі ланцюжки, що читаються зліва направо так само, як і справа наліво. Наприклад, для $V = \{a, b, c\}$ ланцюжок $acbbca$ - паліндром. Вивід його в граматиці G_{12} для даного термінального алфавіту $V = \{a, b, c\}$ буде мати вигляд

$$S \Rightarrow aSa \Rightarrow\ acSca \Rightarrow\ acbSbca \Rightarrow\ acb\varepsilon bca \Rightarrow acbbca$$

6. ***Регулярні граматики***. У такої граматики кожне правило має вигляд

$$A \to a \text{ або } A \to aB \text{ , де } A, B \in N \text{ , } a \in T,$$

де a є або термінал, або порожній ланцюжок ε, на відміну від право лінійних граматик, де u – термінальний ланцюжок, довжиною $n \geq 1$. Регулярні граматики - окремий випадок праволінійних граматик, u - один термінальний символ .

Наведемо без доведення деякі твердження про класи граматик.

Теорема 1.4.1.

1. Для будь-якої невкорочувальної граматики може бути побудована еквівалентна їй КЗ-граматика.
2. Для будь-якої ліволінійної граматики може бути побудована еквівалентна їй праволінійна граматика, і, навпаки, для будь-праволінійної граматики може бути побудована еквівалентна їй ліволінійна граматика.
3. Для будь праволінійної граматики може бути побудована еквівалентна їй регулярна граматика.

Відзначимо, що доведення першого пункту теореми 1.4.1 вельми нетривіальне, а пункти 2 та 3 будуть доведені в наступних розділах.

Між класами граматик існує наступне вкладення:

регулярні граматики (РЕГ)	$\subset$	праволінійні граматики
праволінійні граматики	$\subset$	лінійні граматики
ліволінійні граматики	$\subset$	лінійні граматики
лінійні граматики (ЛІН)	$\subset$	контексно-вільні граматики
контексно-вільні граматики (КВ)	$\subset$	контексно-залежні граматики
контексно-залежні граматики (КЗ)	=	невкорочувальні граматики
контексно-залежні граматики	$\subset$	граматики загального типу

Тобто, будь-яка праволінійна граматика є лінійною. Будь-яка лінійна граматика є контекстно-вільною. Будь-яка контекстно-вільна граматика є контекстно-залежною і т.д..

РЕГ ⊂ Праволінійні ⊂ ЛІН ⊂ КВ ⊂ КЗ = невкорочувальні ⊂ типу 0

Класифікація мов, породжуваних граматиками, тісно пов'язана з класифікацією самих граматик, хоча й не тотожна їм.

Мову відносять до класу С, якщо існує граматика класу С, що породжує дану мову.

Таким чином визначаються мови типу 0, невкорочувальні мови, контекстно-залежні мови (КЗ-мови), контекстно-вільні мови {КВ-мови), лінійні мови, право- і ліволінійні мови, регулярні мови.

Принципова відмінність між класифікацією граматик і мов полягає в наступному. Щоб визначити клас граматики, досить *подивитися на множину її правил виводу.* Щоб довести твердження про те, що дана мова є мова такого-то класу, досить *придумати будь-яку граматику з відповідного класу*, яка його породжує. Але щоб довести протилежне твердження, *що мова не належить певному класу мов, потрібно довести, що не існує граматики відповідного класу, яка його породжує.* Це завдання набагато важче. Деякі методи побудови подібних доказів будуть розглянуті далі.

Мова називається **контекстно залежною**, якщо існує принаймні одна контекстно залежна граматика, яка породжує цю мову.

Мова називається **контекстно вільною**, якщо існує принаймні одна контекстно вільна граматика, яка породжує цю мову.

І, нарешті, мова називається **регулярною (лінійною)**, якщо існує принаймі одна регулярна граматика, яка породжує цю мову.

Граматика прикладу 1.3.1 є типу 1, оскільки всі продукції задовольняють умові $|\alpha| \le |\beta|$. Вона не є типу 2, оскільки останнє правило зліва має два елементи - нетермінальний і термінальний. Відповідно мова є КЗ.

Граматика і відповідно мова у прикладах 1.3.3, 1.3.4 є праволінійна. Граматика і мова прикладу 1.3.5 є типу 2, контекстно-вільна.

Вивід у мовах, породжених контекстно-вільними граматиками, може зображатися графічно з використанням орієнтованих кореневих дерев. Ці дерева називають **деревами виводу** або синтаксичного розбору.

Кореню цього дерева відповідає початковий символ. Внутрішнім вершинам відповідають нетермінальні символи, що зустрічаються у виведенні. Листям відповідають термінальні символи. Виведений ланцюжок складається з виписаних зліва на право міток листків.

Приклад 1.4.2.

Визначити, чи слово $cbab$ належить мові, породженій граматикою $G = (\{S, A, B, C\}, \{a, b, c\}, \{S \to AB, A \to Ca, B \to Ba, B \to Cb, B \to b,\ C \to cb, C \to b\}, S)$.

Попробуємо вивести цей ланцюжок: $S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB \Rightarrow cbab$.

Отже, слово $cbab$ належить мові $L(G)$. Дерево виведення для заданого рядка $cbab$ зображено на рис.1.4.1.

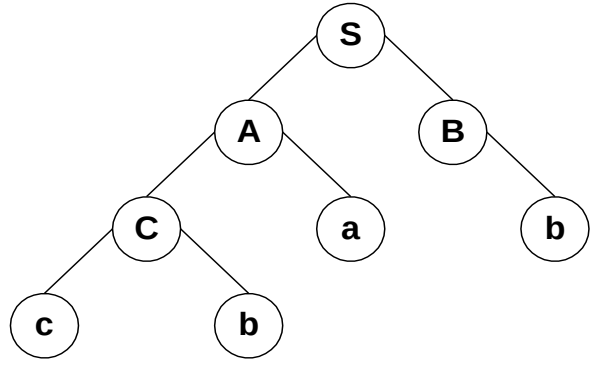


Рис. 1.4.1. Дерево виводу для рядка $cbab$.

Вивід у контекстно-вільній граматиці називається **лівостороннім**, якщо на кожному кроці виводу замінюється перший зліва нетермінал.

Існує взаємно однозначна відповідність між лівостороннім виводом слова α у граматиці *G* та його деревом виводу.

Граматика називається **неоднозначною**, якщо існує слово, яке має два або більше лівосторонніх виводи.

КОНТРОЛЬНІ ЗАПИТАННЯ ДО РОЗДІЛУ 1.

1. Дайте визначення алфавіту, ланцюжка (слова).
2. Що таке порожній ланцюжок?
3. Дайте визначення формальної породжуючої граматики.
4. Що таке вивід в граматиці? Довжина виводу?
5. Дайте визначення мови, породженої граматикою.
6. Визначити ієрархію Хомського для граматик.
7. Класифікація формальних мов. Чим відрізняються підходи при визначенні класу граматики та класу формальної мови.
8. Дайте визначення наступним поняттям: лівосторонній та правосторонній виводи, дерево виводу, однозначність граматик

ВПРАВИ ДО РОЗДІЛУ 1.

Вправа 1. Нехай $\Sigma = \{a, b, c\}$. Чи рівні мови
$\{ (abc)^n \mid n \geq 2 \}$ та $\{ ab(cab)^n ca \mid n \geq 1 \}$?

Вправа 2.
Знайти праволінійну граматику, еквівалентну граматиці

$S \to aSb$ $\quad$ $K \to aK$
$S \to K$ $\quad$ $J \to Jb$
$S \to J$ $\quad$ $K \to a$
$\quad$ $\quad$ $J \to \varepsilon$

Вправа 3.
Чи являється граматика контекстно-залежною? Контекстно-вільною?

$S \to ASTA$ $\quad$ $AT \to UT$
$S \to AbA$ $\quad$ $UT \to UV$
$A \to a$ $\quad$ $UV \to TV$
$bT \to bb$ $\quad$ $TV \to TA$

Вправа 4.
До якого класу належить граматика

$$S \to TS \qquad A \to a$$
$$S \to US \qquad TA \to AAT$$
$$S \to b \qquad UAb \to b$$
$$Tb \to Ab \qquad UAAA \to AAU$$

Вправа 5.

Нехай $A = \{ab, c\}$ і $B = \{c, ca\}$ - дві формальні мови над алфавітом $\{a, b, c\}$. Знайти наступні формальні мови:

1. $A \cup B$
2. $A \setminus B$
3. A^2
4. $A^2 \setminus B^2$
5. $A \cdot B$

Вправа 6.

1. Чи належить ланцюжок $x = abaababb$ мові, що породжується граматикою з правилами

$$S \to SaSb \mid \varepsilon$$

Якщо так, то побудувати його дерево виводу

2. Чи належить ланцюжок x = (()())() мові, що породжується граматикою з правилами:

$$S \to SA \mid A$$
$$A \to (S) \mid ()$$

Якщо так, то побудувати його дерево виводу

3. Чи належить ланцюжок x = 00011011 мові, що породжується граматикою з правилами:

$$S \to SS \mid A$$
$$A \to 0A1 \mid S \mid 01$$

Якщо так, то побудувати його дерево виводу

4. Чи належить ланцюжок x = 0111000 мові, що породжується граматикою з правилами:

$$S \to A0B \mid B1A$$
$$A \to BB \mid 0$$
$$A \to AA \mid 1$$

Якщо так, то побудувати його дерево виводу

Вправа 7.

Нехай G - граматика з правилами:

$$S \to CD \qquad C \to aCA \mid bCB \mid \varepsilon \qquad AD \to aD$$
$$BD \to bD \qquad Aa \to aA \qquad Ab \to bA$$
$$Ba \to aB \qquad Bb \to bB \qquad D \to \varepsilon$$

Показати, що $L(G) = \{xx \mid x \in \{a,b\}^*\}$.

Вправа 8.

Показати, що граматика G неоднозначна.

$$G: S \to abC \mid aB \qquad B \to bc \qquad bC \to bc$$

Вправа 9.

Чи еквівалентні граматики з правилами

$$G1: A \to AB \qquad B \to bc \qquad A \to aAc \mid Sa \qquad C \to c \mid Ca$$
$$G2: S \to AB \qquad AB \to aDB \qquad DB \to ABB \qquad B \to b \qquad C \to c$$
$$G3: S \to AS \mid Bc \qquad B \to Ac \mid cS \qquad A \to Bd \qquad C \to c$$

Вправа 10.

Нехай грамматика G визначається правилами

$$G: S \to AB \quad AB \to aDB \quad DB \to ABB \quad B \to b \quad Ab \to b$$

Якому классу (по Хомському) вона належить? Чи породжується L(G) граматикою більш вузького класу?

Вправа 11.

Нехай грамматика G визначається правилами

$$G: S \to AaB \quad AaB \to aAaBb \quad aBb \to abb \quad A \to \varepsilon$$

Якому классу (по Хомському) вона належить? Чи породжується L(G) граматикою більш вузького класу?

Вправа 12.
Визначити КВ-граматики, які б породжували наступні мови:
1) всі рядки - елементи множини {0, 1}* такі, що в кожному з них безпосередньо справа від кожного символу 0 стоїть символ 1.
2) всі рядки - елементи множини {0, 1}*, які містять символів 0 вдвоє більше, ніж символів 1 ;
3) всі рядки - елементи множини {0, 1}*, які містять парне число символів 0 і непарне число символів 1 ;
6) рядки - елементи множини {0, 1}*, в яких дужки розставлені правильно.

Вправа 13.
Побудувати КВ-граматику, яка породжує мову

$a)$ $\{a^n\, c\, b^n\} \cup \{b^n\, a\, a^n\}: n \geq 0$

$b)$ $\{x \mid x \in \{a,b\}^* \setminus \varepsilon: \quad x \neq y\, y^R\}$, де $y^R -$ дзеркальне відображення y

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Нехай $G = (T, N, P, S)$ - формальна породжуюча граматика.

1. Визначити тип граматики за Хомським.
2. Визначити мову заданої граматики G.
3. Вивести ланцюжок α. Намалювати дерево виведення.
4. Побудувати граматику для заданої мови $L(G)$

Варіанти завдань до лабораторної роботи.

№ вар	Граматика *G*	α	*L(G)*
1	$P = \{S \to 00S, S \to 11\}$	$0^4 1^2$	$L(G) = \{2^{3n}0, n = 0,1,...\}$
2	$P = \{S \to 0S, A \to A1, S \to A, A \to \varepsilon\}$	$0^4 1^2$	$L(G) = \{2^{2n}1^{2n}0, n = 0,1,...\}$
3	$P = \{S \to 0S, S \to \varepsilon, S \to 1\}$	0^3	$L(G) = \{2^{2n}0^n, n = 0,1,...\}$
4	$P = \{S \to 1A, A \to 101, S \to 0B, B \to 010\}$	$0^2 10$	$L(G) = \{2^{2+n}, n = 0,1,...\}$
5	$P = \{S \to 111S, S \to \varepsilon\}$	1^6	$L(G) = \{2^n 1^m, n, m = 0,1,...\}$
6	$P = \{S \to 1B0, B \to 1B, B \to 0\}$	$1^3 0^2$	$L(G) = \{2^n aba, n = 0,1,...\}$
7	$P = \{S \to 1S, S \to 0, S \to B, B \to 01\}$	$1^3 0$	$L(G) = \{2^n 1, n = 2,3,...\}$
8	$P = \{S \to B01, B \to 1B, B \to 0\}$	$1^3 0^2 1$	$L(G) = \{2^n 1^{m+2}, n, m = 0,1,...\}$
9	$P = \{S \to S1, S \to A, S \to 1, S \to \varepsilon, A \to 0\}$	1^3	$L(G) = \{0^n 1^{2m}, n, m = 0,1,...\}$
10	$P = \{S \to 0S, S \to 1, S \to 1B, B \to 1B \mid 0\}$	$0^4 1$	$L(G) = \{a^{3n} b^n, n = 0,1,...\}$
11	$P = \{S \to 0S0, S \to 1, S \to 1B1, B \to 0\}$	$0^2 10^2$	$L(G) = \{a^n 00 b^n, n = 0,1,...\}$
12	$P = \{S \to AB, B \to 11B \mid \varepsilon, A \to 000A \mid 0\}$	$1^2 0^6$	$L(G) = \{0^{3n+1} b^m, n, m = 0,1,...\}$
13	$P = \{S \to 0S1, S \to A, A \to 11A11 \mid 0\}$	$0^3 1^2$	$L(G) = \{(a^{3n})^m, n, m = 0,1,...\}$
14	$P = \{S \to 00S00, S \to A, A \to 1A1 \mid 0\}$	$1^2 01^2$	$L(G) = \{a^n ccb^n, n = 0,1,...\}$

Розділ 2. СКІНЧЕННІ АВТОМАТИ.

2.1. УЗАГАЛЬНЕНЕ ПОНЯТТЯ АВТОМАТУ

Однією з найбільш важливих задач, що вирішуються в теорії формальних мов, є ***задача розпізнавання.***

Нехай задана деяка породжувальна граматика G з термінальним алфавітом V і ланцюжком x в алфавіті V. Необхідно визначити чи належить ланцюжок x мові, що породжена граматикою G. Цю задачу називають проблемою приналежності для граматики G. В теорії формальних мов доводиться, що проблема приналежності є розв'язною (існує алгоритм її розв'язку) для КЗ-граматик і КВ-граматик, але в загальному випадку нерозв'язна для граматик типу 0.

Вирішення проблеми приналежності полягає в розробці алгоритму розпізнавання, який для довільної граматики G (з заданого класу граматик) і ланцюжка x за скінченне число кроків визначає, чи породжується заданий ланцюжок x граматикою G. В основі таких алгоритмів лежить математична модель мови, яка називається розпізнавальною моделлю або аналізуючою моделлю і в деякій мірі є протилежною до задачі породження слів мови граматикою G. Традиційно аналізуючі моделі називають ***автоматами***. У кожному класі мов може бути визначена пара "породжувальна модель – аналізуюча модель" або, іншими словами, пара "граматика - автомат", де автомат аналізує (розпізнає) ланцюжок, породжений граматикою.

Неформально автомат можна описати як пристрій, що складається з блоку управління, вхідної стрічки, головки автомату і блоку внутрішньої пам'яті автомата (рис. 2.1.1).

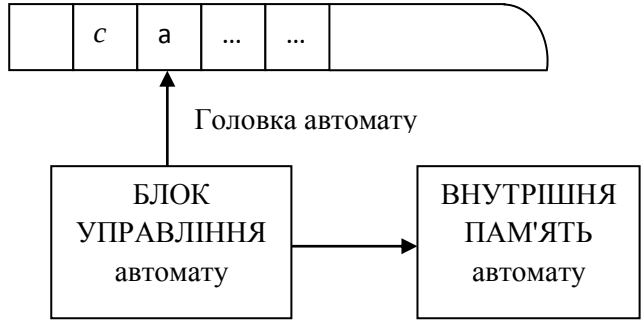


Рис. 2.1.1 Узагальнена схема автомату.

На стрічці, яка є напівнезкінченною (не обмежена праворуч) і розділена на комірки, записуються ланцюжки у вхідному алфавіті V так, що букви ланцюжка розміщені послідовно, починаючи з першої комірки далі і без пропусків, в кожній комірці стрічки - по одній букві. Ланцюжок, записаний таким чином на вхідній стрічці автомату, будемо називати вхідним ланцюжком. Блок управління в кожен момент часу знаходитися в одному з скінченної множини станів (позначимо його через Q), а головка встановлена на одній клітинці вхідної стрічки. У такому випадку говорять, що автомат фіксує дану комірку.

Автомат, читаючи вхідний ланцюжок, працює по кроках (або по тактах). Нехай в деякий момент часу автомат фіксує якусь комірку стрічки, а блок управління знаходиться в деякому стані $q \in Q$. Такт роботи автомата полягає в тому, що в залежності від вмісту зафіксованої комірки, стану q, а також вмісту внутрішньої пам'яті автомат переміщує (зсуває) головку вправо або вліво на одну клітинку або залишає її на попередньому місці, блок управління автомату переходить в деякий новий стан r (можливо, що цей стан співпаде з попереднім станом q), а вміст внутрішньої пам'яті певним чином модифікується. У загальному випадку автомат може і поміняти вміст зафіксованої комірки стрічки, тобто автомат може працювати зі стрічкою або в режимі тільки читання, не змінюючи її вмісту, а лише певним чином зсуваючи головку, або в режимі читання і запису.

Вводять поняття конфігурації автомата: вона визначається

- станом q блоку управління,
- вмістом a зафіксованої комірки
- вмістом внутрішньої пам'яті.

Автомат в загальному випадку не є недетермінованим пристроєм, тобто для нього із заданої конфігурації можливі переходи в декілька різних конфігурацій. Правила, згідно з якими автомат переходить з однієї конфігурації в іншу, складають в сукупності систему команд автомата. Кожна команда дозволяє перехід з однієї конфігурації в якусь іншу. Це нагадує шахову гру, коли з поточної позиції на дошці можна, зробивши хід, отримати нову позицію - одну з множини всіх позицій, в які можна потрапити з поточної позиції, зробивши хід згідно з правилами гри. В даному випадку правила гри аналогічні системі команд, а позиція на дошці - конфігурації автомату.

Автомати класифікуються відповідно до структури своєї внутрішньої пам'яті, за режимом роботи з стрічкою (тільки читання або читання / запис), а також за типом руху головки по стрічці (односторонній, двосторонній, число комірок, на які за один такт можна перемістити головку). Множина команд є скінченною, тобто автомат, як і граматика, має скінчений опис.

Уявімо тепер наступну ситуацію. Нехай на вхідній стрічці автомата записано деякий ланцюжок $x \in V^*$. Припустимо також, що серед станів блоку управління виділено деякий спеціальний стан, який назвемо початковим, а також деяку підмножину станів, які назвемо заключними.

У початковий момент часу блок управління знаходиться в початковому стані, головка оглядає першу (крайню ліву) комірку стрічки, в якій записаний перший символ вхідного ланцюжка x. Читаючи ланцюжок x і роблячи один такт за іншим, автомат, після того як він прочитає останню букву ланцюжка, буде знаходитися в якомусь стані (точніше кажучи, в цьому стані буде знаходитися блок управління). Якщо цей стан є заключним, то говорять, що автомат допустив ланцюжок. Виникає запитання: яким чином з допомогою граматики можна описати множину всіх ланцюжків в алфавіті V, які автомат допускає?

Виявляється, що кожному класу граматики відповідає свій клас автоматів, причому для кожної граматики відповідного класу може бути побудований автомат, який допускає ланцюжки, породжені даною граматикою, і тільки їх. Образно кажучи, в кожному класі мов виникає пара "письменник – читач", де граматика виступає в ролі "письменника", породжує певну множину "текстів" (ланцюжків в заданому алфавіті), а "читач" (автомат) перевіряє правильність цих текстів, допускаючи ті і тільки ті ланцюжки, які породжуються його граматикою. В ролі "письменника" може виступати програміст, що пише тексти комп'ютерних програм, а в якості "читача" - системні програми, що забезпечують перевірку правильності написаного тексту (відповідності граматиці тієї чи іншої мови програмування). Тим самим допускаючий автомат стає прообразом деякого розпізнавального алгоритму, що вирішує проблему приналежності до того чи іншого класу граматики.

Зауважимо, однак, що автомат сам по собі ще не являється алгоритмом розпізнавання. Крім того, виявляється, наприклад, що в класі граматик типу 0 в загальному випадку побудувати алгоритм для вирішення проблеми

приналежності неможливо, хоча автомати, відповідні граматикам типу 0, існують (так звані машини Тюринга).

Ми починаємо з найпростіших аналізуючих моделей - скінченних автоматів. Скінченні автомати - це аналізуючі моделі для регулярних мов. Скінченний автомат не має внутрішньої пам'яті, головка рухається по стрічці тільки вправо - на одну комірку за такт. З стрічки можна тільки читати. Крім того, автомат може переходити "спонтанно" з одного стану в інший, не читаючи стрічку і не пересуваючи головку. Такий такт можна розглядати як перехід зі стану в стан по порожньому ланцюжку. Його називають λ -тактом.

Отже, з кожного стану автомат може переходити в інший стан, читаючи той чи інший символ вхідного ланцюжка, або робити перехід по порожньому ланцюжку, причому приймається за визначенням, що ці два типи переходів виключають один одного. Поведінка скінченного автомата визначається його системою команд, в якій кожна команда задається відповідним записом

$$qa \rightarrow r\,, \tag{2.1}$$

що означає: "зі стану q по символу $a \in V$ або по порожньому ланцюжку (тоді $a = \lambda$) можна перейти в стан r " (можливо, що $q = r$).

При цьому за визначенням приймається, що перехід по порожньому ланцюжку і перехід по вхідному символу виключають один одного. Тобто для будь-якої пари $(q\ ,\ r)$ станів скінченного автомату має місце наступне: якщо існує команда (2.1) при $a = \lambda$, то немає жодної іншої команди (для такої ж пари станів $(q\ ,\ r)$) при $a \in V$ і навпаки .

Конфігурація скінченного автомату визначається як впорядкована пара $(q\ ,\ ay)$, де q - стан блоку упра¬вління, a – символ, який читається головкою автомату в комірці вхідної лєнти, y - ланцюжок у вхідному алфавіті, розташований в комірках праворуч від головки автомату (ланцюжок ay прийнято називати непрочитаною частиною вхідного ланцюжка). При цьому, якщо комірка, на якій зафіксована головка, порожня, тобто не містить будь-якого символу вхідного алфавіту, то непрочитана частина вхідного ланцюжка вважається порожнім ланцюжком.

Щоб дати математичне визначення скінченного автомату, потрібно зауважити, що він, в світлі щойно викладеного неформального опису, допускає природну інтерпретацію в термінах розмічених орієнтованих графів. Будемо

розглядати стани блоку управління скінченного автомату як вершини орієнтованого графа, множина дуг якого визначається системою команд наступним чином: дуга веде зі стану q в стан r тоді і тільки тоді, коли в системі команд автомату є команда (2.1), тобто можливий перехід зі стану q в стан r. Кожна дуга має мітку, яка є або множиною літер вхідного алфавіту, або порожнім ланцюжком.

Мітка дуги $(q\ ,\ r)$ є порожній ланцюжок λ, якщо з q в r можна перейти по порожньому ланцюжку; в іншому випадку мітка дуги $(q\ ,\ r)$ є множиною всіх вхідних символів, по яких можливий перехід зі стану q в стан r.

Приклад 2.1.1

Задамо автомат з вхідним алфавітом $\{a, b, c\}$, множиною станів $\{q_0, q_1, q_2, q_3\}$, та наступною системою команд

$$q_0\,\lambda \rightarrow q_1$$
$$q_0\,\lambda \rightarrow q_3$$
$$q_1\,a \rightarrow q_2$$
$$q_1\,b \rightarrow q_2$$
$$q_1\,a \rightarrow q_3$$
$$q_2\,b \rightarrow q_1$$
$$q_3\,b \rightarrow q_2$$
$$q_3\,c \rightarrow q_2$$
$$q_3\,c \rightarrow q_3$$

За цією системою команд побудуємо розмічений орієнтований граф, зображений на рис. 2.1.2.

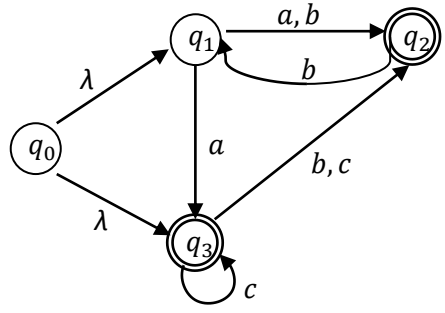


Рис.2.1.2. Граф автомату з прикладу

2.1.1

Серед станів автомату виділені початковий стан q_0 і два заключних стани q_2, q_3 .На рис. 2.1.3 показана послідовність конфігурацій, яку проходить скінченний автомат, читаючи ланцюжок $abac$. Цей ланцюжок автомат допускає, оскільки, читаючи його, він переходить з початкового стану - q_0 в один із заключних. У формальному записі процес аналізу ланцюжка $abac$ на рис. 2.1.3 визначається як послідовність конфігурацій :

$$(\ q_0, abac), (\ q_1, abac), (\ q_2, bac), (\ q_1, ac), (\ q_3, c), (\ q_3, \lambda).$$

Такій послідовності конфігурацій автомату відповідає наступний шлях в графі

$$q_0 \xrightarrow[\lambda]{} q_1 \xrightarrow[a]{} q_2 \xrightarrow[b]{} q_1 \xrightarrow[a]{} q_3 \xrightarrow[c]{} q_3$$

(під кожною стрілкою підписана буква, що належить мітці відповідної дуги графу і яка є буквою вхідної строки, яка зчитується автоматом в даному стані). Зауважимо, що, наприклад, перебуваючи в стані q_1 і читаючи символ a, автомат міг би, згідно з системою команд, зробити перехід в стан q_2, а не в стан q_3, але тоді він би "завис" в стані q_2 і не зміг би прочитати останнього символу записанного на стрічці ланцюжка, тобто символ с, так як серед команд немає такої, яка б дозволяла перехід зі стану q_2 куди-небудь по символу с.

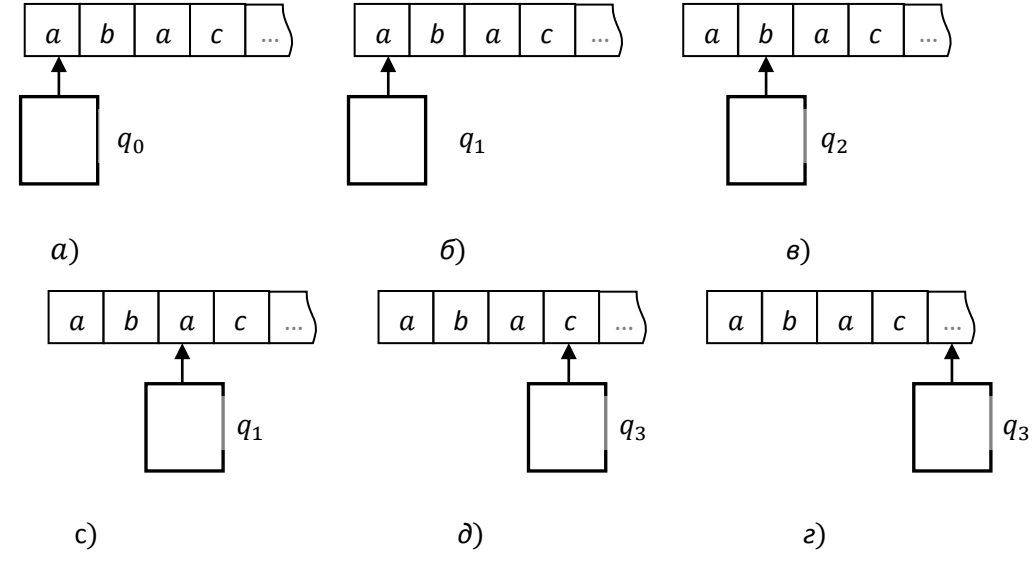


Рис.2.1.3 Аналіз ланцюжка abac автоматом

Ця ситуація демонструє якраз недетермінованість скінченного автомату як розпізнавального пристрою: система команд ("правила гри") дозволяє автомату допустити ланцюжок $abac$ ("гравцю" знайти послідовність ходів, які ведуть до "виграшу"), але з цього зовсім не випливає, що послідовність конфігурацій, яку проходить автомат, читаючи записаний на стрічці ланцюжок, є єдиною. Автомат може "помилитися", зробивши "неправильний" хід. Але й послідовність "правильних" ходів може бути не єдиною. Наприклад, читаючи останній символ ланцюжка, тобто символ c, автомат міг би "вибрати" перехід в стан q_2, який також є заключним. Розглянутий автомат допускає не всякий ланцюжок в алфавіті {a, b, c}. Видно, що жодний ланцюжок, який починається з префікса ca, не буде допущений автоматом.

2.2. НЕДЕТЕРМІНОВАНІ СКІНЧЕННІ АВТОМАТИ

Дамо строге математичне визначення скінченного автомату.

Визначення 2.2.1.

Скінченний автомат (finite automaton, finite-state machine) - це п'ятірка $M = \langle Q, \Sigma, \Delta, I, F \rangle$, де

Σ - скінченний *вхідний алфавіт* (або просто *алфавіт*) скінченного автомату,

Q - скінченна множина *станів* (state) автомату,

Δ - скінченна множини команд (переходів) автомату, $\Delta \subseteq Q \times \Sigma^* \times Q$, $F \subseteq Q$,

I – скінченна множина початкових (initial) станів автомату $I \subseteq Q$,

F – скінченна множина *заключних* або *допускаючих* (final, accepting) станами. Якщо $\langle p, x, q \rangle \in \Delta$, то $\langle p, x, q \rangle$ називається *переходом* (transition) з p в q, а слово $x \in \Sigma^*$ - *міткою* (label) цього переходу.

Інколи функцію переходів представляють як команду автомату $px \rightarrow q$ - автомат переходить зі стану p в стан q по дузі з міткою x. Для позначення всіх

станів, в які може перейти автомат зі стану p по дузі з міткою x використовують функцію $\delta\ (p, x)$.

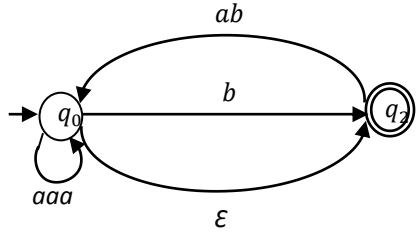
Приклад 2.2.1.

Нехай *Q = {1,2}*, $\Sigma = \{a, b\}$, *I = {1}*, *F = {2}*,
$\Delta = \{ < 1, aaa, 1 >, \quad < 1, ab, 2 >, \quad < 1, b, 2 >, \quad < 2, \varepsilon, 1 > \}$

Тоді < *Q*, Σ, Δ ,*I*,*F* > - скінченний автомат.

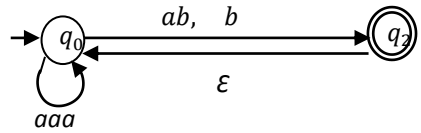
Скінченні автомати можна зображати у вигляді *діаграм станів* (state diagram) (іноді їх називають *діаграмами переходів* (transition diagram)). На діаграмі кожний стан позначається кружком, а перехід - стрілкою. Стрілка з *p* в *q*, помічена словом *x*, показує, що $< p, x, q >$ являється переходом даного скінченного автомату. Кожний початковий стан розпізнається по вхідній в нього короткій стрілці. Кожний заключний стан відмічається на діаграмі подвійним кружком.

Приклад 2.2.2.
На діаграмі зображено скінченний автомат з прикладу 2.2.1.



Якщо в скінченому автоматі є декілька переходів з спільним початком і спільним кінцем, то такі переходи називаються *паралельними*. Іноді на діаграмі станів скінченого автомату паралельні переходи зображаються одною стрілкою. При цьому мітки переходів перераховують через кому.

Приклад 2.2.3.
На діаграмі зображено скінченний автомат з прикладу 2.2.1 з паралельними переходами.



Визначення 2.2.2.

Шлях (path) скінченого автомату - це кортеж

$< q_0, d_1, q_1, d_2, \dots, q_n >$, де $n \geq 0$ і $d_i = < q_{i-1}, w_i, q_i > \in \Delta$ для кожного і. При цьому q_0 – початок шляху q_n - *кінець шляху*, n - *довжина шляху*, $w_1, w_2, \dots, w_n$ - *мітка шляху*.

Для будь-якого стану $q \in Q$ існує шлях $< q >$. Його мітка – Ɛ (інколи ще позначають λ), початок і кінець співпадають.

Визначення 2.2.3.

Шлях називається ***успішним***, якщо його початок належить *I*, а кінець належить *F*.

Приклад 2.2.4.

Розглянемо скінченний автомат з прикладу 2.2.1.

Шлях

$< 1, < 1, b, 2 >, \quad 2, < 2, \varepsilon, 1 >, \quad 1, < 1, aaa, 1 >, \quad 1, < 1, b, 2 >, \quad 2 >$

являється успішним. Його мітка - baaab. Довжина цього шляху - 4.

Визначення 2.2.4.

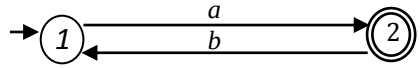
Слово **w** ***допускається*** (is accepted, is recognized) скінченим автоматом M, якщо воно являється міткою деякого успішного шляху.

Визначення 2.2.5.

Мова, що розпізнається скінченим автоматом M, - це мова L(M), яка складається з міток усіх успішних шляхів (тобто з усіх слів, що допускаються даним автоматом). Будемо також говорити, що автомат M *розпізнає* (recognizes, accepts) мову L(M).

Приклад 2.2.5.

Нехай дано автомат $M = \langle Q, \Sigma, \Delta, I, F \rangle$, де Q = {1,2}, $\Sigma = \{a, b\}$, I = {1}, F = {1,2}, $\Delta = \{ \langle 1, a, 2\rangle, \langle 2, b, 1\rangle\}$



Тоді

$$L(M) = \{ (ab)^n \mid n \geq 0 \} \cup \{ (ab)^n a \mid n \geq 0 \}$$

Визначення 2.2.6.

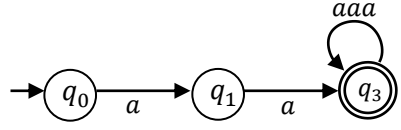
Два скінченних автомати ***еквівалентні,*** якщо вони розпізнають одну і ту ж мову.

Визначення 2.2.7.

Мова L називається ***автоматною*** (finite-state language), якщо існує скінченний автомат, що розпізнає цю мову.

Приклад 2.2.6.

Розглянемо однобуквений алфавіт $\Sigma = \{a\}$. При довільних фіксованих $k \in N, m \in N$ мова $\{ a^{k+mn} \mid n \in N$ являється автоматною. На рисунку нижче зображено автомат для $k = 2, m = 3$



Кожна скінченна мова являється автоматною.

Дамо декілька важливих визначень спеціальних видів автоматів.

Визначення 2.2.8.

Автомат $M = \langle Q, \Sigma, \Delta, I, F \rangle$ називається ***всюди визначеним***, якщо для кожного стану і для кожного символу вхідного алфавіту автомату існує перехід

в інший стан, тобто з кожного стану повинні виходити дуги, позначені по черзі всіма буквами алфавіту (кількість таких дуг дорівнює потужності алфавіту |Σ|).

Іншими словами автомат $M = \langle Q, \Sigma, \Delta, I, F \rangle$ ***всюди визначений***, якщо

$$\forall\, q \in Q,\ \forall\, a \in \Sigma,\ \delta(q,a) \neq \emptyset\ .$$

Наприклад, автомат $M = \langle Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0,1\}, \Delta, I = \{q_0\}, F = \{q_3\} \rangle$, зображений на рис.2.2.1., не є всюди визначеним.

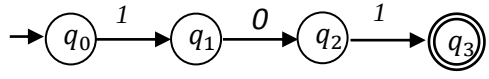


Рис 2.2.1.Автомат, який розпізнає мову $L(M) = \{101\}$ *з одного ланцюжка 101*.

Проте він легко перетворюється у всюди визначений автомат шляхом додавання стану ∅, з якого не має переходу в заключний стан і , відповідно, через який не йде розпізнавання ланцюжків мови(рис2.2.4).

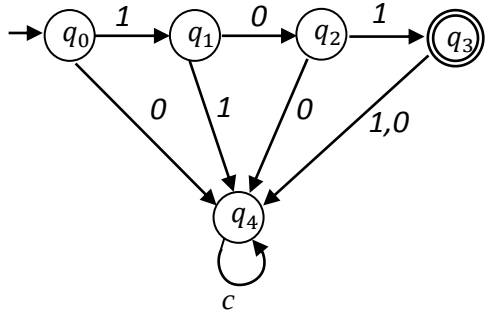


Рис.2.2.2.Всюди визначений скінченний автомат для аналізу мови $L(M) = \{101\}$.

На перший погляд, таке перетворення виглядає штучним і по меншій мірі зайвим, проте воно тісно пов’язане з поняттям детермінованості автомату та використовується для розв’язку цілого класу нетривіальних задач (див. параграф 2.7 – доповнення автоматних мов).

Визначення 2.2.9.

Скінченний автомат $M = \langle Q, \Sigma, \Delta, I, F \rangle$ називається **детермінованим,** якщо

- **a.** в автоматі не має дуг з мітками, поміченими пустими словами λ
- **b.** для кожного стану і для кожного символу вхідного алфавіту автомату ***існує тільки*** один перехід в інший стан, тобто з кожного стану виходять по одній дузі для кожної букви вхідного алфавіту, тобто

$$\forall\, q \in Q, \qquad \forall\, a \in \Sigma, \qquad |\delta\,(q, a)| = 1$$

Визначення 2.2.10.

Скінченний автомат $M = \langle Q, \Sigma, \Delta, I, F \rangle$ називається **квазідетермінованим,** якщо

- **a.** в автоматі не має дуг з мітками, поміченими пустими словами λ
- **b.** для кожного стану і для кожного символу вхідного алфавіту автомату ***існує не більше ніж*** один перехід в інший стан, тобто

$$\forall\, q \in Q, \qquad \forall\, a \in \Sigma, \qquad |\delta\,(q, a)| \leq 1$$

Остання умова означає, що зі стану q мітки дуг визначені не для кожної букви вхідного алфавіту автомату a .

Скінченний автомат на рис 2.2.1 є квазідетермінованим, а автомат на рис 2.2.2. являється детермінованим автоматом.

2.3. СКІНЧЕННІ АВТОМАТИ З ОДНОБУКВЕННИМИ ПЕРЕХОДАМИ

Лема 2.3.1. *Кожна автоматна мова розпізнається скінченним автоматом з мітками довжини 1 і одним початковим та одним кінцевим станом*

Для виконання умов леми необхідно перетворити довільний автомат наступним чином

1. При наявності декількох початкових станів автомату додати додатковий стан, який стане єдиним початковим станом, та з'єднати його дугами з міткою λ з кожним з попередніх початкових станів, при цьому попередні початкові стани надалі вважаються звичайними станами.
2. При наявності декількох заключних станів автомату, необхідно додати новий стан, який стане надалі єдиним заключним станом, та з'єднати кожен попередній заключний стан з новим дугою з міткою λ. Попередні заключні стани стають при цьому звичайними станами.
3. При наявності багатобуквенної мітки $a_1 \ldots a_i$ на дузі між двома станами p та q, необхідно додати додаткові стани $p_1, \ldots, p_{i-1}$, які будуть проміжними між станами p та q, та з'єднати стан p з станом p_1 дугою з міткою a_1 , стан p_1 з станом p_2 дугою з міткою a_2 і т.д.

Приклад 2.3.1.

Розглянемо мову, задану скінченним автоматом $M_1 = \langle Q, \Sigma, \Delta, I, F \rangle$, де Q = {1,2},

$\Sigma = \{a, b, c, d\}$, I = {1}, F = {2}, $\Delta = \{ \langle 1, ab, 2\rangle, \langle 2, cd, 1\rangle$.

Діаграма автомату зображена на рис. 2.3.1.

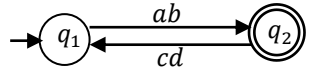


Рис. 2.3.1.Автомат M_1 з двобуквенними переходами

Застосуємо процедуру перетворення автомату з однобуквенними переходами з леми 2.3.1.

Отримаємо автомат $M_2 = \langle Q', \Sigma, \Delta', I', F' \rangle$ де Q' = {1,2,3,4}, I' = {1}, F' = {2} , Δ { $\langle 1, a, 3 \rangle$, $\langle 4, d, 1 \rangle, \langle 3, b, 2 \rangle$, $\langle 2, c, 4 \rangle$.

Діаграма атавтомату M_2 зображена на рис 2.3.2

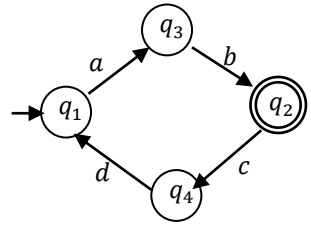


Рис. 2.3.2 Автомат M_2 з однобуквенними переходами, еквівалентний M_1

2.4. ХАРАКТЕРИЗАЦІЯ ПРАВОЛІНІЙНИХ МОВ

Встановимо зв'язок між процесом породження (***синтезом***) мови та процесом її ***аналізу*** скінченним автоматом.

Якщо для процесу породження формальної мови використовується породжувальна граматика, то для процесу аналізу мови використовується скінченний автомат. Наступні теореми встановлюють зв'язок між скінченними автоматами та праволінійними граматиками.

Теорема 2.4.1.
Кожна автоматна мова являється праволінійною.

Щоб довести теорему, потрібно:

вказати спосіб побудови регулярної граматики G по заданому скінченному автомату M, такий, щоб мова, породжувана граматикою G, збігалася з мовою, що допускаються автоматом M: $L(G) = L(M)$;

Доведення

Побудуємо за автоматом $M = \langle Q, \Sigma, \Delta, I, F \rangle$ праволінійну граматику, що породжує ту ж автоматну мову, яку розпізнає автомат.

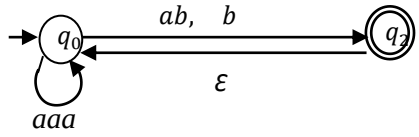
Визначимо граматику $G = (V, N, S, P)$ наступним чином:

- термінальний алфавіт V співпадає з вхідним алфавітом Σ автомату M: $V = \Sigma$;
- нетермінальний алфавіт N побудуємо як взаємно однозначну відповідність до множини станів Q автомату M наступним чином: кожному стану $q \in Q$ поставимо у відповідність нетермінал $S_q \in N$, аксіома співставляється початковому стану $q_0 \in I$, тобто $S = S_{q_0}$
- множину правил виводу P будуємо за системою команд Δ так: правило виводу $S_q \to a\, S_r$, де $a \in \Sigma$, належить Р тоді і тільки

тоді, коли в Δ є команда $qa \to r$ (або відповідно перехід $\langle q, a, r\rangle \in \Delta$); крім того, якщо (і тільки якщо) стан r – заключний ($r \in F$), то в P додається правило виводу $S_q \to a$, якщо ж q_0 є одночасно і початковим і заключним станом (тобто $q_0 \in I, F$, то додаємо правило $S_{q_0} \to \lambda$.

Приклад 2.4.1.

Мова, що розпізнається наступним скінченним автоматом,



породжується праволінійною граматикою з правилами

$$S_1 \to aaaS_1$$
$$S_1 \to abS_2$$
$$S_1 \to bS_2$$
$$S_2 \to S_1$$
$$S_2 \to \lambda$$

Теорема 2.4.2.

Кожна праволінійна мова являється автоматною.

Для доведення теореми необхідно

вказати спосіб побудови скінченного автомату $M = \langle Q, \Sigma, \Delta, I, F\rangle$ по заданій граматиці $G = (V, N, S, P)$, так, щоб мова, яка допускається автоматом M, збігалася з мовою, породжуваною граматикою G: $L(M) = L(G)$;.

Зауваження.

Як наслідок теорем 2.4.1 та 2.4.2 граматику G і скінченний автомат M, такі, що $L(G) = L(M)$, іноді називають *еквівалентними*.

Доведення.

Покладемо

- вхідний алфавіт автомата M рівним термінальному алфавіту граматики G : Σ = V,
- для визначення множини станів Q та функції переходів Δ , перетворимо наступні правила праволінійної граматики G виду
  $A \to u$, де $u \in V^*$ на два правила
  $A \to uB$ та $B \to \varepsilon$,
  додавши новий нетермінал B, який буде відповідати заключному стану автомату. В цьому випадку новий нетермінальний алфавіт граматики $N' = N \cup \{B\}$ доповниться новими нетерміналами B. Тоді множина станів Q буде співпадати з новим нетермінальним алфавітом $N' = N \cup \{B\}$ граматики G,
- множину заключних станів утворять нові нетермінали $B, (B \in F)$, вони реалізують команду (функцію переходу) автомату $B \to \lambda$
- початковий стан збігається з аксіомою S,
- система команд (переходів) визначається за принципом: команда $Aa \to C$ (перехід $\langle A, a, C \rangle$) належить Δ тоді і тільки тоді, коли в множині правил граматики є правило виводу $A \to aC$, де $A, C \in N, a \in \Sigma$.

Приклад 2.4.2.

Нехай $\Sigma = \{a, b\}$

Розглянемо граматику над алфавітом Σ з наступними правилами виводу:

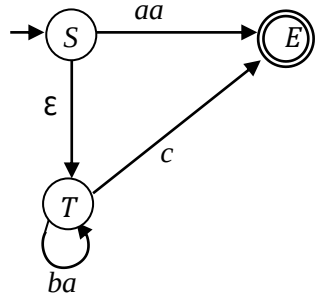
$$S \to aa$$
$$S \to T$$
$$T \to baT$$
$$T \to a$$

Вона еквівалентна граматиці

$$S \rightarrow aaE$$
$$S \rightarrow T$$
$$T \rightarrow baT$$
$$T \rightarrow aE$$
$$E \rightarrow \varepsilon$$

Мова, яка породжується цими граматиками, розпізнається скінченним автоматом $\langle Q, \Sigma, \Delta, I, F \rangle$, *де Q = {S,T,E}, I = {S}, F = {E} і*

$$\Delta = \{ \langle S, aa, E\rangle, \langle S, \varepsilon, T\rangle, \langle T, ba, T \rangle, \langle T, a, E\rangle \}$$



2.5. ВИДАЛЕННЯ НЕПРОДУКТИВНИХ ТА НЕДОСЯЖНИХ СТАНІВ СКІНЧЕННИХ АВТОМАТІВ.

Визначення 2.5.1.

Стан q ***досягається*** (reachable) зі стану p, якщо існує шлях, початком якого являється p, а кінцем - q.

Лема 2.5.1.

Кожна автоматна мова розпізнається деяким скінченим автоматом, в якому кожний стан досягається з деякого початкового стану і з кожного стану досягається хоча б один заключний стан.

Для доведення леми достатньо задати алгоритми видалення непродуктивних і недосяжних станів автомату і показати, що такі перетворення є еквівалентними.

Стан автомату називається ***недосяжним***, якщо не існує шляху до нього з початкового стану.

Стан автомату називається ***непродуктивним***, якщо не існує шляху від нього до заключного стану.

Доцільно шукати не безпосередньо самі недосяжні стани, а послідовно визначати множину досяжних станів, починаючи з початкового; аналогічно множину продуктивних станів починаємо будувати, вибравши стани, в які можна потрапити з заключних, рухаючись в напрямку, протилежному до напрямку дуг автомату. Перетин множин досяжних і продуктивних станів автомату утворять множину станів нового автомату, який є еквівалентним до перетворюваного - всі інші стани потрібно видалити.

Спочатку видаляються непродуктивні стани, а потім недосяжні, оскільки процедура видалення недосяжних станів може привести до появи нових непродуктивних станів. Одночасне визначення досяжних і продуктивних символів неможливо, оскільки відповідні процеси йдуть у протилежних напрямках (від заключного стану до початкового і навпаки).

Алгоритм 2.5.1. Визначення продуктивних станів автомату

Вхід. Автомат $M = \langle Q,\ \Sigma,\ \Delta, I, F\rangle$

Вихід.Автомат $M' = \langle Q',\ \Sigma',\ \Delta',\ I',F'\rangle$ без непродуктивних станів, такий, що $L\ (M') = L\ (M)$.

Метод. Виконати кроки 1-4:

(1) Покласти $N_1 = \{\ A \mid A \in F$- заключні стани$\}$ та i = 1,

(2) Покласти $N_i = \{\ X \mid X \in Q$ якщо в Δ є перехід $\langle X, u, A\rangle$ і $A \in N_{i-1}\ \}$ U N_{i-1},

(3) Якщо $N_i \neq N_{i-1}$, покласти $i = i + 1$ і перейти до кроку 2, в противному випадку перейти до кроку 4,

(4) Покласти $Q' = N_i \cap Q$, $I' = N_i \cap I$. Включити в Δ′ всі переходи з Δ , обидва стани яких входять в N_i .

Алгоритм 2.5.2. Визначення досяжних станів автомату

Вхід. Дано автомат $M = \langle Q,\ \Sigma,\ \Delta, I, F\rangle$

Вихід. Автомат $M' = \langle Q',\ \Sigma',\ \Delta',\ I',F'\rangle$ без недосяжних станів, такий, що $L\ (M') = L\ (M)$.

Метод. Виконати кроки 1-4:

(1) Покласти $V_0 = \{\ A \mid A \in I$ - всі початкові стани автомату$\}$ та i = 1,

(2) Покласти $V_i = \{\ X \mid X \in Q$ якщо в Δ є перехід $\langle A, u, X\rangle$ і $A \in V_{i-1}\ \}$ U V_{i-1},

(3) Якщо $V_i \neq V_{i-1}$, покласти $i = i + 1$ і перейти до кроку 2, в противному випадку перейти до кроку 4,

(4) Покласти $Q' = V_i \cap Q$, F′ = $V_i \cap F$. Включити в Δ′ всі такі переходи з Δ, обидва стани яких входять в V_i .

Важливо: Щоб видалити всі непродуктивні та недосяжні стани, необхідно застосувати до автомата спочатку Алгоритм 2.5.1, а потім Алгоритм 2.5.2.

Приклад 2.5.1.

Розглянемо скінченний автомат $M_1 = \langle Q,\ \Sigma,\ \Delta,\ I,\ F\rangle$, де Q = {1,2,3,4}, $\Sigma = \{a, b, c, d\ )$, I = {1,2,4}, F = {1,3,4},

функція переходів $\Delta = \{\ \langle 1, b, 4\rangle,\ \langle 1, a, 2\rangle,\ \langle 3, c, 4\rangle,\ \ \langle 4, d, 1\rangle$

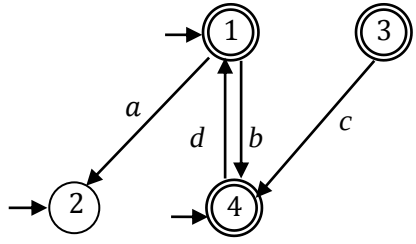


Рис. 2.5.1. Автомат M_1

Знайдемо всі продуктивні стани автомату M_1.

Покладемо $N_1 = \{1, 3, 4\}$ заключні стани автомату. Знайдемо всі стани M_1, з яких виходить дуга автомату до станів з N_1. Такими станами є $N_2 = \{1, 3, 4\}$, тобто $N_1 = N_2$, а це означає що знайдено всі продуктивні стани, тому стан 2 є непродуктивним і його слід видалити та видалити всі переходи, пов'язані з ним. Отримаємо

$M_2 = \langle Q_2,\ \Sigma,\ \Delta_2, I_2, F_2\rangle$, де Q_2 = {1,3,4}, $\Sigma = \{a, b. c, d\ )$, I_2 = {1,4}, F_2 = {1,3,4},

функція переходів $\Delta_2 = \{\ \langle 1, b, 4\rangle, \langle 3, c, 4\rangle, \langle 4, d, 1\rangle$

Знайдемо тепер всі досяжні стани автомату M_2.

Покладемо $V_1 = \{1,\ 4\}$ – вхідні стани автомату M_2. В множину V_2 включимо всі такі стани, в які входить дуга зі станів множини V_1 - такими станами є стан 1 та 4, тобто $V_2 = \{1,4\}$. Оскільки $V_1 = V_2$, то всі досяжні стани

знайдено і шуканий автомат без недосяжних та непродуктивних станів матиме вигляд

$M_3 = \langle Q_3, \Sigma, \Delta_3, I_3, F_3 \rangle$, де Q_3 = {1,4}, $\Sigma = \{a, b. c, d\}$, I_3 = {1,4}, F_3 = {1,4},

функція переходів $\Delta_3 = \{\langle 1, b, 4\rangle, \langle 4, d, 1\rangle$. Далі наведена діаграма станів M_3.

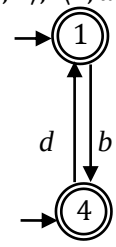


Рис. 2.5.2. Автомат M_3 без недосяжних та непродуктивних станів.

2.6. ДЕТЕРМІНОВАНІ СКІНЧЕННІ АВТОМАТИ

Визначення 2.6.1.

Скінченний автомат $M = \langle Q, \Sigma, \Delta, I, F \rangle$ називається ***детермінованим*** (deterministic), якщо

1. множина I містить рівно один елемент;
2. для кожного переходу $\langle p, x, q\rangle \in \Delta$ виконується рівність $| x | = 1$, тобто мітки переходів автомату є однобуквенними;
3. автомат не містить дуг з пустими λ мітками;
4. для кожного символу $a \in \Sigma$ і для довільного стану $p \in Q$ існує тільки один стан $q \in Q$ такий що $\langle p, a, q\rangle \in \Delta$, тобто перехід зі стану p в стан q по дузі з міткою a повинен бути єдиним для кожної букви алвавіту.

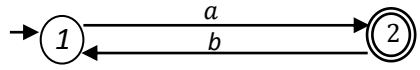
Автомат є **квазідетермінованим,** якщо виконуються умови 1-3 детермінованості автомату, але не для кожного символу вхідного алфавіту з кожного стану виходить одна дуга .

Приклад 2.6.1.

Скінченний автомат

$M = \langle Q,\ \Sigma,\ \Delta, I, F\rangle$, де Q = {1,2}, $\Sigma = \{a, b\}$, I = {1}, F = {1,2}, $\Delta = \{\langle 1, a, 2\rangle,\ \langle 2, b, 1\rangle$

квазідетермінований.



Він розпізнає мову

$$L(M) = \{\ (ab)^n \mid n \geq 0\ \} \cup \{\ (ab)^n a \mid n \geq 0\ \}$$

Теорема 2.6.1. Про приведення недермінованого скінченного автомату до детермінованого.

Для будь-якого недетермінованого скінченного автомату можна побудувати еквівалентний йому детермінований скінченний автомат.

Доведення.

Перетворення довільного скінченного автомату до еквівалентного детермінованого здійснюється в два етапи:

1) спочатку видаляються дуги з міткою λ,
2) потім проводиться власне детермінізація.

1. Видалення λ -переходів (дуг та шляхів з міткою λ).

Щоб перейти від вихідного скінченного автомату $M = \langle Q,\ \Sigma,\ \Delta, I, F\rangle$ до еквівалентного скінченного автомату $M' = \langle Q',\ \Sigma,\ \Delta', I', F'\rangle$ без λ -переходів, досить у вихідному графі M здійснити наступні перетворення.

1. Всі стани, крім початкового, в які заходять тільки дуги з міткою λ, видаляються; тим самим визначається множина Q' скінченного автомату M'. Зрозуміло, що $Q' \subseteq Q$. При цьому вважаємо, що початковий стан залишається попереднім
2. Множина дуг скінченного автомату M' і їх міток (тим

самим і функція переходів M') визначається так:

для довільних двох станів $p, r \in Q'$ перехід з p в r по дузі з міткою a:

$$p \underset{a}{\rightarrow} r$$

має місце тоді і тільки тоді, коли $a \in \Sigma$, а в графі М

або існує дуга з р в г, мітка якої символ a,

або існує такий стан q, що $p \overset{+}{\underset{\lambda}{\Rightarrow}} q$ і $q \underset{a}{\rightarrow} r$

При цьому вершина q, взагалі кажучи, може не належати Q' і надалі бути видалена в процесі подальшого видалення λ переходів (рис. 2.6.1.)

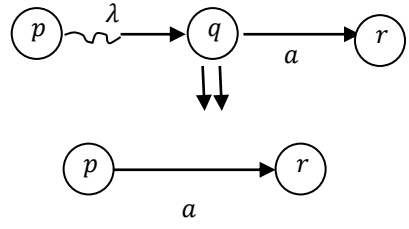
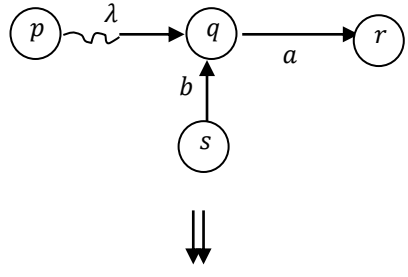


Рис.2.6.1. Верхній автомат перетворюється до нижнього, в якому видалено λ перехід

Якщо ж $q \in Q'$ (тобто крім дуги з міткою λ в нього входять принаймі одна дуга з міткою $b \in \Sigma, b \neq \lambda$), природньо, в М' збережеться дуга $p \underset{a}{\rightarrow} r$ і символ a буде одним з символів, що належать мітці цієї дуги (рис. 2.6.2).

Таким чином, в М 'зберігаються всі дуги М, мітки яких відмінні від λ і які з'єднують пару станів з множини Q' , що не видаляються згідно з п. а). Крім цього, для будь-якої трійки станів p, q, r (не обов'язково різних!), таких, що p, $r \in Q'$ і існує шлях не нульової довжини з p в q, мітка якого дорівнює λ (тобто шлях по λ -переходам), а з q в r веде дуга, мітка якої містить символ a вхідного алфавіту, в М' будується дуга з p в r, мітка якої містить символ a (див. рис. 2.6.1).



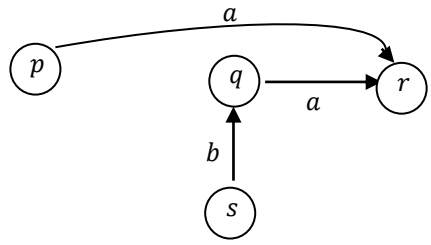


Рис.2.6.2. Верхній автомат перетворюється до нижнього, де видалено λ перехід

3. Множина заключних станів F' скінченного автомату M' містить всі стани $q \in Q'$, які або належали до заключних станів початкового автомату M , або з яких веде шдях не нульової довжини з q в заключний стан $f \in F$ початкового автомату M з міткою шляху λ (рис. 2.6.3).

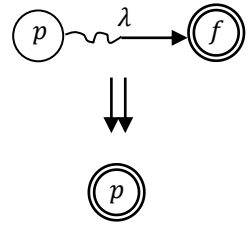


Рис.2.6.3. Верхній автомат перетворюється до нижнього, в якому видалено λ перехід

2. ***Власне детермінізація***.

Вважаємо, що автомат, який підлягає процедурі власне детермінізації вже не містить λ переходів.

Спочатку зауважимо, що автоматна граматика (надалі А-граматика) відповідає детермінованому автомату тоді і тільки тоді, коли в кожному її правилі виду

$A \to a_1A_1|...|a_nA_n$ або $A \to a_1A_1|...|a_nA_n \mid \varepsilon$

виконується умова: $a_i \neq a_j$ при $i \neq j$. (2.1)

Нагадаємо, що правило $A \to \varepsilon$ відповідає заключному стану автомату.

Для приведення автомату до детермінованого виду достатньо перетворити відповідну йому А-граматику $G = (N,T,P,S)$ до вказаного виду (2.1), оскільки в параграфі 2.4 розділу 2 ми довели, що між автоматними граматиками та скінченними автоматами існує взаємнооднозначна відповідність.

Для кожного правила, ліві частини яких є нетерміналами з множини $\{A_1,...,A_n\}$, а праві частини цих правил розпочинаються однаковими термінальними символами, введемо новий нетермінальний символ, який позначимо як $[A_1 \ ... \ A_n]$ і визначимо для нього правило граматики так, щоб його права частина складалася з ***правих частин правил для кожного*** $\boldsymbol{A_i}$*, що входить в множину* $\{A_1,...,A_n\}$. тобто задамо правило

$$[A_1 \ ... \ A_n] \to r_1 \mid ... \mid r_n ,$$

де r_i - права частина правила для A_i , тобто $A_i \to r_i$

Тепер побудуємо граматику $G' = (N', T, P', S)$, де N' отримується з N додаванням визначених вище нових нетермінальних символів, а P' отримано додаванням до P правил для нових нетермінальних символів: в кожному правилі всі члени виду aA_1, ..., aA_n з одним і тим же терміналом $a \in T$ заміняються одним правилом $a[A_1,...,A_n]$.

Таким правилам відповідає детермінований автомат, еквівалентний недетермінованому, так як $a[A_1,...,A_n]=aA_1|...|aA_n$.

Так як не всі нетермінальні символи $\boldsymbol{[A_1,...,A_n]}$ ***досяжні з S, то при побудові*** $\boldsymbol{P'}$ ***достатньо включати тільки необхідні правила, починаючи з правила для S.***

Приклад 2.6.1.

Недетермінований автомат

$M_d = \langle Q,\ \Sigma,\ \Delta, I, F\rangle$, де $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $I = \{q_0\}$, $F = \{q_3\}$,

$\Delta = \{\langle q_0, a, q_1\rangle,\ \langle q_0, b, q_2\rangle, \langle q_1, a, q_1\rangle, \langle q_1, b, q_1\rangle, \langle q_2, a, q_2\rangle, \langle q_2, b, q_2\rangle,$
$\langle q_1, b, q_2\rangle, \langle q_2, a\ q_1\rangle, \langle q_1, a, q_3\rangle,\ \langle q_2, b, q_3\rangle,\ \langle q_3, a, q_3\rangle, \langle q_3, b, q_3\rangle,$

зображений на малюнку зліва. Він не містить λ переходів, тому до нього можна одразу застосувати власне детермінізацію.

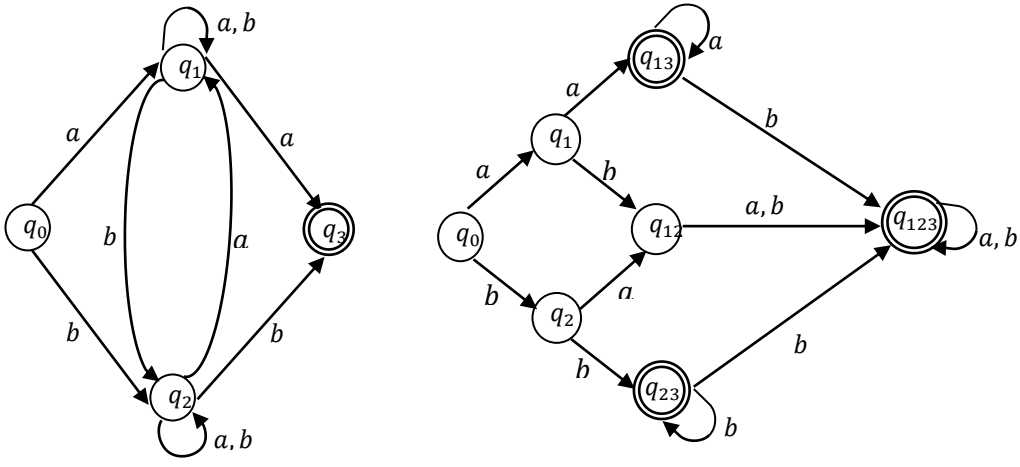


Рис.2.6.4. Приведення скінченного автомату M_d до детермінованого виду.

Побудуємо еквівалентну даному автомату граматику з правилами (теорема 2.4.1)

$q_0 \rightarrow aq_1 \mid bq_2$,

$q_1 \rightarrow aq_1 \mid aq_3 \mid bq_1 \mid bq_2$,

$q_2 \rightarrow aq_1 \mid aq_2 \mid bq_2 \mid bq_3$,

$q_3 \to aq_3 \mid bq_3 \mid \varepsilon$

Правило $q_3 \to \varepsilon$ додається в граматику, оскільки q_3 є заключним станом.

Перетворимо його в детермінований автомат по методу теореми 2.6.1. В нову граматику ввійде старе правило для q_0, так як воно задовільняє умові детермінованості. Правило для q_1 перетворюється до виду

$q_1 \to a(\, q_1 \mid q_3) \quad \mid \quad b(\, q_1 \mid q_2\,)$,
тоді
$q_1 \to aq_{13} \mid bq_{12}$.

Аналогічно отримуємо правило для

$q_2 \to a\,(\, q_1 \mid q_2\,) \quad \mid \quad b(\, q_2 \mid q_3)$,
тоді
$q_2 : \to aq_2 \mid aq_{12} \mid bq_{23}$.

Для $q_{13} = q_1 \mid q_3$ правило має вид

$q_{13} \to aq_1 \mid aq_3 \mid bq_1 \mid bq_2 \mid bq_3 \mid \varepsilon$,
отримане шляхом включення правих частин правил для нетерміналів q_1 та q_3, або, остаточно,

$q_{13} \to aq_{13} \mid bq_{123} \mid \varepsilon$, де $q_{123} = q_1 \mid q_2 \mid q_3$.

Аналогічно, для $q_{12} = q_1 \mid q_2$, $q_{23} = q_2 \mid q_3$ і q_{123} правила (після перетворення) мають вид

$q_{12} \to aq_{123} \mid bq_{123}$,
$q_{23} \to aq_{123} \mid bq_{23} \mid \varepsilon$,
$q_{123} \to aq_{123} \mid bq_{123} \mid \varepsilon$.

Діаграма станів побудованого детермінованого автомату зображена на рис.2.6.4 справа.

Зауваження.

Отриманий детермінований скінченний автомат повинен далі обов'язково пройти процедуру видалення непродуктивних та недосяжних станів (алгоритми 2.5.1. та 2.5.2), поскільки процес детермінізації автомату призводить до появи таких станів.

Приклад 2.6.2.

На рис 2.6.5. представлено скінченний недетермінований автомат, який містить λ переходи.

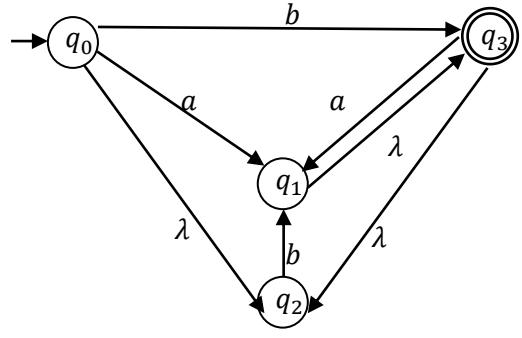


Рис. 2.6.5.

Продемонструємо на ньому процедуру видалення λ переходів. Поскільки в стан q_2 є два λ переходи, спробуємо видалити їх.

Шлях $q_0 \underset{\lambda}{\Rightarrow} q_2 \underset{b}{\Rightarrow} q_1$ при видаленні λ переходу перетвориться на $q_0 \underset{b}{\Rightarrow} q_1$,

шлях $q_3 \underset{\lambda}{\Rightarrow} q_2 \underset{b}{\Rightarrow} q_1$ при видаленні λ переходу перетвориться на $q_3 \underset{b}{\Rightarrow} q_1$,

шлях $q_1 \underset{\lambda}{\Rightarrow} q_3 \underset{\lambda}{\Rightarrow} q_2$ взагалі може бути видалений, бо він складається виключно з λ переходів. В результаті стан q_2 не входить в жодний новоутворений шлях та не задіяний в інших шляхах початкового автомату і може бути видалений.

Отриманий автомат забражений на рис.2.6.6.

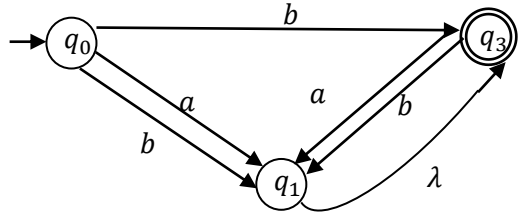


Рис. 2.6.6.

Зауважимо, що λ перехід $q_1 \underset{\lambda}{\Rightarrow} q_3$ не може бути видалений на даному етапі, бо він приймає участь ще в деяких шляхах, що складаються з міток вхідного алфавіту. Далі перетворюючи автомат з рис 2.6.6., отримаємо

Шлях $q_0 \underset{b}{\Rightarrow} q_1 \underset{\lambda}{\Rightarrow} q_3$ при видаленні λ переходу перетвориться на $q_0 \underset{b}{\Rightarrow} q_3$, який вже є в автоматі.

Шлях $q_0 \underset{a}{\Rightarrow} q_1 \underset{\lambda}{\Rightarrow} q_3$ при видаленні λ переходу перетвориться на $q_0 \underset{a}{\Rightarrow} q_3$.

Шлях $q_1 \underset{b}{\Rightarrow} q_3 \underset{\lambda}{\Rightarrow} q_1$ при видаленні λ переходу перетвориться на $q_1 \underset{b}{\Rightarrow} q_1$,

Шлях $q_1 \underset{a}{\Rightarrow} q_3 \underset{\lambda}{\Rightarrow} q_1$ при видаленні λ переходу перетвориться на $q_1 \underset{a}{\Rightarrow} q_1$,

Оскільки є λ перехід $q_1 \underset{\lambda}{\Rightarrow} q_3$ і q_3 - заключний стан, то при його видаленні q_1 стає теж заключним.

В результаті отримаємо наступний автомат без λ переходів, який зображений на рис.2.6.7.

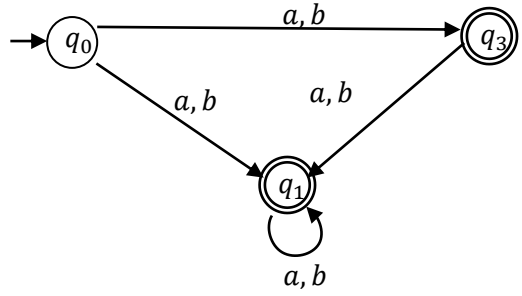


Рис.2.6.7.

Для отриманого автомату без λ переходів пропонуємо подальшу детермінізацію провести самостійно.

2.7. ДОПОВНЕННЯ АВТОМАТНИХ МОВ.

Одним з важливих теоретичних наслідків теореми про детермінізацію являється наступна теорема.

Теорема 2.7.1.
Доповнення автоматної мови є автоматною мовою.

Доведення

Нехай $L(M)$ – автоматна мова, що розпізнається автоматом $M = \langle Q, \Sigma, \Delta, I, F \rangle$.

Тоді доповнення мови $L(M)$ як доповнення до множини слів є мова $\overline{L(M)} = \Sigma^* \setminus L(M)$, яка складається зі слів в алфівіті Σ, які не належать мові $L(M)$.

Згідно з теоремою про детермінізацію будь-якого скінченного автомату, для мови $L(M)$ автомат M може бути приведений до детермінованого виду, який буде допускати ту ж саму мову $L(M)$.

Оскільки в детермінованому автоматі з кожної вершини по кожному вхідному символу визначено перехід строго тільки в одну вершину, то, яким би не був ланцюжок $x \in \Sigma^*$ в алфавіті Σ, який належить до мови $L(M)$, для нього знайдеться єдиний шлях в М, що починається в початковому стані, по якому читається цей ланцюжок. Зрозуміло, що ланцюжок x допускається автоматом М, тобто $x \in L(M)$, тоді і тільки тоді, коли останній стан зазначеного шляху є заключним. Звідси випливає, що ланцюжок $x \notin L(M)$ тоді і тільки тоді, коли останній стан зазначеного шляху не є заключним.

Але нам якраз і потрібен скінчнний автомат M', який допускає ланцюжок тоді і тільки тоді, коли його не допускає вихідний скінченний автомат M.

Отже, перетворюючи кожен заключний стан M в незаключний і навпаки, отримаємо детермінований автомат M', що допускає доповнення до мови $L(M)$

Доведена теорема дозволяє будувати скінченний автомат, який не допускає певну множину ланцюжків, наступним методом:

- будуємо спочатку автомат, що допускає потрібну множину ланцюжків,
- детермінізуємо його
- переходимо до автомату для доповнення мови так, як це зазначено в доведенні теореми 2.7.1.

Приклад 2.7.1.

Побудуємо скінченний автомат, який допускає всі ланцюжки в алфавіті {0,1}, крім ланцюжка 101.

Спочатку побудуємо скінченний автомат, що допускає єдиний ланцюжок 101.

Цей автомат наведено на рис. 2.7.1

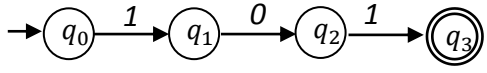


Рис. 2.7.1

Цей автомат квазідетермінірованний, але недетермінований, оскільки він не всюди визначений, тобто зі стану q_0 виходить дуга з міткою 1, але не виходить дуга з міткою 0. Аналогічно і для станів q_1, q_2, q_3 .

Доповнимо його дугами, яких не вистарчає для того, щоб автомат став детермінованим, але в такий спосіб, щоб він розпізнавав і надалі мову, утворену одним ланцюжком 101.

Зі стану ∅ не виходить жодна дуга і він не є заключним, тому автомати на рис 2.7.1 та 2.7.2 розпізнають одну і ту ж мову $L(M) = \{101\}$, тому є еквівалентними.

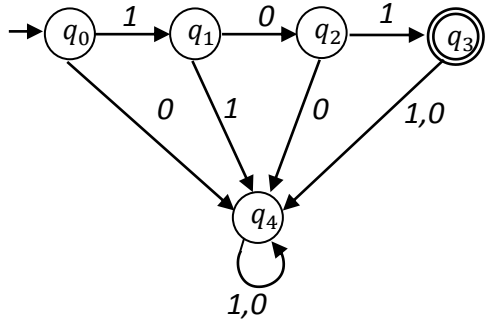


Рис.2.7.2. Детермінований автомат, що розпізнає мову $L(M) = \{101\}$

Автомат на рис 2.7.2. є детермінованим, він не містить λ дуг, з кожного його стану виходить рівно 1 дуга з кожною буквою алфавіту 0 або 1, він містить один початковий і один заключний стан та тільки однобуквенні переходи. Для мови, що аналізується ним $L(M) = \{101\}$ згідно теореми 2.7.1. може бути побудована мова $\Sigma^* - L(M)$ - доповнення мови $L(M)$ до універсальної мови Σ^* , яка може бути описана як мова, що містить будь-які ланцюжки з 0 та 1, крім ланцюжка 101. Автомат, який буде розпізнавати таку мову ми отримаємо, якщо замінимо всі заключні стани автомату з рис.2.7.2. на звичайні, а звичайні стани цього автомату – замінимо на заключні.

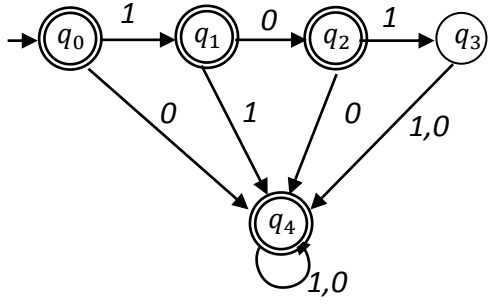


Рис.2.7.3.Автомат, що розпізнає мову з ланцюжків 0 та 1, крім ланцюжка 101.

Звернемо увагу, що в отриманому автоматі всі вершини, крім вершини q_3, є заключними. Зауважимо також, що перехід до доповнення, про який йде мова в доведенні теореми 2.7.1, може бути проведений тільки в детермінованому автоматі. Якщо б ми поміняли ролями заключні та незаключні вершини в автоматі, зображеному на рис. 2.7.1, то отримали б автомат, який допускає мову {λ, 1,10}, яка аж ніяк не є множиною всіх ланцюжків, відмінних від ланцюжка 101.

Відзначимо також, що скінченний автомат на рис. 2.7.3 допускає всі ланцюжки, що містять входження ланцюжка 101, тобто ланцюжки виду α101β, але не є точно самим цим ланцюжком 101, бо α, β ≠ λ.

Розглянемо ще один важливий приклад використання доповнення автоматних мов для розв'язку нетривіальних задач в теорії синтезу та аналізу формальних мов.

Приклад 2.7.2.

Побудуємо скінченний автомат, що допускає всі ланцюжки в алфавіті (0,1}, крім тих, які містять входження ланцюжка 101. Розглянемо мову L, кожний ланцюжок якої містить входження ланцюжка 101. Його можна задати так:

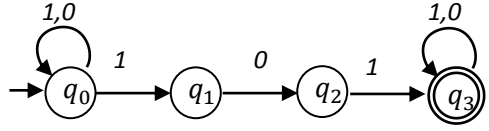


Рис.2.7.4. Автомат, що розпізнає мову ланцюжків з 0 та 1, кожен з яких містить підланцюг 101.

Автомат не містить λ переходів, має один вхід та один вихід і тільки однобуквення переходи. Тому для приведення його до детермінованого виду достатньо провести тільки процедуру власне детермінізації.

Визначимо праволінійну граматику, яка породжує таку ж мову, яку розпізнає даний автомат, для проведення детермінізації автомату з рис. 2.7.4. Автоматна право лінійна граматика містить наступні правила

$$q_0 \rightarrow 0q_0 \mid 1q_0 \mid 1q_1$$

$$q_0 \rightarrow 1q_1$$

$$q_1 \rightarrow 0q_2$$

$$q_2 \rightarrow 1q_3$$

$$q_3 \rightarrow 0q_3 \mid 1q_3 \mid \varepsilon$$

Детермінізуємо перше правило

$$q_0 \rightarrow 0q_0 \mid 1q_{01}$$

$$q_{01} \rightarrow 0q_0 \mid 1q_{01} \mid 0q_2$$

Друге з цих правил детермінізується наступним чином

$$q_{01} \rightarrow 0q_{02} \mid 1q_{01}$$

$$q_{02} \rightarrow 0q_0 \mid 1q_{01} \mid 1q_3$$

Далі

$$q_{02} \rightarrow 0q_0 \mid 1q_{01\,3}$$

$$q_{013} \rightarrow 0q_0 \mid 1q_{01} \mid 0q_2 \mid 0q_3 \mid 1q_3 \mid \varepsilon = 0q_{023} \mid 1q_{013} \mid \varepsilon$$

$$q_{023} \rightarrow 0q_0 \mid 1q_{01} \mid 1q_3 \mid 0q_3 \mid 1q_3 \mid \varepsilon = 0q_{03} \mid 1q_{013} \mid \varepsilon$$

$$q_{03} \rightarrow 0q_0 \mid 1q_{01} \mid 0q_3 \mid 1q_3 \mid \varepsilon = 0q_{03} \mid 1q_{013} \mid \varepsilon$$

Остаточно граматика, що еквівалентна детермінованому автомату має наступні правила:

$$q_0 \rightarrow 0q_0 \mid 1q_{01}$$
$$q_{01} \rightarrow 0q_{02} \mid 1q_{01}$$
$$q_{02} \rightarrow 0q_0 \mid 1q_{01\,3}$$
$$q_{013} \rightarrow 0q_{023} \mid 1q_{013} \mid \varepsilon$$
$$q_{023} \rightarrow 0q_{03} \mid 1q_{013} \mid \varepsilon$$
$$q_{03} \rightarrow 0q_{03} \mid 1q_{013} \mid \varepsilon$$

Автомат, що аналізує мову, породжену наведеною граматикою зображено на рис. 2.7.5.

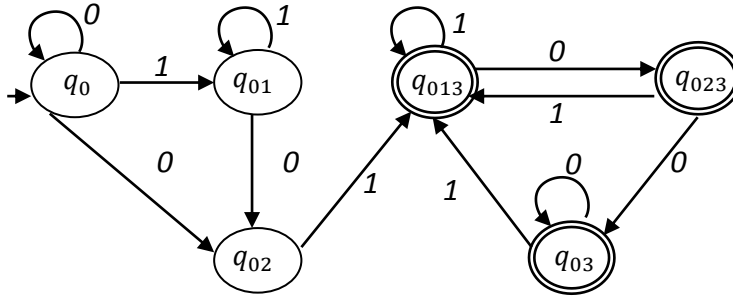


Рис.2.7.5. Детермінований автомат, що розпізнає мову ланцюжків з 0 та 1, в які входить 101

Для побудови автомату, який допускає мову з ланцюжків 0 та 1, які не містять 101, побудуємо доповнення до автомату на рис. 2.7.5, замінивши в ньому всі заключні стани на звичайні, а звичайні – на заключні. Отримаємо автомат

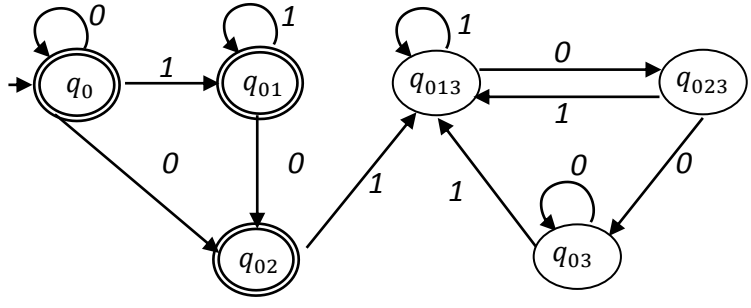
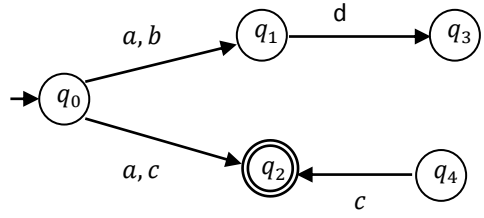


Рис.2.7.5. Детермінований автомат, що розпізнає мову ланцюжків з 0 та 1, в які не входить 101

2.8. **МІНІМІЗАЦІЯ СКІНЧЕННОГО АВТОМАТУ.**

В подальшому при програмуванні скінчених автоматів важливо мати справу з так званими "мінімальними автоматами". Мінімальним для даного скінченого автомата є еквівалентний йому автомат з мінімальною кількістю станів. Те, що скінчені автомати можна мінімізувати покажемо на наступному прикладі:



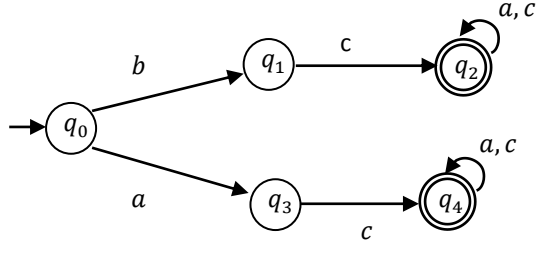
Навіть при поверхневому аналізі діаграми переходів наведеного скінченого автомата видно, що стан q_3 є непродуктивним (з нього не має шляху до заключного стану), а стан q_4 – недосяжним (немає шляху до нього з початкового стану q_0). При їх вилученні новий автомат буде еквівалентний початковому.

З наведеного вище прикладу видно, що для отриманого автомата можна запропонувати еквівалентний йому автомат з меншою кількістю станів, тобто мінімізувати скінчений автомат. Очевидно що серед зайвих станів автомату є недосяжні та непродуктивні стани.

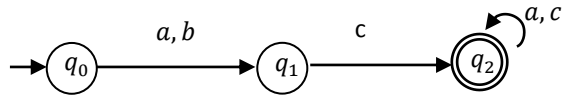
Проте видалення недосяжнимх та непродуктивних станів не завершує процедуру мінімізації кількості станів скінченного автомату.

Автомат, у котрого відсутні недосяжні та непродуктивні стани, піддається подальшій мінімізації шляхом "склеювання" еквівалентних станів.

Продемонструємо це на конкретному прикладі:



Очевидно, що для наведеного вище скінченого автомата можна побудувати еквівалентний йому скінчений автомат з меншою кількістю станів:



Ми досягли бажаного нам результату шляхом “склеювання” станів q_1, q_3 в один стан та склеювання q_2, q_4.

Щоб представити алгоритм побудови для будь-якого скінченного автомату еквівалентного йому автомату з мінімальною кількістю станів введемо *наступні обмеження на вид скінченного автомату*, які не порушують загальності наступних викладок:

- будемо припускати, що автомат, який підлягає мінімізації є детермінованим (будь-який скінченний автомат може бути приведеним до детермінованого виду – теорема 2.6.1)
- будемо також припустити, що у вихідному скінченному автоматі немає станів, що недосяжні з початкового стану (алгоритм 2.5.2 видалення недосяжних станів).

На множині станів автомату $M = \langle Q,\ \Sigma,\ \Delta, I, F\rangle$ задамо сімейство відношень еквівалентності наступним чином:

1) Побудуємо відношення 0-еквівалентності $\cong^0$ наступним чином

$$q_1 \ \cong^0 \ q_2\ ,$$

якщо обидва стани одночасно належать F або Q\F, тобто обидва стани одночасно є заключним або одночасно - незаключними.

2) Побудуємо відношення k-еквівалентності $\cong^k$:

при $k \geq 1$ покладемо $q_1 \ \cong^k \ q_2$ тоді і тільки тоді, коли

- $q_1 \cong^{k-1} q_2$ тобто стани q_1, q_2 є $(k-1)$ - еквівалентні,
- для будь-якого вхідного символа a стани, в які можна потрапити по дузі з міткою a зі стану q_1 та зі стану q_2 також $(k-1)$ - еквівалентні. Позначимо множину станів, в які можна потрапити з q_1 по дузі з міткою a через $\delta(q_1, a)$, а множину станів, в які можна потрапити з q_2 по дузі з міткою a через $\delta(q_2, a)$.

Щоб зрозуміти сенс відношення k- -еквівалентності, звернемося до рис 2.8.1.

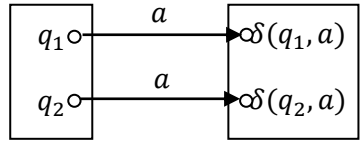


Рис.2.8.1.

На цьому малюнку q_1, q_2 є $(k-1)$ – еквівалентні, тобто вони належать одному і тому ж классу $(k-1)$ - еквівалентності С. Ці стани, стануть k - еквівалентнимиі, якщо для будь-якого вхідного символа a стани $\delta(q_1, a)$ та $\delta(q_2, a)$ також є $(k-1)$ –еквівалентними, тобто належать до того самого класу еквівалентності С.

Можна сказати, що $(k-1)$ -еквівалентні стани будуть також і k - еквівалентними, якщо перехід з них по будь-якому вхідному символу "зберігає" $(k-1)$ - еквівалентність станів. Якщо ж знайдеться хоча б один вхідний символ a, такий, що стани $\delta(q_1, a)$ та $\delta(q_2, a)$, отримані переходом з q_1, q_2 по хоча б по одній дузі з певною міткою опиняться в різних класах $(k-1)$ –еквівалентності (рис. 2.8.2), то стани q_1, q_2 вже НЕ БУДУТЬ k - еквівалентними (вони розійдуться по різних класах k -еквівалентності, оскільки перехід з них по деякому символу порушує $(k-1)$ -еквівалентність.

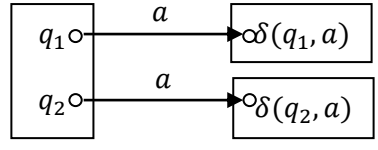


Рис.2.8.2.

Таким чином, кожен клас k –еквівалентності або розбивається на кілька попарно непересічних класів $(k+1)$ - еквівалентності, або не зміниться, якщо всі його стани залишаться $(k+1)$ - еквівалентними. Це означає, що кількість множин станів в класі k – еквівалентності з ростом k може збільшуватись або залишатися незмінною.

Мінімізація скінченного автомата полягає в послідовному подрібненні ("розбитті") множини станів Q автомату M на класи еквівалентності до тих пір, поки не отримаємо розбиття, яке вже не можна подрібнити (очевидно, що таке розбиття завжди існує через скінченність множини станів). Більш строго: зазначені вище відношення еквівалентності будуються до такого найменшого k, що відношення $\cong^{k-1}$ співпаде з відношенням $\cong^{k}$. Це відношення і визначає найдрібніше розбиття множини станів. Позначимо його просто $\equiv$. Тоді мінімальний скінченний автомат $M' = \langle Q',\ \Sigma,\ \Delta', I',\ F' \rangle$, що еквівалентний початковому автомату M, визначається таким чином:

- в множину станів Q' включаються тільки по одному стану з кожного класу еквівалентності – позначимо такий узагальнений для класу еквівалентності n стан через $[q_n]$.
- вхідним станом (єдиним) стане узагальнений стан $[q_0]$, який позначає клас, куди входять початкові стани автомату M: $I' = \{[q_0]\}$.
- Заключним станом (єдиним) стане узагальнений заключний стан $[f]$, що представляє клас еквівалентності, куди входять заключні стани автомату M: $F' = \{[f]\}$.
- Множина переходів $\Delta' = \{\ \langle [q_n], a, [\delta([q_n], a] \rangle\ \}$, тобто в множину переходів включаються тільки переходи між класами еквівалентності з класу еквівалентності з узагальненим станом $[q_n]$ по дузі з міткою a в узагальнений стан класу еквівалентності, куди попали всі стани $q \in Q$ *автомату* M, в які можна попасти по дузі з міткою a з станів, що входять в клас $[q_n]$.

Теорема 2.8.1.

Для довільного скінченного автомату може бути побудований еквівалентний йому скінченний автомат з найменшим числом станів, що базується на наступній процедурі побудови:

1) будуємо еквівалентний вихідному детермінований скінченний автомат;
2) якщо в отриманому скінченному автоматі залишилися недосяжні стани з початкового, видаляємо їх (алгоритм 2.5.2)
3) до отриманого скінченного автомату застосовуємо наведений вище алгоритм побудови розбиття множини станів на класи еквівалентності по відношенню ≡ і будуємо мінімальний скінченний автомат M', як описано вище.

Приклад 2.8.1. Мінімізуємо скінченний автомат, зображений на рис. 2.8.3.

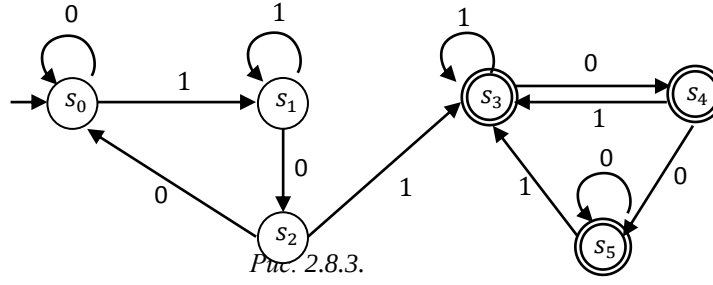


Рис. 2.8.3.

Запишемо розбиття множини станів автомата для відношення $\cong^0$:

Дане відношення розбиває множину станів на 2 підмножини: заключних та незаключних станів.

$\{s_0\ ,\ s_1\ , s_2\ \}$ - незаключні стани, $\{s_3\ ,\ s_4\ , s_5\ \}$ - заключні стани

Оскільки стани

$$\delta(s_2\ , 0) = s_0 \ \text{ та } \ \delta(s_2\ , 1) = s_3$$

не є 0-еквівалентними станами, бо вони належать різним классам, то в розбитті для 1-еквівалентності вони "розійдуться" по різних класах; розбиття, обумовлене відношенням $\cong^1$, буде мати вигляд

$$\{s_0\,,\ s_1\ \}\,,\quad \{\,s_2\ \}\,,\quad \{s_3\,,\ s_4\,,s_5\,\}\text{-}$$

Далі, при переході до 2-еквівалентності доведеться "розвести "стани $s_0\,,$ $s_1\,,$ бо

$$\delta(s_0\,,0)=\ s_0 \text{ та } \delta(s_1\,,0)=\ s_2$$

а s_0 та s_2 належать до різних класів 1 – еквівалентності.

Оскільки для всіх станів з множини $\{s_3\,,\ s_4\,,s_5\ \}$ скінченний автомат переходить в один з цих же станів, то розбиття на класи 2-еквівалентності і є шукане "найдрібніше" розбиття:

$$\{s_0\ \}\,,\quad \{\,s_1\,\},\quad \{\,s_2\,\}\,,\quad \{s_3\,,\ s_4\,,s_5\,\}$$

Позначимо узагальнений стан для класу $\{s_3\,,\ s_4\,,s_5\,\}$ через q, На рис. 2.8.4.приведена діаграма станів мінімального автомату

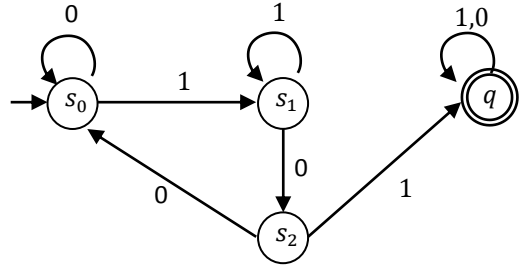


Рис. 2.8.4.

Зауважимо, що застосування розглянутої процедури мінімізації може дати два крайніх випадки:

1) всі стани початкового скінченного автомату виявляться еквівалентними і тоді в мінімальному скінченному автоматі залишиться тільки один стан;
2) підсумкове розбиття складатиметься з одноелементних класів еквівалентності - це означає, що початковий скінченний автомат не можна мінімізувати, він вже мінімальний.

Важливо, що процедуру мінімізації автомату часто використовують для *встановлення факту еквівалентності двох скінченних автоматів*. Для цього обидва автомати мінімізуються. Якщо після мінімізації автомати мають різну кількість станів, то вони завідомо нееквівалентні. У випадку співпадіння кількості станів обох мінімізованих автоматів, намагаються знайти взаємнооднозначне відображення між станами та напрямками і мітками дуг таких автоматів, якщо таке відображення вдається побудувати – автомати вважаються еквівалентними.

ЗАДАЧІ ТА ВПРАВИ ДО РОЗДІЛУ 2 .

1. Формальне визначення того, що стан s скінченного автомату A = (S, Σ, δ,I={ s0}, F) є досяжним, записується так:
   Стан s досяжний $\Leftrightarrow \exists\, \alpha \in \Sigma^*, [\delta^*(s_0, \alpha) = s\ ]$,
   що словами можна сказати так:
   Стан s досяжний в скінченному автоматі A = (S, Σ, δ,I={ s0}, F) тоді і тільки тоді, коли існує ланцюжок символів над вхідним словником Σ автомату, під впливом якого автомат з початкового стану переходить в стан s.
   Запишіть формальне визначення і словесне формулювання визначення недосяжного стану в скінченному автоматі.

2. Формальне визначення того, що мова L над словником Σ допускається скінченним автоматом A = (S, Σ, δ,I={ s0}, F), записується так:
   мова $L \subseteq \Sigma^*$ допускається автоматом A $\Leftrightarrow$
   $$\forall\, \alpha \in \Sigma^*, [\alpha \in L \equiv \delta^*(s_0, \alpha) \in F\ ],$$
   що словами можна сказати так:
   Мова L над словником Σ допускається скінченним автоматом A = (S, Σ, δ,I={ s0}, F) тоді і тільки тоді, коли будь-який ланцюжок символів цього словника належить мові L якщо і тільки якщо він переводить автомат A в один з заключних станів.
   Запишіть формально і словесно визначення того, що мова $L \subseteq \Sigma^*$ не допускається скінченним автоматом A = (S, Σ, δ,I={ s0}, F).

3. Побудуйте скінченний автомат з вхідним алфавітом Σ = {a, b}, що розпізнає:
   а) порожню мову;
   б) мову, що містить тільки один порожній ланцюжок λ;
   в) мову, що складається з усіх можливих ланцюжків в словнику Σ = {a, b};
   г) мову, що складається з усіх непустих ланцюжків в словнику Σ = {a, b}.

4. Побудуйте детермінований скінченний автомат з вхідним алфавітом {0,1}, який розпізнає:
   а) всі ланцюжки, що закінчуються 11.
   б) всі ланцюжки, крім тих ланцюжків, які закінчуються кодом 11.
   в) всі ланцюжки, в яких не зустрічається трьох одиниць поспіль.
   г) всі ланцюжки, крім тих, які включають код 110 в якості подланцюжка.
   д) всі ланцюжки, в яких число входжень 0 непарне.

5. В якості адреси електронної пошти не можна вибрати довільний ланцюжок символів. Такі адреси визначаються наступним чином.
   До символу @ може йти будь –яка послідовність, що складається з непустих груп латинських букв і цифр, розділених крапками. Після символу @ можуть йти будь-які непорожні послідовності латинських літер і цифр, розділених крапками. Побудуйте скінченний автомат, що розпізнає правильнуструктуру адреси електронної пошти.

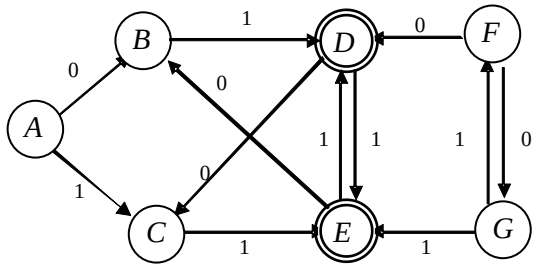
6. Побудуйте детермінований скінченний автомат з вхідним алфавітом {a,b, c}, що розпізнає всі ланцюжки:
7.
   а) точно з одним входженням символу b.
   б) не більше ніж з трьома входженнями символу b.
   в) у яких третя буква від кінця b.
   г) не менше ніж з двома входженнями символу b.

8. Нехай L - автоматна мова над словником Σ = {a, b, c}. Чи буде автоматним доповнення цієї мови, тобто мова $\Sigma^* \setminus L$?
   Якщо так, то за скінченним автоматом, що розпізнає мову

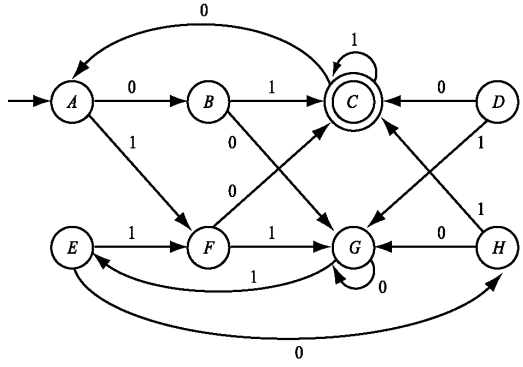
$$L = \{ a^n bbc^m \mid n, m > 0\}$$

   побудуйте скінченний автомат, що розпізнає $\Sigma^* \setminus L$.

9. Мінімізувати автомат, стан A – початковий, стани D,E – заключні



10. Мінімізувати автомат



11. Мінімізувати автомат, в якому функція переходів задана таблично, в цій таблиці символ

→ *- відмічає вхідний стан автомату,*

* *- заключний стан автомату,*

1 стовпчик таблиці – стани автомату,

1 стрічка – термінальні символи цього автомату,

на перетині ***i*** *стрічки* ($i = \overline{2..n}$, *n – кількість стрічок таблиці, i стрічка відповідає певному стану* ***q****) та стовпчика, що відповідає певному термінальному символу* ***a****, вказано стан, в який переходить автомат зі стану* ***q****, по дузі з міткою* ***a.***

	0	*1*
$\rightarrow A$	B	A
B	A	C
C	D	E
$*D$	D	A
E	D	F
F	G	E
G	F	G
H	G	D

12. Мінімізувати автомат, в якому функція переходів задана таблично:

	0	1
→A	B	E
B	C	F
*C	D	H
D	E	H
E	F	I
*F	G	B
G	H	B
H	I	C
*I	A	E

13. Побудувати автомат над алфавітом $\{a, b\}$, що розпізнає мову, яка
    а) – не містить ланцюжків abb
    б) – не містить входжень abb в жодне слово мови
    с) – не містить ланцюжків aa та bb

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Завдання 1. Побудувати скінченний автомат, що розпізнає мову породжену граматикою

1 $P = \{S \to 0A,\ S \to \Lambda,\ S \to 0,\ A \to 1\}$;

2 $P = \{S \to 101A,\ A \to 1A,\ A \to 0\}$;

3 $P = \{S \to 0A,\ S \to 1B,\ A \to 0,\ B \to 0\}$

4 $P = \{S \to 0A,\ A \to 01,\ S \to 0B,\ B \to 10\}$;

5 $P = \{S \to 1S,\ S \to 0,\ S \to B,\ B \to 01\}$;

6 $P = \{S \to B01,\ B \to 1B,\ B \to 0\}$;

7 $P = \{S \to 1A,\ A \to 1,\ A \to 0,\ S \to 0\}$;

8 $P = \{S \to 111S,\ S \to \Lambda\}$;

9 $P = \{S \to 0A,\ A \to 01B,\ S \to 0B,\ B \to 10\}$;

10 $P = \{S \to 0S,\ A \to A1,\ S \to A,\ A \to \Lambda\}$;

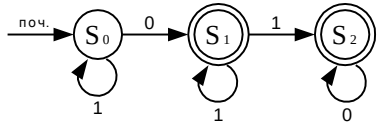
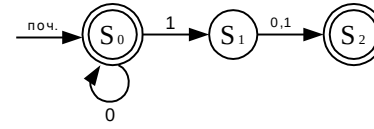
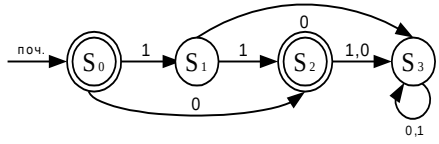
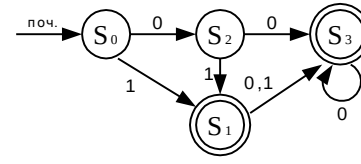
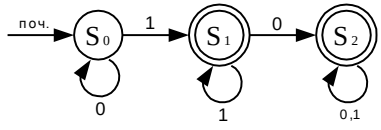
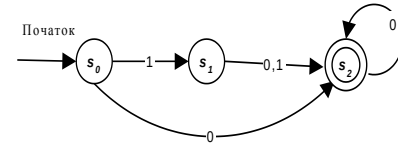
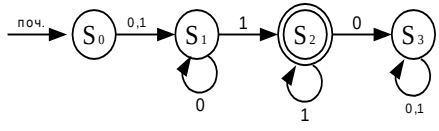
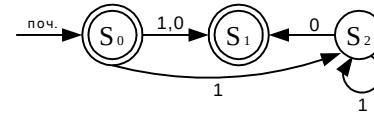
11 $P = \{S \to S1,\ S \to A,\ S \to 1,\ S \to \Lambda,\ A \to 0\}$;

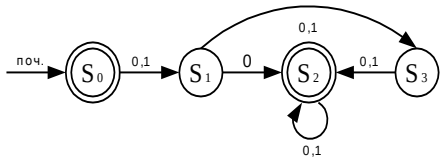
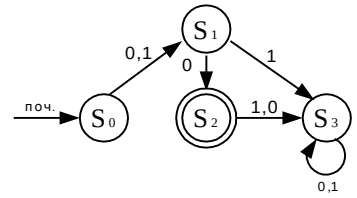
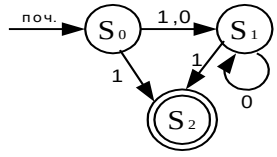
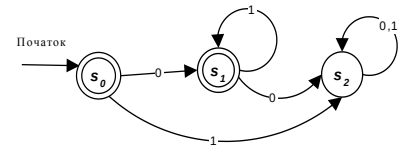
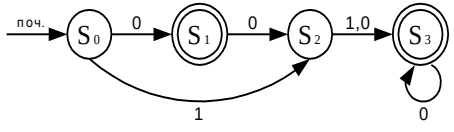
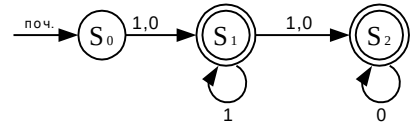
12 $P = \{S \to 1A,\ A \to 101,\ S \to 0B,\ B \to 010\}$;

13 $P = \{S \to 01A,\ A \to 00,\ S \to 0B,\ B \to 110\}$;

14 $P = \{S \to 1A,\ S \to 0,\ S \to \Lambda,\ B \to 1,\ A \to 0B\}$;

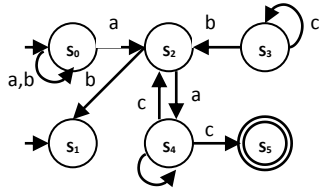
Завдання 2. Визначити граматику, що породжує автоматну мову, яка розпізнається наступним автоматом

1		8	
2		9	
3		10	
4		11	

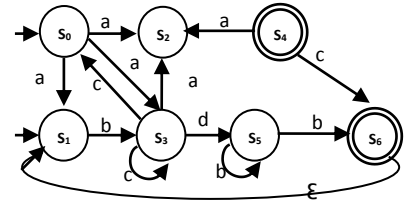
5	 поч. S0 S1 S2 S3 0,1 0 0,1 0,1 0,1	12	 S1 0,1 0 1 поч. S0 S2 1,0 S3 0,1
6	 поч. S0 1,0 S1 1 1 S2 0	13	 Початок s0 s1 s2 0 0 1 1 0,1
7	 поч. S0 0 S1 0 S2 1,0 S3 1 0	14	 поч. S0 1,0 S1 1,0 S2 1 0

Завдання 3. Видалити непродуктивні та недосяжні стани в автоматах та звести їх до детермінованого виду.

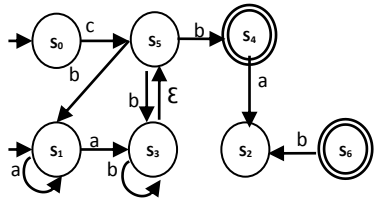
1



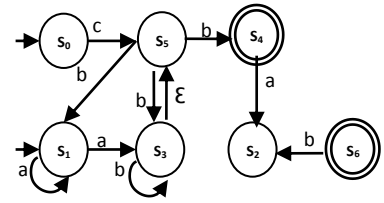
2



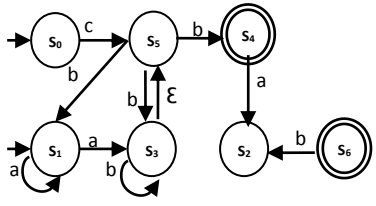
3.



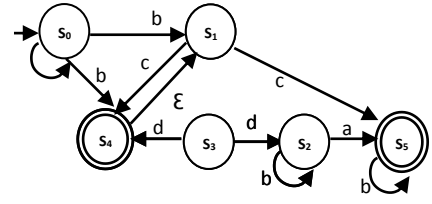
4



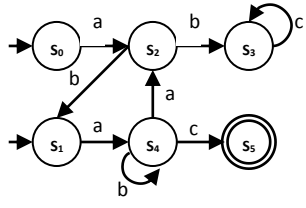
5



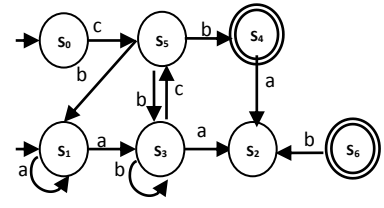
6

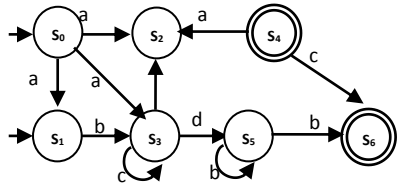
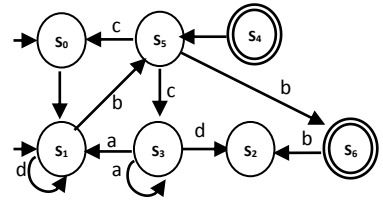
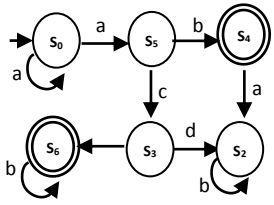
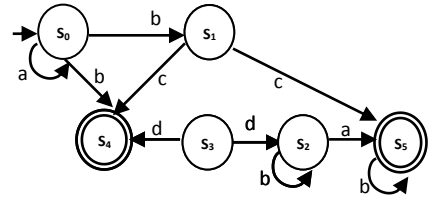
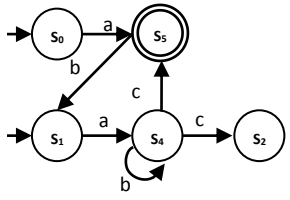
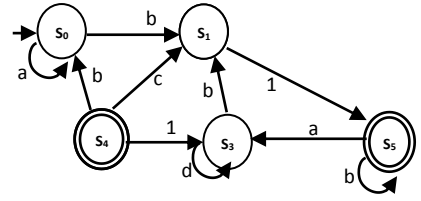


7



8



9 	**10** 
11 	**12** 
13 	**14** 

Завдання 4.

Мінімізувати отримані в завданні 3 детерміновані скінченні автомати.

Завдання 5.

Побудувати мову, яка є доповненням до мови, що допускається автоматом з завдання 4, до універсальної мови, заданої алфавітом цього автомату.

Розділ 3. РЕГУЛЯРНІ ВИРАЗИ

3.1. РЕГУЛЯРНІ ВИРАЗИ ТА РЕГУЛЯРНІ МОВИ.

До сих пір ми розглядали два способи опису формальної мови: граматики і автомати. Третій спосіб, часто найбільш зручний і компактний, - регулярні вирази. В них застосовуються операції: об'єднання, конкатенація, ітерація мов.

Нагадаємо деякі з них.

Нехай $A, B \subset \Sigma^*$,- де Σ алфавіт.

Визначення 3.1.1.

Конкатенацією множин A та B (позначають $A \cdot B$ або AB) називають множину всіх ланцюжків вигляду $\alpha\beta$, де $\alpha \in A$ і $\beta \in B$.

Визначення 3.1.2.

Ітерацією множини A (позначають A^*) називають множину, визначену індуктивно наступним чином:

$A^0 = \varepsilon$;

$A^n = AA^{n-1}$, $n \geq 1$;

$A^* = \bigcup_{n=0}^{\infty} A^n$, де ε - порожній ланцюжок.

Приклад 3.1.1.

Нехай $A = \{aa, bb\}$, $B = \{\varepsilon, a\}$.

Тоді, $AB = \{aa, bb, aaa, bba\}$ і

$A^* = \{\varepsilon, aa, bb, aaaa, aabb, bbaa, bbbb, aaaaaa, ...\}$.

Визначення 3.1.3.

Регулярний вираз над алфавітом Σ визначається рекурсивно так:

0 є регулярним виразом;

1 є регулярним виразом;

a є регулярним виразом, якщо $a \in \Sigma$;

$(e + f)$,

$(e \cdot f)$

e^* є регулярними виразами, якщо e і f - регулярні вирази.

Для зменшення кількості дужок у виразах введено наступний пріоритет вище введених операцій. Визначено, що операція * має більш високий пріоритет, ніж множення, а множення має більш високий пріоритет, ніж додавання. Замість $e \cdot f$ часто пишуть просто ef .

Приклад 3.1.2.

Нехай $\Sigma = \{a, b\}$. Тоді $((a \cdot b)^* \cdot (1 + a))$ є регулярним виразом над алфавітом Σ .

Визначення 3.1.4.

Кожний регулярний вираз e над алфавітом Σ **задає мову** над алфавітом Σ (позначається $L(e)$ де e – регулярний вираз), яка визначається рекурсивно так, якщо e, f – регулярні вирази:

$L(0) = \varnothing;$

$L(1) = \varepsilon;$

$L(a) = \{a\},$ *якщо* $a \in \Sigma;$

$L(e + f) = L(e) \cup L(f);$

$L(e \cdot f) = L(e) \cdot L(f);$

$L(e^*) = L(e)^*$

Замість $L(e)$ часто пишуть просто e.

Приклад 3.1.3.

Нехай $\Sigma = \{a, b\}$. Згідно визначення

$L((a \cdot b)^* \cdot (1 + a)) = \{(ab)^n \mid n \geq 0\} \bigcup \{(ab)^n a \mid n \geq 0\}$.

Такий вираз отриманий відкриттям дужок регулярного виразу, що представляє вказану мову

Отже, мова L називається ***регулярною,*** якщо вона задається деяким регулярним виразом. Для кожного регулярного виразу існує регулярна мова, а для довільної регулярної мови існують різні регулярні вирази.

Будемо говорити, що ***регулярні вирази рівні***, якщо вони задають однакові регулярні множини

Для будь-яких регулярних виразів e, f, g виконуються **наступні тотожності**:

$$e + f = f + e;$$

$$e + 0 = e;$$

$$e \cdot 1 = 1 \cdot e = e;$$

$$e \cdot 0 = 0 \cdot;$$

$$e + e = e$$

$$e + e^* = e^*$$

$$(e^*)^* = e^*$$

$$(e + f) + g = e + (f + g);$$

$$(e \cdot f) \cdot g = e \cdot (f \cdot g);$$

$$e \cdot (f + g) = e \cdot f + e \cdot g;$$

$$(f + g) \cdot e = f \cdot e + g \cdot e;$$

$$(1 + e + ee + ... + e^{n-1})(e^n)^* = e^*, \forall n \geq 1$$

$$(e^* + f)^* e^* = (e + f)^*$$

$$1 + e(fe)^* f = (ef)^*$$

Між регулярними виразами, автоматними мовами та праволінійними граматиками є зв'язок: всі вони описують один і той самий клас мов.

Ще в розділі 1 ми сформулювали твердження, що регулярні мови входять в клас праволінійних мов

$$\text{РЕГ} \subseteq \text{клас праволінійних мов.} \qquad (3.1.1)$$

В розділі 2 було розглянуто характеризацію автоматних мов і доведено, що скінченний автомат розпізнає мови, які породжуються праволінійними граматиками і навпаки, для кожної праволінійної мови можна побудувати відповідний автомат.

Доведення включення (3.1.1) випливає з теореми Кліні, яка встановлює взаємноодназначну відповідність між регулярними мовами та скінченними автоматами.

3.2. ЗВ'ЯЗОК МІЖ РЕГУЛЯРНИМИ ВИРАЗАМИ ТА СКІНЧЕННИМИ АВТОМАТАМИ. ТЕОРЕМА КЛІНІ.

Теорема Кліні.
Мова L регулярна тоді і тільки тоді, коли вона є автоматною.

Доведення теореми здійснюється шляхом
- визначенням алгоритмів побудови за регулярним виразом відповідного атомату і навпаки,
- за автоматом – регулярного виразу.

Розглянемо процедуру побудови скінченного недетермінованого автомату, який розпізнає регулярну мову, задану регулярним виразом.

Для регулярних мов $\emptyset, \{\lambda\}, \{a\}$ скінченні автомати, які допускають ці мови, представлені на рис 3.1.1.

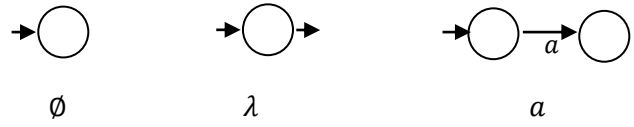


Рис.3.1.1. Скінченні автомати, які допускають регулярні мови $\emptyset, \{\lambda\}, \{a\}$

Використовуючи метод математичної індукції побудуємо скінченні недетерміновані автомати для регулярних виразів $e + f$, $e \cdot f$, e^* .

Нехай для регулярних мов $L(e)$ та $K(f)$, де e, f - регулярні вирази, еквівалентні їм скінченні недерміновані автомати $M_1 = \langle Q_1, \Sigma, \Delta_1, I_1 = \{q_{01}\}, F_1\rangle$ та $M_2 = \langle Q_2, \Sigma, \Delta_2, I_2 = \{q_{02}\}, F_2\rangle$, вже побудовані.

Звернемо увагу на те, що вхідні алфавіти цих автоматів збігаються (ми працюємо на множині мов в довільному, але фіксованому алфавіті Σ) і автомати не мають ні спільних вершин, ні спільних дуг.

1. Тоді скінченний автомат для *об'єднання мов* $L(e) \cup K(f)$, яке

описується регулярним виразом $e + f$, при збереженні всіх дуг і вершин автоматів M_1, M_2 будується шляхом

- додавання нового початкового стану s_0
- проведення з нього дуг з порожніми мітками в початковий стан q_{01} автомату M_1 та стан q_{02} автомату M_2
- об'єднання множин F_1, F_2заключних станів вершин автоматів M_1та M_2 відповідно.

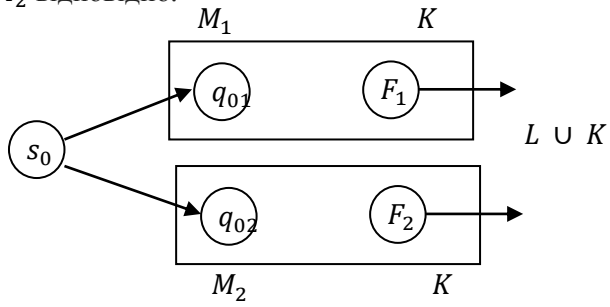


Рис.3.2.2.Скінченний автомат для розпізнавання об'єднання мов $L(e) \cup K(f)$ "паралельне з'єднання" автоматів

Отримуємо "паралельне з'єднання" автоматів для об'єднання мов.
Довільний ланцюжок x, який допускається об'єднаним автоматом може бути представленим наступним чином: або $x = \lambda x_1$ або як $x = \lambda x_2$, де x_1, ланцюжок що допускається автоматом M_1, а x_2 - автоматом M_2.
По мітці λ автомат переходить зі стану s_0 в стан q_{01} і далі ланцюжок x_1 переводить в один з заключних станів множини F_1, або по мітці λ автомат переходить зі стану s_0 в стан q_{02} і далі ланцюжок x_2 переводить в один з заключних станів множини F_2.

2. Скінченний автомат *для конкатенації мов* $L(e) \cdot K(f)$, яке описується регулярним виразом $e \cdot f$, при збереженні всіх дуг і вершин автоматів M_1, M_2 будується шляхом

   - оголошення стану q_{01} новим початковим станом "послідовного з'єднання" автоматів M_1та M_2
   - множину F_2 оголошуємо множиною заключних станів автомату

$M_1 \cdot M_2$

- кожен заключний стан з F_1 автомату M_1 з'єднуємо дугою з міткою λ з початковим станом q_{02} автомату M_2.

Тим самим ми визначили операцію конкатенації скінченних автоматів $M_1 \cdot M_2$, яку можна інтерпретувати як їх послідовне з'єднання (див рис. 3.2.3).

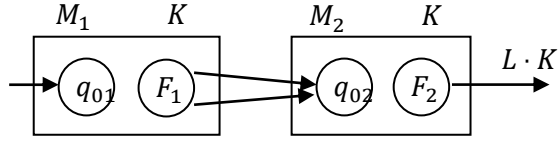


Рис.3.2.3.Скінченний автомат для розпізнавання конкатенації мов $L(e) \cdot K(f)$ " послідовне з'єднання" автоматів

3. Скінченний автомат *для ітерації мови* $(L(e))^*$, яка описується регулярним виразом e^* будується наступним чином.
   Потрібно:

- ввести нове початкову вершину s_0
- ввести нову заключну вершину t, провівши порожню дугу з s_0 в t;
- провести порожні дуги з нової початкової вершини s_0в колишню початкову вершину q_0 автомату, для якого будується ітерація
- провести з кожної заключної вершини автомату, для якого будується ітерація аналізованої ним мови, пусті дуги в нову заключну вершину t і колишню початкову вершину q_0.

Побудова скінченного автомату для ітерації мови $(L(e))^*$ зображено на рис. 3.2.4.

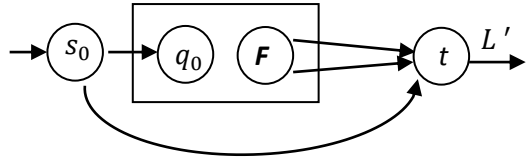


Рис.3.2.4. Автомат для розпізнавання ітерації мови $(L(e))^*$

Зауваження.

Звернемо увагу на те, що в загальному випадку при побудові ітерації не можна обійтися без додавання нових початкових і заключних вершин. Розглянемо в зв'язку з цим побудову автомату для ітерації мови, що допускається автоматом на рис. 3.2.5 а.

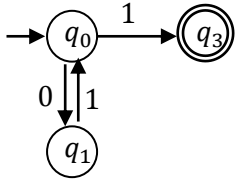


Рис. 3.2.5 а

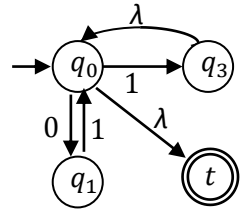


Рис. 3.2.5.б

Якщо ми побудуємо автомат для ітерації так, як показано на рис. 3.2.5 б, не вводячи нову початкову вершину, то цей автомат допускатиме, зокрема, всі ланцюжки мови (01) *. Однак ці ланцюжки не містяться в ітерації вихідної мови. Зокрема, $01 \notin ((01)^* 1)^*$. Дійсно, будь-який ланцюжок мови вихідного автомату задається регулярним виразом $(01)^* 1$ і є відповідно 1 або ланцюжок, що закінчується 11, а ітерація цієї 1^* або ланцюжки які будуть закінчуватись 11. тому 01 не належить до ітерації мови, що розпізнається автоматом 3.2.5.а, Тоді як автомат 3.2.5.б допускає цей ланцюжок.

Аналогічні приклади можна побудувати і для випадку, коли при побудові ітерації не додавалась додаткова заключна вершина.

Правильно побудований автомат для ітерації мови, що допускається автоматом 3.2.5.а зображено на рис 3.2.6.

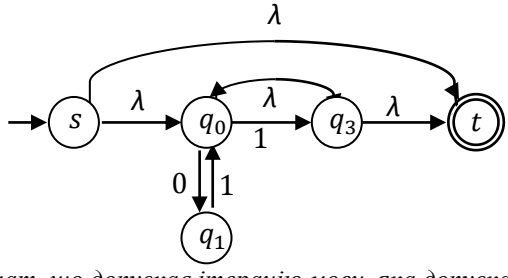


Рис.3.2.6. Автомат, що допускає ітерацію мови, яка допускається автоматом 3.2.5.а

Алгоритм побудови регулярного виразу для заданого скінченного автомату.

Алгоритм базується на побудові узагальненого скінченного автомату на основі заданого недермінованого.

Визначення 3.1.5.

Узагальненим скінченним автоматом називається аналог скінченного автомату з одним вхідним та одним вихідним станом, де переходи помічені не словами, а регулярними виразами. *Мітка шляху* такого автомату - добуток регулярних виразів на переходах даного шляху. Слово *w допускається* узагальненим скінченним автоматом, якщо воно належить мові, що задається міткою деякого успішного шляху (шлях закінчується у заключному стані).

Нехай автоматна мова L розпізнається деяким (недетермінованим) скінченним автоматом з одним початковим станом і одним заключним станом. Існує еквівалентний йому узагальнений скінченний автомат $(Q, \Sigma, \Delta, \{q_1\}, \{q_2\})$, де $q_1 \neq q_2$.

Перехід зі стану q_1 в стан q_2 по регулярному виразу r будемо позначати так: $\langle q_1, r, q_2 \rangle$. Якщо є декілька переходів зі спільним початком і спільним кінцем (такі переходи називаються *паралельними*), замінимо їх на один перехід, використовуючи операцію "+".

Наприклад, два переходи

$$\langle q_1, r_1, q_2 \rangle \text{ і } \langle q_1, r_2, q_2 \rangle$$

замінимо на один $\langle q_1, r_1 + r_2, q_2 \rangle$.

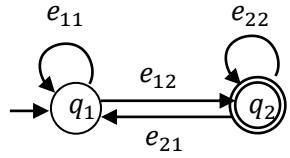
Видалимо по черзі всі стани, крім q_1 і q_2. При видаленні стану q потрібно для кожного переходу виду $\langle p_1, f_1, q \rangle$, де $p_1 \neq q$, і для кожного переходу виду $\langle q, f_2, p_2 \rangle$, де $p_2 \neq q$, додати перехід $\langle p_1, f_1 r^* f_2, p_2 \rangle$, де регулярний вираз r - мітка переходу з q в q (якщо немає переходу з q в q, то потрібно додати перехід $\langle p_1, f_1 f_2, p_2 \rangle$, і знову всюди замінити паралельні

переходи на один перехід, використовуючи операцію “+”.

Після видалення всіх станів, крім q_1 і q_2, отримаємо узагальненій скінченний автомат

$$M(\{q_1, q_2\}, \Sigma, \Delta', \{q_1\}, \{q_2\}),$$

де $\Delta' = \{\langle q_1, e_{11}, q_1\rangle, \langle q_1, e_{12}, q_2\rangle, \langle q_2, e_{21}, q_1\rangle, \langle q_2, e_{22}, q_2\rangle\}$.



Очевидно, що $L = L(e_{11}^* e_{12} (e_{22} + e_{21} e_{11}^* e_{12})^*)$

Приклад 3.1.1.

Розглянемо мову, яка розпізнається скінченним автоматом (рис. 3.1.1.)

$$M(\{q_1, q_2, q_3, q_4\}, \Sigma, \Delta, \{q_1\}, \{q_2\}),$$

де

$\Sigma = \{a, b, c\}$ і

$\Delta = \{\langle q_1, a, q_4\rangle, \langle q_2, cb, q_2\rangle,$

$\langle q_2, bac, q_4\rangle, \langle q_3, a, q_1\rangle, \langle q_3, c, q_2\rangle, \langle q_3, c, q_4\rangle, \langle q_4, \varepsilon, q_3\rangle, \langle q_4, b, q_4\rangle\}$

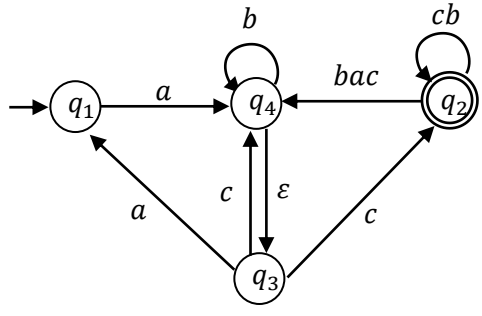


Рис. 3.1.1.

Та ж мова породжується узагальненим скінченним автоматом (Рис. 3.1.2.).

$$M(\{q_1, q_2, q_3, q_4\}, \Sigma, \Delta_1, \{q_1\}, \{q_2\}),$$

де $\Delta = \{\langle q_1, a, q_4\rangle, \langle q_2, cb, q_2\rangle, \langle q_2, bac, q_4\rangle,$
$\langle q_3, a, q_1\rangle, \langle q_3, c, q_2\rangle, \langle q_3, c, q_4\rangle, \langle q_4, 1, q_3\rangle, \langle q_4, b, q_4\rangle\}$

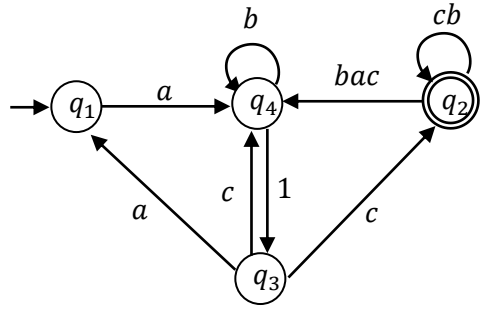


Рис. 3.1.2.

Після видалення стану q_3 отримуємо узагальнений скінченний автомат (Рис. 3.1.3)

$$M(\{q_1, q_2, q_4\}, \Sigma, \Delta_2, \{q_1\}, \{q_2\}),$$

де $\Delta_2 = \{\langle q_1, a, q_4\rangle, \langle q_2, cb, q_2\rangle, \langle q_2, bac, q_4\rangle,$
$\langle q_4, 1\cdot a, q_1\rangle, \langle q_4, 1\cdot c, q_2\rangle, \langle q_4, b+1\cdot c, q_4\rangle\}$

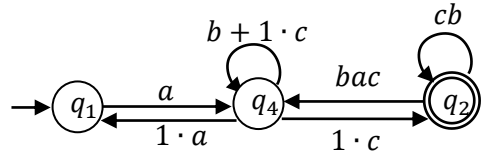


Рис. 3.1.3

Можна спростити регулярні вирази і отримати (рис. 3.1.4).

$\Delta'_2 = \{\langle q_1, a, q_4\rangle, \langle q_2, cb, q_2\rangle,$
$\langle q_2, bac, q_4\rangle, \langle q_4, a, q_1\rangle, \langle q_4, c, q_2\rangle, \langle q_4, b+c, q_4\rangle\}$

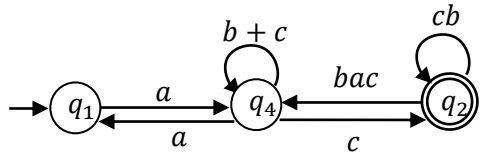


Рис. 3.1.4.

Після видалення стану q_4 і спрощення регулярних виразів отримуємо узагальненій скінченній автомат (рис. 3.1.5).

$$M(\{q_1, q_2\}, \Sigma, \Delta_3, \{q_1\}, \{q_2\}),$$

$$\Delta_3 = \{\langle q_1, a(b+c)^* a, q_1\rangle, \langle q_1, a(b+c)^* c, q_2\rangle,$$

$$\langle q_2, bac(b+c)^* a, q_1\rangle, \langle q_2, cb + bac(b+c)^* c, q_2\rangle,\}$$

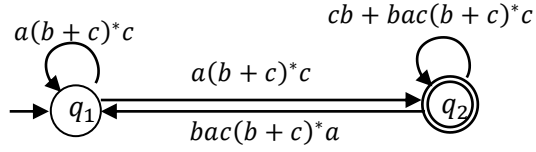


Рис. 3.1.5.

Отже, мова *L(M)* задається регулярним виразом

$$(a(b+c)^* a)^* a(b+c)^* c(cb + bac(b+c)^* c + bac(b+c)^* a(a(b+c)^* a)^* a(b+c)^* c)^*$$

Спростивши цей регулярний вираз, отримаємо

$$L(M) = L(a(aa + b + c + c(cb)^* bac)^* c(cb)^*)$$

Приклад 3.1.2.

Для регулярного виразу $a^* + ba$ над алфавітом $T = \{a, b\}$ побудувати породжуючу його автоматну граматику та еквівалентний граматиці скінченний автомат:

Запишем мову, що задає регулярний вираз $L(a^* + ba) = \{ \{ba\} \cup \{a^n \mid n = 0,1,...\}\}$ та побудуємо автоматну граматику:

$G = (N, T, P, S)$, $N = \{A, B, S\}$, $T = \{a, b\}$, S – початковий символ,

$P = \{S \to aA,$

$A \to aA,$

$A \to a,$

$S \to \varepsilon,$

$S \to bB,$

$B \to a\}$

Побудуємо скінченний автомат, еквівалентний граматиці. Множину термінальних символів зробимо вхідним алфавітом автомату $\Sigma = \{a, b\}$. Множині нетермінальних символів $N = \{A, B, S\}$ поставим у відповідність множину станів $Q = \{q_1, q_2, q_3, q_0\}$, для цього правила виду $A \to a$ замінимо на два правила - . $A \to aZ$, $Z \to \varepsilon$, де Z - новий додатково введений нетермінал, $N = \{A, B, S, Z\}$. Початковий символ S зробимо відповідним до початкового стану q_0, зробимо заключним стан q_3, він відповідає нетерміналові Z, для якого задано ε - правило $Z \to \varepsilon$. Продукціям граматики поставимо у відповідність переходи автомату, враховуючи що q_1 відповідає нетерміналові A, q_2 - нетерміналові B.

$$\Delta = \{\langle q_0, a, q_1\rangle, \langle q_1, a, q_1\rangle, \langle q_1, a, q_3\rangle, \langle q_0, \varepsilon, q_0\rangle, \langle q_0, b, q_2\rangle, \langle q_2, a, q_3\rangle\}$$

Наслідок 3.2.1.

з теореми про доповнення автоматних мов та теореми Кліні:

операції об'єднання та доповнення регулярних мов не виводять за межі класу регулярних мов.

Для зручності опису формул прийнято наступне позначення для операції доповнення $\overline{L}$ - доповнення до множини *L*.

Оскільки кожна автоматна мова є регулярною мовою (з теореми Кліні), а доповнення до автоматної мови є автоматною мовою (теорема 2.7.1), то доповнення до регулярної мови є регулярною мовою. Отже,

- *об'єднання регулярних мов є регулярною мовою*
- *доповнення регулярних мов є регулярною мовою,*

тобто операції об'єднання та доповнення не виводять за межі класу регулярних мов.

Для множин справедливі наступні співвідношення :

1. Перетин множин може бути преставлений як

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

2. Різниця множин представляється як

$$L_1 \setminus L_2 = L_1 \cap \overline{L_2}$$

Співвідношення 1 і 2 представляють перетин та різницю регулярних мов через операції доповнення та об'єднання. Оскільки, перетин та різниця регулярних мов може бути виражена через операції об'єднання та доповнення, то вони теж не виводять за межі класу регулярних мов.

Отже,

- *Перетин регулярних мов є регулярною мовою*
- *Різниця регулярних мов є регулярною мовою.*

Тому

клас регулярних мов замкнений відносно операцій перетину та різниці.

3.3. ЛЕМА ПРО РОЗРОСТАННЯ ДЛЯ РЕГУЛЯРНИХ МОВ

У теорії формальних мов велике значення мають твердження, в яких формулюється необхідна умова приналежності мови до того чи іншого класу мов. Ці твердження відомі в літературі під назвою лем про розростання (або лем про "накачування"). За допомогою цих лем вдається довести, що та чи інша мова не є мовою даного класу, наприклад, не є регулярною, не є контекстно-вільною і т.п. Доводити подібного роду "негативні" твердження набагато важче, ніж "позитивні" (що мова є мовою даного класу С), бо в останньому випадку достатньо придумати будь-яку граматику відповідного класу, яка породжує дану мову, тоді як в першому потрібно якимось чином довести, що не існує граматики цього класу, яка породжує мову.

Застосування лем про розростання полягає в наступному: довівши, що мова не задовільняє умові леми про розростання, ми можемо бути впевнені в тому, що вона не належить відповідному класу мов.

Ми розглянемо подібного роду твердження для класів регулярних і контекстно-вільних мов як найбільш часто вживаних формальних моделей мов програмування.

Почнемо з леми про розростання для регулярних мов. Ця лема стверджує, що будь-яка регулярна мова допускає представлення всіх своїх ланцюжків у вигляді з'єднання трьох ланцюжків, причому середній ланцюжок з цих трьох не є порожнім, обмеженим по довжині, і його "накачка " - повторення будь-яке число раз - або викидання НЕ ВИВОДИТЬ за межі мови (тобто дає ланцюжки, що належать даній регулярній мові).

Теорема 3.3.1.

Якщо $\boldsymbol{L}$ - регулярна мова, то існує натуральна константа k_L (залежна від L), така, що для будь-якого ланцюжка $x \in L$, довжина якого не менше k_L, x допускає представлення у вигляді $x = uvw$, де $v \neq \lambda$ і $|v| \leq k_L$, причому для будь-якого $n \geq 0$ ланцюжок $x_n = u\, v^n\, w \in L$.

Оскільки мова L регулярна, то, згідно з теоремою Кліні (див. теорему 3.2.1), існує скінченний автомат $M = (V, Q, \delta, q_0, F)$, що допускає L, тобто $L = L(M)$ (на основі теореми 2.6.1 про детермінізацію можна вважати М детермінірованим автоматом). Поклавши $k_L = |Q|$, тобто ввівши константу k_L

рівну кількості станів скінченного автомату М, фіксуємо довільно вибраний ланцюжок $x \in L$, довжина r якого не менше ніж k_L , тобто $r \geq k_L$. Зрозуміло, що $r > 0$, тобто ланцюжок x не є порожнім, і ми можемо покласти

$$x = x(1)x(2) \ldots x(r),\ r > 0.$$

Оскільки скінченний автомат М є детермінованим, то існує єдиний шлях, що веде з початкового стану q_0 в один із заключних станів q_f, на якому читається x (рис. 3.3.1).

Рис. 3.3.1.Шлях автомату, на якому допускається ланцюжок x

Оскільки довжина

$$r \geq k_L = |Q| \qquad (1)$$

ланцюжка x не менша числа станів М, то, оскільки число вершин в будь-якому шляху рівно на одиницю більше числа дуг в цьому шляху (у нашому випадку ми маємо r дуг, помічених $x(1), x(2), \ldots, x(r)$ відповідно), то число вершин у розглянутому вище шляху $r + 1$, а отже строго більше, ніж число всіх станів автомату $|Q|$ (див. формулу 1) Це означає, що хоча б одна з вершин даного шляху повторюється і вона, таким чином, міститься в деякому замкненому контурі (петлі). Позначимо цю вершину через p. Тоді шлях, на якому допускається ланцюжок x, розбивається на три частини (рис.3.3.2)

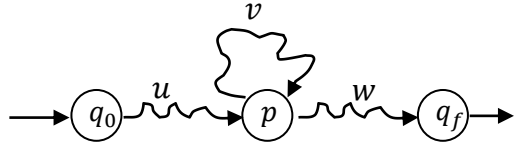


Рис.3.3.2 Шлях допуску ланцюжка x, що містить петлю в стані p

Перша частина – шлях з q_0 в p, друга - контур, що проходить через p, третя - шлях з p в q_f

Позначимо через u ланцюжок, що читається на першій частині шляху, через v - ланцюжок, що читається на контурі, а через w ланцюжок, що читається на третій частині шляху, отримаємо $x = uvw$.

Але тепер цілком очевидно, що контур (на якому лежить вершина p) можна пройти (по дорозі з q_0 в q_f) будь-яке число n (при $n > 0$) раз або жодного разу. У першому випадку на цьому шляху буде прочитаний ланцюжок $uv^n w$ при $n > 0$, а в другому - ланцюжок uw, а це означає, що ланцюжок $uv^n w$ допускається автоматом M, а отже належть мові $L(M)$.

Розглянемо тепер деякі приклади доказів нерегулярності мови з використанням леми про розростання. Стандартний хід міркувань при вирішенні таких завдань полягає в припущенні регулярності даної мови і в наступному аналізі можливості представити будь-який досить довгий ланцюжок мови у вигляді, зазначеному в умові леми про розростання. Аналізуючи всі можливі випадки розміщення підланцюжка v, який ми будемо "накачувати", ми повинні отримати протиріччя.

Приклад 3.3.1.

Доведемо нерегулярність мови
$L(M) = \{ a^n b^n, n \geq 0 \}$.

Вибираючи n настільки великим, щоб воно перевершувало k_L (константу леми), одержуємо наступні можливі випадки розміщення середнього v в ланцюжку $a^n b^n$. Можливі наступні варіанти:

1. $v = a^s$, $s < n$, тобто "накачуваний" підланцюжок v цілком розташовується в "зоні символів a " (рис. 3.3.3. а).

Ясно, що накачування в цьому випадку виведе за межі мови, так як при повторенні ланцюжка v число символів a буде необмежено зростати, а число символів b залишатиметься колишнім

2. $v = b^s, s < n$, тобто "накачуваний" підланцюжок v цілком розташовується в "зоні символів b " (рис. 3.3.3. б).

Накачування неможливе з тієї ж причини, що і в попередньому випадку.

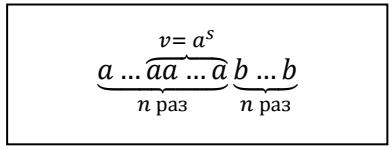


Рис.3.3.3. а

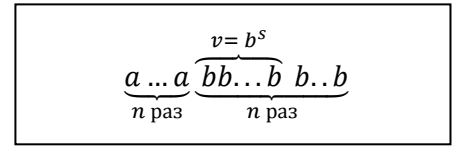


рис. 3.3.3. б

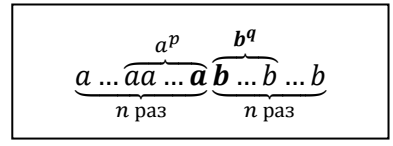


Рис.3.3.3. в

3. $v = a^p b^q$, де $0 < p < n, 0 < q < n$, тобто "накачуваний" підланцюжок v розташовується на стику зон символів a і b (рис.3.3.3. в)

У цьому випадку при накачуванні виникне входження підланцюжка ab в слово, яке, вже не належить мові L.

Таким чином, мова $a^n b^n$ нерегулярна.

Приклад 3.3.2.

Доведемо, що мова $L_2 = L_1{}^* = \{(a^n b^n)^m,\ n > 0, m \geq 0\}$, нерегулярна.

Вважаючи, що мова регулярна, візьмемо слово $(a^n b^n)^m,\ m \geq 1$ для достатньо великого n.

Застосовуючи таку ж схему міркувань, що і в прикладі 3.3.1, легко переконуємося в тому, що ланцюжок v не може складатися з одних символів a чи з одних символів b.

Нехай $v = a^r b^s$, де $r + s < n$. Тоді, повторивши ланцюжок v два рази, отримаємо слово $a^{n-r}\, a^r\, b^s\, a^r\, b^s b^{n-s}$. Якщо $r \neq s$, то ланцюжок такого виду не може належати мові L_2 так як в будь-якому слові цієї мови за підланцюжком символів a має слідувати підланцюжок символів b такої ж довжини. Але навіть якщо $r = s$, то отримаємо підланцюжок $a^n b^s$, де $s < n$ (або $a^r b^n$, r< n), що також суперечить визначенню мови L_2.

Аналогічно розглядається випадок $v = b^r a^s$, де $r + s < n$

Зауважимо, що в силу обмеження по довжині на ланцюжок v ніякі способи її розміщення в зазначеному ланцюжку мови L_2, крім описаних вище, не можливі.

Корисно мати на увазі наслідок з леми про розростання.

Наслідок 3.3.1.

Якщо L - нескінченна регулярна мова, то в ній знайдеться послідовність слів, довжини яких утворюють зростаючу арифметичну прогресію.

В якості такої послідовності можна взяти послідовність слів x_n з формулювання леми про розростання, таких що $x_n = uv^n w$. Їх довжини утворюють арифметичну прогресію зі знаменником $|v|$ (довжина "накачується" за рахунок середнього підланцюжка).

Звідси легко випливає висновок про те, що не є регулярними наступні мови:

1) $\{a^{n^2}, n > 0\}$ - мова в однобуквеному алфавіті, довжини слів якої є повними квадратами;

2) $\{a^p, p$ - просте число$\}$.

На закінчення обговоримо ще одну схему доведення нерегулярності мови з використанням як леми про розростання, так і алгебраїчних властивостей класу регулярних мов, які були встановлені в наслідку 3.2.1.

Приклад 3.3.3.

Доведемо нерегулярність мови правильних дужкових структур, породжуваних КС-граматикою

$G_8 = (T = \{(,)\},\ N = \{S\},\ P = \{S \to ()|(S)|SS\}, S)$

Для доведення поступимо так: розглянемо перетин даної мови з регулярною мовою (*) *, яка містить всі ланцюжки виду $(^m\ )^n$ для будь-яких натуральних $n, m \geq 0$. Оскільки кожна правильна дужкова структура містить однакове число символів "(" і ")", то зазначений перетин буде мовою

$\{(^n\ \ )^n, n \geq 0\ \}$. Якщо позначити через a "відкриваючу дужку" (символ "("), а через b "закриваючу дужку" (символ ")"), то отримаємо мову $L_1 = a^n b^n$, яка, як доведено в прикладі 3.3.1, не є регулярною.

Отже, припускаючи регулярність мови правильних дужкових структур, ми змушені будемо допустити і регулярность мови $L_1 = a^n b^n$, так як перетин регулярних мов в силу наслідку 3.2.1 є регулярною мовою. Отримане протиріччя і доводить нерегулярність мови правильних дужкових виразів. Зауважимо, що доведення цього факту з використанням однієї лише леми про розростання було б досить важким.

Отже, можна вивести такий "рецепт": якщо виникають суттєві труднощі в доведенні нерегулярності якої-небудь мови за допомогою тільки леми про розростання, то можна спробувати перетнути цю мову з деякою регулярною мовою так, щоб нерегулярність перетину легко доводилась з використанням леми про розростання.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення регулярного виразу.
2. Дайте визначення мови, заданої регулярним виразом.
3. Які операції використовуються у регулярних виразах та який їх пріоритет?.
4. Що ви можете сказати про операції об’єднання, доповнення, перетин та різницю регулярних мов?
5. Як зв‘язані регулярні вирази, регулярні множини, праволінійні граматики та скінченні автомати?
6. Що таке узагальнений скінченний автомат?
7. Для чого застосовується лема про накачку регулярних мов?
8. Яким чином операції над регулярними мовами в комплексі з лемою про накачку використовуються для встановлення факту регулярності мови?

ВПРАВИ ТА ЗАДАЧІ ДО РОЗДІЛУ 3.

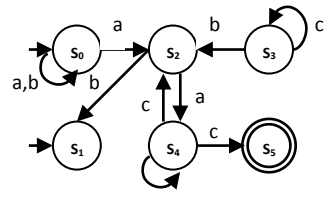
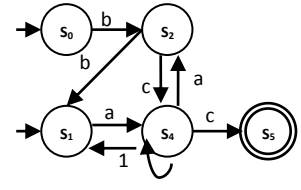
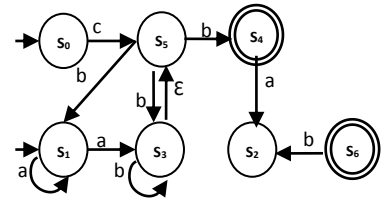
1. Визначте, чи належить слово 1010 мові, яка визначена регулярним виразом 10*(10+01)00*
2. Задайте всі можливі ланцюжки десяткових цифр, які представляють числа, що діляться на 5 (такі числа закінчуються або 0 або 5):
   а) недетермінованим скінченним автоматом;
   б) детермінованим скінченним автоматом;
   в) регулярним виразом.
3. Побудуйте регулярний вираз, який визначає в {a,b,c} всі ланцюжки, які не містять входжень ланцюжка $aacb$. Скористайтесь теоремою про доповнення автоматних мов.
4. Побудуйте регулярний вираз і детермінований скінченний автомат, який задає над алфавітом {0, 1}:
   а) мову, всі слова якої починаються символом 0 і містять, по крайній мірі, два символа 1.
   б) мову, всі слова якої не містять двох одиниць підряд.
5. Побудуйте детермінований скінченний автомат, який розпізнає перетин двох таких регулярних мов над словником {a, b, c}, які задаються регулярними виразами R1=a*(b+c)* та R2=a (b*+c*)
6. Чи співпадають регулярні мови, задані регулярними виразами a*(a*+b)*a та aa*(a+b)*a?
7. Доведіть, що мова над словником $\{a, b\}$, що містить всі ланцюжки виду $a^n b^m$, де $m < n$; $n, m \geq 0$, така що число входжень символа a більше числа входжень символа b, не регулярна.
8. Доведіть, що мова над словником $\{a, b\}$, що містить всі ланцюжки виду $a^m b a^n$, де $0 < m < n$; не регулярна.
9. Доведіть, що мова над словником $\{a, b\}$, що містить всі ланцюжки виду $ww, w \in \{a, b\}^*$ не регулярна.
10. Доведіть, що ((ab)*)*= (ab)*, побудувавши мінімальні детерміновані скінченні автомати, які розпізнають ці регулярні мови.

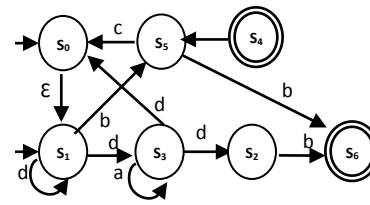
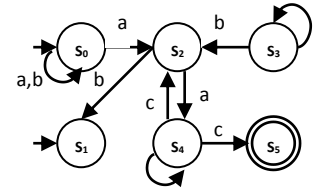
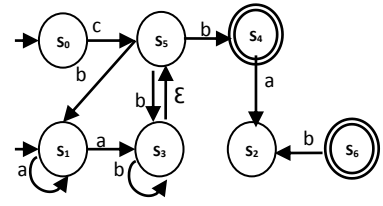
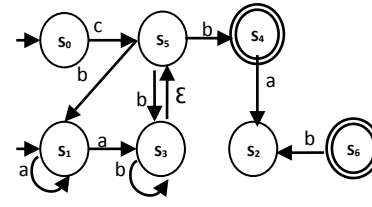
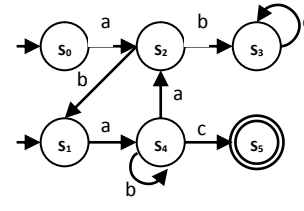
ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

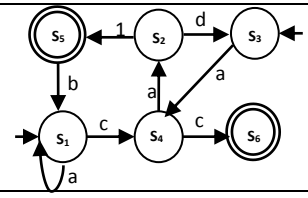
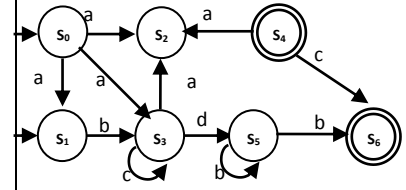
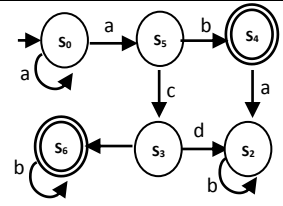
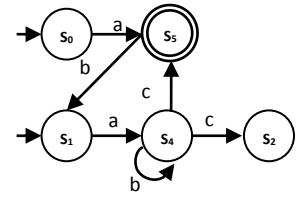
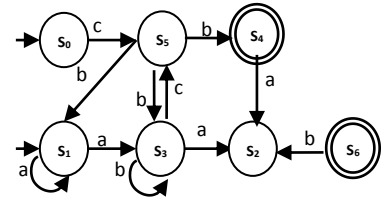
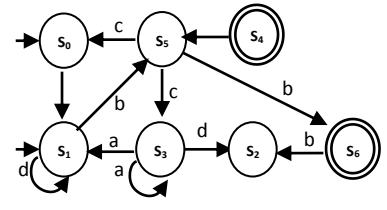
1. Для регулярного виразу R над алфавітом $T = \{a, b\}$ побудувати породжуючу його автоматну граматику та еквівалентний граматиці скінченний автомат.
2. Побудуйте регулярний вираз і недетермінований скінченний автомат з вхідним алфавітом $\{a, b, c\}$, який розпізнає всі ланцюжки α
3. Для заданого недетермінованого скінченного автомата *M* побудувати узагальнений скінченний автомат і записати регулярний вираз для мови, що ним розпізнається. Спростити отриманий регулярний вираз.

Варіанти завдань взяти з табл. 3.1.

Таблиця 3.1.

Вар.	R	α	M
1	$b(ba+b)^* b$	точно з одним входженням символу b	
2	$(ab+b)^* ba + ab$	не більше ніж з трьома входженнями символу b	
3	$(a+b)^* ba(a+b)$	не менш ніж з двома входженнями символу b	

4	$(a+b)^* ab(a+b)^*$	які починаються та закінчуються однаковими символами	
5	$(ab+b)^* ab(a+ab)^*$	З непарною кількістю символів b і будь-якою кількістю символів a	
6	$(ab+b)^* ba+(a+b)^*$	З парною кількістю a і непарною кількістю b . Побудувати відповідну автомату граматику.	
7	$(ab+b)^* (ba+a+b)^*$	кожне слово якої починає та закінчується різними символами.	
8	$(ab+b)(ba+(a+b)^*)^*$	кожне слово якої парної довжини.	

9	$(ab+a)^*(ba+a)^*$	Кількість букв c в слові кратна 2	
10	$(abb+1)^*b+a+(a+b)^*$	кожне слово якої починає та закінчується однаковим символом.	
11	$(a+b+b)^*a+(a+b^*)^*$	кожне слово якої закінчується двома однаковими символами.	
12	$(ab+b^*)(ba^*+a+b^*)^*$	З однаковою кількістю a та b	
13	$(a+ba)(b^*a+a^*+b^*)^*$	З однаковою кількістю a та b та непарною кількістю c	
14	$(a+b+1)^*b+a^*+(a+b)$	Кількість букв c в слові кратна 3	

Розділ 4. ЛЕКСИЧНИЙ АНАЛІЗ

4.1. ЛЕКСИЧНИЙ ТА СИНТАКСИЧНИЙ АНАЛІЗИ ДЛЯ ФОРМАЛЬНИХ МОВ.

Дана частина присвячена обговоренню наступної задачі: по даній КВ-граматиці G і ланцюжку w над основним алфавітом цієї граматики визначити, чи належить ланцюжок мові L(G). Якщо $w \in L(G)$, то необхідно видати вивід цього ланцюжка, наприклад побудувати його дерево виводу в граматиці G. Якщо ж $w \notin L(G)$, то бажано засобами граматики вказати передбачувану помилку (помилки), що призводить до неналежності w до L(G). Це задача називається задачею синтаксичного аналізу. Необхідність розв'язання такої задачі виникає в різних застосуваннях, зокрема при проектуванні компіляторів.

Компілятор - це програма, яка переводить ланцюжок, що належить одній мові (вихідній) в семантично еквівалентний ланцюжок іншої мови (цільової). Найчастіше вихідною мовою є мова програмування високого рівня (зрозуміла людині), а цільовою - мова машинних команд (зрозуміла комп'ютеру).

При цьому компілятору необхідно вирішити задачу синтаксичного аналізу вхідної ланцюжка. Найбільш поширена технологія створення компіляторів така, що рішення задачі синтаксичного аналізу ділиться на дві частини, перша виконується в блоці лексичного аналізу, друга в блоці синтаксичного аналізу.

Блок лексичного аналізу перетворює вихідну програму, написану мовою високого рівня, в ланцюжок так званих токенів або видає повідомлення про помилки.

Блок синтаксичного аналізу аналізує ланцюжок токенів, який надійшов на вхід, і видає інформацію про структуру ланцюжка (наприклад, дерево виводу) або повідомлення про помилки. Цей блок є, як правило, основою компілятора. Принципові рішення, прийняті при проектуванні синтаксичного блоку, в значній мірі визначають структуру і функціональні можливості інших блоків.

Стадія аналізу вихідної програми завершується в блоці семантичного аналізу. В цьому блоці відбувається обчислення деяких семантичних характеристик об'єктів програми, що компілюється, (наприклад, типів ідентифікаторів) і перевірка на наявність ряду семантичних помилок. Після проходження цього блоку компілятор володіє всією необхідною інформацією про вихідну програму і переходить в стадію реалізації, де здійснюється

генерація і оптимізація коду. На рис. 4.1.1 наведена спрощена модель компілятора, в якій вся друга стадія зображена у вигляді одного блоку.

В даному розділі будуть розглянуті методи організації роботи лексичного блоку .

Необхідно відзначити, що побудова лексичного і синтаксичного аналізаторів заснована на нетривіальних, але ясних і досить легко програмованих алгоритмах. Існує клас програм, які називаються «компіляторами компіляторів», призначених для генерації аналізаторів по граматиці. Як метод синтаксичного аналізу зазвичай вибирається LR-аналіз . Мабуть, найвідомішим в даний час генератором лексичних аналізаторів є F1ex, а генератором синтаксичних аналізаторів - Bison (а також їхні попередники Lex і Yacc відповідно).

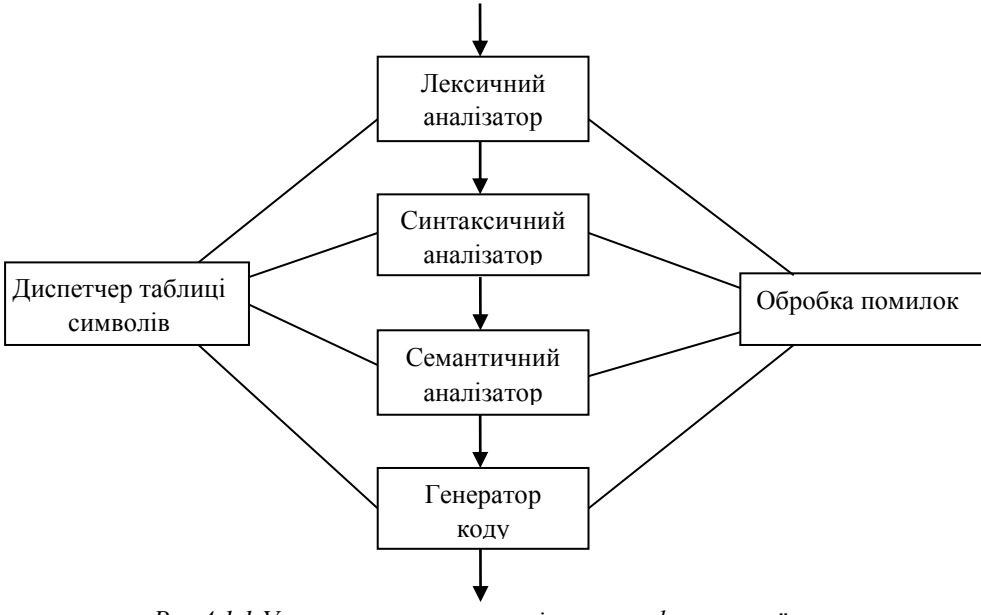


Рис.4.1.1.Узагальнена схема компілятора з формальної мови

Однак навіть при використанні подібних програм без «ручного» доведення компілятора все одно не обійтися, а його проведення вимагає хорошого знання того, як і коли працюють різні методи аналізу.

4.2. ОСНОВНІ ПОНЯТТЯ ЛЕКСИЧНОГО АНАЛІЗУ.

Задача лексичного аналізу полягає у виділенні у вхідному ланцюжку (тексті програми) мінімальних змістовних одиниць, які називаються **лексемами**, та визначенні відповідних цим лексемам синтаксичних одиниць мови - **токенів**. Одночасно з тексту вихідної програми видаляються коментарі і прогалини, які не мають змістовного навантаження, а також виявляються деякі види помилок. Перетворена лексичним аналізатором програма є ланцюжком вже не в вихідному алфавіті, а в алфавіті токенів.

Наприклад, нехай на вхід компілятора подано ланцюжок “if a13> 2300 then ab12: = 1400”

На стадії синтаксичного аналізу неістотно, що зліва від знаку нерівності стоїть змінна, ім'я якої є послідовність з букви і двох цифр, а істотно тільки те, що це ідентифікатор. Точно так само неважливо, що праворуч від знака нерівності стоїть число 2300, а не 2301, а важливо тільки те, що це числова константа. В цьому прикладі блок лексичного аналізу виділить вісім лексем: ключове слово if, ідентифікатор a13, знак нерівності >, ціле число 2300, ключове слово then, ідентифікатор ab12, оператор присвоєння := і ціле число 1400 (передбачається, що пробіли не мають змістовного навантаження). Кожна лексема належить певному класу лексем. Лексеми одного класу не розріняються з точки зору синтаксичного аналізатору (як, наприклад, ідентифікатори a13 і ab12).

Клас лексем називається **токеном**. Таким чином, наведений вище ланцюжок з 27 символів буде перетворений в ланцюжок з 8 токенів. Відзначимо, що кожне ключове слово утворює свій клас лексем, що складається з одного цього слова.

Токени описуються шаблонами, в якості яких найчастіше використовуються регулярні вирази або аналогічні конструкції. В табл. 4.2.1. наведені приклади токенів, лексем і шаблонів. В шаблонах для зручності зазвичай використовують розширений синтаксис регулярних виразів.

Розширені регулярні вирази є одним з основних елементів синтаксису мови Perl. Проте назва “регулярні вирази” там вельми умовна, оскільки далеко не всі дозволені до використання операції можуть бути виражені через об'єднання, множення і ітерацію. Як наслідок, клас мов, що задаються такими виразами, значно ширший класу регулярних мов.

Таблиця 4.2.1.

Токени, лексеми, шаблони

Токен	Приклади лексем	Шаблон
If	If	If
Id	a13, ab12, pi	<ім'я>
re1	<, <=, >	< | <= | = | <> | > | >=
Aop	+, -	+ | -
Mop	*, /	* | /
Num	2300, 3.14, 3.1E12	<константа>
Then	Then	Then
Assign	: =	: =
Comma	,	,

Оскільки алфавіт лінійно впорядкований, символи, що перераховуються через знак альтернативи | , часто утворюють інтервали. Для запису інтервалу використовують дефіс: *a-z* | 0-9 замість *a* | *b* ... | *z* | 0 | ... | 9. Крім того, для регулярного виразу *r* замість rr^* зазвичай пишуть r^+, а замість $r \mid \varepsilon$ відповідно $(r)?$. З урахуванням цього, шаблони токенів id і num можуть бути записані у вигляді

<ім'я> = a-z (a-z | 0-9) *

<константа> = (-)? (1-9 (0-9) * | 0) (. (0-9)$^+$)? (E (+ | -)? (0-9)$^+$)?

Як вже зазначалося, ідентифікатори a13 і ab12 в блоці синтаксичного аналізу можна не розрізняти. Однак в наступних блоках компілятора їх розрізняти доведеться. Отже, якщо токен - це клас, що складається з більш ніж однієї лексеми, то лексичний аналізатор повинен забезпечити можливість отримання відповідної інформації. Зазвичай це досягається шляхом приписування токену так званих атрибутів.

Атрибут - це змінна, що приймає значення в деякій множині. На практиці токени зазвичай мають один атрибут - вказівник на запис в таблиці символів, в якій зберігається інформація про токен. Результат обробки лексичним аналізатором ланцюжка, наведеного раніше, можна записати у

вигляді послідовності

If	id	Rel	num	then	Id	assign	num
	↓	↓	↓		↓		↓
	ptr1	ptr2	ptr3		ptr4		ptr5

Процес занесення даних про токени в таблицю символів з подальшим виведенням пари (токен, атрибут) на вихід аналізатора називають **реєстрацією токена**.

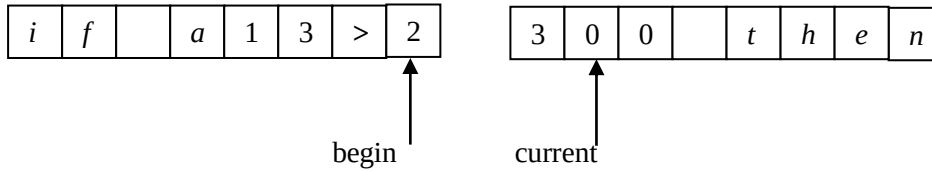
Вміст запису в таблиці символів залежить від токена, і ідентифікатор, очевидно, потребує найбільшої кількості відомостей. В запис для ідентифікатора лексичний аналізатор вносить ім'я лексеми і (для діагностичних цілей) місце в програмі, де вперше з'явився ідентифікатор. На наступних стадіях компіляції будуть додані відомості про тип, адресу для зберігання.

4.3. МЕТОДИ РОБОТИ ЛЕКСИЧНОГО АНАЛІЗАТОРУ

Розглянемо деякі з проблем, що виникають при організації блоку лексичного аналізу. Хоча це, мабуть, найпростіший блок компілятора, йому слід приділити належну увагу хоча б тому, що це єдиний блок, який читає всю вхідну програму, символ за символом. Отже, швидкість роботи лексичного аналізатору істотно позначається на ефективності компілятора в цілому. Зокрема, природньо вимагати, щоб лексичний аналізатор виконував свою роботу (розбиття тексту програми на лексеми та реєстрацію відповідних токенів) за один прохід по вхідному ланцюжку.

Спочатку коротко обговоримо організацію вводу та обробки даних. Якщо програма досить довга, то її посимвольне читання з пам'яті з подальшою обробкою - ідея навряд чи вдала. Набагато ефективніше зчитувати в буфер цілий блок (за допомогою однієї системної команди читання), а потім обробляти. При цьому зручно користуватися не одним буфером, а двома. Справді, при єдиному буфері виникає складність з обробкою ситуації, коли блок закінчився посередині лексеми. Маючи другий буфер, можна зчитати в нього наступний блок і продовжити роботу. Коли буде закінчуватися блок в другому буфері, блок в першому буде вже не потрібен (розмір блоку можна зробити свідомо більше максимального розміру лексеми) і на його місці можна

розмістити наступний блок, і т. д. В процесі роботи з буферами підтримуються два вказівника. Вказівник begin показує на перший символ поточної лексеми, а вказівник current - на оброблюваний символ:



Коли виявлено відповідність шаблону деякого токена, його лексема знаходиться між вказівниками. Тим самим доступна вся інформація про знайдений токен для внесення в таблицю символів. Після реєстрації токена обидва вказівники переміщуються на наступну позицію після знайденої лексеми.

Лексичний аналізатор виконує свою роботу (розбиття тексту програми на лексеми та реєстрацію відповідних токенів) за один прохід по вхідному ланцюжку.

Неоднозначності, що виникають при розпізнаванні лексем, вирішуються за *допомогою принципу найдовшої лексеми:* серед усіх лексем, що починаються в даній позиції, вибирається найдовша. Так, у наведеному прикладі між вказівниками знаходиться лексема 230, але даний принцип вимагає продовжити читання і вибрати лексему 2300 (що, очевидно, правильно).

Тепер перейдемо до опису процесу розпізнавання лексем.

Основна ідея - використати скінченний автомат, що розпізнає шаблони всіх токенів, і подавати йому на вхід фрагмент вхідного ланцюжка, що починається в позиції вказівника begin. При цьому кожному заключному стану автомата повинен однозначно відповідати токен, і з цим станом повинна бути пов'язана підпрограма реєстрації токена.

Для вирішення неоднозначності при виборі токена в багатьох мовах програмування ключові слова зарезервовані (тобто if - це завжди початок умовного оператора і ніколи - не ідентифікатор).

Зручно поступати так: побудувати автомати для кожного шаблону окремо, потім з'єднати їх в один є-НСА (не детермінований скінченний

автомат), додавши нову початкову вершину, після чого побудувати еквівалентний мінімальний ДСА(провести процедуру детермінізації та мінімізації скінченного автомату).

Діаграми автоматів для деяких шаблонів з табл. 1. наведені на рис. 4.3.1.-4.3.3. Зверніть увагу на рис. 4.3.3., на якому поєднані діаграми для зарезервованих ключових слів з діаграмою для ідентифікаторів.

Поруч з кожним заключним станом вказана процедура реєстрації токена в одному з двох варіантів, REG або REG-.

Процедура REG- застосовується, якщо останній прочитаний символ не належить знайденій лексемі (але його необхідно було прочитати, щоб виявити цю лексему!). В цьому випадку для ініціалізації пошуку наступної лексеми потрібно присвоїти вказівнику begin значення вказівника current.

Варіант REG використовується, якщо останній символ належить лексемі. В цьому випадку треба обидва вказівника розмістити на символ правіше поточного положення вказівника current. Якщо перехід зі стану з даним символом на діаграмі відсутній, то він здійснюється по дузі з міткою other (іноді його називають «переходом за замовчуванням»).Абривіатура *rel* позначає токен операцій відношення, *le-* скорочення *less-equal* тощо.

Оскільки потрібно один раз побудувати аналізатор, а потім багаторазово обробляти ним програми, то вигідно один раз застосувати обчислювально складні алгоритми детермінування і мінімізації автомату, отримуючи взамін непоганий виграш у швидкості обробки ланцюжків

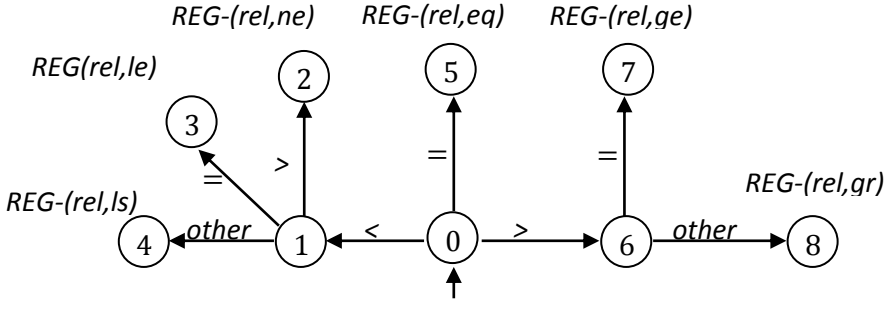


Рис. 4.3.1. Автомат для шаблону rel.

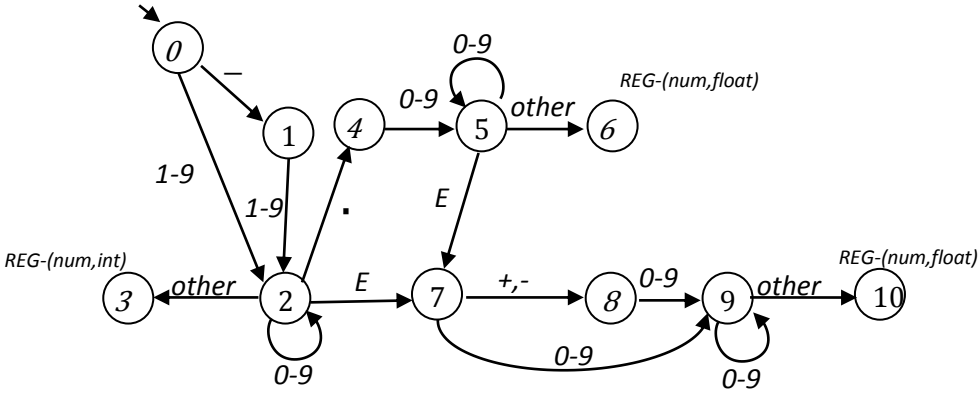


Рис. 4.3.2. Автомат для шаблону num.

Зауваження

В реальному аналізаторі крім переходів, зазначених на діаграмах, будуть виділені ще переходи, що відповідають лексичним помилкам у програмі. Наприклад, при аналізі фрагмента 1.1E3E4 вхідного ланцюжка потрібно видати повідомлення про лексичну помилку. Однак при використанні переходів за замовчуванням на рис. 4.3.2 буде зареєстрований токен num, а решта символів E4 будуть прийняті за ідентифікатор.

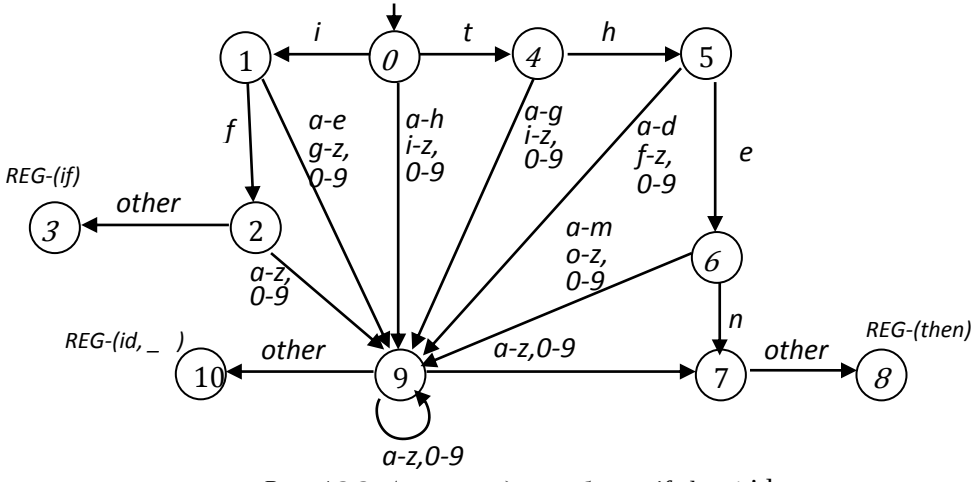


Рис. 4.3.3. Автомат для шаблонів if, then і id.

В цьому параграфі не розглянуті деякі проблеми, що виникають при проектуванні лексичного аналізатора, зокрема обробка лексичних помилок і

ефективна організація таблиці переходів ДКА.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення лексеми.
2. Що таке токен?Шаблон токена?
3. Що таке реєстрація токена?
4. Як прискорити процес зчитування вхідної програми?
5. Де саме в шаблоні токена потрібно задати обробку помилок?

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Для даного фрагменту паскаль-програми побудувати ланцюг токенів, їх атрибути і задати шаблон для заданого токена Т. Варіанти завдань взяти з таб. 4.3.1.

Таблиця 4.3.1

1	function frac_sum(n: integer): real; *{обчислення суми 1 +1/2 + ...+ 1/n}* begin frac_sum:=0; while n>0 do frac_sum:=frac_sum+1/n; n:=n-1; end; токен Т=оператор присвоєння	**2**	function frac_dob(n: integer): real; *{обчислення добутку 1*1/2 *..* 1/n}* begin frac_dob:=1; while n>0 do frac_dob:=frac_dob*1/n; n:=n-1; end; токен Т= оператор циклу
3	function frac_sum(n: integer): real; *{обчислення суми 1 + 1/2 +... + 1/n}* Var k: integer; begin frac_sum:=0; for k:=1 to n do frac_sum:=frac_sum+1/k; end; Т= оператор циклу	**4**	function frac_dob(n: integer): real; *{обчислення 1 * 1/2* 1/3 *... * 1/n}* Var k:integer; begin frac_dob:=1; for k:=1 to n do frac_dob:=frac_dob*1/k; end; Т= оператор опису функції

5	procedure dob (var n: integer; var frac_dob: real); Var k:integer; begin frac_dob:=1; repeat frac_dob:=frac_dob*LN(1/k); k:=k+1; until k>n end; T= оператор опису процедури	**6**	function frac_dob (n: integer): real; *{обчислення 1 * 1/2 * 1/3 *... * 1/n}* Var k:integer; begin frac_dob:=1; repeat frac_dob:=frac_dob*1/k; k:=k+1; until k>n end; токен T= оператор циклу
7	function frac_sum(n: integer): real; Var k:integer; begin sum:=0; while n>0 do frac_sum:=frac_sum+sin(1/n); n:=n-1; end; T= оператор опису функції	**8**	function frac_dob(n: integer): real; Var k:integer; begin frac_dob:=1; k:=1; while (n>0) and (k<10) do begin frac_dob:=frac_dob*cos(1/n); n:=n-1; k:=k+1 end; end; T= логічна умова циклу
9	function frac_sum(n: integer): real; Var k,z: integer; begin frac_sum:=0;z:=10; for k:=1 to n*z do frac_sum:=frac_sum+cos(1/k); end; T= оператор опису змінних	**10**	function frac_dob(n: integer): real; Var k:integer; a:real; begin frac_dob:=1;a:=3,14; for k:=1 to n do frac_dob:=frac_dob*sin(a/k); end; T= оператор опису змінних

11	function frac_sum(n: integer): real; Var k,z: integer; begin frac_sum:=0;z:=10; for k:=1 to n do frac_sum:=frac_sum+funcA(4, z); end; T= оператор виклику функції funcA	12	function frac_dob(n: integer): real; Var k:integer; begin frac_dob:=1; k:=1; while (n>0) and (k<10) do begin frac_dob:=frac_dob*cos(1/n); n:=n-1; call AAA(k); end; end; T= оператор виклику процедури AAA
13	procedure dob(var n: integer; var frac_dob: real); Var k:integer; begin frac_dob:=1; repeat frac_dob:=frac_dob*LN(1/k); k:=k+1; until k>n end; T= побудова арифметичних виразів в операторі присвоєнняи	14	function frac_dob(n: integer): real; Var k:integer; log:boolean; begin frac_dob:=1;k:=1; log:= (n>0) and (k<10); while log do begin frac_dob:=frac_dob*cos(1/n); n:=n-1; call AAA(k); end; end; T= оператор побудови логічних виразів в операторі присвоєння

Розділ 5. ПЕРЕТВОРЕННЯ КОНТЕКСНО-ВІЛЬНИХ ГРАМАТИК.

5.1. НЕОДНОЗНАЧНІСТЬ КОНТЕКСНО-ВІЛЬНИХ ГРАМАТИК.

Нехай G = (N, T, P, S) - КС-граматика. Введемо декілька важливих понять та визначень.

Визначення 5.1.1.

Вивід, в якому на кожному кроці здійснюється підстановка крайнього лівого нетерміналу, називається ***лівосторонім***. Якщо $S=>^{*}$ u в процесі лівостороннього виводу, то u - ***ліва сентенціальна форма.*** Аналогічно визначимо правостороній вивід. Позначимо кроки лівого (правого) виводу $=>_l$ ($=>_r$).

Визначення 5.1.2.

Впорядковане помічене дерево D називається ***деревом виводу*** (або *деревом розбору*) слова w в КВ-граматиці G = (N, T, P, S), якщо виконані наступні умови:

(1) корінь дерева D помічено аксіомою S ;

(2) кожен лист помічено або певним терміналом $a \in T$, або ε ;

(3) кожна внутрішня вершина помічена нетерміналом $A \in N$;

(4) якщо X - нетермінал, яким помічена внутрішня вершина і $X_m \dots X_k$ - мітки її прямих потомків у вказаному порядку, то $X \to X_m \dots X_k$ - правило з множини P ;

(5) Ланцюг, складений з виписаних зліва направо міток листків, рівний w.

Визначення 5.1.3.

Граматика G називається **неоднозначною,** якщо існує ланцюг w, для якого є два або більше різних дерев виводу в G.

Оскільки по дереву виводу ланцюжка w з нетерміналу S однозначно будується як лівий, так і правий вивід даного ланцюжка, то визначення 5.1.3 можна сформулювати інакше:

КВ-граматика однозначна, якщо кожен ланцюжок мови, що породжується нею, має єдиний лівий і єдиний правий виводи.

Розглянемо один хрестоматійний приклад неоднозначної граматики.

Приклад 5.1.1.

Задамо граматику G_{if} наступним чином

$$G_{if} = (\quad T = \{a, b, if, then, else\}; \quad N = \{S\}; \quad S;$$
$$P = \{\ S \ \rightarrow if\ b\ then\ S\ else\ S \ \mid \ if\ b\ then\ S \mid a\ \}\quad)$$

Ця граматика описує "абстрактний синтаксис" операторів умовного переходу в деякій "паскалеподібній" мові програмування. Термін "абстрактний синтаксис" в даному випадку означає, що ми відволікаємося від структури як операторів мови, так і логічних умов. Термінальний символ a означає довільний оператор (або послідовність операторів), а термінальний символ b - довільну логічну умову. Крім того, терміналами граматики є також лексеми $if, then, else$, які формують "скелет" будь-якого оператора умовного переходу.

Визначена таким чином граматика є неоднозначною, що видно з наступного прикладу:

Ланцюжок $if\ b\ then\ a\ if\ b\ then\ a\ else\ a$ має два дерева виводу (рис. 5.1.1а та 5.1.1б)

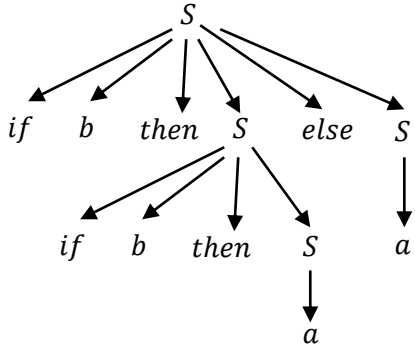


Рис.5.1.1а

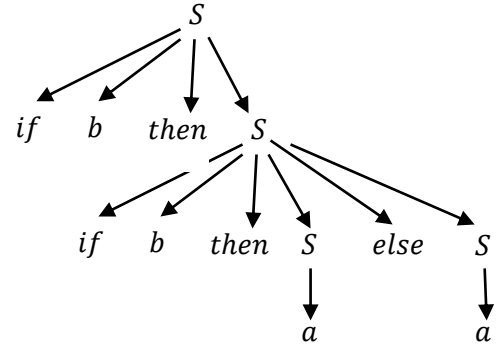


Рис. 5.1.2.б

Зауважимо (неформально), що наявність декількох дерев виводу одного і того ж ланцюжка має відношення до семантики розглянутої мови. В даному

конкретному прикладі зрозуміло, що кожному дереву виводу наведеного вище ланцюжка відповідає своє змістовне прочитання цього ланцюжка: в першому випадку $else$ ми відносимо до першого $then$, а в другому - до другого. В результаті отримуємо дві абсолютно різні інтерпретації оператора умовного переходу.

Даний приклад неоднозначності відомий під назвою "кочівне $else$ ".

Вихідну граматику, однак, можна "виправити", побудувавши еквівалентну їй однозначну граматику:

$$G_{if}{}' = (\quad T = \{a, b, if, then, else\};\quad N' = \{S_1,\ S_2\};\quad S_1;$$
$$P = \{\ S_1 \rightarrow if\ \ b\ \ then\ \ S_2\ else\ \ S_1 \mid if\ \ b\ \ then\ \ S_1 \mid a\ ,$$
$$S_2 \rightarrow if\ \ b\ \ then\ \ S_2\ else\ \ S_2 \mid a \qquad\qquad \}\ )$$

У загальному випадку алгоритму перетворення довільної неоднозначної КВ-граматики до еквівалентної однозначної не існує.

5.2. ВИДАЛЕННЯ НЕКОРИСНИХ СИМВОЛІВ КВ ГРАМАТИКИ.

Назвемо символ $X \in (N \cup T)$ недосяжним в граматиці $G = (N, T, P, S)$, якщо X не з'являється ні в одному виводі слова мови, що породжується цією граматикою. Іншими словами, символ X являється недосяжним, якщо в G не існує виводу $S \Rightarrow^* \alpha X \beta$ ні для яких $\alpha, \beta \in (N \cup T)^*$.

Назвемо символ $X \in (N \cup T)$ *непродуктивним (безплідним)* в тій же граматиці, якщо в ній не існує виводу виду $X \Rightarrow^* xwy$, де $w, x, y \in T^*$.

При уважному вивченні вищенаведених визначень стає зрозумілим, що

а) доцільно шукати не безпосередньо самі недосяжні символи, а послідовно визначати множину досяжних символів, починаючи з аксіоми; аналогічно множину продуктивних символів починаємо будувати, вибравши правила, які не містять нетерміналів в правих частинах. Перетин множин досяжних і продуктивних нетерміналів задає множину корисних нетерміналів граматики (автомату) - всі інші символи виявляються некорисними,

б) одночасне визначення досяжних і продуктивних символів неможливо, так як відповідні процеси йдуть у протилежних напрямках (від кореня до листків і навпаки).

Алгоритм 5.2.1. Визначення досяжних нетерміналів граматики.

Вхід. Дано граматику $G = (N, T, P, S)$.

Вихід. Граматика $G' = (N', T', P', S)$ без недосяжних нетерміналів, така, що L(G') = L(G).

Метод. Виконати кроки 1-4:

(1) Покласти $V_0 = \{S\}$ та i = 1,

(2) Покласти $V_i = \{X \mid$ якщо в P є правило $A \to \alpha X \beta$ і $A \in V_{i-1}\} \cup V_{i-1}$,

(3) Якщо $V_i \neq V_{i-1}$, покласти i = i + 1 і перейти до кроку 2, інакше перейти до кроку 4,

(4) Покласти $N' = V_i \cup N, T' = V_i \cap T$. Включити в P' всі правила з P, які містять тільки символи з V_i.

Алгоритм 5.2.2. Визначення продуктивних символів граматики

Вхід. Граматика $G = (N, T, P, S)$.

Вихід. Граматика $G' = (N', T', P', S)$ без непродуктивних символів, така, що L(G') = L(G).

Метод. Виконати кроки 1-4:

(1) Покласти $N_0 = T$ та i = 1,

(2) Покласти $N_i = \{A \mid A \to \alpha, \alpha \in (N_{i-1})^*\} \cup N_i$,

(3) Якщо $N_i \neq N_{i-1}$, покласти i = i + 1, перейти до кроку 2, інакше покласти $N_e = N_i$, перейти до кроку 4,

(4) Покласти $G' = ((N \cap N_e) \cup \{S\}, T, P', S)$, де P' складається з правил множини P, що містять тільки символи з $N \cap N_e$,

Щоб видалити всі некорисні символи, необхідно застосувати до граматики спочатку Алгоритм 5.2.2, а потім Алгоритм 5.2.1.

5.3.ВИДАЛЕННЯ ЕПСИЛОН-ПРАВИЛ В КВ ГРАМАТИКАХ.

Визначення 5.3.1.

Нетермінал A називається ε ***-породжуючим***, якщо для нього задано правило $A \to \varepsilon$ або правило $A \to C_1 C_2 \ldots C_k$, в якому права частина правила містить тільки нетермінали і кожен нетермінал C_i являється ε -породжуючим.

Теорема 5.3.1. Нехай *мова* L є *контекстно-вільною. Тоді мова* $L - \{\varepsilon\}$ *породжується деякою контекстно-вільною граматикою без* ε *-правил.*

Доведення.

Нехай дано контекстно-вільну граматику $G = (N, T, P, S)$, що породжує мову L. Проведемо серію перетворень множини правил P.

(1) Якщо для якихось $A \in N$, $B \in N$, $\alpha, \beta \in (N \cup T)^*$, множина P містить правила $B \to \alpha A \beta$ і $A \to \varepsilon$, але не містить правила $B \to \alpha\beta$, то додамо це правило в P. Повторюємо цю процедуру, для всіх ε -породжуючих нетерміналів.

(2) Тепер виключимо з множини P всі правила з ε -породжуючими нетерміналами в лівій частині правил граматики. Отримана граматика породжує мову $L - \{\varepsilon\}$.

Приклад 5.3.1.

Розглянемо мову L, що породжується граматикою з наступними правилами

$$S \to \varepsilon,$$
$$S \to aSbS$$

Виконуючи процедуру (1) отримаємо мову $L - \{\varepsilon\}$, що породжується граматикою

$$S \to aSbS\,,$$
$$S \to abS\,,$$
$$S \to aSb\,,$$
$$S \to ab\,.$$

Приклад 5.3.2.

Розглянемо мову L, що породжується граматикою з наступними правилами

$$S \to AB$$
$$A \to aAA \mid \varepsilon$$
$$B \to bBB \mid \varepsilon$$

Спочатку знайдемо ε-породжуючі символи. Нетермінали A і B є безпосередньо ε-породжуючими, так як для них задані ε - правила:

$$A \to \varepsilon \;\; i \;\; B \to \varepsilon$$

Тоді й S являється ε-породжуючим нетерміналом, оскільки тіло продукції S → AB складається тільки з ε-породжуючих символів. Таким чином, всі три нетермінали граматики є ε-породжуючими.

Побудуємо тепер продукції граматики G_1 без ε – правил, перетворюючи праві частини правил, які містять ε породжуючі нетермінали..

Спочатку розглянемо S → AB. Всі нетермінальні символи тіла є ε-породжуючими, тому є 4 способи вибрати присутність або відсутність A і B. Однак нам не можна видаляти всі символи одночасно, тому по черзі отримуємо наступні три продукції.

$$S \to AB \mid A \mid B$$

Далі розглянемо продукцію A → aAA. Другу і третю позиції займають ε-породжуючі символи, тому знову є 4 варіанти їх присутності або відсутності. Всі вони допустимі, оскільки в будь-якому з них залишається термінал a. Два з них збігаються, тому в граматиці G1 будуть наступні три продукції.

$$A \rightarrow aAA \mid aA \mid a$$

Аналогічно, продукція для B призводить до наступних правил в G_1.

$$B \rightarrow bBB \mid bB \mid b$$

Таким чином, в граматику G_1 *без ε – правил входять наступні продукції*

$$S \rightarrow AB \mid A \mid B$$
$$A \rightarrow aAA \mid aA \mid a$$
$$B \rightarrow bBB \mid bB \mid b.$$

Граматика G1 породжує ту ж саму мову що і граматика G, за виключенням пустого слова. Пусте слово породжується правилом $S \rightarrow \varepsilon$, *додавши таке правило до* G_1 *отримаємо граматику еквівалентну до G.*

5.4.ВИДАЛЕННЯ ЛАНЦЮГОВИХ ПРАВИЛ В КВ-ГРАМАТИКАХ.

Визначення 5.4.1.

Ланцюгова продукція *- це продукція виду* $A \rightarrow B$, *де і A, і B є нетерміналами (змінними).*

Ці продукції можуть бути корисними при намаганні зробити граматики однозначними. Разом з тим, ланцюгові продукції можуть ускладнювати деякі доведення і створювати зайві кроки у виводах, що з технічних міркувань не потрібно.

Розглянемо однозначну граматику $G_{\text{ариф}}$ *для простих арифметичних виразів (аксіома E).*

$$E \rightarrow T \mid E + T$$
$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$
$$F \rightarrow I \mid (E)$$
$$T \rightarrow F \mid T * F$$

Позбутись ланцюгового правила $E \rightarrow T$ *можна, підставивши в це правило всі праві частини правил для T, отримаємо замість нього два правила*

$$E \rightarrow F \mid T * F.$$

Ця заміна все ще не позбавляє від ланцюгових продукцій через появу правила

$$E \rightarrow F.$$

Подальша заміна не терміналу F на праві частини його правил дає

$$E \to I \mid (E) \mid T * F,$$

проте при цьому залишається ланцюгове правило

$$E \to I.$$

Але якщо в цій продукції замінити I усіма шістьма можливими способами, то отримаємо такі продукції

$$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F$$

Представлена вище техніка розширення ланцюгових продукцій до їх зникнення працює досить часто.

Однак вона зазнає невдачі, якщо в граматиці є *цикл із ланцюгових продукцій*, наприклад

$$A \to B,\ B \to C \text{ і } C \to A.$$

Техніка, яка гарантує результат у цьому випадку, включає попереднє знаходження всіх пар змінних A і B, для яких $A \Rightarrow^* B$ виводиться з використанням послідовності лише ланцюгових продукцій. (Зауважимо, що $A \Rightarrow^* B$ можливе і без використання ланцюгових продукцій, наприклад, за допомогою продукцій

$$A \to BC \text{ і } C \to \varepsilon,$$

такі правила ми не беремо до уваги при видаленні ланцюгових правил).

Визначивши всі подібні пари, будь-яку послідовність кроків виводу, в якому

$$A \to B_1 \to B_2 \to \ldots \to B_n \to \alpha$$

з неланцюговою продукцією $B_n \to \alpha$, можна замінити однією продукцією

$$A \to \alpha.$$

Однак спочатку розглянемо індуктивну побудову пар (A, B), для яких

$$A \Rightarrow * B$$

виводиться з використанням лише ланцюгових продукцій. Назвемо таку пару *ланцюгової парою* (unit pair).

Визначення 5.4.2.

1. (A_i , A_i) є *ланцюговою парою* для будь-якого нетерміналу A_i , тобто $A_i \Rightarrow^* A_i$ за нуль кроків.
2. Припустимо, що пара (A, B) визначена як ланцюгова, і граматика містить правило B → C . Тоді (A, C) - *ланцюгова пара.*

Приклад 5.4.2.

Розглянемо граматику виразів $G_{ариф}$, описану вище. За визначенням (пункт 1) наступні пари нетерміналів є ланцюговими парами

(E, E), (T, T), (F, F) і (I, I).

Згідно пункту 2 визначення утворюються наступні ланцюгові пари.

1. (E, E) і продукція E → T дають пару (E, T).
2. (E, T) і продукція T → F – пару (E, F).
3. (E, F) і F → I дають пару (E, I).
4. (T, T) і T → F – пару (E, I).
5. (T, F) і F → I – пару (T, I).
6. (F, F) і F → I - пару (F, I).

Більше пар, які можна було б вивести, немає. Насправді ці десять пар представляють всі виводи, що використовують тільки ланцюгові продукції.

Спосіб побудови пар тепер очевидний. Неважко довести, що запропонований алгоритм забезпечує породження всіх потрібних пар. Знаючи ці пари, можна видалити ланцюгові продукції з граматики і показати еквівалентність вихідної і отриманої граматик.

Алгоритм видалення ланцюгових правил в КВ граматиках.

Для видалення ланцюгових продукцій по КВ-граматиці *G = (V, T, P, S)* побудуємо КВ-граматику *G1 = (V1, T, P1, S)* наступним чином.

1. Знайдемо всі ланцюгові пари граматики G.

2. Для кожної пари (A, B) додамо до P1 всі продукції $A \to \alpha$, де $B \to \alpha$ – неланцюгова продукція з P. Зауважимо, що при A = B(тобто пара (A,A) всі неланцюгові продукції для A додаються з P додаються до P1.

Приклад 5.4.3.

Продовжимо приклад 5.4.1, де був виконаний крок 1 описаної побудови для граматики виразів $G_{ариф}$. В табл..5.4.1 представлено крок 2 алгоритму, який будує нову множину продукцій. При цьому перший член пари стає головою продукцій, а в якості їхніх тіл використовуються всі тіла неланцюгових продукцій для другого члена.

Табл 5.4.1.

Пара	Продукція
(E,E)	$E \to E+T$
(E,T)	$E \to T*F$
(E,F)	$E \to (E)$
(E,I)	$E \to a \mid b \mid I_a \mid I_b \mid I0 \mid I1$
(T,T)	$T \to T*F$
(T,F)	$T \to (E)$
(T,I)	$T \to a \mid b \mid I_a \mid I_b \mid I0 \mid I1$
(F,F)	$F \to (E)$
(F,I)	$F \to a \mid b \mid I_a \mid I_b \mid I0 \mid I1$
(I,I)	$I \to a \mid b \mid I_a \mid I_b \mid I0 \mid I1$

На заключному кроці з граматики видаляються всі ланцюгові продукції. У підсумку виходить наступна граматика без ланцюгових продукцій, яка породжує ту ж саму множину виразів, що і грамматика $G_{ариф}$.

$E \to E + T \mid T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
$T \to T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
$F \to (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ.

1. Визначити чи є наступні граматики однозначними:

1.1. $S \to A \mid B;\ \ B \to a\,B \mid b \mid C;\ \ A \to AA \mid a;\ \ C \to cC;$

1.2. $S \to aAc \mid b\,S;\ \ A \to a\,A \mid Aa \mid c;$

1.3. $S \to aA \mid b;\ \ A \to a\,bA \mid abAcb;\ \ B \to c;$

1.4. $S \to aB \mid cA;\ \ A \to BaA \mid a;\ \ B \to A \mid b;$

1.5. $S \to a \mid C;\ \ C \to AB;\ \ A \to aA \mid Ba \mid a;\ \ B \to aB;$

1.6. $S \to BA;\ \ A \to Aa \mid bA \mid \varepsilon;\ \ B \to aB;$

1.7. $S \to b \mid C;\ \ C \to aC \mid AC;\ \ A \to aA \mid Aa \mid a;$

1.8. $S \to AB;\ \ A \to aA \mid bA \mid a;\ \ B \to Ba \mid Bb \mid \varepsilon;$

1.9. $S \to bA \mid ab;\ \ A \to a \mid aS \mid bAA;\ \ B \to bBB \mid b;$

1.10. $S \to aB \mid aBS \mid BAS \mid bA;\ \ A \to bAA \mid a;\ \ B \to bBB \mid b;$

2. Видалити некорисні символи в наступних граматиках G = < {S,A,B,C}, {a, b, c} , S, P >

2.1. P: $S \to A \mid B;\ \ B \to aB \mid b \mid C; C \to cC;$

2.2. P: $S \to aSb \mid Abb \mid \varepsilon;\ \ B \to AB;\ A \to aBCb \mid bAB;\ \ C \to a \mid c;$

2.3. P: $S \to A \mid B;\ \ A \to aB \mid bS \mid b; B \to AB \mid Ba;\ \ C \to AS \mid b;$

2.4. P: $S \to aBb \mid aCb; A \to Dc \mid cA; B \to aS \mid b;\ C \to AB \mid aD; D \to AB \mid cDa$

2.5. P: $S \to SS \mid A; A \to 0A1 \mid C \mid 0; B \to 0C \mid 1; C \to BC \mid CS$

2.6. $P: S \rightarrow a \mid C; C \rightarrow AB; A \rightarrow aA \mid Ba \mid a; B \rightarrow aB;$

2.7. $P: S \rightarrow bA \mid C; C \rightarrow aC \mid AC; A \rightarrow aA \mid Aa \mid a; B \rightarrow aB;$

2.8. $P: S \rightarrow A \mid B; A \rightarrow AA \mid a; B \rightarrow aB \mid b \mid C; C \rightarrow cC;$

2.9. $P: S \rightarrow aAc \mid bS \mid a; A \rightarrow aA \mid Ab; \ B \rightarrow c;$

2.10. $P: S \rightarrow aB \mid cA; A \rightarrow BaB \mid a; B \rightarrow aS \mid C; C \rightarrow aC;$

3. Побудувати КВ-граматики, еквівалентні наступним граматикам. Позбутись ланцюгових правил

3.1. $S \rightarrow AB$
$A \rightarrow aAA \mid \varepsilon$
$B \rightarrow bBB \mid \varepsilon$

3.2. S → aAa | bBb | ε
A → C | a
B → C | b
C → CDE | ε
D → A | B | ab

3.3. $S \rightarrow ABa$
$A \rightarrow C \mid ab$
$C \rightarrow c \mid \varepsilon$
$B \rightarrow aAa$

3.4. $S \rightarrow ABC$
$A \rightarrow BB \mid \varepsilon$
$B \rightarrow CC \mid \varepsilon$
$C \rightarrow AA \mid b$

3.5. $S \rightarrow aSbS$
$S \rightarrow bSaS \mid \varepsilon$

3.6. $S \rightarrow AB$
$A \rightarrow SA \mid BB \mid bB$
$B \rightarrow b \mid aA \mid \varepsilon$

3.7. $S \rightarrow aSbT \mid bTaT \mid ab$
$T \rightarrow baaST \mid TT \mid \varepsilon$

3.8. $S \rightarrow ABC$

$A \rightarrow BB \mid \varepsilon$

$B \rightarrow CC \mid a$

$C \rightarrow AA \mid b$

3.9. $S \rightarrow T$

$T \rightarrow aTa \mid bTa \mid a \mid b \mid \varepsilon$

3.10. $S \rightarrow A1B$

$A \rightarrow 0A \mid \varepsilon$

$B \rightarrow 0B \mid 1B \mid \varepsilon$

Розділ 6. НОРМАЛЬНА ФОРМА ХОМСЬКОГО ДЛЯ КВ ГРАМАТИК. АЛГОРИТМ СИНТАКСИЧНОГО АНАЛІЗУ КОКА-КАСАМІ-ЯНГЕРА.

6.1. ПРИВЕДЕННЯ КВ ГРАМАТИКИ ДО НОРМАЛЬНОЇ ФОРМИ ХОМСЬКОГО.

Визначення 6.1.1.

Граматика в нормальній формі Хомського (*граматика в бінарній нормальній формі, квадратична граматика*, grammar in Chomsky normal form) - контекстно-вільна граматика $\langle N, \Sigma, S, P \rangle$, в якій кожне правило є одним з наступних трьох видів:

$$S \to \varepsilon, \quad S - \text{аксіома}$$
$$A \to a, \quad A \in N, a \in \Sigma$$
$$A \to BC, \quad A \in N, \quad B \in N - \{S\}, C \in N - \{S\}$$

Приклад 6.1.1.

Граматика

$$S \to RR; \quad S \to AB; \quad R \to RR; \; R \to AB;$$
$$A \to a; \; B \to RB; \; B \to b$$

є граматикою в нормальній формі Хомського.

Теорема 6.1.1.

Кожна контекстно-вільна граматика еквівалентна деякій граматиці в нормальній формі Хомського.

Доведення.

Нехай дано контекстно-вільну граматику $G = \langle N, \Sigma, S, P \rangle$. Проведемо ряд перетворень цієї граматики так, щоб породжувана нею мова залишиться незмінною.

(1) Якщо права частина деякого правила містить символ S, то замінимо граматику N, Σ, S, P на граматику

$$\langle N \cup \{S_0\}, \Sigma, S_0, P \cup \{S_0 \to S\}\rangle$$

де S_0 – нова аксіома граматики і відповідно новий нетермінал, що не належить множині $N \cup \Sigma$. Таким чином ми позбуваємось випадку, коли аксіома граматики зустрічається в правих частинах правил.

(2) Замінимо у всіх правилах кожен термінальний символ a на новий нетермінальний символ T_a і додамо до множини P правила $T_a \to a$ для всіх $a \in \Sigma$.

(3) Видалимо правила виду $A \to \alpha$, де $\alpha > 2$, замінивши кожне з них на ряд коротких правил по 2 нетермінали кожному (при цьому додаються нові нетермінальні символи).

(4) Тепер видалимо всі правила виду $A \to \varepsilon$, де A не являється початковим символом. Це можна зробити так само, як при видаленні ε правил в параграфі 5.3.

(5) Якщо для будь-яких $A, B \in N$ і $\alpha \in (N \cup \Sigma)^*$ множина P містить правила $A \to B$ i $B \to \alpha$, але не містить правила $A \to \alpha$, то додаємо це правило в P. Правила $A \to B$ називаються **ланцюговими**. Повторюємо цю процедуру, доки можливо. ПІсля цього виключимо з множини P всі правила виду $A \to B$.

Зауваження. Перетворення КВ граматик необхідно здійснювати САМЕ В ТАКОМУ ПОРЯДКУ

а) видалити ε-продукції;

б) видалити ланцюгові продукції;

в) видалити некорисні символи;

г) ввести нову аксіому, якщо аксіома попередньої граматики вживалась в правих частинах правил

д) замінити правила виду $A \to \alpha$, де $\alpha > 2$

Приклад 6.1.1..

Граматика

$$S \rightarrow \varepsilon$$
$$S \rightarrow aUbU$$
$$U \rightarrow S$$
$$U \rightarrow ba$$

еквівалентна наступній граматиці в нормальній формі Хомського:

$$S_0 \rightarrow \varepsilon$$
$$S_0 \rightarrow AD$$
$$D \rightarrow UC$$
$$D \rightarrow BU$$
$$D \rightarrow b$$
$$C \rightarrow BU$$
$$C \rightarrow b$$
$$U \rightarrow BA$$
$$U \rightarrow AD$$
$$A \rightarrow a$$
$$B \rightarrow b$$

Приклад 6.1.2.
Приведемо граматику $G_{ариф}$ до НФХ.

Зауважимо, що граматика має вісім терміналів, a, b, 0, 1, +, *, (i). Сім з них є поодинокими терміналами тобто вживаються в правилах виду I → a, таким чином, потрібно ввести -сім нових змінних, які відповідають цим терміналам, і вісім наступних продукцій, де змінна замінюється "своїм" терміналом.

$A \rightarrow a$ $B \rightarrow b$ $Z \rightarrow 0$ $O \rightarrow 1$
$P \rightarrow +$ $M \rightarrow *$ $L \rightarrow ($ $R \rightarrow)$

Ввівши ці продукції і замінивши кожен термінал в тілі, що не є поодиноким терміналом відповідної змінної, отримаємо граматику, зображену нижче

$E \rightarrow EPT \mid TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$T \rightarrow TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$F \rightarrow LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$I \rightarrow a \mid b \mid IA \mid IB \mid IZ \mid IO$
$A \rightarrow a$ $B \rightarrow b$ $Z \rightarrow 0$ $O \rightarrow 1$
$P \rightarrow +$ $M \rightarrow *$ $L \rightarrow ($ $R \rightarrow)$

Тепер всі продукції знаходяться в НФХ, за винятком тих, тіла яких мають довжину 3: EPT, TMF, LER. Деякі з цих тіл зустрічаються в декількох продукціях,але кожне тіло перетворюється один раз.

Для тіла *EPT* вводиться нова змінна C_1, і продукція $E \rightarrow EPT$ змінюється на $E \rightarrow EC_1$ і $C_1 \rightarrow PT$. Для тіла *TMF* вводиться змінна C_2. Дві продукції з цим тілом, $E \rightarrow TMF$ і $T \rightarrow TMF$, змінюються на $E \rightarrow TC_2$, $T \rightarrow TC_2$ і $C_2 \rightarrow MF$. Для *LER* вводиться C_3 і три продукції з цим тілом, $E \rightarrow LER$, $T \rightarrow LER$ *i* $F \rightarrow LER$, змінюються на $E \rightarrow LC_3$, $T \rightarrow LC_3$, $F \rightarrow LC_3$ *i* $C_3 \rightarrow ER$. Остаточна НФХ-граматика показана нижче.

$E \rightarrow EC_1 \mid TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$T \rightarrow TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$F \rightarrow LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO$
$I \rightarrow a \mid b \mid IA \mid IB \mid IZ \mid IO$
$A \rightarrow a$ $B \rightarrow b$ $Z \rightarrow 0$ $O \rightarrow 1$ $P \rightarrow +$ $M \rightarrow *$
$L \rightarrow ($ $R \rightarrow)$ $C_1 \rightarrow PT$ $C_2 \rightarrow MF$ $C_3 \rightarrow ER$

6.2. АЛГОРИТМ СИНТАКСИЧНОГО АНАЛІЗУ КОКА-КАСАМІ-ЯНГЕРА ДЛЯ ГРАМАТИК В НОРМАЛЬНІЙ ФОРМІ ХОМСЬКОГО

Існує достатньо ефективний метод визначення чи належить ланцюжок мові, що задана граматикою, який заснований на ідеї "динамічного програмування" та відомий також, як "алгоритм заповнення таблиці" або "Табуляція". Даний алгоритм відомий ще як CYK-алгоритм -- алгоритм Кока-Янгера-Касамі.

Він використовується *лише для граматик в нормальній формі Хомського.*

Нехай дано -граматику G = (V, T, P, S) для мови L в НФХ.

На вхід алгоритму подається ланцюжок w = a_1a_2 ... a_n з T *. . За час O (n3) алгоритм будує таблицю, яка говорить, чи належить w мові L.

X_{15}				
X_{14}	X_{25}			
X_{13}	X_{24}	X_{35}		
X_{12}	X_{23}	X_{34}	X_{45}	
X_{11}	X_{22}	X_{33}	X_{44}	X_{55}
a_1	a_2	a_3	a_4	a_5

Рис.6.2.1. Схема індексації таблиці аналізу

Базис. Обчислюємо перший рядок наступним чином. Оскільки ланцюжок, який починається та закінчується в позиції і, являє собою просто термінал аі, а граматика знаходиться в НФХ, єдиний спосіб породити a_i полягає в використанні продукції виду $A \rightarrow a_i$ граматики *G*. Отже, X_{ii} є множиною змінних *A*, для яких $A \rightarrow a_i$ - продукція *G*.

Індукція. Нехай потрібно обчислити X_{ij} в $(j - i + 1)$ -ому рядку, і всі множини *X* в нижніх рядках вже обчислені, тобто відомі для всіх подланцюжків, більш коротких, ніж $a_ia_{i+1} \ldots a_j$, і зокрема, для всіх власних префіксів і суфіксів цього ланцюжка.Можна припускати, що $j - i > 0$, оскільки

випадок j = і розглянуто в базисі. Тому будь-який вивід $A \Rightarrow^* a_i a_{i+1} \ldots a_j$ має починатися кроком $A \to BC$. Тоді В породжує деякий префікс рядка $a_i a_{i+1} \ldots a_j$, скажімо, $B \Rightarrow^* a_i a_{i+1} \ldots a_k$ для деякого k <j. Відповідно, С породжує залишок $a_{k+1} a_{k+2} \ldots a_j$, тобто $C \Rightarrow^* a_{k+1} a_{k+2} \ldots a_j$

Приходимо до висновку, що для того, щоб *A* потрапило в X_{ij}, потрібно знайти змінні *B і C* і ціле *k*, при яких справедливі наступні умови.

1. $i \le k < j$.
2. *B* належить X_{ik}.
3. *C* належить $X_{k+1, j}$.
4. $A \to BC$ - продукція в *G*.

Пошук таких змінних А потребує обробки не більше n пар обчислених раніше множин: (X_{ii}, X_{i+1}, j), $(X_i, _{i+1}, X_{i+2, j})$ і т.д. до $(X_{i, j-1}, X_{jj})$. Таким чином, ми піднімаємося по колонці, розташованій під X_{ij}, і одночасно спускаємося по діагоналі (дивитись рис 6.2.2.)

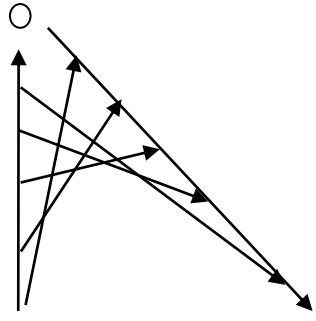


Рис 6.2.3.Схема вибору пар нетерміналів в таблиці аналізу

Приклад 6.2.1

Розглянемо наступні продукції граматики G в НФХ.

$S \to AB \mid BC$ $\qquad$ $A \to BA \mid a$

$B \to CC \mid b$ $\qquad$ $C \to AB \mid a$

Перевіримо, чи належить ланцюжок baaba мові *L (G).* Для побудови першого (нижнього) рядка використовується базисне правило. Потрібно розглядати лише не термінали з тілом(правою частиною) продукції a (*це A і C*) і тілом b (це B). Таким чином $X_{11} = X_{44} = \{B\}$, $X_{22} = X_{33} = X_{55} = \{A, C\}$.

У другому рядку показані значення X_{12}, X_{23}, X_{34} *і* X_{45}. Розглянемо, наприклад, як обчислюється X_{12}. Ланцюжок ba, що займає позиції 1 і 2, можна розбити на непорожні під ланцюжки єдиним способом. Перший повинен займати позицію 1, другий - позицію 2. Для того щоб змінна породжувала *ba*, вона повинна мати продукцію з тілом, перший нетермінал якої належить $X_{11} = \{B\}$ (тобто породжує *b*), а другий $X_{22} = \{A, C\}$ (тобто породжує a). Таким тілом може бути тільки *BA* або *BC*. Переглянувши граматику, знаходимо, що з такими тілами там є тільки продукції $A \rightarrow BA$ *і* $S \rightarrow BC$.

Таким чином, дві голови (ліві частини правил граматики) *A* і *S*, утворюють X_{12}.

$\{S, A, C\}$				
$-$	$\{S, A, C\}$			
$-$	$\{B\}$	$\{B\}$		
$\{C\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$	
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
b	a	a	b	a

Рис 6.2.3. Таблиця аналізу.

У якості більш складного прикладу розглянемо обчислення X_{24}. Ланцюжок aab в позиціях з 2 по 4 можна розбити, закінчуючи перший підланцюжок в 2 або в 3, тобто в визначенні X_{24} можна вибрати k = 2 або k = 3. Таким чином, потрібно розглянути всі тіла в $X_{22}X_{34}$ *U* $X_{23}X_{44}$. Цією множиною ланцюжків є *{A, C} U {S, C} U {B}. Тоді {B} = {AS, AC, CS,CC, BB}.* З п'яти ланцюжків цієї множини тільки *CC* є тілом; його голова - *B*.Таким чином, $X_{24} = \{B\}$

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Перетворіть наступні граматики до нормальної форми Хомського, виконавши наступні кроки – САМЕ В ТАКОМУ ПОРЯДКУ

а) видаліть ε-продукції;

б) видаліть ланцюгові продукції;

в) чи є некорисні символи?Якщо так – видаліть їх;

г) приведіть граматику до нормальної форми Хомського.

Зауваження: При приведенні до нормальної форми Хомського граматики, яка містить правила виду A→a не потрібно вводити додаткові нетермінали виду T_a→a, а скористатись в якості T_a уже існуючим нетерміналом A, щоб не розширювати зайво граматику

Застосуйте алгоритм Кока-Касасмі-Янгера для перетвореної граматики та вказаного ланцюжка, побудуйте таблицю розбору по даному алгоритмові для вказаного слова

Варіант 1. S → 0A0| 1B1 | BB A → C|0D D→11D B → S | A F→1 C → S | ε слово 001100.	***Варіант 2*** S→AAA |cSc|D D→aaaD A → a | B F→ab B → ε|b слово ccbbbcc.
Варіант 3 S → aAa | bBb | F A → C | a B → C | b F→bbbF C → CDE | ε D → A | B | ab слово aaabaa..	***Варіант 4.*** S → aAa | bBb | BB|D A → C D→bbD B → S | A C → S | ε Слово aabbaa. Чи належить воно мові, породженій такою граматикою.

Варіант 5. S → 00A00 | 1B1 | BB A → C|1D D→00D B → S | A F→0 C → S | ε слово 100001..	***Варіант 6*** S → AAA | aSa|D D→aaaD A → b | BS F→ab B → ε|c Слово аасссаа. Чи належить воно мові, породженій такою граматикою
Варіант 7. S→0A0|11B11|BB|D A → C D→11D B → S | A C → S | ε слово 001100. Чи належить воно мові, породженій такою граматикою.	***Варіант 8.*** S -> A|BAB B -> bB| ε |C A -> AA|a D→bbb C -> cC слово bbbaab. Чи належить воно мові, породженій такою граматикою.
Варіант 9 S → 0A0 | 1B1 | F A → C |0 B → C |1 F→111F C → CDE | ε D → A | B | 01 слово 000100.	***Варіант 10.*** S → A|B A → aB|bS| ε B → S|b C → AS|b D→bbb слово abbbab.
Варіант 11. S → aBb|aCb A → Dc|cA| ε B → aS|b C → AB|aD E→F|ccc F→ca D → AB|cDa. слово аccccb.	***Варіант 12.*** S → SS|A A → 0A1|C| ε B → 0C|1 C → A|CS. слово 001101.

Варіант 13. S→aSb|Abb|B B→A D→aCa A→aBb |bAb |ε C→a|c слово aaabbbbb.	***Варіант 14.*** S -> A|BAB B -> bB| ε |C A -> AA|a D→bbb C -> cC слово bbbaab.

Розділ 7. АВТОМАТИ З МАГАЗИННОЮ ПАМ‘ЯТТЮ.

7.1.ПОНЯТТЯ МП АВТОМАТУ

Визначення 7.1.1.

Автомат з магазинною пам‘яттю (МП-автомат) – це сімка об‘єктів $M = (Q, T, \Gamma, D, q_0, Z_0, F)$, де

Q – скінчена множина станів керуючого пристрою;

T – скінчений вхідний алфавіт;

Γ – скінчений алфавіт магазинних символів;

D – функція переходів, що є відображенням множини $Q \times (T \cup \{\varepsilon\}) \times \Gamma$ в $Q \times \Gamma^*$

$q_0 \in Q$ - початковий стан керуючого пристрою;

$Z_0 \in \Gamma$ - початковий символ магазину;

$F \subset Q$ - множина заключних станів.

Визначення 7.1.2.

Конфігурацією МП-автомату називається трійка $(q, w, \alpha) \in Q \times T^* \times \Gamma^*$, де $q \in Q$ - поточний стан керуючого пристрою; $w \in T^*$ - непрочитана частина вхідного рядка; перший символ рядка w знаходиться під головкою читання; якщо $w = \varepsilon$, вважають, що весь вхідний рядок прочитаний; $\alpha \in \Gamma^*$ - вміст магазину; перший зліва символ рядка α - це верхній символ магазину; у випадку $\alpha = \varepsilon$, вважають, що магазин пустий.

Такт (крок) роботи МП-автомату будемо представляти як бінарне відношення $\mapsto$ на множині конфігурацій, Запис $(q, aw, z\alpha) \mapsto (q', w, \gamma\alpha)$ означає перехід від одного такту до іншого, де $(q', \gamma) \in D(q, a, z)$, $q, q' \in Q$, $a \in T \cup \{\varepsilon\}$, $w \in T^*$, $z \in \Gamma$, $\alpha, \gamma \in \Gamma^*$,

Якщо $a \neq \varepsilon$ то у поданому такті МП-автомат, маючи керуючий пристрій у стані q, символ a під головкою читання та z у вершині стеку, змінює стан керуючого пристрою на q', зсовує вхідну головку на одну комірку праворуч і

замінює верхній символ стеку рядком γ магазинних символів. При $\gamma = \varepsilon$ верхній символ просто вилучається і тим самим магазинний список скорочується.

У випадку, коли $a = \varepsilon$, кажуть, що має місце ε-такт. У ε-такті поточний вхідний символ не розглядається і вхідна головка не зсовується. Однак, стан керуючого пристрою і вміст магазину перетворюються згідно з функцією *D*.

Початковою конфігурацією МП-автомату називається конфігурація вигляду (q_0, w, Z_0), де $w \in T^*$, тобто керуючий пристрій знаходиться в початковому стані, вхідна стрічка містить ланцюжок, який потрібно проаналізувати, а в магазині міститься тільки початковий символ Z_0.

Заключна конфігурація - це конфігурація вигляду (q, ε, u), де $q \in F$, $u \in \Gamma^*$, тобто керуючий пристрій знаходиться в одному із заключних станів, а вхідний ланцюжок повністю прочитаний.

За допомогою МП-автоматів можна розпізнавати рядки мови двома способами.

Перший – рядок вважається розпізнаним, якщо після його прочитання автомат переходить у заключний стан, тобто $(q_0, w, Z_0) \mapsto *(q, \varepsilon, u)$, де $q \in F$, $u \in \Gamma^*$.

Другий – якщо після прочитання рядка стек пам‘яті виявляється порожнім, в цьому випадку говорять, що ланцюжок розпізнається ***спустошенням магазину***. Ці два способи еквівалентні.

Визначення 7.1.3.

Множина усіх рядків, що розпізнається (допускається) МП-автоматом, називається ***мовою МП-автомата*** і позначається L(M).

Приклад 7.1.1.

Розглянемо роботу МП-автомата $M = (Q, T, \Gamma, D, q_0, Z_0, F)$, що розпізнає множину $L(M) = \{0^n 1^n \mid n \geq 0\}$.

Автомат М має множину станів $Q = \{q_0, q_1, q_2\}$, множину вхідних символів $T = \{0,1\}$, множину символів пам‘яті $\Gamma = \{0,1\}$, початковий стан q_0, кінцевий стан q_0, початковий символ магазину *x*. Будемо вважати, що автомат допускає вхідний рядок $a_1, a_2, \ldots a_n \in T^*$, якщо після прочитання цього рядка (тобто послідовного проходження голівкою всіх символів рядка) автомат переходить у кінцевий стан. Функція переходів *D* задається такими значеннями:

$$D(q_0, 0, x) = \{(q_1, 0x)\},$$
$$D(q_1, 0, 0) = \{(q_1, 00)\},$$
$$D(q_1, 1, 0) = \{(q_2, \varepsilon)\},$$
$$D(q_2, 1, 0) = \{(q_2, \varepsilon)\},$$
$$D(q_2, \varepsilon, x) = \{(q_0, \varepsilon)\},$$

Перехід $D(q_0, 0, x) = \{(q_1, 0x)\}$, означає, що якщо автомат знаходиться у стані q_0, одержує на вхід 0 і дістає з пам‘яті поточний символ *x*, то він повинен перейти у стан q_1, повернути у пам‘ять символ *x* і помістити у стек символ 0. Перехід $D(q_2, 1, 0) = \{(q_2, \varepsilon)\}$, означає, що якщо автомат знаходиться у стані q_2 одержує на вхід 1 і дістає з пам‘яті поточний символ 0, то він повинен залишитися у стан q_2, і в пам‘ять нічого не додавати.

Таким чином, на цьому такті із стека “виштовхнутий” і загублений символ 0. При роботі автомат зчитує в пам‘ять першу половину ланцюжка, що складається з нулів, а потім видаляє з пам‘яті по одному нулю на кожну одиницю, що поступає на вхід. Крім того, переходи станів гарантують, що всі 0 передують 1.

Наприклад, для вхідного ланцюжка 0011 автомат здійснить таку послідовність тактів:

$$(q_0, 0011, x) \mapsto (q_1, 011, 0x) \mapsto (q_1, 11, 00x) \mapsto (q_2, 1, 0x) \mapsto (q_0, \varepsilon, \varepsilon)$$

Для кожного такту записана конфігурація, яка включає поточний стан, непрочитану частину вхідного ланцюжка, вміст пам‘яті. Оскільки автомат перейшов в заключний стан (і стек порожній), то 0011 розпізнається.

Ланцюжок 0101 не розпізнається автоматом, оскільки одержуємо конфігурацію, з якої перехід автомату не означений:

$$(q_0, 0101, x) \mapsto (q_1, 101, 0x) \mapsto (q_2, 01, x)$$

Приклад 7.1.2.

Визначити мову, що розпізнається МП-автоматом $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{Z, a, b\}, D, q_0, Z, \{q_2\})$, функція переходів D якого задається так:

$$D(q_0, a, Z) = \{(q_0, aZ)\},$$
$$D(q_0, b, Z) = \{(q_0, bZ)\},$$
$$D(q_0, a, a) = \{(q_0, aa), (q_1, \varepsilon)\},$$
$$D(q_0, a, b) = \{(q_0, ab)\},$$
$$D(q_0, b, a) = \{(q_0, ba)\},$$
$$D(q_0, b, b) = \{(q_0, bb), (q_1, \varepsilon)\},$$
$$D(q_1, a, a) = \{(q_1, \varepsilon)\},$$
$$D(q_1, b, b) = \{(q_1, \varepsilon)\},$$
$$D(q_1, \varepsilon, Z) = \{(q_0, \varepsilon)\},$$

МП-автомат спочатку копіює в магазин деяку частину вхідного ланцюжка з допомогою правил 1, 2, 4, 5 та перших правил 3, 6. В будь-який момент, коли йому захочеться, тільки щоб поточний вхідний символ співпадав з верхнім символом магазину, він може перейти в стан q_1 і почати порівнювати ланцюжок в магазині з залишком вхідного ланцюжка. Цей вибір здійснюють другі правила з 3 і 6, а за правилами 7, 8 відбуваються порівняння. Якщо автомат наштовхується на неспівпадіння, то цей екземпляр МП-автомату відкидається. Якщо деякий вибір тактів призводить до того, що Z знову стає верхнім символом магазину, то за правилом 9 автомат стирає Z і переходить в стан q_2.

Наприклад, МП-автомат розпізнає ланцюжок $abba$:

$$(q_0, abba, Z) \mapsto (q_0, bba, aZ) \mapsto (q_0, ba, baZ) \mapsto (q_1, a, aZ) \mapsto (q_1, \varepsilon, Z) \mapsto (q_2, \varepsilon, \varepsilon)$$

Аналогічно можна показати, що МП-автомат розпізнає всі ланцюжки виду

$$w = c_1 c_2 \ldots c_n c_n c_{n-1} \ldots c_1 \mid c_i = a \vee b, i = 1,2,\ldots, n:$$

$$(q_0, w, Z) \mapsto^{n} (q_0, c_n c_{n-1} \ldots c_1, c_n c_{n-1} \ldots c_1 Z) \mapsto$$

$$(q_0, c_{n-1} \ldots c_1, c_{n-1} \ldots c_1 Z) \mapsto^{n-1} (q_1, \varepsilon, Z) \mapsto (q_2, \varepsilon, \varepsilon)$$

Отже, $L(M) = \{ww^R \mid w \in \{a,b\}^+\}$, де w^R - це ланцюжок w, записаний в зворотному порядку.

Розглянемо фундаментальну властивість поведінки МП-автомату, а саме: те, що відбувається з верхнім символом магазину, не залежить від того, що знаходиться у магазині під цим символом.

7.2.ЗВ‘ЯЗОК КВ ГРАМАТИК ТА МП АВТОМАТІВ

Важливим результатом теорії КВ-граматик є доведення еквівалентності МП-автоматів і КВ-граматик.

Теорема 7.2.1.

Мова є контекстно-вільною тоді і тільки тоді, коли вона розпізнається автоматом з магазинною пам‘яттю.

Доведення.

Нехай $G = (N, T, P, S)$ - КС-граматика.

Побудуємо МП-автомат, що допускає мову $L(G)$ *спустошенням магазину.*

Нехай $M = (\{q\}, T \cup \{\varepsilon\}, N \cup T, D, q, S, \varnothing)$, де D визначається так:

1) якщо $(A \to \alpha) \in P$, то $(q, \alpha) \in D(q, \varepsilon, A)$;

2) $D(q, a, a) = (q, \varepsilon), \forall a \in T$;

Фактично, цей МП-автомат моделює усі можливі виводи в граматиці *G*. З допомогою індукції легко показати, що для будь-якого ланцюжка $w \in T^*$ виведення $S \Rightarrow^+ w$ в граматиці *G* існує тоді і тільки тоді, коли існує послідовність тактів $(q_0, w, S) \mapsto^+ (q, \varepsilon, \varepsilon)$ автомата *M*.

Розглянемо протилежну задачу.

Нехай $M = (Q, T, \Gamma, D, q_0, Z_0, \varnothing)$ - МП-автомат, що розпізнає мову *L*(*M*) спустошенням магазину. Побудуємо граматику *G*, породжуючу ту саму мову *L*(*M*).

Нехай $G = (\{[\, qZr \,] \mid q, r \in Q, Z \in \Gamma\} \cup \{S\}, T, P, S)$, де множина правил *P* наступна:

1. $S \to [q_0 Z_0 q] \in P, \forall q \in Q$
2. Якщо $(r, \varepsilon) \in D(q, a, Z)$, то $[qZr] \to a \in P, a \in T \cup \{\varepsilon\}$
3. Якщо $(r, X_1 \ldots X_k) \in D(q, a, Z), k \geq 1$,

тоді

$$[qZs_k] \to a[rX_1 s_1][s_1 X_2 s_2] \ldots [s_{k-1} X_k s_k]$$

для довільних $s_1, s_2, \ldots, s_k$ станів із Q.

Нетермінали і правила виводу граматики визначаються так, що робота автомата *M* при обробці ланцюжка *w*, відповідає лівосторонньому виводу в граматиці G.

З допомогою індукції можна довести, що $(q, w, A) \mapsto^+ (p, \varepsilon, \varepsilon)$ тоді і тільки тоді, коли $[qAp] \Rightarrow^+ w$.

Тоді, якщо $w \in L(G)$, то $S \Rightarrow [q_0 Z_0 q] \Rightarrow^+ w$ для деякого стану $q \in Q$. Отже, $(q_0, w, Z_0) \mapsto^+ (q, \varepsilon, \varepsilon)$ і тому $w \in L(M)$.

Аналогічно, якщо $w \in L(M)$, то $(q_0, w, Z_0) \mapsto^+ (q, \varepsilon, \varepsilon)$.

Значить $S \Rightarrow [q_0 Z_0 q] \Rightarrow^+ w$ і тому $w \in L(G)$.

Визначення 7.2.3.

МП-автомат $M = (Q, T, \Gamma, D, q_0, Z_0, F)$ називається ***детермінованим***, якщо виконуються наступні умови:

1) Множина $D(q, a, Z)$ містить не більше одного елементу, для довільних $q \in Q$, $a \in T \cup \{\varepsilon\}$, $Z \in \Gamma$;
2) Якщо $D(q, \varepsilon, Z) \neq \varnothing$, то $D(q, a, Z) = \varnothing$ для всіх $a \in T$.

Визначення 7.2.3.

Мова, що розпізнається детермінованим МП-автоматом, називається **детермінованою КВ-мовою**.

Приклад 7.2.3.

Побудуємо МП-автомат M, для якого $L(M) = L(G)$, де G - граматика арифметичних виразів: $G = (\{E, T, F\}, \{x, (,), +, *\}, P, E)$ з продукціями $P = \{E \to E + T \mid T; T \to T * F \mid F; F \to (E) \mid x\}$.

Нехай МП-автомат $M = (\{q\}, \{x, +, *, (,)\}, \{E, T, F, x, +, *, (,)\}, D, q, E, \varnothing)$, а функція D визначається таким чином:

$$D(q, \varepsilon, E) = \{(q, E + T), (q, T)\},$$
$$D(q, \varepsilon, T) = \{(q, T * F), (q, F)\},$$
$$D(q, \varepsilon, E) = \{(q, (E)), (q, x)\},$$
$$D(q, a, a) = \{(q, \varepsilon)\}, \forall a \in \{x, +, *, (,)\}$$

При вході $x + x * x$ для M серед інших можлива така послідовність тактів:

$$(q, x + x * x, E) \mapsto (q, x + x * x, E + T) \mapsto (q, x + x * x, T + T) \mapsto (q, x + x * x, F + T)$$
$$\to (q, x + x * x, x + T) \mapsto (q, +x * x, +T) \mapsto (q, x * x, T) \mapsto (q, x * x, T * F) \mapsto$$
$$(q, x * x, F * F) \mapsto (q, x * x, x * F) \mapsto (q, *x, *F) \mapsto (q, x, F) \mapsto (q, x, x) \mapsto (q, \varepsilon, \varepsilon)$$

Як бачимо, робота МП-автомату M відповідає лівому виводу рядка $x + x * x$ з символу E у КВ-граматиці G.

Тип синтаксичного аналізу, що виконує МП-автомат, побудований за останньою теоремою, називається ***"нисхідним аналізом",*** або аналізом "зверху вниз", чи прогнозуючим. Під час такого аналізу дерево виводу будується, починаючи з кореня, у напрямку зверху вниз, і кожний такт можна трактувати як передвіщення чергового кроку лівого виводу.

Нисхідний МП-автомат $M = (\{q\}, T \cup \{\varepsilon\}, N \cup T, D, q, S, q)$, що розпізнає мову КВ-граматики $G = (N, T, P, S)$ моделюється з одним станом q, вхідним алфавітом є термінальні символи Т граматики, алфавіт магазинних символів складається з термінальних та нетермінальних символів: $Z = N \cup T$. Початкова конфігурація визначається так: (q, α, S) – автомат перебуває в своєму єдиному стані q, зчитуюча головка знаходиться на початку ланцюжка $\alpha \in T^*$. В стеку

лежить початковий символ S. Кінцева конфігурація автомата визначається так: (q, λ, λ) – автомат перебуває в своєму єдиному стані q, зчитуюча головка знаходиться під порожнім символом (за кінцем ланцюжка), стек порожній. Функція переходів МП-автомата будується на основі правил граматики:

1) $(q, \alpha) \in D(q, \lambda, A)$, $A \in N$, $(\alpha \in (N \cup T)^*$, якщо $(A \to \alpha) \in P$;
2) $D(q, a, a) = (q, \lambda)$, $\forall a \in T$.

Роботу даного МП-автомату неформально можна описати так:

- якщо в магазині знаходиться нетермінальний символ A, то його можна замінити на ланцюжок α, не рухаючи головку зчитування, якщо в граматиці G є правило $A \to \alpha$
- якщо в магазині знаходиться термінальний символ a, що співпадає з текучим символом вхідного ланцюжка, то цей символ можна викинути з магазину і пересунути головку зчитування на одну позицію вправо

Можна побудувати розширений МП-автомат, який працює "знизу вверх", як "висхідний аналіз", моделюючи в зворотньому порядку правосторонні виводи в КВ-граматиці.

Висхідний МП-автомат $M = (\{q\}, T \cup \{\varepsilon\}, N \cup T, D, q, S, q)$, що розпізнає мову КВ-граматики $G = (N, T, P, S)$ будується на основі розширеного МП-автомата з одним станом q. Вхідний алфавітом автомата складається з термінальнальних символів T граматики, алфавіт магазинних символів складається з термінальних та нетермінальних символів: $Z = N \cup T$. Початкова конфігурація визначається так: (q, α, λ) – автомат перебуває в своєму єдиному стані q, зчитуюча головка знаходиться на початку ланцюжка $\alpha \in T^*$, стек порожній. Кінцева конфігурація автомата визначається так: (q, λ, S) – автомат перебуває в своєму єдиному стані q, зчитуюча головка знаходиться під порожнім символом (за кінцем ланцюжка), в магазині знаходиться початковий сомвол S. Функція переходів МП-автомата будується на основі будується на основі правил граматики:

1) $(q, A) \in D(q, \lambda, \gamma)$, $A \in N$, $(\gamma \in (N \cup T)^*$, якщо $(A \to \gamma) \in P$;

2) $D(q, a, \lambda) = (q, a), \forall a \in T$.

Неформально роботу даного МП-автомату можна описати так:

- якщо в магазині знаходиться ланцюжок γ, то його можна замінити на нетермінальний символ A, не рухаючи при цьому головку зчитування, якщо в граматиці G є правило $A \to \gamma$
- якщо зчитується деякий символ a вхідного ланцюжка, то його можна помістити в магазин та зсунути головку зчитування на одну позицію вправо.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення МП-автомату.
2. Що таке конфігурація МП-автомату?
3. Що таке такт роботи МП-автомату
4. Який ланцюжок вважається розпізнаним МП-автоматом.
5. Дайте визначення мови МП-автомату.
6. Сформулюйте зв‘язок КВ граматик та МП-автоматів.
7. Що таке висхідний МП автомат?
8. Що таке нисхідний автомат

ЗАВДАННЯ І ВПРАВИ ДО РОЗДІЛУ 7.

Побудувати МП автомати, що визначають мови

1. $\{ w\, w^R : \ w \in \{a,b\}^*, w^R -$ дзеркальне відображення w;
2. Мову з 0 та 1 з однаковою кількістю тих і других.
3. $\{ \{a,b\}^* \setminus \{a^m\, b^n\, a^m\, b^n\}, m,n \geq 1 \}$;
4. $\{ \{a,b\}^* \setminus \{a^m\, b^n\, a^m\} , m,n \geq 1 \}$;
5. $\{ \{a,b\}^* \setminus \{ww, \ w \in \{a,b\}^*\} \}$;
6. $(\{a^n b^n\, c^m \mid n,m \geq 1 \}) \ \cup \ (\{a^m b^n\, c^n \mid n,m \geq 1 \})$;
7. $(\{a^n c^k\, b^n \mid k,n \geq 1 \})$;
8. $\{ w\, c\, w^R : \ w \in \{a,b\}^*, w^R -$ дзеркальне відображення w;
9. $(\{0^n\, 1^n \mid n \geq 1 \})$;
10. $\{ x\, c\, x^R y\, c\, y^R : \ x,y \in \{a,b\}^*$

Розділ 8. НИСХІДНИЙ СИНТАКСИЧНИЙ АНАЛІЗ

В попередньому розділі ми дали визначення нисхідного та висхідного синтаксичного аналізу. В нашому підручнику нисхідний аналіз представлений прогнозуючим аналізатором та

Для використання цього методу бажано звести КВ граматики до певного виду. Для цього пропонуються методи усунення лівої рекурсії та лівої факторизації в КВ граматиках.

Дамо декілька важливих визначень, які необхідні при розгляді нисхідного синтаксичного аналізу.

Сентенціальной формою (sentential form) граматики G = (N, T, P, S) називається будь-яке слово в алфавіті $N \cup T$, виведене з початкового символу S.

Нехай задана контекстно-вільна граматика G = (N, T, P, S). Визначимо три функції FIRST_G і FOLLOW_G. Для стислості будемо писати просто FIRST, FOLLOW.

Функція FIRST ставить у відповідність кожному слову $w \in (N \cup T)^*$ множину тих термінальних символів, з яких починаються слова, виведені з w , тобто

$$\text{FIRST}(w) = \{ b \in T, \text{якщо } \exists\, \theta \in (T \cup N)^*, \text{таке що}, w \Rightarrow^* b\theta \},$$

Функція FOLLOW ставить у відповідність кожному нетермінальному символу A множину тих термінальних символів, які можуть зустрічатися в сентенціальних формах безпосередньо праворуч від A, тобто

$$\text{FOLLOW}(w) = \{ b \in T, \text{якщо } \exists\, \beta \in (T \cup N)^*, \exists\, \theta \in (T \cup N)^*, \text{такі що}, w \Rightarrow^* \beta A b \theta.$$

8.1. LL(1) ГРАМАТИКИ. ЛІВА РЕКУРСІЯ В КВ ГРАМАТИКАХ

Існують інші методи синтаксичного аналізу, крім алгоритму Кока Янгера Касамі. Вони набагато оптимальніші по часу виконання та ємності пам'яті. Але ці методи вимагають спеціального виду КВ граматик, хоча більшість КВ граматик вдається достатньо ефективно перетворити до такого виду.

Одним з таких видів граматик є так звані LL(1)-граматики. Граматика G = (N, T, P, S) являється LL(1)-граматикою тоді і тільки тоді, коли для кожної пари правил $A \to \alpha$, $A \to \beta$ з P (тобто правил з однаковою лівою частиною) виконуються наступні 2 умови:

(1) $FIRST\ (\alpha) \cap FIRST\ (\beta) = \varnothing$, тобто перші символи ланцюгів α,β не співпадають

(2) якщо $\varepsilon \in FIRST\ (\alpha)$, то $FIRST\ (\alpha) \cap FOLLOW\ (A) = \varnothing$, тобто якщо пустий символ може бути виведеним з A, то перший символ з β та наступний символ, який слідує за A в будь-якому виводі граматики, не співпадають.

Інколи за допомогою деяких простих перетворень граматику, що не є LL(1), можна привести до еквівалентної LL(1)-граматики. Серед цих перетворень найбільш ефективними являються ліва факторизація та видалення лівої рекурсії.

Тут необхідно зробити два *зауваження.*

По-перше, не всяка граматика після цих перетворень стає LL(1), і, по-друге, після таких перетворень отримана граматика може стати менш зрозумілою.

Алгоритм 8.1.1. видалення безпосередньої лівої рекурсії.

Безпосередню ліву рекурсію, тобто рекурсію виду, $A \to A\alpha$ можна видалити наступним способом (процедура 1).

Спочатку групуємо A -правила:

$$A \to A\alpha_1 \,|A\alpha_2 \,| \ldots |A\alpha_m \,|\, \beta_1 \,|\, \beta_2 \,|\, \ldots \,|\, \beta_n$$

де ніяка з строк β_i не починається з A. Далі замінюємо цей набір правил на

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A'$$

де A' - новий нетермінал.

З нетерміналу A можна вивести ті ж ланцюжки, що і раніше, але тепер немає лівої рекурсії.

З допомогою цієї процедури *видаляються всі безпосередні ліві рекурсії*, але не видаляється ліва рекурсія, яка проявляється через застосування декількох правил під час виводу. Наведений нижче алгоритм 8.1.1 дозволяє видалити всі ліві рекурсії з граматики.

Алгоритм 8.1.2. Видалення лівої рекурсії.

Вхід. КВ-граматика G без ε -правил (виду $A \rightarrow \varepsilon$).

Вихід. КВ-граматика G' без лівої рекурсії, еквівалентна G.

Метод. Виконати кроки 1 і 2.

(1) Впорядкувати нетермінали граматики G в довільному порядку.

(2) Виконати наступну процедуру:

```
for (i = 1; i <= n; i++)
    {   for (j = 1; j = i − 1; j++)
            {нехай A_j → β1 | β2 | ... | βk всі правила для A_j
             замінити всі правила виду A_i → A_j α
              на правила A_i → β1α | β2α | ... | βkα
            }
        видалити правила виду A_i → A_j
        видалити безпосередню ліву рекурсію в правилах для A_i
        }
```

Зовнішній цикл по i вибирає черговий нетермінал A_i з впорядкованої довільним чином множини нетерміналів і перевіряє наявність правил виду $A_i \to A_j\alpha$ (1), де j < i, тобто A_j уже оброблені на ліву рекурсію нетермінали. Далі видаляються ланцюгові правила $A_i \to A_j$, які могли утворитись в результаті заміни правил

$A_i \to A_j\alpha$ на правила виду

$A_i \to \beta_1\alpha \mid \beta_2\alpha \mid \ldots \mid \beta_k\alpha$.

Також в результаті такої заміни правил могла утворитись безпосередня ліва рекурсія виду $A_i \to A_i\alpha$, яка видаляється згідно алгоритму 8.1.1 .

Алгоритм 8.1.1 застосовується, якщо граматика не має ε - правил. Якщо такі правила є, то граматика зводиться до еквівалентної їй граматики без ε - правил. Отримана граматика без лівої рекурсії може мати ε -правила.

8.2.ЛІВА ФАКТОРИЗАЦІЯ КВ ГРАМАТИК

Основна ідея лівої факторизації в тому, що якщо не зрозуміло яку з двох альтернатив потрібно використовувати для розгортки нетерміналі А, потрібно змінити А -правила таким чином, щоб відкласти рішення до тих пір, поки не буде достатньо інформації для прийняття правильного рішення.

Якщо $A \to \alpha\beta_1 \mid \alpha\beta_2$ - два -правила для нетермінала А і вхідний ланцюг починається з послідовності символів, які співпадають з α , ми не знаємо, чи розгортати вивід по першому правилу чи по другому. Можна відкласти рішення, розгорнувши $A \to \alpha A'$. Далі після аналізу того, що слідує у вхідному ланцюгу після α можна розгорнути вивід по $A' \to \beta_1$ або по $A' \to \beta_2$. Лівофакторизовані правила будуть виглядати так:

$$A \to \alpha A'$$

$$A' \to \beta_1 \mid \beta_2$$

Алгоритм 8.2.1. Ліва факторизація КВ граматики.

Вхід. КВ-граматика G.

Вихід. Лівофакторизована КВ-граматика G', еквівалентна G.

Метод.

Для кожного нетерміналу А знайти самий довгий префікс α спільний для двох або більше його альтернатив. Якщо $\alpha \neq \varepsilon$, тобто існує нетривіальний спільний префікс, то всі А –правила $A \to \alpha\beta_1 \mid \alpha\beta_2 \mid , , \mid \alpha\beta_n \mid z$, тут z позначає всі альтернативи, які не починаються з α , замінити на

$$A \to \alpha_1 A' \mid z$$
$$A' \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

де A' - новий нетермінал.

Застосовувати це перетворення, поки ніякі дві альтернативи не будуть мати спільний префікс.

Приклад 8.2.1

Розглянемо граматику умовних операторів.

G = ({S, E}, {if, then, else, a, b}, P, S) з правилами:

$S \rightarrow$ if E then S | if E then S else S | a

$E \rightarrow$ b

Ця граматика являється неоднозначною (а отже і не LL(1) граматикою), що ілюструється наступними двома різними лівими виводами.

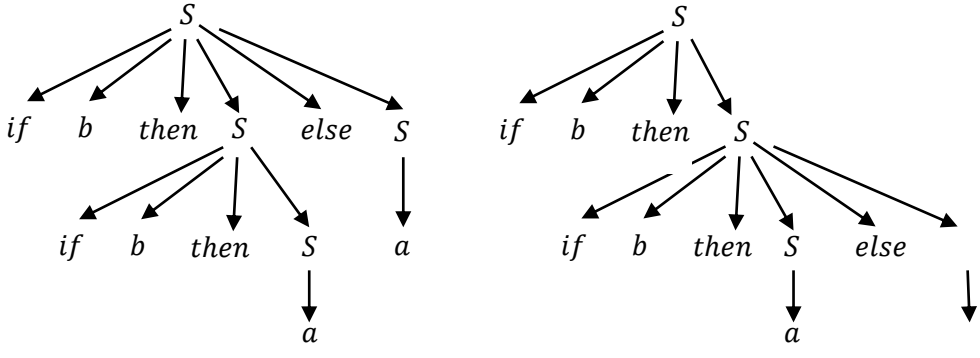
Після лівої факторизації граматика приймає вид

$S \rightarrow$ if E then SS' | a

$S' \rightarrow$ else S | e

$E \rightarrow$ b

Нажаль, граматика залишається неоднозначною, а значить, і не LL(1)-граматикою.



ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ
Ліва рекурсія та ліва факторизація в КВ граматиках

В наступних завданнях необхідно здійснити перетворення КВ граматик до еквівалентних граматик без ліворекурсивних правил

Варіант 1. Видалити ліву рекурсію в граматиці G S→SabA | ba; A→AbA|bAa|c	***Варіант 2.*** Видалити ліву рекурсію в граматиці G S-> Aa |Ab |c; A-> SS |d
Варіант 3. Видалити ліву рекурсію в граматиці G S→SbAc | adA A→BcS |Babc B→Ac | Ad|b D→ccC | DdA|d	***Варіант 4*** Видалити ліву рекурсію в граматиці G S→SabA | ba; A→AbA|bAa|c
Варіант 5. Видалити ліву рекурсію в граматиці G S→abaA|abbA A→ ε | Aa |Ab	***Варіант 6.*** Видалити ліву рекурсію в граматиці G S→ba|A A→ a | Aab |Ab
Варіант 7. Видалити ліву рекурсію в граматиці G S→aaS | abA A→ ε | Aa | Ab	***Варіант 8.*** Видалити ліву рекурсію в граматиці G S→Ba| Ab A→ Sa | AAb | a B→SB | BBa |b

Варіант 9. Видалити ліву рекурсію в граматиці G S→ABS A→ abA | Aa B→Ba | Bab |a	***Варіант 10.*** Видалити ліву рекурсію в граматиці G S→A | B A→ AA | a B→aB | b | Ba
Варіант 11. Видалити ліву рекурсію в граматиці G S→A B A→ Aa | bA | ε B→Bb | aB | b	***Варіант 12.*** Видалити ліву рекурсію в граматиці G S→A B A→ aA | Ab | a B→Bb | Bb | ε
Варіант 13. Видалити ліву рекурсію в граматиці G S→ b | C C→ Ca | AC | c A→Aa | bA | a	***Варіант 14.*** Видалити ліву рекурсію в граматиці G S→SabA | ba; A→AbA|bAa|c

2. В наступних завданнях необхідно здійснити ліву факторизацію для наступних КВ граматик

Варіант 1. S → Aac |Ac A → BSa |BSc |b B → Ac |a	***Варіант 2.*** Граматика задає правила побудови арифметичних операторів S → id + id | id * id S → (S) id → a|b|..|z
Варіант 3. Граматика задає правила побудови логічних виразів в мовах програмування S → id OR id | id AND id | NOT(id) S → (S) id → a|b|..|z	***Варіант 4.*** Граматика задає правила побудови операторів порівняння в мовах програмування S → id<id|id >id|id >=id | id <= id | id = id S → (S) id → a|b|..|z
Варіант 5. Граматика задає правила побудови арифметичних операторів з операціями віднімання та ділення в мовах програмування S → id - id | id / id S → (S) id → a|b|..|z	***Варіант 6.*** Граматика задає правила побудови регулярних виразів з операціями +, • , конкатенація, ітерація (*) в регулярних виразах S → id + id | id • ·id | id * S → (S) id → a|b|..|z

Варіант 7. Граматика задає правила побудови операторів розгалужень з операціями порівння в якості умови такого оператора в мовах програмування S→if B then S | if b then S else S | S B → id < id | id > id | id > =id | id <= id | id = id id → a|b|..|z	***Варіант 8.*** Граматика задає правила побудови виразів для теоретико-множинних операцій S → id ⋃ id | id ⋂ id | id \ id S → (S) id → A|B|..|Z
Варіант 9. Граматика задає правила побудови операторів розгалужень з операціями алгебри логіки в логічних умовах такого оператора в мовах програмування S→if B then S | if b then S else S | S B → id OR id | id AND id | NOT(id) B → (B) id → a|b|..|z	***Варіант 10.*** Граматика задає правила побудови арифметичних операторів в мовах програмування E → E+E E → E-E E → E*E E → (E) E → A|B|..|Z
Варіант 11. <формула> → <число>| (<формула> + <формула>) | (<формула> - <формула>) <число>::= + <цифра> <число> | - <цифра> <число> <цифра>::= 0|1|2|3|4|5|6|7|8|9	***Варіант 12.*** Граматика задає правила побудови арифметичних операторів (операції +,-,*,/, піднесення до степеня) E → E+E | E-E | E*E | E/E | $(E)^{id}$ | id |(E) Id → a|b|..|z

Варіант 13.	*Варіант 14*
операторів присвоєння програмування S → S := B ; | S := E E → E+E |E-E |E*E | (E) E → A|B|..|Z B → id OR id | id AND id | NOT(id) B → (B) id → a|b|..|z	Граматика арифметичних операторів з операціями віднімання, ділення, піднесення до степені S → id - id | id / id|id^n S → (S) id → a|b|..|z n → 1|2|…|9

8.3.ПОБУДОВА ПРОГНОЗУЮЧОГО АНАЛІЗАТОРА

8.3.1. Принципи побудови прогнозуючого аналізатора

Нехай задана КВ-граматика G = (N; T; P; S). Розглянемо *розбір зверху-вниз (прогнозуючий розбір)* для граматики G.

Головна задача *прогнозуючого розбору* - визначення правила граматики, яке потрібно застосувати в ситуації, коли вказівник розбору оглядає черговий термінальний символ вхідного ланцюжка. Процес прогнозуючого розбору з точки зору побудови дерева розбору проілюстровано на Рис.8.3.1.

Фрагменти недобудованого дерева відповідають сентенціальним формам. Спочатку дерево складається тільки з однієї вершини , що відповідає аксіомі S. В цей момент по першому символу вхідного ланцюжка прогнозуючий аналізатор повинен визначити правило $S \to X_1 X_2 \ldots$; яке повинно бути застосовано до S. Далі необхідно визначити правило, яке потрібно застосувати до X_1, і т.д., до тих пір, поки в процесі такої побудови сентенціальної форми, яка відповідає лівому виводу, не буде застосовано правило $Y \to a \ldots$: Цей процес далі застосовується для наступного крайнього лівого нетермінального символу сентенціальної форми.

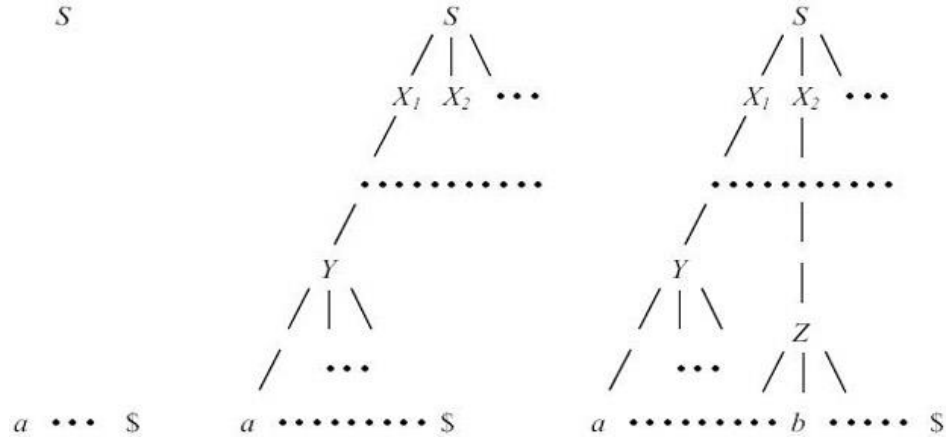


Рис. 8.3.1. Сентенціальні форми виводу.

На Рис. 8.3.2. умовно показана структура прогнозуючого аналізатора, який визначає чергове правило з допомогою таблиці. Таку таблицю можна побудувати безпосередньо по граматиці.

Таблично-керований прогнозуючий аналізатор має вхідну ленту, управляючий пристрій (программу), таблицю аналізу, магазин (стек) і вихідну ленту. Вхідная лента містить строку, що аналізується та закінчується символом $ - маркером кінця строки. Вихідна лента містить послідовність застосованих правил виводу.

Таблиця аналізу - це двохвимірний масив M[A; a], де A - нетермінал, a - термінал або символ $. Значенням *M[A; a] може бути деяке правило граматики* або елемент "помилка".

Магазин містить послідовність термінальних та не термінальних символів граматики та спеціальний маркер $ на дні магазину. В початковий момент магазин містить тільки початковий символ граматики (аксіому) у верхівці магазину та $ на дні.

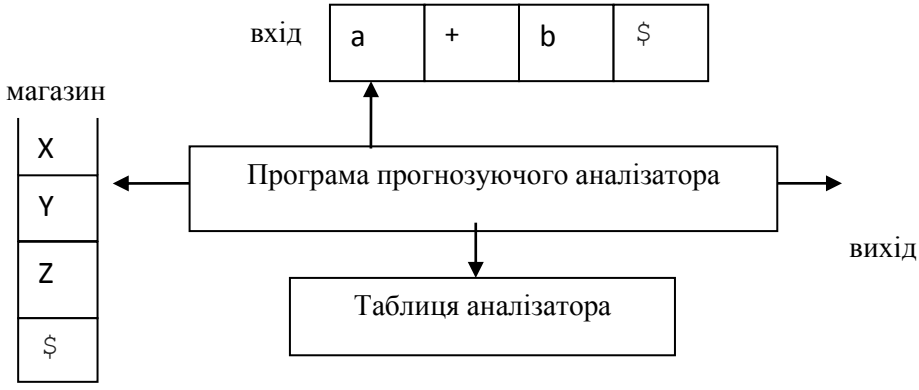


Рис. 8.3.1. Схема функціонування прогнозуючого аналізатора

Аналізатор працює наступним чином. Спочатку аналізатор знаходиться в конфігурації, в якій магазин містить S$, на вхідній ленті w$ (w – ланцюг, що аналізується), вихідна лента порожня. На кожному такті аналізатор розглядає X - символ у верхівці магазину і a - біжучий вхідний символ. Ці два символи визначають дії аналізатора. Існують наступні можливості.

1. Якщо $X = a = \$$, аналізатор зупиняється, повідомляє про успішне завершення розбору і видає вміст вихідної ленти.
2. Якщо $X = a \neq \$$, аналізатор видаляє X з магазину і просуває вказівник входу на наступний вхідний символ.
3. Якщо X - термінал, і $X \neq a$, то аналізатор зупиняється і повідомляє про те, що вхідний ланцюг не належить мові.
4. Якщо X - нетермінал, аналізатор заглядає в таблицю M[X; a].

Можливі два випадки:

- Значенням M[X; a] являється правило для X. В цьому випадку аналізатор замінює X на верхівці магазину на праву частину даного правила, а саме правило розміщує на вихідну ленту. Вказівник входу не пересувається.
- Значенням M[X; a] являється "помилка". В цьому випадку аналізатор зупиняється і повідомляє про те, що вхідний ланцюг не належить мові.

Приклад 8.3.1.

Розглянемо граматику арифметичних виразів
G=({E; E', T, T', F}, {id, +, *, (,)}, P, E)з правилами:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE'$$
$$E' \rightarrow \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT'$$
$$T' \rightarrow \varepsilon$$
$$F \rightarrow (E)$$
$$F \rightarrow id$$

В таб. 8.3.1 наведена таблиця прогнозуючого аналізатора для цієї граматики. Пусті клітинки таблиці відповідають елементу "помилка".

Таблиця 8.3.1.

Нетермінал	Вхідний символ					
	Id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → e	E' → e
T	T → FT'			T → FT'		
T'		T' → e	T' → *FT'		T' → e	T' → e
F	F → id			F → (E)		

При розборі вхідного ланцюжка **id + id * id$** аналізатор здійснює послідовність кроків, зображену в табл. 8.3.2.

Зауважимо, що аналізатор здійснює лівий вивід. Дерево розбору для цього ланцюжка наведено на Рис. 8.3.2.

Таблиця 8.3.2.

Магазин	Вхід	Вихід
E$	id + id * id$	
TE'$	id + id * id$	E → TE'
FT'E'$	id + id * id$	T → FT'
id T'E'$	id + id * id$	F → id
T'E'$	+id * id$	
E'$	+id * id$	T' → e
+TE'$	+id * id$	E' → +TE
TE'$	id * id$	
FT'E'$	id * id$	T → FT'
id T'E'$	id * id$	F → id
T'E'$	*id$	
*F'T'E'$	*id$	T' → *FT'
FT'E'$	id$	
id T'E'$	id$	F → id
T'E'$	$	
E'$	$	T' → e
$	$	E' → e

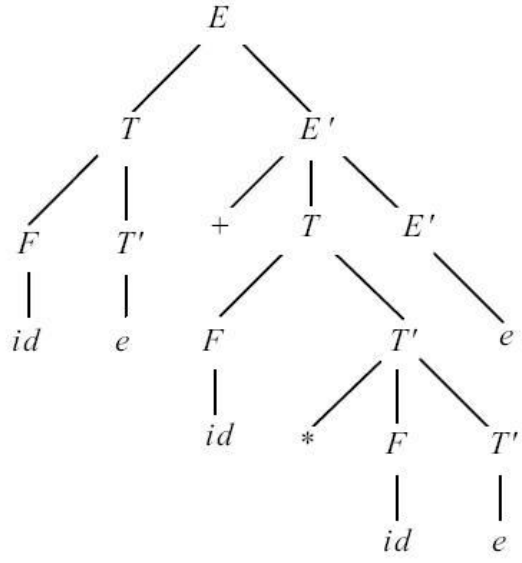


Рис. 8.3.2. Дерево розбору

8.3.2.Функції FIRST та FOLLOW

Для побудови таблиці прогнозуючого аналізатора нам будуть потрібні дві функції - FIRST та FOLLOW. Розглянемо алгоритми побудови цих функцій

Нехай G = (N, T, P, S) - КВ-граматика. Для α- довільного ланцюжка, який складається з термінальних і нетермінальних символів граматики, визначимо $FIRST\ (\alpha)$ як множину терміналів, з яких починаються строки, які виводяться з α. Якщо $\alpha \Rightarrow^{*} \varepsilon$, то ε також належить $FIRST\ (\alpha)$.

Визначимо FOLLOW(A) для нетерміналу A як множину терміналів a, які можуть з'явитись безпосередньо справа від A в деякому виводі граматики, тобто FOLLOW(A) є множиною терміналів $a \in T$ таких, що існує вивід виду $S \Rightarrow^{*} \alpha A a \beta$ для деяких $\alpha, \beta \in (N \cup T)^{*}$. Зауважимо, що між A і a в процесі виводу можуть знаходитись нетермінальні символи, з яких виводиться ε. Якщо A може бути самим правим нетермінальним символом деякого виводу граматики, то $ також належить FOLLOW(A).

Розглянемо алгоритми обчислення функції FIRST.

Алгоритм 8.3.1 *Обчислення FIRST для символів КВ- граматики.*

Вхід. КВ-граматика G = (N, T, P, S).

Вихід. Множина FIRST(X) для кожного термінального і нетермінального символу $X \in (N \cup T)$.

Метод. Виконати кроки 1-4:

1. Для всіх терміналів $a \in T$ покласти $FIRST(a) = \{a\}$, для всіх нетерміналів $A \in N$ визначити $FIRST(A) = \varnothing$
2. Для всіх ε -правил виду $A \to \varepsilon$ додати ε до FIRST(A) $= \{\varepsilon\}$
3. Поки ні до якої множини FIRST(X) не можна буде додати нові елементи, виконувати 4:
4. Вибрати правило: для кожного нетерміналу A та для кожного правила цього нетерміналу виду $A \to B_1 \ldots B_k$

4.1. Якщо $\varepsilon \notin$ FIRST (B_1), то додати до FIRST(A) всі елементи з FIRST(B_1), перейти на крок 4, вибравши наступне правило, інакше перейти на крок 4.2

4.2. Якщо $\varepsilon \in$ FIRST (B_1), то додати до FIRST(A) всі елементи з FIRST(B_1) за виключенням ε та перейти на крок 4.1, розглянувши замість FIRST(B_1) уже наступну множину FIRST(B_2).

Крок 4.1-4.2 повторювати до тих пір поки не будуть перебрані всі нетермінали B_i з правої частини правила $A \to B_1 \ldots B_k$ або наступить ситуація $\varepsilon \notin$ FIRST (B_1), яка приведе до вибору наступного правила і початку кроку 4.

Якщо алгоритм виконувався хоча б один раз по кроку 4.2 – по завершенню розгляду правила $A \to B_1 \ldots B_k$ додати до FIRST(A) ε

Крок алгоритму 4 повторюється циклічно з першого по останнє правило до тих пір поки жодна з множин FIRST(A) (A –нетермінал) не поповниться новими елементами.

Далі побудуємо всі множини FIRST($X_1X_2\ldots X_k$), де $X_1X_2\ldots X_k$ - ланцюжок, що є фрагментом правих частин правил граматики або

зустрічається в виводах граматики. Такі множини використовуються для побудови множин FOLLOW для кожного нетермінала граматики.

Алгоритм 8.3.2. *Обчислення FIRST для ланцюжка.*
Цей алгоритм використовується для обчислення далі функції FOLLOW.

Вхід. КВ-граматика G = (N, T, P, S).

Вихід. Множина $FIRST\ (X_1 X_2 \dots X_n)$, $X_i \in (N \cup T)^*$.

Метод. Виконати кроки 1-3:

(1) З допомогою **алгоритму 8.3.1** обчислити FIRST(X) для кожного $X \in (N \cup T)$..

(2) Вибрати ланцюжок, покласти $FIRST\ (X_1 X_2 \dots X_n) = \varnothing$.Позначимо через *u* ланцюжок $X_1 X_2 \dots X_n$.

(3) для X з ланцюжка $X_1 X_2 \dots X_n$ виконати:

3.1. Якщо $\varepsilon \notin$ FIRST (X_1), то додати до $FIRST\ (X_1 X_2 \dots X_n)$ всі елементи з FIRST(X_1), перейти на крок 2, інакше перейти на крок 3.2

3.2 Якщо $\varepsilon \in$ FIRST (X_1), то додати до $FIRST\ (X_1 X_2 \dots X_n)$ всі елементи з FIRST (X_1) за виключенням ε та перейти на крок 3.1, розглянувши замість FIRST (X_1) уже наступну множину FIRST(X_2). Кроки 3.1-3.2 повторювати до тих пір поки не будуть перебрані всі нетермінали X_i з ланцюжка $X_1 X_2 \dots X_n$ або наступить ситуація $\varepsilon \notin$ FIRST (X_i).

Якщо алгоритм виконувався хоча б один раз по кроку 3.2 то по завершенню кроку 3 додати до $FIRST\ (X_1 X_2 \dots X_n)$ елемент ε .

Розглянемо алгоритм обчислення функції FOLLOW.

Алгоритм 8.3.3. Обчислення FOLLOW для **нетерміналів** граматики.

Вхід. КВ-граматика G = (N, T, P, S).

Вихід. Множина *FOLLOW(X)* для кожного символу $X \in N$ . .

Метод. Виконати кроки 1-4:

(1) Покласти $FOLLOW\ (X) = \varnothing$ для кожного нетермінального символу $X \in N$.

(2) Додати символ кінця вхідної строки $ до FOLLOW(S).

(3) Якщо в P є правило виводу $A \to \alpha B \beta$, де $\alpha, \beta \in (N \cup T)^*$, то всі елементи з $FIRST\ (\beta)$, за виключенням ε , додати до *FOLLOW(B).* Тут β ланцюжок термінальних та нетермінальних символів, для побудови $FIRST\ (\beta)$ використовується алгоритм 8.3.2.

(4) Поки нічого не можна буде додати ні до якої множини *FOLLOW(X),* виконувати:

Якщо в P є правило $A \to \alpha B$ або $A \to \alpha B \beta$, $\alpha, \beta \in (N \cup T)^*$, де $FIRST\ (\beta)$ містить ε $(\beta \Rightarrow^* \varepsilon)$, то всі елементи з *FOLLOW(A)* додати до *FOLLOW(B).* Для розуміння цього факту достатньо в вивід $S \Rightarrow \alpha_1 A \alpha_2$ підставити замість A праву частину правила $A \to \alpha B$, отримаємо $S \Rightarrow \alpha_1 \alpha B \alpha_2$. Зрозуміло, що *FOLLOW(A),* куди входять перші символи виводу з ланцюжка α_2 , містить ті самі символи, які слідуватимуть у виводах після B, тому всі символи з *FOLLOW(A)* повинні додаватись до *FOLLOW(B*).

Приклад 8.3.2.. Розглянемо граматику *з* ***прикладу 8.3.1.***

$$E \to TE'$$
$$E' \to +TE'$$
$$E' \to \varepsilon$$
$$T \to FT'$$
$$T' \to *FT'$$
$$T' \to \varepsilon$$
$$F \to (E)$$
$$F \to id$$

Для цієї граматики для правил, праві частини яких починаються з термінальних символів побудуємо

З правил $F \to (E),\ F \to id$ випливає FIRST(F) = {(, id}

З правил $E' \rightarrow +TE',\ E' \rightarrow \varepsilon$ випливає FIRST(E') = {+, ε }

З правил $T' \rightarrow *FT',\ T' \rightarrow \varepsilon$ випливає (T') = {*, ε }

Для правила $T \rightarrow FT'$ FIRST(T) = FIRST(F) = {(, id}

Для правила $E \rightarrow TE'$ FIRST(E) = FIRST(T) = {(, id}

Побудуємо функції *FOLLOW* для нетерміналів

Поскільки E – аксіома, то початково FOLLOW(E)= {$}. Для подальшої побудови FOLLOW(E) розглянемо всі правила, в яких E зустрічається в правій частині, таке правило єдине $F \rightarrow (E)$, і символ, який буде зустрічатись у виводах після E – це символ), додаємо його до FOLLOW(E)= {),$}.

Розглянемо побудову FOLLOW(E'). З правила $E \rightarrow TE'$, поскільки після E' нічого не слідує, то FOLLOW(E') = FOLLOW(E) ={),$}. Не має інших правил, де б зустрічався E' в правій частині.

Аналогічно

FOLLOW(T) = FOLLOW(T') = {+,), $}

FOLLOW(F) = {+, *,), $}

Наприклад, id і ліва дужка додаються до FIRST(F) на кроці 3 при i = 1, поскільки FIRST(id) = {id} і FIRST(**(**) = { **(** } у відповідності з кроком 1. На кроці 3 при i = 1, у відповідності з правилом виводу T → FT', до FIRST(T) додаються також id і ліва дужка. На кроці 2 в FIRST(E') включається e.

Також при обчисленні множин FOLLOW на кроці 2 в FOLLOW(E) включаються $. На кроці 3, на основі правила F → (E), до FOLLOW(E) додається права дужка. На кроці 4, застосованому до правила E → TE', в FOLLOW(E') включаються $ і права дужка. Оскільки $E' \Rightarrow^* e$, вони також попадають і в множину FOLLOW(T). У відповідності з правилом виводу

E → TE', на кроці 3 в FOLLOW(T) включаються і всі елементи з FIRST(E'), відмінні від ε .

8.3.3. Конструювання таблиці прогнозуючого аналізатора

Для конструювання таблиці прогнозуючого аналізатора по граматиці G може бути використаний алгоритм, що базується на такій ідеї. Допустимо, що $A \to \beta$- правило виводу граматики і $b \in FIRST(R)$. Тоді аналізатор здійснює розгортку A по β, якщо вхідним символом являється b. Складність виникає, коли $\beta = \varepsilon$ або $\beta \Rightarrow^* \varepsilon$. В цьому випадку потрібно розгорнути A в β якщо біжучий символ b належить FOLLOW(A) або якщо досягнуто \$ і $\$ \in FOLLOW(A)$.

Алгоритм 8.3.4. Побудова таблиці прогнозуючого аналізатора.

Вхід. КВ-граматика G = (N, T, P, S).

Вихід. Таблиця M[A;a] прогнозуючого аналізатора, $A \in N, a \in T \cup \$$.

Метод.

Для кожного правила виводу $A \to R$ граматики виконати кроки 1 та 2. Після цього виконати крок 3.

(1) Для кожного терміналу a з FIRST(R) додати правило $A \to R$ до M[A; a].

(2) Якщо $\varepsilon \in FIRST(R)$, додати правило $A \to R$ до M[A; b] для кожного терміналу b, що входить в FOLLOW(A). Крім того, якщо $\varepsilon \in FIRST(R)$ і $\$ \in FOLLOW(A)$, додати A→ R до M[A; \$].

(3) Покласти у всі незаповнені клітинки таблиці M повідомлення "помилка".

Приклад 8.3.3.

Застосуємо **алгоритм 8.3.4**. до граматики з *прикладу* 8.3.2. Оскільки FIRST(TE') = FIRST(T) = {(, id }, у відповідності з правилом виводу E → TE' входи M[E, ( ] та M[E, id] стають рівними E → TE'.

У відповідності з правилом виводу E' → +TE' значення M[E', +] рівне E' → +TE'.

У відповідності з правилом виводу E' → e значення M[E',)] і

M[E', $] рівні E' $\rightarrow \varepsilon$, поскільки FOLLOW(E') = {), $}.

Таблиця аналізу для цієї граматики, наведена в табл. 8.3.3

Табл. 8.3.3

Нетермінал	Вхідний символ					
	Id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → e	E' → e
T	T → FT'			T → FT'		
T'		T' → e	T' → *FT'		T' → e	T' → e
F	F → id			F → (E)		

.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ.

Побудуйте множини FIRST і FOLLOW та таблицю прогнозуючого синтаксичного аналізатору, продемонструйте його роботу на ланцюжку, аналогічно до процедури описаної в лекції - дивитись файл «лекція 7_констр_табл_прогноз_аналізатору».

Варіант 1.	***Варіант 2.***
Ланцюжок abbab , граматика: S → aBS | b A → a B → A | bSB	Ланцюжок виберіть самостійно, граматика: S_0 → S $ S → A*a*S | *b* A → CA*b* | B B → *c*S*a* | ε C → *c* | *ab*

Варіант 3. Дано граматику $S_0 \rightarrow$ begin S end S → if E then S else S | S → begin S L | S → print num L → end | ; S L E → num = num Здійснити розбір ланцюжка begin if num=num then print num ; else begin print num end end	***Варіант 4.*** Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: G = { S →*a*T | TbS, T → *b*T | *ba* } Провести синтаксичний аналіз ланцюжка *bbababba*
Варіант 5. Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: G = {S → BA*b*, A → *a*ABC | *b*B | *a*, B → *b*, C → *c*A } Провести синтаксичний аналіз ланцюжка *Babccaab*	***Варіант 6.*** Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор <програма> → <оператор> <оператор> → begin <опис> : <список операторів> end <опис> → d | d ; <опис> <список операторів> → s ; <список операторів> | ε Провести синтаксичний аналіз ланцюжка begin d; d : s ; s; s; end
Варіант 7. Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: {S→BA, A→BS|d, B→aA|bS|c}. Провести синтаксичний аналіз ланцюжка Bcdcadd	***Варіант 8.*** Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: G = { S' → S$, S → B A, A → -B A \| ε, B → DC, C → /DC \| ε, D → (S) \| *a* } Провести синтаксичний аналіз ланцюжка **a - a / a$**

Варіант 9. Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: S → aS | aA A → bA | aB | aC B → bA | b C → ε | bA Провести синтаксичний аналіз ланцюжка aababba.	***Варіант 10.*** Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: S → abSba | acAb A → bbAa | bcB | a B → bBA | c Провести синтаксичний аналіз ланцюжка acbbbccab.
Варіант 11. Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: S → for S to id do S S → begin S end; S → id:=E E → id+id E → id Провести синтаксичний аналіз ланцюжка *for* x:= m *to* n *do* *begin* x := 3 + x; y :=5 *end*;	***Варіант 12.*** Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: S→if B then S | if b then S else S | S | call id B->id < id | id > id | id > =id | id <= id | id = id id->a|b|..|z Провести синтаксичний аналіз ланцюжка *If* x>0 then call A else *If y=x then* call B; call C;

Варіант 13. Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: S->id ∪ id | id ∩ id | id \ id S->(S) id->A|B|..|Z Провести синтаксичний аналіз ланцюжка (A ∪ **B)** ∩ (C \ **B)**	***Варіант 14.*** Провести ліву факторизацію граматики та побудувати прогнозучий аналізатор: S → S := B ; | S := E E → E+E |E-E |E*E | (E) E → A|B|..|Z B → id OR id | id AND id | NOT(id) B → (B) id → a|b|..|z Провести синтаксичний аналіз ланцюжка S :=a+b*(z+c)

Розділ 9. ВИСХІДНИЙ СИНТАКСИЧНИЙ АНАЛІЗ

9.1. ОСНОВНІ ПОНЯТТЯ ТА ВИЗНАЧЕННЯ

У процесі розбору знизу-вверх за типом зсув-згортка будується дерево розбору вхідного ланцюжка, починаючи з листків (знизу) до кореня (вгору).

Цей процес можна разглядати як «згортка» ланцюжка ω до початкового символу граматики. На кожному кроці згортки підланцюжок, котрий відповідає правій частині деякого правила виведення, замінюється символом лівої частини цього правила виведення, і якщо на кожному кроці вибирається правильний підланцюжок, то в зворотньому порядку простежується правосторонній вивід (мал. 9.1.1). Тут до вхідних ланцюжків, так само як і при аналізі LL(1)-граматик, приписаний кінцевий маркер $.

Основою ланцюжка називається підланцюжок сентенціальної форми, який може бути співставлений з правою частиною деякого правила виводу, згортка за яким до лівої частини цього правила відповідає одному кроку в правосторонньому виводі. Найлівіший підланцюжок, який зіставляється з правою частиною деякого правила виводу $A \to \gamma$, не обов'язково є основою, оскільки згортка за правилом $A \to \gamma$ може дати такий ланцюжок, який не може бути зведений до аксіоми.

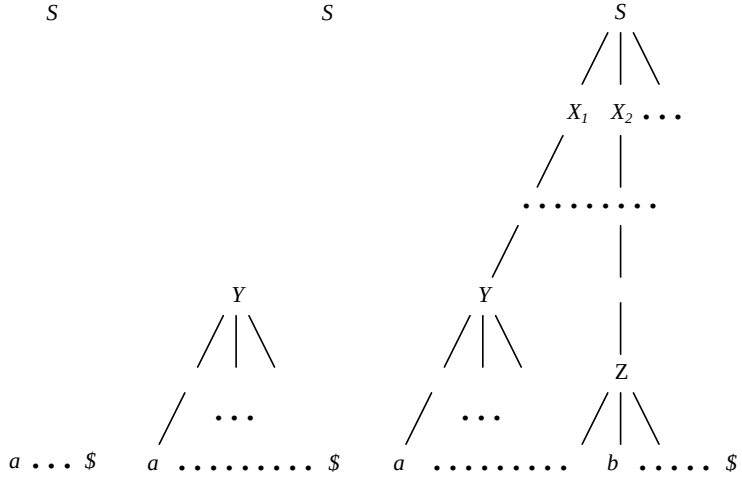


Рис. 9.1.1 Правосторонній вивід

.

Формально, основа правої сентенціальной форми z - це правило виводу $A \to \gamma$ і позиція в z, в якій може бути знайдено ланцюжок γ такий, що в результаті заміни γ на A отримуємо попередню сентенціальну форму в правосторонньому виведенні z. Так, якщо $S \Rightarrow_r^* \alpha A \beta \Rightarrow_r \alpha\gamma\beta$, то $A \to \gamma$ в позиції, наступній за α, то це основа ланцюжка $\alpha\beta\gamma$. Підланцюжок β, що лежить праворуч від основи, містить тільки термінальні символи.

Взагалі кажучи, граматика може бути неоднозначною, тому не єдиним може бути правосторонній вивід $\alpha\beta\gamma$ і не єдиною може бути основа. Якщо граматика однозначна, то кожна права сентенціальна форма граматики має тільки одну основу. Заміна основи в сентенціальній формі на нетермінал лівої частини називається *відсіканням основи*. Згортання правостороннього виведення може бути отримано за допомогою повторного застосування правила відсікання основи, починаючи із вхідного ланцюжка ω.

Якщо ω - слово в розглянутій граматиці, то $\omega = \alpha_n$, де α_n - права сентенціальна форма ще невідомого правостороннього виведення $S \Rightarrow \alpha_0 \Rightarrow_r \alpha_1 \Rightarrow_r \ldots \Rightarrow_r \alpha_{n-1} \Rightarrow_r \alpha_n = \omega$.

Щоб відновити це виведення у зворотньому порядку, виділяємо основу γ_n в α_n і міняємо γ_n на ліву частину деякого правила виводу $A_n \to \gamma_n$, отримуючи $(n-1)$ праву сентенціальну форму α_{n-1}. Потім повторюємо цей процес, тобто знову виділяємо основу γ_{n-1} в α_{n-1} і згортаємо цю основу, отримуючи праву сентенціальну форму α_{n-2}. Якщо, повторюючи цей процес, ми отримаємо праву сентенціальну форму, що складається тільки з початкового символу S, то зупиняємося і повідомляємо про успішне завершення розбору. Згорнута послідовність правил, виконана у виводах, є правостороннім виведенням вхідного ланцюжка.

Таким чином, головне завдання аналізатора типу зсув-згортка - це виділення і відсікання основи.

9.2. LR (1) –АНАЛІЗАТОРИ

У назві LR(1) символ L вказує на те, що вхідний ланцюжок читається зліва-направо, R - на те, що будується правостороннє виведення, нарешті, 1 вказує на те, що аналізатор бачить лише один символ непрочитаної частини вхідного ланцюжка.

Причини привабливості LR(1)-аналізу:

- LR(1)-аналіз - найбільш потужний метод аналізу без повернень типу зсув-згортка;
- LR(1)-аналіз має досить високу ефективність реалізації;
- LR(1)-аналізатори можуть бути побудовані для практично всіх конструкцій мов програмування;
- Класи граматик, які можуть бути проаналізовані LR(1)-методом, строго включають класи граматик, які можуть бути проаналізовані прогнозуючими аналізаторами (зверху-вниз типу LL(1)), тобто клас граматик, які піддаються LR(1)-аналізові є ширшим від класу граматик, що піддаються прогнозуючому аналізові.

Схематично структура LR(1)-аналізатора зображена на рис. 9.2.1. Аналізатор складається з вхідної стрічки, вихідної стрічки, магазину, керуючої програми і таблиці аналізу (LR(1) -таблиці), яка має дві частини - функцію дій (Action) і функцію переходів (Goto). Керуюча програма одна і та ж для всіх LR(1)-аналізаторів, різні аналізатори відрізняються тільки таблицями аналізу.

Аналізатор читає символи на вхідній стрічці по одному за крок. У процесі аналізу використовується магазин, в якому зберігаються рядки виду $S_0 X_1 S_1 X_2 S_2 \ldots X_m S_m$ (S_m - верхній символ магазину). Символ X_i - символ граматики (термінальний або нетермінальний), а S_i - символ стану.

Зауважимо, що символи граматики (або символи станів) не обов'язково повинні розміщуватися в магазині. Однак, їх використання полегшує розуміння поведінки LR-аналізатора.

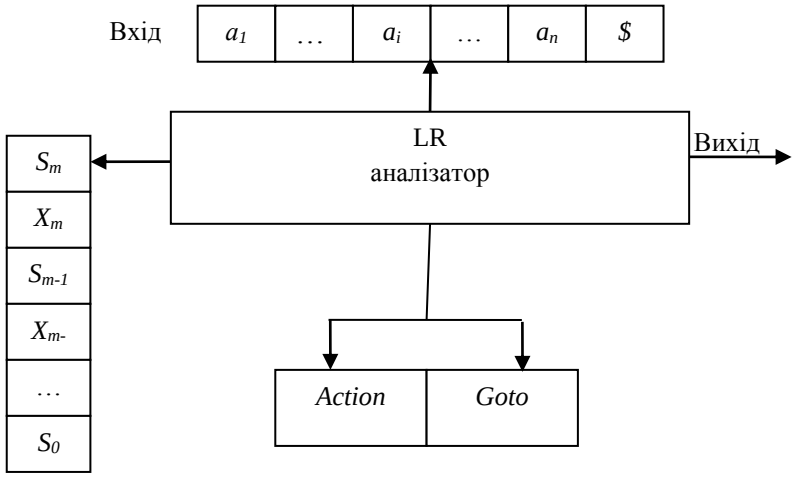


Рис 9.2.1. Структура LR(1)-аналізатора

Елемент функції дій $Action\ [S_m, a_i]$ для символу стану S_m і символу входу $a_i \in T \cup \{\$\}$, може мати одне з чотирьох значень:

1) shift *S* (зсув), де *S* - символ стану,

2) reduce $A \to \gamma$ (згортка за правилом граматики $A \to \gamma$),

3) accept (допуск),

4) error (помилка).

Елемент функції переходів $Goto\,[S_m, A]$ для символу стану S_m і входу $A \in N$, може мати одне з двох значень:

1) *S*, де *S* - символ стану,

2) error (помилка).

Конфігурацією LR(1)-аналізатора називається пара, перша компонента якої - вміст магазину, а друга - непереглянутий вхід:

$$(S_0 X_1 S_1 X_2 S_2 \dots X_m S_m, a_i a_{i+1} \dots a_n \$)$$

Ця конфігурація відповідає правій сентенціальній формі

$$X_1 X_2 \dots X_m a_i a_{i+1} \dots a_n$$

Префікси правих сентенціальних форм, які можуть з'явитися в магазині аналізатора, називаються **активними префіксами**. Основа сентенціальної форми завжди розташовується у верхівці магазину. Таким чином, активний префікс - це такий префікс правої сентенціальної форми, який не переходить праву межу основи цієї форми.

Коли аналізатор починає роботу, в магазині знаходиться тільки символ початкового стану S_0, на вхідній лєнті - аналізований ланцюжок з маркером кінця.

Кожен черговий крок аналізатора визначається поточним вхідним символом a_i і символом стану у верхівці магазину S_m наступним чином.

Нехай LR(1) -аналізатор знаходиться в конфігурації:

$$(S_0 X_1 S_1 X_2 S_2 \dots X_m S_m, a_i a_{i+1} \dots a_n \$)$$

Аналізатор може виконати один з наступних кроків:

1. Якщо $Action[S_m, a_i] = shift\ S$, то аналізатор виконує зсув, переходячи в конфігурацію

$$(S_0 X_1 S_1 X_2 S_2 \dots X_m S_m a_i S, a_{i+1} \dots a_n \$)$$

Тобто, в магазин поміщаються вхідний символ a_i і символ стану *S*, який визначається $Action[S_m, a_i]$. Поточним вхідним символ стає a_{i+1}.

2. Якщо $Action[S_m, a_i] = reduce\ A \to \gamma$, то аналізатор виконує згортку, переходячи в конфігурацію

$$(S_0 X_1 S_1 X_2 S_2 \dots X_{m-r} S_{m-r} A S, a_i a_{i+1} \dots a_n \$)$$

де $S = Goto[S_{m-r}, A]$ і *r* - довжина γ, правої частини правила виводу.

Аналізатор спочатку видаляє з магазину 2*r* символів (*r* символів стану і *r* символів граматики), так що у верхівці появиться стан S_{m-r}. Потім аналізатор поміщає в магазин *A* - ліву частину правила виводу, і *S* - символ стану, який визначається $Goto[S_{m-r}, A]$. На кроці згортки поточний вхідний символ не змінюється. Для LR(1)-аналізаторів послідовність символів граматики $X_{m-r+1} \dots X_m$, що викидаються з магазину, завжди відповідає γ - правій частині правила виводу, за яким робиться згортка. Після здійснення кроку згортки

генерується вихід LR(1) -аналізатора, тобто виконуються семантичні дії, пов'язані з правилом, за яким робиться згортка, наприклад, друкуються номери правил, за якими робиться згортка.

Зауважимо, що функція *Goto* таблиці аналізу, побудованої для граматики *G* фактично є функцію переходів детермінованого скінченного автомата, що розпізнає активні префікси *G*.

3. Якщо $Action[S_m, a_i] = accept$, то розбір успішно завершений.

4. Якщо $Action[S_m, a_i] = error$, то аналізатор виявив помилку і виконуються дії з діагностики та відновлення.

Приклад 9.2.1.

Розглянемо граматику арифметичних виразів $G = (\{E, T, F\}, \{id, +, *\}, P, E)$ з правилами:

(1) $E \to E + T$

(2) $E \to T$

(3) $T \to T * F$

(4) $T \to F$

(5) $F \to id$

На рис. 9.2.2 зображені функції *Action* і *Goto*, що утворюють LR(1) -таблицю для цієї граматики.

Стан	*Action*				*Goto*		
	id	+	*	$	E	T	F
0	S6				1	2	3
1		S4		acc			
2		R2	S7	R2			
3		R4	R4	R4			
4	S6					5	3
5		R1	S7	R1			
6		R5	R5	R5			
7	S6						8
8		R3	R3	R3			

Рис. 9.2.2 Функції Action і Goto, що утворюють LR(1) -таблицю

Елемент s_i функції *Action* означає зсув і переміщення в магазин стану з номером *i*, R_j - згортка за правилом номер *j*, *acc* - допуск, порожня клітинка - помилка. Для функції *Goto* символ *i* означає переміщення в магазин стану з номером *i*, порожня клітинка - помилка.

Активний префікс	Магазин	Вхід	Дія
	0	$id + id * id \$$	зсув
id	$0\ id\ 6$	$+ id * id \$$	$F \to id$
F	$0\ F\ 3$	$+ id * id \$$	$T \to F$
T	$0\ T\ 2$	$+ id * id \$$	$E \to T$
E	$0\ E\ 1$	$+ id * id \$$	зсув
$E +$	$0\ E\ 1 + 4$	$id * id \$$	зсув
$E + id$	$0\ E\ 1 + 4\ id\ 6$	$* id \$$	$F \to id$
$E + F$	$0\ E\ 1 + 4\ F\ 3$	$* id \$$	$T \to F$
$E + T$	$0\ E\ 1 + 4\ T\ 5$	$* id \$$	Зсув
$E + T *$	$0\ E\ 1 + 4\ T\ 5 * 7$	$* id \$$	Зсув
$E + T * id$	$0\ E\ 1 + 4\ T\ 5 * 7\ id\ 6$	$\$$	$F \to id$
$E + T * F$	$0\ E\ 1 + 4\ T\ 5 * 7\ F\ 8$	$\$$	$T \to T * F$
$E + T$	$0\ E\ 1 + 4\ T\ 5$	$\$$	$E \to E + T$
E	$0\ E\ 1$		Допуск

Рис. 9.2.3.Нисхідний аналіз ланцюжка $id + id * id$

На вході $id + id * id$ послідовність станів магазину і вхідної стрічки показані на рис. 9.2.3.. Наприклад, в першому рядку LR-аналізатор знаходиться в нульовому стані і «бачить» перший вхідний символ *id*. Дія s_6 в нульовому рядку і стовпці *id* в поле *Action* (рис9.2.2) означає зсув і переміщення символу

стану 6 на верхівку магазину. Це і зображено у другому рядку: перший символ *id* і символ стану 6 поміщаються в магазин, а id видаляється з вхідної стрічки.

Поточним вхідним символом стає +, і дією в стані 6 на вхід + є згортка за правилом $F \to id$. З магазину видаляються два символи (один символ стану і один символ граматики). Потім аналізується нульовий стан. Оскільки *Goto* в нульовому стані по символу F - це 3, F і 3 переміщаються в магазин. Тепер маємо конфігурацію, що відповідає третьому рядку. Інші кроки визначаються аналогічно.

Конструювання LR(1) -таблиці

Розглянемо алгоритм конструювання таблиці, що керує LR(1)-аналізатором.

Нехай $G = (N, T, P, S)$ - КВ-граматика. **Доповненою** граматикою для даної граматики *G* називається КВ- граматика

$$G' = (N \cup \{S'\}, T, P \cup \{S' \to S\}, S')$$

тобто еквівалентна граматика, в якій введений новий початковий символ S' і нове правило виводу $S' \to S$.

Це додаткове правило вводиться для того, щоб визначити, коли аналізатор повинен зупинити розбір і зафіксувати допуск входу. Таким чином, допуск має місце тоді і тільки тоді, коли аналізатор готовий виконати згортку за правилом $S' \to S$.

LR(1)-**ситуацією** називається пара $[A \to \alpha\beta, a]$, де $A \to \alpha\beta$ - правило граматики, *a* – або термінал або правий кінцевий маркер *$*. Друга компонента ***a*** ситуації називається ***аванланцюжком*.**

Будемо говорити, що LR(1)-ситуація $[A \to \alpha.\beta, a]$ **допустима** для активного префікса δ, якщо існує виведення $S \Rightarrow_r^* \gamma A\omega \Rightarrow_r \gamma\alpha\beta\omega$, де $\delta = \gamma\alpha$ і або *a* - перший символ ω або $\omega = e$ і $a = \$$.

Будемо говорити, що ситуація **допустима**, якщо вона допустима для якого-небудь активного префікса.

Приклад 9.2.2.

Дано граматику $G = (\{S, B\}, \{a, b\}, P, S)$ з правилами

$S \to BB$

$B \to aB \mid b$

Існує правостороннє виведення $S \Rightarrow_r^* aaBab \Rightarrow_r aaaBab$. Легко замітити, що ситуація $[B \to a.B, b]$ допустима для активного префікса $\delta = aaa$, якщо у визначенні вище покласти $\gamma = aa$, $A = B$, $\omega = ab$, $\alpha = a$, $\beta = B$. Існує також правостороннє виведення $S \Rightarrow_r^* BaB \Rightarrow_r BaaB$. Тому для активного префікса *Baa* допустима ситуація $[B \to a.B, \$]$.

Головна ідея методу полягає в тому, що для граматики будується детермінований скінченний автомат, що розпізнає активні префікси. Для цього ситуації групуються в множини, які і утворюють стани автомата. Ситуації можна розглядати як стани недетермінірованного скінченного автомата, що розпізнють активні префікси, а їх групування насправді є процес побудови детермінованого скінченного автомата з недетермінованого.

Аналізатор, що працює зліва-направо по типу зсув- згортка, повинен уміти розпізнавати основи на верхівці магазину. Стан автомата після прочитання вмістимого магазину і поточний вхідний символ визначають чергову дію автомата. Функцією переходів цього скінченного автомата є функція переходів LR-аналізатора. Щоб не переглядати магазин на кожному кроці аналізу, на верхівці магазину завжди зберігається той стан, в котрому повинен знаходитися цей скінчений автомат після того, як він прочитав символи граматики в магазині від дна до верхівки.

Розглянемо ситуацію виду $[A \to a.B\beta, a]$ з множини ситуацій, допустимих для деякого активного префікса *z*. Тоді існує правостороннє виведення $S \Rightarrow_r^* yAax \Rightarrow_r y\alpha B\beta ax$, де $z = y\alpha$. Припустимо, що з βax виводиться рядок терміналів $b\omega$. Тоді для деякого правила виводу виду $B \to q$ існує виведення $S \Rightarrow_r^* zBb\,\omega \Rightarrow_r zqb\,\omega$. Таким чином $[B \to .q, b]$ також допустиме для z і ситуація $[A \to \alpha B.\beta, a]$ допустима для активного префікса z*B*. Тут або *b* може бути першим терміналом, виведеним з β, або з β виводиться *e* у виведенні $\beta ax \Rightarrow_r^* b\omega$ і тоді *b* рівно *a*. Тобто *b* належить $FIRST(\beta ax)$.

Побудовою всіх таких ситуацій для даної множини ситуацій, тобто її замикання, робить наведена нижче функція ***closure***.

Система множин допустимих LR(1)-ситуацій для всіх можливих активних префіксів доповненої граматики називається **канонічною системою** множин допустимих LR(1)-ситуацій. Алгоритм побудови канонічної системи множин наведено нижче.

Алгоритм 9.2.1 *Конструювання канонічної системи множин допустимих LR(1)-ситуацій.*

Вхід. КВ-граматика $G = (N, T, P, S)$.

Вихід. Канонічна система C множин допустимих LR(1)-ситуацій для граматики G.

Метод. Виконати для доповненої граматики G' процедуру items, яка використовує функції closure і goto

function closure (*I*) {/ * *I* - множина ситуацій * /
do {
for (кожній ситуації $[A \to \alpha.B\beta, a]$ із *I*,
кожного правила виведення $B \to \gamma$ із G',
кожного терміналу *b* із $FIRST\ (\beta a)$,
такого, що $[B \to .\gamma, b]$ немає в *I*)
додати $[B \to .\gamma, b]$ до *I*;
}
while (до *I* можна додати нову ситуацію);
return *I*;
}
function goto(*I*, *X*) {/ * *I* – множина ситуацій; *X* - символ граматики * /
Нехай $J = \{[A \to \alpha X.\beta, a] \mid [A \to \alpha.X\beta, a] \in I\}$;
return closure (*J*);
}
procedure items (G') {/ * G' - доповнена граматика * /

$I_0 = \text{closure}(\{[\ S' \to .S, \$]\})$;

$G = \{I_0\}$;

do {

for (кожній множині ситуацій I з системи C,

кожного символу граматики X такого, що $\text{goto}(I, X)$ не пуста

і не належить C)

додати $\text{goto}(I, X)$ до системи C;

}

while (до C можна додати нову множину ситуацій);

Якщо I - множина ситуацій, допустимих для деякого активного префікса δ, то $\text{goto}(I, X)$ - множина ситуацій, допустимих для активного префікса δX.

Робота алгоритму побудови системи C множин допустимих LR(1)-ситуацій починається з того, що в C поміщається початкова множина ситуацій $I_0 = \text{closure}(\{[\ S' \to .S, \$]\})$. Потім за допомогою функції goto обчислюються нові множини ситуацій і включаються до C. По-суті, $\text{goto}(I, X)$ - перехід скінченного автомата зі стану I по символу X.

Тепер розглянемо, як за системою множин LR(1) -ситуацій будується LR(1) -таблиця, тобто функції дій і переходів LR(1)-аналізатора.

Алгоритм 9.2.2. Побудова LR(1) -таблиці.

Вхід. Канонічна система $C = \{I_0, I_1, ..., I_n\}$ множин допустимих LR(1)-ситуацій для граматики G.

Вихід. Функції *Action* і *Goto*, складові LR(1) - таблиці для граматики G.

Метод. Для кожного стану i функції $Action[i, a]$ і $Goto[i, X]$ будуються за множиною ситуацій I_i:

(1) Значення функції дії (*Action*) для стану i визначаються так:

а) якщо $[A \to \alpha.a\beta, b] \in I_i$ (a - термінал) і $\text{goto}(I_i, a) = I_j$, то вважаємо $Action[i, a] = \text{shift } j$;

б) якщо $[A \to \alpha., a] \in I_i$, причому $A \neq S'$, то вважаємо $Action[i, a] = \text{reduce } A \to \alpha$;

в) якщо $[S' \to S.,|\$] \in I_i$, то вважаємо $Action[i,\$] = \text{accept}$.

(2) Значення функції переходів для стану *i* визначають так: якщо $\text{goto}(I_i, A) = I_j$, то $Goto[i, A] = j$ (тут A - нетермінал).

(3) Всі входи в *Action* і *Goto*, що не визначені кроками 2 і 3, вважаємо рівними error.

(4) Початковий стан аналізатора будується з множини, що містить ситуацію $[S' \to .S,\$]$.

Таблиця на основі функцій *Action* і *Goto*, отриманих в результаті роботи алгоритму 9.2.2., називається канонічною LR(1) -таблицею. Працюючий з нею LR(1) -аналізатор, називається канонічним LR(1) -аналізатором.

Приклад 9.2.3.

Розглянемо наступну граматику, що є доповненою для граматики з прикладу 9.2.1.:

(0) $E' \to E$

(1) $E \to E + T$

(2) $E \to T$

(3) $T \to T * F$

(4) $T \to F$

(5) $F \to id$

Множини ситуацій і переходи по goto для цієї граматики наведено на рис. 9.1.2. LR (1) -таблиця для цієї граматики наведена на рис 9.2.1.

Простежимо послідовність створення цих множин більш докладно.

1. Обчислюємо $I_0 = closure(\{[E' \to .E,\$]\})$.

1.1. Ситуація $[E' \to .E,\$]$ потрапляє в нього за замовчуванням як початкова??(вихідна).

1.2. Якщо звернутися до позначень функції *closure*, то для неї

$\alpha = \beta = e$, $B = E$, $a = \$$,

$first\ (\beta a) = first\ (\$) = \{\$\}$.

Це означає, що для терміналу $ додаємо ситуації на основі правил зі знаком Е в лівій частині правила.

Це правила $E \to E + T$ і $E \to T$ і відповідні їм ситуації $[E \to .E + T,\$]$ і $[E \to .T,\$]$.

1.3. Переглядаємо отримані ситуації.

Для ситуації $[E \to .E + T,\$]$ $\beta = +$, тому $first\ (\beta a) = first\ (+\$) = \{+\}$.
На основі цього додаємо до I_0
$[E \to E + .T,+]$ і $[E \to .T,+]$.

1.4. Для ситуації $[E \to .T,\$]$ $\beta = e$, $first\ (\beta a) = \{\$\}$.
Тому додаємо до I_0
$[T \to .T * F,\$]$ і $[T \to .F,\$]$.

1.5. Аналогічно, для ситуації $[E \to .T,+]$ $\beta = e$, $first\ (\beta a) = \{+\}$.
Тому додаємо до I_0
$[T \to .T * F,+]$ і $[T \to .F,+]$.

1.6. Із ситуації $[T \to .T * F,+]$ $\beta = *$, $first\ (\beta a) = \{*\}$.
Тому додаємо до I_0
$[T \to .T * F,*]$ і $[T \to .F,*]$.

1.7. Далі із ситуації $[T \to .F,*]$ отримуємо ситуацію $[F \to .id,*]$.
із ситуації $[T \to .F,\$]$ - ситуацію $[F \to .id,\$]$, а
із ситуації $[T \to .F,+]$ - ситуацію $[F \to .id,+]$.
Таким чином, всі 14 шуканих ситуацій I_0 отримані.

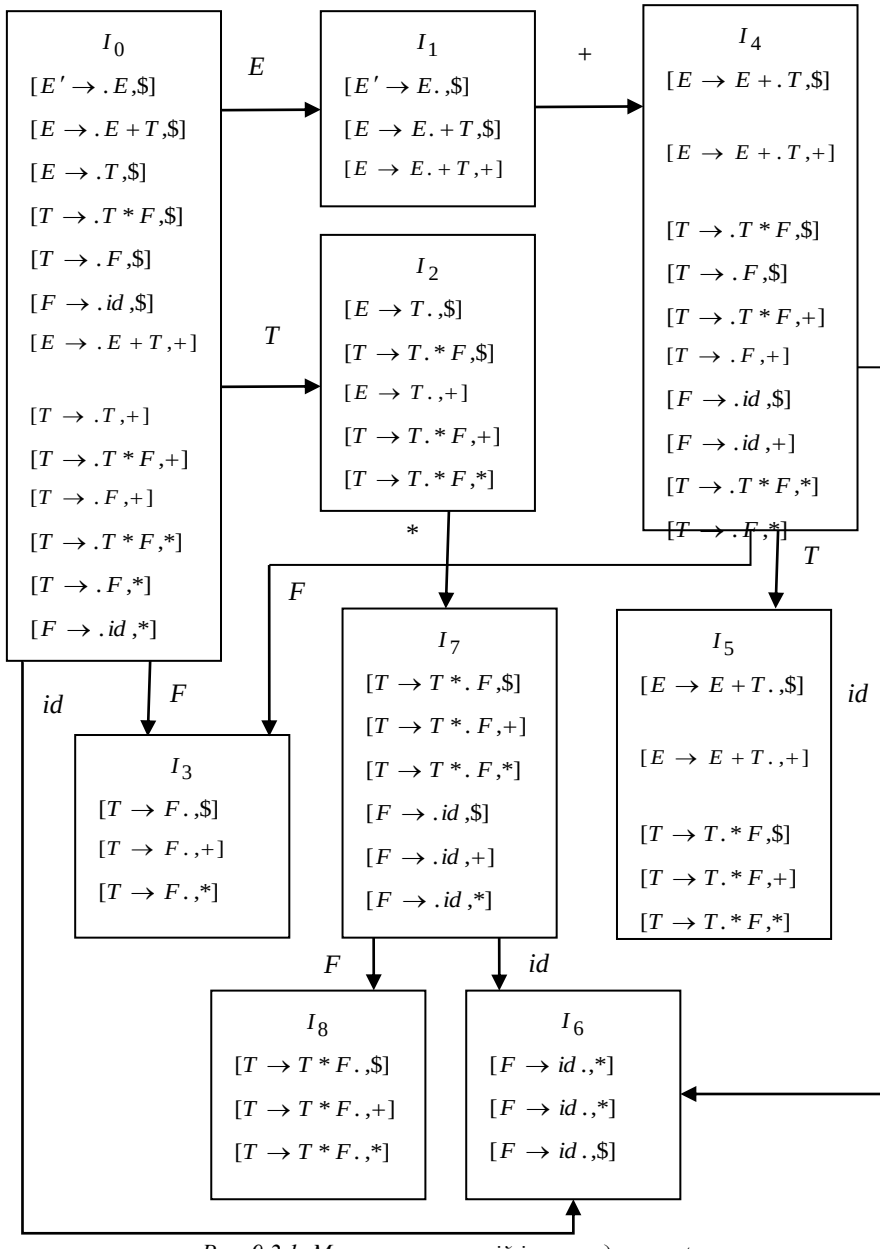


Рис. 9.2.1. Множини ситуацій і переходи по goto

Повертаємося до головної функції *items*, включаємо I_0 в множину C і досліджуємо результати застосування функції $goto(I_0, X)$, де $X \in \{E', E, T, F, +, *, \$, id\}$.

Якщо подивитися на вигляд правил у функції $goto(I_0, X)$, то видно, що X має зустрітися в правій частині хоча б одного правила. Для E' таких правил у нас немає, тому значення функції $goto(I_0, E')$ порожньо.

Візьмемо $goto(I_0, E)$. E зустрічається після крапки в правих частинах двох ситуацій з I_0, значить беремо ці два правила і переносимо в них точки на один символ вправо (поки є куди - НЕ вперлися в кому), отримуємо:

$[E' \to E.,\$]$ і

$[E \to E.+T,\$ \mid +]$.

Обчислимо від кожної з цих ситуацій функцію *closure*. Але, оскільки праворуч від точки тут або порожній ланцюжок, або термінал, то ніяких нових ситуацій не виникає. Далі відслідковуємо, чи може перемістити точка далі на право і по якому символу. Якщо можна, будуємо відповідну множину (див. Рис. 9.2.1). і т.д.

9.3. LR(1)-ГРАМАТИКИ

Якщо для КВ-граматики *G* функція *Action*, отримана в результаті роботи алгоритму 9.2.2., не містить неоднозначно визначених входів, то граматика називається LR(1) -граматикою.

Мова L називається LR(1)-мовою, якщо вона може бути породжена деякою LR(1)-граматикою.

Деколи використовується інше визначення LR(1)-граматики. Граматика називається LR(1), якщо з умов

1. $S' \Rightarrow_r^* uAw \Rightarrow_r uvw$,
2. $S' \Rightarrow_r^* zBx \Rightarrow_r uvy$,
3. $FIRST(\omega) = FIRST(y)$

випливає, що $uAy = zBx$ (тобто $u = z$, $A = B$, $x = y$).

Згідно з цим визначенням, якщо uvw і uvy - правовивідні ланцюжки доповненої граматики, у яких $FIRST(\omega) = FIRST(y)$ і $A \to v$ - останнє правило, використане в правосторонньому виведенні ланцюжка uvw , то правило $A \to v$ повинно застосовуватися і в правому розборі при згортці uvy до uAy . Оскільки A дає v незалежно від w, то LR(1)-умова означає, що в $FIRST(\omega)$ міститься інформація, достатня для визначення того, що uv за один крок виводиться з uA . Тому ніколи не може виникнути сумніву щодо того, як згорнути черговий правовивідний ланцюжок доповненої граматики.

Можна довести, що ці два визначення еквівалентні.

Дамо тепер визначення LR(k)-граматики.

Визначення. Нехай $G = (N, \Sigma, P, S)$ - КВ-граматика і $G' = (N', \Sigma, P', S')$ - отримана з неї доповнена граматика. Будемо називати GLR(k) -граматикою для $k \geq 0$, якщо з умов

(1) $S' \Rightarrow_{G',r}^* \alpha A \omega \Rightarrow_{G',r} \alpha\beta\omega$,

(2) $S' \Rightarrow_{G',r}^* \gamma Bx \Rightarrow_{G',r} \alpha\beta y$,

(3) $FIRST_k(\omega) = FIRST_k(y)$

випливає, що $\alpha Ay = \gamma Bx$ (тобто $\alpha = \gamma$, $A = B$, $x = y$).

Граматика G називається LR-граматикою, якщо вона LR(k) -граматика для деякого $k \geq 0$.

Інтуїтивно це визначення говорить про те, що якщо $\alpha\beta\omega$ і $\alpha\beta y$ - правовивідні ланцюжки доповненої граматики, у яких $FIRST_k(\omega) = FIRST_k(y)$ і $A \to \beta$ - останнє правило, використане в правосторонньому виведенні ланцюжка $\alpha\beta\omega$, то правило $A \to \beta$ повинно використовуватися також у правому розборі при згортці $\alpha\beta y$ до αAy. Оскільки A дає β незалежно від ω, то LR(k) - умова говорить про те, що в $FIRST_k(\omega)$ міститься інформація, достатня для визначення того, що $\alpha\beta$ за один крок виводиться з αA. Тому не повинно виникнути сумнівів відносно того, як згорнути чергову правовивідного ланцюжка доповненої граматики. Крім того, працюючи з LR(k) - граматикою, ми завжди знаємо, чи допустити даний вхідний ланцюжок чи продовжувати розбір.

Приклад 9.3.1

Розглянемо граматику G з правилами $S \to Sa \mid a$.

Згідно з визначенням, G не LR(0) -граматика, оскільки з трьох умов

(1) $S' \Rightarrow^{0}_{G'r} S' \Rightarrow_{G'r} S$,

(2) $S' \Rightarrow_{G'r} S \Rightarrow_{G'r} Sa$,

(3) $FIRST_0(e) = FIRST_0(a) = e$

не випливає, що $S'a = S$. Застосовуючи визначення до цієї сітуації, маємо $\alpha = e$, $\beta = S$, $\omega = e$, $\gamma = e$, $A = S'$, $B = S$, $x = e$ $y = a$. Проблема тут полягає в тому, що не можна встановити, чи є S основою правовивідного ланцюжка Sa, не бачачи символу, що стоїть після S (тобто спостерігаючи «нульову» кількість символів). Інтуїтивно G не повинна бути LR(0)-граматикою і вона не буде нею, якщо користуватися першим визначенням. Це визначення ми і будемо використовувати далі.

Приклад 9.3.2.

Нехай G - ліволінійна граматика з правилами

$S \to Ab \mid Bc$

$A \to Aa \mid e$

$B \to Ba \mid e$

Зауважимо, що G не є LR(k) -граматикою при довільному k.

Припустимо, що G - LR(k) -граматика. Разглянемо два правосторонніх виведення в доповненій граматиці G':

$S' \Rightarrow_r S \Rightarrow_r^* Aa^k b \Rightarrow_r a^k b$,

і

$S' \Rightarrow_r S \Rightarrow_r^* Ba^k c \Rightarrow_r a^k c$,

Ці два виведення задовільняють умові з визначення LR(k) - граматики при $\alpha = e$, $\beta = e$, $\omega = a^k b$, $\gamma = e$, $y = a^k c$. Оскільки висновок невірний, тобто $A \neq B$, то G не LR(k)-граматика. Більше того, оскільки LR(k)-умова порушується для всіх k, то G – не LR(k) -граматика.

Якщо граматика не є LR(1), то аналізатор типу зсув-згортка при аналізі деякого ланцюжка може досягнути конфігурації, в якій він, знаючи вміст магазину і наступний вхідний символ, не може вирішити, робити зсув чи згортку (конфлікт зрушення / згортка), або не може вирішити, яку з декількох згорток застосувати (конфлікт згортка / згортка).

Зокрема, LR(1)-граматика не може бути неоднозначною. Для підтвердження розглянемо два різні правосторонні виводи:

(1) $S \Rightarrow_r u_1 \Rightarrow_r \ldots \Rightarrow_r u_n \Rightarrow_r w$, і

(2) $S \Rightarrow_r v_1 \Rightarrow_r \ldots \Rightarrow_r v_m \Rightarrow_r w$,

Неважко помітити, що LR(1)-умова (згідно з другим визначенням LR (1) -граматики) порушується для найменшого з чисел і, для яких $u_{n-i} \neq v_{m-i}$.

Приклад 9.3.3.

Розглянемо ще раз граматику умовних операторів:

$S \to if\ E\ then\ S \mid if\ E\ then\ S\ else\ S \mid a$

$E \to b$

Якщо аналізатор типу зсув-згортка знаходиться в такій конфігурації, що необроблена частина вхідного ланцюжка є $else \ldots \$$, а в магазині знаходиться $\ldots if\ E\ then\ S$, то не можна визначити, чи є $if\ E\ then\ S$ основою, незалежно від того, що лежить в магазині нижче. Це конфлікт зсув/згортка. Залежно від того, що є на вході після $else$, правильною може бути згортка по $S \to if\ E\ then\ S$ або зсув $else$, а потім розбір іншого S і завершення основи $S \to if\ E\ then\ S\ else\ S$. Таким чином не можна сказати, чи потрібно в цьому випадку робити зсув чи згортку, так що граматика не є LR(1).

Ця граматика може бути перетворена до LR(1)-виду наступним чином:

$S \to M \mid U$

$M \to if\ E\ then\ M\ else\ M \mid a$

$U \to if\ E\ then\ S \mid if\ E\ then\ M\ else\ U$

$E \to b$

Основна різниця між LL(1) - і LR(1) - граматиками полягає в наступному. Щоб граматика була LR(1) - граматикою, необхідно розпізнавати входження правої частини правила виводу, переглянувши все, що виведено з цієї правої частини і поточний символ вхідного ланцюжка. Ця вимога істотно слабша, ніж вимога для LL(1) -граматики, коли необхідно визначити використовуване правило, бачачи тільки перший символ, виведений з його правої частини. Таким чином, клас LL(1)-граматик є підкласом класу LR(1)-граматик. Справедливі також наступні твердження

Теорема 9.3.1. Кожна LR(1)-мова є детермінованою КВ-мовою.

Теорема 9.3.2. Якщо L - детермінована КВ-мова, то існує LR(1)-граматика, що породжує L.

Теорема 9.3.3. Для будь-якої LR(k)-граматики при k> 1 існує еквівалентна їй LR(k-1) -граматики.

Доведено, що проблема визначення, чи породжує граматика LR-мову, є алгоритмічно нерозв‘язною.

9.4. ВІДНОВЛЕННЯ ПРОЦЕСУ АНАЛІЗУ ПІСЛЯ СИНТАКСИЧНИХ ПОМИЛОК

Одним з найпростіших методів відновлення після помилки при LR(1)-аналізі є наступний. При синтаксичній помилці переглядаємо магазин від верхівки, поки не знайдемо стан s з переходом на виділений нетермінал A. Потім скануються вхідні символи, поки не буде знайдений такий, який допустимий після A. У цьому випадку на верхівку магазину поміщається стан $Goto[s, A]$ і розбір продовжуэться. Для нетерміналу A может існувати декілька таких варіантів. Зазвичай A - це нетермінал, що представляє одну з основних конструкцій мови, наприклад оператор.

При більш детальному опрацюванні реакції на помилки можна в кожній порожній клітинці аналізатора поставити звертання до своєї підпрограми. Така підпрограма може вставляти або видаляти вхідні символи або символи магазину, міняти порядок вхідних символів.

9.5. ВАРІАНТИ LR-АНАЛІЗАТОРІВ

Часто побудовані таблиці для LR(1) -аналізатора виявляються досить великими. Тому при практиній реалізації використовуються різні методи їх стиснення. Проте виявляється, що часто при побудові для мови синтаксичного аналізатора типу «зсув- згортка» досить більш простих методів. Деякі з цих методів базуються на основі LR(1) -аналізаторів.

Одним із способів такого спрощення є LR(0) - аналіз - частковий випадок LR -аналізу, коли ні при побудові таблиць, ні при аналізі не враховується аванланцюжок.

Ще одним варіантом ДК-аналізу є так званий SLR(1) -аналізатори (Simple LR(1)). Вони будуються таким чином. Нехай $C = \{I_0, I_1, ..., I_n\}$ - набір множин допустимих LR(0) -ситуацій. Стани аналізатора відповідають I_i. Функції дій і переходів аналізатора визначаються наступним чином.

1. Якщо $[A \to u.av] \in I_i$ і $goto(I_i, a) = I_j$, то визначимо $Action[i, a] = \text{shift}\ \ j$

2. Якщо $[A \to u.] \in I_i$, то, для всіх $a \in FOLLOW\ (A)$, $A \neq S'$, визначимо $Action\,[i,a] = \text{reduce}\ \ A \to u$

3. Якщо $[S' \to S.] \in I_i$, то визначимо $Action\,[i,\$] = \text{accept}$.

4. Якщо $goto\,(I_i, A) = I_j$, де $A \in N$, то визначимо $Goto\,[i, A] = j$.

5. Решта входів для функцій *Action* і *Goto* визначимо як *error*.

6. Початковий стан відповідає множині ситуацій, що містить ситуацію $[S' \to . S]$.

Поширеним варіантом LR(1) -аналізу є також LALR(1) -аналіз. Він заснований на об'єднанні (злитті) деяких таблиць. Назвемо **ядром** множини LR(1) - ситуацій множину їх перших компонент (тобто у множині ситуацій не враховуються аванланцюжок). Об‘єднаємо всі множини ситуацій з однаковими ядрами, а в якості аванланцюжка візьмемо об'єднання аванланцюжків.

Функції *Action* і *Goto* будуються очевидним способом. Якщо функція *Action* таким способом побудованого аналізатора не має конфліктів, то він називається LALR(1)-аналізатором (LookAhead LR(1)). Якщо граматика є LALR(1)-аналізатором, то в таблицях LALR(1) аналізатора можуть появитися конфлікти типу згортка-згортка (якщо одне з об‘єднаних станів мало ситуації $[A \to \alpha, a]$ і $[B \to \beta, b]$, а іншого $[A \to \alpha, b]$ і $[B \to \beta, a]$, то в LALR(1) з'являться ситуації $[A \to \alpha, \{a, b\}]$ і $[B \to \beta, \{b, a\}]$). Конфлікти типу зсув-згортка з'явитися не можуть, оскільки аванланцюжок для зсуву до уваги не береться.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

Побудувати SLR(1) –аналізатори для наступних граматик:

а) Визначити функції FIRST та FOLLOW для граматики.

б) Побудувати *канонічну системи множин допустимих LR(1)-ситуацій*.

в) Побудувати таблицю SLR анализатора

с) Перевірити правильність побудови на трьох прикладах (один правильний, два неправильних).

Варіант №	Задана граматика з правилами
1	S -> Aa A -> Ab | Bb B -> cB | ε
2	S -> ABC A -> BC | a B -> A | b C -> c | ε
3	S -> AB A -> aAb | Ab | ε B -> bB | b
4	S -> aAb A -> BC | bc B -> Aa C -> c | ε
5	S -> AS | c A -> Aa | Ca C -> cA | c
6	S -> BS | c A -> cB | c B -> Ba | Aa
7	S -> ABc A -> aAb | Ab | ε B -> bB | bA

8	S -> Aa A -> Ac | Bb | ε B -> cB | ε
9	S -> aBS | c A -> cB | c B -> Ba | Aa | ε
10	S -> AC | c A -> Aa | Ca C -> cA | c
11	S -> SA | c A -> Aa | Ca C -> cA | c
12	S -> aAb A -> BS | bc B -> AaC C -> c | ε
13	S -> aB | c A -> cB | c B -> Ba | Aa | ε
14	S -> ABC A -> BC | a B -> A | b C -> c | ε

Література.

1. Серебряков В.А. Теория и реализация языков программирования / В.А. Серебряков, М.П. Галочкин, Д.Р. Гончар, М.Г. Фуругян. – М.: МЗ Пресс, 2006. – 352с.
2. Гладкий А.В. Формальные грамматики и языки. / А.В. Гладкий. – Наука, - 1973.
3. Нікольський Ю.В. Дискретна математика: Підручник. / Ю.В. Нікольський, В.В. Пасічник, Ю.М. Щербина – Львів: "Магнолія Плюс", 2005. – 608с.
4. Пиотровский Р.Г. Математическая лингвистика /Р.Г.Пиотровский, К.Б.Бектаев, А.А.Пиотровская. - Высшая школа, 1977.
5. Нікольський Ю.В. Збірник задач з дискретної математики. / Ю.В.Нікольський, Ю.М.Щербина. - Львів, 2002.
6. K.Rosen. Discrete Mathematics and its Applications. McGraw-Hill, 1988. P. 10
7. Кук Д. Компьютерная математика. / Д.Кук, Г.Бейз. - Наука, 1990.
8. Гросс М. Теория формальных грамматик / М.Гросс, А.Лантен. - Мир, 1971.
9. Ахо А. Теория синтаксического анализа, перевода и компиляции, в 2-томах./ Ахо А., Ульман Д. - М.: Мир, 1978.
10. Ахо А. Компілятори: принципи, технології, інструменти / Ахо А., Лам М., Сеті Р., Ульман Д. М. - СПб. - Київ: В.Д.Вілямс,2008 -1184с.

НАВЧАЛЬНЕ ВИДАННЯ

Формальні мови, граматики та автоматів

Навчальний посібник

для студентів напрямку «Комп‘ютерні науки» та «Системний аналіз»

Укладачі: Захарія Л.М.
Заяць М.М.

Редактор:

Комп‘ютерне верстання: